

1. Pensando como la computadora

1.1. Pensamiento computacional

El concepto del pensamiento computacional es un enfoque fundamental en la era digital en la que vivimos, y se ha convertido en una habilidad esencial para abordar problemas de forma lógica y estructurada, al igual que lo hacen las computadoras.

- ¿Qué es el pensamiento computacional?:

Es un enfoque mental y estratégico que nos servirá para resolver problemas y tareas que nos resultan muy complejos de realizar, pero siempre recordando que se basa en los principios fundamentales de la informática. Esto implica:

- La capacidad de descomponer problemas en partes más pequeñas
- La capacidad de identificar aquellas cuestiones que se repitan
- La capacidad de abstraer información esencial
- La capacidad de diseñar un paso a paso detallado que nos lleve a soluciones efectivas.

Se trata de un marco de pensamiento que se asemeja a la forma en que las computadoras procesan información, lo que ayuda a las personas a resolver problemas de manera lógica y estructurada, independientemente de la tecnología o los componentes específicos que se estén involucrando.

- Componentes del pensamiento computacional:

Esta metodología se compone de cuatro componentes que son esenciales y que trabajan en conjunto para abordar cualquier desafío que debamos resolver.

Estos componentes son:

01. **Descomposición:**

Es un componente que implica dividir un problema que es grande y complejo, en tareas pequeñas, y sobre todo, manejables para nosotros. Esto, no solo simplifica las tareas a hacer, sino que también permite abordar cada aspecto de manera más detallada y eficiente.

02. **Reconocimiento de patrones:**

Es la capacidad de identificar similitudes o regularidades en datos o situaciones. El reconocimiento de patrones se aplica a la resolución de problemas, ya que al observar similitudes con problemas anteriores, podemos aplicar soluciones conocidas a problemas nuevos.

03. **Abstracción:**

Consiste en simplificar una situación, enfocándose solo en los detalles que son relevantes, omitiendo cualquier información que sea innecesaria. Al abstraerse, se pueden tomar decisiones más informadas y eficaces.

04. **Algoritmos:**

Son un conjunto de pasos precisos, detallados y secuenciales que debemos seguir para resolver un problema o realizar una tarea específica. Son esenciales y se utilizan a diario para realizar cualquier tipo de tareas, como ordenar datos, buscar información en base de datos o realizar cálculos complejos.

Los programadores utilizan el pensamiento computacional para crear software, es decir, descomponen problemas complejos en tareas de programación más pequeñas, reconocen patrones en los datos que administran, abstraen conceptos que son clave y desarrollan algoritmos para que la computadora los ejecute.

Sin embargo, no solo se puede utilizar el pensamiento computacional en lo que es el desarrollo de software, sino que se lo puede utilizar también, en la vida cotidiana.

- Identificando los conceptos:

Cada elemento del pensamiento computacional puede estar involucrado en otro o viceversa, por ejemplo, se pueden ver patrones al abstraer, o abstraer mientras se descompone.

Es importante mencionar también, que los elementos tampoco son etapas secuenciales, es decir, no hay prioridades dentro de los componentes ni existe la necesidad de realizar uno antes que otro.

En conclusión, el pensamiento computacional es una habilidad fundamental en la sociedad digital actual. Al comprender y aplicar estos cuatro componentes, se pueden resolver problemas de manera más efectiva, tomar decisiones informadas y fundamentadas, y diseñar soluciones innovadoras en una amplia gama de campos.

1.2. Diagramas de flujo

Un diagrama de flujo es una herramienta visual que se utiliza para representar algoritmos de manera sencilla. Como se vio anteriormente, los algoritmos son una serie de pasos precisos (no presentan ambigüedades ni dan lugar a confusiones), ordenados (tienen un orden secuencial que deben obligatoriamente respetarse), y finitos (siempre tienen un inicio y un fin). Los algoritmos son diseñados constantemente por los programadores para llevar adelante el proyecto o software que deseen crear.

