



Universidad  
Rey Juan Carlos

**Gíua docente: uso de Dr. Scratch  
para el desarrollo del pensamiento  
computacional**

**Gregorio Robles, Jesús Moreno, María Luz  
Aguado, Eva Hu**

# Índice

---

<b>1. Introducción</b>	<b>3</b>
<b>2. Usando Dr. Scratch</b>	<b>5</b>
<b>3. Desarrollando los distintos aspectos del pensamiento computacional con Dr. Scratch</b>	<b>6</b>
3.1. Abstracción y descomposición de problemas . . . . .	6
3.1.1. Si has sacado 0 puntos... . . . .	6
3.1.2. Si has sacado 1 punto... . . . .	7
3.1.3. Si has sacado 2 puntos... . . . .	8
3.2. Nociones de algoritmia y control del flujo de los programas . . . . .	9
3.2.1. Si has sacado 0 puntos... . . . .	9
3.2.2. Si has sacado 1 punto... . . . .	10
3.2.3. Si has sacado 2 puntos... . . . .	11
3.3. Pensamiento lógico . . . . .	11
3.3.1. Si has sacado 0 puntos... . . . .	11
3.3.2. Si has sacado 1 punto... . . . .	12
3.3.3. Si has sacado 2 puntos... . . . .	12
3.4. Paralelismo . . . . .	13
3.4.1. Si has sacado 0 puntos... . . . .	13
3.4.2. Si has sacado 1 punto... . . . .	13
3.4.3. Si has sacado 2 puntos... . . . .	14
3.5. Sincronización . . . . .	15
3.6. Representación de la información . . . . .	15
3.7. Interactividad con el usuario . . . . .	15
<b>4. Promoviendo buenos hábitos de programación con Dr. Scratch</b>	<b>15</b>
4.1. Nombrado de objetos . . . . .	15
4.2. Código muerto . . . . .	15
4.3. Inicialización de los atributos de los personajes . . . . .	15
4.4. Repetición de código . . . . .	15
<b>5. Conclusiones finales</b>	<b>15</b>

# 1 INTRODUCCIÓN

En los últimos años estamos presenciando un movimiento global que defiende que el pensamiento computacional debería ser incluido en la formación de todos los niños y niñas, ya que no solo representa un ingrediente vital del aprendizaje de la ciencia, la tecnología, la ingeniería y las matemáticas, sino que es una competencia fundamental para una vida plena en la sociedad digital hacia la que nos dirigimos.

## PERO, ¿QUÉ ES EL PENSAMIENTO COMPUTACIONAL?

Jeannette Wing, en su artículo *Computational Thinking*<sup>1</sup> establece que el “pensamiento computacional implica resolver problemas, diseñar sistemas y comprender el comportamiento humano, haciendo uso de los conceptos fundamentales de la informática”. Es decir, que la esencia del pensamiento computacional es pensar como lo haría un informático cuando nos enfrentamos a un problema, de manera que podamos aprovechar la potencia de los ordenadores para resolverlos.

Otras definiciones de pensamiento computacional han ido surgiendo en la literatura científica desde entonces. Entre las más aceptadas se encuentran la de Aho<sup>2</sup> y la de la Royal Society<sup>3</sup>:

- El pensamiento computacional es el proceso que permite formular problemas de forma que sus soluciones pueden ser representadas como secuencias de instrucciones y algoritmos.
- El pensamiento computacional es el proceso de reconocimiento de aspectos de la informática en el mundo que nos rodea, y aplicar herramientas y técnicas de la informática para comprender y razonar sobre los sistemas y procesos tanto naturales como artificiales.

Una iniciativa muy interesante en relación a la definición del pensamiento computacional es la promovida por la Sociedad Internacional de la Tecnología en la Educación (ISTE) y la Asociación de Profesores de Informática (CSTA), que han colaborado con líderes del mundo de la investigación y la educación superior, la industria y la educación primaria y secundaria para desarrollar una definición operativa<sup>4</sup> que describa con precisión sus características esenciales y ofrezca un marco de trabajo y un vocabulario común con el que los profesionales de la educación puedan trabajar.

