Universidad de Concepción
Campus Los Ángeles
Escuela de Educación

CREACIÓN DE MÓDULOS DIDÁCTICOS PARA PENSAMIENTO COMPUTACIONAL Y MATEMÁTICA:

De actividades desenchufadas a programación con tecnología digital.

Trabajo de Titulación presentado a la Escuela de Educación de la Universidad de Concepción para optar al grado de Licenciado en Educación y al título profesional de Profesor de Matemáticas.

Por: **Raúl Jara Fernández**
Cristóbal Manosalva Inostroza

Profesor Guía: **Dra. Marianela Castillo Fernández**

Los Ángeles, 2025

Comisión:
Dr. Cristian Pérez Toledo
Mg. Álvaro Moya Oliva

Parcialmente financiado por proyecto VRID 2024001289INTLA *Construcción de estrategias para el Desarrollo de las Habilidades del Pensamiento Computacional en la asignatura de Matemática a través de la Programación: Propuestas para abordar los cambios en la educación en Chile*, dirigido por la Dra. Marianela Castillo.

Dedicatoria

Dedicatoria de Raúl Jara Fernández:

A mi compañera de vida, Danitza Órdenes, por su apoyo incansable, su paciencia y sus palabras oportunas que iluminaron cada etapa de este proceso. Su presencia fue un sostén esencial y una fuente constante de ánimo que hizo posible la culminación de este trabajo.

A mi hijo, Dante Jara, cuya llegada, mientras se desarrollaba este trabajo, marcó profundamente este camino. Ser padre me otorgó una motivación renovada y un propósito más grande que cualquier desafío académico. Este logro también es suyo, nacido entre desvelos, expectativas y una alegría inmensa.

A mi familia, por la confianza y respaldo que siempre me han brindado. A mis padres, por su apoyo continuo y su ejemplo; y a mi hermano, por sus ánimos y cercanía en los momentos que más los necesité.

A mi abuela, la única que aún me acompaña, por su cariño y su fortaleza, que han sido guía y refugio en mi vida. Y a mis abuelos que ya no están, quienes sin duda habrían compartido con orgullo este logro. A ellos, con gratitud y memoria.

Dedicatoria de Cristóbal Manosalva Inostroza:

Primeramente a Dios, quien encendió una luz en mí.

A mi familia, la que desde muy pequeño me inculcó lo importante que es la educación, desde el acompañamiento para realizar las tareas, hasta darme todas las posibilidades para poder lograr mi proyecto de vida.

Especialmente a mi Madre, quien me dijo una vez "la mejor herencia que te puedo entregar es la educación", bueno la herencia ha sido entregada.

A mi hermano, agradezco a Dios, por verlo crecer. Y ver en la persona en que se está convirtiendo.

Agradecimientos

Agradecemos sinceramente a nuestra profesora guía, Marianela Castillo Fernández, por su orientación constante y fundamental en el desarrollo de esta tesis.

Cristóbal Manosalva Inostroza y Raúl Jara Fernández.

Agradezco a Dios, mis familiares, amigos y compañeros que han estado presentes en esta etapa de mi vida. A mi compañero Raúl Jara, por su solidaridad y compromiso. A Camila, Ariana y Pía, por todas las anécdotas y momentos compartidos. Finalmente, a los profesores que me han ayudado a crecer como persona.

Cristóbal Manosalva Inostroza.

Agradezco profundamente a mi familia, cuyo apoyo incondicional ha sido el pilar de este proceso; a mis amigos, por su compañía constante; a mi compañera de vida, por su amor, paciencia y confianza que dieron equilibrio a cada etapa; y a mi hijo, cuya llegada llenó este camino de un sentido nuevo y poderoso. Agradezco también a mi compañero Cristóbal Manosalva, por su compromiso, comprensión y disposición permanente para avanzar incluso en los momentos más exigentes. Extiendo también mi gratitud a los profesores y asesores que ofrecieron su guía y conocimientos de manera desinteresada.

Raúl Jara Fernández.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Resumen

En el marco de la Cuarta Revolución Industrial y las recientes actualizaciones curriculares en Chile, este trabajo aborda la brecha existente entre las demandas de alfabetización digital y las competencias docentes para integrar el pensamiento computacional en la enseñanza de la matemática. El objetivo de este trabajo es diseñar módulos didácticos que articulen la resolución de problemas matemáticos con el desarrollo progresivo de habilidades de programación, mediante una trayectoria que transita desde experiencias concretas "desenchufadas" (unplugged) hacia la programación "enchufada" en entornos digitales como Scratch y Python.

La metodología se fundamentó en el enfoque de Investigación de Diseño (Molina et al., 2011) estructurada en tres fases: un análisis epistemológico y curricular bajo los modelos TPACK y CTPK-12; el diseño de secuencias de aprendizaje basadas en las trayectorias de Rich et al.; y la validación interna de la propuesta mediante el instrumento Instructional Quality Assessment (IQA). Como resultado, se elaboraron cinco módulos pedagógicos, a la vez que se evidencia una correspondencia funcional entre las habilidades del pensamiento computacional y los Objetivos de Aprendizaje del currículum nacional en Matemática.

Los hallazgos demuestran que las tareas diseñadas poseen un alto potencial de demanda cognitiva, permitiendo a los estudiantes transitar desde la manipulación concreta hacia la abstracción formal y la modelación computacional. Se concluye que la propuesta constituye un recurso didáctico pertinente y replicable que no solo facilita la labor docente, sino que fortalece competencias transversales del siglo XXI, permitiendo que el pensamiento computacional actúe como un catalizador del razonamiento lógico-matemático en diversos contextos educativos.

Palabras clave: Matemática, Pensamiento Computacional, Programación.

Abstract

Within the context of the Fourth Industrial Revolution and recent curriculum updates in Chile, this work addresses the gap between the demands of digital literacy and teachers' competencies for integrating computational thinking into mathematics education. The objective of this work is to design didactic modules that link mathematical problem-solving with the progressive development of programming skills, through a learning trajectory that moves from concrete, "unplugged" experiences to "plugged-in" programming in digital environments such as Scratch and Python.

The methodology was based on the Design Research approach (Molina et al., 2011), structured in three phases: an epistemological and curricular analysis using the TPACK and CTPK-12 models; the design of learning sequences based on the learning trajectories of Rich et al.; and the internal validation of the proposal using the Instructional Quality Assessment (IQA) instrument. As a result, five pedagogical modules were developed, demonstrating a functional correspondence between computational thinking skills and the Learning Objectives of the national mathematics curriculum.

The findings show that the designed tasks have a high potential for cognitive demand, allowing students to progress from concrete manipulation to formal abstraction and computational modeling. It is concluded that the proposal constitutes a relevant and replicable teaching resource that not only facilitates the teacher's work but also strengthens transversal 21st-century skills, enabling computational thinking to act as a catalyst for logical-mathematical reasoning in diverse educational contexts.

Keywords: Mathematics, Computational Thinking, Programming.

Introducción

En el contexto de la Cuarta Revolución Industrial, caracterizada por la transformación digital, se vuelve imprescindible preparar a los estudiantes para enfrentar los desafíos del mundo laboral y social actual (World Economic Forum, 2023). Esto ha impulsado la necesidad de adquirir las habilidades del siglo XXI, definidas como capacidades esenciales para competir y participar en una sociedad cambiante (Scott, 2015). Chile ha respondido a estos desafíos mediante reformas educativas progresivas; un hito clave fue la aprobación de las Bases Curriculares en 2019 para tercero y cuarto medio, las cuales integraron explícitamente la asignatura de 'Pensamiento Computacional y Programación' en el plan de formación diferenciada.

Actualmente, el sistema educativo se encuentra en la etapa final de un proceso de actualización curricular iniciado en 2022, donde los docentes han manifestado un interés preferente por fortalecer la computación y la alfabetización digital en niveles anteriores (MINEDUC[1], 2024a; UNESCO[2], 2021). Ante este escenario, se evidencia que los profesores, en general, no han recibido una capacitación apropiada para promover estas habilidades, lo que subraya la necesidad crítica de una formación especializada que les permita implementar exitosamente el pensamiento computacional en el aula (Agencia de Calidad de la Educación, 2021; Fundación Kodea, 2023; Fundación País Digital, 2023).

Estos antecedentes permiten situar el problema central de esta tesis: la brecha existente entre las demandas formativas del sistema escolar chileno y las competencias reales que poseen los docentes para integrar el pensamiento computacional en la enseñanza de la matemática.

[1] MINEDUC, por sus siglas en español: Ministerio de Educación de Chile.
[2] UNESCO, por sus siglas en inglés: United Nations Educational, Scientific and Cultural Organization.

A lo largo del tiempo, el concepto de pensamiento computacional ha sido abordado desde diversas definiciones. De estas, esta investigación destaca dos perspectivas: la definición pionera de Wing (2006), quien lo caracteriza como los procesos de pensamiento involucrados en formular problemas y sus soluciones de modo que estas puedan ser representadas para ser ejecutadas eficazmente por un agente que procesa información; y la visión más reciente de García-Peñalvo y Mendes (2018), quienes lo consideran una competencia transversal del siglo XXI para comprender, modelar y resolver problemas en contextos digitales y no digitales.

La coexistencia de ambas perspectivas revela la necesidad de propuestas didácticas que traduzcan estas definiciones en experiencias de aprendizaje concretas, progresivas y coherentes con el currículo escolar.

El objetivo principal de esta investigación es diseñar un módulo didáctico que aborde la Resolución de Problemas de matemática y que promueva desarrollo progresivo del pensamiento computacional y habilidades del siglo XXI, desde la programación desenchufada hacia la programación enchufada en estudiantes de educación básica y media.

Este propósito se materializa mediante un estudio de carácter aplicado y diseño didáctico, cuyo foco es generar una propuesta replicable y fundamentada tanto en la teoría como en la política educativa vigente.

Para lograr estos objetivos, la investigación desarrolló un proceso que incluyó: (1) la revisión crítica de literatura nacional e internacional; (2) la sistematización de definiciones operacionalizadas de pensamiento computacional y habilidades matemáticas; (3) el análisis del currículo vigente y de las demandas formativas docentes; y (4) el diseño progresivo de un módulo didáctico compuesto por secuencias integradas de actividades desenchufadas y enchufadas.

El presente estudio se estructura en cinco capítulos que organizan el proceso investigativo desde sus fundamentos hasta la propuesta didáctica final. El Capítulo 1 expone los antecedentes tanto a nivel internacional como nacional, estableciendo el planteamiento del problema, la justificación de la investigación y los objetivos que guían el estudio. El Capítulo 2 desarrolla el marco referencial, abordando en profundidad las definiciones de Pensamiento Computacional, las Habilidades del Siglo XXI y las estrategias para su aprendizaje. Asimismo, analiza la programación educativa, la integración curricular en Matemática y la Resolución de Problemas, finalizando con un análisis sobre la formación docente en esta área. El Capítulo 3 describe el marco metodológico, detallando el enfoque de investigación empleado y las fases operativas que permitieron el diseño de la propuesta. El Capítulo 4 presenta los tres resultados del estudio. (1), sistematiza los conceptos y definiciones operacionalizadas necesarias para el diseño, (2) se incorpora la correspondencia elaborada entre las habilidades del pensamiento computacional y las habilidades matemáticas de las Bases Curriculares, y (3) despliega la propuesta central: cinco módulos didácticos articulados con actividades que progresan desde la modalidad desenchufada hacia la programación enchufada. El Capítulo 5 ofrece las conclusiones generales, acompañadas de la discusión de los hallazgos, las limitaciones detectadas durante el proceso y las proyecciones para futuras investigaciones.

Por último, el documento incorpora las secciones de Bibliografía y Anexos. La primera compila las fuentes teóricas, normativas y metodológicas que sustentaron la investigación. La segunda sección adjunta el material complementario indispensable para la replicabilidad del estudio. Estas secciones finales refuerzan el carácter aplicado del estudio, asegurando su transferibilidad a contextos reales de enseñanza y ofreciendo herramientas que facilitan su implementación por parte de otros docentes.

Capítulo 1: Problema y Objetivos

1.1. Antecedentes del Problema

La Cuarta Revolución Industrial, caracterizada por la era digital, la revolución informática y las tecnologías emergentes, está transformando diversos aspectos de la vida cotidiana, una de ellas, el trabajo. Así se evidencia en el Informe sobre el Futuro del Empleo (World Economic Forum, 2023), que señala: "Hoy en día, las organizaciones estiman que el 34% de todas las tareas relacionadas con el negocio son realizadas por máquinas, y el 66% restante por humanos." (p.6) Esto ha generado nuevos desafíos para la sociedad actual, lo que se ve reflejado en las habilidades para el Siglo XXI, las que se definen como "los conocimientos, capacidades y actitudes necesarias para ser competitivos en la fuerza laboral del siglo XXI, participar adecuadamente en una sociedad cada vez más diversa, utilizar las nuevas tecnologías y lidiar con mundos laborales que cambian a gran velocidad." (Scott, 2015, p.4). Como también los aprendizajes del siglo XXI, de los cuales, nos enfocaremos en la Resolución de Problemas.

La Resolución de Problemas se entiende como "la habilidad para buscar, seleccionar, evaluar, organizar y sopesar alternativas e interpretar información. Asimismo, la resolución de problemas en el siglo XXI requiere que la persona recurra a múltiples ámbitos para encontrar soluciones a cuestiones complejas." (Scott, 2015, p.5). En Chile, la Resolución de Problemas está presente como una habilidad en el curriculum nacional de matemáticas desde el año 2012, luego de una reforma educacional, donde "La Resolución de Problemas es el foco para la enseñanza de la matemática" (MINEDUC, 2012). No obstante, Felmer et al. (2019) señalan que en las aulas de las escuelas de matemáticas chilenas, la Resolución de Problemas ha estado prácticamente ausente.

En Chile se han logrado avances importantes en conectividad y acceso a internet, situándose incluso sobre algunos países europeos en términos de velocidad de conexión y asequibilidad. Sin embargo, todavía existen brechas internas relacionadas con el ingreso, la edad, el género y la ubicación geográfica, además de un rezago en la adopción de tecnologías digitales avanzadas, lo que limita el desarrollo de sectores estratégicos (Órdenes et al., 2024). En el ámbito educativo, se han implementado iniciativas que buscan fortalecer la formación digital de los docentes, como el Plan Nacional de Lenguajes Digitales, orientado a integrar el pensamiento computacional y la programación en las aulas. (Fundación País Digital, 2023).

Durante el año 2019, se aprueban las nuevas bases curriculares, en donde, en el plan de formación diferenciada Humanístico-Científica, específicamente en el área de matemáticas, nos encontramos una asignatura, titulada "Pensamiento Computacional" (MINEDUC, 2019c), ahora tal como lo propone dichas bases, podemos considerar que,

> la asignatura contribuye también al desarrollo de las habilidades analíticas, la resolución de problemas y la capacidad de diseño, al poner en contacto a los estudiantes con ideas básicas del pensamiento computacional: la descomposición de fenómenos o situaciones y la abstracción, que permiten reducir la complejidad, y el concepto de algoritmo, que describe el proceso necesario para resolver un problema (p.274).

Además, en el año 2022, se inició en Chile el proceso de actualización curricular (MINEDUC, 2024a), que consta de las siguientes etapas:

1. Diagnóstico y levantamiento de información
2. Desarrollo de la Propuesta
3. Congreso Pedagógico y curricular

4. Publicación de la Propuesta
5. Consulta Pública
6. Periodo de ajustes a la propuesta
7. Ingreso de la propuesta al Consejo Nacional de Educación.

En el año 2025, la actualización curricular se encuentra en su última etapa, obteniendo los siguientes antecedentes del proceso, uno de los resultados del Congreso Pedagógico y Curricular señala que las y los Docentes, educadores/as y/o asistentes de la educación en un 10,7% consideran preferentemente como tema/asignatura de interés "Computación y alfabetización digital", acompañado de "Lenguaje/habilidades comunicativas" y "Educación cívica, política, formación ciudadana" (MINEDUC et al., 2024b). Lo que se refleja en la propuesta curricular, que buscaría trabajar sistema binario desde séptimo básico y se nombra el concepto de pseudocódigo en Objetivos de Aprendizaje de octavo básico a segundo medio (MINEDUC, 2024a).

Según Boom y colaboradores (2018), en su estudio sobre la relación entre el pensamiento computacional y la inteligencia como capacidad general de Resolución de Problemas, obtienen como resultado que existe una relación lineal positiva significativa y grande. A medida que aumentaba la capacidad del Pensamiento Computacional, la inteligencia tendía a aumentar también. Son posibles diferentes conclusiones. Por lo que tenemos la necesidad y alternativa de utilizar el Pensamiento Computacional, para desarrollar la habilidad de Resolución de Problemas en los estudiantes de nuestros establecimientos educacionales.

La implementación exitosa de estas actividades de Pensamiento Computacional, depende en gran medida de la competencia docente. Como señalan Isharyadi y Juandi (2023), la competencia del profesorado es un factor esencial para el éxito de la aplicación de la Computational Thinking en el aula. Los profesores deben

tener conocimientos y competencias suficientes sobre el pensamiento computacional y ser capaces de utilizarla en el aprendizaje cotidiano.

Esta afirmación resalta la necesidad de una formación docente continua y especializada en el uso pedagógico de las tecnologías. Solo así se podrá garantizar que los docentes estén preparados para diseñar y llevar a cabo actividades de pensamiento computacional que promuevan aprendizajes significativos y duraderos.

En el contexto chileno, esta necesidad se vuelve aún más evidente. Como queda de manifiesto en el informe de resultados ICILS 2018[3], donde se dice que "los profesores, en general, no han recibido una capacitación apropiada que les permita incluir la promoción de habilidades de alfabetización digital y manejo de la información en su gestión pedagógica" (Agencia de Calidad de la Educación, 2021, p.43). Esto se alinea con los hallazgos del Diagnóstico Universidades en relación con la formación de los futuros docentes, realizado por Fundación Kodea (2023), que evidencian que la formación inicial de docentes en habilidades digitales es limitada, lo que se traduce en una brecha en sus conocimientos y habilidades para enseñar Pensamiento Computacional.

Esta problemática se ve agudizada por una desconexión estructural en la formación profesional. Como advierte la Comisión Nacional de Evaluación y Productividad (2023), existe una "brecha entre las habilidades disponibles en los docentes y las nuevas capacidades docentes que el currículo demanda" (p.46), lo que, sumado a la presión por las pruebas estandarizadas, dificulta la implementación efectiva de estrategias integradoras como el pensamiento computacional en el aula.

[3] ICILS, por sus siglas en inglés: International Computer and Information Literacy Study

1.2 Justificación de la investigación

La creciente relevancia del pensamiento computacional en la era digital ha impulsado la integración de esta habilidad en los currículos educativos a nivel mundial. En Chile, el Decreto 193, proclamado por el MINEDUC (2019a), representa un hito en este sentido, al establecer la necesidad de incorporar el pensamiento computacional y la programación en los planes de estudio de educación media. Sin embargo, la implementación efectiva de estas nuevas competencias se enfrenta a desafíos significativos, particularmente en lo que respecta a la formación docente.

En la misma línea, Los Estándares de la Profesión Docente: Marco Para La Buena Enseñanza, en su actualización, plantea el siguiente desafío docente:

> Las demandas de la sociedad actual frente a la educación han implicado importantes transformaciones que se traducen en dos grandes necesidades para los sistemas educativos: (a) disponer de definiciones curriculares que posibiliten a los/as estudiantes desarrollar las habilidades del Siglo XXI; y (b) contar con docentes que promueven el desarrollo de dichas habilidades a través de una enseñanza que logre, de manera efectiva y equitativa, aprendizajes significativos (Reimers & Chung, 2016, como se citó en CPEIP, 2021, p.11). La actualización del MBE aborda directamente la segunda necesidad, entregando una herramienta que identifica los saberes y desempeños profesionales necesarios para enseñar los objetivos disciplinares y transversales señalados en las Bases Curriculares y otros documentos complementarios, tales como el plan de

> formación ciudadana, los indicadores de calidad que mide la Agencia de la Calidad y el Decreto 67, entre otros. (CPEIP, 2021, p.11)

Lo que se concreta principalmente en el estándar 8: Estrategias para el desarrollo de habilidades del pensamiento, del Domicio C, que nos dice: “Desafía a sus estudiantes promoviendo el desarrollo del pensamiento crítico, creativo y la metacognición, basándose en los conocimientos de la disciplina que enseña, para que aprendan de manera comprensiva, reflexiva y con creciente autonomía.” (CPEIP, 2021, p.50).

Diversas investigaciones, como la que se menciona anteriormente, realizada por Fundación Kodea (2023), revelan una problemática relevante en la preparación de los futuros docentes para enseñar ciencias de la computación. La falta de capacitación adecuada en esta área limita su capacidad para transmitir de manera efectiva los conceptos fundamentales del pensamiento computacional a sus estudiantes. Aquella situación se agrava al considerar que, según UNESCO (2021), “la formación de los docentes sigue estando en el centro de la mayoría de las políticas revisadas, pero no se observan estrategias innovadoras que busquen superar las limitaciones que se han tenido en esta materia” (p.21). Esta situación contrasta con la rápida evolución de las tecnologías y las demandas del mercado laboral, lo que evidencia la necesidad de una formación docente más ágil y flexible.

A pesar de estos desafíos, la comunidad educativa ha reconocido la importancia del pensamiento computacional. Vázquez Uscanga et al. (2019) destacan que esta habilidad se ha posicionado como esencial en la era digital, y numerosos países han implementado iniciativas para promoverla. Sin embargo, los resultados obtenidos hasta el momento sugieren que aún queda un largo camino por recorrer para integrar de manera efectiva el pensamiento computacional en los currículos escolares. En este contexto, una estrategia prometedora para

abordar estos desafíos es un módulo didáctico, que transite desde lo desenchufado a lo digital, considerando lo señalado por Munasinghe et al. (2023), que dicen que las actividades desenchufadas preparan a los estudiantes para programar mediante una progresión pedagógica que conecta lo concreto con lo abstracto. Además, Curzon et al. (2014) demuestran que las actividades desconectadas, "no solo son una forma eficaz de introducir a los niños en los conceptos informáticos, sino que este trabajo demuestra que también son una forma eficaz de introducir estos conceptos a los profesores adultos" (p.92). De este modo, los docentes pueden explorar y enseñar los fundamentos de la programación y la computación a través de métodos accesibles, facilitando una comprensión gradual para luego acceder al uso de herramientas tecnológicas más avanzadas.

A pesar de estos esfuerzos del Plan Nacional de Lenguajes Digitales, aún se requiere avanzar en capacitaciones que aborden temas como metodologías activas, ciudadanía digital y ciberseguridad, de manera que los profesores puedan innovar en sus prácticas y preparar a los estudiantes para enfrentar los desafíos del siglo XXI (Fundación País Digital, 2023).

1.2.1 Preguntas de Investigación

De lo anterior, se evidencia un contraste entre la consolidada infraestructura digital del país y la brecha de habilidades docentes para la integración de tecnologías en el aula, lo cual fundamenta la necesidad de estudiar nuevas estrategias didácticas. Con el fin de operacionalizar el pensamiento computacional mediante una propuesta curricular y progresiva, surgen las siguientes preguntas de investigación:

1. ¿Qué fundamentos curriculares y pedagógicos sustentan la integración y el desarrollo del pensamiento computacional y resolución de problemas en la enseñanza de la Matemática en el contexto escolar?

2. ¿Cómo diseñar módulos didácticos para desarrollar la resolución de problemas en matemática, mediante actividades de Pensamiento Computacional, que articule progresivamente desde lo desenchufado a enchufado, según los niveles educativos?

1.4 Objetivos

1.4.1 Objetivo General

Diseñar módulos didácticos que aborden la Resolución de Problemas de matemática y que promueva desarrollo progresivo de las habilidades del Pensamiento Computacional, desde la programación desenchufada hacia la programación enchufada en estudiantes de educación básica y media.

1.4.2 Objetivos específicos

1. Recopilar los fundamentos curriculares, pedagógicos y académicos que sustentan la integración y el desarrollo del pensamiento computacional y resolución de problemas en la enseñanza de la Matemática en los niveles de educación básica y media.
2. Diseñar actividades articuladas y progresivas, desde la programación desenchufada hacia la programación enchufada, alineadas con las bases curriculares vigentes en el área de Matemática.

Capítulo 2. Marco Referencial

2.1 Pensamiento Computacional

El pensamiento computacional ha experimentado en las últimas dos décadas una expansión significativa que lo ha llevado desde su origen en la informática hacia convertirse en un marco cognitivo transversal, aplicable a diversos campos disciplinares, incluida la educación matemática. Esta evolución ha sido ampliamente documentada en la literatura contemporánea, que destaca cómo el pensamiento computacional se ha consolidado como una habilidad esencial para múltiples áreas del conocimiento y se está convirtiendo rápidamente en una habilidad fundamental para todos, no solo para los científicos de la computación, pues su aplicación se extiende a una amplia gama de disciplinas y problemas del mundo real" (Shute et al., 2017)

En su formulación seminal, Wing (2006) definió el pensamiento computacional como "los procesos de pensamiento involucrados en formular problemas y sus soluciones de modo que estas puedan ser representadas para ser ejecutadas eficazmente por un agente que procesa información". Esta definición, ampliamente citada en la literatura, posicionó al pensamiento computacional como una habilidad cognitiva fundamental "para todos, no solo para los científicos de la computación" (Wing, 2006, p. 33), estableciendo su carácter universal y transversal. Años más tarde, Wing (2011) amplió este concepto señalando que el pensamiento computacional "es el proceso de pensamiento en que la solución puede llevarse a cabo por un ser humano o máquina, o más en general, por combinaciones de seres humanos y máquinas" (p. 1). Esta ampliación introduce la dimensión socio–técnica del Pensamiento Computacional, subrayando que las soluciones pueden ser ejecutadas tanto por agentes humanos como artificiales.

En línea con Wing, Aho (2012) profundiza en la base teórica del constructo, indicando que el pensamiento computacional se funda en "nociones de cómputo, lenguajes, datos y complejidad" que funcionan como puente conceptual entre la formulación de problemas y los procedimientos ejecutables. Esta perspectiva destaca la importancia de la formalización, la estructuración y la relación entre problemas, modelos y algoritmos.

Selby y Woollard (2013) fortalecen esta visión al definir el pensamiento computacional como un conjunto de procesos de pensamiento que implican descomponer problemas complejos, reconocer patrones, abstraer principios y diseñar algoritmos. A diferencia de Wing, cuyo énfasis inicial está en la representabilidad y ejecutabilidad, estos autores centran la atención en los procesos cognitivos nucleares, articulándolos con habilidades de orden superior.

Desde una perspectiva actualizada, García-Peñalvo y Mendes (2017) sostienen que el pensamiento computacional es una competencia transversal del siglo XXI, clave para comprender, modelar y resolver problemas en contextos digitales y no digitales, lo que lo sitúa como una habilidad inter y transdisciplinaria requerida en áreas como la ingeniería, la matemática, las ciencias sociales e incluso la vida cotidiana. Esta ampliación conceptual es coherente con las políticas educativas internacionales, que conciben el pensamiento computacional como parte de la alfabetización necesaria para la participación ciudadana en sociedades altamente digitalizadas (Bocconi et al., 2016; INTEF, 2022).

La siguiente tabla sintetiza los aportes centrales de las principales definiciones, articulando sus énfasis conceptuales:

Tabla 1 - Principales definiciones del Pensamiento Computacional

Autor	Definición clave	Enfoque principal
Wing (2006)	"Procesos de pensamiento para formular problemas y soluciones de modo que puedan representarse y ejecutarse por un agente que procesa información."	Representación y ejecutabilidad de soluciones.
Wing (2011)	El pensamiento computacional como proceso que permite soluciones ejecutables por humanos, máquinas o sistemas híbridos.	Amplitud socio–técnica del Pensamiento Computacional.
Aho (2012)	Fundado en cómputo, lenguajes, datos y complejidad como puente entre problemas y automatización.	Fundamentos del cómputo como puente conceptual.
Selby y Woollard (2013)	"Conjunto de procesos de pensamiento que implican descomponer, reconocer patrones, abstraer y diseñar algoritmos."	Procesos cognitivos nucleares.
García-Peñalvo y Mendes (2018)	Competencia transversal para comprender, modelar y resolver problemas.	Enfoque interdisciplinar y competencial.

Fuente: Adaptado de: Wing (2006, 2011); Aho (2012); Selby & Woollard (2013); García-Peñalvo & Mendes (2018).

Esta síntesis muestra una progresiva evolución: desde definiciones centradas en la ejecución computacional (Wing, 2006) hacia enfoques cognitivos complejos y aplicados (Brennan & Resnick, 2012; Grover & Pea, 2013; García-Peñalvo & Mendes, 2018). En este tránsito, el pensamiento computacional se consolida como un marco que integra conceptos, prácticas y perspectivas, ampliando su alcance más allá de lo estrictamente técnico.

2.1.1 Perspectiva educativa y relevancia en el currículo contemporáneo

Ambas propuestas coinciden en que el pensamiento computacional se instala como una de las nociones más influyentes en la educación actual debido a su rol en el desarrollo de habilidades cognitivas de orden superior (Wing, 2006, 2011; Bocconi et al., 2016). En términos pedagógicos, el pensamiento computacional se aleja explícitamente de la idea de ser un sinónimo de programación. Las orientaciones de CSTA e ISTE [4] (2011) lo definen como un proceso de resolución de problemas estructurado en etapas: formulación de problemas, análisis de datos, representación por modelos y abstracciones, pensamiento algorítmico y generalización. Esta secuencia corresponde directamente con procesos cognitivos que la literatura didáctica reconoce como fundamentales para el aprendizaje matemático, particularmente en relación con la abstracción, la representación y el razonamiento estructurado (Weintrop et al., 2016).

Por otra parte, la noción ampliada de Brennan y Resnick (2012) agrega dimensiones prácticas (como la iteración, depuración y experimentación) y perspectivas asociadas a la agencia, la expresión personal y la colaboración. Esta visión sitúa al pensamiento computacional como una cultura cognitiva

[4] CSTA, por sus siglas en inglés: Computer Science Teachers Association.
ISTE, por sus siglas en inglés: International Society for Technology in Education.

integrada a las prácticas educativas, y no solo como un conjunto de habilidades técnicas.

Los sistemas educativos del mundo han avanzado en la incorporación del pensamiento computacional desde la educación inicial, las que proponen trayectorias progresivas desde actividades concretas hacia niveles crecientes de abstracción (INTEF, 2022). En todas estas iniciativas, las actividades desenchufadas cumplen un rol clave, pues permiten introducir conceptos complejos sin necesidad de computadores (Bell et al., 2009; Bel et al., 2015).

En el caso chileno, el MINEDUC (2019a) incorpora el pensamiento computacional como competencia transversal en diferentes asignaturas y niveles, y como electivo específico en tercero y cuarto medio dentro de la Formación Diferenciada Matemática. Además, se impulsa su desarrollo mediante políticas como el Plan Nacional de Lenguajes Digitales (MINEDUC, 2019b), el cual articula progresiones formativas desde la educación parvularia hasta la media.

2.1.2 Habilidades del Pensamiento Computacional

La literatura especializada coincide en que el pensamiento computacional puede desagregarse en un conjunto de habilidades cognitivas de orden superior que permiten analizar, estructurar y resolver problemas de manera eficiente en contextos digitales y no digitales. Si bien distintos autores emplean nomenclaturas específicas, las dos propuestas entregadas convergen en cinco componentes nucleares: descomposición, reconocimiento de patrones y generalización, abstracción, pensamiento algorítmico y evaluación/depuración (CAS, 2015; Csizmadia et al., 2015; Brennan & Resnick, 2012; Bordignon & Iglesias, 2019). Estas habilidades constituyen el “esqueleto cognitivo” del pensamiento computacional y resultan especialmente potentes para el diseño didáctico en Matemática, pues establecen un puente natural entre la resolución de problemas matemáticos y los procesos de la Ciencia de la Computación.

Ilustración 1. Habilidades del Pensamiento Computacional

Fuente: Tomado de "Introducción al pensamiento computacional" (p.30), por F. Bordignon y A. Iglesias, (2019). Educar S. E.

a) Descomposición

La descomposición consiste en dividir un problema complejo en partes más simples y manejables, permitiendo reducir la carga cognitiva y posibilitar soluciones incrementales. Desde los aportes de Wing (2011), esta habilidad supone identificar subproblemas, relaciones internas y estructuras jerárquicas que facilitan el análisis y la planificación. Descomponer supone dividir un problema complejo en subproblemas manejables para reducir la carga cognitiva, articulándolo con tareas matemáticas como separar un problema de probabilidad en eventos simples. Esta mirada coincide con Grover y Pea (2013) y Shute et al.

(2017), quienes subrayan el rol de la descomposición en la estructuración del pensamiento.

Bordignon e Iglesias (2019) complementan este enfoque señalando que la descomposición constituye un mecanismo cognitivo mediante el cual los estudiantes aprenden a gestionar la complejidad y organizar pasos lógicos. En Matemática, esta habilidad emerge cuando se segmenta un problema verbal, se identifican variables relevantes o se determinan las operaciones necesarias para avanzar hacia la solución. La descomposición, por tanto, opera como un recurso cognitivo que permite a los estudiantes enfrentar problemas de creciente dificultad.

b) Reconocimiento de patrones y generalización

Ambas propuestas coinciden en que el reconocimiento de patrones y la generalización son dos procesos interrelacionados. Reconocer patrones implica identificar regularidades, estructuras repetidas o similitudes en diferentes problemas o en partes de un mismo problema. Generalizar consiste en extender una solución o procedimiento a nuevas situaciones, lo que permite construir modelos transferibles (Csizmadia et al., 2015; Grover & Pea, 2013).

En Matemática, estas operaciones son fundamentales: identificar series, reconocer estructuras algebraicas o abstraer relaciones geométricas son ejemplos típicos. La generalización es clave en el tránsito desde ejemplos particulares hacia formulaciones generales y conjeturas. Asimismo, Shute et al. (2017) sostienen que esta habilidad permite reutilizar soluciones y construir estrategias que operan sobre clases de problemas, no sobre casos aislados.

Bordignon e Iglesias (2019) refuerzan esta perspectiva con ejemplos concretos, como la generalización de la relación pitagórica, mostrando que la capacidad de reconocer y extrapolar patrones constituye un recurso estructural para el pensamiento matemático y computacional.

c) Abstracción

La abstracción es una de las habilidades más complejas del Pensamiento Computacional, pues implica seleccionar la información relevante, ignorar detalles superfluos y construir representaciones simplificadas, simbólicas, gráficas o computacionales, que capturan la esencia de un problema (Wing, 2006). Esta habilidad permite ocultar lo irrelevante y seleccionar buenas representaciones, lo que se manifiesta directamente en contenidos matemáticos como funciones, álgebra, geometría o modelación.

Para Csizmadia et al. (2015), la abstracción constituye el proceso mediante el cual el estudiante logra manejar la complejidad sin perder significado, generando modelos mentales que permiten operar sobre un problema desde una perspectiva conceptual. Weintrop et al. (2016) agregan que la abstracción se relaciona estrechamente con la habilidad de representar situaciones del mundo real mediante modelos matemáticos o computacionales, un aspecto clave en currículos que integran STEM o programación.

d) Pensamiento algorítmico

El pensamiento algorítmico implica diseñar, describir y ejecutar secuencias finitas de pasos lógicamente ordenados, que permiten resolver un problema de manera reproducible y eficiente (Brennan & Resnick, 2012; Bordignon & Iglesias, 2019). Para fines didácticos, esta habilidad se expresa mediante estructuras de control (secuencias, condiciones, repeticiones y variables) que permiten organizar procedimientos complejos.

Esto demanda diseñar secuencias finitas y ordenadas de pasos con estructuras de control que aseguren reproducibilidad y eficiencia, lo cual se concreta en Matemática a través del uso de pseudocódigos, algoritmos numéricos, procedimientos de factorización, o diagramas de flujo que explicitan el razonamiento detrás de una demostración o cálculo iterativo.

Weintrop et al. (2016) enfatizan que el pensamiento algorítmico es fundamental no solo para programar, sino para estructurar procesos lógicos, comunicar soluciones y automatizar procedimientos, actividades que también forman parte de la enseñanza matemática contemporánea.

e) Evaluación y depuración

La evaluación consiste en verificar la corrección, eficiencia y pertinencia de una solución mientras que la depuración implica identificar, explicar y corregir errores. Esta última es particularmente relevante en procesos iterativos de diseño y resolución (Csizmadia et al., 2015). Evaluar requiere verificar corrección, eficiencia y robustez de la solución, lo cual se traduce en Matemática en la comprobación de resultados, análisis de casos límite, estimaciones y argumentaciones justificadas.

Shute et al. (2017) señalan que la evaluación promueve metacognición y autorregulación, mientras que Bordignon e Iglesias (2019) la vinculan con la revisión sistemática que realizan los estudiantes para garantizar que un procedimiento matemático o computacional funcione correctamente. En contextos educativos, ambas operaciones constituyen oportunidades privilegiadas para desarrollar pensamiento crítico y autonomía intelectual.

2.1.3 Convergencias entre Pensamiento Computacional y Educación Matemática

La literatura analizada muestra que el pensamiento computacional y la matemática comparten estructuras cognitivas fundamentales: abstracción, análisis de patrones, modelación y razonamiento algorítmico. Weintrop et al. (2016) subrayan que esta convergencia abre oportunidades para integrar el pensamiento computacional en la enseñanza matemática no solo como un recurso tecnológico, sino como una herramienta conceptual que mejora la comprensión y el razonamiento.

Desde la didáctica de la matemática, las habilidades de representación, modelación y resolución de problemas, presentes en las Bases Curriculares chilenas, se alinean de manera natural con los procesos del pensamiento computacional (MINEDUC, 2015). Ambas propuestas teóricas destacan que la progresión desde lo concreto a lo abstracto favorece la adquisición de estas habilidades, un principio compartido tanto por el pensamiento computacional como por el enfoque matemático contemporáneo.

2.2 Construccionismo y Pensamiento Computacional

El pensamiento computacional, en su evolución histórica y pedagógica, encuentra una de sus bases epistemológicas más robustas en el construccionismo desarrollado por Seymour Papert (Badilla & Murillo, 2004). Este enfoque sostiene que las personas aprenden mejor cuando construyen artefactos públicos, "objetos para pensar", que pueden ser compartidos, discutidos, depurados y mejorados (Papert & Harel, 1991). Estos objetos, como mencionan Badilla y Murillo (2004) funcionan simultáneamente como mediadores cognitivos y como herramientas epistemológicas. Desde esta perspectiva aprender no consiste en recibir información, sino en construir representaciones externas que permiten modelar ideas, ponerlas a prueba y reflexionar sobre ellas.

Papert (1980) veía en la programación el medio ideal para materializar este enfoque, pues ofrece micromundos con precisión matemática, control explícito de acciones y retroalimentación inmediata. Al programar, el estudiante debe externalizar su pensamiento, traducirlo a instrucciones formales y depurarlo mediante ciclos de prueba y error, lo que obliga a hacer explícitas las reglas, relaciones y suposiciones implícitas en sus soluciones. Esta dinámica coincide plenamente con los procesos esenciales del Pensamiento Computacional.

Como expresa Papert (1980), el giro pedagógico central consiste en que en vez de que la computadora enseñe a los niños, sean los niños quienes la programen. Esta inversión transforma la tecnología en un medio para pensar y no en un fin instrumental, situándola como herramienta epistemológica que permite afinar el pensamiento y desarrollar formas avanzadas de razonamiento lógico y matemático.

2.3 Fundamento Cognitivo del Pensamiento Computacional: Modelo Cattell Horn Carroll (CHC)

El pensamiento computacional posee un fundamento cognitivo profundo que ha sido descrito con mayor precisión a partir de la teoría psicométrica contemporánea. El Modelo Cattell–Horn–Carroll (CHC) constituye una de las teorías más robustas y empíricamente validadas sobre la estructura de la inteligencia humana, integrando capacidades generales, amplias y específicas que intervienen en la resolución de problemas (McGrew, 2009; Schneider & McGrew, 2018). En esta línea, Roman-González et al. (2017) demostraron a través de un estudio factorial confirmatorio que el pensamiento computacional se sostiene de manera significativa en cuatro factores amplios del modelo CHC, evidenciando que esta competencia se fundamenta en procesos cognitivos generales involucrados en tareas matemáticas y científicas.

Razonamiento fluido (Gf) se relaciona con la capacidad de identificar patrones, resolver problemas novedosos, formular analogías y enfrentar situaciones sin una solución previamente aprendida (Cattell, 1963; Flanagan & Dixon, 2014). Roman-González et al. (2017) demostraron que Gf es uno de los predictores más fuertes del desempeño en Pensamiento Computacional, destacando que actividades como el diseño de algoritmos, la abstracción y la generalización son

operaciones que requieren pensamiento inductivo y deductivo, procesos tradicionalmente asociados al razonamiento matemático.

Procesamiento visual (Gv) constituye una habilidad esencial para trabajar con información espacial, interpretar diagramas, visualizar estructuras algorítmicas y comprender representaciones gráficas complejas (Stanton, 1995). Según Roman-González et al. (2017), la dimensión Gv se activa especialmente en tareas de programación en bloques, diagramas de flujo y modelación computacional, donde el estudiante debe manipular mentalmente objetos, secuencias, rutas y relaciones espaciales, lo que evidencia la conexión entre pensamiento computacional y razonamiento geométrico.

Memoria de trabajo (Gsm) es otro de los factores críticos del modelo CHC y se refiere a la capacidad de mantener, manipular y coordinar simultáneamente múltiples elementos mientras se resuelve un problema (Baddeley, 2012). El estudio de Roman-González et al. (2017) concluye que Gsm desempeña un rol fundamental en el pensamiento computacional debido a que la ejecución de algoritmos exige sostener condiciones, variables, bucles y reglas en la mente mientras se anticipan y monitorizan los efectos de cada paso, proceso también reportado en investigaciones sobre resolución matemática (Swanson, 2016).

La principal contribución del estudio de Roman-González et al. (2017) es demostrar que el pensamiento computacional no es un constructo aislado ni exclusivamente tecnológico, sino que se articula directamente con estructuras cognitivas profundas que forman parte del razonamiento humano general. Esta evidencia empírica coincide con la literatura que vincula el pensamiento computacional con habilidades lógicas, matemáticas y de resolución de problemas de alto nivel (Wing, 2006; Shute et al.,2017). El hecho de que el pensamiento computacional dependa de Gf, Gv, Gsm indica que su enseñanza potencia procesos cognitivos transversales como análisis, síntesis, inferencia,

planificación y generalización, todos esenciales en la educación matemática y científica.

Esta convergencia cognitiva entre pensamiento computacional y pensamiento matemático permite afirmar que actividades como la programación, el diseño algorítmico y la modelación computacional no son ejercicios técnicos desconectados de la cognición general, sino expresiones de procesos psicológicos universales. Investigaciones como las de Weintrop et al. (2016) han demostrado que el pensamiento computacional comparte con la matemática procesos como el razonamiento estructurado, la visualización, la abstracción y la argumentación, lo que fortalece la justificación curricular de incorporar pensamiento computacional como un catalizador del desarrollo cognitivo integral.

2.4 Habilidades del Siglo XXI y el Pensamiento Computacional

Las habilidades del siglo XXI constituyen un marco internacional que orienta los sistemas educativos hacia las demandas cognitivas, tecnológicas, sociales y ciudadanas del mundo contemporáneo (Binkley et al., 2012; Fadel, Bialik & Trilling, 2016). Diversos organismos, tales como UNESCO, OCDE[5], ISTE, y en Chile el MINEDUC, han coincidido en que estas habilidades son esenciales para la formación integral de las personas y constituyen un fundamento para el desarrollo del pensamiento computacional en el currículo escolar (UNESCO, 2023; OCDE, 2024; ISTE, 2017, MINEDUC, 2021a).

[5] OCDE, por sus siglas en español: Organización para la Cooperación y el Desarrollo Económicos

De acuerdo con MINEDUC (2021a), las Habilidades del Siglo XXI se organizan en cuatro ámbitos:

1. maneras de pensar,
2. maneras de trabajar,
3. herramientas para trabajar,
4. y herramientas para vivir en el mundo.

Cada uno de estos ámbitos presenta subámbitos específicos, los cuales se articulan de manera directa con el Pensamiento Computacional, tal como se detalla a continuación.