La creación de estos gráficos ayudan a expresar y acomodar las ideas que se tengan para los algoritmos. Ayudan también, a detallar el flujo de la información administrada y a modelar y modificar, en caso de ser necesario, las soluciones planteadas. Sin embargo, al momento de crearlos se deben tener en cuenta algunas características clave, ya que, en un diagrama de flujo, se utilizan diferentes formas para representar elementos específicos:

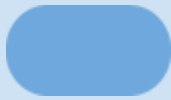






- Óvalo: **Terminal**, representa el inicio y el fin de un algoritmo.
- Rectángulo: **Proceso**, son para las acciones concretas que se deban realizar.
- Rombo: **Decisión**, representan decisiones o preguntas que se hagan a la computadora, y cuya respuesta se pueden responder solo con sí o no, o verdadero o falso.
- Flechas: **Flechas de Flujo**, indican la dirección del flujo de información.

¿Por qué son tan importantes los algoritmos en la programación?

Los diagramas de flujo permiten planificar, diseñar y entender todos los algoritmos que se realicen antes de sentarse a escribir el código.

Esto facilita la comunicación entre programadores, y también facilita la resolución de problemas de manera eficiente, ya que permite tener un pantallazo general antes de comenzar a escribir código.

Además, son útiles para identificar posibles problemas en el flujo lógico de un programa permitiendo una depuración en el tiempo adecuado. Por supuesto, va a ser mucho más sencillo y beneficioso encontrar un error o alguna cuestión que se deba modificar al momento de realizar estos diagramas, que cuando se tenga el proyecto avanzado y se requiera realizar muchos cambios en el código para poder abarcar las modificaciones necesarias.

Símbolo		Función
Terminal		Indicar el inicio y fin del diagrama.
Entrada/Salida		Entrada o salida simple de información.
Proceso		Realizar cualquier operación o cálculo con la información.
Salida a Impresora		Salida de información a la impresora.
Salida a Pantalla		Mostrar Información de salida a la pantalla.
Teclado		Introducir datos manualmente por el teclado.
Decisión		Indica operaciones lógicas o de comparación y tienen dos salidas dependiendo del resultado.
Conectores		Una o dos partes del diagrama a la misma o diferente página.
Flechas de Flujo		Indica la dirección del flujo de la información.

En conclusión, los diagramas de flujo son una herramienta valiosa para el programador, y en muchas otras áreas de la vida también. Nos ayudan a visualizar procesos, a tomar decisiones lógicas, y fundamentalmente, a solucionar problemas de manera eficiente, antes de comenzar el proyecto.

1.3. Paradigmas de programación

- ¿Qué son los paradigmas de programación?:

Son enfoques o estilos de programación que determinan cómo se escribe y estructura un código para resolver problemas. Cada paradigma tiene sus propias reglas, principios y técnicas.

- Programación Imperativa:

Este paradigma se centra en describir explícitamente los pasos necesarios para lograr un resultado. Los programas imperativos se componen de instrucciones secuenciales que modifican el estado del programa.

Dentro de la programación imperativa se destacan tres enfoques:

● **Programación Estructurada:**

Es un tipo de programación imperativa donde el flujo de control se define mediante bucles anidados, condicionales y subrutinas, en lugar de a través de GO-TO.

GO-TO: es una instrucción que permite saltar de una sección del código a otra, sin ejecutar el código intermedio. Es decir, permite cambiar el flujo de ejecución del programa. Se utiliza para implementar estructuras de control de flujo, como bucles y condicionales, y también para saltar a una sección específica del código en función de alguna condición.

Es un enfoque que se basa en organizar el código en una serie de estructuras lógicas simples. Estas estructuras son tres:

- Estructuras de Secuencia: ejecutan instrucciones una tras otra, siempre manteniendo un orden.
- Estructuras de Selección: se trata de tomar decisiones basadas en condiciones que se pueden cumplir o no, y en base a ello, realizar una u otra acción.
- Bucles: se trata de repetir un conjunto de instrucciones que se detallan una sola vez, pero que permite repetirlas la cantidad de veces que sea necesario hasta que se cumpla una condición.

● **Programación Procedimental:**

Consiste en basarse en un número muy bajo de expresiones repetidas, englobadas todas en un procedimiento o función y llamarlo cada vez que tenga que ejecutarse.

Es una extensión de la programación estructurada que se centra en organizar el código en distintos procedimientos o funciones. Cada uno de estos procedimientos o funciones, es un conjunto de instrucciones que van a interactuar entre sí para realizar una tarea específica, y que se encuentran englobados bajo un único nombre.