Según esta definición operativa, el pensamiento computacional es un proceso de resolución de problemas que incluye las siguientes características:

<sup>1</sup><https://www.cs.cmu.edu/CompThink/papers/Wing06.pdf>

<sup>2</sup><http://comjnl.oxfordjournals.org/content/55/7/832.abstract>

<sup>3</sup><https://royalsociety.org/education/policy/computing-in-schools/report/>

<sup>4</sup><http://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf>

- Formular problemas de forma que se permita el uso de un ordenador y otras herramientas para ayudar a resolverlos.
- Organizar y analizar lógicamente la información.
- Representar la información a través de abstracciones como los modelos y las simulaciones.
- Automatizar soluciones haciendo uso del pensamiento algorítmico (estableciendo una serie de pasos ordenados para llegar a la solución).
- Identificar, analizar e implementar posibles soluciones con el objetivo de lograr la combinación más efectiva y eficiente de pasos y recursos.
- Generalizar y transferir este proceso de resolución de problemas para ser capaz de resolver una gran variedad de familias de problemas.

## PROGRAMACIÓN INFORMÁTICA Y PENSAMIENTO COMPUTACIONAL

Aunque el pensamiento computacional puede trabajarse desde muchas disciplinas e incluso sin necesidad de contar con dispositivos electrónicos, diversas investigaciones demuestran que la programación informática es una muy buena herramienta para desarrollar esta competencia. (FIXME referencia)

Por consiguiente, responsables educativos de todo el mundo han comenzado a incluir la programación en los currículum nacionales y regionales. Los ejemplos con mayor repercusión han sido, en el panorama internacional, los de Inglaterra, con una nueva asignatura “Computing” obligatoria desde primero de primaria, y Estonia, donde la programación se usa de manera transversal para trabajar diversas asignaturas de primaria y secundaria; y en el caso de España, los de Navarra, que ha incluido la programación en 4º y 5º de primaria asociada al área de matemáticas, y la Comunidad de Madrid, que ha incluido contenidos de programación en la asignatura de tecnología de secundaria y ha creado una asignatura optativa de programación en primaria.

Desde la propia Comisión Europea se está urgiendo a los ministros de la Unión a incluir la programación informática en los planes de estudio para que todos los niños y niñas tengan la oportunidad de desarrollar su pensamiento computacional desde la escuela, ya que están convencidos de la importancia de esta competencia para la competitividad y la innovación de nuestro continente. (FIXME referencia) En consecuencia, en los próximos años veremos cómo la programación se incluye de forma paulatina en los currículum de todos los países europeos, muy probablemente desde la educación primaria.

## CÓMO TRABAJAR Y EVALUAR EL PENSAMIENTO COMPUTACIONAL

Sin duda alguna, la inclusión de actividades de programación con Scratch, un lenguaje de programación visual desarrollado específicamente para niños a partir de 6 años, se está implantando en todo el mundo como el estándar para introducir la programación y trabajar el pensamiento computacional en la educación. De hecho, en el momento de escribir este texto, hay más de 8 millones de usuarios registrados en la web de Scratch y más de 11 millones de proyectos compartidos.

No obstante, aunque es posible encontrar rúbricas preparadas por distintas entidades educativas para evaluar el pensamiento computacional del alumnado a partir de los proyectos Scratch desarrollados por los aprendices, no existen apenas herramientas que permitan automatizar parte de este proceso para ayudar a los docentes. En consecuencia, muchos docentes tienen problemas para poder estudiar en profundidad los proyectos de sus alumnos y poder sacar conclusiones para, por ejemplo, aconsejar a sus alumnos acerca de otros bloques que podrían incorporar a sus programas, formar grupos de estudiantes para trabajar un concepto específico que no parecen haber comprendido completamente, o plantearles proyectos avanzados para desarrollar un aspecto concreto una vez alcanzado un determinado nivel. Y este es el motivo que nos llevó a crear la herramienta Dr. Scratch, con el objetivo fundamental de asistir a los docentes en el proceso de enseñanza y evaluación de esta competencia.

