FORMACIÓN DE PROFESORADO EN PENSAMIENTO COMPUTACIONAL. GUÍA DIDÁCTICA.

Herramientas, reflexiones, aprendizajes y ejemplos prácticos resultantes del proyecto PECOFIM

pecofim.udg.edu

UdiGitalEdu

Centre de Recerca per a l'Educació Científica i Matemàtica

Universitat de Girona
Grup de Recerca en Educació Científica i Ambiental GRECA

Financiado por: AGAUR Agència de Gestió d'Ajuts Universitaris i de Recerca

“Formación de Profesorado en Pensamiento Computacional. Guía Didáctica” es un documento derivado del proyecto PECOFIM [2015 ARMIF 00031], financiado en la convocatoria d’Ajuts de Recerca per a la Millora en la Formació Inicial de Mestres, que concede la Agència de Gestió d’Ajuts de Recerca (AGAUR).

Ha sido elaborado por [**Estebanell, M.; López, V.; Peracaula, M.; Simarro, C.; Cornellà, P.; Couso, D.; González, J.; Alsina, A.; Badillo, E.; Heras, R.**].
Con la colaboración de [**Freixenet, J.; Muntaner, E.; Sabaté, F.; Serrats, L.; Córdoba, P.; Garcia, N.; Niell, M.; Bertran, A.; Bosch, D.**]
Diseño y Maquetación: [**Ferrarons, A.**].
Traducción: [**Weiss, M.**] (inglés), [**Bosch, D.**] (español).

Citar como:
[**Estebanell, M.; López, V.; Peracaula, M.; Simarro, C.; Cornellà, P.; Couso, D.; González, J.; Alsina, A.; Badillo, E.; Heras, R.**] (2018). Pensament Computacional en la formació de mestres. Guia didàctica. Girona: Servei de Publicacions UdG.

FORMACIÓN DE PROFESORADO EN PENSAMIENTO COMPUTACIONAL

Herramientas, reflexiones, aprendizajes y ejemplos prácticos resultantes del proyecto PECOFIM

El Pensamiento Computacional (PC), entendido como el conjunto de estrategias y razonamientos necesarios para la resolución de problemas y para el diseño de soluciones sistemáticas como lo haría una máquina o un ordenador, está cada vez más presente en el mundo educativo. Las actividades de programación y los robots educativos aparecen tanto en la enseñanza formal como en múltiples espacios no formales (campus, clubes de código, museos, etc.), y por todas partes afloran proyectos educativos y las redes promueven este tipo de actividades. A mismo tiempo, los currículos escolares han empezado a incluir la programación y la robótica educativa como propuestas mediante las que trabajar la resolución de problemas y ayudar a desarrollar las competencias digital o matemática. Sin embargo, ¿cuáles son las estrategias y las capacidades necesarias que se engloban dentro de lo que se denomina "Pensamiento Computacional"?, ¿cómo debe que concebirse el Pensamiento Computacional para considerarlo contenido escolar? y ¿qué puede hacer la Universidad, responsable de la formación inicial de los maestros, para promover la competencia docente necesaria en este ámbito?

Este documento, resultado del trabajo desarrollado en el proyecto PECOFIM, pretende abordar estas cuestiones.

El Proyecto PECOFIM (Pensamiento Computacional en la Formación Inicial de Maestros) ha sido financiado por el programa de Ayudas a la Investigación para la Mejora en la Formación Inicial de Maestros (ARMIF), modalidad 2, de la Agencia de Gestión de Ayudas Universitarias e Investigación de la Generalitat de Catalunya (AGAUR), cuya resolución de concesión fue el 29 de junio de 2016, y se publicó el 30 de junio de 2016. En este proyecto ha participado profesorado universitario de la Universitat de Girona y de la Universitat Autònoma de Barcelona, así como maestros y maestras de cuatro escuelas catalanas con una amplia experiencia en la realización de actividades relacionadas con la programación y con la robótica en contextos educativos.

Esta guía propone un marco didáctico sobre Pensamiento Computacional y sobre su papel en la escuela como objeto de enseñanza y aprendizaje. A partir de recursos y ejemplos prácticos, se espera contribuir a la formación del profesorado sobre el PC, abordando el qué, el cómo y el para qué del Pensamiento Computacional en la escuela.

PeCoFiM

El Pensament Computacional en la Formació inicial de Mestres

Web del projecto: pecofim.udg.edu

¿Qué os presenta este documento?

1. ¿De qué hablamos al referirnos a Pensamiento Computacional?

En 1980 Seymourt Papert introdujo el término “Pensamiento Computacional” en su libro Mindstorms: Children, computers, and powerful ideas (traducido al español como Desafío a la mente). En 1996, el mismo autor, en el artículo “An exploration in the space of mathematics educations”, expuso la idea en un contexto de aprendizaje de las matemáticas, pero no fue hasta el año 2006 cuando Jeannette M. Wing popularizó el concepto de Pensamiento Computacional en el ámbito de la investigación educativa y psicológica, con la publicación de un artículo en la revista Communications of the ACM (2006). En concreto, argumentó que:

> “el Pensamiento Computacional implica la resolución de problemas, el diseño de sistemas y la comprensión de la conducta humana, usando los conceptos fundamentales en informática.” (Wing, 2006: p 33)

La autora plantea que esta forma de pensar es aplicable a la resolución de diversos problemas, y es una habilidad fundamental para toda la población y no sólo para los informáticos y para los programadores. Desde esta perspectiva, la autora destaca la necesidad de integrar las ideas computacionales en otras disciplinas, planteando soluciones que puedan ser llevadas a cabo también por los humanos, y no sólo por las máquinas.

En los diez años posteriores a la primera publicación de Wing, muchos autores han centrado su atención en la idea del Pensamiento Computacional y han aportado definiciones complementarias. Berry (2014) y Selby & Woollard (2014) han propuesto sus propias definiciones, que, a pesar de los matices, coinciden al entender el Pensamiento Computacional como la capacidad para identificar problemas que puedan ser resueltos de una manera similar a como lo haría un programador al dar instrucciones a un ordenador usando un lenguaje de programación:

- dividir problemas complejos en módulos de menor tamaño,
- secuenciar procesos largos y complejos en pasos,
- organizar y analizar los datos reconociendo patrones lógicos,
- partir de casos concretos para llegar a situaciones abstractas y generalizables,
- utilizar algoritmos para automatizar soluciones y
- evaluar la validez de las soluciones.

PARA SABER MÁS

- *Berry, M. (2014). Computational Thinking in Primary Schools. http://milesberry.net/2014/03/computational-thinking-in-primary-schools/*
- *Selby, C. & Woollard, J. (2014) Refining and understanding Computational Thinking. https://eprints.soton.ac.uk/372410/1/372410UnderstdCT.pdf*
- *Wing, J. (2006). Computational Thinking. COMMUNICATIONS OF THE ACM. Vol. 49, No. 3. https://dl.acm.org/citation.cfm?id=1118215*

2. ¿Qué sentido tiene el Pensamiento Computacional en la escuela?

La tecnología informática, muy vinculada al Pensamiento Computacional, no es en sí misma una novedad en la escuela. En muchos centros educativos, desde hace ya muchas décadas hay un "aula de informática" integrada dentro del ecosistema escolar. Y, a pesar de haber experimentado cambios asociados al propio desarrollo tecnológico, tanto en dispositivos como en programario, a menudo se ha asociado al trabajo con el ordenador (a veces como asignatura propia, a veces integrada en otras áreas de conocimiento).

A nuestro entender, la perspectiva del Pensamiento Computacional pretende trascender esta visión tradicional del aula de informática delimitada en el espacio y en el tiempo, y entiende la actividad escolar vinculada a la programación y a la robótica como una estrategia educativa para el desarrollo de una competencia del siglo XXI que no tiene que ser una habilidad exclusiva de los profesionales vinculados a carreras STEM (Ciencia, Tecnología, Ingeniería y Matemática), sino para toda la ciudadanía. En este sentido, se presenta el Pensamiento Computacional en la escuela partiendo de tres premisas:

- Superando los conceptos computacionales: el Pensamiento Computacional entendido como un conjunto de prácticas y perspectivas.
- Superando el soporte digital: el Pensamiento Computacional entendido como una metodología para la resolución de problemas cotidianos ordenador o sin él.
- Superando la propia informática: el Pensamiento Computacional entendido como una pieza clave en el paradigma emergente STE(A)M.