Este enfoque permite repetir estos procedimientos la cantidad de veces que sean necesarias.

● **Programación Modular:**

Consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más manejable y legible. Se trata de una evolución de la programación estructurada para resolver problemas de programación más complejos.

Lleva la idea de funciones o procedimientos un paso más allá, al organizar el código en módulos o componentes que sean reutilizables. Cada uno de estos módulos contiene varias funciones o procedimientos que están relacionados entre sí y que se van a poder utilizar en diferentes partes del programa.

- Programación Declarativa:

Este paradigma se centra en lo que se quiere lograr en lugar de cómo se quiere lograrlo.

Enfoques:

- **Programación Lógica:**

Un programa puede ser descrito definiendo ciertas relaciones entre conjuntos de objetos, a partir de las cuales otras pueden ser calculadas empleando deducción.

Se basa en brindar declaraciones para describir hechos y reglas, y luego pedirle al sistema que derive a conclusiones lógicas a raíz de esas declaraciones previamente definidas. Es decir, no es necesario escribir un conjunto de pasos algorítmicos explícitos, sino que simplemente se aplican las reglas lógicas que se necesitan para deducir una solución.

- **Programación Funcional:**

Se basa en la evaluación de expresiones, no en la ejecución de instrucciones.

Se basa en el concepto de funciones puras, que son como cajas negras que toman datos de entrada y producen datos de salida sin afectar el entorno. En lo que es programación funcional, se van a construir los programas utilizando muchas de estas cajas negras, en la que cada una de estas, toma ciertos datos como entrada y produce un resultado de salida sin afectar nada más en el programa. Además, permiten ser combinadas para crear programas más grandes y más complejos.

Se enfoca en evitar cambiar las cosas dentro de un programa de maneras inesperadas, en otras palabras, no hay preocupación de que se modifique algo en otro lugar del código, permitiendo que los programas sean más fáciles de entender. La programación funcional se trata de utilizar estas cajas negras que se van codificando como piezas que encajan perfectamente produciendo resultados predecibles y que no rompen el resto de la creación.

- Programación Orientada a Objetos:

Se construyen modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones. Permite separar los diferentes componentes de un programa. Se acerca de alguna manera a cómo se expresarían las cosas en la vida real.

Se basa en la creación de objetos que contienen datos y métodos que operan esos datos. Se pueden utilizar estas declaraciones (propiedades o características) que son predefinidas, como una especie de molde y a raíz de esto, crear múltiples instancias del objeto con diferentes características (datos).

La ventaja que presenta este paradigma es que permite ahorrar tiempo y esfuerzo al poder reutilizar el código existente, permite modelar conceptos del mundo real, lo que facilita mucho la comprensión y el diseño de software, permite la protección de datos y limitar el

acceso a ciertas partes del proyecto, y brinda la posibilidad de volver al código más fácil de entender y de modificar.

- Programación Reactiva:

Se enfoca en construir sistemas que sean responsivos, resilientes, elásticos y orientados a mensajes.

- **Responsivo:**

Significa que responde rápidamente a eventos o solicitudes, lo que es esencial para aplicaciones en tiempo real.

- **Resiliente:**

Están diseñados para recuperarse de errores y fallas de manera efectiva manteniendo la disponibilidad y la integridad de los datos.

- **Elásticos:**

Significa que pueden adaptarse automáticamente al aumentar o disminuir la demanda, optimizando el uso de los recursos.

- **Orientados a mensajes:**

Los componentes del sistema se comunican entre sí mediante el intercambio de mensajes en lugar de llamadas directas, lo que permite una comunicación flexible y desacoplada.

- Comunicación y lenguaje:

La comunicación es la transmisión de señales de un emisor a un receptor mediante un código común (el lenguaje). Es una serie de señales que un emisor envía y un receptor recibe. Estas señales deben ser obligatoriamente en un lenguaje que ambos participantes comprendan, o de lo contrario la comunicación no se logrará de forma exitosa.

Esto lleva a pensar en cómo se va a lograr que la computadora pueda comprender los mensajes: la respuesta es **código**.