Ilustración 2. Un recorrido por las habilidades para el siglo XXI

Fuente: Tomado de "Un recorrido por las habilidades para el siglo XXI", por MINEDUC, (2021a), Currículum Nacional.

2.4.1 Maneras de pensar

Este ámbito incluye creatividad/innovación, pensamiento crítico y metacognición, competencias consideradas centrales para aprender en contextos caracterizados por la complejidad y el cambio acelerado (Binkley et al., 2012; Bialik & Fadel, 2015).

a) Creatividad e innovación

La creatividad es fundamental para generar ideas originales y transformarlas en productos, soluciones o expresiones significativas (Bialik & Fadel, 2015). En el marco del Pensamiento Computacional, la creatividad se relaciona directamente con la producción de algoritmos, modelos, animaciones o simulaciones; procesos que requieren idear, diseñar, probar y ajustar soluciones (Brennan & Resnick, 2012). Maggio (2018) enfatiza que la creatividad será una de las competencias más demandadas por los empleos futuros, dado que las innovaciones tecnológicas requieren pensamiento flexible y adaptativo.

b) Pensamiento crítico

El pensamiento crítico implica analizar información, evaluar argumentos, identificar inconsistencias y tomar decisiones fundamentadas (Scott, 2015). Este proceso dialoga estrechamente con el Pensamiento Computacional, el cual exige comparar rutas de solución, evaluar la validez de algoritmos y revisar críticamente los resultados mediante procesos de depuración (CSTA & ISTE, 2011). Montt (2018) señala que los sistemas educativos deben promover el pensamiento crítico como respuesta a la velocidad del cambio científico y tecnológico.

c) Metacognición

La metacognición comprende la capacidad de reflexionar sobre el propio aprendizaje, planificarlo y monitorizarlo (Bialik & Fadel, 2015). En el Pensamiento Computacional, esta dimensión se manifiesta en la evaluación y ajuste sistemático de soluciones, la reflexión sobre errores, la revisión del proceso

algorítmico y la capacidad de explicitar cómo se tomó cada decisión (Shute et al., 2017).

2.4.2 Maneras de trabajar

Este ámbito incluye comunicación y colaboración, competencias esenciales para desenvolverse en entornos interdependientes y digitalmente mediados (Binkley et al., 2012).

a) Comunicación

La comunicación implica expresar ideas con claridad en diversos formatos y lenguajes (Bialik & Fadel, 2015). El pensamiento computacional potencia la comunicación multimodal al requerir que los estudiantes expresen ideas mediante instrucciones, algoritmos, diagramas de flujo o modelos computacionales (Weintrop et al., 2016). En entornos como Scratch, las producciones programadas funcionan como mensajes complejos que integran narración, visualidad y lógica (Resnick, 2017).

b) Colaboración

La colaboración involucra construir soluciones colectivas, coordinar acciones y negociar significados (Scott, 2015). En el Pensamiento Computacional, la colaboración se evidencia en actividades de programación conjunta, diseño de proyectos y participación en comunidades digitales de aprendizaje (Bers et al., 2018). CPEIP[6] (2021) refuerza que el trabajo colaborativo constituye una expectativa profesional docente para promover aprendizajes profundos en el aula.

[6] CPEIP, por sus siglas en español: Centro de Perfeccionamiento, Experimentación e Investigaciones Pedagógicas.

2.4.3 Herramientas para trabajar

Este ámbito incluye alfabetización digital y uso de TIC, entendidas como habilidades para navegar, producir y participar críticamente en entornos digitales (Binkley et al., 2012).

a) Alfabetización digital

La alfabetización digital comprende interpretar, producir y evaluar información en medios digitales (Bialik & Fadel, 2015). Maggio (2018) afirma que la fluidez digital, más que la programación, será esencial para todos los trabajos del futuro. El pensamiento computacional promueve esta alfabetización al exigir el uso de datos, simulaciones, algoritmos y herramientas digitales para resolver problemas y construir conocimiento (Google for Education, 2016).

b) Uso crítico y ético de tecnologías (TIC)

El uso de TIC implica seleccionar herramientas digitales, evaluar sus límites y actuar con responsabilidad en entornos virtuales (UNESCO, 2023). En la propuesta de actualización curricular chilena (MINEDUC, 2024a), se enfatiza que los estudiantes deben actuar como ciudadanos digitales responsables, capaces de comprender el funcionamiento de tecnologías, incluidas las inteligencias artificiales, y utilizarlas de manera efectiva y ética. El pensamiento computacional contribuye a esta competencia al promover comprensión técnica y toma de decisiones fundamentadas en contextos digitales (Bialik & Fadel, 2015).

2.4.4 Herramientas para vivir en el mundo

Este ámbito incluye autonomía, responsabilidad y ciudadanía, competencias orientadas a la participación en sociedades democráticas y tecnológicas (Bialik & Fadel, 2015).

a) Autonomía y autorregulación

La autonomía implica tomar decisiones informadas, gestionar el propio aprendizaje y actuar con independencia (MINEDUC, 2021a). En el Pensamiento Computacional, esta habilidad se refleja en la capacidad de planificar algoritmos, revisar errores, ajustar estrategias y perseverar frente a desafíos (Shute et al., 2017).

b) Ciudadanía responsable y participación

La ciudadanía digital requiere comprender los impactos sociales de la tecnología y participar de manera ética en entornos digitales (UNESCO, 2016). El pensamiento computacional favorece esta competencia al permitir que los estudiantes comprendan la lógica de los sistemas digitales, evalúen riesgos, analicen datos y participen activamente en comunidades tecnológicas (Maggio, 2018). Además, la Propuesta de Actualización de Bases Curriculares (2024) sitúa al pensamiento computacional como una herramienta clave para que los estudiantes actúen como ciudadanos críticos en un entorno digitalizado.

2.5 Enfoque STEAM y el Pensamiento Computacional

El enfoque STEAM (Science, Technology, Engineering, Arts, Mathematics) constituye un marco pedagógico interdisciplinar que integra ciencias, tecnología, ingeniería, artes y matemáticas para promover aprendizajes profundos mediante proyectos de diseño, modelación, experimentación y resolución creativa de problemas (Yakman, 2008; Martín-Páez et al., 2019). Desde una perspectiva de integración disciplinar, STEAM opera como un escenario para problematizar, modelar y crear soluciones que movilizan de forma simultánea contenido matemático, prácticas de ingeniería y herramientas tecnológicas, favoreciendo la creatividad, la colaboración y el pensamiento crítico en contextos reales.

Según Sanders (2009), STEAM, denominado STE(A)M en su conceptualización original, debe comprenderse como una metadisciplina, en la que los saberes de ciencia, tecnología, ingeniería y matemáticas no se abordan como compartimentos estancos, sino como un sistema integrado, orientado al desarrollo de innovación, competitividad y resolución de problemas complejos. Esta concepción es especialmente útil para fundamentar proyectos de modelación y análisis de datos en Matemática, permitiendo transitar progresivamente desde actividades desenchufadas hacia experiencias enchufadas, mediadas por simulaciones, programación y robótica.

El enfoque STEAM promueve, asimismo, contextos de aprendizaje auténticos en los que los estudiantes pueden diseñar soluciones mediante prototipos, simulaciones computacionales o artefactos tecnológicos, posicionando a la Matemática como un lenguaje que permite comprender, describir y optimizar dichas soluciones (Yakman, 2011). La integración de las artes añade una dimensión expresiva y creativa que amplifica las posibilidades de exploración y comunicación de ideas en proyectos interdisciplinarios.

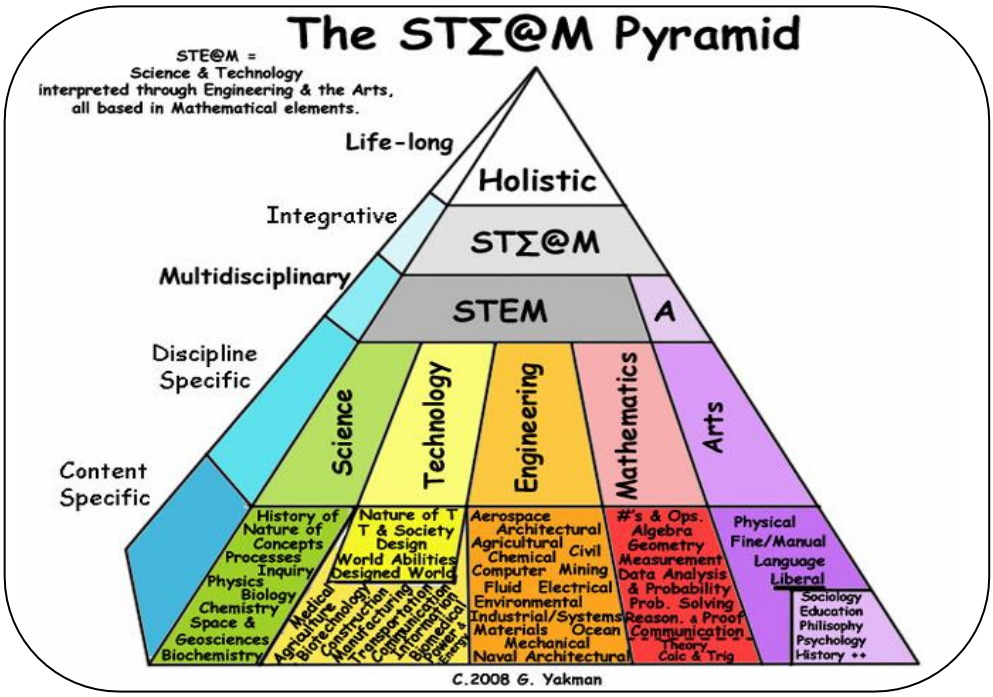


Ilustración 3. Modelo STEAM

Fuente: Tomado de "STE@M Educational Model: An overview of creating a model of integrative education" (p. 17), por G. Yakman, (2008).

En este marco, el pensamiento computacional actúa como andamiaje cognitivo que articula los procesos implicados en STEAM. Weintrop et al. (2016) sostienen que el pensamiento computacional proporciona un marco conceptual y metodológico para modelar fenómenos, analizar datos, construir simulaciones, diseñar algoritmos y automatizar procedimientos, aspectos esenciales en las prácticas científicas, tecnológicas y matemáticas del siglo XXI. La programación se convierte en un vehículo representacional que permite:

- expresar modelos matemáticos en lenguajes formales,
- visualizar relaciones complejas,
- probar hipótesis mediante simulación,

- y diseñar soluciones iterativas y eficientes.

En el contexto chileno, los principios STEAM se han incorporado progresivamente en iniciativas como la ruta formativa en pensamiento computacional del MINEDUC (2022) y el Plan Nacional de Lenguajes Digitales del MINEDUC (2019b), que integran diseño, creatividad, programación y pensamiento algorítmico como ejes formativos. De igual manera, la Propuesta de Actualización Curricular 2024 (MINEDUC, 2024a) refuerza esta dirección al señalar la necesidad de promover experiencias que articulen tecnologías digitales, creación computacional y modelación matemática como competencias esenciales del ciudadano contemporáneo.

2.6 Resolución de problemas

La resolución de problemas se ha consolidado como una de las competencias clave para la educación contemporánea. De acuerdo con Jonassen (2000), la resolución de problemas implica un proceso cognitivo complejo que demanda identificar la situación problemática, analizar la información disponible, generar posibles alternativas y seleccionar estrategias pertinentes para alcanzar una solución.

En el plano internacional, la OCDE (2014) ha situado la resolución de problemas en el centro de las competencias para la vida en el siglo XXI. En las evaluaciones PISA, se define como la capacidad de un individuo para participar en el procesamiento cognitivo para entender y resolver situaciones problemáticas donde un método de solución no es inmediatamente obvio. Incluye la disposición a involucrarse con tales situaciones con el fin de alcanzar el potencial como un ciudadano constructivo y reflexivo.

Por su parte, la UNESCO plantea que la Resolución de Problemas es una competencia transversal indispensable para que los estudiantes puedan

participar activamente en la sociedad global, marcada por la transformación digital. Desde esta perspectiva, no solo se trata de superar obstáculos académicos, sino también de formar ciudadanos capaces de enfrentar dilemas sociales, éticos y tecnológicos con creatividad y sentido crítico (UNESCO, 2023).

En el contexto chileno, el MINEDUC (2012; 2015; 2019c)ha incorporado la resolución de problemas como una habilidad transversal en las bases curriculares y en los marcos de competencias docentes. El Marco Orientador de Competencias Digitales Docentes (MINEDUC, 2025) enfatiza que los estudiantes deben ser capaces de resolver problemas de información, comunicación y conocimiento en ambientes digitales, enfrentando también dilemas sociales y éticos. Asimismo, se vincula la resolución de problemas con el aprendizaje profundo, entendido como la capacidad de transformar el conocimiento para aplicarlo en la resolución de situaciones reales y contextualizadas, lo que refuerza su carácter práctico y formativo.

2.6.1 Aprendizaje basado en Resolución de Problemas

El Aprendizaje Basado en Problemas (ABP) se concibe como una metodología activa que sitúa al estudiante en el centro del proceso de aprendizaje, en la que la resolución colaborativa de un problema complejo promueve la construcción significativa del conocimiento y el desarrollo de habilidades cognitivas superiores (Hmelo-Silver, 2004). Esta aproximación enfatiza el trabajo cooperativo, la formulación de explicaciones, la autorregulación y la reflexión constante, aspectos que resultan fundamentales para el desarrollo de habilidades cognitivas y sociales en contextos educativos diversos.

En el contexto de la enseñanza de las matemáticas, el ABP ha demostrado ser una herramienta eficaz para fomentar el pensamiento crítico, la autonomía y la capacidad de aplicar conocimientos en situaciones reales. Arévalo et al. (2024) señalan que esta metodología permite a los estudiantes asumir un rol activo en su aprendizaje, relacionando los contenidos matemáticos con problemas

vinculados a su entorno profesional, especialmente en carreras como la ingeniería agropecuaria.

Los estudios recientes evidencian que el ABP favorece el desarrollo de competencias clave en educación superior. Grández et al. (2024) muestran que la implementación sistemática del ABP influye significativamente en el desarrollo del pensamiento crítico y en habilidades cognitivas de alta demanda, observándose un progreso sustantivo en el desempeño de los estudiantes tras participar en sesiones estructuradas bajo esta metodología. Estos resultados destacan que el ABP no solo fortalece procesos analíticos complejos, sino que también promueve la participación activa, la toma de decisiones fundamentadas y la colaboración, consolidándose como una estrategia eficaz para el aprendizaje profundo en la formación universitaria.

Según Reina Neira et al. (2016), el Aprendizaje Basado en Problemas se desarrolla mediante una secuencia estructurada que incluye la presentación del problema, la clarificación de conocimientos previos, la formulación de objetivos de aprendizaje, la investigación autónoma y la discusión grupal. Esta organización permite activar conocimientos relevantes, orientar la búsqueda de información y consolidar la comprensión mediante la argumentación colaborativa, favoreciendo un aprendizaje significativo y contextualizado.

2.6.2 Aprendizaje de Resolución de problemas mediante Pensamiento Computacional

La relación entre pensamiento computacional y pensamiento matemático ha sido ampliamente documentada. Grover y Pea (2013) señalan que ambos comparten procesos fundamentales como la descomposición, la formulación de algoritmos y la abstracción, lo que permite que el razonamiento matemático y el computacional se potencien mutuamente. Estos procesos favorecen la

experimentación, la iteración y la flexibilidad cognitiva, elementos esenciales para desarrollar autonomía y aprendizaje autorregulado en los estudiantes.

En esta misma línea, Gram-Hansen y Jonasen (2019) sostienen que la relación entre el pensamiento computacional y el ABP es de carácter recíproco, pues ambos enfoques se enriquecen mutuamente en contextos de aprendizaje significativo. En consecuencia, el pensamiento computacional no solo se ve fortalecido al integrarse en experiencias de ABP, sino que también contribuye a mejorar el proceso de resolución de problemas mediante la abstracción, la descomposición y el diseño de soluciones algorítmicas, reforzando su valor como competencia transversal en la educación actual.

2.7 Modelos de Integración Pedagógica

La integración del pensamiento computacional en la enseñanza de la Matemática requiere un marco teórico capaz de articular tres dimensiones inseparables: qué se enseña (contenido disciplinar), cómo se enseña (pedagogía) y con qué herramientas se enseña (tecnología). Sin un modelo que ordene estas relaciones, la integración tecnológica corre el riesgo de ser superficial, instrumental o desconectada de los objetivos curriculares (Hernando, 2015).

En la literatura internacional, dos marcos han adquirido especial relevancia para comprender y orientar los procesos de integración de tecnología y pensamiento computacional en el aula: Modelo TPACK y Modelo CTPK-12. Ambos modelos convergen con las orientaciones del sistema educativo chileno, el cual exige que la docencia incorpore tecnologías digitales alineadas con el currículo, la didáctica y las competencias del siglo XXI (CPEIP, 2021; MINEDUC, 2021b), destacando la ciudadanía digital, el pensamiento crítico y la resolución de problemas como ejes centrales del aprendizaje moderno.

2.7.1 Modelo TPACK

El Modelo TPACK (Technological Pedagogical Content Knowledge), desarrollado por Mishra y Koehler (2006), plantea que el conocimiento docente necesario para integrar tecnología en el aula se organiza en tres dimensiones interrelacionadas:

- CK (Content Knowledge): conocimiento disciplinar, en este caso, Matemática.
- PK (Pedagogical Knowledge): estrategias didácticas, comprensión del aprendizaje, evaluación, gestión del aula.
- TK (Technological Knowledge): conocimiento y uso de tecnologías, herramientas digitales, lenguajes de programación.

La intersección de estas dimensiones genera el TPACK, un conocimiento especializado que habilita al docente para tomar decisiones coherentes y fundamentadas acerca de cómo integrar tecnología con contenido y pedagogía.

En el contexto de la enseñanza de la Matemática con Pensamiento Computacional, el modelo TPACK permite seleccionar tecnologías pertinentes (pseudocódigo, diagramas, Scratch, Python) para representar, modelar y resolver situaciones matemáticas. Lye & Koh (2014) señalan que TPACK proporciona una arquitectura conceptual robusta para integrar el pensamiento computacional en el aula, pues obliga al docente a justificar didácticamente el uso de herramientas digitales en función de objetivos de aprendizaje matemático.

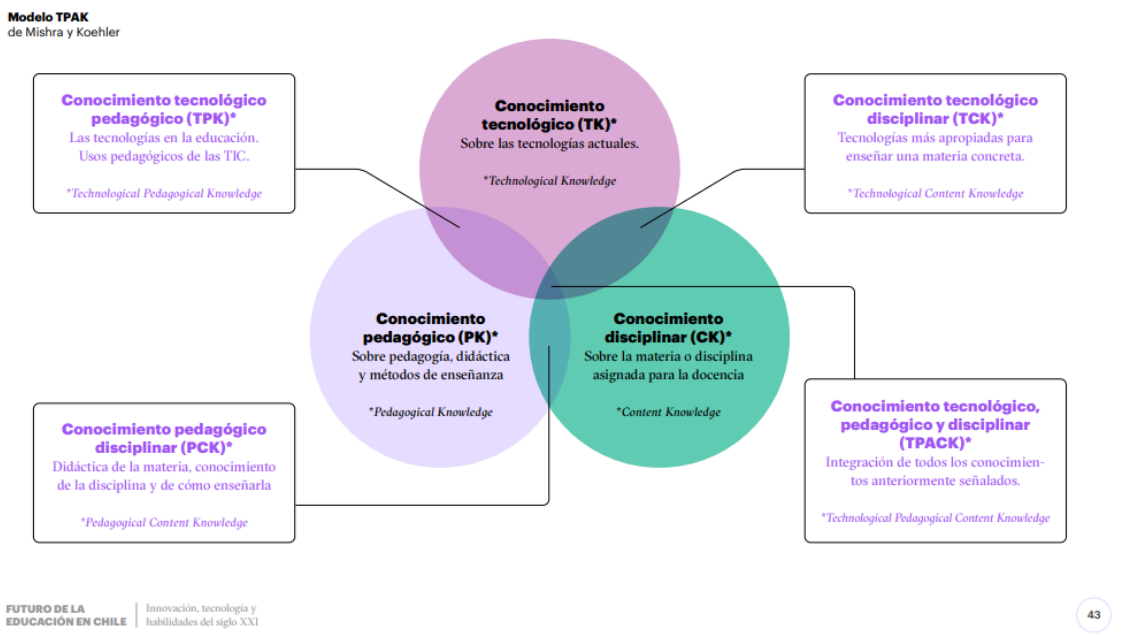


Ilustración 4. Modelo TPAK de Mishra y Koehler

Fuente: Tomado de "Futuro de la educación en Chile: Innovación, tecnología y habilidades del siglo XXI" (p. 43), por Fundación País Digital, (2023).

La Propuesta de Actualización Curricular 2024 (MINEDUC, 2024a) y el Marco para la Buena Enseñanza (CPEIP, 2021) convergen con este enfoque al establecer que los docentes deben diseñar experiencias de aprendizaje mediadas por tecnologías digitales que promuevan habilidades de ciudadanía digital, resolución de problemas, modelación y creación computacional.

2.7.2 Modelo CTPK-12

El modelo CTPK-12 (Computational Thinking Pedagogical Knowledge), desarrollado por Tikva y Tambouris (2021), constituye una adaptación específica del TPACK al dominio del pensamiento computacional en educación escolar. Este modelo identifica tres saberes esenciales:

- CTK (Computational Thinking Knowledge): conceptos, prácticas y perspectivas del Pensamiento Computacional.

- PK (Pedagogical Knowledge): estrategias didácticas para enseñar Pensamiento Computacional, como explicitación, iteración, depuración, retroalimentación y representaciones múltiples.
- TK (Technological Knowledge): conocimiento de herramientas para enseñar Pensamiento Computacional: lenguajes visuales, simuladores, editores de pseudocódigo, entornos textuales.

El CTPK-12 reconoce que enseñar pensamiento computacional exige un conocimiento didáctico diferente al de la enseñanza de programación. No basta con saber codificar; se requiere comprender cómo aprenden los estudiantes, qué representaciones son cognitivamente accesibles, cómo se articulan progresiones desde actividades desenchufadas hasta programación textual, y cómo evaluar el pensamiento computacional en relación con habilidades transversales como comunicación o resolución de problemas.

Tikva y Tambouris (2021) sostienen que el CTPK-12 se vuelve especialmente relevante para integrar pensamiento computacional con Matemática, pues orienta al docente a diseñar representaciones puente, tales como diagramas de flujo, pseudocódigo, tablas de valores, simulaciones matemáticas y gráficos dinámicos. En la enseñanza escolar chilena, este modelo dialoga estrechamente con la Ruta de Aprendizaje del pensamiento computacional del MINEDUC (2022), que organiza progresiones para secuencialidad, condiciones, bucles, algoritmos, depuración y variables.

2.8 Aprendizaje del Pensamiento Computacional

El aprendizaje del pensamiento computacional en el aula es un proceso gradual y no lineal que exige una arquitectura didáctica cuidadosamente diseñada, capaz de articular experiencias analógicas y digitales, representaciones múltiples, explicitación de procedimientos, retroalimentación iterativa y metarreflexión

(Brennan & Resnick, 2012; Shute et al., 2017). Uno de los pilares fundamentales en esta arquitectura es la distinción entre pensamiento computacional desenchufado y pensamiento computacional enchufado. Bell et al. (2009, 2015) argumentan que las actividades desenchufadas constituyen una vía pedagógica altamente eficaz para introducir los conceptos esenciales de la computación sin la carga cognitiva asociada a las interfaces o a la sintaxis de un lenguaje de programación. Estas actividades permiten a los estudiantes "pensar el problema por sí solos" antes de interactuar con una máquina, promoviendo la comprensión profunda de ideas como secuencialidad, patrones, decisiones, complejidad algorítmica o compresión de datos mediante materiales del entorno cotidiano y dinámicas lúdicas.

En la fase enchufada, el uso de entornos como Scratch facilita la transición hacia la representación computacional mediante bloques, permitiendo que los estudiantes se conviertan en "creadores activos" que externalizan ideas complejas, modelan fenómenos y resuelven problemas reales (Resnick et al., 2009). Lye y Koh (2014) sostienen que Scratch reduce la barrera sintáctica sin sacrificar la estructura algorítmica, mientras que lenguajes textuales como Python ofrecen un puente hacia la modelación matemática avanzada, habilitando simulaciones, análisis de datos y la representación eficiente de patrones funcionales. Esta complementariedad entre lo manual y lo digital, lo concreto y lo simbólico, refleja una progresión cognitiva esencial para el desarrollo del Pensamiento Computacional.

2.8.1 Ruta de Aprendizaje del Pensamiento Computacional

En Chile, esta lógica de progresión ha sido sistematizada por la Ruta de Aprendizaje del pensamiento computacional del MINEDUC (2022), construida a partir del marco de trayectorias de Rich et. al (2017; 2018; 2019; 2020). Estas trayectorias identifican dependencias conceptuales para secuencialidad, condiciones, repeticiones, descomposición y depuración, proponiendo una

progresión desde la comprensión de instrucciones simples hasta el diseño de algoritmos complejos. Entre los aportes más significativos se encuentra el énfasis en representaciones previas a la programación (diagramas, pseudocódigo, tarjetas, modelos físicos) y la claridad en la transición entre programación por bloques y programación textual, aspecto crítico para la alfabetización computacional escolar.

La progresión didáctica por trayectorias de Rich es el eje que conecta ambas modalidades:

- Para secuencialidad, se trabajan dos ramas, precisión y orden de instrucciones, con dependencias internas; el nivel inicial enfatiza precisión, el intermedio coordina precisión y orden, y a partir de allí se empalma con repeticiones y condiciones (Rich et al., 2017).

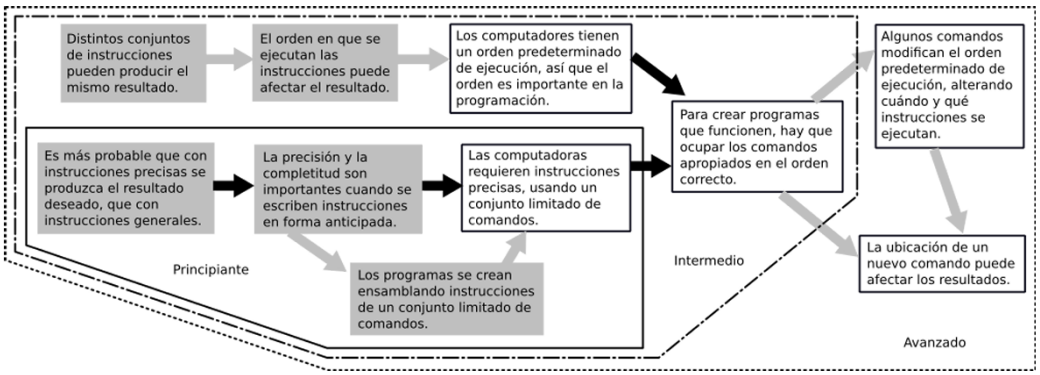


Ilustración 5. Trayectoria de Aprendizaje: Secuencialidad

Fuente: Tomado de "Ruta de Aprendizaje para el Pensamiento Computacional", por MINEDUC (2022). *Ruta de aprendizaje para el pensamiento computacional*. Departamento de Ciencias de la Computación, Universidad de Chile.

- Para repeticiones, se avanza desde ciclos simples y finitos hacia la composición de distintos tipos de ciclos, culminando en ciclos anidados y en el uso de variables y condiciones para inicializar o detener la iteración (Rich et al., 2017).

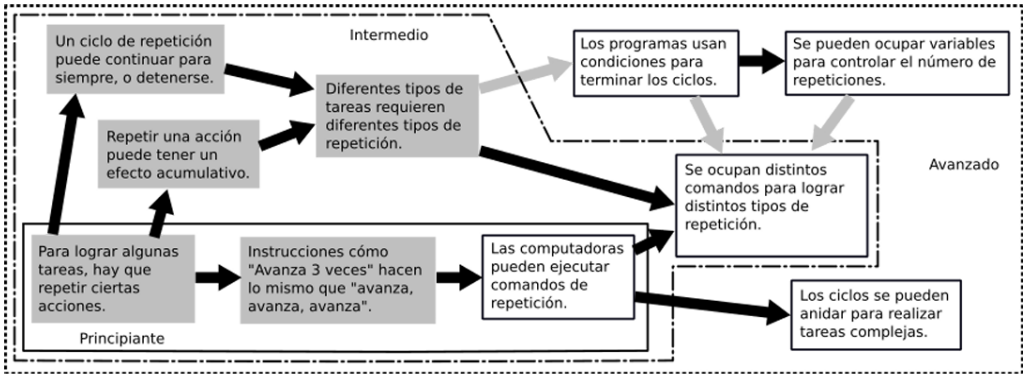


Ilustración 6. Trayectoria de Aprendizaje: Repeticiones

Fuente: : Tomado de "Ruta de Aprendizaje para el Pensamiento Computacional", por MINEDUC (2022). *Ruta de aprendizaje para el pensamiento computacional*. Departamento de Ciencias de la Computación, Universidad de Chile.

- Para condiciones, la trayectoria progresa desde el concepto binario y programas basados en eventos (sin cláusulas explícitas) hasta múltiples condiciones anidadas y ramas complejas con variables lógicas (Rich et al., 2017).

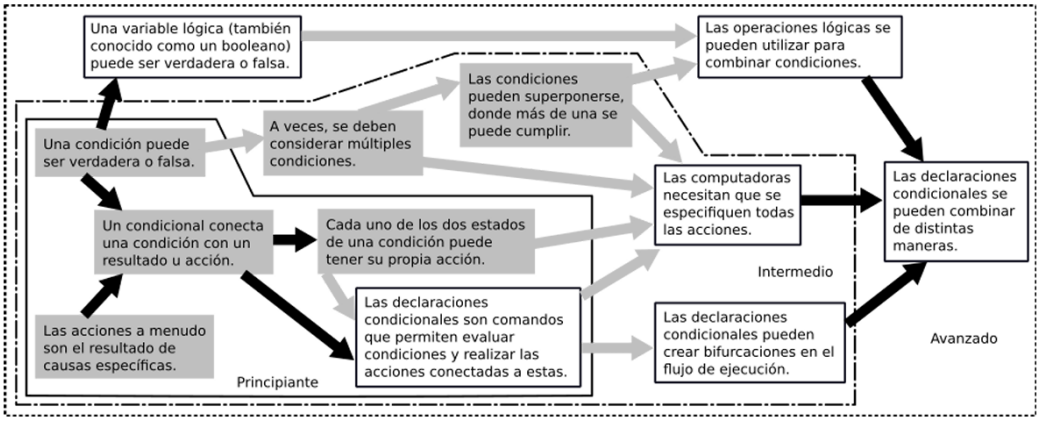


Ilustración 7. Trayectoria de Aprendizaje: Condiciones

Fuente: Tomado de "Ruta de Aprendizaje para el Pensamiento Computacional", por MINEDUC (2022). *Ruta de aprendizaje para el pensamiento computacional*. Departamento de Ciencias de la Computación, Universidad de Chile.

- En descomposición, se transita por objetivos que incluyen identificar componentes de un sistema, dividir tareas a un nivel de abstracción apropiado y comprender la naturaleza jerárquica de los sistemas, integrando estrategias de planificación top-down y ensamblaje bottom-up mediante código modular (Rich et al., 2018).

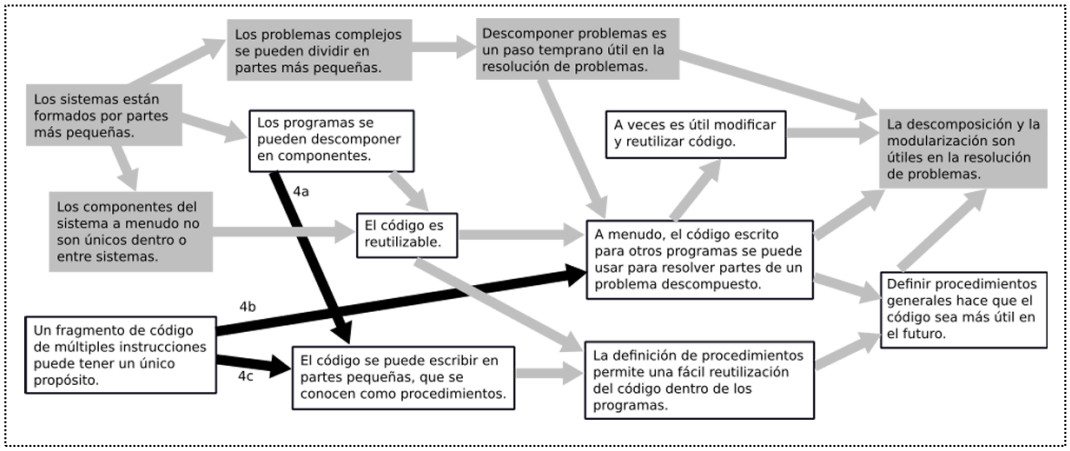


Ilustración 8. Trayectoria de Aprendizaje: Descomposición

Fuente: Tomado de "Ruta de Aprendizaje para el Pensamiento Computacional", por MINEDUC (2022). *Ruta de aprendizaje para el pensamiento computacional*. Departamento de Ciencias de la Computación, Universidad de Chile.

- Finalmente, depuración enfatiza estrategias para identificar y corregir errores, tipos de errores y el rol de los errores en la resolución de problemas, proponiendo un ciclo que anticipa la salida esperada, ejecuta y verifica el cumplimiento de la condición de éxito, análogo a método científico en Matemática (Rich et al., 2019).

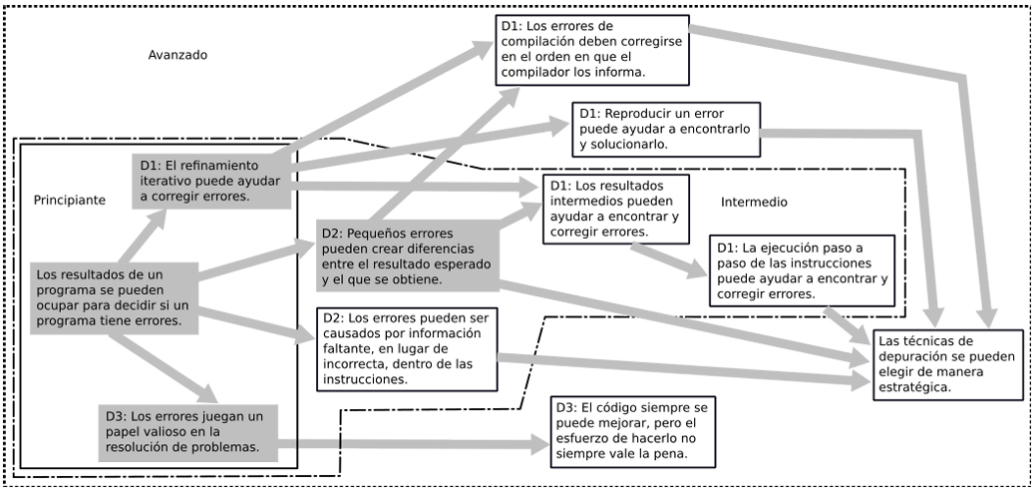


Ilustración 9. Trayectoria de Aprendizaje: Depuración.

Fuente: Tomado de "Ruta de Aprendizaje para el Pensamiento Computacional", por MINEDUC (2022). *Ruta de aprendizaje para el pensamiento computacional*. Departamento de Ciencias de la Computación, Universidad de Chile.

Las trayectorias de Rich et al. describen con precisión cómo evolucionan las ideas del Pensamiento Computacional: en secuencialidad, la progresión va desde instrucciones simples hasta la coordinación con ciclos y condiciones; en repeticiones, se avanza desde ciclos finitos básicos hacia la composición de ciclos complejos que dependen de variables; en condiciones, la evolución ocurre desde decisiones binarias hasta anidamiento y múltiples criterios lógicos; en

descomposición, se transita desde identificar partes de un problema hasta emplear planificación jerárquica y diseño modular; en depuración, se promueve la identificación y corrección sistemática de errores mediante ciclos de anticipación, verificación y ajuste, análogos al método científico en Matemática (Rich et al., 2017, 2018, 2019). Este resultado de investigación ha sido adoptado íntegramente por el Estado de Chile, otorgándole legitimidad nacional y convirtiéndolo en base estructural para el diseño curricular y didáctico.

Como resultado el Programa de Estudio "Pensamiento Computacional y Programación" (3.º y 4.º medio) refuerza esa progresión al explicitar una transición metodológica "desde el lenguaje natural hacia el lenguaje computacional", donde la descomposición, la abstracción y la repetición de patrones se plasman primero en representaciones manuales, luego en programación por bloques y finalmente en lenguajes textuales (MINEDUC, 2021b). Los OA señalan explícitamente la necesidad de aplicar conceptos de ciencias de la computación en la creación de código, construir estrategias colaborativas para resolver problemas no rutinarios y argumentar procedimientos con lenguaje simbólico, manual o digital.

Esta estructura progresiva no solo facilita la alfabetización computacional, sino que fortalece la alfabetización matemática al promover razonamientos secuenciales, analíticos y estructurados. En este sentido, el aprendizaje del pensamiento computacional se comprende como un camino formativo que integra diseño, representación, iteración y reflexión, lo que permite a los estudiantes avanzar desde actividades concretas hacia la modelación compleja, articulando pensamiento matemático y pensamiento computacional en un continuo de creciente sofisticación cognitiva.

2.9 Programación

La programación constituye un lenguaje de representación y acción que permite modelar, simular y resolver problemas mediante procedimientos estructurados. Desde el enfoque construccionista (Papert, 1980; Resnick et al., 2009), programar implica externalizar el pensamiento, depurar ideas y construir modelos ejecutables que hacen visible la lógica del razonamiento matemático. En el aula de Matemática, la programación amplifica el análisis de patrones, la comprensión de relaciones funcionales, la visualización de modelos dinámicos y la validación de hipótesis mediante simulaciones (Weintrop et al., 2016).

Para integrar programación y Matemática de forma progresiva y coherente con el desarrollo del Pensamiento Computacional, se organizan tres niveles de representación y complejidad: (1) pseudocódigo y diagramas de flujo, (2) programación por bloques, y (3) programación en texto. Esta secuencia, reconocida por el Programa de Estudio de pensamiento computacional y Programación (MINEDUC, 2021b), permite transitar desde ideas cercanas al lenguaje cotidiano hacia la formalización algorítmica propia de la modelación matemática.

2.9.1 Pseudocódigo y diagramas de flujo

El primer nivel de programación se centra en representaciones previas al código, que permiten estructurar el pensamiento sin la carga cognitiva de la sintaxis. Weerdt (2019) define el pseudocódigo como una descripción clara, compacta y no ambigua de un algoritmo, situada entre el lenguaje natural y el lenguaje de programación. Su valor didáctico radica en que ayuda a los estudiantes a concentrarse en la lógica del procedimiento matemático antes de implementarlo digitalmente.

Los diagramas de flujo complementan esta representación textual mediante un formato visual. Chinofunga et al. (2024) los describen como una estructura gráfica de decisiones, procesos y rutas alternativas, que facilita comparar estrategias, justificar procedimientos y verificar la coherencia de un algoritmo. En Matemática, estos diagramas ayudan a representar procesos como resoluciones por casos, progresiones aritméticas, algoritmos numéricos o comprobaciones sistemáticas.

El Programa de Estudio del Electivo de Pensamiento Computacional y Programación (MINEDUC, 2021b) enfatiza que pseudocódigo y diagramas son "representaciones equivalentes de un mismo problema" (p.69) y recomienda trazas con casos de prueba para validar la solución antes de codificarla.

2.9.2 Programación por bloques

El segundo nivel introduce entornos visuales como el de Scratch donde la estructura algorítmica se construye arrastrando bloques. Este enfoque reduce errores de sintaxis y favorece que los estudiantes se concentren en la lógica del control de flujo (Resnick et al., 2009; Maloney et al., 2010).

La programación por bloques es especialmente útil en Matemática, porque permite representar bucles, decisiones, iteraciones numéricas, paralelismo y manejo básico de variables mediante una interfaz visual que reduce la carga cognitiva asociada a la sintaxis. Investigaciones en educación en computación muestran que los entornos basados en bloques favorecen la comprensión conceptual, ya que hacen visible la estructura del algoritmo y permiten observar de manera directa cómo se ejecutan expresiones, repeticiones y funciones simples (Grover & Basu, 2017).

En Chile, la Ruta de Aprendizaje del Pensamiento Computacional incluye explícitamente Scratch como herramienta para trabajar secuencialidad, decisiones y bucles en progresión gradual (MINEDUC, 2022). Esto permite

vincular contenidos como funciones lineales, patrones geométricos o simulaciones probabilísticas con estructuras algorítmicas básicas.

2.9.3 Programación en código

El tercer nivel se basa en lenguajes textuales como Python, que permiten modelación avanzada, manipulación de datos y simulaciones matemáticas con mayor expresividad formal. Python es recomendado en educación media por su sintaxis sencilla y la disponibilidad de bibliotecas orientadas a Matemática, estadística y visualización (Lye & Koh, 2014).

La programación textual incrementa la demanda cognitiva: requiere modular, abstraer, depurar y manejar estructuras de control sin apoyo gráfico. Esta exigencia fortalece habilidades como generalización, modelación matemática y pensamiento lógico avanzado (Weintrop et al., 2016).

El MINEDUC (2021b) justifica su incorporación como puente natural desde los bloques hacia aplicaciones complejas, permitiendo trabajar funciones, ecuaciones, modelos geométricos, probabilidades, análisis de datos y simulaciones experimentales. Por ejemplo, resolver ecuaciones cuadráticas, modelar movimiento parabólico o estimar probabilidades mediante simulación Monte Carlo.

2.10 Plataformas para el Desarrollo del Pensamiento Computacional

El desarrollo del Pensamiento Computacional (Pensamiento Computacional) requiere entornos que permitan representar, modelar y ejecutar algoritmos de manera accesible, flexible y pedagógicamente significativa. Tal como señalan Grover y Pea (2013), la elección de herramientas digitales influye directamente en la forma en que los estudiantes conceptualizan, externalizan y depuran ideas

computacionales, por lo que los entornos deben favorecer la comprensión gradual de estructuras algorítmicas y la transición desde lo concreto hacia lo abstracto. En esta misma línea, el Programa de Estudio pensamiento computacional y Programación del MINEDUC (2021b) establece que la enseñanza del pensamiento computacional debe apoyarse en plataformas que faciliten progresiones representacionales. Considerando estas orientaciones nacionales e internacionales, tres plataformas se han consolidado como recursos esenciales en el contexto global y chileno para la enseñanza progresiva del Pensamiento Computacional: Scratch, Code.org y Google Colab. Cada una responde a distintos niveles de complejidad cognitiva y ofrece oportunidades complementarias para articular Matemática, programación y resolución de problemas dentro de un marco didáctico coherente.

2.10.1 Scratch

Scratch es un entorno visual de programación creado por el MIT Media Lab, diseñado para que aprendices de todas las edades creen proyectos interactivos mediante bloques encajables. Desde la perspectiva construccionista, Scratch promueve la creación de artefactos significativos y la externalización de ideas a través de la programación, favoreciendo la construcción activa del conocimiento (Resnick et al., 2009; Resnick, 2017).

En la enseñanza de la Matemática, Scratch permite modelar conceptos que tradicionalmente resultan abstractos:

- Animaciones geométricas, como rotaciones y traslaciones.
- Gráficos dinámicos, que muestran variación y cambio.
- Simulaciones de proporcionalidad, razones y funciones.
- Algoritmos de cálculo y juegos basados en lógica aritmética.
- Modelos de crecimiento lineal y exponencial.

Grover y Basu, (2017) evidencian que los entornos visuales como Scratch fortalecen la comprensión de la estructura algorítmica y las relaciones matemáticas, ya que los programas hacen explícito el comportamiento de las variables y los procesos lógicos.

2.10.2 Code.org

Code.org es una plataforma internacional sin fines de lucro creada para ampliar el acceso a la educación en Ciencias de la Computación. Ha sido adoptada oficialmente por el Ministerio de Educación de Chile dentro del Plan Nacional de Lenguajes Digitales (MINEDUC, 2019b).

Fundada por Hadi y Ali Partovi en 2013, Code.org ofrece cursos progresivos desde Kínder a II ciclo básico, combinando actividades desenchufadas, programación por bloques y desafíos gamificados que desarrollan pensamiento lógico, secuencias, bucles, condicionales, patrones y depuración (Partovi & Partovi, 2013).

