Pensamiento computacional

Al final de este módulo vamos a:

Tener una idea clara de qué es, para qué sirve y cómo desarrollar el pensamiento computacional.

Conocer los cuatro pilares del pensamiento computacional y afrontar retos complejos de forma metódica.

Evaluar la solución obtenida, logrando que esta sea una solución eficiente.

Practicar la resolución de problemas cotidianos desde el punto de vista del Pensamiento Computacional.

Introducción al pensamiento computacional

¿Qué es el pensamiento computacional?

Información basada en "Introduction to computational thinking" de BBC.

Los ordenadores pueden usarse para ayudarnos a **resolver problemas**, sin embargo, antes de que se pueda abordar un problema, es necesario **comprender** el problema en sí mismo y las formas en que podría resolverse.

El pensamiento computacional nos facilita **afrontar** un problema complejo, **comprender** cuál es el problema y **desarrollar** posibles estrategias para resolverlo. Luego podemos presentar estas soluciones de una manera tal que un ordenador, una persona o ambos puedan entenderlo.

Los cuatro pilares del pensamiento computacional

La **clasificación de la BBC** se basa en 4 pilares (Descomposición, reconocimiento de patrones, algoritmos y abstracción) haciendo una mención especial a la **evaluación** . En otros estudios podrás encontrar otras clasificaciones más detalladas. En el módulo 2 de este curso: actividades desenchufadas, se trabaja con estos cuatro pilares mas otros dos: Modelado y simulación, y Evaluación. Te explicamos aquí los fundamentales:

- **Descomposición** - Dividir un problema o sistema complejo en partes más pequeñas y manejables.
- **Reconocimiento de patrones** – Buscar similitudes entre problemas y dentro del problema.
- **Abstracción** – Centrarse sólo en la información importante, ignorando los detalles irrelevantes.

- **Algoritmos** - desarrollar una solución pasos a paso, o establecer reglas a seguir para resolver el problema.

Los cuatro pilares son **importantes**. Son como las patas de una mesa: si falta una pata, la mesa se caerá. La aplicación correcta de las cuatro técnicas ayudará a la hora de programar un ordenador.

El pensamiento computacional en la práctica

Un **problema complejo** es aquel que, a primera vista, no sabemos cómo resolver fácilmente.

El pensamiento computacional implica coger ese problema complejo y dividirlo en una serie de problemas pequeños y más manejables (**descomposición**). Luego, cada uno de estos problemas más pequeños se puede analizar individualmente, considerando cómo se han resuelto problemas similares anteriormente (**reconocimiento de patrones**) y centrándose solo en los detalles importantes, mientras se ignora la información irrelevante (**abstracción**). A continuación se pueden diseñar pasos o reglas simples para resolver cada uno de los problemas más pequeños (**algoritmo**).

Finalmente, estos pasos simples o reglas son usados para **programar un ordenador** o para ayudar a **resolver un problema complejo** de manera más eficaz.

Pensar de forma computacional

Pensar de forma computacional no es programar. Ni siquiera es pensar como un ordenador, ya que los ordenadores no pueden pensar.

En pocas palabras, la programación dice a un ordenador qué hacer y cómo hacerlo. El pensamiento computacional te permite **averiguar exactamente qué decir al ordenador que haga**. Es por tanto una **habilidad humana**, independiente del uso de ordenadores.

Por ejemplo, si quedas en reunirte con tus amigos en algún lugar en el que nunca has estado antes, probablemente planificarás tu ruta antes de salir de casa. Puedes considerar las rutas disponibles y qué ruta es la “mejor”. Esta podría ser la ruta más corta, la más rápida o la que pasa por tu tienda favorita en el camino. Luego seguirías las instrucciones paso a paso hasta llegar allí. En este caso, la parte de planificación es como el pensamiento computacional, y seguir las instrucciones es como programar.

Ser capaz de convertir un problema complejo en uno que podamos entender fácilmente es una **habilidad** que es **extremadamente útil**. De hecho, es una habilidad que ya tienes y que probablemente uses todos los días.

Por ejemplo, puede ser que necesites decidir qué hacer con tu grupo de amigos/as. Si a todos/as os gustan cosas diferentes, tendréis que **decidir**:

- ¿Qué podríamos hacer?
- ¿Dónde podríamos ir?
- ¿Quién quiere hacer qué?
- ¿Qué hemos hecho en el pasado que fue un éxito?
- ¿Con cuánto dinero contamos y cuánto cuesta cada opción posible?
- ¿Qué previsión de tiempo dan?
- ¿De cuánto tiempo disponemos?

Con toda esta **información**, tú y tus amigos/as podréis decidir más fácilmente a dónde ir y qué hacer y que la mayoría esté **conforme**. También podrías usar un ordenador para ayudarte a recopilar y analizar los datos para idear **la mejor solución** al problema, tanto ahora como si volviera a surgir en el futuro.

Otro ejemplo puede ser al jugar a un **videojuego**. Dependiendo del juego, para completar un nivel necesitarás saber:

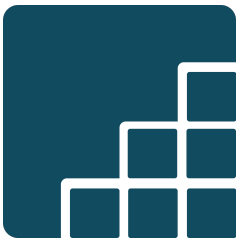
- ¿Qué artículos necesitas recoger?
- ¿Cómo puedes recogerlos?
- ¿Cuánto tiempo tienes para recogerlos?
- ¿Dónde está la salida?
- ¿La mejor ruta para llegar a ella en el menor tiempo posible?
- ¿Qué tipo de enemigos hay?
- ¿Cuáles son sus puntos débiles?

A partir de estos detalles puedes elaborar una **estrategia** para completar el nivel de la manera más **eficiente**.

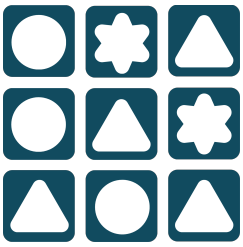
Si tuvieras que crear tu propio juego, estas **preguntas** son exactamente las que tendrías que plantearte y responder antes de programar el juego.

Los dos ejemplos anteriores son ejemplos de cómo se ha utilizado el pensamiento computacional para resolver un problema complejo:

- Cada problema complejo se ha dividido en partes más simples - **descomposición.**

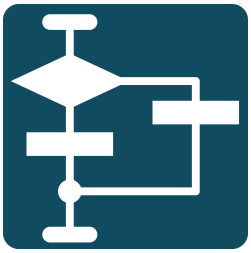


- Se utilizó el conocimiento de problemas similares - **reconocimiento de patrones**.



- Solo se centraron en los detalles relevantes - **abstracción.**

- Para elaborar un plan de acción paso a paso -**algoritmo.**



> Si quieres **saber más,** y practicar tu inglés, mira este vídeo:

https://www.youtube.com/embed/Nc-V948dXWI

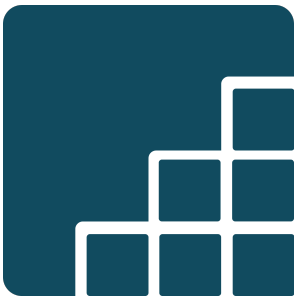
> Computational Thinking and problem solving, by Miles Berry Europe Code Week

Pilares del pensamiento computacional: Descomposición

Descomposición: Divide y vencerás

Imagen de pikisuperstar en Freepik

¿Qué es la descomposición?



Antes de que se pueda **resolver** un problema, se debe **comprender** el problema y las **formas** en que se puede resolver. La descomposición implica **dividir** un problema o sistema complejo en partes más pequeñas que son más **manejables** y fáciles de entender. Luego, las partes más pequeñas se pueden **examinar y resolver**, o diseñar individualmente, ya que es más fácil trabajar con ellas.

¿Por qué es importante la descomposición?

Si un problema **no** se descompone:

- **Es mucho más difícil de resolver**. Tratar con muchas variables diferentes a la vez es mucho más difícil que dividir un problema en varios problemas más pequeños y resolver cada uno, de uno en uno. Dividir el problema en partes más pequeñas significa que cada problema más pequeño se puede examinar con más detalle.
- **Es mucho más difícil de entender**. Tratar de entender cómo funciona un sistema complejo es más fácil fijándose en cada uno de los sub-sistemas. Por ejemplo, entender cómo funciona una bicicleta es más sencillo si toda la bicicleta se separa en partes más pequeñas y se examina cada parte para ver cómo funciona con más detalle.

La descomposición en la práctica

Hacemos muchas **tareas a diario** sin siquiera pensar en ellas, o descomponerlas, como lavarnos los dientes.

Ejemplo 1: Lavarnos los dientes

Para descomponer el problema de cómo lavarnos los dientes, necesitaríamos considerar:

- ¿Qué cepillo usar?
- ¿Cuánto tiempo estar cepillándome los dientes?
- ¿Cuánta presión debo hacer con el cepillo?
- ¿Qué pasta usar?

Ejemplo 2: Resolver un crimen

Normalmente, solo cuando se nos pide que hagamos una tarea nueva o más compleja, comenzamos a pensar en ella en detalle, a descomponer la tarea.

Imagina que se ha cometido un crimen. Resolver un crimen puede ser un problema muy **complejo** ya que hay muchas cosas a considerar.

Por ejemplo, la policía deberá averiguar la respuesta a una serie de **problemas menores**:

- ¿Qué crimen ha sido cometido?
- ¿Cuándo fue cometido el crimen?
- ¿Dónde fue cometido el crimen?
- ¿Qué evidencia hay?
- ¿Hay algún testigo?
- ¿Han ocurrido crímenes similares?

El problema complejo de resolver el delito cometido ahora se ha descompuesto en problemas más simples que pueden ser examinados individualmente, en **detalle**.

Ejemplo 3: Crear una aplicación

Imagínate que vas a crear tu primera aplicación. Este es un problema complejo en el que hay un montón de cosas a considerar.

¿Cómo descompondrías la tarea de crear una app?

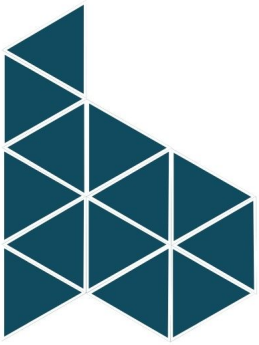
Para descomponer esta tarea, necesitarás encontrar la respuesta a una serie de problemas menores:

- ¿Qué necesita para cubrir tu aplicación?
- ¿Qué aspecto tendrá tu app?
- ¿Quién es el objetivo público de tu app?
- ¿Qué tipo de gráficos usarás?
- ¿Qué tipo de audio usarás?
- ¿Qué tipo de software usarás para crear la app?
- ¿Cómo navegará el usuario por la aplicación?
- ¿Cómo probarás la aplicación?
- ¿Dónde se venderá la aplicación?

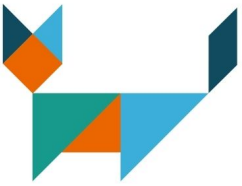
Esta lista ha **desglosado** el problema complejo de crear una aplicación en problemas mucho más simples, que ahora se pueden resolver. También puedes hacer que **otras personas** te ayuden con diferentes partes individuales de la aplicación. Por ejemplo, puedes conocer a alguien que pueda crear los gráficos...

IDEA Descomponer tareas cotidianas: prepara un bocadillo, prepara una excursión... (cuanto más cercana sea la tarea a la experiencia del alumnado mejor).

IDEA Descomponer una figura geométrica en figuras sencillas ya conocidas.