## DR. SCRATCH Y EL PENSAMIENTO COMPUTACIONAL

La herramienta Dr. Scratch permite, entre otras funcionalidades, evaluar el grado de desarrollo del pensamiento computacional a partir de un programa Scratch. Las dimensiones evaluadas son las siguientes:

- Abstracción y descomposición de problemas
- Nociones de algoritmia y control del flujo de los programas
- Pensamiento lógico
- Paralelismo
- Sincronización
- Representación de la información
- Interactividad con el usuario

## 2 USANDO DR. SCRATCH

---

### 3 DESARROLLANDO LOS DISTINTOS ASPECTOS DEL PENSAMIENTO COMPUTACIONAL CON DR. SCRATCH

Para cada una de las dimensiones del pensamiento computacional que Dr . Scratch evalúa, la herramienta asigna una puntuación entre 0 y 3 puntos, en función del grado de desarrollo demostrado en la programación del proyecto analizado. Para aquellas dimensiones en las que existe margen de mejora, Dr . Scratch ofrece información para conocer nuevas posibilidades y seguir mejorando cada uno de estos aspectos.

#### 3.1 ABSTRACCIÓN Y DESCOMPOSICIÓN DE PROBLEMAS

La capacidad de abstracción y descomposición de problemas te ayuda a dividir un problema en partes más pequeñas que serán más fáciles de comprender, programar y depurar.

##### 3.1.1 SI HAS SACADO 0 PUNTOS...

Cuando se comienza a programar con Scratch, en ocasiones puede parecer que la solución más sencilla es programar todo el comportamiento de un personaje en un único programa. Sin embargo, lo ideal es que el comportamiento del personaje sea controlado por diferentes programas y que cada uno de estos programas se ocupe de una cuestión concreta. Veamos un ejemplo:



FIGURA 1: FRAGMENTO DE CÓDIGO CON NIVEL 0 EN ABSTRACCIÓN.

Este proyecto, que pinta un dibujo en la pantalla, ha sido programado en un único programa que dibuja las dos líneas que componen el dibujo. Aunque es una opción válida, una opción más sencilla de programar y de mantener es dividir el programa en dos partes, de forma que tengamos dos programas diferentes, uno para pintar la primera línea y otro para pintar la segunda:



FIGURA 2: FRAGMENTO DE CÓDIGO CON NIVEL 1 EN ABSTRACCIÓN.

De este modo, si queremos por ejemplo realizar alguna modificación en una de las líneas dibujadas, es mucho más sencillo saber a qué parte del programa tenemos que dirigirnos para llevar a cabo los cambios.

### 3.1.2 SI HAS SACADO 1 PUNTO...

Scratch permite crear nuevos bloques definidos por los usuarios que se componen de una secuencia de instrucciones. Estas abstracciones permiten crear programas más sencillos de leer, de programar y de mantener. Veamos un ejemplo:



FIGURA 3: FRAGMENTO DE CÓDIGO CON BLOQUES REPETIDOS.

Este proyecto Scratch dibuja dos líneas naranjas de diferente longitud en la pantalla. En lugar de repetir el código 2 veces, tal como vemos en el ejemplo, es posible definir un bloque ?PintaNaranja? que se compone de los bloques que pintan una línea naranja en la pantalla y al

que se le puede indicar cuál es la longitud de la línea. Para ello hay que irse a la categoría ?Más Bloques? y pulsar en el botón Crear un bloque:



FIGURA 4: FUNCIÓN PROPIA PARA EVITAR REPETIR CÓDIGO.

Una vez definido el bloque ?PintaNaranja? es posible utilizarlo en cualquier programa del proyecto, tal como vemos a continuación:

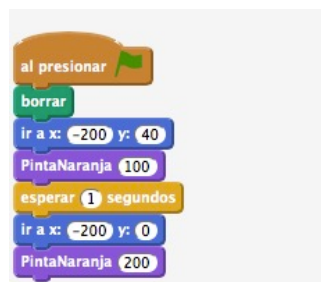


FIGURA 5: USANDO NUESTRA PROPIA FUNCIÓN.