La plataforma se fundamenta en tres principios pedagógicos:

1. Accesibilidad universal: cursos gratuitos, disponibles en más de 60 idiomas.
2. Aprender haciendo: el estudiante construye, prueba, depura y reflexiona a través de proyectos digitales.
3. Progresión cognitiva: actividades escalonadas según complejo conceptual.

En Matemática, Code.org permite diseñar proyectos como:

- Laberintos basados en coordenadas,
- Modelación de patrones numéricos,
- Representaciones de funciones mediante animaciones,
- Simulación de problemas geométricos simples.

El MINEDUC (2021b) destaca que la plataforma permite trabajar descomposición, patrones, algoritmos y depuración de forma progresiva y articulada con los Objetivos de Aprendizaje de Matemática.

2.10.3 Google Colab

Google Colab es un entorno de Google basado en notebooks en la nube que permite escribir y ejecutar Python sin instalación local. Su uso ha aumentado en la educación media y superior porque facilita la modelación computacional y el análisis matemático de forma inmediata y colaborativa.

Colab habilita:

- Simulaciones matemáticas y geométricas,
- Representación gráfica de funciones mediante librerías como *Matplotlib*,
- Análisis estadístico con *Pandas* y *NumPy*,
- Modelación numérica avanzada,
- Prototipos de machine learning, pertinentes al área de datos en Matemática aplicada.

Shute et al. (2017) destacan que trabajar pensamiento computacional mediante lenguaje textual promueve prácticas de pensamiento computacional como abstracción, modelación y depuración iterativa. Además, Weintrop et al. (2016) señalan que Python potencia el razonamiento matemático al permitir explorar comportamientos bajo distintas condiciones de forma rápida y visual, especialmente en entornos notebook.

Google Colab funciona como un “laboratorio matemático computacional” accesible, lo que lo convierte en una herramienta potente para enseñanza media y para el Electivo de Pensamiento Computacional y Programación (MINEDUC, 2021b).

2.11 Currículum Nacional en Matemática, Resolución de Problemas y Pensamiento Computacional

El currículum nacional chileno concibe la educación matemática como un espacio para desarrollar competencias cognitivas de alto nivel, especialmente representación, modelación, argumentación y resolución de problemas. Las Bases Curriculares (MINEDUC, 2015; 2019c) y el proceso de Actualización Curricular 2024 (MINEDUC, 2024a) establecen que "hacer matemática" implica comprender fenómenos, analizar relaciones, construir modelos y comunicar soluciones mediante representaciones diversas, dentro de marcos socioculturales significativos. Esta visión coincide de manera natural con los componentes del Pensamiento Computacional, que exige descomponer problemas, abstraer información, identificar patrones, diseñar algoritmos y evaluar soluciones.

2.11.1 Objetivos de Aprendizaje en Matemática y el Pensamiento Computacional

Los Objetivos de Aprendizaje (OA) se definen como el conjunto articulado de conocimientos (OAC), habilidades (OAH) y actitudes (OAA) que favorecen el desarrollo personal, social y cognitivo del estudiantado (MINEDUC, 2015; 2019c). En Matemática, los OAC corresponden a conceptos, procedimientos, representaciones y modelos organizados en los ejes formativos de:

- Números,
- Álgebra y Funciones,
- Geometría,
- Probabilidades y Estadística.

Las habilidades (OAH) en Matemática (resolver problemas, representar, modelar, argumentar y comunicar) se desarrollan progresivamente según el nivel escolar.

Según las *Bases Curriculares de Matemática* (MINEDUC, 2015; 2019c), la habilidad de Resolver Problemas se desarrolla de manera progresiva a lo largo de toda la escolaridad, articulando estrategias cada vez más complejas y alineadas con las demandas cognitivas del nivel. En 7.º y 8.º Básico, los OAH se centran en destacar información relevante, usar ensayo y error sistemático, aplicar procesos reversibles y descartar información irrelevante (MINEDUC, 2015). En 1.º y 2.º Medio, las Bases formalizan estrategias que evidencian un mayor nivel de abstracción, tales como descomponer problemas en partes más simples, buscar patrones y utilizar herramientas computacionales como apoyo al razonamiento matemático (MINEDUC, 2015). Finalmente, en 3.º y 4.º Medio, los OA enfatizan la resolución de problemas no rutinarios, la construcción colaborativa de estrategias y la modificación de parámetros en modelos para analizar cómo afectan los resultados, evidenciando una actividad matemática altamente compatible con los componentes del pensamiento computacional (MINEDUC, 2019c).

2.11.2 Habilidades de Matemáticas

En las Bases Curriculares del MINEDUC (2015), la enseñanza de la Matemática se organiza en torno al desarrollo de cuatro habilidades fundamentales:

- resolver problemas,
- representar,
- modelar y
- argumentar y comunicar.

Estas habilidades se conciben de manera interrelacionada, cumpliendo un rol esencial en la construcción de nuevos conceptos y destrezas matemáticas, así como en la aplicación de los conocimientos en contextos diversos.

En coherencia con este marco curricular, se considerarán estas cuatro habilidades matemáticas como ejes centrales de análisis y desarrollo,

asumiéndolas como referentes fundamentales para el diseño, implementación y evaluación de la propuesta didáctica, así como para la comprensión de los procesos de aprendizaje involucrados.

De igual manera, al considerar la enseñanza de la Matemática, la habilidad de resolver problemas ocupa un lugar central, ya que integra y moviliza de manera articulada las demás habilidades matemáticas, tales como representar, modelar y argumentar y comunicar. En efecto, las Bases Curriculares establecen que estas habilidades no deben desarrollarse de forma aislada, sino que se ponen en juego de manera conjunta al enfrentar situaciones problemáticas, permitiendo a los estudiantes construir significados, seleccionar representaciones pertinentes, modelar relaciones y comunicar sus razonamientos y conclusiones (MINEDUC, 2019c). Desde esta perspectiva, la resolución de problemas se constituye tanto en un objetivo de aprendizaje como en un medio fundamental para el desarrollo del pensamiento matemático y la aplicación del conocimiento en contextos diversos.

Desde esta perspectiva, la resolución de problemas se configura como un puente natural entre el Pensamiento Computacional y la Matemática, dado que ambos comparten procesos cognitivos fundamentales como la descomposición de situaciones complejas, la identificación de patrones, la abstracción y la formulación de estrategias para alcanzar una solución. Diversos autores destacan que el Pensamiento Computacional no se limita al uso de tecnologías, sino que corresponde a una forma de abordar problemas de manera lógica, sistemática y reflexiva, aspectos que son inherentes a la actividad matemática escolar (Wing, 2006; 2011; García-Peñalvo et al, 2017; Shute, et al. 2017). Al situar la resolución de problemas como eje articulador, se favorece así una integración auténtica del Pensamiento Computacional en la enseñanza de la Matemática, fortaleciendo la capacidad de los estudiantes para aplicar sus conocimientos de manera flexible y transferible en contextos reales.

2.11.3 Ciudadanía Digital, Matemática y Pensamiento Computacional

La Propuesta de Actualización Curricular 2024 (MINEDUC, 2024a) incorpora la ciudadanía digital como eje transversal de todas las asignaturas. Se establecen cuatro dimensiones:

1. Alfabetización digital crítica y reflexiva,
2. Cuidado y responsabilidad digital,
3. Participación ciudadana digital,
4. Creatividad digital e innovación.

Donde el MINEDUC (2024a) explicita así:

> En Matemática se consideran las dimensiones de Alfabetización digital crítica y reflexiva junto a la de Creatividad digital e innovación, dado que abarcan el desarrollo de conocimientos y despliegue de habilidades y actitudes para el uso de forma autónoma, crítica y creativa de tecnologías digitales en la construcción de conocimientos matemáticos de forma transversal, en todos los niveles de la asignatura, usando herramientas digitales para resolver problemas, modelar, representar, argumentar y comunicar y desarrollar el Pensamiento Computacional (p.56).

Esta formulación legitima pedagógicamente la presencia de algoritmos, simulaciones, programación y análisis de datos dentro de la asignatura.

La ciudadanía digital, en este marco, no es un contenido adicional, sino una condición epistemológica para aprender Matemática en una sociedad intensiva en información: analizar datos, construir modelos, evaluar resultados y participar en entornos digitales. Esto refuerza la importancia del pensamiento

computacional como competencia transversal, alineada con marcos internacionales como UNESCO (2018) y resultados de Maggio (2018) donde se plantea que los sistemas educativos del siglo XXI deben ser holísticos, críticos y capaces de responder al desafío tecnológico global.

2.11.4 Electivo de Profundización de Pensamiento Computacional

El Decreto 193 (MINEDUC, 2019a) establece el Electivo "Pensamiento Computacional y Programación", cuyo propósito es desarrollar competencias para analizar problemas, diseñar algoritmos, programar soluciones y evaluar sus resultados en contextos matemáticos y disciplinares diversos.

Este electivo reconoce explícitamente la convergencia entre Matemática y Pensamiento Computacional, planteando que la computación favorece el pensamiento lógico, el análisis estructurado, la abstracción, la representación algorítmica, la verificación y la modelación. La Unidad 1 del programa propone una transición desde el lenguaje natural hacia el lenguaje computacional, mediante pseudocódigo y diagramas de flujo, coherente con trayectorias investigadas por Rich et al. (2017; 2018; 2019; 2020). Estos lineamientos, aun cuando se orientan a la educación media superior, ofrecen criterios pedagógicos para introducir pensamiento computacional en cursos inferiores como 2.º Medio, fortaleciendo la coherencia vertical de la progresión curricular.

2.11.5 Articulación vigente y proyección 2024

El currículum vigente ya operacionaliza la integración Matemática – Pensamiento Computacional a través del Programa "Pensamiento Computacional y Programación" (MINEDUC, 2021b), que demanda:

- Programar algoritmos que modelen procedimientos matemáticos.
- Traducir algoritmos a pseudocódigo y diagramas de flujo.
- Elaborar representaciones digitales y manuales equivalentes.

- Validar modelos mediante trazas y casos de prueba.

La Propuesta de Actualización Curricular 2024 (MINEDUC, 2024a) generaliza esta articulación a toda la escolaridad, estableciendo que el uso de tecnologías digitales en Matemática debe desarrollar Pensamiento Computacional, construir modelos, explorar patrones, argumentar resultados y comunicar mediante diversos lenguajes.

En este marco, los ejes de Matemática (Números; Patrones, Álgebra y Funciones; Geometría; Tratamiento de Datos) proveen puntos de anclaje naturales para diseñar rutas didácticas desde lo desenchufado hacia lo enchufado, coherentes con las trayectorias de Rich y con las progresiones del MINEDUC (2021a).

2.11.6 Competencias Digitales y Estándares Docentes en Educación Digital

El desarrollo de competencias digitales constituye hoy un eje central para comprender e implementar prácticas educativas alineadas con las demandas del siglo XXI. A nivel internacional, la UNESCO (2018) plantea un marco amplio que concibe la competencia digital docente como la articulación entre conocimiento disciplinar, pedagógico y tecnológico orientado a un uso ético, crítico, creativo y significativo de las tecnologías. Este enfoque dialoga con modelos consolidados como ISTE Standards for Educators (ISTE, 2017) y DigCompEdu[7] (Redecker & Punie, 2017), los cuales enfatizan que el rol docente no se limita al dominio técnico de herramientas, sino que implica el diseño de experiencias auténticas de aprendizaje, la promoción de la ciudadanía digital, la toma de decisiones basada en datos, la integración de lenguajes de programación y la incorporación progresiva del Pensamiento Computacional (Pensamiento Computacional) como competencia clave para el desarrollo cognitivo contemporáneo.

[7] DigCompEdu, por sus siglas en inglés: Digital Competence Framework for Educators.

La literatura internacional coincide en que la integración de tecnologías es un proceso evolutivo y progresivo. Newhouse et al. (2013), junto con Tong y Trinidad (2005), proponen un modelo que avanza desde fases iniciales de inacción e investigación hacia un uso pedagógico que logra niveles de integración y, eventualmente, transformación. Desde esta perspectiva, la apropiación tecnológica no ocurre de forma inmediata, sino que se construye a través del aumento del dominio técnico, la pertinencia pedagógica del uso, la capacidad reflexiva y el impacto efectivo sobre los aprendizajes. Esta mirada es coherente con la visión de alfabetización digital de la UNESCO (2023), que entiende la apropiación tecnológica como un proceso cultural y formativo, no meramente instrumental.

En el contexto chileno, estos enfoques convergen en el Marco de Competencias Digitales Docentes (MINEDUC, 2025), que organiza la progresión profesional en cuatro niveles: iniciación, aplicación, integración y transformación. Esta progresión se sustenta en cuatro criterios: conocimiento tecnológico, calidad pedagógica del uso, complejidad cognitiva de las tareas y juicio profesional reflexivo. En iniciación, el docente reconoce y utiliza herramientas con fines funcionales; en aplicación, comienza a incorporarlas en actividades pedagógicas con apoyo técnico; en integración, toma decisiones intencionadas orientadas a objetivos de aprendizaje; y en transformación, emplea la tecnología con un manejo profesional avanzado para innovar pedagógicamente y mejorar los resultados mediante decisiones fundamentadas.

Este marco se alinea con el Marco para la Buena Enseñanza (CPEIP, 2021), que reconoce que la tecnología es un componente inseparable de las prácticas pedagógicas contemporáneas. Desde esta mirada, la competencia digital implica comprender cuándo, cómo y para qué utilizar herramientas digitales según las necesidades formativas de los estudiantes. La ciudadanía digital se convierte,

así, en un criterio estructural para el diseño, implementación y evaluación de experiencias de aprendizaje.

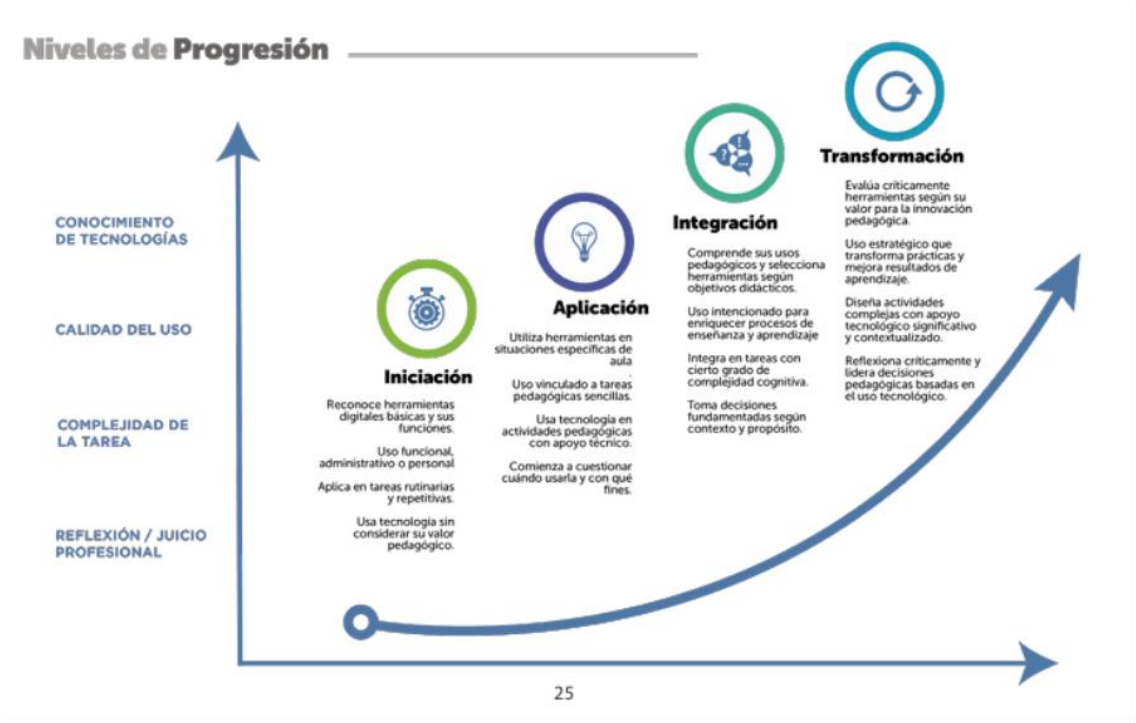


Ilustración 10. Niveles de progresión de la competencia digital docente

Fuente: Tomado de "Marco Orientador de Competencias Digitales Docentes" (p.25), por MINEDUC, (2025), Centro de Innovación.

2.12 Formación de Profesores y Pensamiento Computacional

La formación docente constituye un elemento decisivo para la incorporación efectiva del Pensamiento Computacional en el aula. La literatura especializada coincide en que la capacidad de integrar tecnologías digitales, diseñar actividades orientadas a la resolución de problemas y promover habilidades de pensamiento de orden superior depende profundamente del desarrollo profesional del profesorado (Voogt et al., 2015). En este sentido, enseñar

Pensamiento Computacional no se reduce a capacitar docentes en lenguajes de programación: implica comprender cómo se articulan procesos como la descomposición, la abstracción, la identificación de patrones, el diseño algorítmico y la depuración dentro de contextos disciplinares reales (Yadav et al., 2017).

A nivel internacional, marcos como el de la UNESCO (2018) plantean que la competencia digital docente exige el uso ético, crítico y creativo de tecnologías para diseñar experiencias de aprendizaje auténticas e inclusivas. Este enfoque enfatiza la capacidad del profesorado para seleccionar herramientas adecuadas, evaluar información digital, promover ciudadanía digital y diseñar ambientes de aprendizaje mediados por tecnologías, elementos estrechamente vinculados al desarrollo del pensamiento computacional en el aula.

En Chile, estas visiones convergen con el Marco para la Buena Enseñanza (MBE), el cual indica que el docente contemporáneo debe integrar tecnologías de manera pedagógica, favoreciendo procesos de resolución de problemas, argumentación, modelación y representación matemática en ambientes digitalmente enriquecidos (CPEIP, 2021). A la vez, la noción de ciudadanía digital promovida por el MINEDUC define que los estudiantes deben aprender a participar críticamente en entornos tecnológicos, y que el docente debe ser capaz de guiar el trabajo con programación, análisis de datos y pensamiento algorítmico de manera segura y consciente (MINEDUC, 2019c).

La Propuesta de Actualización Curricular 2024 refuerza esta visión declarando explícitamente que Matemática debe integrarse con tecnologías digitales y contribuir al desarrollo del Pensamiento Computacional, la alfabetización digital crítica y la creatividad digital (MINEDUC, 2024). Esta articulación transforma el rol docente: para que el pensamiento computacional se integre de forma efectiva y coherente, el profesorado debe contar con herramientas conceptuales y

didácticas que les permitan planificar secuencias progresivas, evaluar procedimientos algorítmicos y facilitar la modelación matemática con tecnologías.

Desde una perspectiva evaluativa y didáctica, el Programa de Estudio pensamiento computacional y Programación (MINEDUC, 2021b) ofrece lineamientos concretos: propone actividades con pseudocódigo, diagramas de flujo, lenguajes por bloques y lenguajes textuales que permiten observar evidencias de pensamiento algorítmico y modelación. Sus objetivos formativos establecen que el pensamiento computacional posibilita analizar críticamente problemas, crear y evaluar modelos computacionales, y comprender la articulación entre sistemas, tecnologías y razonamiento matemático.

En este marco, la preparación docente se convierte en condición habilitante para implementar módulos didácticos que integren Matemática y Pensamiento Computacional. La formación continua es indispensable: como señala UNESCO (2023), los sistemas educativos deben ofrecer programas flexibles y contextualizados que permitan a los docentes comprender la lógica interna del pensamiento computacional y su valor disciplinar. Esta necesidad se alinea directamente con el propósito del presente módulo didáctico, orientado a facilitar que docentes en ejercicio integren progresiones desde lo desenchufado hacia la programación textual en Matemática de manera coherente, gradual y fundamentada.

Capítulos 3: Metodología

La presente investigación se desarrolla bajo un paradigma cualitativo, adoptando el enfoque metodológico de la Investigación de Diseño. Se ha seleccionado esta perspectiva dado que el propósito del estudio no se limita a describir una realidad educativa existente, sino que busca generar una intervención didáctica específica, un módulo articulado de pensamiento computacional y Matemática, y comprender los principios teóricos que fundamentan su construcción y funcionamiento.

Para definir el alcance de este trabajo, nos basamos en los planteamientos de Molina et al. (2011), quienes sostienen que el objetivo central de la investigación de diseño es examinar el aprendizaje dentro de su propio contexto. Según estos autores, dicha meta se logra mediante la creación y el análisis sistemático de formas concretas de aprendizaje, estrategias y herramientas educativas, atendiendo siempre a la complejidad sistémica que vincula los procesos de aprender, enseñar y evaluar.

Bajo esta óptica, y considerando el carácter proyectivo de la tesis, el proceso investigativo incorpora mecanismos de aseguramiento de la calidad del diseño previos a su implementación (Molina et al, 2011). Por ello, se integra un indicador observable del diseño de aprendizaje, sustentadas en el Marco de Tareas Matemáticas de Stein y Smith, (1998) mediante el instrumento *Instructional Quality Assessment* de Boston (2012).

De acuerdo con la estructura metodológica propia de la investigación de diseño, la cual exige una preparación fundamentada (Cobb et al., 2003; Molina et al., 2011) y un diseño instruccional iterativo (Cobb et al., 2003), el estudio se organizó en las siguientes etapas operativas:

3.1 Fase 1: Preparación y Análisis Epistemológico y Curricular

Esta fase inicial se centró en la recopilación y sistematización crítica de los fundamentos que sustentan la integración del pensamiento computacional en el sistema escolar. El propósito fue construir un marco referencial robusto que no solo describiera conceptos aislados, sino que estableciera las bases teóricas para la propuesta didáctica posterior. Para ello, el análisis se estructuró en tres dimensiones interrelacionadas:

- Dimensión Curricular y Normativa: Se contrastó la normativa vigente, específicamente las Bases Curriculares de Matemática y los documentos orientadores del MINEDUC (2012; 2015; 2019c; 2021b; 2022; 2024a; 2024b; 2025), con la literatura especializada, identificando los espacios curriculares en educación básica y media donde convergen las Habilidad de resolver problemas y los Objetivos de Aprendizaje (OA) disciplinares.
- Dimensión Académica y Epistemológica: Se analizaron estudios internacionales y nacionales para correlacionar teóricamente tres ejes fundamentales: el pensamiento computacional, la resolución de problemas y la educación matemática. Este análisis permitió identificar los procesos cognitivos compartidos (como la abstracción y la descomposición) que justifican la "conexión natural" entre estas áreas.
- Dimensión Pedagógica: Se revisaron modelos didácticos probados como las Trayectorias de Aprendizaje para fundamentar la progresión de la enseñanza, asegurando que los "cimientos teóricos" de la propuesta fueran coherentes con el desarrollo cognitivo de los estudiantes en los niveles seleccionados.

3.2 Fase 2: Diseño y Construcción de la Propuesta Didáctica

Esta fase constituyó la etapa de ingeniería didáctica del estudio. Su propósito fue construir una secuencia de aprendizaje coherente que no partiera de cero, sino que capitalizara prácticas probadas para luego expandirlas hacia niveles de mayor complejidad tecnológica. El procedimiento se sistematizó en cuatro etapas operativas:

1. Búsqueda y Selección de Referentes Didácticos. Se inició con una revisión exhaustiva de repositorios especializados en educación en ciencias de la computación como *CS Unplugged* y *Code.org*, para identificar actividades base que cumplieran dos criterios fundamentales:
   - Baja carga cognitiva extrínseca. Se priorizaron actividades "desenchufadas" o *unplugged* que permitieran aislar los conceptos lógicos (como algoritmos o bucles) sin la distracción inmediata de la sintaxis de programación.
   - Potencial de vinculación matemática: Se seleccionaron aquellas dinámicas que, aunque originalmente diseñadas para informática, presentaran una estructura subyacente compatible con los Objetivos de Aprendizaje (OA) de Matemática chilenos, tales como la ubicación espacial, los patrones y la probabilidad.
2. Adaptación y Contextualización Matemática. Las actividades seleccionadas fueron sometidas a un proceso de adaptación didáctica para transformarlas de simples "juegos lógicos" a módulos de aprendizaje matemático.
3. Creación de la Progresión "Enchufada" (Extensión del Diseño). Dado que la literatura y los recursos existentes suelen enfocarse mayoritariamente en la etapa introductoria (desenchufada), fue necesario crear y diseñar los

componentes de continuidad hacia la programación digital para cumplir con la trayectoria completa.

4. Diseño de Puentes Tecnológicos: Se crearon extensiones específicas para cada módulo que permiten transferir la experiencia física al entorno digital. Todas las actividades (adaptadas y creadas) se articularon en una ruta progresiva que respeta los niveles de competencia de Rich et al.:
   - Nivel Inicial: Precisión y Secuencia.
   - Nivel Intermedio: Repetición y Patrones.
   - Nivel Avanzado: Condiciones, Variables y Datos.

3.3 Fase 3: Indicadores observables del Diseño de Aprendizaje

Para garantizar que la propuesta didáctica supere la mera transmisión de información y asegure el rigor disciplinar, se utilizó el Instrumento de Análisis de Demanda Cognitiva. Este mecanismo de validación se fundamenta teóricamente en el Marco de Tareas Matemáticas (*Mathematical Tasks Framework*) de Stein y Smith (1998), quienes postulan que las tareas instruccionales configuran las oportunidades de aprendizaje de los estudiantes al determinar el tipo de procesamiento cognitivo requerido para su resolución.

Según este marco, las actividades no son neutras, sino que se clasifican en cuatro niveles de demanda cognitiva agrupados en dos categorías:

- Demandas de Bajo Nivel: Comprenden la Memorización (reproducción de datos sin comprensión) y los Procedimientos sin Conexiones (uso algorítmico o mecánico de fórmulas sin acceso al significado).
- Demandas de Alto Nivel: Abarcan los Procedimientos con Conexiones (uso de algoritmos para profundizar conceptos) y Hacer Matemáticas

(*Doing Mathematics*), nivel superior que exige pensamiento complejo no algorítmico, exploración de relaciones y formulación de conjeturas.

Para operacionalizar este marco teórico en una herramienta de evaluación concreta, se adaptó la Rúbrica de Potencial de la Tarea (*Rubric 1: Potential of the Task*) del instrumento Instructional Quality Assessment (IQA) desarrollado por Boston (2012). Este instrumento permite clasificar las actividades diseñadas en una escala ordinal de 0 a 4, considerando los niveles señalados por Stein y Smith (1998), donde los puntajes más altos (3 y 4) aseguran que la tarea tiene el potencial de involucrar a los estudiantes en procesos de razonamiento matemático profundo y comprensión conceptual, evitando que el uso de la programación se reduzca a una codificación rutinaria.

Tabla 2 - Instrumento Instructional Quality Assessment

Tabla A1. Rúbrica 1: Potencial de la tarea	
Pregunta Guía: ¿Tuvo la tarea el potencial de involucrar a los estudiantes en un pensamiento riguroso sobre el contenido desafiante	
Puntaje/ Nivel: 4 / Alta Demanda: Hacer Matemáticas	Se Manifiesta a través de: • Hacer matemáticas: Uso de pensamiento complejo y no algorítmico (es decir, no existe un enfoque o camino predecible y bien ensayado sugerido explícitamente por la tarea, las instrucciones o un ejemplo resuelto); O BIEN • Procedimientos con conexiones: Aplicación de un procedimiento general

	amplio que permanece estrechamente conectado a conceptos matemáticos. Requisito clave: La tarea debe solicitar explícitamente evidencias del razonamiento y comprensión de los estudiantes. *Ejemplos:* Resolver un problema genuino y desafiante donde el razonamiento es evidente; desarrollar una explicación de por qué funcionan las fórmulas; identificar patrones y justificar generalizaciones; hacer conjeturas y apoyar conclusiones con evidencia matemática; hacer conexiones explícitas entre representaciones.
3 / Alta Demanda pero con limitaciones / Procedimiento con conexiones	La tarea tiene el potencial de involucrar a los estudiantes en pensamiento complejo o en la creación de significado para conceptos matemáticos. Sin embargo, la tarea no amerita un "4" porque: • No solicita explícitamente evidencia del razonamiento o comprensión del estudiante.

	• Se pide a los estudiantes involucrarse en "hacer matemáticas" o "procedimientos con conexiones", pero la matemática subyacente no es apropiada para el grupo específico (muy fácil o muy difícil). • Se pide identificar patrones, pero no se presiona para formar o justificar generalizaciones. • Se pide usar múltiples estrategias, pero no se exige desarrollar conexiones entre ellas. • Se pide hacer conjeturas, pero no proporcionar evidencia o explicaciones para apoyarlas.
2/ Baja Demanda: Procedimientos sin conexiones	El potencial de la tarea se limita a involucrar a los estudiantes en el uso de un procedimiento que es solicitado específicamente o cuyo uso es evidente (basado en instrucción previa o la ubicación de la tarea). • Existe poca ambigüedad sobre qué se necesita hacer y cómo hacerlo. • La tarea no requiere que los estudiantes hagan conexiones con los conceptos o el

	significado que subyace al procedimiento utilizado. • El enfoque parece estar en producir respuestas correctas más que en desarrollar comprensión matemática (ej. aplicar una estrategia específica de resolución de problemas, practicar un algoritmo computacional). • *O bien:* Hay evidencia de que el contenido matemático está al menos 2 niveles de grado por debajo del grado de los estudiantes.
1 / Baja Demanda: Memorización	La tarea no requiere que los estudiantes hagan conexiones con los conceptos o significados que subyacen a los hechos, reglas, fórmulas o definiciones que están siendo memorizados o reproducidos.
0	La tarea no requiere actividad matemática

Fuente: Adaptado de "Assessing instructional quality in mathematics" (p. 99), por M. D. Boston, 2012, *The Elementary School Journal*, *113*(1).

Capítulo 4: Resultados

En el siguiente capítulo se presentan los principales hallazgos de la investigación, organizados en tres puntos: (1) la definición operacional de las habilidades del Pensamiento Computacional utilizadas en el estudio, (2) la correspondencia funcional establecida entre dichas habilidades y las habilidades matemáticas de las Bases Curriculares, y (3) los resultados del diseño de los módulos didácticos fundamentados en esta articulación. Esta estructura permite transitar desde los conceptos esenciales hacia su integración pedagógica y curricular, ofreciendo el sustento que orienta la propuesta final.

4.1. Recopilación: Definiciones Operacionalizadas

A partir de la síntesis de los planteamientos de Wing (2006, 2011), Aho (2012), Selby y Woollard (2013) y García-Peñalvo y Mendes (2018), se establece la siguiente definición operativa para esta investigación:

El pensamiento computacional se entiende como un proceso de resolución de problemas que moviliza las habilidades de descomposición, reconocimiento de patrones, abstracción, algoritmos y evaluación. Esta competencia se desarrolla a través de un ciclo que inicia con la ejecución humana de soluciones en entornos desconectados y culmina con su automatización mediante agentes tecnológicos, facilitando así la modelación y comprensión profunda de objetos matemáticos.

a) Descomposición

La descomposición se operacionaliza como la capacidad de dividir un problema complejo en partes más simples y manejables, facilitando así su análisis y solución incremental. Esta habilidad, descrita por Wing (2011) y Grover y Pea (2013), permite reducir la carga cognitiva al identificar subproblemas y estructuras jerárquicas.

b) Reconocimiento de Patrones y Generalización

Se definen como procesos cognitivos interdependientes que permiten observar datos para identificar regularidades o tendencias (patrones) y, posteriormente, abstraer esas reglas para aplicarlas a nuevas situaciones o clases de problemas (generalización). De acuerdo con Csizmadia et al. (2015) y Shute et al. (2017), estas habilidades son fundamentales para la eficiencia computacional, pues permiten construir modelos transferibles y reutilizar estrategias en lugar de resolver cada problema desde cero.

En el nivel operativo de la propuesta didáctica, el reconocimiento de patrones constituye el prerrequisito lógico para la optimización algorítmica, bajo la premisa de que para automatizar una tarea mediante un bucle (*loop*), es necesario primero identificar la estructura que se repite. En el dominio matemático, esto se traduce en la capacidad de identificar regularidades en series numéricas, estructuras algebraicas o transformaciones geométricas, facilitando el tránsito desde el análisis de casos particulares hacia la formulación de conjeturas generales.

c) Abstracción

Se conceptualiza como el proceso cognitivo de filtrar y seleccionar información relevante, suprimiendo los detalles superfluos para construir una representación simplificada del problema (Wing, 2006). Esta habilidad resulta fundamental para gestionar la complejidad sin perder el significado esencial, permitiendo la generación de modelos mentales o simbólicos que capturan la estructura subyacente de una situación (Csizmadia et al., 2015).

En el contexto del diseño didáctico, la abstracción es crítica para la modelación de fenómenos matemáticos, tales como funciones o propiedades geométricas, ya que habilita al estudiante para construir representaciones computacionales

que operan sobre conceptos generalizables en lugar de limitarse a la resolución de casos aislados (Weintrop et al., 2016).

d) Pensamiento Algorítmico

Se operacionaliza como la capacidad de diseñar, describir y ejecutar secuencias finitas, ordenadas y precisas de instrucciones no ambiguas para resolver un problema, tal como lo describen Brennan y Resnick (2012) al identificar los conceptos computacionales fundamentales. Esta habilidad se materializa mediante estructuras de control lógico que garantizan la reproducibilidad y eficiencia de la solución propuesta.

Para efectos del diseño del módulo didáctico, el pensamiento algorítmico se trabaja articuladamente a través de tres conceptos clave:

- Algoritmo: Se define como la secuencia lógica de pasos planificada para alcanzar un objetivo o resolver una tarea específica.
- Eventos: Se conceptualiza como la comprensión de que el flujo de un programa (o acción) es controlado por "disparadores" o acciones externas, un principio esencial en la programación interactiva.
- Bucles (*Loops*): Corresponden a estructuras de control que permiten automatizar tareas repetitivas, optimizando el código para hacerlo más eficiente y legible una vez que se ha identificado un patrón subyacente.

e) Evaluación y Depuración

Esta dimensión se define como el proceso sistemático y metacognitivo orientado a verificar la corrección, eficiencia y robustez de una solución propuesta. Integra la habilidad crítica de Depuración (Debugging), entendida como el ciclo iterativo de detección y corrección de errores (*bugs*) en un algoritmo, el cual implica identificar la falla, analizar su causa raíz, proponer una corrección y volver a probar (*re-testear*) la secuencia.

De acuerdo con Shute et al. (2017) y Bordignon e Iglesias (2019), este enfoque pedagógico es clave porque normaliza y resignifica el error, considerándolo no como un fracaso, sino como una parte fundamental y positiva del proceso de aprendizaje y mejora continua. En el contexto matemático, esta competencia se manifiesta en prácticas como la comprobación de resultados, el análisis de casos límite y la argumentación justificada de los procedimientos empleados.

4.2. Relación entre habilidades del Pensamiento Computacional y las habilidades de Matemáticas de las Bases Curriculares.

Según la exhaustiva revisión de literatura realizada, la correspondencia entre las habilidades de Matemática definidas por el MINEDUC (2012; 2015; 2019c; 2024a) y las habilidades del pensamiento computacional no es solo terminológica, sino también funcional y sustentada metodológicamente. De esta forma, desde la perspectiva de la psicometría contemporánea, Larsen y Bong (2016) enuncian que dos habilidades pueden considerarse correspondientes cuando existe *construct correspondence*; esto es, cuando, aunque provengan de tradiciones distintas, describen procesos cognitivos equivalentes, utilizan terminología parcialmente convergente y muestran patrones de relación similares con otras habilidades del sistema cognitivo.

Bajo este marco, afirmar que las habilidades de Matemática y las del pensamiento computacional se corresponden no implica forzar paralelos arbitrarios, sino identificar que ambas movilizan estructuras de razonamiento compartidas, tales como la descomposición, la identificación de patrones, la abstracción, el diseño estratégico y la evaluación de soluciones. Esta convergencia ha sido ampliamente documentada en la literatura educativa (Csizmadia et al., 2015; Sáez-López et al., 2023; Weintrop et al., 2016),

ofreciendo evidencia de que ambas áreas activan los mismos procesos cognitivos subyacentes, incluso cuando se expresan con lenguajes disciplinares diferentes.

Por ello, la relación entre habilidades matemáticas y de pensamiento computacional propuesta en esta tesis responde a una correspondencia funcional, entendida como el alineamiento entre el rol cognitivo que ambas desempeñan en la resolución de tareas auténticas. Este enfoque se articula con las orientaciones curriculares nacionales, que desde las Bases Curriculares y la actualización 2024 reconocen explícitamente la necesidad de integrar razonamiento matemático, tecnológico y computacional en un mismo marco competencial (MINEDUC, 2019c, 2024). En paralelo, la evidencia internacional, como el marco de PISA 2022, refuerza que habilidades antes consideradas propias de la computación, por ejemplo, definir algoritmos o depurar procedimientos, son ahora parte constitutiva del desempeño matemático avanzado (OCDE, 2023).

a) Descomponer en relación con Resolver problemas

Descomponer consiste en dividir un problema complejo en subproblemas manejables, identificando metas intermedias, restricciones y decisiones clave. En el ámbito matemático, esta operación se corresponde funcionalmente con la habilidad de resolver problemas, tal como se formula en las Bases Curriculares: analizar la situación, seleccionar estrategias, ejecutar procedimientos y revisar la solución (MINEDUC, 2015, 2021b). En términos cognitivos, descomponer apoya directamente los componentes de razonamiento fluido (Gf) y memoria de trabajo (Gsm) implicados en la resolución de problemas no rutinarios (Geary, 2011; Roman-González et al., 2017).

Cuando el estudiante descompone un problema de probabilidad, de funciones o de geometría, realiza exactamente el mismo tipo de trabajo mental que se demanda en la descomposición computacional: aislar casos, distinguir etapas,

reconocer qué información es necesaria y qué operaciones deben realizarse en cada paso (Csizmadia et al., 2015). Por ello, la vinculación de Descomponer con Resolver problemas es estructural: a mayor dominio de la descomposición, mayor capacidad de enfrentar problemas matemáticos complejos con estrategias planificadas y no impulsivas (Jonassen, 2000; OCDE, 2014).

b) Generalizar en relación con Modelar

Generalizar, dentro del pensamiento computacional, supone pasar de ejemplos particulares a reglas o estructuras más amplias: detectar regularidades, formular conjeturas y expresarlas como patrones reutilizables (Rich et al., 2018; Sáez-López et al., 2023). Esta habilidad encuentra su correlato funcional más claro en la habilidad de modelar en Matemática, la cual implica precisamente identificar relaciones invariantes en fenómenos reales o abstractos, y expresarlas mediante ecuaciones, funciones, gráficos o simulaciones (MINEDUC, 2015, 2021b).

La correspondencia entre Generalizar y Modelar se sustenta en que ambos procesos requieren identificar regularidades, abstraer relaciones esenciales y construir estructuras que trascienden casos particulares. En Matemática, modelar implica representar fenómenos mediante ecuaciones, funciones o sistemas simbólicos que capturan relaciones invariantes (MINEDUC, 2015, 2019c; 2021b); en pensamiento computacional, generalizar supone derivar patrones y reglas aplicables a múltiples situaciones, permitiendo construir modelos computacionales que describen o simulan comportamientos de manera sistemática (Weintrop et al., 2016). Desde esta perspectiva, la relación entre ambas habilidades es funcional: sin la capacidad de generalizar, la modelación matemática queda limitada a la descripción puntual; mientras que, cuando existe generalización sólida, la modelación se transforma en la creación de estructuras matemáticas y computacionales transferibles, en línea con las orientaciones internacionales sobre competencias avanzadas en resolución de problemas (OCDE, 2023).

c) Evaluar en relación con Argumentar y comunicar

En pensamiento computacional, evaluar (o evaluar/depurar) implica revisar soluciones, detectar errores, contrastar alternativas y decidir si un algoritmo, modelo o programa satisface los criterios del problema. En Matemática, esto se corresponde funcionalmente con argumentar y comunicar, habilidad que exige justificar procedimientos, validar resultados y explicitar las razones que hacen aceptable una solución (MINEDUC, 2015, 2019c).

Rich et al. (2019) muestran que la depuración computacional constituye un proceso sistemático de refinamiento progresivo que implica comparar resultados esperados y obtenidos, identificar inconsistencias, revisar supuestos y ajustar la solución hasta lograr coherencia interna. Estos pasos reflejan estrechamente las demandas de la argumentación matemática, donde el estudiante debe justificar procedimientos, verificar la validez de cada etapa y corregir errores cuando los resultados no coinciden con lo esperado. Por ello, Evaluar en pensamiento computacional y Argumentar y Comunicar en Matemática mantienen una correspondencia funcional: la evaluación sistemática de algoritmos fortalece la habilidad de justificar y comunicar razonamientos matemáticos con claridad y fundamento (Sáez-López et al., 2023; Shute et al., 2017).

d) Abstraer en relación con Representar

Abstraer supone seleccionar los elementos relevantes de una situación y omitir los detalles irrelevantes, creando una versión simplificada que mantiene la estructura esencial del problema (Aho, 2012; Wing, 2011). Esta operación cognitiva se alinea de manera directa con la habilidad de representar en Matemática, que implica escoger y coordinar sistemas simbólicos, numéricos, algebraicos, geométricos, gráficos, para expresar relaciones y estructuras.

Zapata-Ros (2019) define la abstracción matemática como detectar elementos clave de un problema ignorando detalles irrelevantes, definición que coincide con

la noción de abstracción computacional. Representar exige precisamente este mismo proceso: decidir qué magnitudes incluir en una gráfica, qué variables introducir en una ecuación o qué diagrama usar para describir una situación. Desde el modelo CHC, ambos procesos se apoyan fuertemente en habilidades de procesamiento visual (Gv) y en la capacidad de manipular configuraciones simbólicas (McGrew, 2009; Roman-González et al., 2017). En consecuencia, Abstraer y Representar es una correspondencia funcional: sin una adecuada abstracción, las representaciones matemáticas se vuelven confusas o irrelevantes; una abstracción sólida permite construir representaciones potentes tanto en papel como en entornos digitales.

e) Pensar de forma algorítmica en relación con Resolver Problemas y Modelar

Pensar de forma algorítmica implica diseñar secuencias ordenadas de pasos, con condiciones y repeticiones, que garanticen la resolución de una tarea o la simulación de un fenómeno (CSTA & ISTE, 2011; Shute et al., 2017). Esta habilidad se vincula funcionalmente con dos habilidades matemáticas del MINEDUC (2015; 2019c): resolver problemas y modelar.

Por una parte, en la resolución de problemas, el pensamiento algorítmico se concreta cuando el estudiante elabora un 'plan de solución' que puede describirse como un algoritmo: determinar un orden de operaciones, establecer condiciones ("si el valor es mayor que… entonces…"), decidir repeticiones (iteraciones) y contemplar verificaciones finales. PISA 2022 incorpora explícitamente la capacidad de definir algoritmos como parte de una solución matemática detallada, reconociendo el carácter algorítmico de la resolución de problemas matemáticos complejos (OCDE, 2023).

Por otra parte, en la modelación, muchas veces el modelo matemático no se agota en la ecuación, sino que se operacionaliza como un procedimiento

explícito: un algoritmo numérico iterativo, una simulación computacional, una hoja de cálculo o un programa que ejecuta el modelo en distintos escenarios (Weintrop et al., 2016). Aquí, pensar algorítmicamente permite transformar el modelo en una herramienta dinámica de exploración, probando parámetros, analizando sensibilidad y observando comportamientos emergentes. Por eso, Pensar de forma algorítmica y Resolver problemas / Modelar describe una correspondencia funcional doble: el pensamiento algorítmico estructura tanto la estrategia de solución como la implementación operativa del modelo.

A continuación se presenta un esquema que resume cómo cada habilidad matemática se vincula con habilidades específicas del pensamiento computacional.