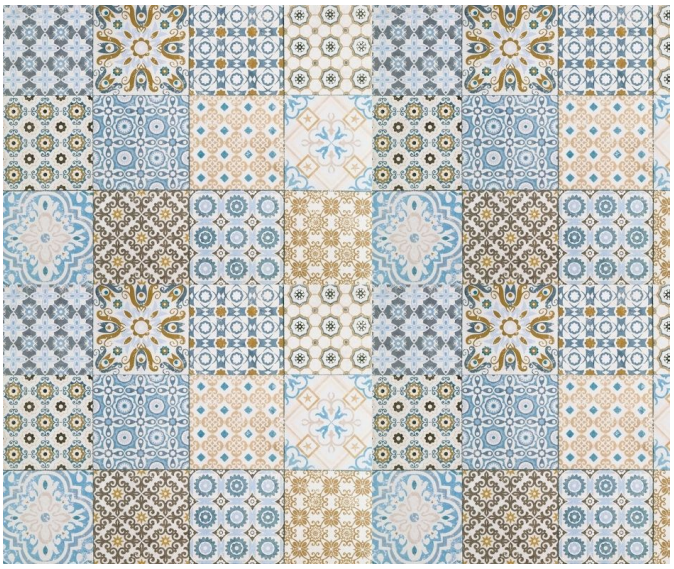
IDEA Jugar al tangram



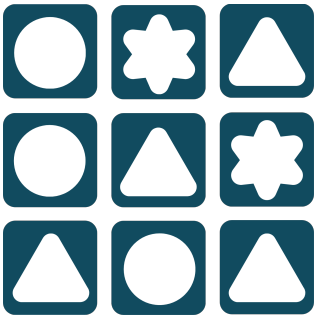
IDEA hacer mapas mentales

Pilares del pensamiento computacional: Reconocimiento de patrones

Reconocimiento de patrones: Esto lo he visto antes…



¿Qué es reconocimiento de patrones?



El **reconocimiento de patrones** es uno de los cuatro pilares del pensamiento computacional. Implica **encontrar patrones entre problemas pequeños y descompuestos**. A menudo, cuando descomponemos un problema complejo encontramos patrones entre los problemas más pequeños que creamos. **Identificar** estas similitudes nos ayuda a resolver problemas más complejos de manera más **eficiente**. Los patrones son pues, similitudes, estructuras repetitivas, características comunes...

¿Qué son patrones?

Imagina que queremos dibujar una **serie de gatos**.

Todos los gatos comparten **características comunes**. Entre otras cosas, todos tienen ojos, cola y pelaje. También les gusta comer pescado y maullar.

Como sabemos que todos los gatos tienen ojos, cola y pelaje, podemos hacer un buen intento de **dibujar un gato** simplemente incluyendo estas características comunes.

En el pensamiento computacional, estas características se conocen como **patrones**. Una vez que sabemos cómo describir un gato, podemos describir otros, simplemente siguiendo este patrón. Lo único que es diferente son los detalles:

- Un gato puede tener ojos verdes, cola larga y pelaje negro.
- Otro gato puede tener ojos amarillos, cola corta y pelaje rayado.

¿Porqué necesitamos buscar patrones?

Encontrar patrones es **extremadamente importante**. Los patrones **simplifican** nuestra tarea. Los problemas son más fáciles de resolver cuando comparten patrones, porque podemos usar **una misma resolución** donde exista un determinado patrón.

Cuantos más patrones podamos encontrar, más fácil y rápida será nuestra tarea general de resolución de problemas.

Si queremos dibujar varios gatos, encontrar un patrón para describir a los gatos en general, como por ejemplo que todos tienen ojos, cola y pelaje, hace que esta tarea sea más **rápida** y **sencilla**.

Sabemos que todos los gatos siguen este patrón, así que no tenemos que detenernos cada vez que empezamos a dibujar un nuevo gato para resolver esto. A partir de los patrones que sabemos que siguen los gatos, podemos dibujar rápidamente varios gatos.

¿Qué ocurre cuando no buscamos un patrón?

Supongamos que no hubiéramos buscado patrones en los gatos. Cada vez que quisiéramos dibujar un gato, tendríamos que **detenernos** y **averiguar** cómo era un gato. Esto nos **retrasaría**.

Aún así, podríamos dibujar a nuestros gatos, y se reconocerían como gatos, pero **tardaríamos más** en dibujar cada gato. Esto sería muy **ineficiente** y una mala manera de resolver la tarea de dibujar gatos.

Además, si no buscamos patrones, es posible que **no** nos demos cuenta de que todos los gatos tienen ojos, cola y pelaje. Cuando están dibujados, es posible que nuestros gatos **ni siquiera parezcan gatos**. En este caso, al no reconocer el patrón, estaríamos resolviendo el problema incorrectamente.

Reconocimiento de patrones dentro del mismo problema o entre problemas

diferentes

Para encontrar patrones en problemas buscamos elementos que son iguales (o muy similares) en cada problema. Puede resultar que no existan características comunes entre los problemas, pero aun así deberíamos buscar.

Existen patrones entre **diferentes** problemas y **dentro** de problemas individuales. Tenemos que buscar los dos.

Patrones entre diferentes problemas

Para encontrar patrones entre problemas, buscamos cosas que sean iguales (o muy similares) para cada problema.

Por ejemplo, descomponer la tarea de **hornear un pastel** supone que conozcamos las soluciones a una serie de problemas más pequeños:

- Qué tipo de pastel queremos hornear
- Qué ingredientes necesitamos y cuánto de cada uno
- Para cuántas personas queremos hacer el pastel
- Cuánto tiempo tenemos que hornear el pastel
- Cuándo necesitamos agregar cada ingrediente
- Qué equipo necesitamos

Una vez que sabemos cómo hornear un tipo particular de pastel, podemos ver que **hornear otro tipo de pastel** no es tan diferente, porque existen **patrones**.

Por ejemplo:

- Cada pastel necesitará una cantidad precisa de ingredientes específicos.
- Los ingredientes se agregarán en un momento concreto.
- Cada pastel se horneará durante un período de tiempo concreto.

Tenemos los patrones identificados, podemos trabajar en soluciones comunes entre los problemas.

Patrones dentro de los problemas

Los patrones también pueden existir **dentro** de los problemas más pequeños, en los que hemos descompuesto el problema.

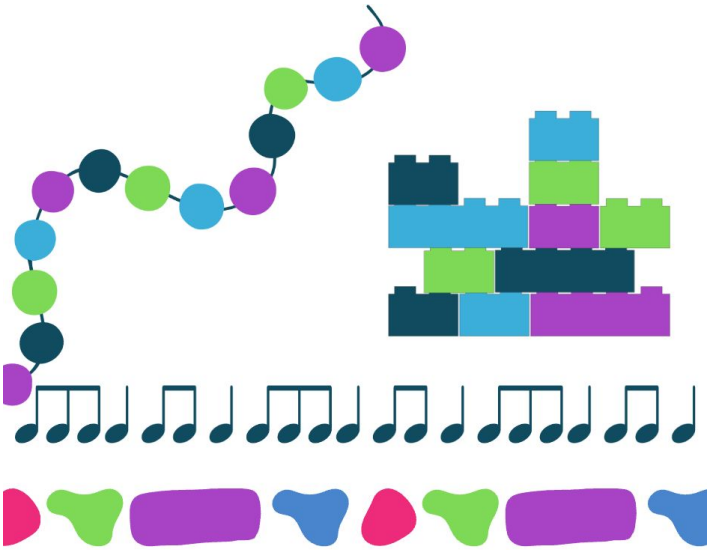
Si nos fijamos en hornear un pastel, por ejemplo, para cada pastel se necesitará una **cantidad precisa** de ingredientes específicos, cada ingrediente necesita:

- Identificar el ingredientes (nombrarlo)
- Una medida específica

Una vez que sabemos cómo identificar cada ingrediente y su cantidad, podemos aplicar ese patrón a todos los ingredientes. Una vez más, todo lo que cambia son los **detalles**.

El reconocimiento de patrones dentro de un problema es sencillo de ver si nos **imaginamos** que estamos haciendo un pastel con una niña o niño. Si por ejemplo hacemos un bizcocho cuyos ingredientes son: harina, azúcar, cacao en polvo, levadura, yogurt y huevos. Podemos hacer juntas la tarea de **agregar la harina**. **Miramos la cantidad requerida, la pesamos y la apartamos en un bol**. Una vez hecho, puedo pedir a mi compi que haga sola o solo el paso correspondiente al azúcar. Para ello empleará el **mismo procedimiento** que hemos empleando para la harina, es decir, usará un **patrón**. Este mismo puede repetirse para otros ingredientes como el cacao o la levadura. **Son patrones dentro del mismo problema.**

> **IDEAS**. Además de usar soportes tipo ficha como los que usaremos en el módulo 2, puedes utilizar otros soportes y realizar estas actividades de manera transversal en varias asignaturas y eventos escolares. **Bloques de construcción:** Usando color, forma o ambos elementos. **Hacer collares, pulseras, guirnaldas** para navidad o la fiesta de fin de curso, elementos para decorar el aula. **Patrones musicales y rítmicos:** En esta actividad puedes implica todo el cuerpo. **Secuencias con plastilina**.



> Los **edificios** escolares a menudo siguen patrones, por ejemplo en las baldosa, en cada cuanto hay una ventana, donde están los baños, tal vez los ciclos estén identificados por colores.... podéis jugar a encontrar los de vuestra escuela.

> **Juegos de mesa tipo el SET**. Recomendado para mayores de 6 años y con partidas de entre 10 y 20 min de duración. En él usamos unas cartas con diferentes atributos de color, cantidad, forma y tipo de relleno para crear los patrones y hacer tríos de cartas según los que definamos. Es un juego complejo ya que incluye patrones de diferencia, es decir, yo

puedo asociar cartas porque tienen todas el mismo color o porque ningún color esta repetido. A su vez, puedo combinar patrones de igualdad y de diferencia. Puedo hacer un trio donde: todas las cartas tengan el mismo color, no se repita ninguna forma , no se repita ningún fondo y haya la misma cantidad de elementos. Las combinaciones son múltiples. Si no te ha quedado claro, lo mejor es que mires el siguiente video:

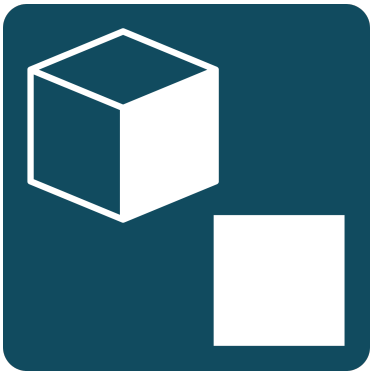
https://www.youtube.com/embed/7UOjO712O2M

Este es un juego competitivo. Esto puede suponer un problema a la hora de llevarlo al aula dependiendo del grupo que tengas y los recursos de los que dispongas. Una variante del juego puede ser que en vez de jugar a ver quién hace más sets sea a ver cuántos sets pueden hacer entre todas las personas que están jugando en un tiempo limitado.

Pilares del pensamiento computacional: Abstracción

Abstracción. Me sirve, no me sirve

¿Qué es la abstracción?



Una vez que hemos reconocido patrones en nuestros problemas, usamos la abstracción para **reunir las características generales y filtrar los detalles** que no necesitamos para resolver nuestro problema para así concentrarnos en las que sí necesitamos.