De este modo, evitamos la repetición de código, lo que hace que nuestros proyectos sean más fáciles de programar y mantener. Tal como se puede observar, la primera vez que se usa el bloque PintaNaranja se le indica una longitud de 100 pasos, mientras que la segunda vez la longitud es de 200 pasos.

### 3.1.3 SI HAS SACADO 2 PUNTOS...

En algunos proyectos Scratch queremos tener muchos personajes iguales que realizan exactamente las mismas acciones. La primera idea que se nos ocurre para conseguirlo es crear un personaje, programar todo su comportamiento y, una vez que está listo, crear tantas copias como necesitemos. Por tanto, si queremos 20 marcianitos, hay que crear 20 objetos iguales. Sin embargo, ¿qué ocurriría si quiero realizar un cambio en un programa? Tendría que ir objeto por objeto realizando esa modificación.

Para este tipo de situaciones es preferible utilizar clones, un tipo de abstracción que nos ayuda a poder programar un solo objeto, y crear de forma dinámica copias exactas con el mismo comportamiento. Veamos cómo funcionan con un ejemplo. Imagina que queremos simular que está nevando en un proyecto. Podemos dibujar un objeto que sea un copo de nieve, y una vez que comience la ejecución del proyecto, ir creando clones constantemente que aparezcan en la parte superior de la pantalla y vayan cayendo hasta la parte inferior:





FIGURA 6: USANDO CLONES.

De este modo, tan solo programando un personaje podemos tener infinitos clones, que se crean en un momento determinado de la ejecución del proyecto y se borran cuando ya no son necesarios.

En resumen... Aprender a abstraer ayuda a ver grandes problemas difíciles de resolver en pequeños problemas de fácil solución, favoreciendo, además, a que se desarrolle un proyecto de una forma más eficiente.

### EJERCICIO

Vamos a proponer un ejercicio para practicar la abstracción. Construye un programa en Scratch en el que el personaje tenga que pintar las letras A, B y C.

## 3.2 NOCIONES DE ALGORITMIA Y CONTROL DEL FLUJO DE LOS PROGRAMAS

Las instrucciones relacionadas con las nociones algorítmicas de control de flujo pueden ayudarte a controlar el comportamiento de tus personajes, haciendo, por ejemplo, que repitan ciertos bloques un número de veces concreto o que lo repitan hasta que se produzca una situación.

### 3.2.1 SI HAS SACADO 0 PUNTOS...

La forma más básica de controlar el comportamiento de tus personajes es creando un programa compuesto por un conjunto de bloques que se ejecutan uno detrás de otro, tal como vemos en la imagen:

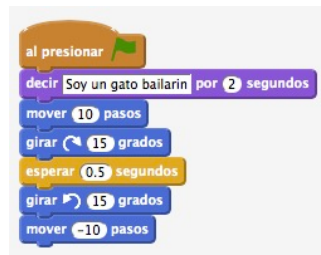


FIGURA 7: FRAGMENTO DE CÓDIGO CON UN SOLO FLUJO.

¿Cómo funciona este programa? Cuando el usuario presione sobre la bandera verde se ejecutarán uno detrás de otro todos los bloques que se han incluido en el programa. Comenzará por el primero 'decir soy un gato que baila por 2 segundos', luego ejecutará 'mover 10 pasos', a continuación 'girar 15 grados a la derecha', y así hasta llegar al último bloque del programa.

### 3.2.2 SI HAS SACADO 1 PUNTO...

En ocasiones, cuando queremos que un conjunto de bloques se repita constantemente, en lugar de repetir los mismos bloques una y otra vez, es posible utilizar instrucciones de repetición que permiten conseguir el mismo efecto, pero de forma más cómoda y manejable. Veamos un par de ejemplos:



FIGURA 8: FRAGMENTOS DE CÓDIGO CON DOS FLUJOS DE EJECUCIÓN.