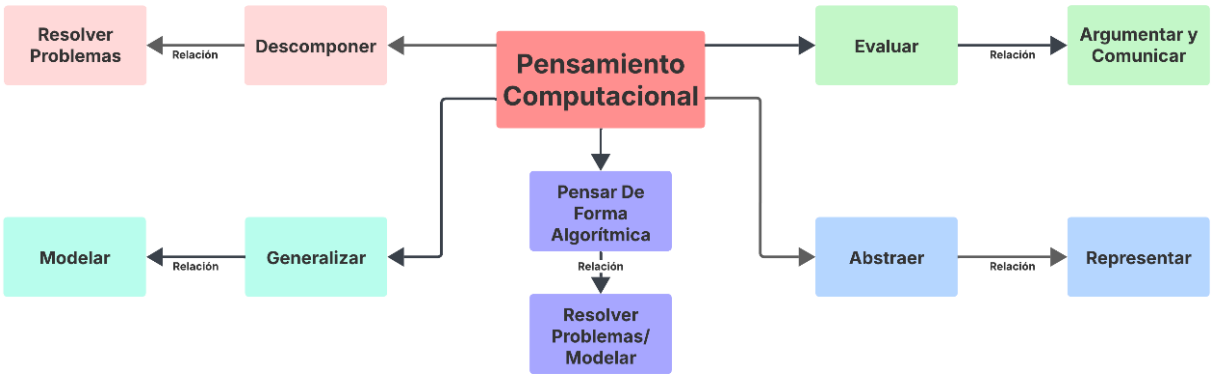


Ilustración 11 - Elaboración Propia

4.3. Módulos Didáctico

El presente capítulo expone en detalle la propuesta didáctica diseñada para integrar la resolución de problemas matemáticos con el desarrollo progresivo del pensamiento computacional, articulando una secuencia de actividades que avanzan desde experiencias desconectadas (unplugged) hacia entornos de programación visual y, en algunos casos, programación textual. La construcción de estos módulos se fundamenta en las trayectorias de aprendizaje propuestas por Rich et al. (2017, 2018, 2019, 2020), quienes describen un desarrollo conceptual que progresa desde la comprensión de secuencias simples, pasando por repeticiones, estructuras condicionales, depuración de errores y finalmente abstracciones más complejas. Esta progresión se alinea con el currículum nacional de Matemática (MINEDUC, 2015, 2019c) y con los objetivos formativos del Electivo de Pensamiento Computacional y Programación (MINEDUC, 2021b, 2022).

A continuación, se presentan cinco módulos didácticos organizados de menor a mayor complejidad cognitiva y computacional: Soy un Robot, Fiesta de Baile, Magia con Números, Detección de Errores por Paridad – Hamming, y Máquina de Galton. Cada módulo incorpora actividades desenchufadas para introducir conceptos fundamentales mediante experiencias corporales, manipulativas o narrativas, seguidas de actividades enchufadas que permiten trasladar dichos conceptos a entornos digitales como Code+, Scratch o Python. Este tránsito asegura que el pensamiento computacional no se reduzca a la programación, sino que opere como un marco de pensamiento transversal que fortalece habilidades matemáticas tales como modelar, representar, argumentar, buscar patrones y resolver problemas no rutinarios.

Los módulos incluyen los siguientes elementos:

a) nombre de la actividad y fuente;

b) objetivos de aprendizaje sugeridos (alineados al currículum nacional);
c) habilidad matemática predominante;
d) habilidad de pensamiento computacional asociada;
e) descripción de las actividades desconectadas y conectadas;
f) materiales y anexos;
g) fundamentos narrativos o contextuales cuando corresponda.

Antes de detallar cada módulo, se presenta una tabla general que resume la secuencia completa, permitiendo visualizar cómo progresa la complejidad del pensamiento computacional y la Matemática a lo largo de la propuesta.

Tabla 3 - Secuencia General de los Módulos Didácticos

Módulo	Concepto central de Pensamiento Computación (según Rich et al.)	Tipo de actividad	Descripción breve del desafío
1. Soy un Robot	Secuencia Condición Depuración	Unplugged Code+	Programar rutas físicas y digitales para un robot, usando comandos secuenciales y lógicas SI–ENTONCES.
2. Fiesta de Baile	Repetición Bucles anidados	Unplugged Code+	Representar secuencias repetitivas y optimizarlas mediante bucles simples y anidados, comparando versiones largas vs. eficientes.
3. Magia con Números	Abstracción Representación de datos	Unplugged Scratch	Descubrir cómo funciona un truco basado en binario y luego programarlo usando variables y estructuras condicionales.
4. Paridad y Código Hamming	Depuración Evaluación	Unplugged Scratch Python	Detectar y corregir errores en mensajes mediante paridad y el algoritmo Hamming (7,4), simulando la comunicación digital real.

5. Máquina de Galton	Aleatoriedad Simulación Modelos probabilísticos	Unplugged Scratch Python	Simular fenómenos aleatorios mediante experimentos físicos y digitales, analizando distribuciones emergentes.

Fuente: Adaptada de Rich et al., (2017; 2018; 2019; 2020)

A. Soy un Robot

Nombre: Soy un Robot
Fuente: CSUnplugged. Kidbots. https://www.csunplugged.org/en/topics/kidbots/ NASA. Un día en la vida de un Mars Rover D. Mars Exploration Program.A Day in the Life of a Mars Rover Driver. https://mars.nasa.gov/mars2020/mission/rover/driver/ Lucidchart. *Símbolos comunes de los diagramas de flujo*. https://www.lucidchart.com/ Code.org. *Curso C (2021): Lección 3. Programación con Angry Birds*. https://studio.code.org/ Code.org. *Curso C (2021): Lección 4. Depuración en el laberinto*. https://studio.code.org/ Code.org. *Curso Express (2021): Lección 15. Condicionales (Si/sino) con Abeja*. https://studio.code.org/ Code.org. *Curso Express (2021): Lección 16. Bucles 'mientras' en Granjera*. https://studio.code.org/
Objetivo de la actividad: Diseñar, implementar y depurar algoritmos secuenciales y con estructuras condicionales para guiar un robot/sprite a través de laberintos físicos y digitales, descomponiendo el problema en subpasos, utilizando diagramas de flujo y programación por bloques, y comprobando colaborativamente la corrección de los procedimientos propios y de sus compañeros.

Objetivo de Aprendizaje sugeridos:	MA 8B OA 13. Describir la posición y el movimiento (traslaciones, rotaciones y reflexiones) de figuras 2D, de manera manual y/o con software educativo, utilizando: • Los vectores para la traslación. • Los ejes del plano cartesiano como ejes de reflexión. • Los puntos del plano para las rotaciones.
	MA-Pensamiento ComputacionalPR-3y4-OAC-01. Aplicar conceptos de Ciencias de la Computación -abstracción, organización lógica de datos, análisis de soluciones alternativas y generalización- al crear el código de una solución computacional.
Habilidad de resolver problema	MA08 OAH b. Evaluar procedimientos y comprobar resultados propios y de otros, de un problema matemático. problemas no rutinarios.
	MA-Pensamiento ComputacionalPR-3y4–OAH a. Construir y evaluar estrategias de manera colaborativa al resolver.
Habilidades del pensamiento computacional	Pensamiento Algorítmico

A.1 Primera Actividad Desconectada: Soy un Robot

Motivación: Controlar un robot en la superficie de Marte, a más de 200 millones de kilómetros de distancia, es uno de los mayores desafíos de la ingeniería moderna. Debido a la distancia, las señales de radio tardan entre 5 y 20 minutos en viajar de la Tierra a Marte. Esto significa que es imposible conducir el rover (como él o el PerseveranceCuriosity) en tiempo real con un joystick. Un error podría enviar al vehículo de mil millones de dólares a un cráter o atascarlo en la arena para siempre.

El problema es: ¿Cómo se le dan instrucciones a un robot para que navegue de forma segura y precisa por un terreno desconocido y peligroso cuando no puedes ver lo que hace al instante?

La solución es la programación algorítmica: Un equipo de ingenieros en el Laboratorio de Propulsión a Chorro (JPL) de la NASA actúa como "Programador". Analizan las imágenes del terreno marciano y escriben una secuencia de comandos muy específica (un algoritmo) para las tareas del día siguiente. Antes de enviar estas instrucciones

En esta actividad, nuestra sala de clases se convierte en el centro de control de misión del JPL. La cuadrícula en el suelo es su "superficie marciana". Ustedes son el equipo de ingenieros a cargo de guiar a su robot para que complete su misión con éxito. Cada comando cuenta.

Referencia: Ciencia de la NASA. (s.f.). Un día en la vida de un Mars Rover D. Mars Exploration Program. A Day in the Life of a Mars Rover Driver https://mars.nasa.gov/

Misión a Marte: ¡Soy un Rover!

Conceptos de Pensamiento Computacional:

- Algoritmos: Un algoritmo es una secuencia finita, ordenada y precisa de instrucciones no ambiguas diseñadas para resolver un problema. En esta actividad, los estudiantes no solo aprenden la definición, sino que la construyen. Crean un algoritmo tangible al escribir una lista de comandos (AVANZAR, GIRAR). La naturaleza literal del "Rover" humano les obliga a ser absolutamente precisos y a pensar secuencialmente, ya que cualquier ambigüedad o error en el orden provocará una falla visible e inmediata en la misión.

- Descomposición: Este pilar implica dividir un problema complejo en subproblemas más pequeños y manejables. El desafío "llevar el Rover de A a B" se descompone de forma natural en una serie de mini-misiones: "avanzar hasta la primera esquina", "realizar el giro correcto", "avanzar hasta el siguiente punto de decisión", etc. Los estudiantes deben resolver cada uno de estos pequeños pasos para construir la solución completa.

- Depuración (Debugging): Es el proceso sistemático de encontrar y corregir errores ("bugs") en un algoritmo. La actividad está diseñada para que los primeros intentos fallen. El momento "¡ANOMALÍA DETECTADA!" es el catalizador del proceso de depuración, que incluye: identificar dónde ocurrió el error, analizar por qué la instrucción fue incorrecta, proponer una corrección y volver a probar toda la secuencia. Esto normaliza el error como una parte fundamental y positiva del proceso de resolución de problemas.

Problema: Los estudiantes forman parte del equipo de control de misión del JPL de la NASA. Su desafío es programar la secuencia de comandos para un rover en la superficie de Marte (una cuadrícula en el suelo). Debido al retardo en las comunicaciones, no pueden controlarlo en tiempo real. Deben crear un algoritmo completo y preciso para guiar al rover desde su punto de aterrizaje hasta un objetivo científico, anticipando todos los movimientos y corrigiendo cualquier "anomalía" en la secuencia a través de un proceso de depuración.

Instrucciones:

- Preparación del Entorno

– Crear la Cuadrícula: Utiliza cinta adhesiva para dibujar una cuadrícula de 8x8 (o similar) en el suelo. Cada casilla debe ser lo suficientemente grande para que una persona pueda pararse dentro.

- Marcar Puntos Clave: Define y marca claramente una casilla como "INICIO" (Zona de Aterrizaje) y otra como "FIN" (Objetivo Científico). También agrega los obstáculos como rocas y marcianos.

- Formar Equipos: Al comenzar, indica a los estudiantes que se agrupen en equipos de tres integrantes.

- Inicio

1. Motivación - Noticia de Último Minuto desde el JPL: Comienza la clase con mencionando:
   "Noticias desde el Laboratorio de Propulsión a Chorro de la NASA. En el silencio tenso del centro de control, un equipo de ingenieros observa una pantalla. Lo que los separa de su objetivo son más de 200 millones de kilómetros de vacío helado. En la superficie de Marte, el rover Perseverance espera instrucciones. Pero hay un problema fundamental: debido a la distancia, cualquier señal de radio que envíen tardará entre 5 y 20 minutos en llegar. Eso significa que cada comando es un acto de fe enviado al futuro. Un solo error en el código, una instrucción ambigua, y para cuando el equipo en la Tierra se dé cuenta, el robot más avanzado del mundo podría haberse convertido en un monumento de mil millones de dólares a un cálculo fallido."

2. La Pregunta y la Solución:

   "Entonces, ¿cómo se navega con precisión milimétrica cuando se conduce a ciegas a través del sistema solar?". Guía la conversación hacia la respuesta:
   "La solución es la programación algorítmica, basada en mapas topográficos. Los ingenieros del JPL dividen el terreno marciano en un mapa de cuadrículas de navegación. Cada cuadro de la cuadrícula representa un área segura que el rover puede atravesar. Antes de que el

rover mueva un solo centímetro, el equipo escribe la secuencia completa de comandos, un algoritmo, para que navegue de forma autónoma por esta cuadrícula, moviéndose de un punto seguro al siguiente."

3. La Mision: *"En esta actividad, nuestra sala se convierte en ese centro de control de misión. La cuadrícula en el suelo no es solo un piso, es su mapa topográfico digital de Marte. Ustedes son el equipo de ingenieros a cargo de guiar a su rover para que complete su misión con éxito. Cada comando cuenta."*

4. Asignación de Roles de Misión: Explica los roles especializados:

    - Ingeniero(a) de Secuencia (Programador): El arquitecto de la ruta. Diseña y escribe la secuencia completa de comandos basándose en el mapa de cuadrículas.
    - Rover (Robot): La máquina en Marte. Solo ejecuta órdenes de forma literal. No piensa, no interpreta.
    - Controlador(a) de Misión (Tester): El enlace de comunicaciones. Transmite los comandos al Rover y supervisa la ejecución, siendo el primero en detectar anomalías.

- Desarrollo

1. Planificación de la Secuencia: Organiza los equipos. El/La Ingeniero(a) de Secuencia recibe la bitácora de misión (hoja y lápiz) y diseña la secuencia de comandos completa para mover al Rover desde la "Zona de Aterrizaje" al "Objetivo Científico".

2. Ejecución: El equipo se reúne en la "superficie marciana". El Rover se posiciona y el/la Controlador(a) de Misión comienza la "transmisión", leyendo los comandos uno por uno para que el Rover los ejecute.

3. El Ciclo de Depuración (Análisis de Anomalías): Cuando el Rover ejecuta un comando que resulta en un error, el/la Controlador(a) de Misión anuncia: "¡ANOMALÍA DETECTADA! ¡MISIÓN EN RIESGO!". La simulación se pausa, el Rover regresa al inicio, y el equipo analiza la secuencia para encontrar y corregir el error antes de reintentar la misión desde el principio.

Nota: Como Facilitador, fomenta una cultura de "no culpas". En la NASA, los errores se analizan para aprender. Usa frases como: *"Excelente hallazgo, equipo. Esta anomalía nos da información valiosa. ¿Qué hipótesis tienen sobre por qué falló la secuencia en ese punto?"*.

- Cierre

1. Misión Cumplida y Abstracción: Cuando un equipo completa el recorrido, han logrado el "Éxito de la Misión". Su siguiente tarea es documentarla creando un Diagrama de Flujo de la Misión. Muéstrales cómo usar el Óvalo para "Inicio/Fin de Secuencia" y Rectángulos para cada comando.
2. Informe de Misión: Invita a un equipo a presentar su secuencia final y su diagrama de flujo. Pide que expliquen cuál fue la "anomalía" más difícil de resolver.
3. Reflexión Guiada: Concluye con preguntas para conectar la simulación con el concepto:
    - *"¿Por qué fue crucial que el Rover solo siguiera las órdenes literalmente?"*
    - *"¿Qué sintieron cuando detectaron una anomalía? ¿Fue frustrante o una oportunidad para mejorar?"*

"Aparte de los rovers, ¿qué otras tecnologías creen que dependen de algoritmos que se programan por adelantado?"

Materiales: Superficie cuadriculada en el suelo (cinta adhesiva, tiza). Hoja y marcadores. Tarjetas de simbología básica de diagramas de flujo.

Guía del estudiante.

Misión a Marte: ¡Soy un Rover!

¡Bienvenido al Control de Misión! Tu equipo está a cargo de un rover en la superficie de Marte, a más de 200 millones de kilómetros. Las señales de radio tardan tanto en llegar que es imposible controlarlo en tiempo real. Un comando erróneo podría significar el fin de una misión de mil millones de dólares.

Para navegar de forma segura, los ingenieros de la NASA usan mapas de cuadrículas del terreno marciano. Tu misión es actuar como uno de esos ingenieros: deberás analizar el mapa de cuadrículas en el suelo y escribir una secuencia de comandos perfecta —un algoritmo— para que tu Rover se mueva de forma autónoma desde la "Zona de Aterrizaje" hasta un "Objetivo Científico". El éxito de la misión depende de tu precisión y planificación.

Instrucciones:

1. Formen su Equipo de Misión: Reúnanse en un grupo de tres y elijan sus roles:
    - Ingeniero(a) de Secuencia: El cerebro. Diseña y escribe la lista de comandos para navegar por el mapa de cuadrículas.
    - Rover: La máquina en Marte. Se mueve por la cuadrícula ejecutando órdenes sin pensar ni dudar.

- Controlador(a) de Misión: El enlace. Lee los comandos y supervisa la operación.

2. Fase 1: Diseña la Secuencia de Comandos
    - El/La Ingeniero(a) de Secuencia escribe en la "bitácora de misión" (una hoja) el plan completo para llegar a la meta. Solo puedes usar estos comandos:
        - AVANZAR (un cuadro en el mapa)
        - GIRAR_IZQUIERDA (girar 90° en el cuadro actual)
        - GIRAR_DERECHA (girar 90° en el cuadro actual)
3. Fase 2: Ejecuta
    - El Rover se posiciona en la "Zona de Aterrizaje".
    - El/La Controlador(a) de Misión toma la bitácora y lee el primer comando en voz alta. El Rover lo ejecuta. Continúen comando por comando.
4. Fase 3: Depura
    - Si el Rover se desvía de la ruta planificada en el mapa, el Controlador(a) de Misión anuncia: "¡ANOMALÍA DETECTADA!".
    - La misión se pausa. El Rover vuelve a la "Zona de Aterrizaje".
    - El equipo analiza el plan para encontrar el fallo. El/La Ingeniero(a) corrige la secuencia.
    - ¡Vuelvan a ejecutar la misión desde el principio con la secuencia corregida!
5. Fase 4: ¡Misión Cumplida!

- Cuando el Rover llegue al "Objetivo Científico", ¡felicidades!
- Documenten su éxito dibujando un Diagrama de Flujo de la Misión. Usen un óvalo para INICIO y FIN, y un rectángulo para cada COMANDO.

Para Comenzar:

- ¿Cuáles son las partes más pequeñas de este gran viaje? Piensa en llegar al primer punto de giro en la cuadrícula, luego al segundo...
- Antes de escribir, traza la ruta con tu dedo sobre el "mapa". "Traduce" cada movimiento en un comando. ¿Cuál es el primer comando: avanzar o girar?
- Los ingenieros de la NASA revisan sus planes varias veces. ¡No tengas miedo de equivocarte y corregir! Cada error es un dato nuevo para mejorar la misión.

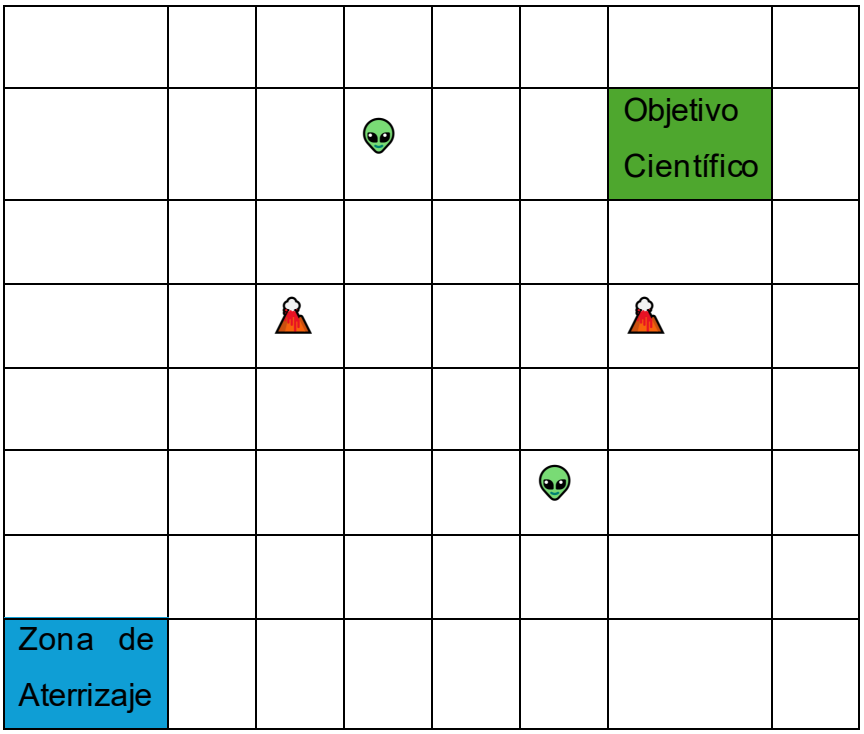


Ilustración 12. Cuadrícula Dibujada en el suelo

Fuente: Creación Propia.

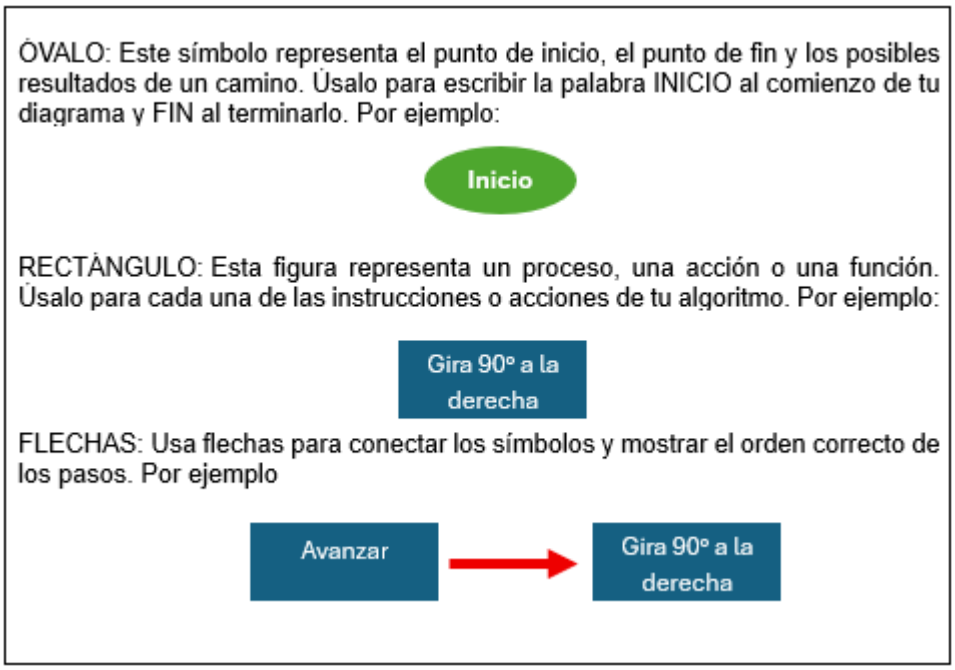


Ilustración 13. Simbología Diagrama de flujo

Fuente: Adaptado de "Símbolos comunes de los diagramas de flujo", por Lucidchart, (s.f).

A.2. Segunda Actividad Desconectada: Soy un Robot: Algoritmos con Lógica Condicional

Misión a Marte 2: Protocolos de Contingencia

Conceptos del pensamiento computacional:

- Algoritmos con Estructuras Condicionales: La actividad escala de algoritmos secuenciales a algoritmos dinámicos. Se introduce la Estructura Condicional ("Si... Entonces..."), definida como una instrucción que permite al algoritmo tomar decisiones y alterar su flujo de ejecución basándose en la evaluación de una condición (ej. ¿Hay un obstáculo al frente?). Los estudiantes ya no programan un camino fijo, sino un conjunto

de reglas de comportamiento que permiten al Rover reaccionar a un entorno impredecible.

- Descripción del Problema o Desafío Central: ¡Alerta de misión! Los mapas satelitales de la zona de aterrizaje eran incompletos. El terreno está lleno de rocas y obstáculos imprevistos, formando un laberinto. Una secuencia de comandos pre-programada es ahora inútil y peligrosa. El equipo de ingenieros debe abandonar el plan original y programar nuevos protocolos de contingencia: un conjunto de reglas condicionales que le den al Rover la "inteligencia" para analizar su entorno inmediato y tomar la mejor decisión en cada paso para encontrar una salida segura al objetivo científico.

Instrucciones:

- Preparación del Entorno
  - Crear la Cuadrícula: Utiliza cinta adhesiva para dibujar una cuadrícula de 8x8 (o similar) en el suelo. Cada casilla debe ser lo suficientemente grande para que una persona pueda pararse dentro.
  - Marcar Puntos Clave: Define y marca claramente una casilla como "INICIO" (Zona de Aterrizaje) y otra como "FIN" (Objetivo Científico). También agrega los obstáculos como rocas y marcianos.
  - Formar Equipos: Al comenzar, indica a los estudiantes que se agrupen en equipos de tres integrantes.
- Inicio

1. Motivación: ¡Misión en Peligro!: Comienza con un cambio dramático en la narrativa. *"Equipo de control de misión, tenemos una alerta crítica. El Rover ha aterrizado, pero la telemetría muestra que el terreno no coincide con nuestros mapas. Está rodeado de obstáculos imprevistos. El plan de*

navegación original es inútil. Si enviamos esa secuencia, el Rover chocará en el primer paso. La misión está en riesgo."

2. La Nueva Pregunta Clave: Plantea el nuevo problema: *"¿Cómo le ordenamos a un robot que navegue por un laberinto que no conocemos? No podemos darle un mapa paso a paso, pero... ¿podríamos darle inteligencia? ¿Podríamos enseñarle a pensar?"*
3. La Solución - Protocolos Condicionales: Introduce el concepto de forma intuitiva. *"La solución es darle reglas de decisión. En lugar de decirle 'Avanza, Avanza, Gira', le diremos: 'En cada paso, mira al frente. SI el camino está despejado, ENTONCES avanza. SI NO, gira a la derecha'. Este 'SI... ENTONCES...' es la base de toda la inteligencia artificial y se llama Lógica Condicional."*
4. Roles de Misión Actualizados: Los roles se mantienen, pero su función se vuelve más dinámica.
    - Ingeniero(a) de Secuencia: Ya no escribe una lista, sino el "libro de reglas" del Rover.
    - Controlador(a) de Misión (Tester): Ahora actúa como los sensores del Rover. Su trabajo es observar el entorno inmediato y reportar el estado al Ingeniero.
    - Rover: Sigue siendo el ejecutor, pero ahora espera una decisión en cada paso.

- Desarrollo

1. Diseño de los Protocolos: El/La Ingeniero(a) de Secuencia ya no escribe un guion largo, sino un conjunto de reglas condicionales. Anímales a empezar con un algoritmo simple de "seguir la pared":
    - *Regla 1:* SI (frente_bloqueado) ENTONCES (GIRAR_DERECHA)
    - *Regla 2:* SI (frente_despejado) ENTONCES (AVANZAR)
2. El Bucle de Navegación Autónoma: El proceso ahora es un ciclo que se repite en cada casilla:

- Paso 1 (Sentir): El Controlador de Misión (Sensor) observa el espacio justo delante del Rover y anuncia el estado: "¡FRENTE DESPEJADO!" o "¡OBSTÁCULO DETECTADO!".
- Paso 2 (Decidir): El Ingeniero de Secuencia escucha el estado, consulta su libro de reglas y anuncia qué regla se aplica. Por ejemplo: "Estado: Obstáculo detectado. Aplicar Regla 1: Comando es GIRAR_DERECHA".
- Paso 3 (Actuar): El Rover ejecuta ese único comando.
- El ciclo vuelve a empezar desde el Paso 1 en la nueva posición.

3. Depuración de la Lógica: El equipo notará que sus reglas pueden tener fallos (ej. el Rover se queda girando en un rincón). Aquí la depuración es más profunda.

Nota: Guía la reflexión. *"Veo que el Rover está en un bucle. ¿Significa que el Rover está roto? No, significa que nuestras reglas están incompletas. ¿Qué nueva regla o condición podríamos añadir para que sepa cómo salir de esa esquina?"*

- Cierre

1. Éxito y Abstracción con Decisión: Una vez que el Rover llega a la meta, el equipo ha diseñado un algoritmo autónomo exitoso. Ahora deben documentarlo en un Diagrama de Flujo.
    - Introduce el nuevo símbolo: el Rombo (Decisión). Explica: *"Dentro del rombo va la pregunta que el Rover se hace (ej: '¿Frente bloqueado?'). De él siempre salen dos flechas: una para 'Sí' y otra para 'No', cada una llevando a una acción diferente."*
2. Informe de Protocolos (Puesta en Común):
   Pide a un equipo que muestre su diagrama de flujo, enfocándose en cómo usaron el rombo para representar la "inteligencia" del Rover.
3. Reflexión Guiada:

- *"¿Qué fue más difícil: programar un camino fijo o programar un conjunto de reglas?"*
- *"¿Por qué un algoritmo con reglas condicionales es mucho más poderoso que uno con solo una secuencia de pasos?"*
- *"Piensen en su día a día. ¿En qué momentos usan la lógica 'SI... ENTONCES...'? (Ej: SI suena la alarma, ENTONCES me levanto. SI el semáforo está en rojo, ENTONCES me detengo)."*

Materiales: Superficie cuadriculada en el suelo (cinta adhesiva, tiza). Hoja y marcadores.

Guía del estudiante

Misión a Marte 2: Protocolos de Contingencia

Misión/Desafío:
¡ALERTA DE MISIÓN! El terreno en Marte es más peligroso de lo que mostraban los mapas. ¡Tu Rover está rodeado de obstáculos imprevistos! Un plan de ruta fijo es ahora un suicidio.

Tu nueva misión es darle "inteligencia" al Rover. En lugar de darle una lista de pasos, le darás un conjunto de reglas de decisión para que pueda navegar por el laberinto de forma autónoma. Debes programar un protocolo condicional (SI... ENTONCES...) que le permita analizar el terreno y tomar la decisión correcta en cada paso para llegar al objetivo.

Instrucciones:

1. Formen su Equipo de Misión: Mantengan sus roles, pero con funciones actualizadas:
    - Ingeniero(a) de Secuencia: Eres el cerebro. Diseñas las *reglas de comportamiento* del Rover.
    - Rover: El cuerpo. Ejecutas los comandos que te dan en cada paso.
    - Controlador(a) de Misión: Eres los sensores del Rover. En cada paso, observas y reportas lo que hay delante.
2. Fase 1: Diseña las Reglas de Decisión
    - En tu "bitácora de misión", escribe las reglas. No escribas AVANZAR, AVANZAR, GIRAR. Escribe reglas como:
        - SI *la casilla de enfrente está bloqueada*, ENTONCES *[escribe una acción aquí, ej: GIRAR_DERECHA]*.

- SI *la casilla de enfrente está libre*, ENTONCES *[escribe una acción aquí, ej: AVANZAR]*.

3. Fase 2: Navegación Autónoma (Paso a Paso)
    - Este es un ciclo que repetirán una y otra vez:
    - 1. OBSERVAR: El Controlador de Misión mira la casilla frente al Rover y anuncia: "¡OBSTÁCULO!" o "¡CAMINO LIBRE!".
    - 2. DECIDIR: El Ingeniero de Secuencia escucha el reporte, mira su libro de reglas y dice el comando a ejecutar.
    - 3. ACTUAR: El Rover ejecuta ese único comando.
    - ¡Repitan el ciclo desde el paso 1!
4. Fase 3: Depuración de Lógica
    - ¿El Rover se queda atascado en un rincón? ¡No es un bug del Rover, es un bug en tus reglas! Pausen la misión, discutan cómo mejorar las reglas (quizás necesitan una nueva regla para esa situación) y reanuden.
5. Fase 4: Diagrama de Decisión
    - Cuando el Rover llegue a la meta, ¡lo lograron! Ahora, dibujen su "inteligencia" en un Diagrama de Flujo. Usen el nuevo símbolo:
    - El Rombo (Decisión): Úsalo para las preguntas (ej: "¿Obstáculo al frente?"). De él deben salir dos caminos: uno para "Sí" y otro para "No".

Pistas para Comenzar:

- En lugar de pensar en toda la ruta, piensa en la pregunta más simple que el Rover debe hacerse en cada casilla.

- ¿Cuáles son las únicas dos respuestas posibles a esa pregunta? (Sí / No).
- ¿Qué acción única y simple debería tomar el Rover para cada una de esas dos respuestas?

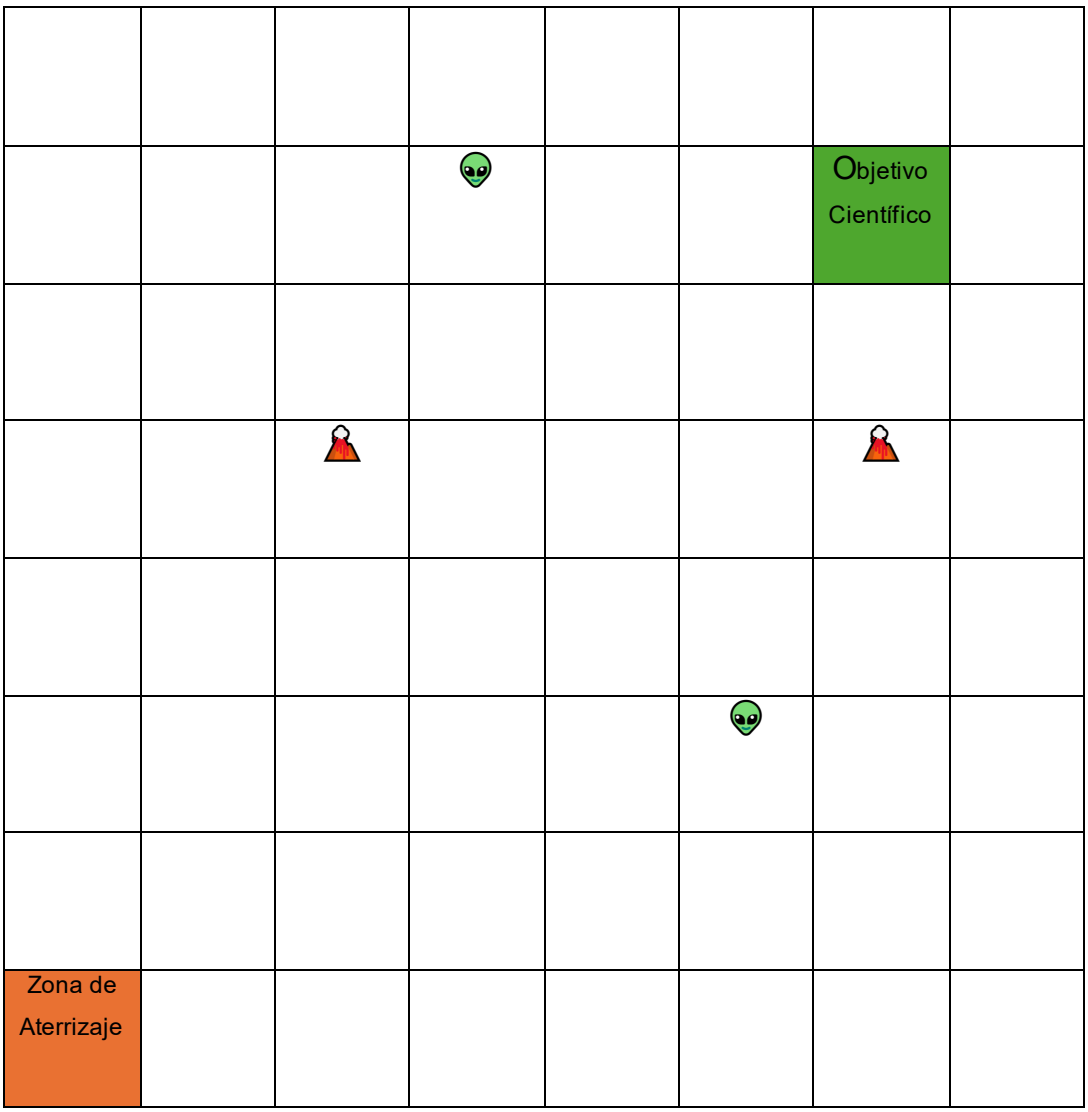


Ilustración 14 cuadricula dibujada en el suelo

Fuente: Creación Propia.

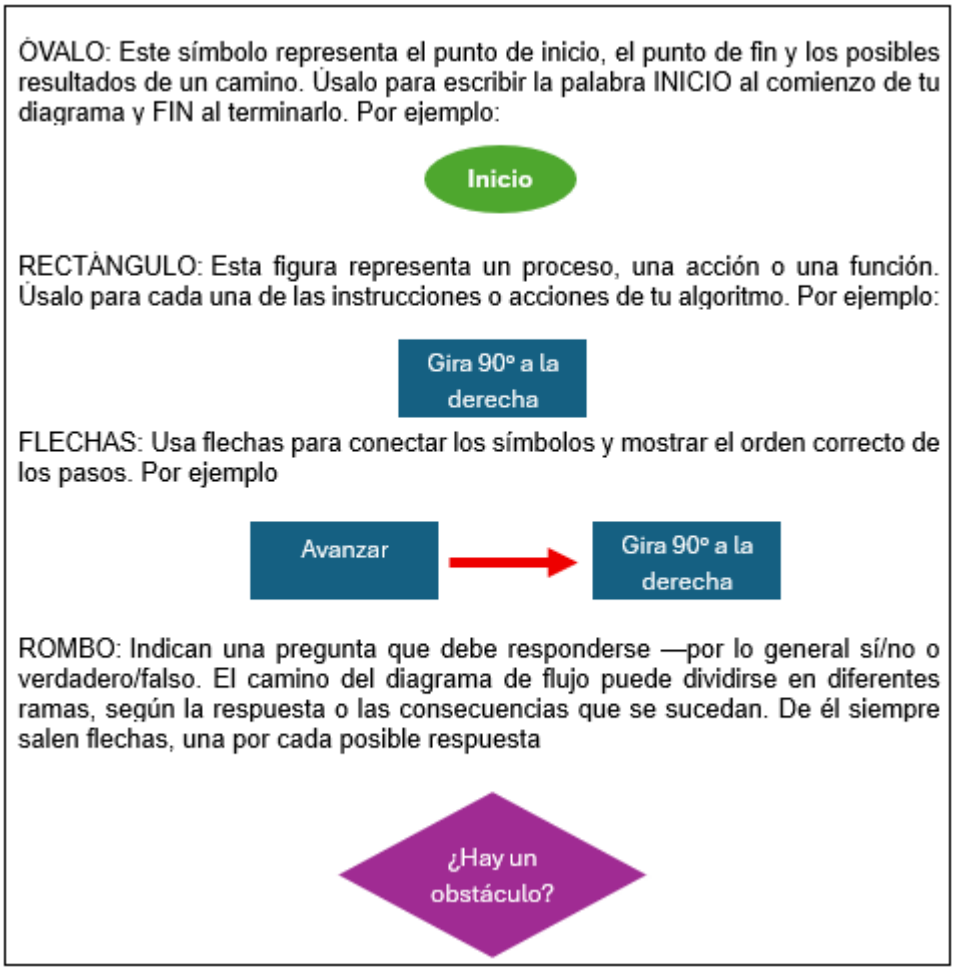


Ilustración 15 Simbología Diagrama de flujo

Fuente: Adaptado de "Símbolos comunes de los diagramas de flujo", por Lucidchart, (s.f).

A.3. Primera Actividad Conectada: Robot Digital: Secuencias en Code+

Concepto de pensamiento computacional:

- Evento: En programación es una acción específica, como un clic del mouse o presionar una tecla, que el programa puede detectar para desencadenar una respuesta. Hasta ahora, se ha usado un evento clave: el botón "Ejecutar".

El programa no se ejecuta solo. Está en un estado de espera. Un clic en "Ejecutar" es el evento que le da la señal para que reaccione, ejecutando una secuencia de bloques. Esta idea de "esperar a que algo pase para actuar" es la base de toda la interactividad.

El objetivo de esta actividad es transferir los conceptos algorítmicos al entorno digital utilizando un lenguaje de Programación Visual (o por bloques). Para aquello es necesario tener Computadoras o tablets con acceso a Code+. Para cada estudiante. Proyecto base configurado con un laberinto de ruta única.

Esta etapa traslada la experiencia física a la plataforma Code+. Se introduce la Programación por Bloques, un paradigma de programación que permite crear programas manipulando elementos gráficos (bloques) en lugar de escribir código textual, lo que reduce los errores de sintaxis y permite enfocarse en la lógica. Cada bloque representa una instrucción o estructura de control (parte del algoritmo).

Los estudiantes enfrentan un desafío de navegación en pantalla con una ruta predefinida. Deben construir el algoritmo seleccionando, arrastrando y conectando bloques secuencialmente (uno debajo del otro, indicando el orden de ejecución). La plataforma digital acelera el ciclo de prueba y error. Al presionar "Ejecutar", el estudiante visualiza inmediatamente el comportamiento del personaje. Si este no llega al objetivo, el entorno visual facilita la identificación

del bug (ej. un giro hacia el lado equivocado), permitiendo una depuración rápida mediante la reordenación o cambio de parámetros en los bloques.

Anexo.

1. Programación con Angry Birds - Curso C (2021): Lección 3.

https://studio.code.org/

2. Depuración en el laberinto - Curso C (2021): Lección 4.

https://studio.code.org/

A.4 Segunda Actividad Conectada: Robot Digital: Optimización de Rutas en Code+

Mediante el uso de la plataforma de programación por bloques Code+, los estudiantes darán un salto conceptual desde la programación secuencial a la creación de agentes autónomos. El objetivo es que programen un personaje (sprite) que pueda navegar un entorno complejo y con múltiples rutas de forma independiente. Para lograrlo, deberán abandonar la creación de largas listas de comandos fijos y, en su lugar, diseñar un "cerebro" para el sprite utilizando la estructura de control fundamental de la programación: la lógica condicional (SI... ENTONCES...).

La actividad introduce el bloque condicional como la herramienta que otorga independencia al personaje. Se explicará que la parte del SI actúa como un sensor, permitiendo al sprite "hacer una pregunta" sobre su entorno inmediato (ej. *"¿Hay un muro al frente?"*, *"¿Hay un camino a la derecha?"*). La parte del ENTONCES contiene la acción que el sprite ejecutará únicamente si la respuesta a esa pregunta es verdadera. Al anidar y combinar estas reglas, los estudiantes ya no programan una ruta, sino que programan un comportamiento, una lógica de decisión que permite al sprite reaccionar y adaptarse al entorno en tiempo real.

Este nuevo enfoque redefine el concepto de optimización. Ya no se trata de encontrar la ruta con el menor número de pasos, sino de diseñar el algoritmo más eficiente, robusto y elegante. Una solución con unos pocos bloques condicionales que puede resolver múltiples laberintos es inherentemente más optimizada que un algoritmo secuencial de cincuenta pasos que solo funciona para un único camino. Este es un ejercicio de abstracción, donde los estudiantes deben crear reglas generales que funcionen en muchas situaciones, en lugar de instrucciones específicas para un solo caso.

El desafío comienza cuando los estudiantes se enfrentan a un laberinto en Code+ que es demasiado complejo para resolverlo con una secuencia manual. Al intentar programar cada paso, se darán cuenta de la ineficiencia y fragilidad de ese método. Este será el punto de partida para explorar los bloques condicionales. El proceso de aprendizaje se centrará en el ciclo de prueba y error, donde los estudiantes diseñarán un conjunto de reglas, las ejecutarán y observarán el comportamiento del sprite. La depuración (debugging) se volverá más sofisticada, ya que no buscarán un error en una secuencia, sino una falla en la lógica de las reglas que hace que el sprite se quede atascado en un bucle o tome decisiones ineficientes.

La actividad concluye cuando los estudiantes logran un algoritmo de reglas que guía a su sprite de forma autónoma hasta el objetivo. En la fase final, se analizarán y compararán las diferentes soluciones lógicas de los equipos. Se guiará una discusión para que los estudiantes puedan argumentar por qué un código basado en condicionales es más poderoso y optimizado que uno secuencial, consolidando así su comprensión sobre cómo se construye la autonomía y la "inteligencia" en un programa a través de reglas lógicas.

Anexo.