En el pensamiento computacional, cuando descomponemos un problema, buscamos patrones con problemas similares o dentro del mismo problema que simplifiquen el problema complejo. Cuando abstraemos, **filtramos**, nos quedamos sólo con la **información relevante**. A partir de esto creamos una **representación** (idea) de lo que estamos tratando de resolver.

¿Cuáles son los detalles o características específicas?

En el reconocimiento de patrones, analizamos el problema de tener que dibujar una serie de gatos.

Notamos que todos los gatos tienen **características generales**, que son comunes a todos los gatos, por ejemplo, ojos, cola, pelaje, gusto por los peces y la capacidad de hacer maullidos. Además, cada gato tiene **características específicas**, como pelaje negro, cola larga, ojos verdes, amor por el salmón y un fuerte maullido. Estos detalles se conocen como específicos.

Para dibujar un gato básico, necesitamos saber que tiene cola, pelaje y ojos. Estas características son **relevantes**. No necesitamos saber qué sonido hace un gato o que le gusta el pescado. Estas características son **irrelevantes** y se pueden filtrar.

A partir de las características generales que tenemos (cola, pelaje, ojos) podemos construir una **idea básica de un gato**, es decir, cómo es básicamente un gato. Es entonces cuando podemos describir cómo dibujar un gato básico.

¿Por qué la abstracción es importante?

La abstracción nos permite crear una idea general de cuál es el problema y cómo resolverlo. El proceso nos indica que eliminemos todos los detalles específicos y cualquier patrón que no nos ayude a resolver nuestro problema. Esto **nos ayuda a formar nuestra idea del problema**. Esta idea se conoce como un "modelo".

Si no abstraemos, podemos terminar con una **solución incorrecta** al problema que estamos tratando de resolver. Con nuestro ejemplo del gato, si no hiciéramos abstracción, podríamos pensar que todos los gatos tienen colas largas y pelaje corto. Habiéndonos abstraído, sabemos que aunque los gatos tienen cola y pelaje, no todas las colas son largas y no todo el pelaje es corto. En este caso, la abstracción nos ha ayudado a formar **un modelo más claro** de un gato.

Cómo abstraer

La abstracción es la **recopilación de las características generales** que necesitamos y la **filtración** de los detalles y características que **no** necesitamos.

Al hornear un pastel, hay algunas características **generales** entre los pasteles. Por ejemplo:

- Un pastel necesita ingredientes
- Cada ingrediente necesita una cantidad específica
- Un pastel necesita un tiempo de horneado

Al abstraer, **eliminamos detalles específicos** y mantenemos los patrones generales relevantes.

Creando un modelo

Un modelo es una idea general del problema que estamos tratando de resolver

Por ejemplo, un gato modelo sería cualquier gato. No es un gato específico con cola larga y pelaje corto: el modelo representa a **todos los gatos**. A partir de nuestro modelo de gatos, podemos aprender cómo es cualquier gato, usando los patrones que comparten todos los gatos.

Del mismo modo, al hornear un pastel, un pastel modelo no sería un pastel específico, como un bizcocho o un pastel de frutas. En cambio, el modelo representaría todos los pasteles. A partir de este modelo podemos aprender a hornear cualquier pastel, utilizando los patrones que se aplican a todos los pasteles.

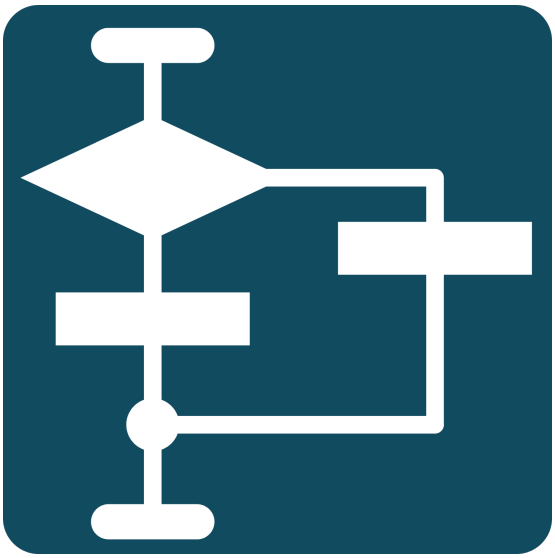
Una vez que tenemos un **modelo** de nuestro problema, podemos diseñar un **algoritmo** para resolverlo.

Pilares del pensamiento computacional: Algoritmos

Algoritmos. Vamos paso a paso

Un algoritmo es un plan, un **conjunto de instrucciones paso a paso** para resolver un problema. En un algoritmo se **identifica** cada instrucción y se **planifica** el orden en que se deben ejecutar.

¿Qué es un algoritmo?



Los algoritmos son uno de los cuatro pilares del pensamiento computacional. Un algoritmo es un **plan**, un conjunto de instrucciones paso a paso para resolver un problema. Si puedes atarte los cordones de los zapatos, preparar una taza de té, vestirte o preparar una comida, entonces ya sabes **cómo seguir un algoritmo**.

En un algoritmo se **identifica** cada instrucción y se **planifica** el orden en el que se deben llevar a cabo. Los algoritmos a menudo se usan como **punto de partida** para crear un programa de ordenador y, a veces, se escriben como un **diagrama de flujo** o en **pseudo-código**.

Si queremos decirle a un ordenador que haga algo, tenemos que escribir un programa informático que le diga al ordenador, paso a paso, exactamente **qué** queremos que haga y **cómo** queremos que lo haga. Este programa paso a paso necesitará planificación, y para hacerlo usamos un algoritmo.

Las ordenadores son tan buenos como los algoritmos que se les da. Si introduces en el ordenador un algoritmo deficiente, obtendrás un resultado deficiente.

Los algoritmos se utilizan para muchas cosas diferentes, incluidos los cálculos, el procesamiento de datos y la automatización.

Haciendo un plan

Es importante planificar la solución a un problema para asegurarse de que será correcta. Usando el pensamiento computacional y la descomposición, podemos dividir el problema en partes más pequeñas y luego podemos planificar cómo volver a encajarlo en un **orden adecuado** para resolver el problema.

Este orden se puede representar como un algoritmo. **Un algoritmo debe ser claro**.

Debe tener:

- un punto de partida
- un punto de llegada
- un conjunto de instrucciones claras entre ambos.

Representación de un algoritmo: Pseudo-código

Hay dos formas principales en que se pueden representar los algoritmos: **pseudo-código** y **diagramas de flujo**.

La mayoría de los programas se desarrollan utilizando **lenguajes de programación**. Estos lenguajes tienen una **sintaxis** específica que debe usarse para que el programa funcione correctamente. El **pseudo-código** no es un lenguaje de programación, es una forma simple de describir un conjunto de instrucciones que **no** tiene que usar una sintaxis específica.

Escribir en pseudo-código es **similar** a escribir en un lenguaje de programación. Cada paso del algoritmo está escrito en una **línea propia** en secuencia. Por lo general, las instrucciones se escriben en mayúsculas, las variables en minúsculas y los mensajes en forma de oración.

En pseudo-código, **ENTRADA** hace una pregunta. **SALIDA** imprime un mensaje en pantalla.

Se podría crear un programa simple para preguntarle a alguien su nombre y edad, y hacer un comentario basado en esto. Este programa representado en pseudo-código tendría este aspecto:

SALIDA “Cómo te llamas?”

ENTRADA el usuario introduce, teclea, su nombre

ALMACENAR la entrada del usuario en la variable **nombre**

SALIDA “Hola” + nombre

SALIDA “¿Qué edad tienes?”

ENTRADA el usuario introduce su edad

ALMACENAR la entrada del usuario en la variable **edad**

SI edad >= 70 ENTONCES

    SALIDA “¡Estás estupendo!”

SI NO

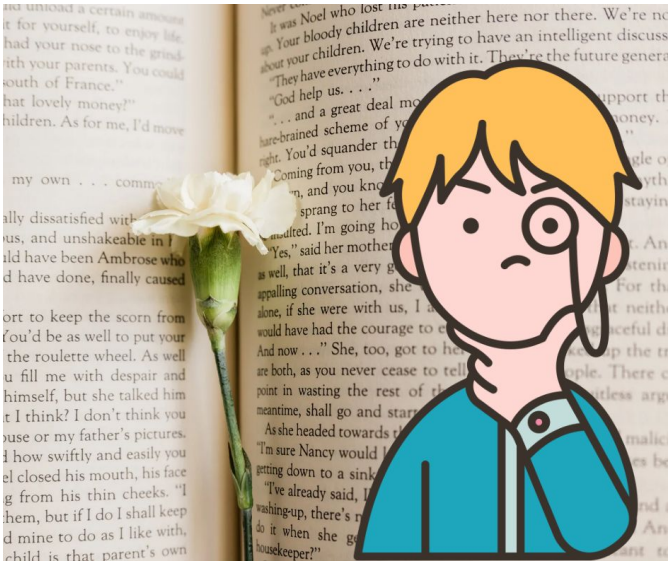
    SALIDA “¡Eres un chaval!”

IDEA.Pueden jugar a **guiar el dibujo** de otra persona. Una persona da las instrucciones, por ejemplo: dibuja un círculo, haz unos triángulos saliendo del círculo, dibuja una casa debajo del círculo…. Puede haber varias personas dibujando lo que una persona indica. Las diferencias entre los dibujos en parte se deberán a la que **las instrucciones no han sido lo suficientemente concretas**. A medida que jueguen irán **afinando** las instrucciones, por ejemplo: dibuja un círculo de 3 cm de diámetro en el centro del la hoja, haz 5 triángulos iguales saliendo del círculo. Los triángulos tienen dos lados iguales, la base es el trazado del círculo y no es igual a los lados, dibuja una casa debajo del círculo, compuesta por un tejado triangular igual a los que has dibujado para hacer el sol…. Para facilitar el trabajo puedes ofrecerles papel cuadriculado para hacer esta actividad y entregarles fichas a los que deben describir para que las otras personas dibujen o que lo hagan de manera libre.

IDEA. Hacer un pseudo-código para hacer **tareas sencillas**, como ir al baño, sacar punta a un lápiz, levantar una silla….. Cada equipo escribe las instrucciones y se las pasa a otro equipo. El otro equipo las tiene que seguir al pie de la letra y hacer **solo** lo que pone en las instrucciones.