2.1. Superando los conceptos computacionales: el Pensamiento Computacional entendido como un conjunto de prácticas y perspectivas

Aunque a menudo se concibe el Pensamiento Computacional como un conjunto de contenidos conceptuales a enseñar (los conceptos clave para aprender a programar, qué son los conceptos de secuencias, bucles, paralelismos, acontecimientos, condicionales, operadores y datos), diferentes propuestas han remarcado también otras dimensiones del Pensamiento Computacional, asociadas a los contenidos procedimentales y epistemológicos. Por un lado, hablamos de prácticas computacionales como aquellas actividades propias del trabajo en contextos y con apoyos computacionales, como lo son incrementar e iterar los procesos, testear y depurar los programas elaborados, reutilizar y combinar programas, abstraer y modularizar, entre otros.

Finalmente, también hablamos de perspectivas computacionales como el conjunto de actitudes y miradas a la propia disciplina, que incluyen aspectos como expresar, conectar, cuestionar y empoderarse para crear.

PARA SABER MÁS

- *Brennan, K. & Resnick, M. (2012), New frameworks for studying and assessing the development of computational thinking. Proceedings of the American Educational Research Association 2012, pp, 1-25. http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf*
- *Stephenson, C., & Barr, V. (2011). Defining Computational Thinking for K-12. CSTA:- The Voice of K-12 Computer Science and Its Educators, 7(2).*

Las dimensiones del Pensamiento Computacional según Brennan y Resnick (2012)

Conceptos

- Secuencias
- Bucles
- Paralelismos
- Acontecimientos
- Condicionales
- Operadores
- Datos

Prácticas

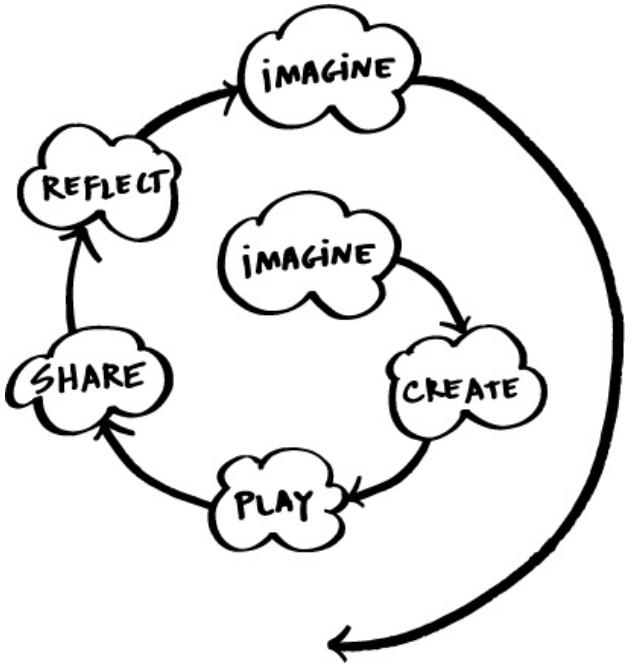
- Descomponer problemas
- Hacer cambios incrementales
- Testear y corregir
- Reutilizar
- Persistir

Perspectivas

- Expresar. Nuevos medios para crear
- Conectar. Poder crear con los otros
- Cuestionar. Poder utilizar la computación para plantearse preguntas y resolverlas.

2.2. Superando el soporte digital: el Pensamiento Computacional entendido como una metodología para la resolución de problemas cotidianos con ordenador o sin él

A pesar de que se asocie el Pensamiento Computacional al uso de tecnologías digitales, proponemos entenderlo como una manera de pensar, hacer y comunicarse que trasciende el componente digital, y que tiene múltiples aplicaciones en la resolución de problemas de todo tipo, especialmente cotidianos.



Espiral del pensamiento creativo según Resnick (2007).

http://web.media.mit.edu/~mres/papers/CC2007-handout.pdf

Así, Beauchamp (2016) destaca la estrecha relación entre el Pensamiento Computacional y el enfoque educativo del Aprendizaje Basado en Problemas, puesto que toda resolución de problemas cuenta con diferentes fases:**entender el problema, plantear un diseño, ejecutarlo y revisar el resultado**

Por otro lado, Resnick (2007) ha defendido la estrecha relación entre el Pensamiento Computacional y el Pensamiento Creativo usando el término creative computing y destaca cómo los niños aprenden en la etapa de educación infantil, donde a menudo se da un proceso de aprendizaje con un denominador común: imaginar, crear, jugar (experimentar), compartir, reflexionar y volver a imaginar (Fig.3).

Según Resnick una de las mejores maneras que tenemos de ayudar a los niños y a los jóvenes es garantizándoles la oportunidad de seguir sus intereses, de explorar sus ideas y de disponer de herramientas para hacer oír su voz. No se trata simplemente de enseñarles a utilizar la tecnología, sino de ofrecerles oportunidades de utilizar la tecnología para crear cosas.

PARA SABER MÁS

- *Beauchamp, G. (2016). ICT and computing in the primary school. In Computing and ICT in the Primary School: From pedagogy to practice (p. 252).*
- *Griffin et al. (2012). Assessment and Teaching of 21st Century Skills http://www.springer.com/us/book/9789400723238*
- *Resnick, M. (2007). All I Really Need to Know (About Creative Thinking) I Learned (By Studying How Children Learn) in Kindergarten. Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition, 1-6.*

2.3. Superando la propia informática: el Pensamiento Computacional entendido como una pieza clave en el paradigma emergente STE(A)M

La idea de Pensamiento Computacional ha ido adquiriendo un creciente interés en los últimos años en paralelo a la aparición del paradigma STEM (Ciencia, Tecnología, Ingeniería y Matemática), que surge tanto a raíz de la creciente demanda de profesionales con perfiles científico, tecnológico, ingenieril y matemático, a partir de nuevas propuestas educativas que abogan por una mayor interrelación entre estas disciplinas a lo largo de la escolarización. En paralelo, nos encontramos con la eclosión de espacios de educación maker, así como una apuesta por promover el área STEAM, donde las artes tienen un papel fundamental, y transformador, que permiten diversificar tareas y trabajos que habrá que desarrollar. En esta visión, son complementarias, pues ayudan a expresarse, a conectar y a comprender el saber objeto de estudio.

Al concebir este ensamblaje del Pensamiento Computacional con el ámbito educativo STEAM, queremos destacar tanto las oportunidades que ofrece el Pensamiento Computacional como herramienta clave en el ámbito STEAM como las oportunidades que ofrece el ámbito STEAM como contexto para desarrollar el Pensamiento Computacional.

En ese sentido, los lenguajes y los apoyos computacionales ofrecen nuevas oportunidades para confrontar a los estudiantes a los retos de carácter científico, ingenieril y matemático, como, por ejemplo, modelizar computacionalmente, simular situaciones del mundo real en entornos virtuales, representar abstracciones matemáticas, etc. De hecho, actualmente uno de los principales documentos de referencia en el ámbito STEAM, el K-12 Next Generation Science Standards (NRC, 2012), incluye el uso del Pensamiento Computacional como una de las ocho prácticas STEM clave. Además, desarrollar el Pensamiento Computacional no solo puede ayudar a los estudiantes a aprender más contenidos STEAM, sino a entender mejor el papel de la computación en el ámbito profesional STEAM y su impacto. Por ejemplo, Weintrop (2016) destaca cómo la biología computacional, basada en estructuras de datos y algoritmos, está transformando la propia manera de pensar de la biología como ciencia.

A su vez, el aprendizaje en el ámbito STEAM no solo puede enriquecerse del desarrollo del Pensamiento Computacional de los estudiantes, sino también al revés. Los contextos propios del ámbito STEAM (construcción de explicaciones científicas y modelos matemáticas, diseño de soluciones ingenieriles, etc.), son especialmente idóneos para promover y dar sentido al uso de lenguajes computacionales, y también aportan fenómenos relevantes que abordar desde la perspectiva computacional. De hecho, han sido a menudo las disciplinas STEAM las que han aportado formas de pensar y razonar de las cuales se nutre el Pensamiento Computacional, como por ejemplo el pensamiento lógico-matemático o las habilidades de indagación.

Ejemplo de campos de investigación científica emergentes basadas en el apoyo computacional.