1. Condicionales (Si/sino) con Abeja - Curso Express (2021) Lección 15 https://studio.code.org/
2. Bucles 'mientras' en Granjera - Curso Express (2021) Lección 16 https://studio.code.org/

B. Fiesta de Baile

<table>
<tr><td colspan="2">Nombre: Fiesta de Baile</td></tr>
<tr><td colspan="2">Fuente:
Animum Creativity Advanced School. (2023, 21 de septiembre). ¿Qué es un loop y qué papel juega en la animación 3D? Animum. https://www.animum3d.com/

Code.org. (s.f.). Fiesta Desenchufada: Unidad 1 - Lección 1: Calentamiento. https://studio.code.org/
Code.org. (s.f.). Fiesta de baile (edición 2019): Unidad 1. https://studio.code.org/
Code.org. (s.f.). Curso B (2021): Lección 7. Bucles con la cosechadora. https://studio.code.org/
Code.org. (s.f.). Curso B (2021): Lección 8. Bucles con Laurel. https://studio.code.org</td></tr>
<tr><td colspan="2">Objetivo de la actividad: Aplicar bucles simples y bucles anidados para representar y optimizar secuencias repetitivas de movimientos en coreografías y en programas por bloques (Code+), identificando patrones en la secuencia, reduciendo la cantidad de instrucciones y explicando por qué la nueva versión es más eficiente.</td></tr>
<tr><td>Objetivos de Aprendizaje sugeridos:</td><td>MA1M OA 11
Representar el concepto de homotecia de forma vectorial, relacionándolo con el producto de un vector por un escalar, de manera manual y/o con software educativo.</td></tr>
<tr><td>Habilidad de resolver problemas</td><td>MA01M OA a.
Resolver problemas utilizando estrategias como las siguientes:
• Simplificar el problema y estimar el resultado.</td></tr>
</table>

	• Descomponer el problema en subproblemas más sencillos. • Buscar patrones. • Usar herramientas computacionales.
Habilidades del Pensamiento Computacional	Pensamiento Algorítmico.

B.1. Primera Actividad Desconectada: Fiesta de Baile

Conceptos de pensamiento computacional:

- Eventos: La programación es dirigida por eventos, donde el flujo del programa es determinado por acciones o "disparadores". Los estudiantes actúan como un sistema reactivo que espera una señal (el botón de color) para ejecutar una acción (el baile).
- Bucles (Loops): Un Bucle es una estructura de control en programación que permite repetir un bloque de instrucciones un número determinado de veces o mientras se cumpla una condición. Su propósito principal es la automatización de tareas repetitivas para hacer el código más corto, eficiente y legible.
- Reconocimiento de Patrones: Es la habilidad de observar datos o secuencias de información (como nuestra coreografía) e identificar tendencias, similitudes o sub-secuencias que se repiten. Es un paso fundamental antes de poder optimizar, ya que para crear un bucle, primero se debe encontrar el patrón que se va a repetir.

El Secreto Mágico de la Animación

Motivación: Los "bucles" son un pilar fundamental en el mundo de la animación y los videojuegos. Antes de comenzar, se puede plantear una pregunta para captar el interés: *"¿Alguna vez se han preguntado cómo los creadores de videojuegos hacen que un personaje camine sin parar o cómo en los GIFs un movimiento se repite perfectamente?"*. La respuesta es el uso inteligente de los bucles, una técnica que permite que una secuencia de movimiento se repita de forma infinita, creando una ilusión de continuidad sin necesidad de animar cada segundo desde cero.

La gran ventaja de los bucles es que ahorran una enorme cantidad de tiempo y trabajo. Esta idea de eficiencia se alinea perfectamente con el desafío central de la actividad: los estudiantes primero experimentarán el "trabajo duro" de seguir una secuencia larga y repetitiva, para luego descubrir el "poder" de optimizarla. Para que un bucle funcione en animación, el animador primero debe identificar el patrón exacto del movimiento que quiere repetir.

Referencia: Animum Creativity Advanced School. (2023, 21 de septiembre). *¿Qué es un loop y qué papel juega en la animación 3D?* Animum. https://www.animum3d.com/

Actividad: ¡Fiesta de Código! Descubriendo el Secreto del Animador

Objetivo de Aprendizaje: Aplicar el concepto de bucle para optimizar una secuencia repetitiva, conectando este proceso con su aplicación en diversos desafios.

El Problema: Los estudiantes actuarán primero como un personaje de videojuego, reaccionando en tiempo real a "eventos". Luego, se pondrán en el rol de un animador profesional que enfrenta un desafío de eficiencia: recibir una coreografía larga y repetitiva y encontrar una manera mucho más corta y

optimizada de escribir esas mismas instrucciones, descubriendo así la necesidad y el poder de los bucles.

Materiales:

Proyector, pantalla y computador con programa de presentación; diapositivas preparadas (pasos de baile, botones de evento); música animada y sistema de sonido; y hojas de papel y lápices.

Anexo:

Lista Pública Code.org Dance Party (all-ages) – Spotify Code.org Dance Party (all-ages) - playlist by Code.org | Spotify

Drive: https://drive.google.com/

Guía para el Estudiante

¡Fiesta de Código! Descubriendo el Secreto del Animador

Tu misión es descubrir el secreto que usan los animadores profesionales para crear movimientos infinitos en los videojuegos y GIFs. Primero, actuarás como un personaje interactivo que reacciona a los eventos (comandos de colores). Luego, te pondrás en el rol de un animador y deberás encontrar la forma de optimizar una larga secuencia de baile usando un poderoso truco de la industria: el bucle.

Los personajes de videojuegos y los GIFs no se mueven por arte de magia. Usan trucos de programación para repetir acciones de forma eficiente. Hoy, tú descubrirás uno de esos trucos, ¡y lo harás bailando!

Instrucciones:

1. Fase 1: Aprende tus "Fotogramas de Animación"
    - Si ves el BOTÓN VERDE → Realiza un APLAUSO.
    - Si ves el BOTÓN NARANJA → Realiza un GIRO.
    - Si ves el BOTÓN AZUL → Realiza un SALTO.
2. Fase 2: Modo Personaje Interactivo
    - ¡Es hora de la acción! Cuando la música comience, reacciona a los eventos (botones de colores) que aparezcan en la pantalla. ¡Actúa como un personaje de videojuego respondiendo a los comandos de un jugador!
3. Fase 3: Modo Animador
    - Toma tu hoja y lápiz. Anota la secuencia de colores (fotogramas) que el profesor dictará. Sé preciso, ¡cada fotograma cuenta!
4. Fase 4: Modo Animador PRO
    - Mira la larga lista que escribiste. Es ineficiente. Busca un patrón, un grupo de movimientos que se repita una y otra vez.
    - Tu desafío es reescribir toda la secuencia usando la menor cantidad de palabras posible. ¿Cómo le dirías a alguien que repita ese patrón varias veces?

Antes de Comenzar:

- Para reaccionar: No pienses, ¡solo actúa! Enfócate en el evento que está sucediendo *ahora*.

- Para optimizar: Lee tu lista en voz alta. ¿Suena repetitivo? Identifica el grupo de acciones que se repite y cuenta cuántas veces lo hace.

Ilustración 16 PPT con los pasos de baile

Fuente: Tomado de "Dance Party: Unplugged", por Code.org, (s.f).

Pasos a Utilizar:

- ***[A]*** = *Aplauso*
- ***[G]*** = *Giro*
- ***[S]*** = *Salto*

Secuencia a Dictar:

1. ***"Salto"***
2. ***"Aplauso"***
3. ***"Giro"***
4. ***"Aplauso"***
5. ***"Giro"***
6. ***"Salto"***
7. ***"Aplauso"***
8. ***"Giro"***
9. ***"Aplauso"***
10. ***"Giro"***
11. ***"Salto"***

Ilustración 17. Ejemplo de secuencia de pasos de baile.

Fuente: Creación Propia.

B.2. Segunda Actividad Desconectada: Fiesta de Baile 2

¡Fiesta de Código 2! El Ritmo Anidado

Conceptos de pensamiento computacional:

- Bucles Anidados (Nested Loops). Es una estructura de control donde un bucle se ejecuta dentro de otro. Donde el bucle interno completa todas sus repeticiones por cada repetición individual del bucle externo. Esto permite la creación de patrones complejos y la automatización de tareas que tienen una estructura repetitiva de múltiples niveles.

Esta actividad se presenta como un desafío coreográfico avanzado para explorar la eficiencia algorítmica y la creación de patrones complejos. Se inicia repasando un repertorio de pasos de baile a los que se les asigna un "evento" (un disparador visual). Se dicta una coreografía compleja, diseñada para contener patrones dentro de otros patrones. Por ejemplo: (Aplauso, Giro), (Aplauso, Giro), (Dab), (Aplauso, Giro), (Aplauso, Giro), (Dab). El primer nivel de análisis consiste en que los estudiantes identifiquen el patrón más inmediato (Aplauso, Giro) y lo optimicen con un Bucle simple. El desafío se eleva al solicitar que analicen esta nueva secuencia resumida para encontrar un patrón de nivel superior, lo que conduce a la introducción del concepto de Bucle Anidado (Nested Loop), que se define como un bucle que se encuentra dentro de otro bucle. La actividad culmina con el análisis de cómo los bucles anidados permiten crear comportamientos complejos con muy pocas líneas de instrucción, un principio fundamental en la programación.

Objetivo de Aprendizaje: Aplicar bucles anidado para representar una secuencia de acciones de la forma óptima.

El Problema: Los estudiantes asumen el rol de "Coreógrafos Maestros" cuyo trabajo es no solo crear un baile, sino escribir sus instrucciones de la forma más eficiente posible. Se les presenta una coreografía larga y compleja que parece difícil de memorizar y transcribir. El desafío es analizarla, identificar sus patrones ocultos en diferentes niveles y reescribirla usando la menor cantidad de instrucciones, descubriendo así el poder de los bucles anidados para generar complejidad de forma simple.

Materiales: Proyector, pantalla y computador; diapositivas preparadas (pasos de baile, botones de evento); y hojas de papel y lápices para cada estudiante.

Anexo:

Lista Pública Code.org Dance Party (all-ages) – Spotify Code.org Dance Party (all-ages) - playlist by Code.org | Spotify

Drive: https://drive.google.com/

¡Fiesta de Código 2! El Ritmo Anidado

¡Has sido ascendido a Coreógrafo Maestro! Tu misión es decodificar una rutina de baile compleja que contiene patrones ocultos dentro de otros patrones. Deberás analizar la coreografía, optimizarla por capas y descubrir el poder de los bucles anidados para escribir las instrucciones de la forma más corta, elegante y profesional posible.

El Problema: Ya sabes usar bucles simples para repetir acciones. Pero las coreografías y los programas reales a menudo tienen una complejidad mayor, con ritmos que se repiten dentro de una estructura más grande. Hoy aprenderás a manejar esa complejidad.

Instrucciones:

1. Fase 1: Los Movimientos
    - Repasemos el repertorio: Aplauso, Giro y Dab.
2. Fase 2: Transcripción de la Coreografía
    - Toma tu hoja y lápiz. Escucha con atención y escribe la secuencia completa de pasos que dictará el profesor. ¡No te saltes ninguno!
3. Fase 3: Optimización Nivel 1 (El Bucle Simple)
    - Mira tú larga lista. Ignora todo lo demás y busca el patrón más pequeño y obvio que se repita.
    - Reescribe toda la coreografía en una nueva línea, reemplazando ese patrón repetido con una instrucción de "Repetir X veces".
4. Fase 4: Optimización Nivel 2 (El Bucle Anidado)
    - Ahora, enfócate únicamente en la nueva línea que acabas de escribir. ¿Ves un patrón más grande que se repita?
    - Tu desafío final es reescribir la secuencia una última vez, de la forma más corta posible, usando un bucle para controlar otro bucle.

Antes de Comenzar:

- Para el Nivel 1: La acción "Dab" es una pista. A menudo, las acciones únicas separan los bloques que se repiten. ¿Qué pasa justo antes de cada "Dab"?
- Para el Nivel 2: Lee en voz alta la secuencia que escribiste en la Fase 3. ¿Suena como si estuvieras diciendo la misma combinación de bloques dos veces? Si es así, ¡has encontrado el patrón más grande!

Ilustración 1 Presentación ppt con los pasos de baile.

Fuente: Tomado de "Dance Party: Unplugged", por Code.org, s.f.

Pasos a Utilizar:

- ***[A]*** = *Aplauso*
- ***[P]*** = *Paso Lateral (derecha)*
- ***[D]*** = *Dab*

Secuencia a Dictar:

1. ***"Paso Lateral"***
2. ***"Paso Lateral"***
3. ***"Aplauso"***

4. *(Pausa corta)*
5. *"Paso Lateral"*
6. *"Paso Lateral"*
7. *"Aplauso"*
8. *(Pausa larga)*
9. *"Dab"*
10. *(Pausa muy larga para marcar el fin del bloque grande)*
11. *"Paso Lateral"*
12. *"Paso Lateral"*
13. *"Aplauso"*
14. *(Pausa corta)*
15. *"Paso Lateral"*
16. *"Paso Lateral"*
17. *"Aplauso"*
18. *(Pausa larga)*
19. *"Dab"*

Ilustración 18. Ejemplo de secuencia de pasos de baile

Fuente: Creación Propia.

B. 3. Primera Actividad Conectada: Optimización de Código con Bucles en Code+

La actividad transita al entorno digital de Code+, donde los estudiantes trabajan en parejas. Se les presentan escenarios de navegación (laberintos o caminos) diseñados intencionadamente con secciones muy repetitivas. Por ejemplo, un largo pasillo recto que requiere 10 bloques de "avanzar", o un camino en zigzag que repite la secuencia "avanzar, girar derecha, avanzar, girar izquierda" varias veces.

Inicialmente, se anima a los estudiantes a resolver el problema de la forma que les resulte más intuitiva, lo que probablemente resultará en una larga y redundante secuencia de bloques idénticos. Una vez que tienen una solución funcional pero ineficiente, se plantea el desafío de la optimización. Se les pregunta cómo podrían lograr el mismo resultado con muchos menos bloques.

Este es el momento de introducir o reforzar el bloque de "repetir" o "bucle" de Code+. El objetivo es que los estudiantes identifiquen los patrones en su propio código (ej. los diez bloques de "avanzar") y los reemplacen con una única estructura de bucle (ej. un bloque "repetir 10 veces" que contenga un solo bloque de "avanzar"). La actividad se centra en la refactorización del código: pasar de una solución que funciona a una solución que es elegante, eficiente y fácil de leer.

Anexo:

Curso B (2021). Bucles con la cosechadora: https://studio.code.org/

Curso B (2021). Bucles con Laurel: https://studio.code.org/

B.4 Segunda Actividad Conectada: Optimización de Código con Bucles en Code+

Esta actividad se presenta como un desafío para explorar la eficiencia algorítmica a través de la creación de patrones complejos. Se inicia con un escenario donde los estudiantes deben resolver una tarea que requiere la generación de un resultado elaborado, como dibujar una figura geométrica elaborada o completar un laberinto. La característica principal del desafío es que el resultado final contiene un patrón pequeño que se repite para formar una estructura más grande. La tarea de los estudiantes es escribir el algoritmo más corto y eficiente posible para lograr el objetivo.

Se procede a guiar a los estudiantes para que primero analicen el resultado final y descompongan el problema. Inicialmente, se les anima a concentrarse en crear

solo una pequeña parte del patrón general. Este enfoque los llevará a identificar un patrón inmediato, que es la secuencia de bloques que crea el elemento básico y repetitivo. Se formaliza este hallazgo reforzando el uso de un Bucle simple para encapsular esta secuencia. Al ejecutar este primer programa, los estudiantes observarán que su código genera correctamente una parte de la estructura final, pero no la completa.

El desafío se eleva al solicitar que analicen por qué su solución actual es incompleta y qué se necesita para que la repetición del patrón continúe en la posición o estado correcto. Se guía la observación para que noten que después de completar la primera serie de repeticiones, a menudo se necesita una secuencia de comandos de "transición" o "reajuste" que prepara al programa para la siguiente gran repetición. Esto conduce a la introducción del concepto de Bucle Anidado (Nested Loop), que se define como un bucle que se encuentra y se ejecuta dentro de las instrucciones de otro bucle.

Anexo.

Curso D (2021) Bucles anidados en Laberinto. https://studio.code.org/

Curso E (2021). Formas elegantes con bucles anidados
https://studio.code.org/

C. Magia con Números Develado el Código Binario

<table>
<tr><td colspan="2">Nombre: Magia con Números Develado el Código Binario</td></tr>
<tr><td colspan="2">Fuente:
SOCHIEM (s.f.). Kit Didáctico – Magia. Festival de Matemática. https://festivaldematematica.org/kit-didacticos/#magia
The National Museum of Computing. (s.f.). Colossus. https://www.tnmoc.org/</td></tr>
<tr><td colspan="2">Objetivo de la actividad: Representar y descomponer números naturales hasta como suma de potencias de 2, utilizando tarjetas binario-mágicas y notación binaria, para descubrir el principio matemático del truco de adivinación y justificar la conversión entre representaciones decimal y binaria.</td></tr>
<tr><td rowspan="3">Objetivo de Aprendizaje sugeridos:</td><td>MA 08B OA 3. Explicar la multiplicación y la división de potencias de base natural y exponente natural hasta 3, de manera concreta, pictórica y simbólica.</td></tr>
<tr><td>MA1M OA2 Mostrar que comprenden las potencias de base racional y exponente entero:
• Transfiriendo propiedades de la multiplicación y división de potencias a los ámbitos numéricos correspondientes.
• Relacionándolas con el crecimiento y decrecimiento de cantidades.
Resolviendo problemas de la vida diaria y otras asignaturas.</td></tr>
<tr><td>MA-Pensamiento ComputacionalPR-3y4-OAC-02.
Representar diferentes tipos de datos en una variedad de formas que incluya textos, sonidos, imágenes y números.</td></tr>
<tr><td>Habilidad de resolver problema:</td><td>MA08 OAH a
Resolver problemas utilizando estrategias tales como:
• Destacar la información dada.</td></tr>
</table>

	• Usar un proceso de ensayo y error sistemático. • Aplicar procesos reversibles. • Descartar información irrelevante. • Usar problemas similares.
	MA1M OAH a Resolver problemas utilizando estrategias como las siguientes: • Simplificar el problema y estimar el resultado. • Descomponer el problema en subproblemas más sencillos. • Buscar patrones. • Usar herramientas computacionales.
	MA-Pensamiento ComputacionalPR-3y4-OAH a. Construir y evaluar estrategias de manera colaborativa al resolver problemas no rutinarios
Habilidades del pensamiento computacional:	Abstracción y Descomposición.

C.1. Primera Actividad Desconectada Magia con Números: Develando el Código Binario

Motivación: Durante la Segunda Guerra Mundial, las comunicaciones del Alto Mando alemán estaban protegidas por un sistema de cifrado extremadamente complejo llamado "Lorenz". Los mensajes enviados con esta máquina eran, en teoría, indescifrables y contenían información estratégica vital. Para los Aliados, romper este código no era un simple desafío; era una necesidad crítica que podía cambiar el curso de la guerra. El problema era que el cifrado Lorenz era tan avanzado que analizarlo a mano resultaba imposible. La solución llegó en

Bletchley Park (Reino Unido), donde un equipo liderado por el ingeniero Tommy Flowers diseñó y construyó Colossus, la primera computadora programable, electrónica y digital del mundo.

Para procesar la increíble cantidad de información y encontrar los patrones que revelaran el código, Colossus no usaba números como nosotros, sino un sistema mucho más simple y rápido: el código binario. Representaba cada letra y cada instrucción como una serie de pulsos eléctricos que estaban "encendidos" (1) o "apagados" (0). Esta máquina demostró que cualquier información, por compleja que fuera, podía ser descompuesta y representada usando solo dos símbolos. El "secreto" para descifrar los mensajes alemanes era entender este lenguaje fundamental de las máquinas.

Referencia: The National Museum of Computing. (s.f.). *Colossus*. https://www.tnmoc.org/colossus

En esta actividad, los estudiantes no descifrarán un código de guerra, pero sí se convertirán en "criptógrafos" que revelarán el secreto detrás de un truco de magia. Descubrirán que el principio de codificación que usaremos para adivinar un número es exactamente el mismo que Colossus usó para procesar información y cambiar la historia: el poder del código binario

Esta actividad consta de dos partes, la primera comprender el “truco de magía”, mientras que la segunda, corresponde a crear sus propias tarjetas para aplicar al grupo.

Para aquello necesitamos: Sets del juego de magia de 5 cartas (0-31). Sets de tarjetas de potencias de 2 (2^0 a 2^4). Guía de trabajo sobre descomposición en potencias de 2 y escritura binaria.

Primera Parte

La actividad se inicia como una experiencia lúdica e intrigante. Como grupo curso, los estudiantes observan un set de 5 tarjetas (para números del 0 al 31) proyectada por el docente en la pizarra. Un estudiante voluntario/a piensa un número secreto en ese rango, mientras el docente ocupa el rol de "mago". El mago muestra cada una de las 5 tarjetas y pregunta: "¿Está tu número en esta tarjeta?". Basándose únicamente en las respuestas afirmativas ("sí"), el mago es capaz de adivinar el número secreto sumando el primer número que aparece en la esquina superior de cada tarjeta correspondiente.

Si el número secreto es 21, el participante dirá "Sí" a las siguientes tarjetas:

- La que empieza con 16 (porque 21 contiene 16)
- La que empieza con 4 (porque 21 contiene 4)
- La que empieza con 1 (porque 21 contiene 1)

El mago suma: 16 + 4 + 1 = 21.

Lo que realmente ocurrió en lenguaje binario es esto:

Potencia de 2	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
¿Está el 21?	Sí	No	Sí	No	Sí
Valor del Bit	1	0	1	0	1

Tras realizar el truco varias veces para generar curiosidad, la actividad transita hacia una fase de análisis y conjetura. Se desafía a los estudiantes a que, en sus grupos, examinen las tarjetas y busquen patrones. Deben discutir y formular hipótesis sobre por qué el truco siempre funciona. El docente puede guiar preguntando:

- Pregunta de Patrones: Miren los números en cada tarjeta. ¿Qué tienen en común? ¿Por qué el número 31 está en todas las tarjetas? ¿Y el 1 solo en una?

- Pregunta Clave: Observen el primer número de cada tarjeta (1, 2, 4, 8, 16). ¿Qué tipo de secuencia matemática es esta? ¿Cómo se relaciona con el sistema binario?
- Pregunta Lógica: ¿Qué representa realmente una respuesta "Sí"? Si una tarjeta representa una potencia de 2, ¿qué significa que tu número esté en ella?

Posteriormente, se guía la formalización del concepto. El foco se dirige al primer número de cada tarjeta: 1, 2, 4, 8, 16. Se establece que estos números no son aleatorios, sino que corresponden a las potencias de 2 (2^0, 2^1, 2^2, 2^3, 2^4). Se explica el fundamento matemático: cualquier número entero puede ser expresado de forma única como una suma de distintas potencias de 2. Cada tarjeta agrupa a todos los números que, en su "receta" de potencias de 2, incluyen la potencia que la identifica. Un "sí" a una tarjeta significa que esa potencia de 2 es un ingrediente del número secreto. La suma final, por lo tanto, reconstruye el número original.

Segunda Parte

Para consolidar este conocimiento, se les pregunta a los estudiantes, ¿Se podrá realizar el truco de magia con números desde el 1 hasta el 63? ¿Qué se necesita para lograrlo? Se espera que los estudiantes puedan realizar conjeturas en voz alta, y recoger esta información para brindar un ayuda a los grupos y termina solicitando que se separen por grupos de 4 estudiantes y cada uno realizará sus propias tarjetas para que el truco pueda ser aplicado hasta 63 números.

Se le entregará a cada grupo hojas de oficio para que puedan utilizarlas de borrador y escribir ahí las “tarjetas mágicas”, además se les realizará las siguientes preguntas de apoyo:

- ¿Cuántas tarjetas se deben utilizar?
- ¿El 63 en que tarjetas debe estar?

- ¿El 47 en que tarjetas debe estar?

Una vez que tengan listas las tarjetas mágicas, se les entrega 6 trozos de block, hojas de oficio, etc. Donde puedan crear sus tarjetas mágicas definitivas, las que incluso pueden ser plastificadas con posterioridad.

Anexo:

1	9	17	25
3	11	19	27
5	13	21	29
7	15	23	31

2	10	18	26
3	11	19	27
6	14	22	30
7	15	23	31

4	12	20	28
5	13	21	29
6	14	22	30
7	15	23	31

8	12	24	28
9	13	25	29
10	14	26	30
11	15	27	31

16	20	24	28
17	21	25	29
18	22	26	30
19	23	27	31

Ilustración 19. Tarjetas binario-mágicas del 1 al 31

Fuente: Adaptación de Cartillas de festival de matemática.

C.2. Segunda Actividad Desconectada: Magia con Números: Construyendo el Código Binario.

Esta actividad utiliza el formato de un "truco de magia" para introducir a los estudiantes en el sistema de numeración binario, la descomposición de números en potencias de 2 y los fundamentos de la teoría de la información.

Para aquello necesitamos: Sets del juego de magia de 5 cartas (0-31). Material para la creación de nuevas tarjetas (cartulina, marcadores). Guía de trabajo sobre sistema binario y conversiones.

La actividad se inicia como una experiencia lúdica e intrigante. Como grupo curso, los estudiantes observan un set de 5 tarjetas (para números del 0 al 31) proyectada por el docente en la pizarra. Un estudiante voluntario/a piensa un número secreto en ese rango, mientras el docente ocupa el rol de "mago". El mago muestra cada una de las 5 tarjetas y pregunta: "¿Está tu número en esta

tarjeta?". Basándose únicamente en las respuestas afirmativas ("sí"), el mago es capaz de adivinar el número secreto sumando el primer número que aparece en la esquina superior de cada tarjeta correspondiente.

Si el número secreto es 21, el participante dirá "Sí" a las siguientes tarjetas:

- La que empieza con 16 (porque 21 contiene 16)
- La que empieza con 4 (porque 21 contiene 4)
- La que empieza con 1 (porque 21 contiene 1)

El mago suma: 16 + 4 + 1 = 21.

Lo que realmente ocurrió en lenguaje binario es esto:

Potencia de 2	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
¿Está el 21?	Sí	No	Sí	No	Sí
Valor del Bit	1	0	1	0	1

Esta actividad utiliza el formato de un "truco de magia" para introducir a los estudiantes en el sistema de numeración binario, la descomposición de números en potencias de 2 y los fundamentos de la teoría de la información.

Para aquello necesitamos: Sets del juego de magia de 5 cartas (0-31). Material para la creación de nuevas tarjetas (cartulina, marcadores). Guía de trabajo sobre sistema binario y conversiones.

La actividad se inicia como una experiencia lúdica e intrigante. Como grupo curso, los estudiantes observan un set de 5 tarjetas (para números del 0 al 31) proyectada por el docente en la pizarra. Un estudiante voluntario/a piensa un número secreto en ese rango, mientras el docente ocupa el rol de "mago". El mago muestra cada una de las 5 tarjetas y pregunta: "¿Está tu número en esta tarjeta?". Basándose únicamente en las respuestas afirmativas ("sí"), el mago es

capaz de adivinar el número secreto sumando el primer número que aparece en la esquina superior de cada tarjeta correspondiente.

Si el número secreto es 21, el participante dirá "Sí" a las siguientes tarjetas:

- La que empieza con 16 (porque 21 contiene 16)
- La que empieza con 4 (porque 21 contiene 4)
- La que empieza con 1 (porque 21 contiene 1)

El mago suma: 16 + 4 + 1 = 21.

Lo que realmente ocurrió en lenguaje binario es esto:

Potencia de 2	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
¿Está el 21?	Sí	No	Sí	No	Sí
Valor del Bit	1	0	1	0	1

Luego los estudiantes interactúan con las tarjetas, desempeñando los roles de "adivino" y "participante", verificando empíricamente la infalibilidad del método: sumar el primer número de las tarjetas donde el número secreto está presente.

El núcleo de la actividad es el desafío de la ingeniería inversa. Se plantea a los grupos la pregunta: ¿cuál es el sustento matemático y lógico que permite que esto funcione? Los estudiantes deben analizar la distribución de los números en las tarjetas, identificar patrones y formular hipótesis. El docente puede guiar preguntando:

- Pregunta de Patrones: Miren los números en cada tarjeta. ¿Qué tienen en común? ¿Por qué el número 31 está en todas las tarjetas? ¿Y el 1 solo en una?

- Pregunta Clave: Observen el primer número de cada tarjeta (1, 2, 4, 8, 16). ¿Qué tipo de secuencia matemática es esta? ¿Cómo se relaciona con el sistema binario?
- Pregunta Lógica: ¿Qué representa realmente una respuesta "Sí"? Si una tarjeta representa una potencia de 2, ¿qué significa que tu número esté en ella?

La investigación debe llevarlos a la conclusión de que el truco se basa en el sistema de numeración binario. Se formaliza esta idea explicando que cada tarjeta representa una posición o "bit" en un número binario de 5 dígitos (desde 2^0 hasta 2^4). Una respuesta afirmativa ("sí") equivale a un '1' en esa posición, mientras que un "no" equivale a un '0'. La suma de los primeros números de las tarjetas es, en esencia, la conversión del número de binario a decimal.

Una vez comprendido el sistema de 5 bits ($2^5 = 32$ combinaciones posibles, para los números 0-31), se plantea el desafío de la expansión. Los estudiantes deben deducir qué se necesitaría para ampliar el juego hasta el número 63. Esto requiere que comprendan que necesitan un sexto bit, correspondiente a la potencia $2^5 = 32$. La tarea culmina cuando los estudiantes diseñan y crean su propia sexta tarjeta, calculando y escribiendo en ella todos los números entre 32 y 63 que la requieren en su descomposición binaria.

Para consolidar este conocimiento, se les pregunta a los estudiantes, ¿Se podrá realizar el truco de magia con números desde el 1 hasta el 63? ¿Qué se necesita para lograrlo? Se espera que los estudiantes puedan realizar conjeturas en voz alta, y recoger esta información para brindar un ayuda a los grupos y termina solicitando que se separen por grupos de 4 estudiantes y cada uno realizará sus propias tarjetas para que el truco pueda ser aplicado hasta 63 números.

Se le entregará a cada grupo hojas de oficio para que puedan utilizarlas de borrador y escribir ahí las “tarjetas mágicas”, además se les realizará las siguientes preguntas de apoyo:

- ¿Cuántas tarjetas se deben utilizar?
- ¿El 63 en que tarjetas debe estar?
- ¿El 47 en que tarjetas debe estar?

Una vez que tengan listas las tarjetas mágicas, se les entrega 6 trozos de block, hojas de oficio, etc. Donde puedan crear sus tarjetas mágicas definitivas, las que incluso pueden ser plastificadas con posterioridad.

Se le entregará a cada grupo hojas de oficio para que puedan utilizarlas de borrador y escribir ahí las "tarjetas mágicas", además se les realizará las siguientes preguntas de apoyo:

- ¿Cuántas tarjetas se deben utilizar?
- ¿El 63 en que tarjetas debe estar?
- ¿El 47 en que tarjetas debe estar?

Una vez que tengan listas las tarjetas mágicas, se les entrega 6 trozos de block, hojas de oficio, etc. Donde puedan crear sus tarjetas mágicas definitivas, las que incluso pueden ser plastificadas con posterioridad.

Anexo:

1	9	17	25
3	11	19	27
5	13	21	29
7	15	23	31

2	10	18	26
3	11	19	27
6	14	22	30
7	15	23	31

4	12	20	28
5	13	21	29
6	14	22	30
7	15	23	31

8	12	24	28
9	13	25	29
10	14	26	30
11	15	27	31

8	12	24	28
9	13	25	29
10	14	26	30
11	15	27	31

Ilustración 20. Tarjetas binario-mágicas del 1 al 31

Fuente: Adaptación de Cartillas de festival de matemática.

C.3. Actividad Conectada: Programando un Adivino Binario

En esta actividad, los estudiantes, trabajando en parejas, digitalizan el truco de magia con números en la plataforma Scratch. Se les proporciona un proyecto de plantilla que incluye los recursos visuales necesarios: cinco fondos de escenario, cada uno mostrando una de las tarjetas del juego 0-31, y un botón de "Siguiente".

El primer paso es que los estudiantes diseñen en papel el algoritmo que el programa debe seguir. Deben planificar el uso de una variable, por ejemplo, llamada NumeroAdivinado, que se inicializará en 0. Por cada tarjeta, el programa debe preguntar al usuario si su número está presente. Si la respuesta es afirmativa, el valor correspondiente a esa tarjeta (1, 2, 4, 8 o 16) debe sumarse a la variable NumeroAdivinado. Al hacer clic en el botón "Siguiente", el fondo debe cambiar a la siguiente tarjeta.

Para facilitar la implementación, se proporcionan plantillas de código o bloques pre-estructurados. Este andamiaje permite que los estudiantes se concentren en la lógica principal: la creación de la variable, la estructura condicional (si... entonces...) para verificar la respuesta del usuario, y la operación matemática de suma sobre la variable. Al finalizar las cinco tarjetas, el programa debe mostrar como resultado final el valor total acumulado en la variable NumeroAdivinado.

Antes de comenzar, recuerda que:

Todo número se puede escribir como suma de potencias de 2.

Ejemplo:

$$21 = 16 + 4 + 1 \rightarrow 2^4 + 2^2 + 2^0$$

El programa usa cinco tablas o tarjetas, cada una asociada a una potencia de 2:

$$Tabla\ 1 \rightarrow valor\ 1$$

$$Tabla\ 2 \rightarrow valor\ 2$$

$$Tabla\ 3 \rightarrow valor\ 4$$

$$Tabla\ 4 \rightarrow valor\ 8$$

$$Tabla\ 5 \rightarrow valor\ 16$$

Cada vez que respondas "Sí", el programa sumará ese valor al número adivinado.

Así, si tu número está en las tarjetas $1, 3\ y\ 5$, entonces $1 + 4 + 16 = 21$

1. Preparar el entorno
    a. Abre Scratch y crea un nuevo proyecto.
    b. Borra el gato o cámbiale la apariencia por un "mago" o personaje a elección.
    c. Carga los fondos (Fondo Teatro 1 y 2, Tablas 1 a 5).
    d. Cada tabla mostrará un conjunto distinto de números del 1 al 31.
2. Crear las variables
    a. Crea una variable llamada: "Número adivinado"
    b. Esta almacenará el valor total que el programa va calculando según tus respuestas.
3. Programar la primera tarjeta.
    a. Mostrar [Fondo Teatro 1]
    b. Cuando el fondo cambie a la primera tabla, por ejemplo, se hace la primera pregunta:

cuando el fondo cambie a [Fondo Tabla 1];

preguntar [¿Ves el día de tu nacimiento entre los números de la tabla? (Sí o No)] y esperar

- sí <respuesta = [Sí]> entonces
    - cambiar (o sumar) [Número adivinado v] en (1)
- cambiar fondo a [Fondo Tabla 2]

Lo mismo para cada una de los Fondo Tablas.

Si el número está en la tabla 1, se suma 1.

Luego, el fondo cambia automáticamente a la siguiente tabla.

4. Mostrar el resultado final

Cuando llegues al último fondo, el mago revela el número.

Nota: El "mago" no adivina realmente: calcula.

Cada respuesta "Sí" equivale a un 1 binario, y cada "No" a un 0 binario.

Por ejemplo, si el número pensado fue 21:

Tarjeta	Valor	¿Está tu número?	Bit
1	1	Sí	1
2	2	No	0
3	4	Sí	1
4	8	No	0
5	16	Sí	1

$$Binario: 10101 \rightarrow Decimal: 21$$

Así, el programa reconstruye el número usando las potencias de 2:

$$16 + 4 + 1 = 21$$

Nota:

Se puede aumentar la complejidad de la actividad al escalar su programa para

adivinar números más grandes, obligándolos a comprender el principio del sistema binario en lugar de solo seguir instrucciones.

El proceso consiste en que primero analicen el número de tarjetas qué se necesita para ampliar el rango (como una sexta tarjeta con valor 32) y luego implementen esos cambios en su código de Scratch.

A través de esta tarea, los estudiantes descubren la relación exponencial entre el número de bits (tarjetas) y la cantidad de números que se pueden representar.

Materiales: Computadoras con acceso a Scratch. Proyecto de plantilla de Scratch con los fondos de las tarjetas y botones.

Anexo:

Drive: https://drive.google.com/

Ilustración 21. Preparación del Entorno

Fuente: Creación Propia.

Ilustración 22. Preparación del truco

Fuente: Creación Propia.

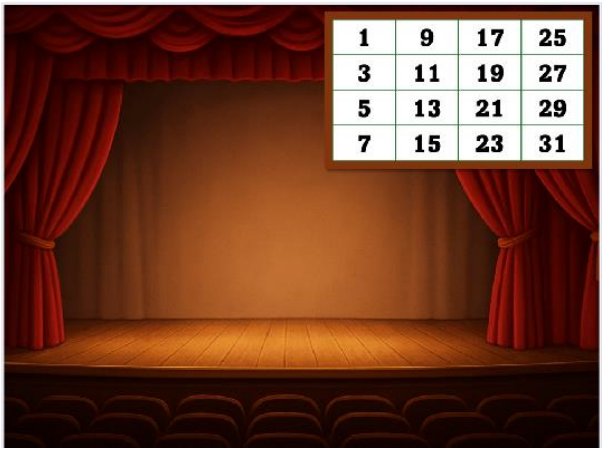


Ilustración 23. Fondo Tabla 1

Fuente: Creación Propia.

D. Detección de errores a través de Paridad

<table>
<tr><td colspan="2">Nombre: Detección de errores a través de Paridad</td></tr>
<tr><td colspan="2">Fuente:

CS Unplugged. (s.f.). Parity Magic (Junior). https://www.csunplugged.org/

Rosquete, D. H., Martínez, A. A., y Perozo, F. J. (2006). Codificación y Decodificación Eficiente Utilizando Códigos Hamming. Trabajo presentado en la XXII Conferencia Latinoamericana de Informática (CLEI 2006). https://clei.org/
NASA (s.f.). Deep Space Network. https://www.nasa.gov/</td></tr>
<tr><td colspan="2">Objetivo de la Actividad: Aplicar el concepto de paridad para la detección y corrección de errores en la transmisión de datos, mediante actividades desconectadas con tarjetas y la programación del algoritmo de Hamming (7,4) en bloques y Python.</td></tr>
<tr><td rowspan="2">Objetivos de Aprendizaje sugeridos:</td><td>Pensamiento Computacional OA 1: Aplicar conceptos de Ciencias de la Computación –abstracción, organización lógica de datos, análisis de soluciones alternativas y generalización– al crear el código de una solución computacional.</td></tr>
<tr><td>Pensamiento Computacional OA 3: Desarrollar y programar algoritmos para ejecutar procedimientos matemáticos, realizar cálculos y obtener términos definidos por una regla o patrón.</td></tr>
<tr><td>Habilidad de Resolver Problemas</td><td>OA a. Construir y evaluar estrategias de manera colaborativa al resolver problemas no rutinarios.

OA b. Resolver problemas que impliquen variar</td></tr>
</table>

	algunos parámetros en el modelo utilizado y observar cómo eso influye en los resultados obtenidos.
Habilidades del Pensamiento Computacional	Evaluar

D.1. Primera Actividad Desconectada: Magia con Paridad

Protegiendo los Secretos del Universo

El Telescopio Espacial James Webb (JWST) es el observatorio espacial más potente jamás construido. Orbitando a 1.5 millones de kilómetros de la Tierra, captura imágenes de una claridad asombrosa de las primeras galaxias del universo. Cada fotografía es una obra maestra científica irremplazable, producto de una inversión de 10 mil millones de dólares. El problema es que estos datos preciosos deben viajar a través del hostil entorno del espacio profundo, donde la señal de radio es bombardeada por rayos cósmicos y ruido solar. Estas interferencias pueden, aleatoriamente, voltear un bit de los datos, cambiando un 1 por un 0. Un solo bit erróneo podría corromper una imagen o falsear un descubrimiento.

La solución es la fiabilidad proactiva. Antes de que el JWST transmita sus datos, sus computadoras a bordo les aplican Códigos de Corrección de Errores. Estos códigos añaden bits de información extra, llamados bits de paridad, calculados matemáticamente a partir de los datos originales. En la Tierra, las gigantescas antenas de la Red del Espacio Profundo de la NASA capturan la débil señal. Las computadoras en el centro de control analizan los datos recibidos y sus bits de paridad. Si un bit se ha corrompido, el patrón de los bits de paridad actúa como una huella digital que no solo revela que hay un error, sino que identifica

exactamente qué bit es el incorrecto, permitiendo a los ingenieros reconstruirlo con una pureza perfecta.

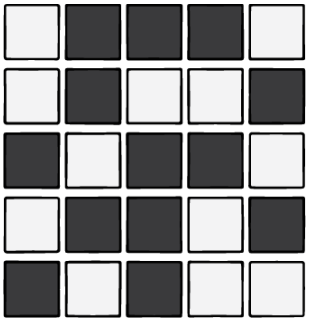
Referencia: NASA (s.f.). *Deep Space Network*. https://www.nasa.gov/

En esta actividad, los estudiantes se convertirán en el sistema de comunicaciones del James Webb. Se enfrentarán al mismo problema que los ingenieros de la NASA resuelven cada día: proteger datos irremplazables de la corrupción del espacio, utilizando el poder de la paridad.

Se introduce el concepto de detección de errores de forma tangible, a través de un juego con tarjetas (o papeles) que representan bits. El objetivo es que los y las estudiantes comprendan cómo la paridad puede ayudar a detectar y localizar un error en una cuadrícula de datos.

Instrucciones:

1. Cada grupo arma una cuadricula de 5x5 dispuestas al azar.



2. Un participante del grupo de ayuda se integrará a un grupo y añadirá una sexta fila y columna considerando que exista un número par de tarjetas negras en filas y columnas.

3. Como grupo, tomarán la decisión de voltear una tarjeta, sin que el integrante del grupo de ayuda la vea.
4. El integrante del grupo de ayuda buscará visualmente la fila y columna que tenga un número impar de tarjetas negras, hasta encontrar la tarjeta volteada.
5. Se dará un tiempo de reflexión, para que el grupo conjeture una posible
6. Solución para encontrar el "truco". Considerando las siguientes preguntas:
    a. ¿Cómo son las filas y columnas con respecto al lado visible de la tarjeta?
    b. ¿Ves algún patrón en la posición de las tarjetas, con respecto al lado visible?
7. Se repite la actividad hasta que el grupo encuentre la solución.

Una vez que los estudiantes hayan descubierto la solución, se pueden plantear las siguientes preguntas de reflexión.

¿Qué permitió descubrir cuál tarjeta cambió?

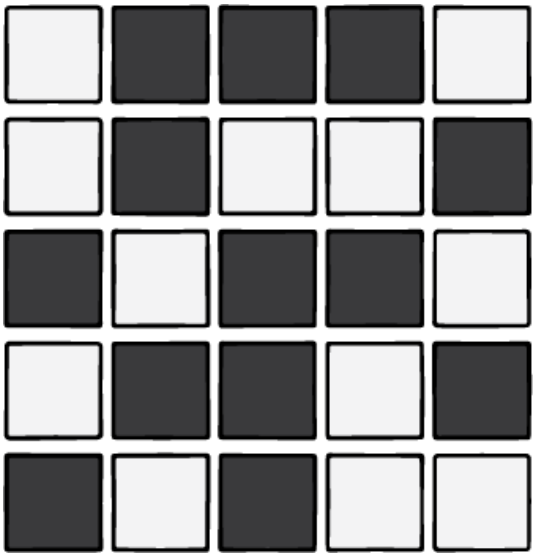
¿Por qué bastó con revisar las filas y columnas para hallar el error?