IDEA. Tal vez conozcas el texto de **Julio Cortazar** “Instrucciones para subir una escalera”. Puede ser divertido escoger textos de instrucciones y usarlos de base para hacer un pseudo-código. O coger un pseudo-código sencillo y convertirlo en un texto difícil de entender cómo es el caso de este cuento. También puedes adaptar el cuento para que no se sepa de qué se habla y jugar a adivinar. A partir de aquí puedes animarles ha hacer su propio cuento , en el que describan una actividad que las demás deban adivinar. Una posibilidad de **adaptación** podrías ser: “Instrucciones para realizar una tarea misteriosa”: "*Nadie habrá dejado de observar que con frecuencia el suelo se pliega de manera tal que una parte sube en ángulo recto con el plano del suelo, y luego la parte siguiente se coloca paralela a este plano, para dar paso a una nueva perpendicular, conducta que se repite en espiral o en línea quebrada hasta alturas sumamente variables. Agachándose y poniendo la mano izquierda en una de las partes verticales, y la derecha en la horizontal correspondiente, se está en posesión momentánea de un segmento. Cada uno de estos segmentos, formados como se ve por dos elementos, se sitúa un tanto más arriba y adelante que el anterior, principio que da sentido a la estructura, ya que cualquiera otra combinación producirá formas quizá más bellas o pintorescas, pero incapaces de cumplir su función. Estas estructuras se encaran de frente, pues hacia atrás o de costado resultan particularmente incómodas. La actitud natural consiste en mantenerse de pie, los brazos colgando sin esfuerzo, la cabeza erguida aunque no tanto que los ojos dejen de ver los peldaños inmediatamente superiores al que se pisa, y respirando lenta y regularmente. Para realizar nuestra tarea misteriosa se comienza por levantar esa parte del cuerpo situada a la derecha abajo, envuelta casi siempre en cuero o gamuza, y que salvo excepciones cabe exactamente en el segmento. Puesta en el primer*

segmento dicha parte, que para abreviar llamaremos pie, se recoge la parte equivalente de la izquierda (también llamada pie, pero que no ha de confundirse con el pie antes citado), y llevándola a la altura del pie, se le hace seguir hasta colocarla en el segundo segmento, con lo cual en éste descansará el pie, y en el primero descansará el pie. (Los primeros segmentos son siempre los más difíciles, hasta adquirir la coordinación necesaria. La coincidencia de nombre entre el pie y el pie hace difícil la explicación. Cuídese especialmente de no levantar al mismo tiempo el pie y el pie). Llegado en esta forma al segundo segmento, basta repetir alternadamente los movimientos hasta encontrarse con el final de la estructura. Se sale de ella fácilmente, con un ligero golpe de talón que la fija en su sitio, del que no se moverá hasta el momento del descenso"



IDEA. Jugar a ser **robots**: Se puede jugar por tríos: una persona es un robot tipo BeeBot, otra quien programa y la tercera al ver el código escrito tiene que **adivinar a dónde llegará** el “robot”. Quien programa escribe la secuencia usando flechas, se lo muestra a quien adivina y este hace una predicción. Luego se “programa” a quien hace de robot haciendo como que tiene una botonera en la espalda (adelante, atrás, izquierda, derecha). El juego aumenta la dificultad si quien hace de robot tiene los **ojos vendados** o si pones **obstáculos**. Facilita el juego hacerlo en un lugar donde haya baldosas o similares que delimiten cuál es la unidad de desplazamiento.

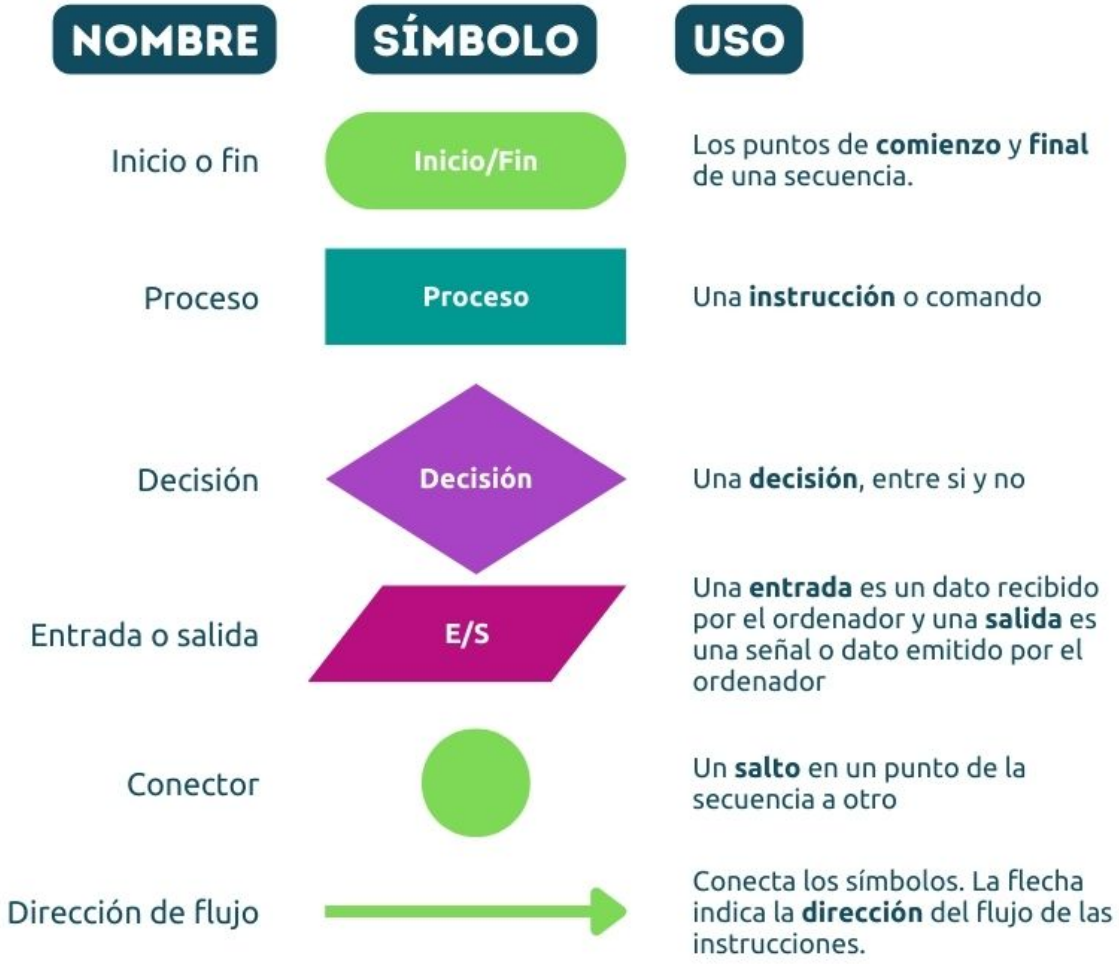
Si no conoces como funciona la Beebot te dejamos un video, podrás ver cómo funciona a partir de el minuto 1: 47

https://www.youtube.com/embed/sqqmH5PwR4s

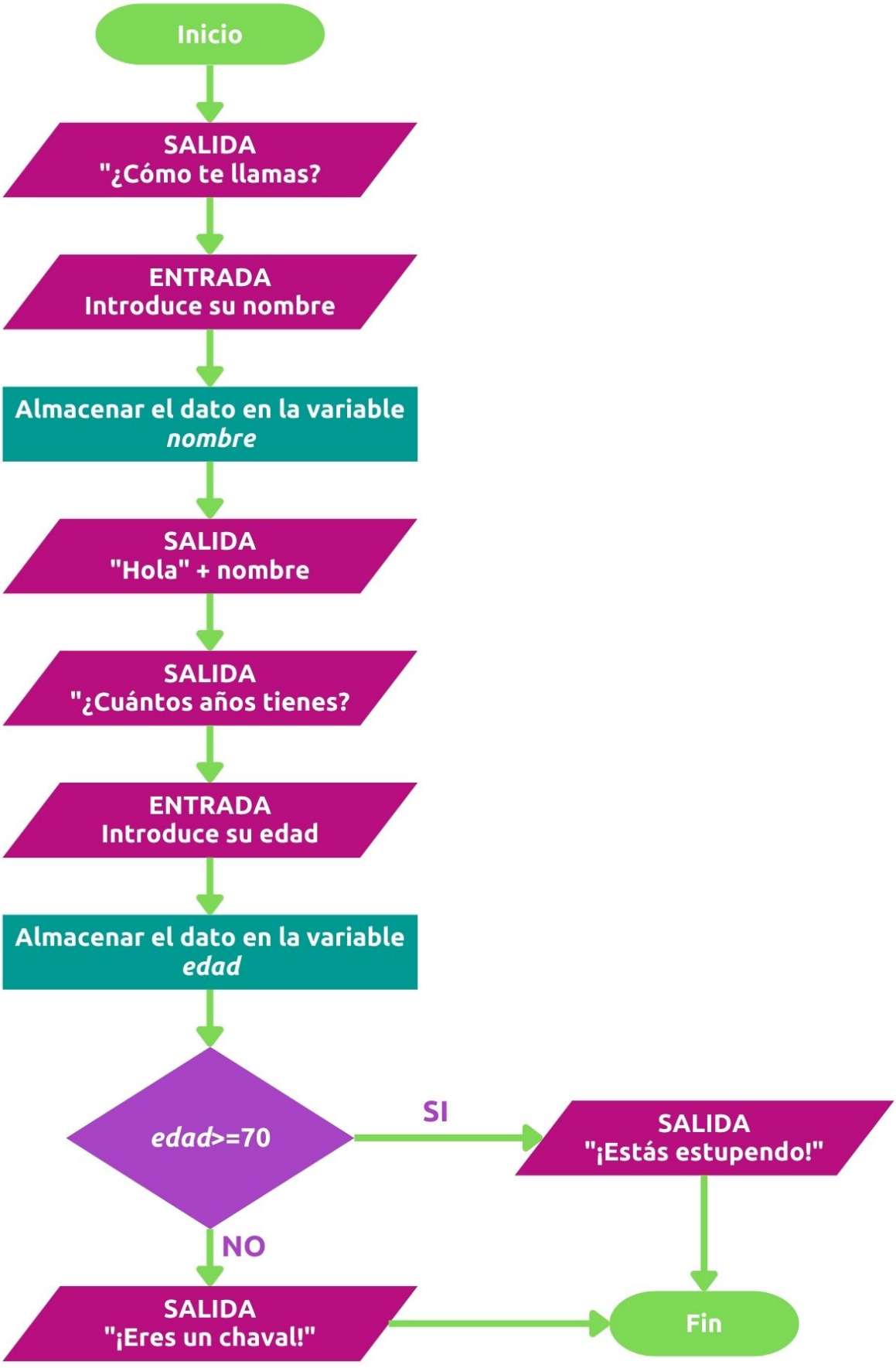
Representación un algoritmo: diagramas de flujo

Un diagrama de flujo es una **representación** de un conjunto de instrucciones. Los diagramas de flujo normalmente usan **símbolos** estándar para representar las diferentes instrucciones. Hay pocas reglas reales sobre el **nivel de detalle** necesario en un diagrama de flujo. A veces, los diagramas de flujo se dividen en muchos pasos para brindar muchos detalles sobre lo que está sucediendo exactamente. A veces se simplifican para que una serie de pasos ocurran en un solo paso.

Símbolos del diagrama de flujo



Se podría crear un programa simple para preguntarle a alguien su nombre y edad, y hacer un comentario basado en esto. Este programa **representado** como un diagrama de flujo se vería así:



Los diagramas de flujo pueden emplearse para resolver problemas de cualquier naturaleza, sin

importa el tema ni la dificultad (de hecho, cuanto más complejo es el problema más útil) Veamos un **ejemplo**:

https://www.youtube.com/embed/3sU6KIxjqMI

> **IDEA**. Se pueden crear diagramas de flujo con lápiz y papel pero también hay aplicaciones interesante para hacer diagramas de flujo. Una de ellas es **Diagrams**. Esta aplicación gratuita y de código abierto ayuda a crear una gran variedad de diagramas en Windows, Linux y macOS, agregando imágenes, configurando los textos y eligiendo el tipo de fuente, etc. El alumnado tendrá la opción de diseñar diagramas de flujo, mapas mentales.... y otro tipo de representaciones aunque para este curso con las que hemos nombrado tenemos suficiente. Su interfaz es muy práctica e intuitiva e incorpora capas que facilitan la edición. Otra opción es **Mindomo**. En este caso no hablamos de software libre. Se trata de un software colaborativo en línea para realizar diagramas de flujo, mapas mentales... y otras muchas cosas... aunque para este curso estas nos parecen las más adecuadas. Software Freemium ofrece sus servicios básicos de forma gratuita, si bien se cobra por las funciones prémium. La versión gratuita tiene un límite de tres mapas mentales y funciones prémium inhabilitadas, como carga de audio/video, descarga de PDF, PowerPoint y Excel y protección con contraseña. El software se puede usar desde cualquier navegador web estándar o instalando la aplicación gratuita para escritorio, iPad y Android.