PARA SABER MÁS

- NRC. (2012). A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas. Washington, DC.: National Academy of Sciences.
- Computer Science Teachers Association: Computational Thinking resources. *http://www.csteachers.org/page/CompThinking*
- ISTE's Computational Thinking Toolkit. *https://www.iste.org/explore/articleDetail?articleid=152*
- 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, 25(1), 127–147.

3. ¿Cuáles son los aspectos del Pensamiento Computacional que hay que promover y evaluar?

Si bien no existe una única definición de los elementos centrales o aspectos del Pensamiento Computacional, en la literatura especializada se pueden encontrar diferentes propuestas que desgranan cuáles son sus aspectos clave.

Wing, 2006	Stephenson and Barr, 2011	Selby and Woollard, 2013	Selby and Woollard, 2014
• Pensamiento recursivo y procesamiento paralelo. • Reformular un problema que inicialmente parece complicado en un problema que sepamos solucionar. • Simplificación, integración, transformación, simulación. • Elegir una representación adecuada o modelar los aspectos más relevantes de un problema para hacerlo manejable. • Interpretar los códigos como datos y los datos como códigos. • Utilizar la abstracción y la descomposición para abordar tareas amplias y complejas. • Juzgar un diseño en función de su simplicidad y elegancia.	• Formular problemas de manera que nos permita utilizar un ordenador y otros soportes para solucionarlos. • Organizar de forma lógica y analizar datos. • Representar los datos a través de abstracciones como son los modelos y las simulaciones. • Generar soluciones automáticas a través del pensamiento algorítmico (una serie de pasos ordenados). • Identificar, analizar e implementar posibles soluciones con el objetivo de conseguir la combinación más eficiente y efectiva de pasos y recursos. • Generalizar y transferir el proceso de solución de un problema a una amplia variedad de problemas.	• La habilidad de pensar de manera abstracta. • La habilidad de pensar en términos de descomposición. • La habilidad de pensar algorítmicamente. • La habilidad de pensar en términos de evaluación. • La habilidad de pensar en generalizaciones.	• Proceso mental. • Abstracción. • Descomposición. • Razonamiento heurístico. • Pensamiento lógico. • Pensamiento matemático. • Pensamiento enfocado a la ingeniería. • Diseño algorítmico. • Solucionar problemas. • Análisis. • Evaluación. • Generalización. • Recursión. • Diseño de sistemas. • Automatización. • Modelización, simulación y visualización.

Aspectos clave del Pensamiento Computacional

Basándonos en estas propuestas, presentamos un listado de aspectos clave que se deben promover y que podrían ser evaluables para ayudar a desarrollar el PC en la escuela:

- Procedimientos característicos de los lenguajes computacionales
- Estrategias para resolver problemas computacionales
- Habilidades transversales en contextos computacionales

Procedimientos característicos de los lenguajes computacionales	Estrategias para resolver problemas computacionales	Habilidades transversales en contextos computacionales
TRATAR DATOS COMPUTACIONALES	IDENTIFICAR Y DELIMITAR	CREATIVIDAD E INGENIO
REPRESENTAR ABSTRACCIONES COMPUTACIONALES	CONSIDERAR MULTIPLES VIAS	TRABAJO EN EQUIPO
AUTOMATIZAR CON ALGORITMOS COMPUTACIONALES	DESGLOSAR Y SIMPLIFICAR	AUTONOMÍA E INICIATIVA
SECUENCIAR PASOS COMPUTACIONALES	TESTEAR, VALIDAR, DEPURAR	INTERACCIÓN Y COMUNICACIÓN

PARA SABER MÁS

- Wing, J. (2006). Computational Thinking. COMMUNICATIONS OF THE ACM. Vol. 49, No. 3. https://dl.acm.org/citation.cfm?id=1118215
- Stephenson, C., & Barr, V. (2011). Defining Computational Thinking for K-12. CSTA:-The Voice of K-12 Computer Science and Its Educators, 7(2).
- Selby, C. & Woollard, J. (2013) Computational Thinking: The Developing Definition. https://eprints.soton.ac.uk/356481/
- Selby, C. & Woollard, J. (2014) Refining and understanding Computational Thinking. https://eprints.soton.ac.uk/372410/1/372410UnderstdCT.pdf

3.1. Procedimientos característicos del Pensamiento Computacional

Uno de los aspectos clave del PC es la utilización de los lenguajes computacionales. Si bien existe una gran variedad de lenguajes (lenguajes de código, por bloques, con iconos, mixtos, etc.), existen algunos denominadores comunes. Uno de ellos es **tratar de datos y variables computacionales**, entendiendo de dónde proceden, cuál es su significado y cómo pueden usarse para generar nuevos datos, así como reconocer patrones para organizar estos datos. Los retos computacionales a los que se enfrentan los estudiantes en la escuela a menudo requieren un trabajo con datos numéricos y su tratamiento sistemático, la identificación de las variables que permiten describir y controlar un proceso computacional, y la relación entre estas variables.

Por eso también planteamos que requiere desarrollar la capacidad de **representar abstracciones computacionales** de todo tipo: condiciones, patrones, relaciones geométricas, relaciones algebraicas, etc., que se expresan a través del código.

De hecho, antes de programar con código, es conveniente que los estudiantes representen sus ideas empleando papel y lápiz, así como a través de juegos vivenciales, para, posteriormente, traducirlo al lenguaje computacional.

Además, una de las maneras más comunes de representar y estructurar las relaciones entre variables y las condiciones que las determinan es a través de **automatizar algoritmos computacionales**, que permiten automatizar el funcionamiento de un programa informático o el comportamiento de un robot, usando bucles, condicionales, operadores, variables y parámetros formando un algoritmo para resolver casos particulares de un problema general.

A la vez, esto pasa por desarrollar habilidades en torno a **secuenciar pasos computacionales**, a sabiendas de ordenar de forma óptima las instrucciones que configuran un programa, usando secuencias, repeticiones o paralelismos.

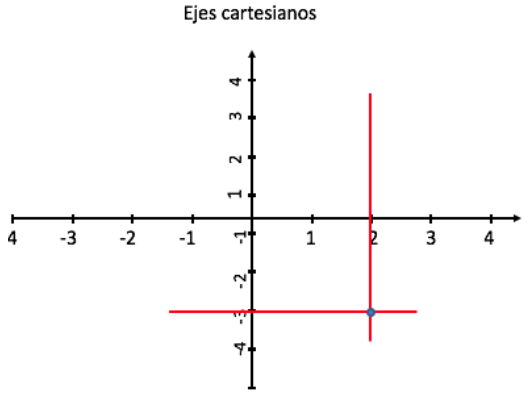
TRATAR DATOS Y VARIABLES COMPUTACIONALES

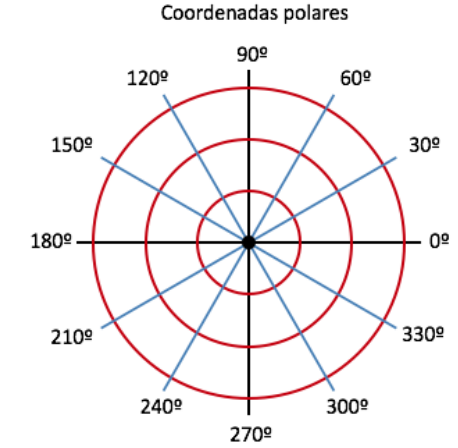
EJEMPLO PRÁCTICO

Elegimos qué coordenadas nos ayudan a definir mejor un movimiento en el plano

Cuando se propone a un grupo de niños diseñar un pequeño videojuego con Scratch, uno de los principales retos a los que se enfrentarán es definir el movimiento de los personajes. Este lenguaje permite definir este movimiento, tanto a partir de las coordenadas cartesianas (X e Y) como de las polares (dirección y desplazamiento), y según cómo quieran interactuar con los personajes, los niños tendrán que elegir qué tipo de coordenadas se pueden usar como datos en su código.

Por ejemplo, si se quiere definir el movimiento de un personaje que choca contra los extremos de la pantalla, el dato que les será más útil es la dirección expresada en grados, mientras que, si quieren definir la caída vertical de un personaje, el dato que les será más útil será la coordenada Y.