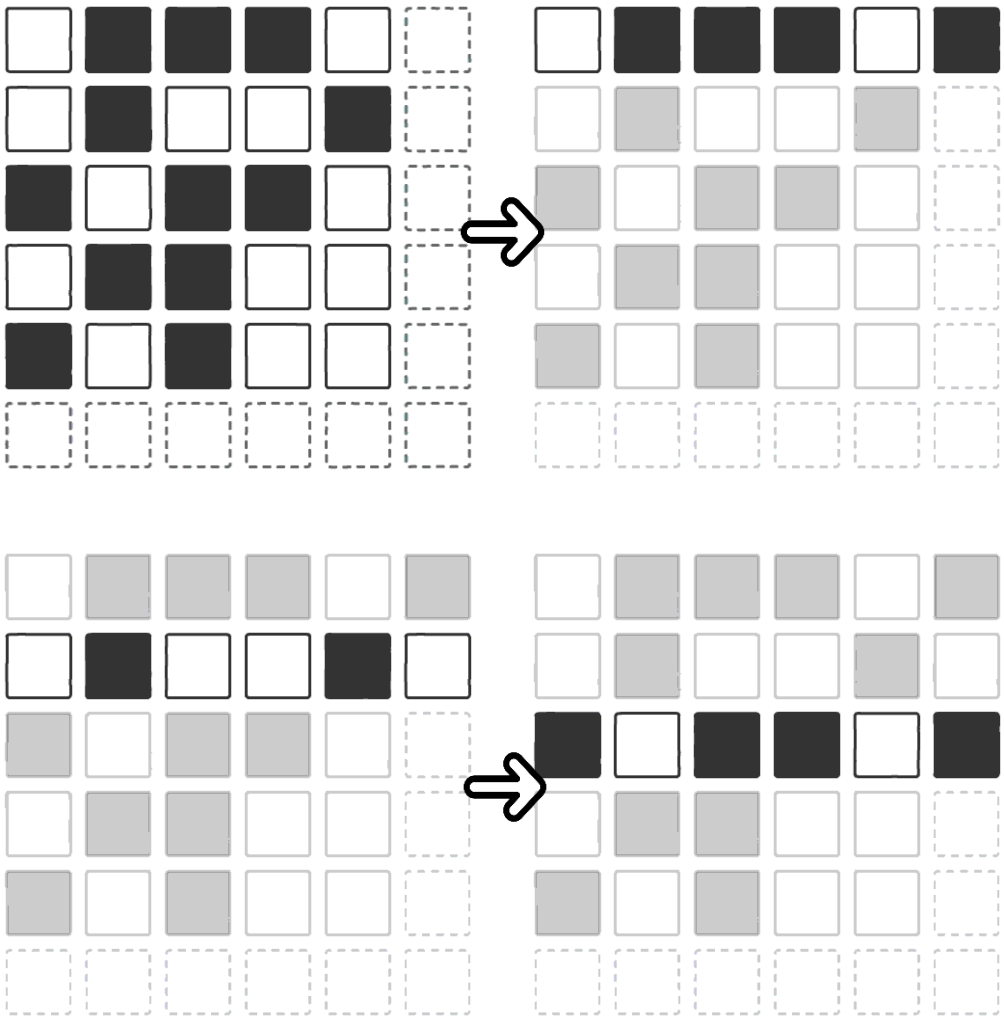
¿Qué patrones o reglas matemáticas se pueden observar?

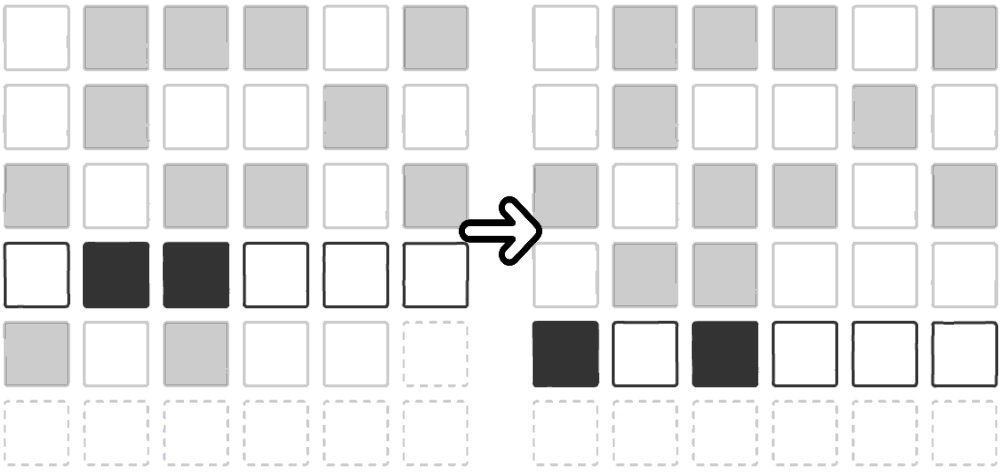
Si en lugar de una cuadrícula tuviéramos una **secuencia de bits en una sola línea**, ¿cómo podríamos aplicar una idea similar?

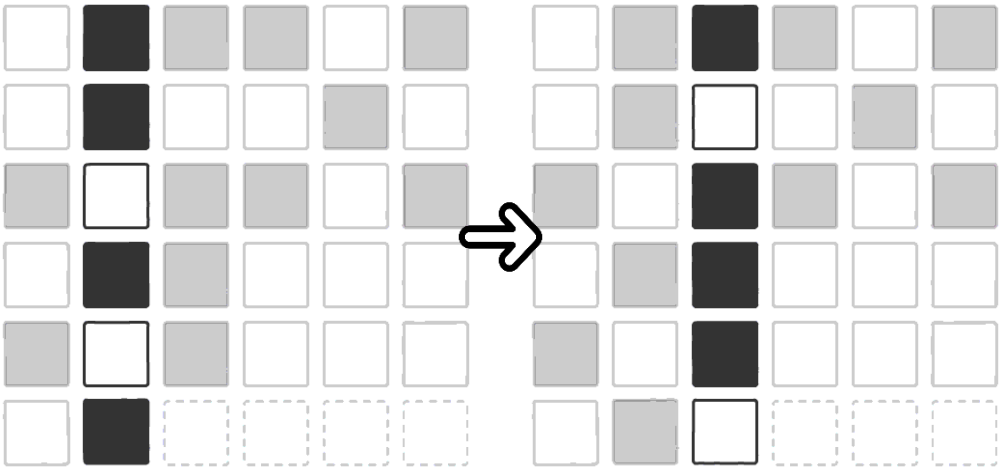
¿Qué crees que hace un **computador** puede detectar que un dato se dañó durante su transmisión?

Anexo:









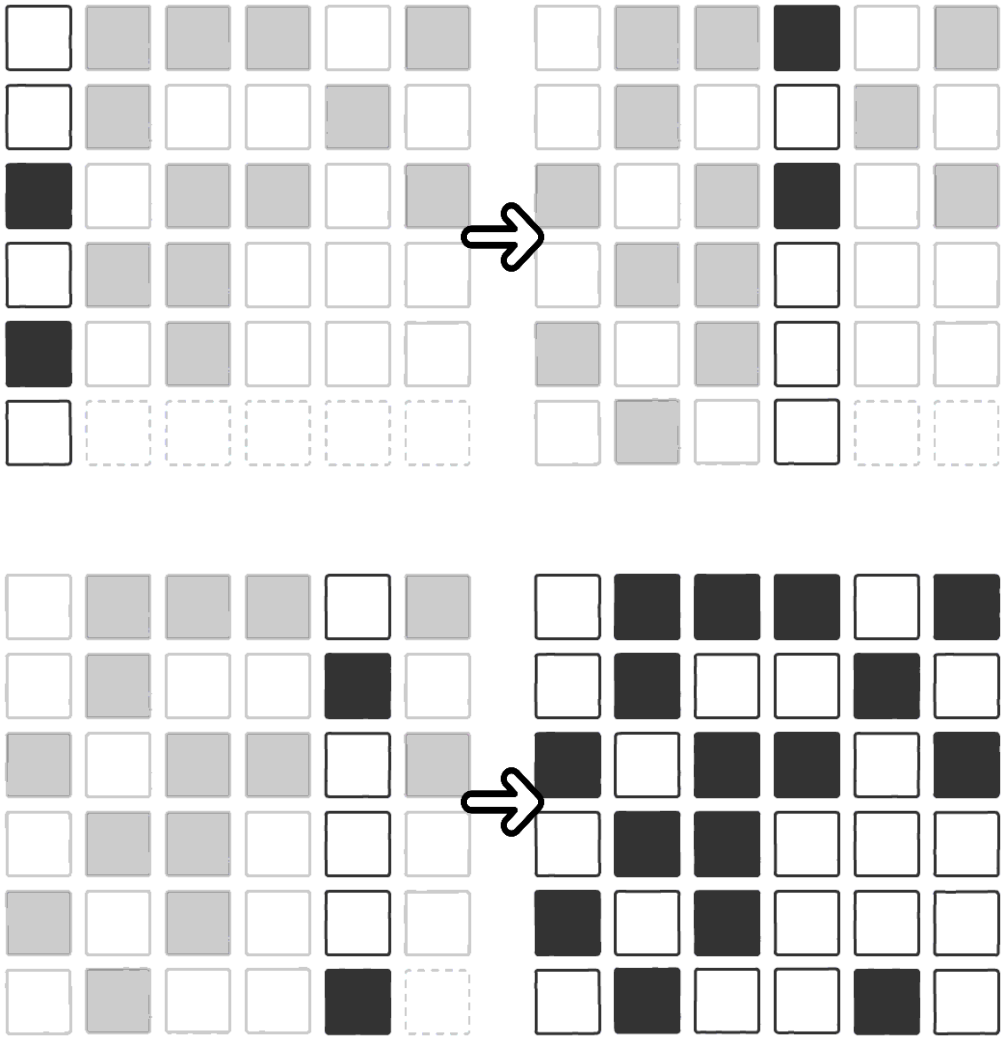


Ilustración 24. Proceso de paridad en filas y columnas.

Fuente: Tomado de "Parity Magic (Junior)", por CS Unplugged, s.f.

Explicación del Algoritmo de Hamming

La siguiente explicación del algoritmo de Hamming está adaptada de la metodología descrita por Rosquete et al. (2006). La que podemos separar en la primera etapa de codificación y la segunda de decodificación.

Codificación.

En esta etapa se asignan los *bits* de paridad necesarios para controlar todos los *bits* de datos del mensaje. El número de *bits* de paridad requeridos para d *bits* de datos está determinado por la desigualdad: $2^p \geq d + p + 1$, donde p es el número de *bits* de paridad y d es el número de *bits* de datos. Además la longitud de la palabra de código se determina sumando el número de bits de datos más el número de bits de paridad.

En nuestro caso, se tiene 4 *bits* de datos, por tanto para que se cumpla la desigualdad deben haber 3 *bits* de paridad. Así la longitud de la palabra de código a generar es 7.

Se debe tener en cuenta que, en la palabra de código a generar, las posiciones en que son potencias de dos (1, 2, 4) se utilizan los *bits* de paridad, mientras que el resto de las posiciones (3,5,6,7) son utilizadas como *bits* de datos.

La posición de cada *bit* de paridad en la palabra de código que se genera en la etapa de codificación determina la secuencia de los *bits* de datos que verifica en la misma, tal como se presenta en la siguiente tabla.

Posiciones	2^2	2^1	2^0
	P_3	P_2	P_1
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Por lo tanto:

- P_1 ocupa la posición 1 y comprueba las posiciones 7, 5, 3 y 1
- P_2 ocupa la posición 2 y comprueba las posiciones 7, 6, 3 y 2
- P_3 ocupa la posición 4 y comprueba las posiciones 7, 6, 5 y 4

Para obtener el valor de los *bits* de paridad se deben sumar la cantidad de unos que hubo en las posiciones comprobadas; si esta suma es impar, el valor del bit de paridad evaluado es 1, en caso contrario el valor es 0.

Ejemplo: Consideremos el código 1010, donde nuestros bits de datos

$$D_4 = 1,\ D_3 = 0, D_2 = 1, D_1 = 0.$$

por lo que la palabra de código a generar es:

$D_4\ D_3\ D_2\ P_3\ D_1\ P_2\ P_1 \Leftrightarrow\ 1\ 0\ 1\ P_3\ 0\ P_2\ P_1$

Luego.

Para determinar P_1:

las posiciones 3, 5 y 7 son 0, 1 y 1 respectivamente, la suma da 2, que es par, por lo tanto $P_1 = 0$

Para determinar P_2:

las posiciones 3, 6 y 7 son 0, 0 y 1 respectivamente, la suma da 1, que es par, por lo tanto $P_2 = 1$

Para determinar P_3:

las posiciones 5, 6 y 7 son 1, 0 y 1 respectivamente, la suma da 2, que es par, por lo tanto $P_3 = 0$

Obteniendo la palabra de código generado:

$$1\ 0\ 1\ 0\ 0\ 1\ 0$$

Decodificación

Si uno comprueba la palabra de código generada se obtiene lo siguiente:

Para C_1:

la posición 1, 3, 5 y 7 es, 1, 1, 0 y 0 respectivamente, la suma da 2, que es par y por lo tanto $C_1 = 0$.

Para C_2:

la posición 2, 3, 6 y 7 es, 0, 1, 1 y 0 respectivamente, la suma da 2, que es par y por lo tanto $C_2 = 0$.

Para C_3:

- la posición 4, 5, 6 y 7 es, 1, 0, 1 y 0 respectivamente, la suma da 2, que es par y por lo tanto $C_3 = 0$.

Supongamos que ocurre una falla en la transmisión de la palabra código generado y en vez de tener $1\ 0\ 1\ 0\ 0\ 1\ 0$ tenemos $1\ 1\ 1\ 0\ 0\ 1\ 0$.

Se comprueba la nueva palabra código:

Para C_1:

- la posición 1, 3, 5 y 7 es, 0, 0, 1 y 1 respectivamente, la suma da 2, que es par y por lo tanto $C_1 = 0$.

Para C_2:

- la posición 2, 3, 6 y 7 es, 1, 0, 1 y 1 respectivamente, la suma da 3, que es impar y por lo tanto $C_2 = 1$.

Para C_3:

la posición 4, 5, 6 y 7 es, 0, 1, 1 y 1 respectivamente, la suma da 3, que es impar y por lo tanto $C_3 = 1$.

Y para determinar la posición del *bit* con error, se calcula mediante:

Error_posición = $4 \cdot C_3 + 2 \cdot C_2 + C_1$

Error_posición = $4 \cdot 1 + 2 \cdot 1 + 0 = 6$

Por lo tanto, el error se encuentra en la posición 6.

Así, reemplazamos el bit de la posición 6 por su valor contrario, pasando de 1 1 1 0 0 1 0 a 1 0 1 0 0 1 0.

D.2. Segunda Actividad Desconectada: Algoritmo de Hamming.

Se realiza una simulación del algoritmo de hamming utilizando naipes que representaran bits, donde la cara tomará el valor de "1" y el reverso el valor de "0". El objetivo es que los y las estudiantes comprendan cómo la paridad puede ayudar a detectar y localizar un error en una lista de datos, usando el Algoritmo de Hamming (7,4).

Para realizar esta simulación es necesario comprender la serie de pasos del algoritmo, que serán presentados a continuación.

1. Se debe formar un código con los 4 bits de datos, como por ejemplo:

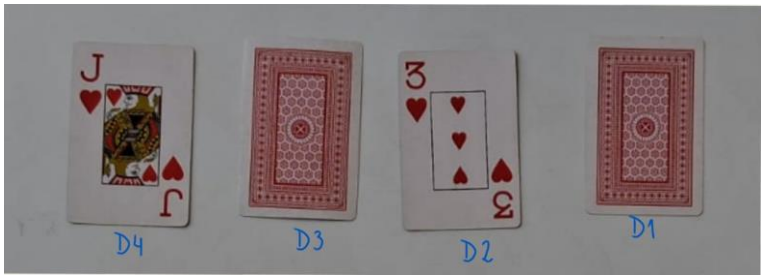


Ilustración 25. ejemplo de bits de datos

Fuente: Creación Propia

Aquí, se genera el código 1 0 1 0.

2. Comenzamos con el procedimiento de generar un código de hamming de 7 bits. Posicionando los bits de datos, en sus posiciones correspondientes y determinando los valores de $P1, P2$ y $P3$, utilizando:
    a. $P1$ ocupa la posición 1 y comprueba las posiciones 7, 5, 3 y 1
    b. $P2$ ocupa la posición 2 y comprueba las posiciones 7, 6, 3 y 2
    c. $P3$ ocupa la posición 4 y comprueba las posiciones 7, 6, 5 y 4

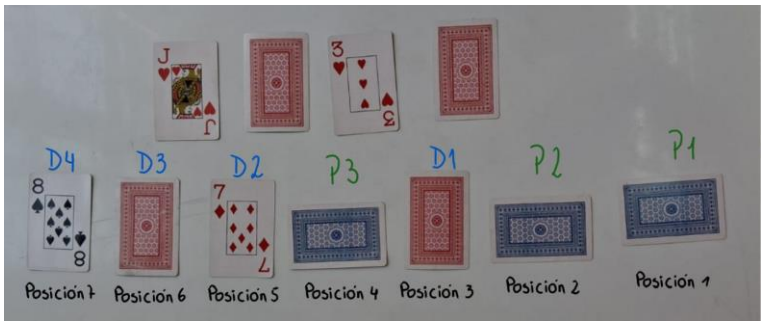


Ilustración 26. Posiciones de un código de haming de 7 bits

Fuente: Creación Propia

3. Para determinar el valor de $P1$ que verifica las posiciones 7, 5 y 3.

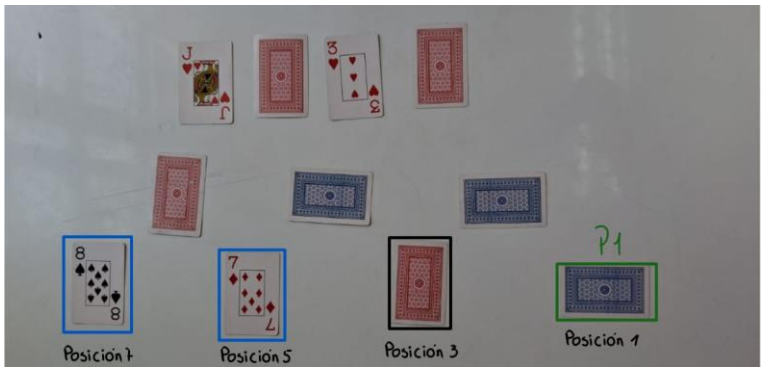


Ilustración 27. Procedimiento para determinar valor de P1

Fuente: Creación Propia

Como hay dos caras, $P1$ toma el valor de 0. Es decir, consideramos el reverso.

4. Continuamos con el valor de $P2$ que verifica las posiciones 7,6 y 3.

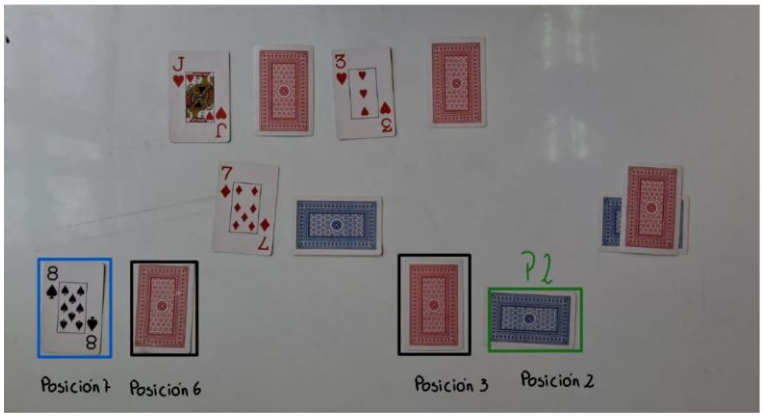


Ilustración 28. Procedimiento para determinar valor de P2

Fuente: Creación Propia

Como hay una cara, $P2$ toma el valor de 1. Es decir, consideramos la cara.

5. Análogamente con el valor de $P3$ que verifica las posiciones 7, 6 y 5.

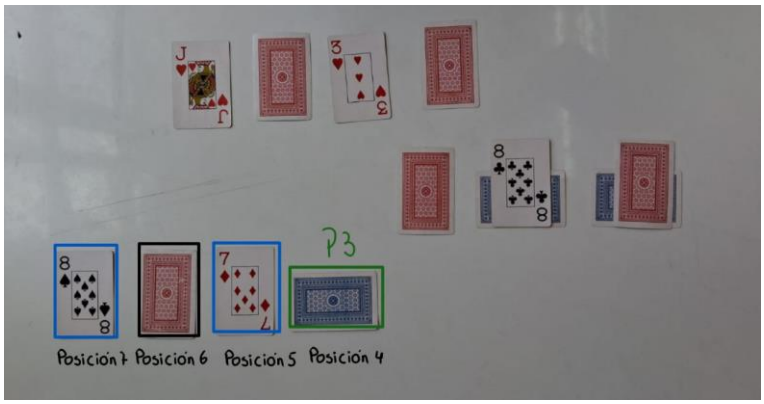


Ilustración 29. Procedimiento para determinar valor de P3

Fuente: Creación Propia

Como hay dos caras, $P3$ toma el valor de 0. Es decir, consideramos el reverso.

6. Obteniendo la siguiente palabra de código generado.

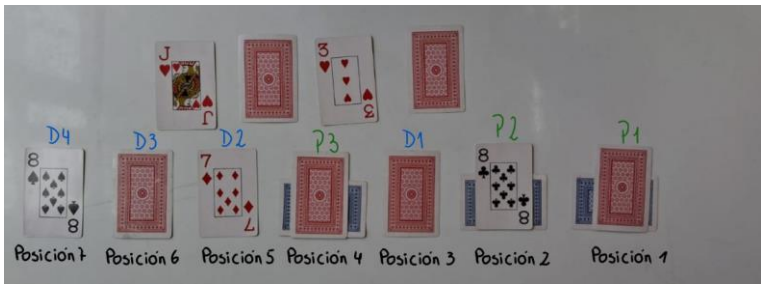


Ilustración 30. Palabra código generado

Fuente: Creación Propia

7. Representaremos la falla en la transmisión de la palabra código generado, en una nueva fila, modificando un bit.

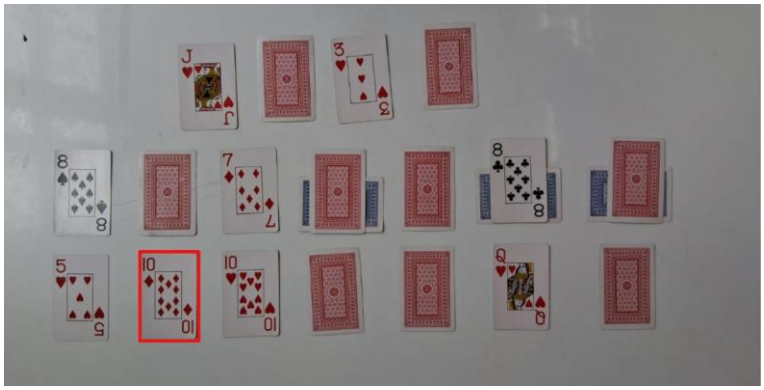


Ilustración 31. Representación de falla en un bit

Fuente: Creación Propia

8. Ahora, comenzaremos a con el proceso de codificación, para aquello, tendremos tres variables, $C3, C2$ y $C1$. Los cuales se comprobarán de la siguiente manera:
    a. Para $C1$ consideramos las posiciones 7, 5, 3 y 1.

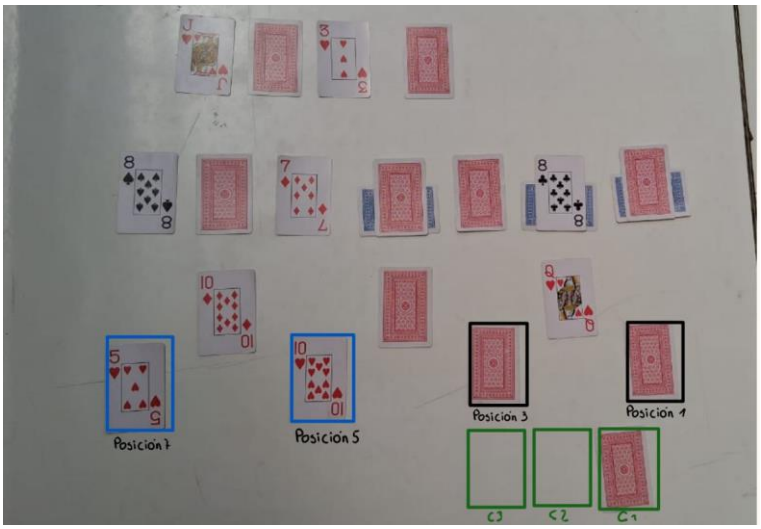


Ilustración 32. Procedimiento para determinar valor de C1

Fuente: Creación Propia

Como hay dos caras, $C1$ toma el valor de 0. Es decir, consideramos el reverso.

b. Para $C2$ consideramos las posiciones 7, 6, 3 y 2.

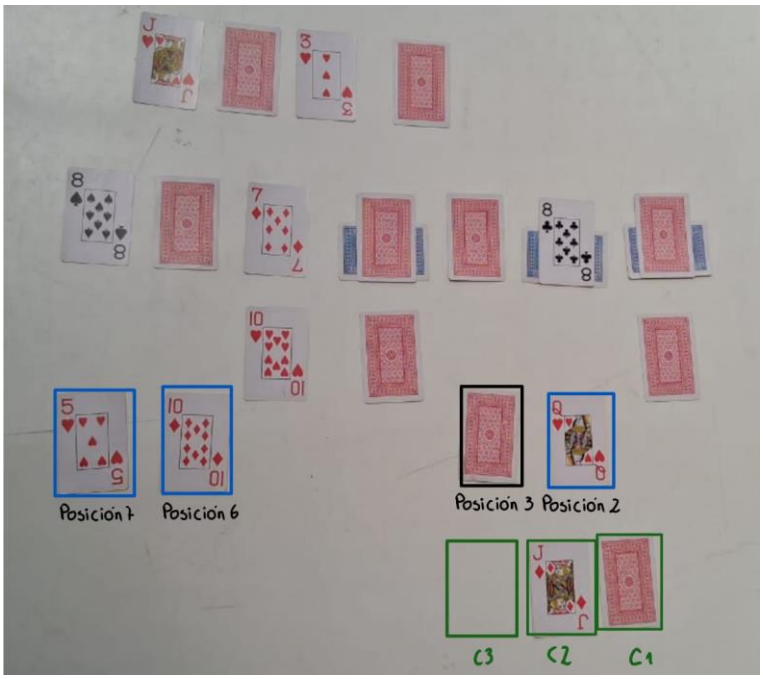


Ilustración 33. Procedimiento para determinar valor de C2

Fuente: Creación Propia

Como hay tres caras, $C2$ toma el valor de 1. Es decir, consideramos la cara.

c. Para $C3$ consideramos las posiciones 7, 6, 5 y 4.

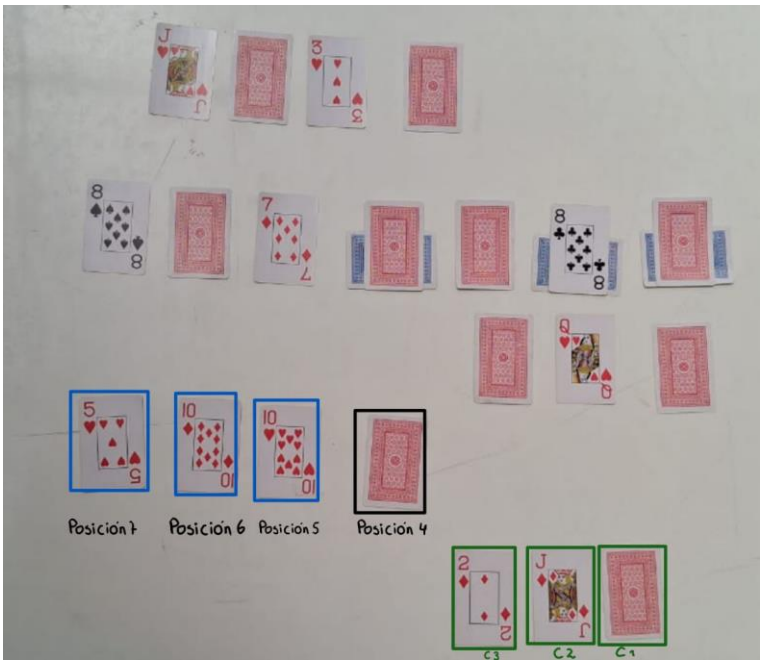


Ilustración 34. Procedimiento para determinar valor de C3

Fuente: Creación Propia

Como hay tres caras, $C3$ toma el valor de 1. Es decir, consideramos la cara.

9. Ahora, los valores de $C3, C2$ y $C1$. Nos indican en el sistema binario la posición del bit errado.

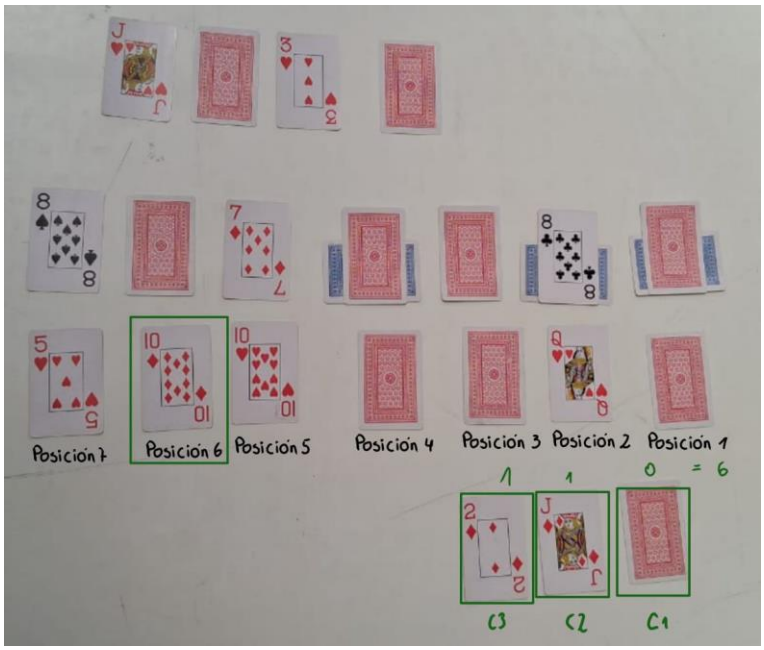


Ilustración 35. Posición del bit errado

Fuente: Creación Propia.

En este caso, el bit con error se encuentra en la posición 6.

Instrucciones:

1. Separar el grupo curso entre 4 hasta 8 equipos. A los cuales se les entregara 10 tarjetas en las que se distingan ambos lados, un código con los 4 bits de datos y una hoja de registro. 4 de las tarjetas serán para representar los bits de datos, 3 para representar los bits de paridad y 3 para que puedan comprobar la posición del bit con error.
2. Posteriormente, deberán codificar la palabra de código generado, utilizando el algoritmo de haming. Registrando la palabra de código generado en su hoja.
3. Luego, se juntarán entre dos equipos e intercambiarán su palabra de código generado con un error en uno de sus bits.
4. El equipo contrario deberá encontrar la posición del bit con error.

5. Una vez terminado, se pueden rotar con distintos equipos para continuar determinando el error en un uno de sus bits.

D.3. Primera Actividad Conectada: Algoritmo de Hamming en Scratch

Creando un código protegido.

En esta actividad se agregarán bits de paridad a la información para detectar y corregir errores en los datos que se transmiten o almacenan. Estos bits de paridad funcionan como una especie de "control de calidad" que acompaña al mensaje original. Cuando los datos se envían, el sistema puede verificar si alguno de los bits cambió debido a una interferencia, un fallo de memoria o un error de comunicación.

Mediante el uso del algoritmo de Hamming, se descubre cómo, a partir de solo 4 bits de información real, se generan 3 bits adicionales llamados bits de paridad, que permiten identificar y corregir automáticamente un error en el mensaje recibido.

Durante la actividad se usará el módulo 2 (mod 2), una operación matemática muy usada en informática. Con ella se podrá comprobar si la cantidad de "1" en un conjunto de bits es par (resultado 0) o impar (resultado 1). Este principio de la paridad es la base del algoritmo, ya que permite determinar si los datos fueron alterados durante su transmisión.

Al finalizar, programa mostrará un código protegido de 7 bits, que representa la versión "segura" de tu mensaje original. Este código puede ser enviado o almacenado con mayor fiabilidad, ya que incluye información suficiente para detectar y corregir un error de un solo bit, asegurando la integridad del mensaje, tal como lo hacen los sistemas reales en discos duros, memorias o redes digitales.

Pasos para Crear el Programa en Scratch

- Configurar el entorno:
- Crear dos listas vacías desde el conjunto de bloques “Variables”:
    - **Código**, donde se almacenarán los 4 bits ingresados.
    - **Código Protegido,** donde se formará el código final de 7 bits.
- Al iniciar con el bloque “al hacer clic en la bandera verde”, ubica al personaje y limpia ambas listas usando:
- eliminar todos de Código
- eliminar todos de Código Protegido
- Solicitar el código al usuario:
- Usar el bloque “preguntar” para solicitar al usuario un código de 4 bits.
- Con un ciclo repetir 4, extraer cada carácter de la respuesta y añadirlo a la lista Código.
- Posicionar los bits en la lista “Código Protegido”:

El código protegido debe tener 7 posiciones. Donde Las posiciones 1, 2 y 4 son para los bits de paridad (P1, P2 y P3), y las posiciones 3, 5, 6 y 7 son para los bits de datos (D1, D2, D3, D4).

Bit	**D4**	**D3**	**D2**	**P3**	**D1**	**P2**	**P1**
Posición	7	6	5	4	3	2	1

En Scratch se trasladarán los elementos de Código hasta Código Protegido:

- insertar (elemento 1 de Código) en 1 de Código Protegido.

Y se insertará un elemento vacío en la ubicación donde irán los bits de paridad.

- insertar () en 4 de Código Protegido.

Nota: Scratch, al manejar listas, ordena sus elementos desde la izquierda (posición 1), mientras que en el sistema binario y en el algoritmo de Hamming, los bits se numeran de derecha a izquierda (desde el bit menos significativo). Esto genera una diferencia importante:

- En Hamming, la posición 1 es el bit de menor peso (a la derecha).
- En Scratch, la posición 1 está al comienzo (a la izquierda).

Por esta razón, al construir el algoritmo, es necesario invertir el orden de los bits o ajustar las posiciones manualmente para que las comprobaciones de paridad se realicen de forma correcta.
Si no se hace este ajuste, las combinaciones de bits que verifica cada P1, P2 y P3 no corresponderían a las posiciones que realmente deben controlar.

Bit	**D4**	**D3**	**D2**	**P3**	**D1**	**P2**	**P1**
Posición	7	6	5	4	3	2	1
Lista Scratch	1	2	3	4	5	6	7

- Calcular los bits de paridad (P1, P2, P3):

Cada bit de paridad controla un conjunto específico de posiciones y se determina con las posiciones:

- P1 (posición 1) se verifica con las posiciones 3, 5 y 7.
- P2 (posición 2) se verifica con las posiciones 3, 6 y 7.
- P3 (posición 4) se verifica con las posiciones 5, 6 y 7.

En Scratch, se calcula usando el bloque de módulo (mod), con la operación “mod 2”.

Por ejemplo, para P1:

- sí ((elemento 1 + elemento 3 + elemento 5) mod 2 = 0)
- entonces reemplazar elemento 1 con 0
- si no, reemplazar elemento 1 con 1

Esto se repite de forma similar para P2 y P3.

Nota: El operador "mod" (módulo) calcula el resto de una división. Cuando usamos mod 2, estamos calculando el resto al dividir por 2, que nos indica si un número es par o impar:

- Si un número tiene resto 0 al dividir por 2 → es par.
- Si tiene resto 1 → es impar.

Como la paridad binaria se basa precisamente en saber si hay un número par o impar de bits "1", el uso de mod 2 es la forma más directa de comprobarlo matemáticamente.

- Mostrar el código final:

Se, usa el bloque decir para enunciar el Código Protegido.

- Corrector de código.

En esta actividad se verifica y corregir un código aplicando las mismas reglas de paridad que se usaron para crearlo. Esto te permitirá entender cómo los computadores detectan si un mensaje fue alterado y cómo pueden encontrar exactamente en qué posición ocurrió el error.

El programa analizará los bits del código recibido, sumará los valores según las posiciones de verificación y, usando el módulo 2 (mod 2), determinará si en cada grupo de comprobación hay una cantidad par o impar de "1". Con esa

información, el sistema podrá identificar la posición del bit que falló y corregirlo automáticamente, mostrando el mensaje original reparado.

A través de un programa en Scratch, se simularán cómo una computadora:

- Revisa bits de un mensaje recibido,
    - Identifica si hay un error y en qué posición,
    - Corrige automáticamente el bit equivocado.
- Preparar el entorno Scratch
    - Crea los siguientes datos:
    - Lista: Código Para Corregir
    - Variables: Bit Fallado, Contador
    - Al iniciar con el bloque “al hacer clic en la bandera verde”, ubica al personaje y limpia la lista usando:
    - eliminar todos de [Código Para Corregir]
- Ingreso del código
    - Pide al usuario ingresar el código binario: preguntar y esperar.
    - Inicializa variables:
    - poner [Bit Fallado] a 0
    - poner [Contador] a 1
- Crea un bucle con repetir para llenar la lista con los 7 bits, extraer cada carácter de la respuesta y añadirlo a la lista Código para Corregir.

Así, la lista queda con los bits del mensaje, donde el elemento 1 corresponde al primer dígito ingresado (izquierda).

Corrección de Bits:

El código Hamming toma 4 bits de información y agrega 3 bits de paridad (P1, P2, P3) que sirven como detectores de error. Estos bits se ubican en posiciones que son potencias de 2 (1, 2 y 4) y verifican ciertos grupos de posiciones:

Bit de paridad	Posiciones que controla	Valor que suma si hay error
P_1 (posición 1)	1, 3, 5, 7	1
P_2 (posición 2)	2, 3, 6, 7	2
P_3 (posición 4)	4, 5, 6, 7	4

Cada vez que uno de estos controles detecta una paridad incorrecta (es decir, un número impar de 1s), el algoritmo suma su valor correspondiente (1, 2 o 4) a la variable Bit Fallado.

En Scratch, se usa la operación (mod) dentro de los operadores de números para cada verificación:

Comprobación P_1: controla posiciones 1, 3, 5, 7

- sí < ((elemento 1 + elemento 3 + elemento 5 + elemento 7) mod 2) = 0> entonces
    - cambiar (o sumar) a [Bit Fallado] (0)
- Si no
    - cambiar (0 sumar) a [Bit Fallado] (+1)

En el contexto de Hamming:

- Si la suma de los bits en un grupo da un número impar de 1, significa que la paridad esperada no se cumple.

Entonces, el bloque si ((suma mod 2) = 1) indica que hubo un error en ese conjunto.

- Determinar si existe error

Después de todas las comprobaciones:

- sí < (Bit Fallado = 0)> entonces [No hay error detectado]

Por ende se finaliza con:

- detener [todos]

Si existe error, se procede con la Corrección automática. Por ejemplo cuando Bit Fallado es distinto de 0, el programa debe invertir el valor del bit en esa posición.

Esa suma, en Bit Fallado no es arbitraria: el resultado final indica la posición exacta del bit erróneo.

Por ejemplo:

- Si P_1 y P_2 detectan error, entonces 1 + 2 = 3, es decir, error en la posición 3.
- Si solo P_3 detecta error, entonces error en la posición 4.

6. Mostrar el código final corregido.

Anexo.

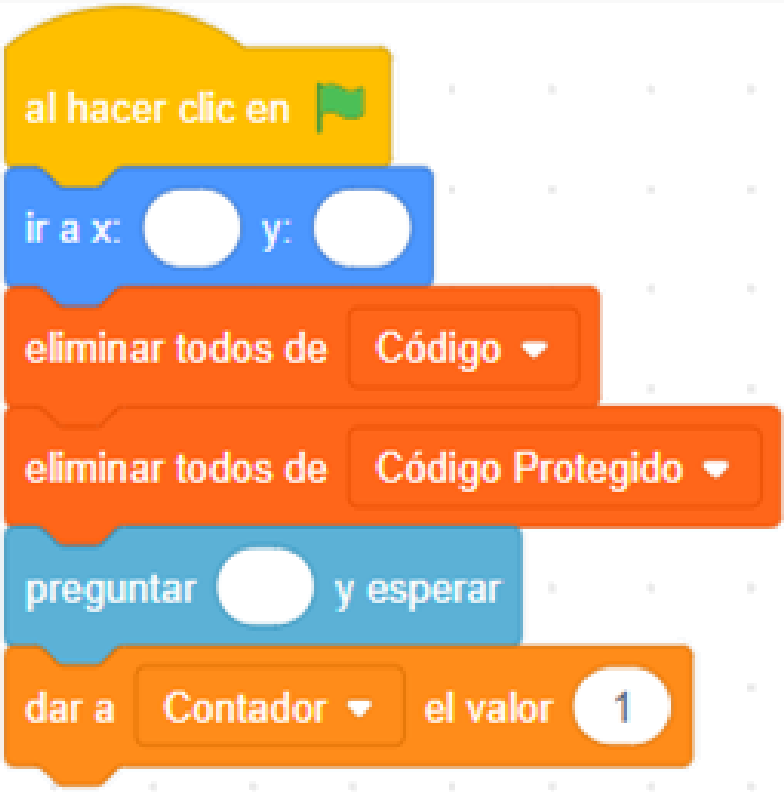


Ilustración 36 Preparación entorno actividad I

Fuente: Creación Propia.

Ilustración 37: añadir elementos de una respuesta a la lista (actividad I)

Fuente: Creación Propia.

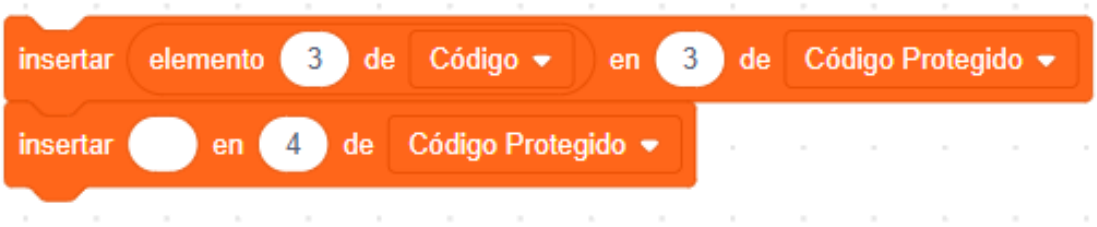


Ilustración 38: Generación de lista de 7 bits (actividad I)

Fuente: Creación Propia.

Ilustración 39: Creación de P_1 (actividad I)

Fuente: Creación Propia.

Ilustración 40: Preparación entorno actividad I.

Fuente: Creación Propia.

Ilustración 41: Comprobación P_1

Fuente: Creación Propia.

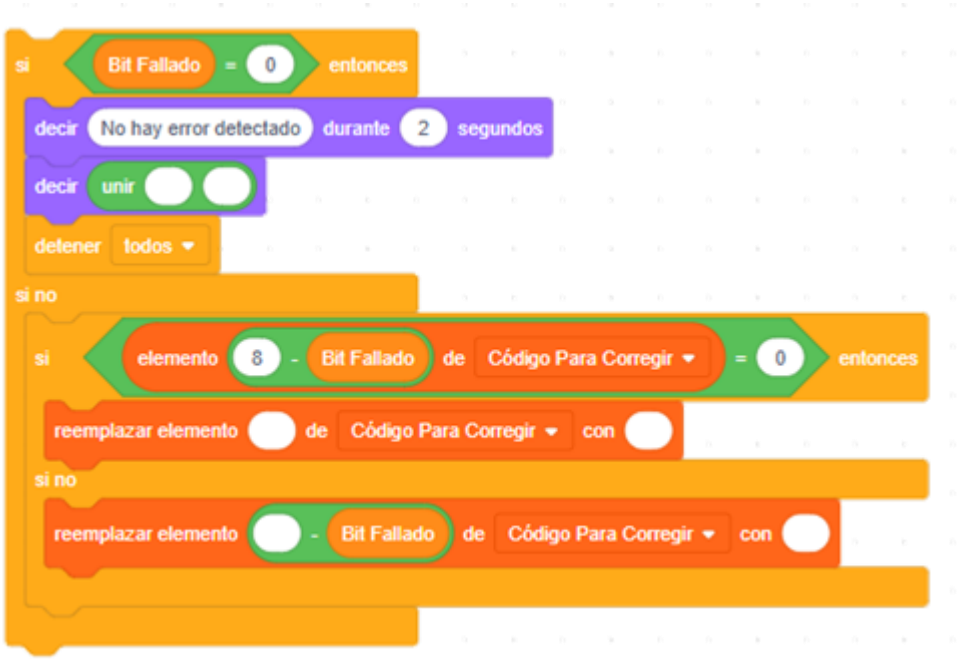


Ilustración 42: Error y corrección.

Fuente: Creación Propia.

D.4. Segunda Actividad Conectada: Algoritmo de Hamming en Python.

Los estudiantes aplican los principios de paridad y organización lógica de datos para programar en Python un sistema capaz de detectar y corregir errores de un bit usando el Algoritmo de Hamming (7,4).