Evaluación de soluciones

Antes de poder programar las soluciones, es importante **asegurarse de que satisfagan adecuadamente el problema** y que lo hagan de manera **eficiente**. Esto se hace a través de la evaluación.

¿Qué es evaluación?

Una vez que se ha diseñado una solución utilizando el pensamiento computacional, es importante asegurarse de que la **solución** sea **adecuada** para su propósito.

La evaluación es el proceso que nos permite asegurarnos de que nuestra solución hace el trabajo para el que ha sido diseñada y pensar en cómo se podría mejorar.

Una vez escrito, se debe **verificar** un algoritmo para asegurarse de que:

- Se entiende fácilmente, ¿está completamente descompuesto?
- Está completa, ¿resuelve todos los aspectos del problema?
- Es eficiente, ¿resuelve el problema, haciendo el mejor uso de los recursos disponibles (p. ej., lo más rápido posible usando el menor espacio)?
- Cumple con cualquier criterio de diseño que se nos haya dado

Si un algoritmo cumple con estos cuatro criterios, es probable que funcione bien. Entonces se puede programar el algoritmo.

La falta de evaluación puede dificultar la escritura de un programa. La evaluación ayuda a **garantizar** que se nos encontraremos con la **menor cantidad posible de problemas** al programar la solución.

¿Por qué necesitamos evaluar nuestra solución?

El pensamiento computacional ayuda a resolver problemas y a diseñar una solución, un algoritmo, que se puede usar para programar un ordenador. Sin embargo, **si el algoritmo falla**, puede ser difícil escribir el programa. Peor aún, es posible que el programa terminado no resuelva el

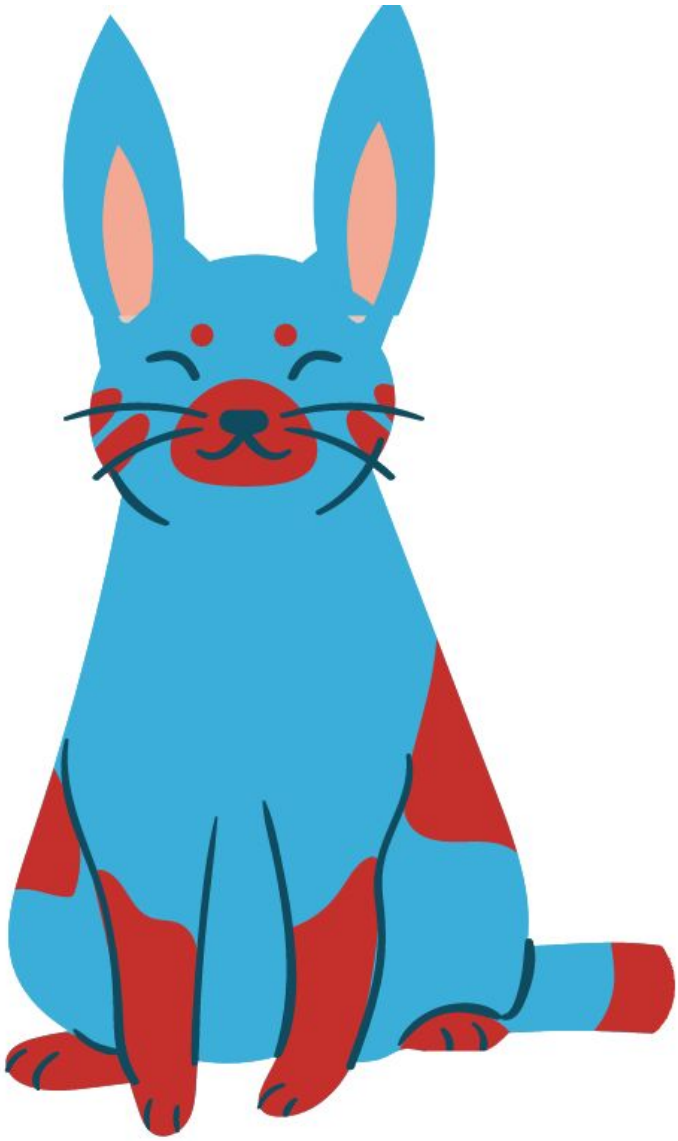
problema correctamente.

La evaluación nos permite considerar la solución a un problema, asegurarnos de que cumpla con los criterios de diseño originales, produzca la solución correcta y se ajuste al propósito, **antes de que comience la ejecución.**

¿Qué ocurre si no evaluamos una solución?

Una vez que se ha decidido una solución y se ha diseñado el algoritmo, puede ser **tentador** perderse la etapa de evaluación y comenzar a la etapa de ejecución de inmediato. Sin embargo, sin la evaluación, **no se detectarán los errores** en el algoritmo, y es posible que el programa no resuelva correctamente el problema o no lo resuelva de la mejor manera.

Los fallos pueden ser menores y no muy importantes. Por ejemplo, si se creó una solución a la pregunta "¿cómo dibujar un gato?" y esta tenía errores, todo lo que estaría mal es que el gato dibujado podría no parecerse a un gato.



Sin embargo, los errores pueden tener **efectos enormes y terribles**, por ejemplo, si la solución para el piloto automático de un avión tuviera defectos.

Formas en que las soluciones pueden ser defectuosas

Podemos encontrar que las soluciones fallan **porque**:

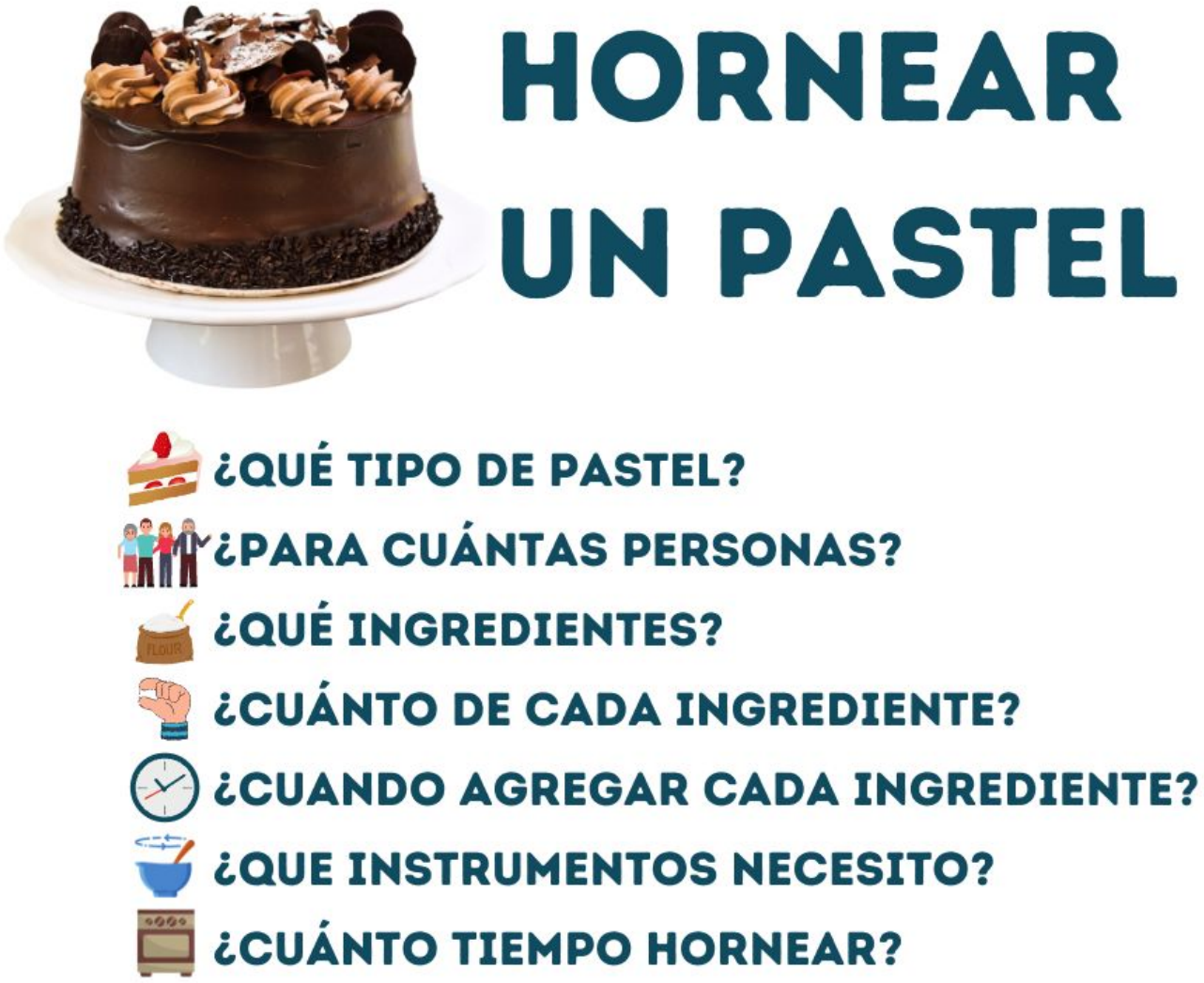
- **El problema no se entiende completamente**; es posible que no lo hayamos descompuesto correctamente

- **La solución está incompleta**; es posible que algunas partes del problema se hayan omitido accidentalmente
- **Es ineficiente**; la solución puede ser demasiado complicada o demasiado larga
- **No cumple con los criterios de diseño originales**, por lo que no es adecuada para su propósito

Una solución defectuosa puede incluir uno o más de estos errores.

Soluciones que no se descomponen correctamente

Si se aplican técnicas de pensamiento computacional al problema de cómo hornear un pastel, al descomponer el problema, es necesario saber:

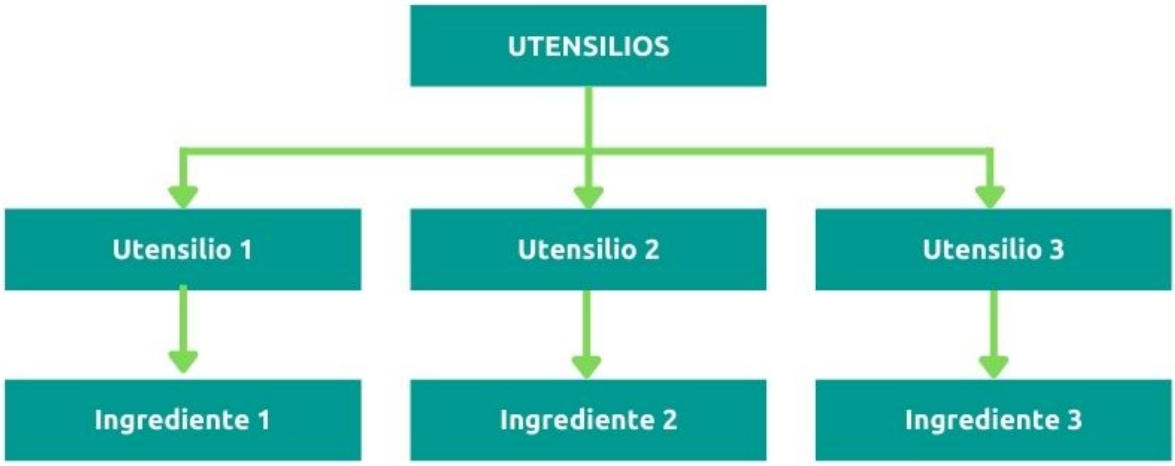


Un diagrama de una **descomposición de ingredientes** se vería así:

Por el momento, un diagrama de la descomposición adicional del equipo necesario se vería así:

La parte de 'Utensilios' **no está correctamente desglosada (o descompuesta)**. Por lo tanto, si la solución, o el algoritmo, se creara a partir de esto, hornear el pastel tendría problemas. El algoritmo diría qué utensilio se necesita, pero no cómo usarlo, por lo que una persona podría terminar tratando de usar un cuchillo para medir la harina y una batidora para cortar un trozo de mantequilla, por ejemplo. Esto sería incorrecto y, por supuesto, no funcionaría.