REPRESENTAR ABSTRACCIONES COMPUTACIONALES

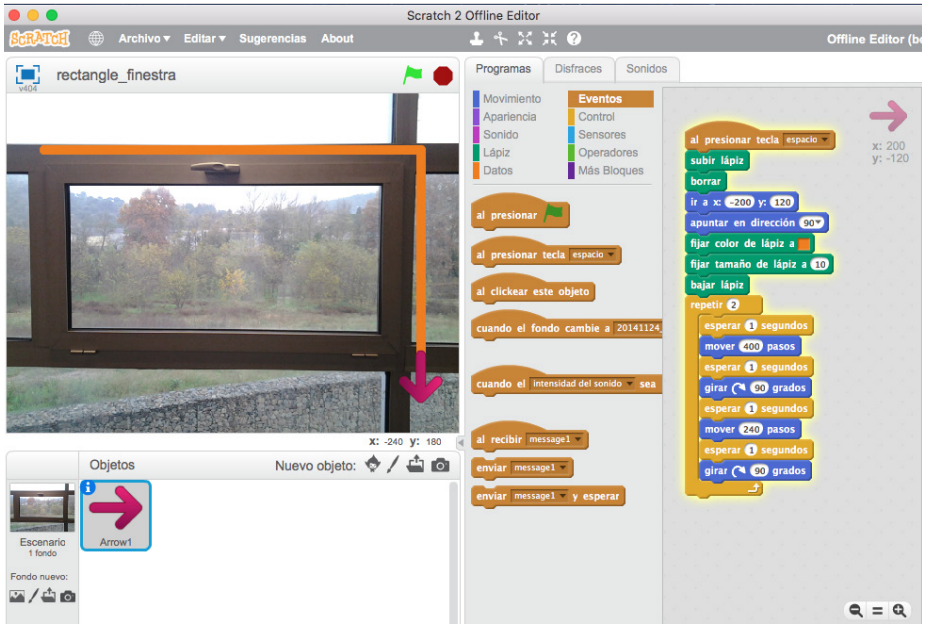
EJEMPLO PRÁCTICO

Reproducimos los polígonos que vemos a nuestro alrededor

Podemos proponer a los niños reconocer formas geométricas planas en su entorno físico, para que las fotografíen y las lleguen a describir matemáticamente con la ayuda del entorno de programación Scratch.

Para hacerlo, pueden colocar una de las fotografías como escenario de fondo, y programar un personaje para que, con un clic, siga el contorno de la forma geométrica que aparece.

Esto requiere un proceso de abstracción en el que entran en juego datos y variables como la longitud de los lados del polígono y el ángulo que tiene que girar el personaje en cada vértice.



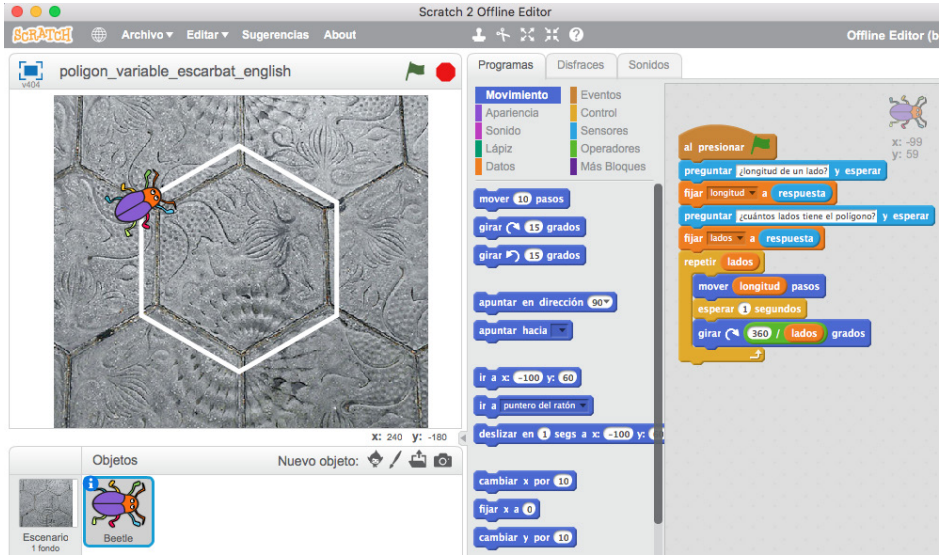
AUTOMATIZAR CON ALGORITMOS COMPUTACIONALES

EJEMPLO PRÁCTICO

Polígonos regulares: Generalizamos su expresión matemática

Continuando con el ejemplo anterior, si la forma es un polígono regular, los niños pueden llegar a encontrar la relación entre el ángulo de giro y el número de lados del polígono, relacionando los dos parámetros con un operador, que en este caso será la división.

Así podrán construir un algoritmo general para dibujar el contorno de cualquier polígono regular a través de la relación que establece que cada ángulo de giro es 360 dividido por el número de lados del polígono.



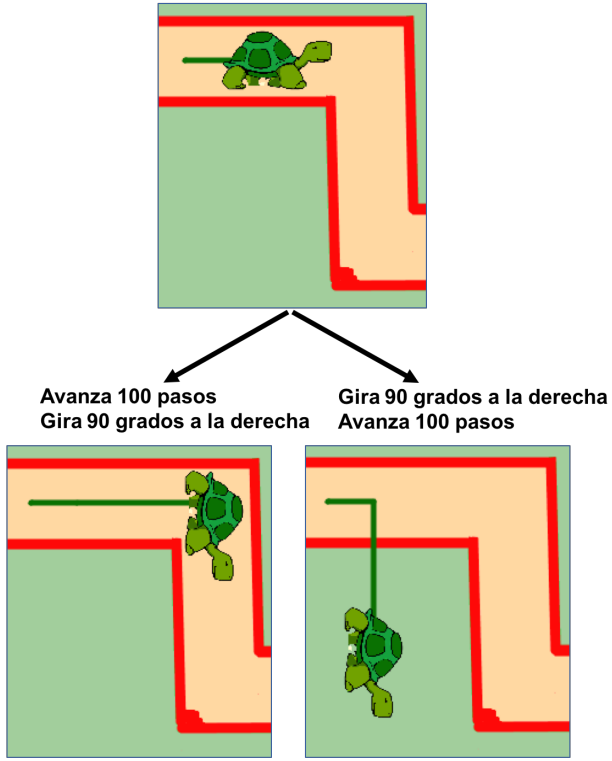
SECUENCIAR PASOS COMPUTACIONALES

EJEMPLO PRÁCTICO

Descubrimos que las instrucciones que se dan a un robot no cumplen la propiedad conmutativa

Si se propone a un grupo de niños programar un robot, o un personaje que tenga que seguir un itinerario concreto, estos tendrán que definir una secuencia de órdenes o instrucciones que este personaje tendrá que seguir.

Si las dos instrucciones de que se dispone son “Avanzar 100 pasos” (movimiento de traslación) y “Girar 90º a la derecha” (movimiento de rotación), los niños pueden comprobar que la orden con que se coloca la secuencia de instrucciones no lleva al mismo resultado, y que, por lo tanto, las instrucciones dadas al robot no cumplen la propiedad conmutativa.



3.2. Estrategias para resolver problemas computacionales

En paralelo al dominio de los elementos clave del dominio de los lenguajes computacionales (sus componentes, sus estructuras, reglas, etc.), el PC tiene una estrecha relación con el modo que en las personas nos enfrentamos a la resolución de problemas.

Las estrategias de resolución de problemas, de hecho, han sido ampliamente discutidas en la literatura educativa bajo diferentes paraguas (habilidades de resolución de problemas, pensamiento crítico, etc.). Pero las habilidades para desarrollar estas estrategias no pueden quedar desligadas de los contextos en que se aplican, y, por lo tanto, adquieren un sentido muy particular cuando se trata de enfrentarse y resolver problemas computacionales.

Así, el Pensamiento Computacional implica la capacidad para **identificar y delimitar** un problema computacional por resolver, analizando sus posibles soluciones, optimizando los pasos y recursos.

Cuando se hace, hay que **considerar múltiples vías**, entendiendo, también, que puede haber varios caminos o soluciones válidas. La resolución de problemas pasa también por la capacidad de **desglosar y simplificar,** modularizando y fragmentando situaciones complejas en otras más sencillas.

Finalmente, también hay que **testear, validar y depurar** las soluciones de forma iterativa, aprendiendo de los errores identificados, puesto que, en computación, depurar implica buscar el error entendiendo qué es lo que no funciona.