¿Cómo funciona estos bloques de control? En el bloque 'repetir', cuando la ejecución llega a este punto se repetirán tantas veces como se haya indicado en el parámetro del bloque 'repetir', en el ejemplo serían 3 veces, todos los bloques incluidos dentro de este bloque. Podemos ver que el bloque 'repetir' tiene una flechita en la parte inferior que indica que cuando se terminan de ejecutar los bloques se vuelve otra vez para arriba. El bloque 'por siempre' funciona igual que el 'repetir' pero sin terminar nunca de ejecutar los bloques que contiene en su interior.

### 3.2.3 SI HAS SACADO 2 PUNTOS...

En algunas ocasiones no sabemos previamente el número de veces que queremos que un conjunto de bloques se ejecute, ya que esto depende de una determinada situación. En estas ocasiones, el bloque 'repetir hasta que...' es realmente útil. Veamos un ejemplo:



FIGURA 9: USANDO EL BLOQUE 'SI'.

En el ejemplo, el personaje repetirá constantemente la instrucción decir 'No me pillas...' mientras no esté tocando al enemigo. En el momento que se cumpla la condición de la instrucción 'repetir hasta que...', terminará de ejecutarse el bloque contenido en su interior y saltará a la siguiente instrucción, en este caso decir 'Me pillaste!'.

En resumen... Aprender sobre algoritmia y control de flujo nos ayuda a esquematizar y a poner orden en las acciones que se realizan en nuestro programa.

#### EJERCICIO

Vamos a proponer un ejercicio para practicar el control de flujo. Construye un programa en Scratch en el que se guíe paso a paso a un cocinero a la hora de realizar la receta de tu plato favorito.

## 3.3 PENSAMIENTO LÓGICO

Las instrucciones relacionadas con el pensamiento lógico pueden ayudarte a que tus proyectos sean dinámicos, de forma que se comporten de modo distinto en función de la situación. En las historias, por ejemplo, este tipo de instrucciones no son tan importantes, ya que suelen tener una estructura lineal que siempre queremos que se ejecute del mismo modo, pero en otro tipo de proyectos, como los videojuegos, son fundamentales para ejecutar acciones diferentes dependiendo de la situación.

### 3.3.1 SI HAS SACADO 0 PUNTOS...

El bloque más básico con el que puedes comenzar a trabajar el pensamiento lógico es éste:



FIGURA 10: USANDO EL BLOQUE 'SI'.

¿Cómo funciona este bloque? Cuando la ejecución llega a este punto, se evalúa la condición que aparece en el bloque, y si ésta es verdadera, se ejecuta el conjunto de bloques que están dentro del si. En el ejemplo, si el personaje está tocando el color azul diría ¿Llegué a la meta?.

### 3.3.2 SI HAS SACADO 1 PUNTO...

Ahora que conoces las instrucciones si, quizás puedas comenzar a utilizar bloques si/sino, que pueden ser muy prácticos en muchos tipos de proyectos. Los bloques si/sino son muy parecidos a los bloque si, pero contienen dos conjuntos de bloques en su interior.

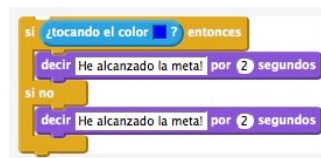


FIGURA 11: USANDO EL BLOQUE 'SI, SI NO'.

¿Cómo funciona este bloque? Cuando la ejecución llega a este punto, se evalúa la condición que aparece en el bloque, y si ésta es verdadera, se ejecuta el conjunto de bloques que están dentro del si. En el ejemplo, si el personaje está tocando el color azul diría ¿Llegué a la meta?, pero si no está tocando el color morado diría ¿Aún no he llegado a la meta?.

### 3.3.3 SI HAS SACADO 2 PUNTOS...

Para tomar decisiones en algunos proyectos en ocasiones se requiere evaluar más de una condición al mismo tiempo para saber qué hay que ejecutar. En esas situaciones es muy útil utilizar operaciones lógicas que permiten combinar las condiciones. Las operaciones lógicas disponibles en Scratch son Y, O y No.



FIGURA 12: EVALUANDO CONDICIONES.