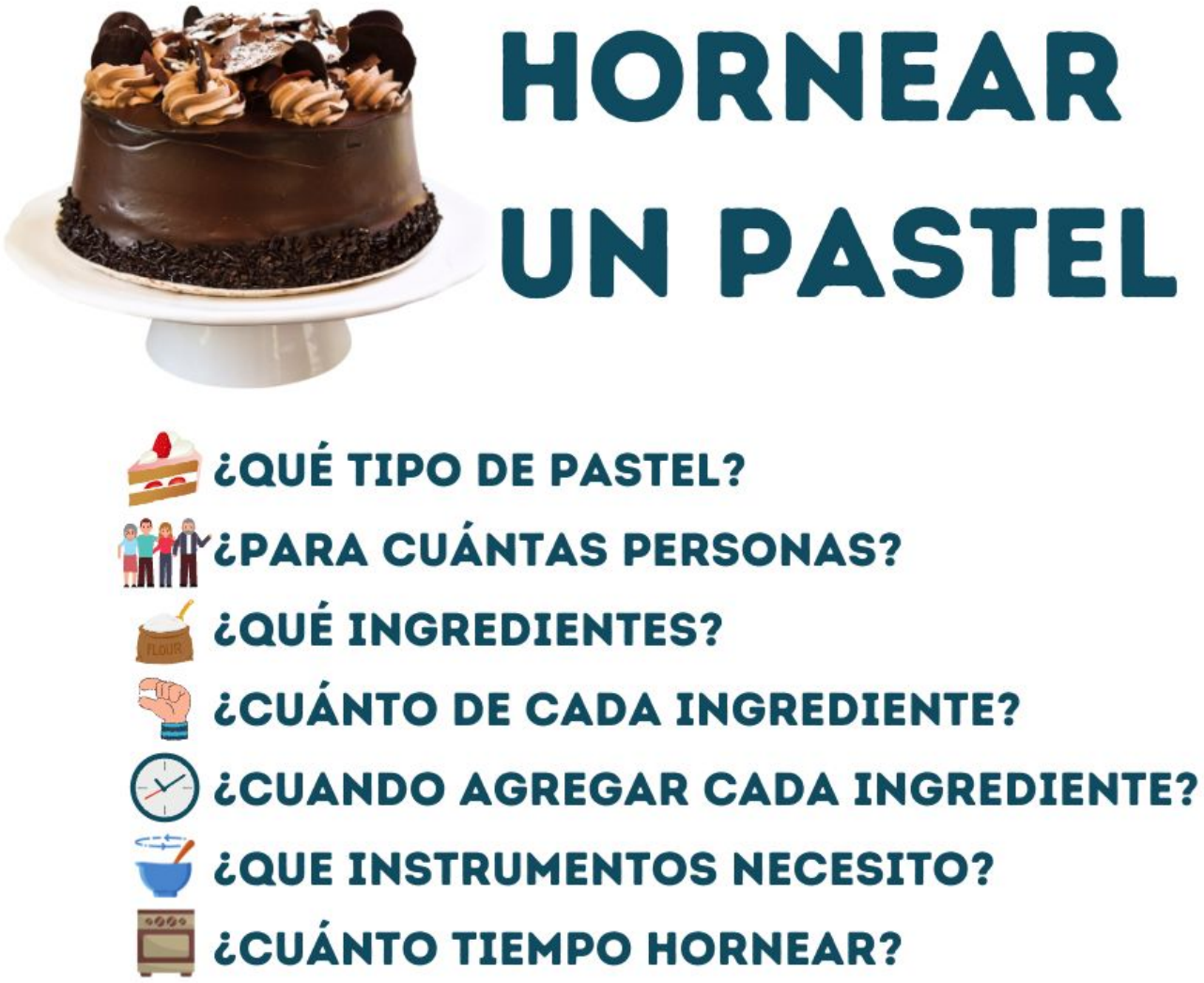
Idealmente, entonces, 'Utensilios' debería descomponerse aún más, para indicar **qué utensilios se necesitan y con qué ingredientes.**



El error se da aquí porque el problema de qué equipo usar y con qué ingredientes usarlo no se había descompuesto por completo.

Soluciones incompletas

Si se aplican técnicas de pensamiento computacional al problema de cómo hornear un pastel, al descomponer el problema, es necesario saber:



Sin embargo, esto está incompleto: se ha **omitido** parte del problema. Todavía necesitamos saber:

- Dónde hornear el pastel
- A qué temperatura hornear el pastel.

Por lo tanto, si esta información se usara para crear la solución, el algoritmo diría cuánto tiempo se debe hornear el pastel, pero no indicaría que se debe colocar el pastel en el horno o la temperatura a la que debe estar el horno. Incluso si el pastel llegó al horno, podría terminar poco cocido o quemado hasta convertirse en cenizas.

Se han omitido factores muy importantes, por lo que las posibilidades de hacer un gran pastel son escasas.

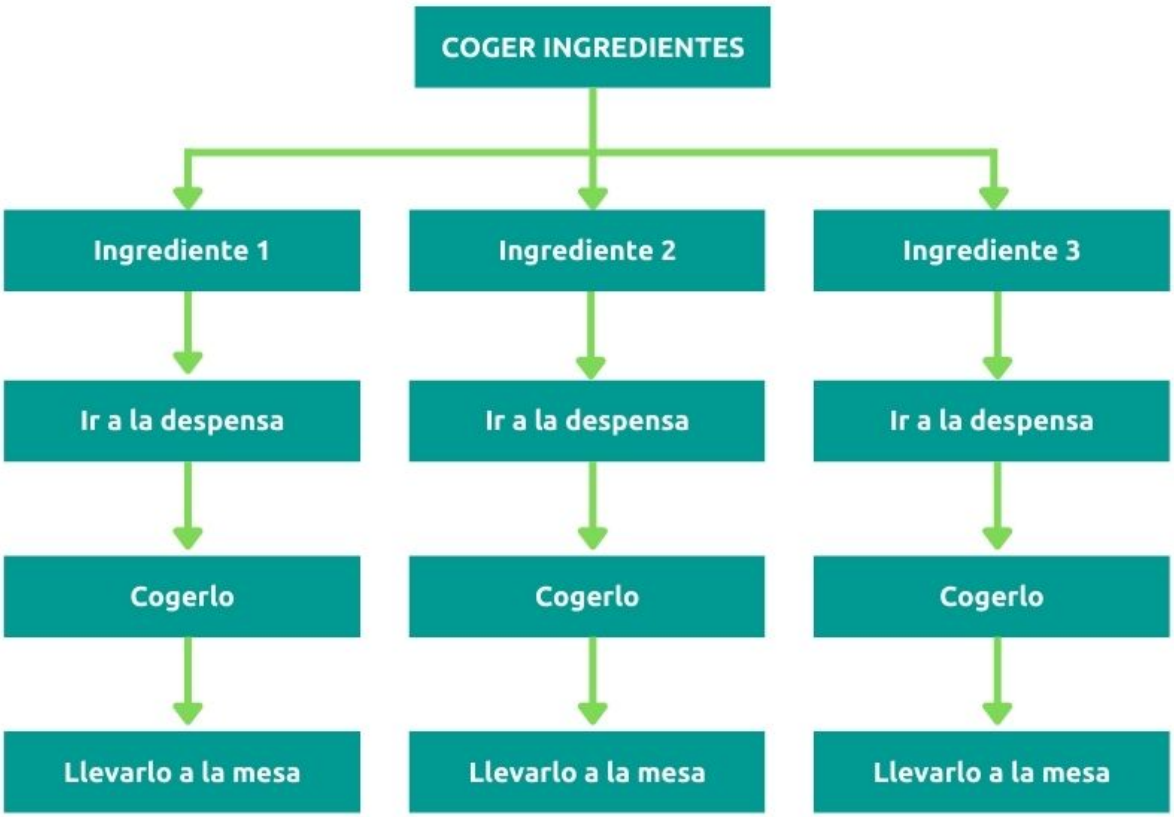
El error se dio aquí porque no se había incluido colocar el pastel en el horno y especificar la temperatura del horno, lo que hacía que la solución fuera incompleta.

Soluciones ineficientes

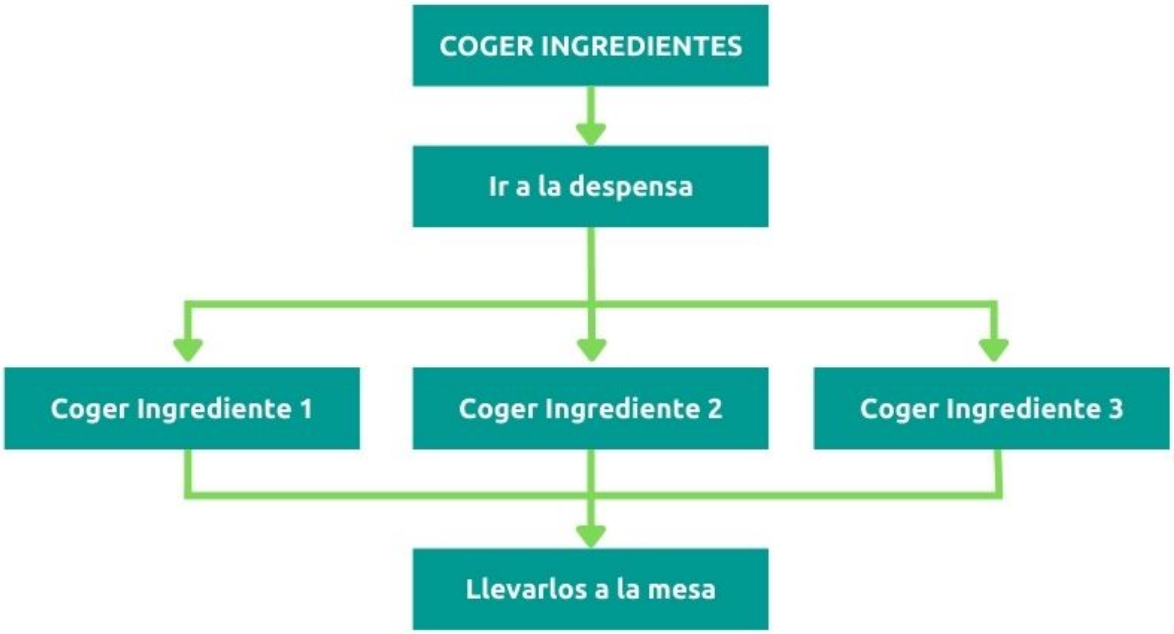
Si se aplican técnicas de pensamiento computacional al problema de cómo hornear un pastel, al descomponer el problema, la solución sería, entre otras cosas, que se necesitan **ciertas cantidades de ingredientes** particulares para hacer el pastel.