IDENTIFICAR Y DELIMITAR

EJEMPLO PRÁCTICO

Nos preguntamos qué tenemos y qué necesitamos antes de empezar a programar un robot educativo programable (Bee-Bot, Cubeto, otros)

Cuando un grupo de niños se encuentran ante el reto de programar un robot que tiene que desplazarse desde un lugar a otro, es conveniente que previamente reflexionen sobre cuál es el problema al que se enfrentan. Por ejemplo, no es lo mismo definir un desplazamiento a partir de un punto de salida y un punto de llegada que llegar a un punto partiendo de uno de salida y habiendo hecho un desplazamiento.

Es necesario, por lo tanto, delimitar cuál es exactamente el reto que tienen que abordar, e identificar qué posibles maneras hay de resolverlo.

CONSIDERAR MÚLTIPLES VÍAS

EJEMPLO PRÁCTICO

Construimos una torre con vasos (I)

Una actividad muy común para reflexionar sobre la necesidad del lenguaje computacional para dar instrucciones claras a un robot es emular la programación de un brazo robótico que tiene que construir una torre de vasos con una estructura dada.

Por equipos, algunos niños o jóvenes pueden convertirse en programadores, teniendo que escribir el programa que permitirá al brazo robótico construir la estructura, utilizando un conjunto de símbolos previamente pactado (que indiquen moverse a derecha, izquierda, arriba, abajo, girar el vaso, etc.).

Una vez hayan escrito el programa, otro grupo de niños o jóvenes que no haya participado en la escritura, ejecutará las instrucciones construyendo la torre de manera real con vasos físicos. Así, los programadores discutirán y reflexionarán sobre las diferentes maneras en que podrían construir la torre, en qué orden colocar los diferentes vasos y las ventajas de cada opción.

TESTEAR, VALIDAR, DEPURAR

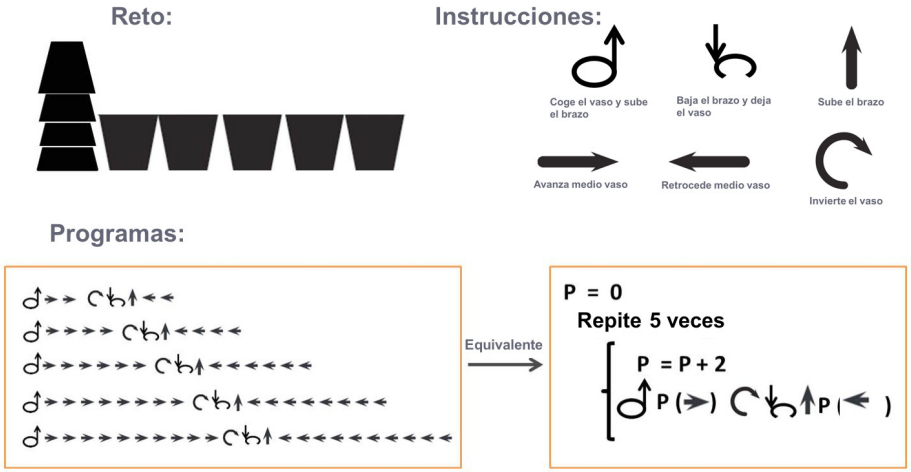
EJEMPLO PRÁCTICO

Construimos una torre con vasos (II)

En la actividad descrita en el ejemplo anterior, los niños que escriben las instrucciones para construir la torre testean, validan y corrigen la disposición de los símbolos de manera iterativa para conseguir la estructura solicitada.

Una vez realizado, pueden optimizar la solución, en un proceso de abstracción, introduciendo conceptos computacionales como bucles y variables.

Una posible actividad para trabajar el “depurar”, podría ser la de proveer un ejemplo de código que no hace lo que tendría que hacer, y hacer analizar el código a los niños para buscar el error. De este modo, el ejercicio les obliga a entender por qué no funciona un programa y a corregirlo para conseguir una posible solución.



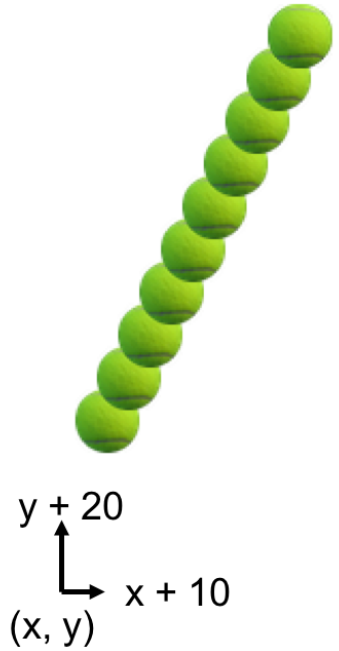
DESGLOSAR Y SIMPLIFICAR

EJEMPLOS PRÀCTICOS

Descomponemos un movimiento complejo en dos más simples

A menudo los niños quieren reproducir computacionalmente comportamientos o fenómenos complejos, que requieren, previamente, de una simplificación a partir de transformar un reto grande en varios más pequeños.

Pensamos, por ejemplo, en la composición de los movimientos. Un movimiento complejo puede estar formado por varios más simples, como, por ejemplo, un movimiento diagonal puede estar formado por uno vertical y uno horizontal.



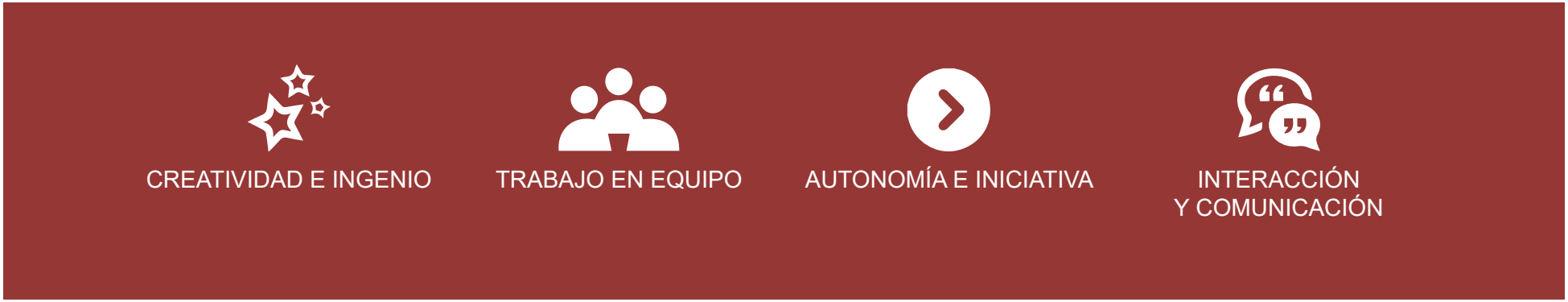
Repetir 10 veces:
deslizar en 1 segundo
hasta x = x + 10
hasta y = y + 20

3.3. Habilidades transversales en contextos computacionales

En los últimos años se han ofrecido múltiples propuestas de formulación de las competencias transversales o las competencias del siglo XXI, que engloban nociones clásicas de HOTS (High Order Thinking Skills), las también llamadas soft skills, y otros aspectos para tener en cuenta las demandas de la dinámica, global y profundamente digital de la sociedad actual. Igual como sucede con la resolución de problemas, la literatura cognitiva parece indicar que toda competencia se desarrolla y se expresa de una forma determinada según el contexto; por lo tanto, la cuestión no es tanto cuáles son las habilidades transversales en abstracto, o desvinculadas de los contenidos escolares, sino qué sentido adquieren en contextos computacionales como los que se discuten en este documento.

Así, hablamos de habilidades transversales como la **creatividad, el trabajo en equipo, la autonomía e iniciativa** personales o la **interacción y la comunicación** no como competencias transversales a desarrollar aparte del Pensamiento Computacional, sino dentro del Pensamiento Computacional.

Dicho de otra forma: no hay que poner el foco en enseñar la creatividad o el trabajo en equipo en sí mismo, sino conseguir enseñar computación desarrollando la creatividad y el trabajo en equipo.



CREATIVIDAD E INGENIO

EJEMPLO PRÁCTICO

Pensamos soluciones ingeniosas para hacer que el robot siga una línea

Cuando un grupo de niños disponen de un robot con ruedas y un sensor de luz, que debe moverse siguiendo una línea de color dibujada en el suelo, existen diferentes maneras creativas de enfrentarse al reto.