El código, son las instrucciones que se le van a dar a la computadora, pero que se las escriben con un conjunto de palabras clave y sintaxis específicas que harán que la computadora pueda comprender correctamente lo que se le esté solicitando.

- Lenguaje y ambigüedad:

Un lenguaje de programación es un conjunto de reglas y símbolos que permiten a los programadores, escribir instrucciones para una computadora. Estas instrucciones, conocidas como código, se utilizan para crear programas y software que controlan el comportamiento de la computadora.

Cada lenguaje de programación tiene su propia sintaxis y semántica, que determina cómo se deben escribir las instrucciones y cómo se interpretarán o compilarán para que la computadora las comprenda.

Los lenguajes de programación se utilizan para una gran variedad de tareas, desde el desarrollo de aplicaciones de software, hasta la creación de sitios web y automatización de tareas.

Por ejemplo: Python, Java, C, C++, C#, Javascript, entre muchos otros.

Es importante resaltar que cada uno de estos lenguajes tiene sus propias características y se utiliza en diferentes contextos según las necesidades del proyecto que se esté desarrollando.

Por su parte, la ambigüedad en programación se refiere a situaciones en las que el código escrito podría tener más de una interpretación o significado, lo que puede llevar a resultados inesperados o errores en los programas. Esto puede surgir cuando no se es lo suficientemente preciso al escribir las instrucciones que son para la computadora, o también cuando existen múltiples formas de comprender o ejecutar una parte del código.

Esto puede llevar a errores en el programa que hacen que sea muy difícil rastrear y corregir los problemas. Cuando los programadores no entienden claramente qué es lo que hace un código, se puede correr el riesgo de perder demasiado tiempo haciendo correcciones que no eran necesarias.

Para evitar que esto suceda, se acostumbra a hacer uso de lo que denomina como **buenas prácticas**, que son un conjunto de pautas, normas y enfoques recomendados, que garantizan que un código sea legible, mantenible, eficiente y libre de errores.

2. Integración de contenidos

2.1. Desafío práctico

Modelar un diagrama de flujo para cada uno de los ejercicios que siguen. Incluir todos los pasos necesarios y considerar detenidamente cuándo sería apropiado añadir preguntas o repeticiones de acciones para optimizar el proceso.

- Preparar un sándwich:

Crear un diagrama de flujo que represente el proceso de preparar un sándwich. Incluir pasos como seleccionar los ingredientes, armar el sándwich y servirlo.

- Decidir qué ropa usar para el día:

Crear un diagrama de flujo que modele el proceso de decidir qué ropa usar para el día. Considera factores como el clima, la actividad a realizar, etc.

- Hacer ejercicio en casa:

Crear un diagrama de flujo que represente las actividades para hacer ejercicio en casa. Incluir calentamiento, ejercicios aeróbicos, estiramientos, etc.