Para el primer ingrediente, podría decirnos que vayamos a la despensa, tomemos el ingrediente y lo traigamos de vuelta a la mesa. Para el segundo, y todos los demás ingredientes, podría decirnos que hagamos lo mismo.

Si el pastel tuviera tres ingredientes, eso significaría **tres viajes a la alacena**. Si bien el programa funcionaría así, sería innecesariamente largo y complicado:



Sería más eficiente obtener todos los ingredientes de una sola vez y, como resultado, el programa sería más corto:



La solución ahora es más **simple y eficiente**, y se ha reducido de nueve pasos a cinco.

El problema ocurrió aquí porque algunos pasos se repitieron innecesariamente, lo que hizo que la solución fuera ineficiente y demasiado larga.

Soluciones que no cumplen con los criterios de diseño originales

Las soluciones deben evaluarse según la **especificación original** o los **criterios de diseño** cuando sea posible. Esto asegura que la solución no se haya desviado demasiado de lo que se requería originalmente, que solucione el problema original y que sea adecuado para los usuarios.

Imagina tener que aplicar el pensamiento computacional al problema de cómo hornear un pastel. Al descomponer el problema, es necesario saber:

HORNEAR UN PASTEL

¿QUÉ TIPO DE PASTEL?
¿PARA CUÁNTAS PERSONAS?
¿QUÉ INGREDIENTES?
¿CUÁNTO DE CADA INGREDIENTE?
¿CUANDO AGREGAR CADA INGREDIENTE?
¿QUE INSTRUMENTOS NECESITO?
¿CUÁNTO TIEMPO HORNEAR?

El primer punto considera qué tipo de pastel hornear. A menudo, cuando se idean soluciones a problemas, se da una especificación para el diseño. Por ejemplo, es posible que el pastel tenga que ser un pastel de chocolate, que sigue siendo bastante general, o un pastel de dulce de chocolate con glaseado de chocolate y almendras por encima, que es más específico.

Para cumplir con los criterios de diseño, es importante asegurarse de hornear **exactamente el tipo correcto de pastel**. De lo contrario, la solución puede no ser adecuada para su propósito.

El error ocurrió aquí porque la solución no cumplía con los criterios de diseño originales, no era exactamente lo que se solicitaba.

¿Cómo evaluamos nuestra solución?

Hay varias formas de evaluar las soluciones. Para estar seguro de que la solución es correcta, es importante **preguntar**:

¿Tiene sentido la solución?

¿Entiendo ahora completamente cómo resolver el problema? Si todavía **no tengo claro** cómo hacer algo para solucionar nuestro problema, debo volver atrás y asegurarme de que todo se haya descompuesto correctamente. Una vez que sepa cómo hacer todo, nuestro problema se descompone completamente.

¿La solución cubre todas las partes del problema?

Por ejemplo, si dibujo un gato, ¿la solución describe **todo lo necesario** para dibujar un gato, no solo los ojos, la cola y el pelaje? De lo contrario, debo regresar y seguir agregando pasos a la solución hasta que esté completa.

¿La solución pide que se repitan las tareas?

Si es así, ¿hay alguna manera de reducir la repetición? He de regresar y **eliminar las repeticiones innecesarias** hasta que la solución sea eficiente.

Una vez que esté satisfecho/a con una solución, puedo pedirle a otra persona que la **revise**. Otro punto de vista suele ser bueno para detectar errores.

Es importante que tu solución siga los principios **KISS** y **DRY**.

> Los principios **KISS** (Keep It Simple, Stupid) y **DRY** (Don't Repeat Yourself) son dos conceptos importantes en el desarrollo de software y en la generación de algoritmos en general. El principio **KISS** se refiere a mantener las cosas **simples en el diseño y desarrollo**. Sugiere que es mejor optar por soluciones **simples** y **directas** en lugar de complicar innecesariamente el proceso. Por otro lado, el principio **DRY** se enfoca en evitar la repetición de tareas (o código) y de información en un sistema. En lugar de duplicar las tareas o la lógica, se promueve la creación de **abstracciones.** Aplicado a programación promueve la **reutilización** de código a través de funciones, clases o módulos. Esto reduce la duplicación, mejora la legibilidad y facilita los cambios, ya que solo se requiere modificar una única fuente de información. Ambos principios buscan fomentar la **claridad y** la **eficiencia.** Para más información sobre estos principios podéis consultar aquí.

Simulacro

Una de las mejores formas de probar una solución es realizar lo que se conoce como "simulacro". **Con lápiz y papel, trabaja en el algoritmo y traza un camino a través de él**.

Por ejemplo, en el apartado de Algoritmos, se creó un algoritmo simple para preguntarle a alguien su nombre y edad, y hacer un comentario basado en estos. Puedes probar este algoritmo usando dos edades, 15 y 75 años. Al usar 75 años, ¿adónde va el algoritmo? ¿Da la salida correcta? Si usas la edad de 15 años, ¿te lleva por un camino diferente? ¿Sigue dando la salida correcta?

Si el simulacro no da la respuesta correcta, hay algo mal que debe corregirse. Revisar la ruta a través del algoritmo ayudará a mostrar dónde está el error.

Los simulacros también se utilizan con programas completos. Los programadores usan ensayos para ayudar a encontrar errores en el código de su programa.

> **IDEA.** Puedes jugar a que se **intercambien sus soluciones para evaluarlas y detectar fallos**. Dependiendo de cuál haya sido la actividad propuesta, por ejemplo, si hemos hecho lo del pastel, pues se trataría de que otro grupo revisara la solución. Esto puede hacerse con papel y lápiz o pueden pequeñas representaciones teatrales de la solución: Voy a hacer como que hago un pastel con estas instrucciones. También puedes darles soluciones que tengan fallos y animarles a encontrarlos.

Caso práctico

Te proponemos aplicar el **pensamiento computacional** a un **problema real** al que podríamos tener que enfrentarnos. Resolveremos este reto paso a paso viendo que herramientas o alternativas podemos usar en cada uno de ellos.

Como ya hemos visto, el pensamiento computacional consta de 4 pilares básicos y cada uno de ellos es imprescindible. En el aula lo recomendable es proponer actividades para ejercitar cada uno de ellos. De esta manera puedes ver fácilmente si alguno de ellos necesita trabajo extra por parte de alumnado y reforzarlo con actividades específicas. Los cuatro pilares están interconectados por lo que es frecuente que usemos varios a la vez. Cuando decimos practicar uno, nos referimos al predominante, no quedando otros excluidos

Reto: Prepara un día en la playa de Salou con un amigo.

No me tengo que olvidar de la crema, ni de coger algo para beber, ni del bañador, ni de las gafas de sol, ni de ……

Veamos si somos capaces de poner orden en el problema para no dejarnos nada.

Siguiendo los pilares del pensamiento computacional debemos plantearnos:

Descomposición: ¿Podemos descomponer el problema en partes más simples?

Abstracción: ¿Podemos ver qué detalles son relevantes y cuales irrelevantes?

Reconocimiento de patrones: ¿Podemos recordar viajes similares o experiencias parecidas con esta amiga o amigo?

Algoritmo: ¿Podemos preparar un plan de acción?

Descomposición

Intentemos descomponer el reto en retos más simples que podamos afrontar fácilmente:

¿Cómo vamos a ir a Salou?

¿Hora de salida y de vuelta?

¿Qué comida y bebida vamos a llevar? ¿O mejor comprarla allí?

¿Qué ropa vamos a llevar?

¿Cómo nos protegeremos del sol?

¿Qué prevemos hacer en Salou?

Intentemos resolver cada uno de estos problemas. Algunos los resolveré yo y otros mi amigo:

¿Cómo vamos?

Queremos aprovechar todo el día, con lo que iremos a primera hora y volveremos para llegar a casa a las 9 ó 10 de la noche. Si llevamos el coche, tendremos que pagar el parking y en Salou no es nada fácil aparcar. Miraremos si hay viajes en autobús y si quedan plazas.

Mi amigo ha ido varias veces en autobús a Salou, con lo que es más fácil que se encargue él de mirar autobús, horarios... ya que ya sabe con qué compañía hacerlo.

¿Comidas?

Tenemos que comer al mediodía, tomar algo a media mañana y algo para merendar. Cenaremos en casa.

A media mañana podemos tomar algo de fruta.

Para comer podemos llevar bocadillos y comprar la bebida allí, para que esté fresquita. Y a media tarde, podemos tomarnos un helado.

Yo me puedo encargar de comprar algo de fruta y de preparar los bocadillos. Hablaré con mi amigo para ver de qué le gusta el bocadillo.Tendré cuidado de no usar ingredientes que se puedan

estropear con el calor.

No hay que olvidarse de coger algo de dinero.

La cena podemos hacerla en casa.

¿Ropa?

Pensemos en el calzado para la playa: ¿chancletas?

¿Bañador y bañador de repuesto?

¿Un pantalón corto y camiseta será suficiente?

¿Gorra para proteger la cabeza del sol?

Un jersey para el autobús, que a veces hace frío con el aire acondicionado.

¿Y cómo nos protegemos del sol?

Gafas de sol.

Crema solar de protección alta, que vamos a estar todo el día.

¿Sombrilla?

¿Qué haremos?

Por la mañana estaremos en la playa.

A media mañana podemos tomar la fruta.

Al mediodía nos tomamos los bocatas y el café en algún chiringuito.

A media tarde un helado.

Y a las 8 cogeremos el bus de vuelta.

Como vemos en este ejemplo la descomposición puede hacerse en varios niveles de detalle. Cada parte del problema puede a su vez subdividirse. En nuestro caso de qué es el bocadillo supone desglosar el punto referente a la comida. Cuanto mayor es el problema más necesario es aplicar la

descomposición en cada parte.

Reconocimiento de patrones

Al haber descompuesto el reto en problemas más pequeños, resulta más sencillo centrarse en cada uno de ellos y proponer distintas soluciones e incluso decidir quién de los dos amigos puede afrontarlo con más facilidad, por que ya lo ha hecho antes, ha tenido experiencias similares, tiene conocimientos sobre el tema en cuestión,…

En nuestro ejemplo podemos encontrar patrones en:

“Tendremos que pagar el parking, en Salou no es nada fácil aparcar” Hemos ido más veces, o tal vez nos han hablado de ello o incluso puede que lo extrapolemos teniendo en cuenta otras playas similares en época estival

“Mi amigo ha ido varias veces en autobús a Salou, con lo que es más fácil que se encargue él de mirar autobús, horarios… ya que ya sabe con qué compañía hacerlo.” En su momento esta personas buscaría que opciones de autobuses hay, es una parte del trabajo que ya está hecha y solo hace falta recuperar.

“Un jersey para el autobús, que a veces hace frío con el aire acondicionado”. Nuestras experiencias previas en autobuses nos indican que aunque haga 30º conviene llevar una chaqueta.

Abstracción

Como podéis ver, nos hemos intentado centrar en los temas relevantes, obviando detalles que no lo son. No nos centramos en si la gorra para protegernos del sol debe ser de un tipo u otro, de una marca u otra o de un color u otro. Es importante que llevemos una crema de protección solar que evite que nos quememos la piel, pero no lo es tanto la marca de la misma.