Por ejemplo, pueden situar el robot en el margen de la línea (frontera entre oscuro y claro) de forma que el robot avance en pequeños tramos, haciendo un giro hacia el color oscuro cuando el sensor detecta claridad y haciendo un giro hacia el color claro cuando el sensor detecta oscuridad. Para hacer cada giro sólo se mueve una de las dos ruedas y se avanza, en lugar de estar moviendo las dos ruedas en línea recta, moviendo alternativamente cada una de ellas.

Para trabajar la creatividad es importante diseñar los talleres donde se tengan que utilizar diferentes materiales, dejar que los niños expresen su voz, dejar que sigan sus intereses. Que inventen poemas, historias, robots y juegos a partir de lo que les guste, y que aprendan a reutilizar y mezclar proyectos y soluciones, así como que expresen libremente sus ideas.

TRABAJO EN EQUIPO

EJEMPLO PRÁCTICO

Creamos una historia multimedia colaborativa

El desarrollo de programas computacionales a nivel profesional requiere a menudo del trabajo en equipo, y este se puede reproducir de manera análoga dentro del aula.

Por ejemplo, proponiendo a un grupo de niños el reto de crear una historia trabajando de manera colaborativa. Inicialmente, el argumento general de la historia es consensuado entre toda la clase, pero después los niños se dividen en grupos y cada uno de estos grupos crea una parte, realizando un guion y programando su animación digital mediante Scratch.

Para visualizar el resultado final se sitúan los ordenadores en fila, poniendo en marcha su programa todos al mismo tiempo de forma que haya personajes que, aparentemente, salten de un ordenador al otro. La coordinación entre grupos tiene que permitir tanto la coherencia argumentativa de las partes de la historia como la coherencia de los programas, garantizando la sincronización de acontecimientos a partir del acuerdo de los tiempos a transcurre desde el comienzo de la ejecución.

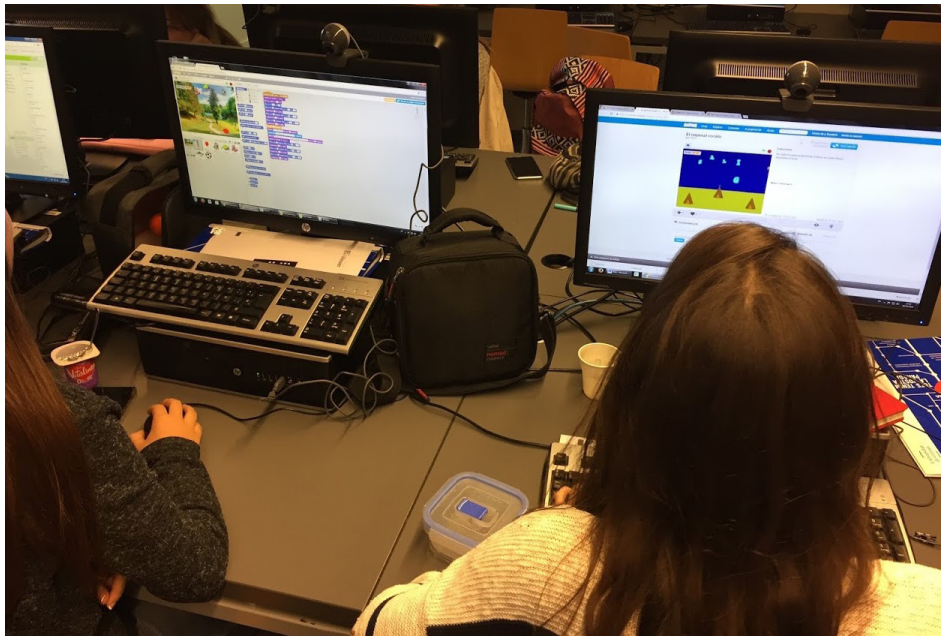
AUTONOMÍA E INICIATIVA

EJEMPLO PRÁCTICO

Decidimos hasta qué grado de dificultad queremos llegar cuando diseñamos un videojuego

Con el objetivo final de que cada grupo de trabajo cree un videojuego al que podamos jugar juntos, se presentan diferentes herramientas de creación de videojuegos que van desde las más simples a las más complejas: Twine, Scratch, Kodu, Stencyl y Unity. La curva de aprendizaje de cada una de las aplicaciones es diferente. Esto quiere decir que, si un grupo elige desarrollar su proyecto con una herramienta que comporta cierta complejidad, tendrá que destinar una parte importante de su tiempo a aprender, por su cuenta, el funcionamiento de la herramienta.

La experiencia nos dice que aproximadamente un tercio de los grupos decide desarrollar el videojuego con una herramienta que les exigirá un grado elevado de autonomía para encontrar soluciones en los problemas que se les irán planteando a lo largo del desarrollo de su proyecto.



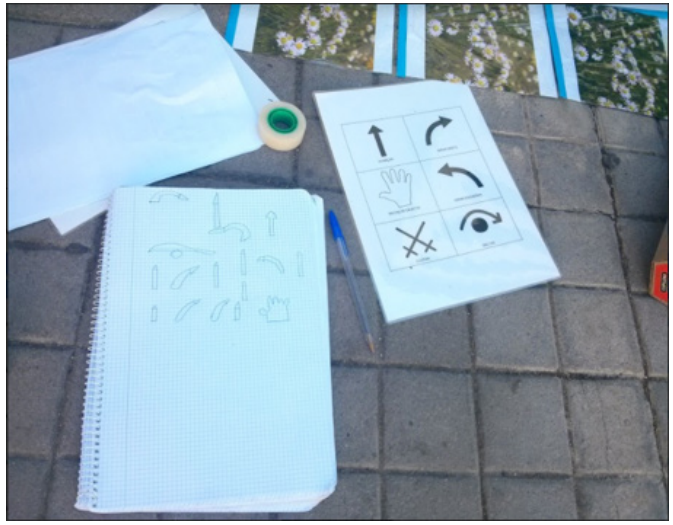
INTERACCIÓN Y COMUNICACIÓN

EJEMPLO PRÁCTICO

Pensamos como tenemos que dar instrucciones claras y comprensibles al Niño-robot

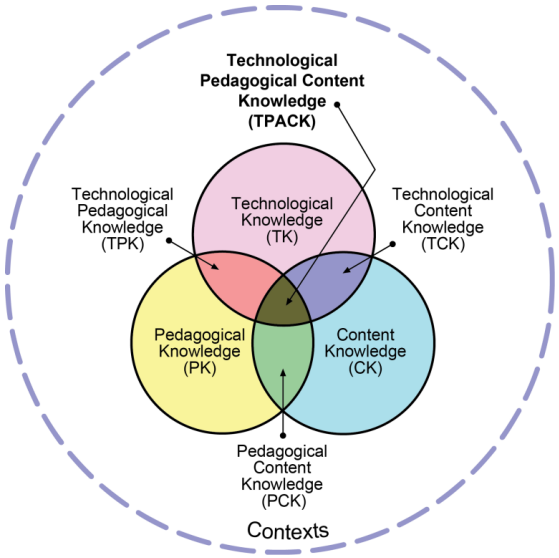
En general, en muchos talleres y proyectos que se desarrollen en el aula puede ser valioso que, en un momento determinado, uno de los niños pueda explicar a sus compañeros un hallazgo que haya conseguido, o al final del taller presentar el resultado y el proceso a la clase o escuela, documentar los trabajos, o al acabar un taller hacerles escribir a los niños “consejos” que darían a un compañero si tuvieran que volver a hacer el taller.

Por ejemplo, en una actividad llamada “Niño-robot”, los propios niños pueden fabricar los códigos para dar las instrucciones al niño-robot en lo que se puede denominar el lenguaje de los robots, un conjunto de símbolos que ellos consensuarán. Los niños que lo programen tendrán que poder comunicarse con los compañeros que hagan de robot dándoles las instrucciones de manera clara y comprensible.



4. ¿Cómo abordar el Pensamiento Computacional en la formación de maestros/as?

Debido a que se trata de un campo bastante emergente, el Pensamiento Computacional tiene ahora mismo un papel poco relevante en la formación inicial de maestros y algunos estudios han identificado esta carencia (Bower & Falkner, 2015).
Abordar el Pensamiento Computacional en la formación inicial de maestros puede concebirse como una integración de lo que se ha denominado el conocimiento didáctico-tecnológico del contenido (Shulman, 2005), normalmente llamado TPCK por sus siglas en inglés (también conocido como modelo TPACK, Technological Pedagogical and Content Knowledge). EL TPCK se puede entender como "la capacidad del profesorado para transformar su conocimiento del contenido en formas que sean didácticamente poderosas y aun así adaptadas a la variedad que presentan sus alumnos en cuanto a habilidades y bagajes".