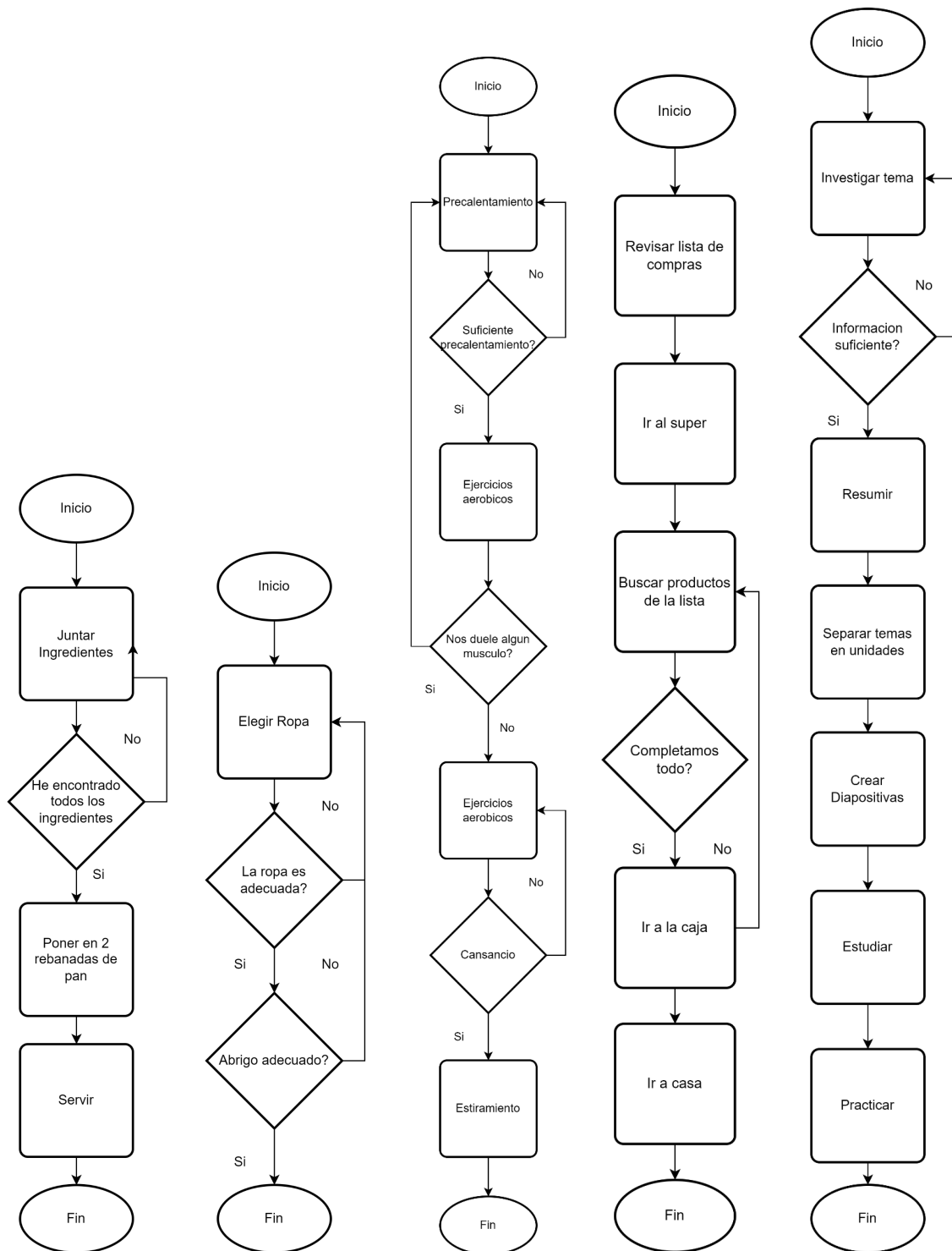
- Compra de víveres en el supermercado:

Diseñar un diagrama de flujo que modele el proceso de compra de víveres en el supermercado. Incluir acciones como hacer una lista de compras, seleccionar productos, etc.

- Preparar una presentación para la escuela/trabajo:

Crear un diagrama de flujo que represente el proceso de preparar una presentación para la escuela o el trabajo. Incluir pasos como investigar el tema, crear diapositivas, practicar la presentación, etc.

2.2. Resolución del desafío práctico



3. Checkpoint de conocimientos

- ¿Qué es el pensamiento computacional?

Un enfoque para descomponer problemas en pasos más pequeños y manejables

- ¿Qué son los algoritmos en el contexto del pensamiento computacional?

Conjuntos de pasos ordenados para resolver un problema

- ¿En qué consiste la abstracción en el pensamiento computacional?

Priorizar los datos relevantes ignorando la información que no aporte al cumplimiento del objetivo

- ¿Para qué se utilizan los diagramas de flujo en programación?

Para representar visualmente algoritmos o procesos

- ¿Cuál es el propósito principal de un diagrama de flujo?

Comunicar de manera visual el flujo de un proceso o algoritmo

- ¿Qué define a un paradigma de programación?

Un estilo o enfoque particular para resolver problemas de programación

- ¿Cuál es la principal diferencia entre la programación imperativa y la programación declarativa?

La programación imperativa se centra en describir el "cómo" se quiere lograr algo, mientras que la programación declarativa se enfoca en "qué" se quiere lograr.

- ¿Qué caracteriza a la programación lógica dentro del paradigma "Programación Declarativa"?

Se centra en establecer reglas lógicas y relaciones para la resolución de problemas