Algoritmo

Y por último, deberíamos preparar un plan de acción. Como hemos visto el algoritmo se puede representar como pseudo-código o como diagrama de flujo

Primero deberemos comprobar que hay billetes de autobús y que el horario nos parezca correcto: salir hacia las 9 y volver hacia las 8. No tiene sentido afrontar el resto de los retos si no hemos

resuelto este tema.

Segundo, haremos un check list con todo lo que debemos llevar, diferenciando lo que es para uso individual, como las gafas de sol, de lo que cogeremos para usar los dos, como lo es la crema de protección solar, que ya tengo un bote en casa

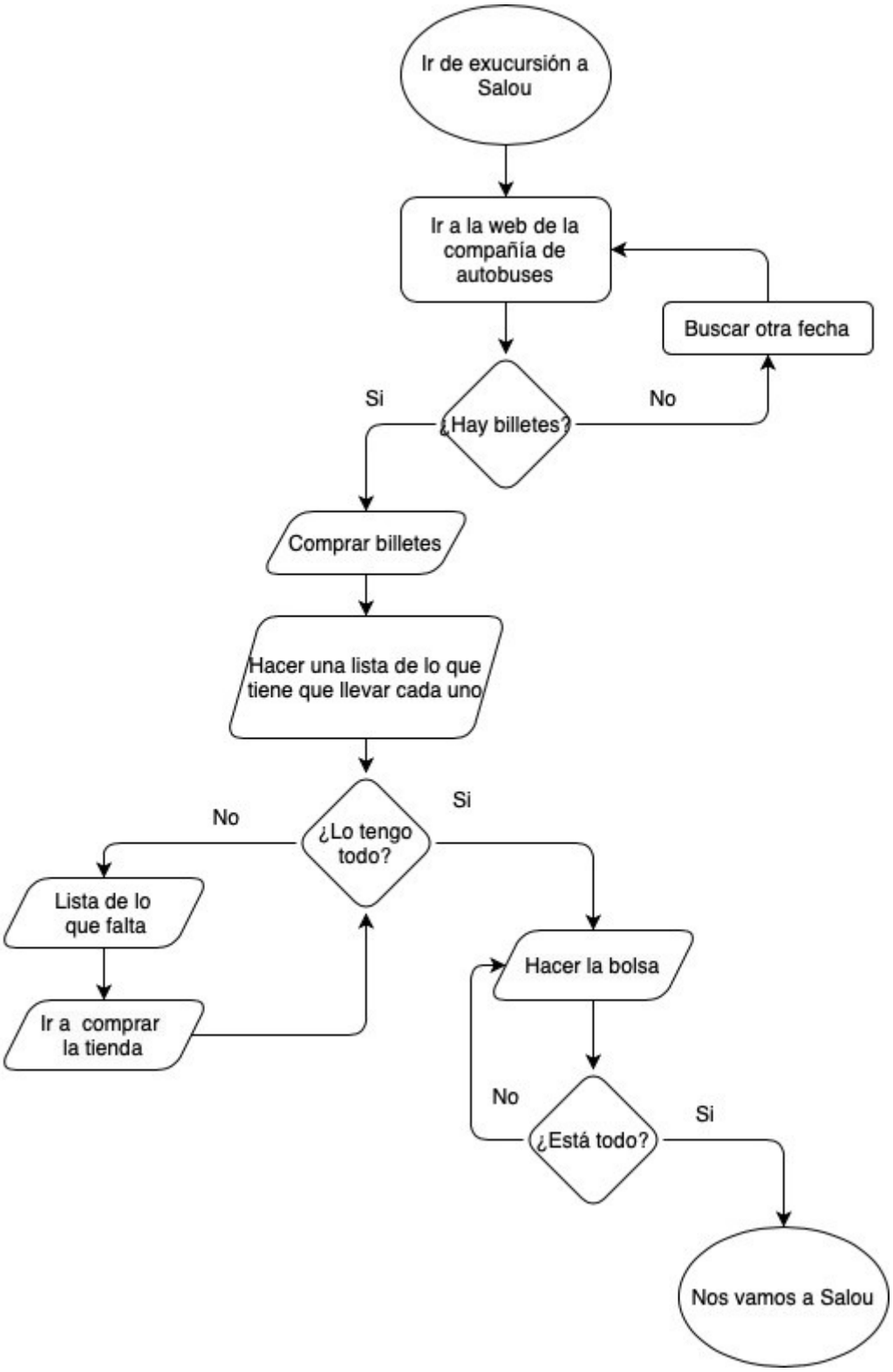
En este check list indicaremos quién se encarga de cada una de las tareas.

De manera esquemática podría escribirse como:

- Comprar billetes
- Preparar bolsa:
- Bañador
- Gafas de sol
- Gorra
- Dinero
- Fruta: yo
- Bocadillo: yo
- Crema de sol: Mi amiga
- Sombrilla: Mi amiga

Al hacer el check list nos hemos dado cuenta de que se nos olvidaba la toalla.

Diagrama de flujo:



Evaluación

Se debe entender fácilmente, en caso contrario es posible que no está suficientemente descompuesto. En nuestro caso, si no especificamos qué debemos llevar a la playa tal vez nos olvidemos cosas, como por ejemplo la toalla.

Debe resolver todos los aspectos del problema: En nuestro caso hay un punto que está sin resolver.... ¿Paras un minuto y piensas cual puede ser?

Debe ser eficiente. No tiene ningún sentido que cojamos dos sombrillas o que nos encarguemos los dos en llamar a la compañía de autobuses para ver horarios y plazas disponibles,....

Debe cumplir con los criterios iniciales. En nuestro caso, si ante el hecho de no haber billetes optamos por buscar otra playa a la que ir en el día que hemos seleccionado estaríamos incumpliendo los criterios iniciales que eran ir a Salou.

Es interesante poder hacer un simulacro. Puede hacerse en lápiz y papel repasando punto por punto o algo más teatralizado. Es interesante que participe en la evaluación alguien que no ha estado en el proceso.

Y ahora...¿Has encontrado ya el fallo de nuestra solución?

¡¡Efectivamente!! No hemos consultado la previsión del tiempo. Aunque pensemos que el tiempo está asegurado, puede no ser así.

Pensamiento computacional en el aula

En el aula es común que encontremos problemas de este tipo:

“Un trastero tiene una forma cuadrada de 5 metros de lado y 2m de alto . ¿Cuál es el área del suelo del trastero?"

Las tres fases para resolver estos problemas son:

- Correspondencia
- Identificación de reglas
- Aplicación de reglas

En la **primera fase** necesitamos encontrar los **datos necesarios** para la resolución del problema. Para ello es muy importante que sepamos **qué nos pide el problema**. Los datos relevantes no son los mismos si nos preguntan el área del suelo o cuánto aire cabe dentro trastero. Para completar la primera fase necesitamos saber que hablamos de un cuadrado de 5m de lado y que nuestra **incógnita** es el área.

A continuación tenemos que **identificar la regla** que relaciona los datos conocidos con los desconocidos. En nuestro caso el lado de un cuadrado con su área. La regla sería *área = lado x lado.*

Por último **aplicamos la regla**, sustituimos los datos conocidos y hallamos lo desconocido: 5 x 5= 25 m.

Este tipo de problemas son los que nos encontramos **más habitualmente** en el aula, tienen como resultado un número y además comparten otras características como:

- Son **deterministas** en su **solución**: La solución es única
- Son **deterministas** en el **proceso** de solución: Hay un proceso conocido hasta llegar a la solución

Nuestro alumnado frecuentemente está adiestrado para resolver este tipo de problemas, de hecho, si doy en el enunciado **más datos de los necesarios** para resolver un problema les generaré dudas. Incluso puede pasar que apliquen reglas que no corresponden al problema **con tal de usar todos lo datos**. A pesar de que estos planteamientos son los más comunes en el entorno escolar

no lo son en nuestro día a día.

Los problemas de la **vida real,** en general, **no son deterministas** ni en la solución (hay varias posibles) ni en el proceso de llegar a ellas.

Veamos dos problemas que podrían pasar en un cole:

Problema 1: “Cuando llueve, los pasillos del cole se mojan y el suelo resbala”

Las soluciones para este problema pueden ser:

- Construir una zona cubierta a la entrada del cole y en la transición entre las aulas y el patio
- Colocar felpudos
- Usar otras zapatillas dentro del cole
- ...

Estos son **tres** enfoques posibles y cada uno requeriría su nivel de desarrollo:

- ¿Qué dimensiones tiene que tener la zona cubierta?
- ¿Cuántos y de qué dimensiones tienen que ser los felpudos para ser efectivos? ¿Dónde tendrían que ir colocados?
- ¿Dónde realizar el cambio de zapatos? ¿ Qué estructuras o armarios habría que usar, pensando en tener espacio para el calzado de todo el alumnado?
- ¿Cuál es el costo aproximado para cada una de las soluciones propuestas?
- ¿Qué otros beneficios, además de evitar los resbalones, tiene cada solución?
- ¿Cuáles son los inconvenientes?

Problema 2: “Muchas veces no se oye bien en clase”

- Incluir material de aislamiento acústico en paredes y techos
- Reducir el número de alumnado por clase
- Crear espacios “ruidosos” y “ silenciosos” dentro de la escuela
- Mejorar los sistema de audio del centro
- Poner protectores a las patas de sillas y mesas
- ...

Al igual que en el caso anterior y cada enfoque requeriría su nivel de desarrollo e investigación:

- ¿Qué material de aislamiento usar? ¿Cuánto? ¿Qué hacer con las cristaleras?
- ¿Qué salas tiene el centro? ¿Todas están siempre en uso?
- ¿Hay recursos humanos suficientes? ¿Cómo podrían optimizarse?
- ¿Qué equipos de audio existen en las aulas? ¿Funcionan con la calidad adecuada? ¿Podrían mejorarse?
- ¿Cuántas mesas y sillas hay en el cole? ¿Qué tipo de protectores necesitan? ¿Habría que renovar mobiliario?
- ¿Cómo afectan otros factores, como por ejemplo la ventilación del aula cuando hace calor, a la acústica?
- ¿Cuál es el costo aproximado para cada una de las soluciones propuestas?
- ¿Qué otros beneficios, además de reducir el ruido tiene cada solución?
- ¿Cuáles son los inconvenientes?...

En ambos casos hay muchas respuestas y **la mayoría no son conocidas a priori**.

Son este **tipo de problemas** los que requerirán para su resolución más **habilidades** relacionadas con el **pensamiento computacional**. Su resolución, además, requiere de un **nuevo lenguaje**. La respuesta no va a ser un número. La respuesta tiene mucho que ver con el mismo proceso y su expresión nos debe permitir crear, simular, explorar, comprobar y experimentar.

En **resumen**, cuando hablamos de pensamiento computacional en el aula recuerda:

-

No tiene por qué tener nada de tecnológico. A lo largo del módulo hemosvisto diversas actividades, adecuadas para diversas edades, que no necesitan deeste tipo de soportes.

- **Trabaja en grupo**: Una de las riquezas de las actividades de pensamiento computacional es la discusión, la confrontación de ideas, ponerse de acuerdo.... Y, además, también es una parte divertida

- El **resultado** es importante, pero no más que el **proceso**

Para que las habilidades asociadas al pensamiento computacional se desarrollen debemos generar **situaciones adecuadas de aprendizaje**. Como todo en la vida, requieren de entrenamiento y práctica. Se adquieren de forma progresiva hasta alcanzar un grado en el que se convierten en operativas. Del entrenamiento de estas destrezas nos encargaremos en el módulo 2.