Modelo TPCK, o TPACK, Technological Pedagogical and Content Knowledge (Koehler, Mishra y Cain, 2013)

LICENCIA: "Reproduced by permission of the publisher, © 2012 by tpack.org"

Tal como sucede con otros procesos de formación del profesorado, esta formación debe entenderse a diferentes niveles, los cuales se combinan y se interrelacionan a lo largo de la progresión de aprendizaje que hacen a lo largo de su carrera.

Si el objetivo final de la formación de profesorado en Pensamiento Computacional es que estos sean capaces de diseñar situaciones de aprendizaje con sus futuros alumnos, lo primero que hace falta es que ellos experimenten en primera persona el aprendizaje y desarrollo de su propio Pensamiento Computacional: deben conocer las herramientas que existen, los lenguajes computacionales, el desarrollo de estrategias de razonamiento y resolución de problemas, etc. Esto solo es posible si los futuros maestros se enfrentan a la resolución de problemas computacionales como estudiantes/aprendices: programar un robot, diseñar un videojuego o historia interactiva, desarrollar una app, o teniendo que resolver problemas computacionales unplugged. Por ello decimos que un primer nivel necesario en la formación de maestros es el **nivel usuario**. En este nivel formativo no se busca que el futuro maestro se plantee todavía cómo ayudar a otra persona (o a sus potenciales alumnos) a desarrollar el Pensamiento Computacional, sino que se plantee cuestiones como, por ejemplo, cómo usar un lenguaje computacional determinado para lograr algunos de los hitos propuestos (con un robot, en un videojuego, en una app, etc.).

Solo cuando una persona ha experimentado el hecho de enfrentarse a un reto computacional (diseñando un pequeño programa, elaborando un código, etc.), es posible hacer una reflexión sobre qué ha tenido que hacer para lograr finalizarlo.

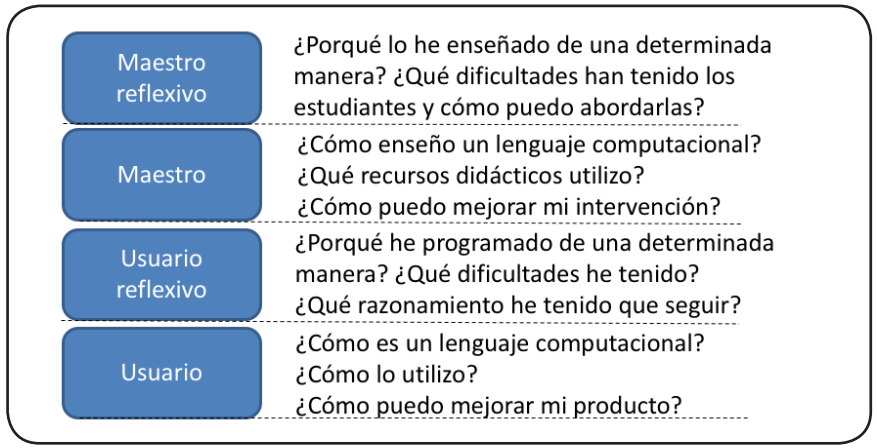
PARA SABER MÁS

- Bower, M. i Falkner, F. (2015) Computational Thinking, the Notional Machine, Pre-service Teachers, and Research Opportunities. Proceedings of the 17th Australasian Computing Education Conference, 27-30.
- Koehler, M. J., Mishra, P., i Cain, W. (2013). What is technological pedagogical content knowledge (TPACK)? Journal of Education, 193(3), 29-37
- Koehler, M. J., Mishra, P., i Cain, W. (2015). ¿Qué son los Saberes Tecnológicos y Pedagógicos del Contenido (TPACK)?. Virtualidad, Educación y Ciencia, 10 (6), pp. 9-23. *http://www.punyamishra.com/wp-content/uploads/2016/08/11552-30402-1-SM.pdf*

Entendemos, por lo tanto, el **nivel de usuario reflexivo** como aquel donde el sujeto está reflexionando sobre lo que ha hecho cuando ha tenido que enfrentarse a un reto computacional: ¿qué razonamiento ha realizado?, ¿qué elementos del lenguaje computacional ha usado y por qué? Este es un ejercicio de metacognición en el que toman especial relevancia los 12 aspectos claves discutidos en el apartado anterior, puesto que permiten reflexionar de forma estructurada sobre las propias actuaciones.

Después de haber reflexionado como usuario/a tiene sentido situar a los futuros maestros ante el **reto de hacer de maestros** propiamente dicho, es decir, de decidir qué quieren enseñar, qué esperan que sus estudiantes aprendan sobre Pensamiento Computacional, qué recursos y estrategias cabe seleccionar, etc. En la formación inicial de maestros existen diferentes momentos en los que promover este proceso: en los prácticums y en las estancias en la escuela, en actividades de colaboración universidad-escuela (visitas, talleres, ferias, etc.), pero también haciendo que los futuros maestros simulen breves situaciones de aula entre iguales, las llamadas experencias de microteaching, donde unos adoptan el rol de maestros y el resto el de alumnos.

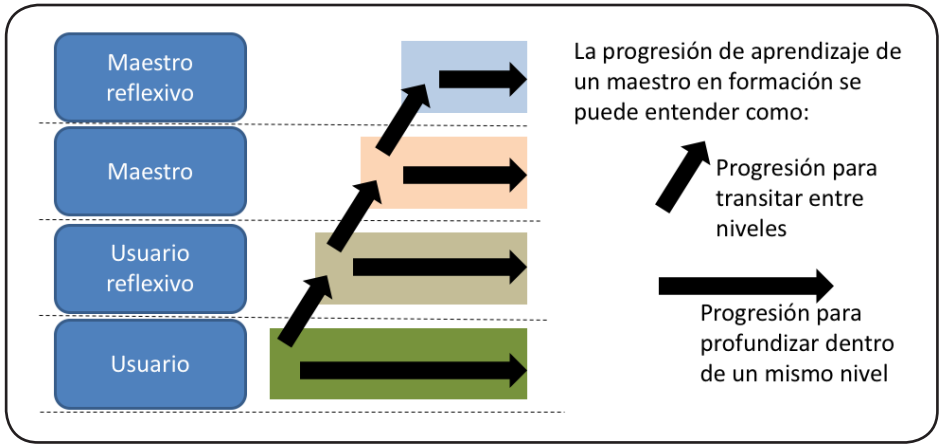
Finalmente, solo después de haber experimentado en primera persona y haber actuado como maestro es posible reflexionar sobre el proceso de enseñanza y aprendizaje relacionado con el Pensamiento Computacional, es decir, actuar como maestro **reflexivo**. Esto pasa por preguntarse qué y cómo enseñar, pero también por qué hacerlo, analizar las dificultades y casuística de cada contexto educativo y las estrategias para abordarlas.



Niveles de progresión en el aprendizaje del Pensamiento Computacional de los futuros maestros.

La propuesta de estos cuatro niveles clave sirve para orientar la progresión de aprendizaje en Pensamiento Computacional de los futuros maestros. Por un lado, esta progresión puede entenderse como la profundización progresiva dentro de un mismo nivel. Por ejemplo, en el nivel usuario, la formación de maestros puede estar centrada en aprender lenguajes computacionales y estrategias de resolución de problemas computacionales cada vez más complejos. En el caso de la robótica, un posible ejemplo podría ser introducir los robots fácilmente programables (ejemplo Bee-Bot https://www.bee-bot.us/), seguir con los robots construibles (ejemplo Lego WeDo https://education.lego.com/) y, posteriormente, continuar con los robots programables (ejemplos: Edison https://meetedison.com/, mbots https://www.makeblock.com/steam-kits/mbot y/o los Lego Mindstorms https://education.lego.com/).

A la vez, la progresión puede entenderse como el tránsito entre niveles, de tal modo que los maestros en formación sean primero usuarios, después usuarios reflexivos, después maestros y finalmente maestros reflexivos. Así, por ejemplo, en una secuencia formativa sería conveniente proponerles primero hacer un pequeño videojuego con Scratch, después reflexionar sobre qué han aprendido, después llevar a cabo un taller (con otros maestros en formación o con estudiantes) y, finalmente, reflexionar sobre qué han aprendido al tener que enseñarlo, qué dificultades han presentado sus “alumnos” y cómo les podrían haber ayudado a superarlas y/o evitarlas.