Instrucciones:

1. formar grupos de 4 estudiantes, a los cuales se les entregará un código incompleto.
2. Cada grupo deberá completar la función “generar_hamming(data_bits)” que genera la palabra de código, utilizando el algoritmo de Hamming.
3. Cada grupo deberá completar la función “verificar_hamming(bits)” que verifica la palabra de código generado, detectando la posición del bit con error y lo corrige.

Para la función “generar_hamming(data_bits)” es importante considerar:

- La entrada de la función será una lista de 4 elementos, qué corresponde a los bits de datos.
- Deberá determinar los valores de los bits de paridad (P_1, P_2 y P_3)
- La salida de la función será una lista de 7 elementos, qué corresponde a la palabra código generado.

Para la función “verificar_hamming(bits)” es importante considerar:

- La entrada de la función será una lista de 7 elementos, que corresponde a una nueva entrada posiblemente con un bit errado.
- Calcula los bits de comprobación y la posición del posible error.
- En caso de no existir error, debe indicar que "No se detectan errores."
- En caso de existir error, debe indicar la posición del error y corregir la palabra de código generado. En este ítem hay que considerar que Python cuenta las posiciones de izquierda a derecha y comenzando desde la posición cero, a diferencia del sistema binario, por lo que es necesario realizar dos ajustes (Anexo ilustración 47).
- La salida de la función será una lista de 7 elementos, qué corresponde a la palabra código generado correcta.

Anexo.

```
[ ]
    # ----------------------------------------
    # ACTIVIDAD: Algoritmo DE HAMMING (7,4)
    # ----------------------------------------
    # Objetivo:
    # Implementar un sistema de detección y corrección de errores de un bit
    # usando el Algoritmo de Hamming (7,4).
    # Los estudiantes deben completar las funciones:
    #   - generar_hamming(data_bits)
    #   - verificar_hamming(bits)
    # ----------------------------------------

    # Función para ingresar los bits de datos
    def ingresar_bits():
        while True:
            datos = input("Ingrese 4 bits de datos (por ejemplo 1010): ")
            if len(datos) == 4 and all(b in "01" for b in datos):
                return [int(b) for b in datos]
            else:
                print("Entrada no válida. Debe ingresar exactamente 4 bits (0 o 1).")

    # ----------------------------------------------------
    # Función que genera el código de Hamming (a completar)
    # ----------------------------------------------------
    def generar_hamming(data_bits):

        # Calcular bits de paridad (paridad par)

        return []

    # ----------------------------------------------------
    # Función que verifica y corrige el código (a completar)
    # ----------------------------------------------------
    def verificar_hamming(bits):

        # Calcular bits de comprobación

        # Calcular posición del error (C3C2C1 en binario)

        #Determinar si se detectaron errores.

        return

    # ----------------------------------------------------
    # Programa principal
    # ----------------------------------------------------
    print("=== CÓDIGO DE HAMMING (7,4) ===")

    # Paso 1: Ingresar los 4 bits de datos
    data = ingresar_bits()

    # Paso 2: Generar el código de Hamming
    codigo = generar_hamming(data)
    print("Código Hamming generado (7 bits):", codigo)

    # Paso 3: Simular un error
    print("\nAhora ingresa un segundo código de 7 bits (con posible error).")
    while True:
        entrada2 = input("Segunda entrada (7 bits, ej. 1110010): ")
        if len(entrada2) == 7 and all(bit in "01" for bit in entrada2):
            recibido = [int(b) for b in entrada2]
            break
        print("Entrada inválida. Deben ser 7 bits (solo 0 o 1).")

    # Paso 4: Verificar y corregir el código
    resultado = verificar_hamming(recibido)
    print("Código corregido:", resultado)
```

Ilustración 43. Código incompleto

Fuente: Creación Propia.

```python
# ----------------------------------------------------
# Función que genera el código de Hamming (a completar)
# ----------------------------------------------------
def generar_hamming(data_bits):
    D4, D3, D2, D1 = data_bits
    # Calcular bits de paridad (paridad par)
    # determinar primer bits de paridad (P1)
    if D4 + D2 + D1 == 0 or D4 + D2 + D1 == 2:
      P1=0
    if D4 + D2 + D1 == 1 or D4 + D2 + D1 == 3:
      P1=1
    # determinar segundo bits de paridad (P2)
    if D4 + D3 + D1 == 0 or D4 + D3 + D1 == 2:
      P2=0
    if D4 + D3 + D1 == 1 or D4 + D3 + D1 == 3:
      P2=1
    #determinar tercer bits de paridad (P3)
    if D4 + D3 + D2 == 0 or D4 + D3 + D2 == 2:
      P3=0
    if D4 + D3 + D2 == 1 or D4 + D3 + D2 == 3:
      P3=1

    return [D4, D3, D2, P3, D1, P2, P1]
```

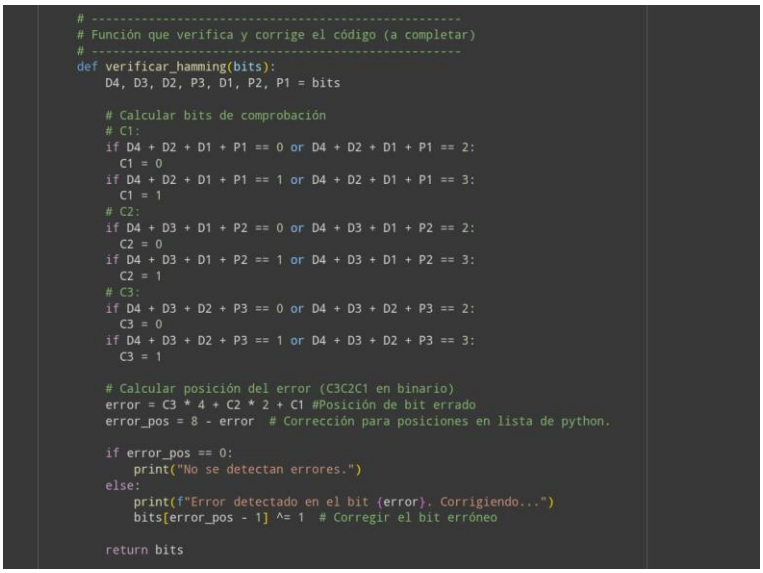
Ilustración 44. Primera solución para función "generar_hamming(data_bits)"

Fuente: Creación Propia

```python
# ----------------------------------------------------
# Función que genera el código de Hamming (a completar)
# ----------------------------------------------------
def generar_hamming(data_bits):
    D4, D3, D2, D1 = data_bits
    print(data_bits)
    # Calcular bits de paridad (paridad par)
    P1 = D4 ^ D2 ^ D1  # controla posiciones 1,3,5,7
    P2 = D4 ^ D3 ^ D1  # controla posiciones 2,3,6,7
    P3 = D4 ^ D3 ^ D2  # controla posiciones 4,5,6,7

    return [D4, D3, D2, P3, D1, P2, P1]
```

Ilustración 45. Segunda solución para función "generar_hamming(data_bits)"

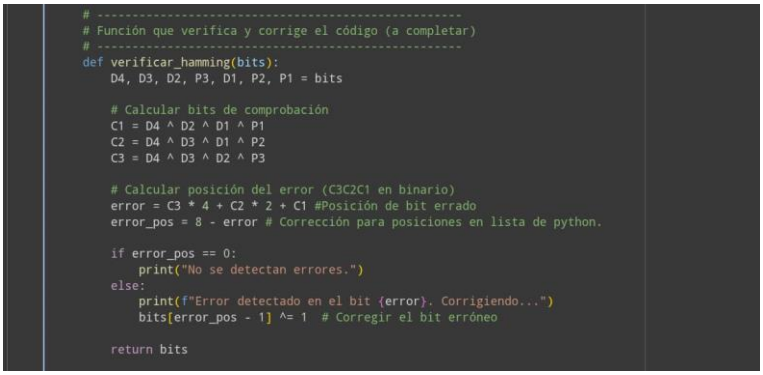
Fuente: Creación Propia.



```python
# ------------------------------------------------------
# Función que verifica y corrige el código (a completar)
# ------------------------------------------------------
def verificar_hamming(bits):
    D4, D3, D2, P3, D1, P2, P1 = bits

    # Calcular bits de comprobación
    # C1:
    if D4 + D2 + D1 + P1 == 0 or D4 + D2 + D1 + P1 == 2:
      C1 = 0
    if D4 + D2 + D1 + P1 == 1 or D4 + D2 + D1 + P1 == 3:
      C1 = 1
    # C2:
    if D4 + D3 + D1 + P2 == 0 or D4 + D3 + D1 + P2 == 2:
      C2 = 0
    if D4 + D3 + D1 + P2 == 1 or D4 + D3 + D1 + P2 == 3:
      C2 = 1
    # C3:
    if D4 + D3 + D2 + P3 == 0 or D4 + D3 + D2 + P3 == 2:
      C3 = 0
    if D4 + D3 + D2 + P3 == 1 or D4 + D3 + D2 + P3 == 3:
      C3 = 1

    # Calcular posición del error (C3C2C1 en binario)
    error = C3 * 4 + C2 * 2 + C1 #Posición de bit errado
    error_pos = 8 - error  # Corrección para posiciones en lista de python.

    if error_pos == 0:
        print("No se detectan errores.")
    else:
        print(f"Error detectado en el bit {error}. Corrigiendo...")
        bits[error_pos - 1] ^= 1  # Corregir el bit erróneo

    return bits
```

Ilustración 46. Primera solución para función "verificar_hamming(bits)"

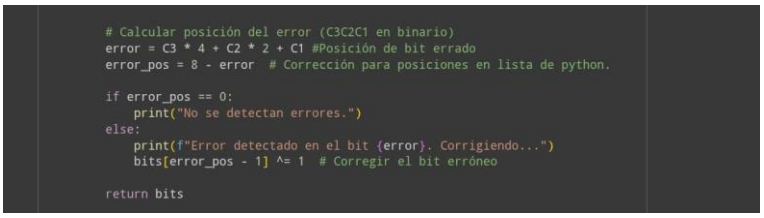
Fuente: Creación Propia.



```python
# ------------------------------------------------------
# Función que verifica y corrige el código (a completar)
# ------------------------------------------------------
def verificar_hamming(bits):
    D4, D3, D2, P3, D1, P2, P1 = bits

    # Calcular bits de comprobación
    C1 = D4 ^ D2 ^ D1 ^ P1
    C2 = D4 ^ D3 ^ D1 ^ P2
    C3 = D4 ^ D3 ^ D2 ^ P3

    # Calcular posición del error (C3C2C1 en binario)
    error = C3 * 4 + C2 * 2 + C1 #Posición de bit errado
    error_pos = 8 - error # Corrección para posiciones en lista de python.

    if error_pos == 0:
        print("No se detectan errores.")
    else:
        print(f"Error detectado en el bit {error}. Corrigiendo...")
        bits[error_pos - 1] ^= 1  # Corregir el bit erróneo

    return bits
```

Ilustración 47: Segunda solución para función "verificar_hamming(bits)"

Fuente: Creación Propia.



```
# Calcular posición del error (C3C2C1 en binario)
error = C3 * 4 + C2 * 2 + C1 #Posición de bit errado
error_pos = 8 - error  # Corrección para posiciones en lista de python.

if error_pos == 0:
    print("No se detectan errores.")
else:
    print(f"Error detectado en el bit {error}. Corrigiendo...")
    bits[error_pos - 1] ^= 1  # Corregir el bit erróneo

return bits
```

Ilustración 48: Correciones respecto a la posición del bit errado.

Fuente: Creación Propia.

- $error = C3 * 4 + C2 * 2 + C3$, Calcula la posición del bit errado, de derecha a izquierda y Python, considera las posiciones de la lista de izquierda a derecha.
- $error\_pos = 8 - error$, Es una corrección para poder contar los bits de derecha a izquierda.
- En $bits[error\_pos - 1] \wedge= 1$: $bits[error\_pos - 1]$ busca corregir el elemento de en la $error\_pos$ de la lista "$bits$", no obstante, el "-1" se agrega para considerar posiciones entre 1 y 7, en vez de 0 a 6.

E. Máquina de Galton

<table>
<tr><td colspan="2">Nombre: Maquina de Galton</td></tr>
<tr><td colspan="2">Fuente:
Castellano Sánchez, D., García Carmona, M. Á., Prados Martínez, A., Moya Sánchez, A., Rodríguez Corona, M., y Aguilar Pérez, F. J. (2025). Las matemáticas son la leche: Máquina humana de Galton. Uno. Revista de Didáctica de las Matemáticas, 107, 61-73.
IBM. (2023). ¿Qué es la simulación de Monte Carlo? IBM. https://www.ibm.com/</td></tr>
<tr><td colspan="2">Objetivo de la actividad: Experimentar y describir el comportamiento azaroso en una máquina de Galton humana, calculando y representando frecuencias relativas de los contenedores finales, para luego trasladar estos resultados al entorno digital y comparar cómo el mismo fenómeno probabilístico se manifiesta en la simulación computacional.</td></tr>
<tr><td>Objetivos de Aprendizaje sugeridos:</td><td>MA07 OA 18 Explicar las probabilidades de eventos obtenidos por medio de experimentos de manera manual y/o con software educativo:
Estimándolas de manera intuitiva.
Utilizando frecuencias relativas.
Relacionándolas con razones, fracciones o porcentaje.
MA1M OA15 Mostrar que comprenden el concepto de azar:
Experimentando con la tabla de Galton y con paseos aleatorios sencillos de manera manual y/o con software educativo.
Realizando análisis estadísticos, empezando por frecuencias relativas.</td></tr>
</table>

	• Utilizando probabilidades para describir el comportamiento azaroso. • Resolviendo problemas de la vida diaria y de otras asignaturas.
	MA 4M OA2 Fundamentar decisiones en situaciones de incerteza, a partir del análisis crítico de datos estadísticos y con base en los modelos binomial y normal. Pensamiento Computacional OA 3. Desarrollar y programar algoritmos para ejecutar procedimientos matemáticos, realizar cálculos y obtener términos definidos por una regla o patrón.
Habilidad de resolver problemas	MA07 OAH a Resolver problemas utilizando estrategias tales como: • Destacar la información dada. • Usar un proceso de ensayo y error sistemático. • Aplicar procesos reversibles. • Descartar información irrelevante. • Usar problemas similares. MA1M OAH a Resolver problemas utilizando estrategias como las siguientes: • Simplificar el problema y estimar el resultado. • Descomponer el problema en subproblemas más sencillos.

	• Buscar patrones. • Usar herramientas computacionales. MA-Pensamiento ComputacionalPR-3y4-OAH-b Resolver problemas que impliquen variar algunos parámetros en el modelo utilizado y observar cómo eso influye en los resultados obtenidos.
Habilidades del Pensamiento Computacional	Pensamiento Algorítmico.

E.1 Actividad desconectada: Maquina de Galton Humana

Simulando el Futuro con el Método de Montecarlo

Cuando nos enfrentamos a un sistema con muchas variables aleatorias, es imposible predecir un único resultado futuro. En su lugar, los ordenadores ejecutan un modelo miles o millones de veces, utilizando la aleatoriedad en cada paso, para generar no una respuesta, sino un mapa de los futuros más probables.

Referencia: IBM. (2023). ¿Qué es la simulación de Monte Carlo? IBM. https://www.ibm.com/

En esta primera actividad los estudiantes experimentarán con una versión manual de la máquina de Galton, utilizando una moneda como generador de azar.

Preparación del Espacio Físico

1. Espacio General: Se necesita un área amplia y despejada (aprox. 12x12 metros), como un gimnasio o patio.
2. Punto de Partida: Marcar un único punto de INICIO en el suelo.
3. Pirámide de Decisiones: Crear 5 filas de "estaciones de decisión" (usando mesas o marcas en el suelo). La estructura debe ser piramidal: la primera fila tiene 1 estación, la segunda 2, y así sucesivamente hasta 5 estaciones en la quinta fila.
4. Rutas: Usar cinta adhesiva para trazar caminos en forma de "V" que conecten cada estación con las dos correspondientes de la fila inferior, creando una red visible.
5. Contenedores Finales: Debajo de la última fila, marcar 6 "contenedores" grandes en el suelo para que los estudiantes se agrupen al final de su recorrido. Etiquetarlos del 0 al 5 para facilitar el conteo.
6. Materiales: Colocar una ficha en cada una de las estaciones de decisión para que los estudiantes la lancen.

La dinámica consiste en que cada estudiante represente una “bolita” que atraviesa una serie de decisiones. En cada nivel, lanzará una ficha: si sale cara avanzará hacia la derecha y si sale sello avanzará hacia la izquierda. Después de 5 lanzamientos, el estudiante llegará a una de las casillas finales, donde se acumularán todos los participantes que siguieron trayectorias similares.

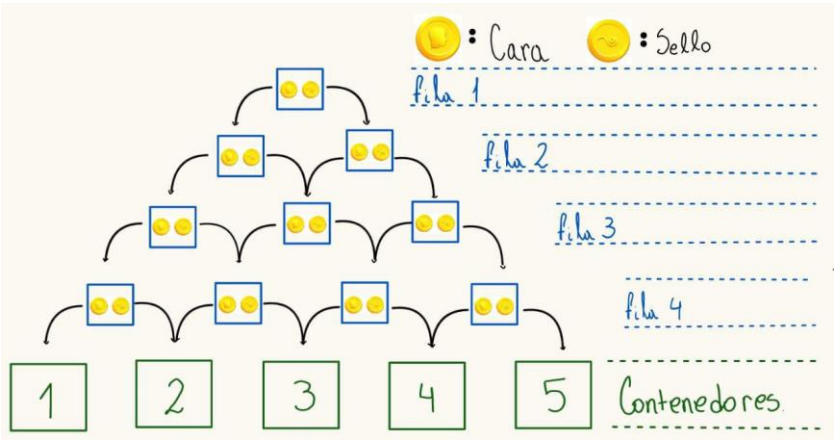


Ilustración 49. Imagen referencial considerando filas y contenedores

Fuente: Creación Propia

En esta actividad, cada estudiante actúa como una simulación computacional individual, replicando manualmente el método de "Montecarlo" que un superordenador de IBM ejecutaría millones de veces por segundo. El objetivo es entender desde dentro cómo funciona este proceso.

El lanzamiento de la moneda introduce el factor clave de riesgo e incertidumbre, un concepto central en el artículo de IBM. Dentro de un escenario de negocio, como el lanzamiento de un videojuego, cada fila representa una fase del proyecto y el resultado de la moneda simboliza un pequeño éxito o contratiempo en esa etapa.

El resultado final no es una predicción única, sino un análisis de probabilidades. La distribución física de los estudiantes en los contenedores crea un mapa visual del rango completo de resultados posibles. Los contenedores centrales muestran el desenlace más probable (un resultado promedio), mientras que los extremos representan los escenarios de baja probabilidad pero alto impacto (un éxito o fracaso rotundo).

Instrucciones:

1. Todo el curso se posicionará al principio de la máquina de Galton, y pasaran uno por uno.
2. En cada mesa habrá una ficha, si la ficha es sello; el estudiante tendrá que tomar la decisión de ir hacia la izquierda. Caso contrario, deberá ir hacia la derecha.
3. Una vez que pase por todas las filas, el estudiante se quedará en el "contenedor" correspondiente, y es ahí cuando pasa el siguiente estudiante por la máquina de Galton. Dependiendo del espacio disponible, los estudiantes de organizan en filas o dejar una marca que represente su pertenencia a cierto contenedor.
4. Una vez que hayan pasado todo el curso. Se realizará un conteo por contenedor y se anotaran los resultados en la pizarra.

Una vez que toda la clase haya participado, se registrarán en la pizarra las frecuencias absolutas de cada contenedor, es decir, cuántos estudiantes terminaron en cada uno. Posteriormente, se calcularán las frecuencias relativas dividiendo cada cantidad por el número total de estudiantes. Con estos datos, los estudiantes podrán comparar los resultados empíricos con lo que esperarían en una situación de azar puro, introduciendo así el concepto de probabilidad.

E. 2 Primera Actividad Conectada: Maquina de Galton Scratch

En esta actividad se simulará digitalmente el tablero de Galton o "máquina de probabilidades", mostrando cómo una bola cae por una estructura triangular de clavijas eligiendo al azar izquierda (0) o derecha (1).

Cada vez que la bola cae y elige entre ir a la izquierda o a la derecha, está realizando una decisión binaria, equivalente a un 0 o un 1 en el sistema numérico binario.

Para representar estas decisiones en el programa, se utiliza la variable “Posición”, que va acumulando los valores obtenidos en cada fila: cuando la bola cae hacia la derecha, se suma 1; cuando cae hacia la izquierda, se suma 0.

Al finalizar su recorrido por las cinco filas de clavijas, el valor total de esta variable indica en qué tubería de llegada (dibujada en el fondo) termina la bola. De esta forma, el número obtenido refleja cuántas veces la bola fue hacia la derecha durante su descenso.

Este proceso no solo modela cómo los computadores realizan operaciones binarias (sumando 0 y 1), sino que también representa un fenómeno de probabilidad combinatoria.

En el recorrido, existen muchas más combinaciones posibles que resultan en valores intermedios (por ejemplo, 2 o 3 “derechas”) que combinaciones que resulten en los extremos (0 o 5 “derechas”).

Nota: Al ejecutar el programa varias veces y soltar muchas bolas, se observa que la mayoría terminan cayendo en las tuberías centrales, generando una distribución en forma de campana. Este patrón refleja cómo, incluso en procesos aleatorios, los resultados tienden a concentrarse alrededor de los valores medios, tal como ocurre en la distribución binomial y en fenómenos naturales o estadísticos reales.

Actividad Scratch:

Para la actividad se incluyen dos elementos base que serán utilizados:

- Fondo con el diseño completo: la pirámide de triángulos naranjos (pivotes) y 6 tuberías o tubos (numerados de 0 a 5) al final de cada recorrido.
- Sprite de la bola (pequeño círculo rojo).

1. Preparar el entorno:

- Desde el conjunto de bloques de Variables se establecen las siguientes:
  - Posición
  - Tubería1, Tubería2, Tubería3, Tubería4, Tubería5, Tubería6
- Número de bolitas (opcional para pedir al usuario cuántas bolas quiere simular)
- En el sprite de la bola, se configurará de esta manera:
  - Con el bloque: al presionar bandera verde

ir a la capa [adelante]

- Se fija cada una de las variables a 0
- Se pregunta al usuario el número de bolitas a trabajar

Esto reinicia los contadores y pide al usuario cuántas bolas se lanzarán.

2. Simulación de la caída

Usa un bucle que se repita tantas veces como bolas ingrese el usuario:

repetir (respuesta)

ir a x: (0) y: (150) (posición inicial centrada sobre la pirámide)

La bola comienza siempre desde arriba, justo sobre la primera fila de triángulos.

3. Caída con decisiones aleatorias

Cada bola baja 5 niveles. En cada uno, elige al azar entre 0 (izquierda) o 1 (derecha).

Para simular eso:

repetir (5)

```
fijar [Probabilidad v] a (elegir número al azar entre (0) y (1))
sí <(Probabilidad) = 1> entonces
  cambiar [Posición v] por (1)
  desplazar en (0.1) segs a x: () y: ( )
si no
  cambiar [Posición v] por (0)
  desplazar en (0.1) segs a x: () y: ()
```

Consejo: ajusta los valores +30, -30, y -30 según el tamaño del fondo para que los movimientos coincidan visualmente con los triángulos del escenario.

4. Llegada a las tuberías

Después de atravesar las 5 filas, la bola baja directamente hasta la base de las tuberías:

```
desplazar en (0.3) segs a x: ( ) y: ()
```

A esta altura la variable Posición (que vale de 0 a 5) define a qué tubo pertenece.

5. Aumentar el contador de la tubería correspondiente

Cada tubo del fondo tiene un contador asociado. Usa los condicionales para actualizar el valor:

```
sí <(Posición) = 0> entonces cambiar [Tubería1] por (1)
```

Nota: Los valores de posición comienzan desde 0 hasta 5, mientras que las tuberías de 1 a 6. En el ejemplo la Posición 0 se asocia a la Tubería 1.

De esta forma, cada vez que una bola termina su recorrido, se incrementa el número que aparece sobre el tubo correspondiente (si las variables están visibles en pantalla).

6. Repetir la simulación

Para reiniciar con la siguiente bolita se finaliza el repetir:

Fijar [Posición] a 0

El bucle repetir (respuesta) permite que el proceso completo se repita tantas veces como el usuario indicó.

7. Mostrar resultados

- Activa las variables Tubería1 a Tubería6 en pantalla (menú Variables → mostrar).
- Colócalas sobre las bocas de los tubos del fondo para que actúen como contadores visuales.

Apuntes:

- Cada bola representa un conjunto de 5 decisiones binarias (0 o 1).
- Posición cuenta cuántas veces la bola fue a la derecha → suma de bits.
- Las posiciones centrales son las más probables porque existen más combinaciones que producen esos resultados (ej. 2 derechas, 3 izquierdas).
- Este principio es la base del modelo binomial y de cómo los computadores trabajan con 0 y 1 para representar información y decisiones.

Anexo.

Drive:
https://drive.google.com/drive/folders/1yeh15y7xfwZwyEo782W7FlyOMpzapj_O?usp=drive_link

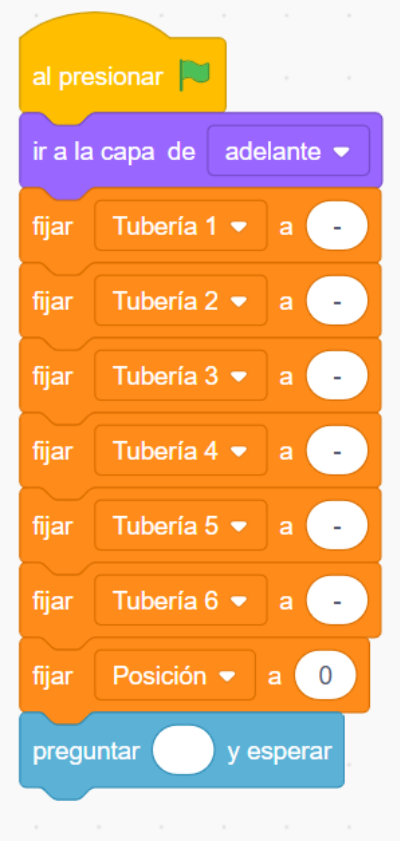


Ilustración 50. Preparación del Entorno

Fuente: Creación Propia

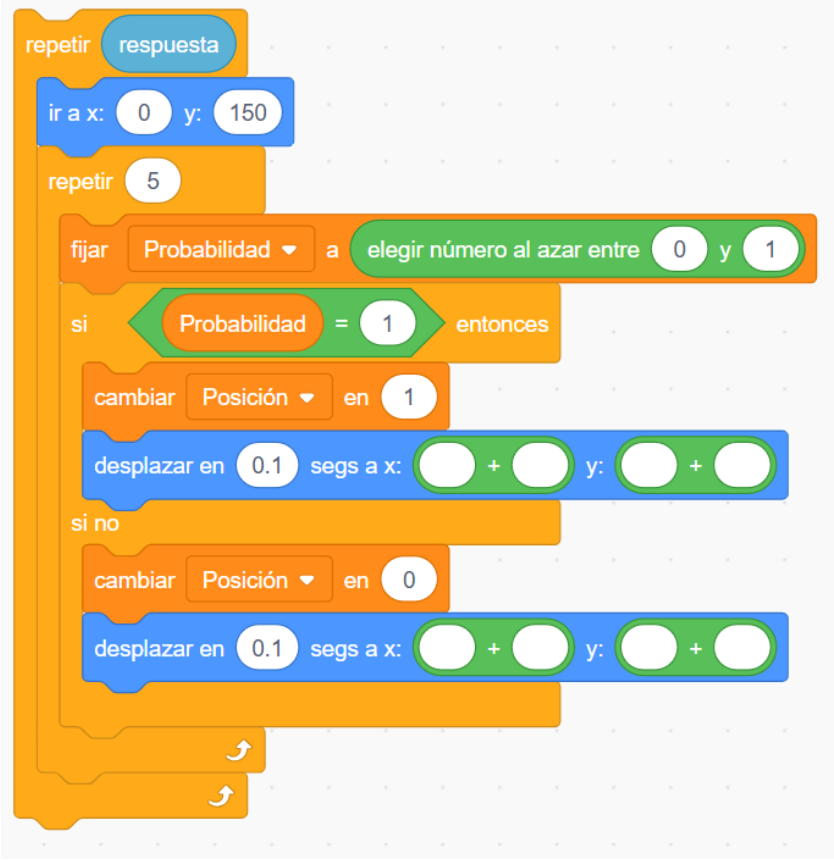


Ilustración 51. Decisiones aleatorias y desplazamiento.

Fuente: Creación Propia

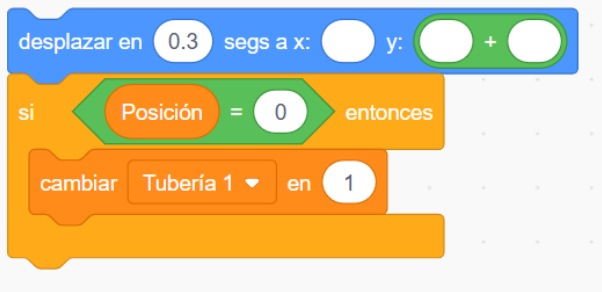


Ilustración 52. Desplazamiento hacia tubería y contador de Tubería 1

Fuente: Creación Propia

Ilustración 53. Finaliza el bucle con reinicio de variable Posición

Fuente: Creación Propia

E.3 Actividad Conectada: Maquina de Galton con Python.

En la segunda actividad, se pretende realizar experimentos con más cantidad de bolitas, con respecto a la versión humana realizada previamente.

Para aquello, se les entregará el siguiente código incompleto:

```
import random
def Maquina_de_galton():
    # 1. Definir parámetros fijos ---
    try:
        numero_filas = int(input("Número de filas: "))
        if numero_filas < 1:
            print("Error: El número de filas debe ser un entero positivo.")
            return
    except ValueError:
        print("Error: Entrada no válida. Por favor, introduce un número entero.")
        return

    try:
        numero_bolitas = int(input("Introduce el número de bolitas a simular: "))
        if numero_bolitas <= 0:
            print("Error: El número de bolitas debe ser un entero positivo.")
            return
    except ValueError:
        print("Error: Entrada no válida. Por favor, introduce un número entero.")
        return


    # 2. ejecución:

    contenedores = [0] * (numero_filas1 + 1)
# ------------------------------------------------
# Secuencialidad a completar
# ------------------------------------------------




#------------------------------------------------
 #resultados:
    for i, cantidad in enumerate(contenedores):
        print(f"Contenedor {i+1}: {cantidad} bolitas")
if __name__ == "__main__":
    Maquina_de_galton()
```

Ilustración 54. Codigo Incompleto

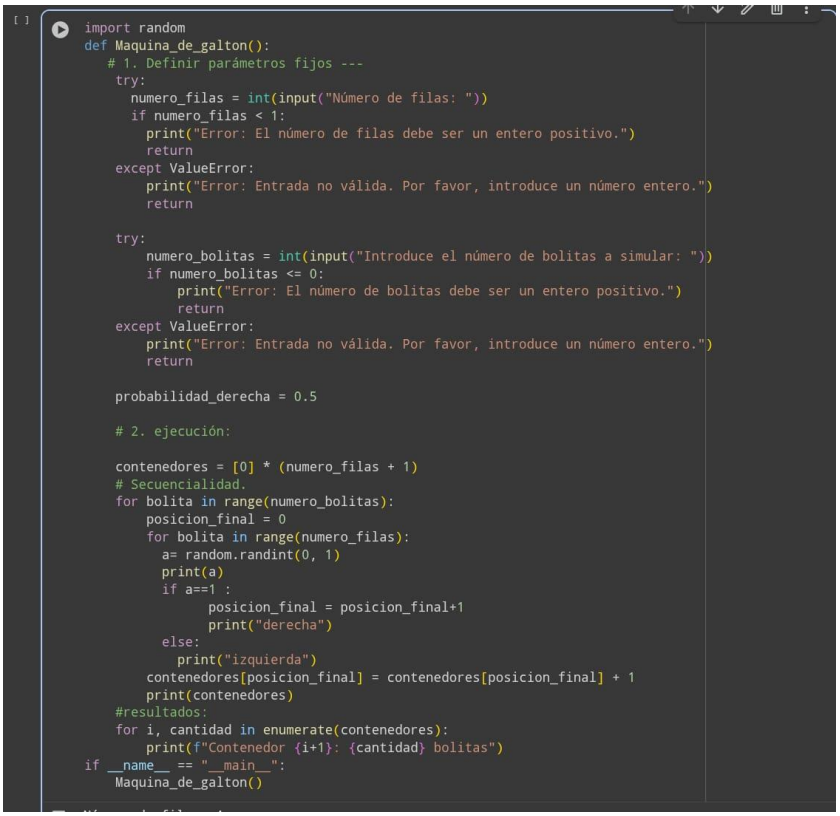
Fuente: Creación Propia

Instrucciones:

1. Completa el código, de tal manera que muestre el recorrido que hace una bolita hasta llegar a un contenedor, considerando 4 niveles.
2. Debes ocupar el comando *random.randint(0, 1),* que te entregará aleatoriamente un 0 o 1.
3. Si el valor obtenido es 0; la bolita debe ir a la izquierda, caso contrario a la derecha.
4. La salida del programa debe indicar el valor obtenido de *random.randint(0, 1)* y las decisiones correspondientes al pasar por las filas del programa.
5. Haz otro programa, donde se pueda simular con grandes cantidades de bolitas y niveles. La salida debe ser solamente la cantidad de bolitas en cada contenedor.

Indicación:

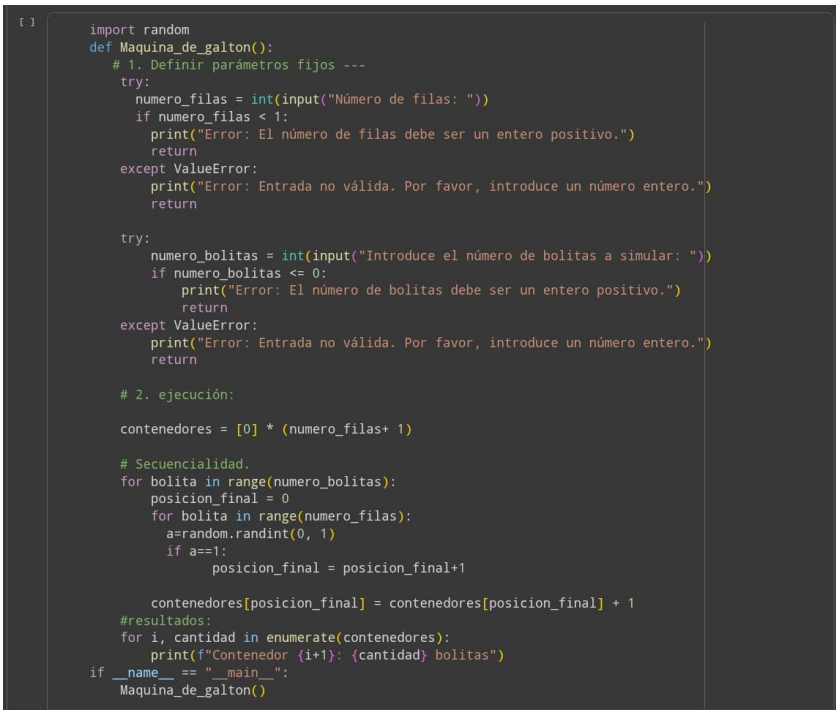
Anexo.



```
[ ]
import random
def Maquina_de_galton():
   # 1. Definir parámetros fijos ---
    try:
      numero_filas = int(input("Número de filas: "))
      if numero_filas < 1:
        print("Error: El número de filas debe ser un entero positivo.")
        return
    except ValueError:
        print("Error: Entrada no válida. Por favor, introduce un número entero.")
        return

    try:
        numero_bolitas = int(input("Introduce el número de bolitas a simular: "))
        if numero_bolitas <= 0:
            print("Error: El número de bolitas debe ser un entero positivo.")
            return
    except ValueError:
        print("Error: Entrada no válida. Por favor, introduce un número entero.")
        return

    probabilidad_derecha = 0.5

    # 2. ejecución:

    contenedores = [0] * (numero_filas + 1)
    # Secuencialidad.
    for bolita in range(numero_bolitas):
        posicion_final = 0
        for bolita in range(numero_filas):
          a= random.randint(0, 1)
          print(a)
          if a==1 :
                posicion_final = posicion_final+1
                print("derecha")
          else:
            print("izquierda")
        contenedores[posicion_final] = contenedores[posicion_final] + 1
        print(contenedores)
    #resultados:
    for i, cantidad in enumerate(contenedores):
        print(f"Contenedor {i+1}: {cantidad} bolitas")
if __name__ == "__main__":
    Maquina_de_galton()
```

Ilustración 55. Programa que muestra el recorrido de una bolita

Fuente: Creación Propia



```
[ ]
    import random
    def Maquina_de_galton():
       # 1. Definir parámetros fijos ---
        try:
          numero_filas = int(input("Número de filas: "))
          if numero_filas < 1:
            print("Error: El número de filas debe ser un entero positivo.")
            return
        except ValueError:
            print("Error: Entrada no válida. Por favor, introduce un número entero.")
            return

        try:
            numero_bolitas = int(input("Introduce el número de bolitas a simular: "))
            if numero_bolitas <= 0:
                print("Error: El número de bolitas debe ser un entero positivo.")
                return
        except ValueError:
            print("Error: Entrada no válida. Por favor, introduce un número entero.")
            return

        # 2. ejecución:

        contenedores = [0] * (numero_filas+ 1)

        # Secuencialidad.
        for bolita in range(numero_bolitas):
            posicion_final = 0
            for bolita in range(numero_filas):
              a=random.randint(0, 1)
              if a==1:
                    posicion_final = posicion_final+1

            contenedores[posicion_final] = contenedores[posicion_final] + 1
        #resultados:
        for i, cantidad in enumerate(contenedores):
            print(f"Contenedor {i+1}: {cantidad} bolitas")
    if __name__ == "__main__":
        Maquina_de_galton()
```

Ilustración 56. Programa que simula grandes cantidades de bolitas.

Fuente: Creación Propia

4.4 Indicadores Observables de las actividades

En el presente apartado se presentan los Indicadores Observables de las actividades que componen los módulos didácticos descritos en el punto 4.3. Para ello, se aplica la Rúbrica de Potencial de la Tarea del instrumento Instructional Quality Assessment (IQA) de Boston (2012) a cada una de las actividades, tanto desenchufadas como conectadas, con el propósito de analizar el nivel de demanda cognitiva que estas promueven en los estudiantes.

De este modo, los indicadores observables presentados a continuación permiten evidenciar la coherencia entre el diseño didáctico de los módulos, las habilidades del Pensamiento Computacional y las habilidades matemáticas del currículo, así como fundamentar que las actividades propuestas alcanzan niveles de alta exigencia cognitiva, particularmente aquellas clasificadas en el nivel Hacer Matemáticas (Doing Mathematics) de la rúbrica IQA.

Soy un Robot		
Actividad Específica	Nivel	Justificación Basada en Evidencia considerando el Criterio IQA.
A1. Primera Actividad Desconectada: **“Soy un Robot”**	3 - Procedimientos con Conexiones	Criterio: Uso de procedimientos para desarrollar comprensión profunda de conceptos. Evidencia: Los estudiantes deben "diseñar la secuencia de comandos completa" para mover al Rover. No es una aplicación mecánica; exige conectar múltiples representaciones (mapa físico, comandos verbales y acción motriz),

		vinculando el algoritmo con el concepto matemático de traslación.
A.2. Segunda Actividad Desconectada: **“Soy un Robot: Algoritmos con Lógica Condicional”**	4 - Hacer Matemáticas *(Doing Mathematics)*	Criterio: Pensamiento complejo no algorítmico; no hay un camino predecible. Evidencia: El desafío es *"darle inteligencia al Rover"* mediante reglas condicionales para un entorno desconocido. El estudiante no sigue una ruta fija, sino que debe generalizar una solución lógica ("Si... entonces...") que funcione ante la incertidumbre, lo cual exige un alto nivel de abstracción.
A.3. Primera Actividad Conectada: **“Robot Digital: Secuencias en Code+”**	3 - Procedimientos con Conexiones	Criterio: Procedimientos conectados al significado. Evidencia: La tarea exige *"transferir los conceptos algorítmicos al entorno digital"*. El estudiante debe traducir su lógica mental a una secuencia lineal de bloques. Aunque el camino está predefinido, la actividad requiere conectar la sintaxis visual con la acción del objeto, evitando la codificación "ciega" mediante la visualización inmediata del error.
A.4 Segunda Actividad Conectada. **“Robot Digital: Optimización de Rutas en Code+”**	4 - Hacer Matemáticas *(Doing Mathematics)*	Criterio: Explorar y comprender la naturaleza de relaciones matemáticas/lógicas. Evidencia: La instrucción pide *"diseñar un 'cerebro' para el sprite"* capaz de resolver *múltiples* laberintos, no solo uno. Esto obliga al estudiante a abandonar la secuencia lineal

		y crear reglas generales (abstracción). Debe analizar la estructura del problema para crear un algoritmo eficiente y robusto, lo que constituye un verdadero

Tabla 4 - Soy un Robot: Rúbrica de Potencial de la Tarea del instrumento IQA.

Fiesta de Baile		
Actividad Específica	Nivel	Justificación Basada en Evidencia considerando el Criterio IQA.
B.1. Primera Actividad Desconectada: **“Fiesta de Baile”**	4 - Hacer Matemáticas (*Doing Mathematics*)	Criterio: Pensamiento complejo no algorítmico; requiere identificar patrones y formar generalizaciones sin una regla previa. Evidencia: El estudiante recibe una secuencia "bruta" de movimientos y la instrucción es *"descubrir la estructura que se repite"* para comprimirla. No se le da el algoritmo; debe realizar una abstracción inductiva (analizar datos -> inferir la regla), explorando la naturaleza de la repetición matemática por sí mismo.
B.2. Segunda Actividad Desconectada: **“Fiesta de Baile 2”**	3 - Procedimientos con Conexiones	Criterio: Uso de procedimientos que permanecen estrechamente conectados a conceptos matemáticos subyacentes.

		Evidencia: “El desafío es analizarla, identificar sus patrones ocultos en diferentes niveles y reescribirla usando la menor cantidad de instrucciones, descubriendo así el poder de los bucles anidados para generar complejidad de forma simple.”. No es una ejecución mecánica, pues incluye la validación lógica del procedimiento (depuración humana).
B. 3. Primera Actividad Conectada: **“Optimización de Código con Bucles en Code+”**	3 - Procedimientos con Conexiones	Criterio: Aplicación de un procedimiento general (bloque Repetir) conectado al concepto de eficiencia. Evidencia: La consigna pide *"reescribir el código usando menos bloques"*. El estudiante no solo arrastra instrucciones; debe analizar la estructura del programa lineal y mapear el patrón lógico encontrado en la fase desenchufada hacia la sintaxis del bloque "Repeat". La conexión con el significado (hacerlo más eficiente) es explícita.
B.4 Segunda Actividad Conectada: **“Optimización de Código con Bucles en Code+”**	4 - Hacer Matemáticas (*Doing Mathematics*)	Criterio: Explorar y comprender la naturaleza de relaciones lógicas complejas (Causa-Efecto) sin un camino predecible. Evidencia: Se introduce el concepto de Eventos ("Al presionar tecla..."). La tarea es abierta: *"Crear una coreografía interactiva propia"*. El estudiante debe diseñar un sistema lógico donde múltiples reglas condicionales interactúan

		de forma no lineal. Al no haber una "solución única" y requerir el diseño de un comportamiento complejo y autónomo, la demanda cognitiva es máxima.

Tabla 5 - Fiesta de Baile: Rúbrica de Potencial de la Tarea del instrumento IQA.