La operación Y es verdadera cuando las dos condiciones de evaluadas son verdaderas. La operación O es verdadera cuando una de las dos condiciones evaluadas son verdaderas. Y la

operación No valdrá lo contrario de la condición, es decir, si la condición es verdadera, No devuelve falso y viceversa.

En resumen... El pensamiento lógico nos ayuda a ejecutar cierto código si se cumplen unas determinadas condiciones, evitando ejecutarlo cuando no es necesario.

## EJERCICIO

(PENSAR EJERCICIO)

### 3.4 PARALELISMO

En la mayoría de las creaciones Scratch se requiere un cierto nivel de paralelismo. Pero, ¿qué es el paralelismo? El paralelismo es la posibilidad de que varias cosas ocurran al mismo tiempo. Por ejemplo, que dos personajes realicen una acción al mismo tiempo, o que un personaje haga varias cosas a la vez.

#### 3.4.1 SI HAS SACADO 0 PUNTOS...

La forma más básica y más evidente de conseguir paralelismo es tener varios programas que comienzan con ¿Al presionar la bandera verde?:



FIGURA 13: BLOQUE .AL PULSAR BANDERA VERDE”.

De este modo, cuando el usuario pinchase sobre la bandera verde, todos los programas que comienzan con este bloque comenzarían a ejecutarse al mismo tiempo, o en paralelo, como también puede decirse. Podrías tener varios programas que comiencen con este bloque en un mismo personaje, si quieres que haga varias cosas a la vez, o en programas de diferentes personajes, si quieres que todos comiencen a realizar una acción al comenzar la ejecución.

#### 3.4.2 SI HAS SACADO 1 PUNTO...

Otra forma de conseguir paralelismo en tus programas es haciendo que ocurran varias cosas cuando el usuario presiona una tecla o hace click sobre un objeto. Veamos un par de ejemplos:



FIGURA 14: CÓDIGO QUE SE EJECUTA EN PARALELO.

¿Cómo funcionan estos bloques de control? En las dos primeras figuras de la izquierda vemos fragmentos de código iguales que pertenecen a dos personajes distintos, de manera que, cuando se pulsa una tecla, realizan una determinada acción. Por tanto, cuando el usuario pulse la tecla a, en este caso, tanto un personaje como el otro ejecutarán al mismo tiempo ¿decir A por 2 segundos?. En el último ejemplo vemos que un personaje tiene dos programas que comienzan con ¿al clicar este objeto?. Por tanto, cuando el usuario haga click sobre este personaje, ambos programas comenzarán a ejecutarse al mismo tiempo, en paralelo.

### 3.4.3 SI HAS SACADO 2 PUNTOS...

Existen varios eventos más que permiten conseguir paralelismo:



FIGURA 15: DISPARADORES DE EVENTOS.

Como ves, podrías crear varios programas que comenzaran a ejecutarse al cambiar el fondo a un determinado escenario, o al recibir un mensaje concreto, o cuando el volumen del ambiente sea superior a un determinado umbral, cuando el movimiento del vídeo sea mayor que un número de pixels concreto o cuando el cronómetro haya superado el valor que tú quieras. Por tanto, existen muchas posibilidades para conseguir que en tus programas ocurran cosas al mismo tiempo. ¿Te animas a probar algunas de ellas?

En resumen... El paralelismo nos permite ejecutar varias acciones a la vez en un mismo personaje y/o en varios capturando determinados eventos(teclado, ratón, sonido, vídeo, mensajes...).

## EJERCICIO

(PENSAR EJERCICIO)

### **3.5 SINCRONIZACIÓN**

### **3.6 REPRESENTACIÓN DE LA INFORMACIÓN**

### **3.7 INTERACTIVIDAD CON EL USUARIO**

## **4 PROMOVRIENDO BUENOS HÁBITOS DE PROGRAMACIÓN CON DR. SCRATCH**

---

### **4.1 NOMBRADO DE OBJETOS**

### **4.2 CÓDIGO MUERTO**

### **4.3 INICIALIZACIÓN DE LOS ATRIBUTOS DE LOS PERSONAJES**

### **4.4 REPETICIÓN DE CÓDIGO**

## **5 CONCLUSIONES FINALES**

---