Etapas en la progresión de la experiencia en Pensamiento Computacional de los maestros.

Esta doble manera de entender la progresión de aprendizaje en la formación de maestros (siendo cada vez más experto/a en cada nivel y siendo capaz de progresar al siguiente nivel) permite situar cada actividad formativa, y así trazar itinerarios que puedan ser aplicables a otros contextos de formación del profesorado.

EJEMPLO PRÁCTICO:

Taller de diseño de un videojuego para aprender a programar a partir de pequeños retos

NIVEL USUARIO

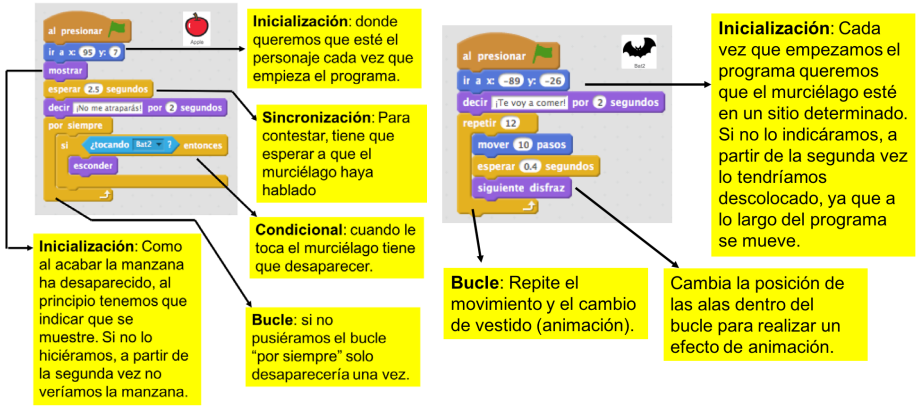
Para aprender y dominar el lenguaje de programación Stencyl, los futuros maestros tienen que crear un videojuego que se base en el conocido videojuego Galaga. La secuencia formativa empieza con una breve introducción sobre el funcionamiento básico de la herramienta: la interfaz, las reglas de funcionamiento, las piezas de código, etc. Una vez presentado este lenguaje de programación, se van presentando a los participantes pequeños retos que tienen que ir resolviendo, con dificultad creciente. Periódicamente se hacen puestas en común para resolver los retos que han ido surgiendo. Al final del proceso los participantes habrán adquirido un conocimiento bastante sólido sobre programación por bloques que les permitirá tanto profundizar en el futuro con lenguajes de programación más sofisticados como, por ejemplo, empezar a preguntarse cómo ayudar a sus futuros alumnos a realizar el aprendizaje que ellos han logrado.

EJEMPLO PRÁCTICO:

Actividad de análisis de un programa después de haberlo diseñado

NIVEL USUARIO REFLEXIVO

A un grupo de maestros en formación que ya ha participado anteriormente en una actividad donde han tenido que hacer un pequeño programa informático (por ejemplo, con Scratch), se les puede plantear que piensen por qué han seguido una serie de pasos, que comparen si todos los grupos lo han hecho de la misma forma, que piensen que podrían haber realizado de otra manera, etc. Este tipo de cuestiones les conducen a plantearse no sólo qué han hecho, sino qué ideas hay detrás de sus decisiones: qué conceptos, qué prácticas y qué perspectivas. En el siguiente ejemplo, se pide a los participantes que justifiquen por qué han programado de una determinada manera, explicando qué hubiera pasado si no hubieran colocado cada una de las piezas del bloque de programación.



EJEMPLO PRÁCTICO:

Microteaching entre iguales sobre pensamiento computacional

NIVEL MAESTRO

Después de haber conocido el lenguaje de programación Scratch (nivel usuario) y después de haber reflexionado sobre qué aspectos del Pensamiento Computacional se trabajan cuando los niños crean una pequeña aplicación a través de este lenguaje (nivel usuario reflexivo), se propone a algunos de los estudiantes del grado de maestros diseñar e implementar una pequeña intervención de 20 minutos ante sus compañeros, para enseñarles algún aspecto particular de Scratch. Así, los participantes encargados de hacer esta intervención, que denominamos microteaching, tienen que pensar cuáles son los objetivos de aprendizaje que se plantean, qué pasos seguirán, como gestionarán el trabajo del grupo, qué ejemplos pondrán, etc., Esto permitirá, posteriormente, reflexionar sobre qué ha funcionado mejor y qué se podría mejorar de su intervención.

EJEMPLO PRÁCTICO:

Valoración posterior a la realización de actividades de Pensamiento Computacional en una intervención con escolares fuera del aula

NIVEL MAESTRO REFLEXIVO

En un contexto de actuación con niños reales, un grupo de estudiantes en formación puede implementar propuestas diseñadas por ellos mismos para ser desarrolladas con niños y niñas, por ejemplo, empleando algún robot o siendo los mismos niños los que actúen recibiendo órdenes como si fueran robots. Estos estudiantes tendrían el reto no solo de llevar a cabo la actividad en sí, sino de ir recogiendo evidencias del aprendizaje realizado por los niños que hayan participado en sus propuestas: conversaciones entre iguales, situaciones críticas, etc. Una vez recogidas estas evidencias, los maestros en formación se reunirán, de nuevo en la universidad, para comentar su experiencia. A través de las evidencias recogidas y de una tabla de aspectos clave del Pensamiento Computacional (ver sección anterior) los participantes discutirán sobre qué tipo de aprendizajes pueden haber promovido entre los niños que participaban en el taller que ellos hayan preparado, consiguiendo, así, hacer una reflexión sobre su práctica.

PARA SABER MÁS

- BBC Bitesize *https://www.bbc.com/education/guides/zp92mp3/revision*
- Bocconi et al., (2016). Developing Computational Thinking in Compulsory Education. Implications for policy and practice. JCR Science for Policy Report. European Comission. *http://publications.jrc.ec.europa.eu/repository/bitstream/JRC104188/jrc104188_computhinkreport.pdf*
- CAS Barefoot *https://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking-further-examples/*
- Computing at School resources: *https://www.computingatschool.org.uk/*
- Computer Science Without a Computer: *https://csunplugged.org/en/*, *http://csedweek.org/unplugged/thinkersmith*
- Google Education Computational Thinking microsite *https://edu.google.com/resources/programs/exploring-computational-thinking/*
- Google CT for Educators *https://computationalthinkingcourse.withgoogle.com/course?use_last_location=true*
- Kiki Computational Thinking Games *http://games.thinkingmyself.com/*

Antes de acabar, algunos consejos prácticos:

- **No hay que saberlo todo antes de empezar.** Sin duda, lo primero que hay que aprender es que no hay que saberlo todo sobre una determinada herramienta, tener respuestas capaces de resolver todos los problemas antes de enfrentarse a ellos. Hay que afrontar un reto o problema y plantear y desarrollar una posible solución. Pero es evidente que cuando este reto o problema se desarrolle en un aula habrá múltiples ideas y caminos que los estudiantes explorarán y, ante sus dudas, el maestro tendrá que aprender a responder “explora, investiga”; a plantear preguntas que les obliguen a pensar, que les orienten hacia descubrir y solucionar las pequeñas trabas con las que se puedan ir encontrando.

- **Conviene plantear retos colaborativos.** Es fundamental que los estudiantes trabajen en equipo, y aprendan a colaborar, comunicar, y gestionar problemas y soluciones en el marco del trabajo colaborativo.

- **Hay que tener en cuenta los espacios donde desarrollar las actividades.** Es muy importante que los maestros/futuros maestros tomen conciencia de los espacios donde se realizarán los talleres/actividades. Es clave para el éxito del taller que los espacios sean adecuados para trabajar en equipo, que inviten a compartir proyectos e ideas, y que el equipamiento resulte funcional.

- **El equipamiento tienen que estar a punto antes de empezar la actividad.** Es importante haber probado los equipos y tecnologías antes de iniciar la sesión de trabajo con los niños y las niñas, y que, cuando empiecen el taller, se encuentren los materiales a punto; así ahorraremos problemas relacionados con la tecnología (“no hay wifi”, “no funciona la wifi”, “no hay bluetooth”, “este plug-in no está instalado”, “no funciona con este navegador”...).