Magia con Números		
Actividad Específica	Nivel	Justificación Basada en Evidencia considerando el Criterio IQA.
C.1. Primera Actividad Desconectada **"Magia con Números: Develando el Código Binario."**	4 - Hacer Matemáticas (*Doing Mathematics*)	Criterio: La tarea requiere explorar y comprender la naturaleza de relaciones matemáticas (patrones) sin un camino algorítmico explícito. Evidencia: El docente realiza el truco, pero no explica cómo funciona. La tarea del estudiante es *"analizar las tarjetas y descubrir el patrón"*. El estudiante debe comparar los números presentes en cada tarjeta para inferir la regla de las potencias de base 2 $(1, 2, 4, 8, 16)$ y la descomposición aditiva. Al exigir la formulación de una conjetura

		matemática propia a partir de la observación de datos, cumple con el nivel más alto de la rúbrica.
C.2. Segunda Actividad Desconectada: **“Magia con Números: Construyendo el Código Binario.”**	3 - Procedimientos con Conexiones	Criterio: Uso de procedimientos generales que permanecen estrechamente conectados a conceptos matemáticos subyacentes. Evidencia: Una vez descubierto el patrón, los estudiantes practican la representación de números usando el sistema "Encendido/Apagado" (Binario). Aunque realizan un procedimiento algorítmico (sumar los valores visibles), la tarea exige conectar esta acción con el concepto de valor posicional y la estructura del sistema binario. No es una suma mecánica, pues deben decidir qué bits activar para alcanzar el número objetivo.
C.3. Actividad Conectada: **“Programando un Adivino Binario”**	3 - Procedimientos con Conexiones	Criterio: Uso de procedimientos para desarrollar comprensión de conceptos (traducción de representaciones). Evidencia: El desafío es *"construir un programa que realice el truco"*. El estudiante debe traducir la lógica matemática descubierta (suma acumulativa) a una estructura sintáctica en Scratch. Debe crear una Variable ("Total") y utilizar Condicionales ("Si respuesta = Sí, entonces cambiar Total por [Potencia]"). Esta actividad conecta el

		procedimiento de codificación con la lógica aritmética: el estudiante valida su comprensión del algoritmo al tener que enseñárselo a la computadora.

Tabla 6 - Magia con Números: Rúbrica de Potencial de la Tarea del instrumento IQA.

Detección de errores a través de Paridad		
Actividad Específica	Nivel	Justificación Basada en Evidencia considerando el Criterio IQA.
D.1. Primera Actividad Desconectada: **“Magia con Paridad”**	3 - Procedimientos con Conexiones	Criterio: Uso de un procedimiento general conectado estrechamente a conceptos matemáticos subyacentes. Evidencia: La tarea consiste en *"agregar la fila y columna extra"* para asegurar que todas las sumas sean pares. El estudiante aplica un procedimiento (contar y completar), pero este no es mecánico; requiere conectar la acción con el concepto de propiedad par. Debe monitorear constantemente el estado de la matriz para mantener la integridad del sistema.

D.2. Segunda Actividad Desconectada: **“Algoritmo de Hamming.”**	3 - Procedimientos con Conexiones	Criterio: Uso de procedimientos (algoritmo) con el propósito de desarrollar una comprensión profunda de conceptos matemáticos/computacionales. Evidencia: La actividad proporciona una "serie de pasos" explícita (pasos 1 al 9) que los estudiantes deben seguir para calcular los bits de paridad (P1, P2, P3) y verificar errores (C1, C2, C3). No se clasifica como *Memorización* (Nivel 1) ni *Sin Conexiones* (Nivel 2) porque la manipulación física de las cartas exige conectar representaciones: el estudiante debe traducir el estado físico ("cara" o "reverso") al valor binario ("1" o "0") y comprender la lógica posicional para triangular la ubicación del error. El objetivo declarado es "comprender cómo la paridad puede ayudar a detectar y localizar un error", vinculando el procedimiento algorítmico con el concepto matemático de corrección de datos.
D.3. Primera Actividad Conectada: **“Algoritmo de**	3 - Procedimientos con Conexiones	Criterio: Uso de un procedimiento general conectado estrechamente a conceptos matemáticos subyacentes. Evidencia: La actividad guía al estudiante para *"calcular los bits de paridad"* usando una

Hamming en Scratch”		operación matemática específica: módulo 2 (mod 2). Aunque los pasos son explícitos, no es una transcripción mecánica (Nivel 2) porque el estudiante debe resolver un conflicto cognitivo clave: la diferencia de indexación. El texto advierte explícitamente que *"en Hamming la posición 1 es a la derecha, en Scratch a la izquierda"*. El estudiante debe adaptar el procedimiento estándar para que funcione en la estructura de datos de Scratch (Listas), conectando la abstracción binaria con la lógica de programación.
D.4. Segunda Actividad Conectada: **“Algoritmo de Hamming en Python.”**	4 - Hacer Matemáticas (*Doing Mathematics*)	Criterio: Pensamiento complejo no algorítmico; requiere explorar relaciones matemáticas, analizar estructuras y manejar la abstracción sin un camino predecible único. Evidencia: La tarea exige *"completar las funciones 'generar_hamming' y 'verificar_hamming'"* para que funcionen con cualquier entrada de datos (generalización). El factor determinante para el Nivel 4 es la resolución de conflictos estructurales entre la matemática y la informática: el texto indica explícitamente que *"Python cuenta las posiciones de izquierda a derecha y comenzando desde cero, a diferencia del sistema binario"*. El estudiante no puede

		simplemente "copiar" la fórmula matemática; debe modelar una solución lógica que traduzca el índice matemático (base 1, derecha-izq) al índice del array (base 0, izq-derecha), realizando ajustes algebraicos (pos - 1, inversiones) para que el corrector funcione. Esto implica comprender la naturaleza de la estructura de datos y no solo ejecutar un comando.

Tabla 7 - Detección de Errores a través de Paridad: Rúbrica de Potencial de la Tarea del instrumento IQA.

Máquina de Galton		
Actividad Específica	Nivel	Justificación Basada en Evidencia considerando el Criterio IQA.
E.1 Actividad desconectada: **"Maquina de Galton Humana"**	3 - Procedimientos con Conexiones	Criterio: Uso de un procedimiento general con el propósito explícito de desarrollar una comprensión profunda de conceptos matemáticos (probabilidad y distribución). Evidencia: La tarea consiste en ejecutar un algoritmo físico: *"lanzar una moneda: si sale cara avanzará hacia la derecha y si sale sello avanzará hacia la izquierda"*. Aunque es un procedimiento reglado, se clasifica en Nivel 3 porque la acción no es

		mecánica; está diseñada para conectar la experiencia individual (el azar de la moneda) con un resultado colectivo matemático. La instrucción final de *"calcular las frecuencias relativas... para comparar los resultados empíricos"* obliga a los estudiantes a conectar los datos físicos (cuerpos en contenedores) con el concepto abstracto de probabilidad y distribución estadística, otorgando significado al procedimiento.
E. 2 Primera Actividad Conectada: **“Maquina de Galton Scratch”**	3 - Procedimientos con Conexiones	Criterio: Uso de procedimientos para desarrollar comprensión de conceptos (traducción de lógica). Evidencia: La tarea es construir el simulador visualmente. El estudiante debe traducir la lógica de la moneda a bloques de colores (Si <número al azar = 1> entonces...). Se mantiene en Nivel 3 porque el foco está en la representación: entender cómo los operadores lógicos y de movimiento controlan el objeto. Es el "andamiaje" visual para entender la lógica condicional antes de la abstracción pura.
E.3 Actividad Conectada: **“Maquina de**	4 - Hacer Matemáticas	Criterio: Pensamiento complejo no algorítmico; requiere analizar patrones, hacer

Galton con Python.”	(*Doing Mathematics*)	conjeturas y apoyar conclusiones con evidencia. Evidencia: El salto a Python permite simular miles de eventos usando listas y bucles for (sin la limitación gráfica de Scratch). La tarea cognitiva cambia: ya no es mover un objeto, es procesar datos. El estudiante debe analizar la salida numérica (el histograma de frecuencias generado en la consola o gráfico) para inferir y explicar la Distribución Normal. Al exigir la generalización matemática a partir del análisis de grandes volúmenes de datos, la demanda es máxima.

Tabla 8 - Máquina de Galton: Rúbrica de Potencial de la Tarea del instrumento IQA

Capítulo 5: Discusión y Conclusiones

5.1 Discusión de los resultados

Los resultados de esta investigación dialogan de manera estrecha con la literatura internacional sobre pensamiento computacional como competencia transversal del siglo XXI. Diversos autores han enfatizado que el pensamiento computacional constituye un modo de pensamiento orientado a la formulación y resolución de problemas mediante procesos de descomposición, abstracción, reconocimiento de patrones, diseño algorítmico y depuración, aplicable a contextos digitales y no digitales (Wing, 2006, 2011; Aho, 2012; Shute et al., 2017; Bordignon & Iglesias, 2019; García-Peñalvo & Mendes, 2017). El módulo diseñado recoge estas ideas al proponer tareas donde el foco no es “aprender a programar por sí mismo”, sino usar la programación, desenchufada y enchufada, como un medio para estructurar el razonamiento, explicitar procedimientos, ensayar soluciones y revisarlas de manera iterativa, coherente con el enfoque de “aprender haciendo” y con una visión construccionista de la tecnología en educación (Papert, 1980; Badilla & Murillo, 2004; Resnick, 2017).

Asimismo, los hallazgos coinciden con los estudios que muestran una relación estrecha entre pensamiento computacional y educación matemática. Investigaciones recientes señalan que el trabajo con algoritmos, bucles, condiciones y representaciones computacionales activa procesos cognitivos cercanos a los utilizados en la resolución de problemas, la modelación y la argumentación matemática (Weintrop et al., 2016; Rycroft-Smith & Connolly, 2019; Sáez-López et al., 2023). Desde una perspectiva cognitiva, estos vínculos se refuerzan al considerar modelos como el CHC, que conectan el razonamiento fluido, el procesamiento visual y la memoria de trabajo con el desempeño en

matemáticas y en pensamiento computacional (Cattell, 1963; McGrew, 2009; Flanagan & Dixon, 2014; Roman-González et al., 2017; Baddeley, 2012; Geary, 2011). La propuesta de módulo confirma, a nivel de diseño, que es posible operacionalizar estas convergencias mediante actividades que exigen a los estudiantes analizar situaciones, construir representaciones, formular procedimientos y someterlos a verificación, tal como se plantea en el marco de Matemática de PISA y en las orientaciones curriculares nacionales (OCDE, 2014, 2023; MINEDUC, 2015, 2019c).

En el plano metodológico, la elección de enfoques basados en problemas y en diseño de tareas se alinea con la evidencia disponible sobre el potencial del Aprendizaje Basado en Problemas (ABP/PBL) y de las tareas de alta demanda cognitiva para promover pensamiento complejo. La literatura muestra que el PBL favorece la integración de conocimientos, el desarrollo de estrategias de resolución de problemas y la motivación intrínseca, siempre que se sustente en situaciones auténticas y bien estructuradas (Hmelo-Silver, 2004; Jonassen, 2000; Escribano González & Valle, 2010). Estudios recientes reportan impactos positivos del PBL en el desarrollo de habilidades matemáticas y de pensamiento crítico (Arévalo et al., 2024; Grández et al., 2024; Reina et al., 2016; Vera Velázquez et al., 2022), así como en el fortalecimiento del pensamiento computacional en contextos de programación (Gram-Hansen & Jonasen, 2019; Aksakal & Küçük, 2025). En sintonía con estos resultados, el módulo propuesto organiza sus actividades alrededor de problemas matemáticos significativos, donde los estudiantes deben explorar, conjeturar, implementar algoritmos y depurar sus soluciones en interacción con pares y con herramientas digitales.

Últimamente, la investigación dialoga con los marcos de competencias digitales docentes y con los estudios sobre cambio educativo mediado por TIC. Modelos como TPACK y CTPK-12 subrayan que la integración efectiva del pensamiento computacional exige articular conocimiento disciplinar, pedagógico y tecnológico

de manera situada (Mishra & Koehler, 2006; Tikva & Tambouris, 2021), mientras que las propuestas de UNESCO, ISTE, DigCompEdu, INTEF y el Marco para la Buena Enseñanza destacan que el profesorado debe diseñar experiencias de aprendizaje desafiantes, éticas e inclusivas, apoyadas en tecnologías digitales (UNESCO, 2018, 2021, 2023; ISTE, 2017; Redecker & Punie, 2017; INTEF, 2022; CPEIP, 2021). Los modelos de cambio escolar en TIC enfatizan, por su parte, que los procesos de innovación pasan por fases de exploración, integración y transformación que requieren tiempo, apoyo institucional y desarrollo profesional progresivo (Newhouse et al., 2013; Tong & Trinidad, 2005). En este contexto, el módulo diseñado aporta un recurso concreto y alineado con dichos marcos, pero su impacto dependerá de la posibilidad real de que los docentes construyan las competencias necesarias para apropiarse del diseño, adaptarlo a sus contextos y sostener su implementación en el tiempo (Voogt et al., 2015; Yadav et al., 2017).

5.2 Conclusiones

En relación con el objetivo general, la investigación permitió elaborar una propuesta coherente con las exigencias del currículum nacional y los marcos internacionales revisados. El módulo se concretó en una secuencia de actividades que transitan desde experiencias corporales y manipulativas (por ejemplo, recorridos en cuadrículas, juegos de rol y simulaciones humanas) hacia entornos de programación por bloques y posteriormente hacia programación textual, manteniendo como eje articulador la resolución de problemas matemáticos auténticos y progresivamente más complejos. Esta arquitectura didáctica responde a la necesidad de que el pensamiento computacional se comprenda como un marco cognitivo transversal y no solo como un conjunto de destrezas técnicas, reforzando la idea de que su enseñanza puede y debe contribuir al desarrollo de competencias de orden superior en matemática, tales

como análisis, modelación, argumentación y comunicación (Wing, 2006, 2011; Shute et al., 2017; MINEDUC, 2015, 2019c, 2021b).

En relación con el primer objetivo específico, el estudio logró sistematizar un cuerpo robusto de referentes que integra marcos internacionales sobre competencias del siglo XXI, políticas nacionales, modelos de integración tecnológica y evidencia empírica reciente. La revisión de trabajos sobre habilidades del siglo XXI (Binkley et al., 2012; Maggio, 2018; Scott, 2015), de marcos de competencia digital docente (UNESCO, 2018; Redecker & Punie, 2017; ISTE, 2017; INTEF, 2022; MINEDUC, 2025; CPEIP, 2021) y de propuestas específicas para el desarrollo del pensamiento computacional en educación obligatoria (Wing, 2006; Bocconi et al., 2016; Voogt et al., 2015; Csizmadia et al., 2015; Lye & Koh, 2014; Shute et al., 2017) permitió construir un marco referencial que no solo legitima curricularmente la integración entre Matemática y pensamiento computacional, sino que además ofrece criterios concretos para orientar el diseño de tareas cognitivamente demandantes y alineadas con los Objetivos de Aprendizaje vigentes. Esta sistematización se constituyó en el soporte teórico que dio coherencia a las decisiones didácticas adoptadas en el módulo.

Respecto del segundo objetivo específico, la investigación permitió elaborar una ruta de actividades que explicita la correspondencia funcional entre las habilidades matemáticas (resolver problemas, representar, modelar, argumentar y comunicar) y las habilidades del pensamiento computacional (descomposición, reconocimiento de patrones, abstracción, pensamiento algorítmico y evaluación/depuración). El diseño de las actividades, que incluye experiencias como “Soy un robot”, “Fiesta de baile”, “Magia con números” y “Máquina de Galton”, se fundamentó en progresiones de aprendizaje documentadas en la literatura sobre pensamiento computacional (Brennan & Resnick, 2012; Rich et al., 2017, 2018, 2019, 2020; Bell et al., 2009, 2015; Resnick et al., 2009) y en las

orientaciones curriculares del MINEDUC para Matemática y para el Electivo de pensamiento computacional y Programación (MINEDUC, 2019c, 2021b). Como resultado, se obtuvo un diseño que no solo es coherente con las trayectorias sugeridas por estos marcos, sino que además ofrece al profesorado una secuencia concreta y adaptable para articular experiencias analógicas y digitales en el aula.

5.3 Limitaciones, desafíos y proyecciones

Una limitación de este trabajo se relaciona con el alcance del estudio, que se concentró en el diseño y validación interna del módulo, sin avanzar hacia una fase de implementación sistemática en diversos contextos escolares. Aunque la propuesta se sustenta en un marco teórico robusto y en criterios didácticos bien fundamentados, la ausencia de datos empíricos sobre su uso en aula impide emitir juicios concluyentes respecto a su impacto real en la resolución de problemas matemáticos, el desarrollo del pensamiento computacional y de las habilidades del siglo XXI. En consecuencia, las conclusiones se sitúan principalmente en el plano del diseño y su coherencia curricular, más que en resultados de aprendizaje evaluados en estudiantes.

La implementación de los módulos diseñados presenta un desafío en relación con las condiciones materiales necesarias para el tramo “enchufado” de los módulos. A pesar de los avances en dotación tecnológica del sistema escolar chileno, persisten brechas en el acceso a dispositivos, conectividad y soporte técnico entre establecimientos y territorios, lo que podría dificultar la puesta en práctica de actividades que dependen de entornos de programación o plataformas digitales (Agencia de Calidad de la Educación, 2021; OCDE, 2018). Este aspecto cobra mayor relevancia en contextos rurales o vulnerables, donde soluciones desenchufadas y de bajo costo son esenciales para asegurar la

equidad en el acceso a experiencias formativas relacionadas con Matemática y pensamiento computacional.

Un segundo desafío tiene que ver con la formación docente. La evidencia internacional y nacional coincide en señalar que la competencia digital y el conocimiento específico sobre pensamiento computacional siguen siendo desafíos significativos para el profesorado (UNESCO, 2018, 2021; Agencia de Calidad de la Educación, 2021; Voogt et al., 2015; Yadav et al., 2017; MINEDUC, 2025). La propuesta de módulo presupone un docente capaz de integrar lenguajes de programación por bloques y textuales, interpretar trayectorias de pensamiento computacional, diseñar actividades basadas en ABP y gestionar interacciones colaborativas. Sin procesos de desarrollo profesional suficientes, tales como, comunidades de práctica, cursos de actualización, mentorías o ajustes en la formación inicial, existe el riesgo de que su implementación sea parcial, fragmentada o reducida a ejercicios técnicos sin profundidad matemática.

Estas limitaciones y desafíos, sin embargo, abren oportunidades claras para proyectar futuras investigaciones y mejoras. En primer lugar, se vuelve imprescindible avanzar hacia estudios de implementación piloto del módulo en distintos niveles educativos y tipos de establecimientos, utilizando metodologías mixtas que permitan analizar su impacto en la resolución de problemas matemáticos, en el desarrollo del pensamiento computacional, en la motivación hacia la disciplina y en la percepción de autoeficacia del estudiantado (Roman-González et al., 2017; Swanson, 2016; Geary, 2011; Sáez-López et al., 2023). Dichas investigaciones podrían incluir diseños cuasiexperimentales con grupos de comparación, análisis de producciones matemáticas y observación de dinámicas de aula para evaluar desempeños reales.

En segundo lugar, se proyecta la necesidad de profundizar en la formación docente asociada al uso del módulo. Investigar modelos de desarrollo profesional que integren pensamiento computacional, Matemática y competencias digitales,

coherentes con marcos como TPACK, CTPK-12, DigCompEdu y el Marco de Competencias Digitales Docentes del MINEDUC (Mishra & Koehler, 2006; Tikva & Tambouris, 2021; Redecker & Punie, 2017; MINEDUC, 2025), permitiría comprender qué apoyos, acompañamientos y recursos favorecen la apropiación docente y su capacidad de adaptar el módulo a sus contextos particulares.

Finalmente, los módulos abren posibilidades de expansión hacia enfoques interdisciplinarios de tipo STEAM, integrando ciencia, tecnología, ingeniería, artes y matemática en proyectos que mantengan la resolución de problemas y el pensamiento computacional como ejes articuladores (Yakman, 2008, 2011; Martín-Páez et al., 2019; Sanders, 2009; UNESCO, 2023). Explorar estas vías permitiría consolidar un itinerario formativo continuo —desde la educación básica hasta la educación media— que articule de manera coherente competencias matemáticas, digitales y ciudadanas. Este tipo de proyección responde tanto a las demandas globales de la sociedad del conocimiento como a los desafíos curriculares que enfrenta actualmente el sistema educativo chileno.

Bibliografía

Agencia de Calidad de la Educación. (2021). Informe de Resultados ICILS 2018. Agencia de Calidad de la Educación. https://archivos.agenciaeducacion.cl/

Aho, A. V. (2012). Computational thinking. The Computer Journal, 55(4), 832–835. https://doi.org/10.1093/comjnl/bxs074

Aksakal, H., & Kucuk, S. (2025). Secondary school students' computational thinking skills, group cohesion, and performance in problem-based programming education. *The Journal of Systems and Software, 230*, 112535. https://doi.org/10.1016/j.jss.2025.112535

Animum Creativity Advanced School. (2023, 21 de septiembre). *¿Qué es un loop y qué papel juega en la animación 3D?* Animum. https://www.animum3d.com/

Arévalo, M., García, M., & Jaramillo, J. (2024). Problem-based learning (PBL) as a methodology to strengthen mathematical skills —problem solving— in basic education. *Cultura, Educación y Sociedad, 15*(1). https://dialnet.unirioja.es/

Baddeley, A. (2012). Working memory: Theories, models, and controversies. *Annual Review of Psychology, 63*, 1–29. https://doi.org/10.1146/annurev-psych-120710-100422

Badilla, M., & Murillo, A. (2004). *El construccionismo en educación: Fundamentos y experiencias*. 10.15517/aie.v4i1.9048

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: school students doing real computing without computers. *New Zealand Journal of Applied Computing and Information Technology*. https://researchportal.bath.ac.uk/

Bell, T., Witten, I. H., & Fellows, M. (2015). *Computer Science Unplugged: An enrichment and extension programme for primary-aged children* (Adaptado para uso en aula por R. Adams & J. McKenzie). University of Canterbury. https://classic.csunplugged.org/

Bers, M., Strawhacker, A., & Vizner, M. (2018). The design of early childhood makerspaces to support positive technological development: Two case studies. *Library Hi Tech, 36*(1), 75–96. https://doi.org/10.1108/LHT-06-2017-0112

Bialik, M., & Fadel, C. (2015). Meta-learning for the 21st century: What should students learn? Canter for CurriculumRedesign. https://www.researchgate.net/

Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., & Rumble, M. (2012). Defining twenty-first century skills. *Assessment and teaching of 21st century skills* (pp. 17–66). Springer. https://doi.org/10.1007/978-94-007-2324-5_2

Boom, K., Bower, M., Arguel, A., Siemon, J., & Scholkmann, A. (2018). Relationship between computational thinking and a measure of intelligence as a general problem-solving ability. Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. https://doi.org/10.1145/3197091.3197104

Bordignon, F., & Iglesias, A. (2019). *Introducción al pensamiento computacional*. Educar S. E.; Universidad Pedagógica Nacional (UNIPE). I-https://www.researchgate.net/

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., & Punie, Y. (2016). *Developing computational thinking in compulsory education: Implications for policy and practice* (EUR 28295 EN). Publications Office of the European Union. https://doi.org/10.2791/792158

Boston, M. D. (2012). Assessing instructional quality in mathematics. *The Elementary School Journal, 113*(1), 76–104. https://doi.org/10.1086/666387

Brennan, K., & Resnick, M. (2012). New Frameworks for Studying and Assessing the Development of Computational Thinking. Proceedings of the 2012 Annual Meeting of the American Educational Research Association. http://scratched.gse.harvard.edu/

CAS (2015). Pensamiento Computacional. Guía para profesores. Computing At School (CAS). https://www.computingatschool.org.uk/

Castellano Sánchez, D., García Carmona, M. Á., Prados Martínez, A., Moya Sánchez, A., Rodríguez Corona, M., y Aguilar Pérez, F. J. (2025). Las matemáticas son la leche: Máquina humana de Galton. *Uno. Revista de Didáctica de las Matemáticas, 107*, 61-73.

Cattell, R. B. (1963). Theory of fluid and crystallized intelligence: A critical experiment. *Journal of Educational Psychology, 54*(1), 1–22. https://doi.org/10.1037/h0046743

Chinofunga, M. D., Chigeza, P., & Taylor, S. (2024). How can procedural flowcharts support the development of mathematics problem-solving skills? Mathematics Education Research Journal. https://doi.org/10.1007/s13394-024-00483-3

Cobb, P., Confrey, J., diSessa, A., Lehrer, R., & Schauble, L. (2003). Design experiments in educational research. *Educational Researcher, 32*(1), 9–13. https://www.jstor.org/

Code.org. (s.f.). *Curso C (2021): Lección 3. Programación con Angry Birds*. https://studio.code.org/

Code.org. (s.f.). *Curso C (2021): Lección 4. Depuración en el laberinto*. https://studio.code.org/

Code.org. (s.f.). *Curso Express (2021): Lección 15. Condicionales (Si/sino) con Abeja*. https://studio.code.org/

Code.org. (s.f.). *Curso Express (2021): Lección 16. Bucles 'mientras' en Granjera*. https://studio.code.org/

Code.org. (s.f.). *Fiesta Desenchufada: Unidad 1 - Lección 1: Calentamiento*. https://studio.code.org/

Code.org. (s.f.). *Fiesta de baile (edición 2019): Unidad 1*. https://studio.code.org/

Code.org. (s.f.). *Curso B (2021): Lección 7. Bucles con la cosechadora*. https://studio.code.org/

Code.org. (s.f.). *Curso B (2021): Lección 8. Bucles con Laurel*. https://studio.code.org/

Comisión Nacional de Evaluación y Productividad. (2023). *Informe Anual de Productividad 2023*. https://cnep.cl

CPEIP. (2021). *Marco para la Buena Enseñanza*. https://estandaresdocentes.mineduc.cl/

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *Computational thinking: A guide for teachers*. Computing at School. https://eprints.soton.ac.uk/

CSTA & ISTE. (2011). *Operational definition of computational thinking*. https://cdn.iste.org/

CS Unplugged. (s.f.). *Parity Magic (Junior)*. https://www.csunplugged.org/

Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R. (2014). Introducing teachers to computational thinking using unplugged storytelling. In *Proceedings of the 9th Workshop in Primary and Secondary Computing*

Education (pp. 89–92). Association for Computing Machinery. https://doi.org/10.1145/2670757.2670767

CSUnplugged. (s.f.). Kidbots. https://www.csunplugged.org/

Escribano González, A., & Valle, Á. D. (Coords.). (2010). *Aprendizaje basado en problemas (ABP): una propuesta metodológica en educación superior* (Ed.). Narcea Ediciones. https://elibro.net/es/

Felmer, P., Perdomo-Díaz, J. & Reyes, C. (2019), The ARPA Experience in Chile: Problem Solving for Teachers' Professional Development. IN Liljedahl, P., Santos-Trigo, M. (eds) Mathematical Problem Solving. (pp. pp 311–337). https://doi.org/10.1007/978-3-030-10472-6_14

Flanagan, D. P., & Dixon, S. G. (2014). The Cattell-Horn-Carroll theory of cognitive abilities. En C. R. Reynolds, K. J. Vannest, & E. Fletcher-Janzen (Eds.), Wiley. https://doi.org/10.1002/9781118660584.ese0431

Fundación Kodea (2023). Diagnóstico Universidades: La formación de los futuros docentes en ciencias de la computación. Fundación Kodea en colaboración con el Hub Chile_Programa. https://kodea.org/

Fundación País Digital. (2023). *Futuro de la educación en Chile: Innovación, tecnología y habilidades del siglo XXI*. Fundación País Digital. https://media.paisdigital.org/

García-Peñalvo, F. J., & Mendes, A. J., Exploring the computational thinking effects in pre-universityeducation, Computers in Human Behavior (2017), https://doi.org/10.1016/j.chb.2017.12.005

Geary, D. (2011). Cognitive predictors of achievement growth in mathematics. *Child Development, 82*(6), 1860–1876. https://doi.org/10.1111/j.1467-8624.2011.01641.x

Google for Education. (2016). *Computational thinking for educators*. https://services.google.com/

Gram-Hansen, S. B., & Jonasen, T. S. (2019). *Computational Thinking in Problem Based Learning – Exploring the Reciprocal Potential*. In *Transforming Learning with Meaningful Technologies* (pp. 573–576). Springer. https://doi.org/10.1007/978-3-030-29736-7_42

Grández, A., Ramírez, F., & Grández, L. (2024). El aprendizaje basado en problemas y su influencia en el desarrollo del pensamiento crítico en estudiantes del nivel superior. *IGOBERNANZA, 7*(28), 246–282. https://doi.org/10.47865/igob.vol7.n28.2024.384

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. *ACM, 17*(1), 1–24. https://doi.org/10.1145/3017680.3017723

Hernando, A. (2015). *Viaje a la escuela del siglo XXI: Así trabajan los colegios más innovadores del mundo*. https://gc.scalahed.com/

Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn? *Educational Psychology Review, 16*(3), 235–266. https://doi.org/10.1023/B:EDPR.0000034022.16470.f3

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development, 48*(4), 63–85. https://link.springer.com/article/10.1007/BF02300500

IBM. (2023). ¿Qué es la simulación de Monte Carlo? IBM. https://www.ibm.com/

Isharyadi, R., & Juandi, D. (2023). A Systematics Literature Review of Computational Thinking in Mathematics Education: Benefits and Challenges. Formatif: Jurnal Ilmiah Pendidikan MIPA, 13(1). http://dx.doi.org/10.30998/formatif.v13i1.15922

ISTE. (2017). *ISTE Standards for Educators*. https://www.iste.org/standards

INTEF. (2022). *Marco de competencia digital docente*. https://intef.es

Jonassen, D. H. (2000). Toward a design theory of problem solving. *Educational Technology Research and Development, 48*(4), 63–85. https://link.springer.com/article/10.1007/BF02300500

Laurillard, D. (2012). *Teaching as a design science: Building pedagogical patterns for learning and technology*. Taylor & Francis Group. https://ebookcentral.proquest.com/

Larsen, K. R. T., & Bong, C. H. (2016). A tool for addressing construct identity in literature reviews and meta-analyses. *MIS Quarterly, 40*(3), 529–551. https://doi.org/10.25300/MISQ/2016/40.3.01

Lucidchart. (s.f.). *Símbolos comunes de los diagramas de flujo*. https://www.lucidchart.com/

Lye, S. Y., & Koh, J. H. L. (2014). Review of research on computational thinking in education. *Computers in Human Behavior, 41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Maggio, M. (2018). Habilidades del siglo XXI. Cuando el futuro es hoy. Ciudad Autonoma de Buenos Aires: Fundación Santillana. https://www.fundacaosantillana.org.br/

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on*

Computing Education, 10(4), Article 16. https://doi.org/10.1145/1868358.1868363

Martín-Páez, T., Aguilera, D., Perales-Palacios, F. J., & Vílchez-González, J. M. (2019). What are we talking about when we talk about STEM education? A review of literature. *Science Education, 103*(4), 799–822. https://doi.org/10.1002/sce.21522

McGrew, K. S. (2009). CHC theory and the human cognitive abilities project: Standing on the shoulders of giants. *Intelligence, 37*(1), 1–10. https://doi.org/10.1016/j.intell.2008.08.004

MINEDUC. (2012). Bases curriculares 1° a 6° básico. Ministerio de Educación de Chile. https://www.curriculumnacional.cl/

MINEDUC. (2015). *Bases Curriculares 7° básico a 2° medio*. Gobierno de Chile. https://www.curriculumnacional.cl/

MINEDUC. (2019a). Decreto 193: Aprobación de Bases Curriculares para los cursos de 3º y 4º año de Educación Media, en asignaturas que indica. Diario Oficial. https://bcn.cl/2let6

MINEDUC. (2019b). Plan Nacional de Lenguajes Digitales. https://sitios.mineduc.cl/lenguajesdigitales/

MINEDUC. (2019c). Bases curriculares 3° y 4° medio. Ministerio de Educación de Chile. https://www.curriculumnacional.cl/

MINEDUC. (2021a) *Un recorrido por las habilidades para el siglo XXI*. https://bibliotecadigital.mineduc.cl/

MINEDUC. (2021b). *Programa de Estudio: Pensamiento Computacional y Programación (3.º y 4.º medio)*. https://bibliotecadigital.mineduc.cl/

MINEDUC (2022). *Ruta de aprendizaje para el pensamiento computacional*. Departamento de Ciencias de la Computación, Universidad de Chile. https://bibliotecadigital.mineduc.cl/

MINEDUC. (2024a). *Propuesta de Actualización Curricular*. https://ceppe.uc.cl/

MINEDUC, Oficina Regional Multisectorial de la UNESCO en Santiago, & Centro Saberes Docentes de la Universidad de Chile. (2024b). *Informe ejecutivo de resultados del Congreso Pedagógico y Curricular: La educación es el tema*. Ministerio de Educación. https://congresopedagogico.mineduc.cl/

MINEDUC. (2025). *Marco orientador de competencias digitales docentes*. https://bibliotecadigital.mineduc.cl

Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record, 108*(6), 1017–1054. https://doi.org/10.1111/j.1467-9620.2006.00684.x

Molina, M., Castro, E., Molina, J. L., & Castro, E. (2011). Un acercamiento a la investigación de diseño a través de los experimentos de enseñanza. *Enseñanza de las Ciencias, 29*(1), 75–88. https://www.researchgate.net/

Munasinghe, B., Bell, T., & Robins, A. (2023). Unplugged activities as a catalyst when teaching introductory programming. *Journal of Pedagogical Research, 7*(2), 56-71. https://doi.org/10.33902/JPR.202318546

NASA. (s.f.). Un día en la vida de un Mars Rover D. Mars Exploration Program.A Day in the Life of a Mars Rover Driver. https://mars.nasa.gov/

NASA (s.f.). *Deep Space Network*. https://www.nasa.gov/

Newhouse, C., Clarkson, B., & Trinidad, S. (2013). *A framework for leading school change in using ICT*. Edith Cowan University. https://www.aare.edu.au/

OCDE. (2014). *PISA 2012 Results: Creative problem solving*. https://doi.org/10.1787/9789264208070-en

OCDE. (2018). *The future of education and skills: Education 2030*. OCDE Publishing. https://www.OCDE.org/education/

OCDE. (2024). *PISA 2022 Mathematics Framework*. https://www.OCDE.org

Órdenes, X., Roberts, R., Rojas, P., & Rojas, F. (2024). *Estrategia de transformación digital Chile Digital 2035: Plan de conectividad efectiva*. Comisión Económica para América Latina y el Caribe (CEPAL). https://www.cepal.org/

Palomino Alca, J. T., & Osorio Vidal, V. G. (2023). *El aprendizaje basado en problemas para el logro de competencias en educación superior*. Revista Dilemas Contemporáneos: Educación, Política y Valores, 10(2), 1–20. http://www.dilemascontemporaneoseducacionpoliticayvalores.com/

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. ACM. https://dl.acm.org/doi/pdf/10.5555/1095592

Papert, S. (1991). Situating Constructionism. In I. Harel, & S. Papert (Eds.), Constructionism: Research reports and essays. Norwood, NJ: Ablex.

Partovi, H., & Partovi, A. (2013). *Code.org vision statement*. https://code.org/es/about

Redecker, C., & Punie, Y. (2017). *European Framework for the Digital Competence of Educators (DigCompEdu)*. https://doi.org/10.2760/159770

Reimers, F., y Chung, C.K. (2016). Enseñanza y aprendizaje en el siglo XXI: metas, políticas educativas y currículo en seis países. Mejico: Fondo de Cultura Económica.

Reina, M., Gómez, L., Felizzola, H., & Hualpa, A. (2016). *Aprendizaje basado en problemas para la enseñanza de diseño y análisis de experimentos*. INGE CUC, 12(2), 86–96. https://doi.org/10.17981/ingecuc.12.2.2016.09

Resnick, M. (2017). *Lifelong Kindergarten: Cultivating Creativity through Projects, Passion, Peers, and Play*. MIT Press. https://doi.org/10.7551/mitpress/11017.001.0001

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017). *K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals*. In Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17) (pp. 182–190). Association for Computing Machinery. https://doi.org/10.1145/3105726.3106166

Rich, K. M., Binkowski, T. A., Strickland, C., & Franklin, D. (2018). *Decomposition: A K–8 computational thinking learning trajectory*. In Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18) (pp. 124–132). Association for Computing Machinery. https://doi.org/10.1145/3230977.3230979

Rich, K. M., Strickland, C., Binkowski, T. A., & Franklin, D. (2019). *A K–8 debugging learning trajectory derived from research literature*. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19) (pp. 745–751). Association for Computing Machinery. https://doi.org/10.1145/3287324.3287396

Rich, K. M., Franklin, D., Strickland, C., Isaacs, A., & Eatinger, D. (2020). A learning trajectory for variables based in computational thinking literature: Using levels of thinking to develop instruction. *Computer Science Education*, 30(4), 1–22. https://doi.org/10.1080/08993408.2020.1866938

Roman-González, M., Pérez-González, J.-C., & Jiménez-Fernández, S. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. https://doi.org/10.1016/j.chb.2016.08.047

Rosquete, D. H., Martínez, A. A., y Perozo, F. J. (2006). Codificación y Decodificación Eficiente Utilizando Códigos Hamming. Trabajo presentado en la XXII Conferencia Latinoamericana de Informática (CLEI 2006). https://clei.org/

Rycroft-Smith, L., & Connolly, C. (2019). Maths and computational thinking: Shared processes in problem-solving. *Digital Education Review, 35,* 45–58. https://doi.org/10.1344/der.2019.35.45-58

Sáez-López, J. M., Cózar-Gutiérrez, R., & González-Calero, J. A. (2023). Computational thinking and mathematics learning. *Education and Information Technologies, 28*, 3457–3475. https://doi.org/10.1007/s10639-023-11545-5

Sanders, M. (2009). STEM, STEM education, STEMmania. *Technology Teacher, 68*(4), 20–26. https://vtechworks.lib.vt.edu/

SOCHIEM (s.f.). *Kit Didáctico – Magia*. Festival de Matemática. https://festivaldematematica.org/kit-didacticos/#magia

Schneider, W. J., & McGrew, K. S. (2018). The Cattell–Horn–Carroll theory of cognitive abilities. *Contemporary intellectual assessment: Theories, tests,*

and issues (4.ª ed., pp. 73–163). The Guilford Press.https://www.researchgate.net/

Scott, C.L. (2015). El futuro del aprendizaje 2 ¿Qué tipo de aprendizaje se necesita en el siglo XXI? Investigación y Prospectiva en Educación UNESCO, París. [Documentos de Trabajo ERF, No. 14]. https://hdl.handle.net/20.500.12799/4661

Selby, C., & Woollard, J. (2013). *Computational thinking: The developing Definition*. Proceedings of the 2014 Special Interest Group on Computer Science Education (SIGCSE). https://eprints.soton.ac.uk/

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. Educational Research Review, 22, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Stanton, N. (1995). Human Cognitive Abilities: A Survey of Factor-Analytic Studies, by J. B. Carroll, Cambridge University Press, Cambridge (1993). *Ergonomics*, *38*(5). https://doi.org/10.1080/00140139508925174

Stein, M. K., & Smith, M. S. (1998). Mathematical tasks as a framework for reflection: From research to practice. *Mathematics Teaching in the Middle School, 3*(4), 268–275. https://www.jstor.org/

Swanson, H. L. (2016). *Word problem solving, working memory and serious math difficulties: Do cognitive strategies really make a difference?* Journal of Applied Research in Memory and Cognition, 5(4), 367–383. https://doi.org/10.1016/j.jarmac.2016.04.012

The National Museum of Computing. (s.f.). *Colossus*. https://www.tnmoc.org/

Tikva, C., & Tambouris, E. (2021). The CTPK-12 model. *Computers & Education, 168*, 104212. https://doi.org/10.1016/j.compedu.2021.104212

Tong, K. Y., & Trinidad, S. (2005). Conditions and constraints of sustainable innovative pedagogical practices using technology. *International Journal of Learning, 12*(5), 237–246. https://files.eric.ed.gov/

UNESCO. (2018). *ICT Competency Framework for Teachers*. https://teachertaskforce.org/

UNESCO (2021). Políticas de tecnologías de la información y comunicación (TIC) en educación: estudios sobre políticas educativas en América Latina (Documento de programa o reunión). UNESCO Biblioteca Digital. https://unesdoc.unesco.org/

UNESCO (2023). *Los futuros que construimos: Habilidades y competencias para los futuros de la educación y el trabajo*. UNESCO. https://www.unesco.org/

Vázquez Uscanga, E. A., Bottamedi, J., & Brizuela, M. L. (2019). Pensamiento computacional en el aula: el desafío en los sistemas educativos de Latinoamérica. RiiTE Revista interuniversitaria de investigación en Tecnología Educativa, (7). https://doi.org/10.6018/riite.397901

Vera Velázquez, R., Merchán García, W. A., Castro Landin, A. L., & Maldonado Zúñiga, K. (2022). *Metodología del aprendizaje basado en problemas aplicada en la enseñanza de las matemáticas*. UNESUM-Ciencias, 6(3), 127–137. https://doi.org/10.47230/unesum-ciencias.v6.n3.2022.377

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. https://doi.org/10.1007/s10639-015-9412-6

Weerdt, M. (2019). What is Pseudocode? Delft University of Technology. https://doi.org/10.4233/uuid:8af9f012-24ff-4cb0-95a3-f740ed52d047

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science

classrooms. *Journal of Science Education and Technology*. https://doi.org/10.1007/s10956-015-9581-5

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2011). Research notebook: Computational thinking—What and why? https://openlab.bmcc.cuny.edu/

World Economic Forum (2023). Future of Jobs report 2023 insight report May 2023, p6. https://es.weforum.org/

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2017). *Expanding computer science education in schools: Understanding teacher experiences and challenges*. Computer Science Education, 27(4), 235–254. https://eric.ed.gov/

Yakman, G. (2008, marzo). *STE@M Educational Model: An overview of creating a model of integrative education*. PATT-19 Conference, https://www.researchgate.net/

Yakman, G. (2011). *Introducing teaching STEAM as an educational framework in Korea*. Proceedings of the Korea STEAM Education Conference. https://www.researchgate.net/

Zapata-Ros, M. (2019). Pensamiento computacional: Una nueva alfabetización digital. *Revista de Educación a Distancia (RED), 59*. https://revistas.um.es/

Anexos

Se adjunta a continuación el enlace con las actividades resueltas de Scratch y Python para cada uno de los módulos didácticos.

Drive:

- https://drive.google.com/TareasResueltas

Cuestionario de auto reporte sobre contribuciones primarias y secundarias a los Objetivos de Desarrollo Sostenible, organizados por categorías.

En caso de que aplique, marque con una "X" un único Objetivo de Desarrollo Sostenible como aporte principal y otro objetivo como aporte secundario.

Bloques	**Objetivos**	**1°**	**2°**
Personas	1. Poner fin a la pobreza en todas sus formas y en el mundo.		
	2. Poner fin al hambre, lograr la seguridad alimentaria y la mejora de la nutrición y promover la agricultura sostenible		
	3. Garantizar una vida sana y promover el bienestar de todos y todas las edades.		
	4. Garantizar una educación inclusiva y equitativa de calidad y promover oportunidades de aprendizaje permanente para todos.	**X**	
	5. Lograr la igualdad de género y empoderar a todas las mujeres y las niñas.		
Planeta	6. Garantizar la disponibilidad y la gestión sostenible del agua y el saneamiento para todos.		
	12. Garantizar modalidades de consumo y producción sostenible.		
	13. Adoptar medidas urgentes para combatir el cambio climático y sus efectos.		
	14. Conservar y utilizar sosteniblemente los océanos, los mares y los recursos marinos para el desarrollo sostenible.		
	15. Proteger, restablecer y promover el uso sostenible de los ecosistemas terrestres, gestionar sosteniblemente los bosques, luchar contra la desertificación, detener e invertir la degradación de las tierras y detener la pérdida de biodiversidad.		
Prosperidad	7. Garantizar el acceso a una energía asequible, fiable, sostenible y moderna para todos.		
	8. Promover el crecimiento económico sostenido, inclusivo y sostenible, el empleo pleno y productivo y el trabajo decente para todos.		

	9. Construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y fomentar la innovación.		**X**
	10. Reducir la desigualdad en los países y entre ellos.		
	11. Lograr que las ciudades y los asentamientos humanos sean inclusivos, seguros, resilientes y sostenibles.		
Paz	16. Promover sociedades pacíficas e inclusivas para el desarrollo sostenible, facilitar el acceso a la justicia para todos y construir a todos los niveles institucionales eficaces e inclusivas que rindan cuentas.		
Asociaciones	17. Fortalecer los medios de implementación y revitalizar la Alianza Mundial para el Desarrollo Sostenible		