



LÓGICA DE PROGRAMACIÓN

Propuesta:

Si queremos aprender a programar en cualquier lenguaje de programación, el primer paso es aprender la lógica de programación.

Cuando programamos dictamos acciones al computador para que este las realice.

CONSEJOS

FOCO: Internet tiene muchísima información, buscar sitios confiables y documentación oficial elegir un lenguaje (no elegir muchos al mismo tiempo) ir paso por paso. Podemos elegir un proyecto y aprender desarrollándolo

ACTITUD: Tenemos que tener la idea ¡PODEMOS! dejar de verlo Imposible, cuando aprendamos las reglas y practiquemos nos daremos cuenta

¿para qué sirve la programación?

La programación es utilizada con 2 usos principales, **el primero es poder manipular datos**

Ejemplo: si tengo una lista de usuarios de una aplicación, con un lenguaje de programación puedo tomar cada elemento de esa lista, es de decir cada Usuario y mostrarlo en una página, esta es una manera de procesar datos.

Otra cosa que podemos hacer es tomar datos de números y sumarlos, o hacer operaciones.

El segundo uso, es poder darle instrucciones al hardware.

Ejemplo: podemos programar el hardware de nuestro pc para que se prenda la cámara de la computadora, controlar el teclado. Etc.

¿Qué es un algoritmo?

Pasos para encontrar una solución a problemas simples o complejos.

Características de un algoritmo:

Preciso: paso a paso en un orden lógico.

Definido: todas las veces que pasemos por él da el mismo resultado.

Finito: tenemos un proceso de inicio y de cierre.

Los algoritmos los podemos representar de manera:

Gráfica: a través de diagramas de flujo.



No gráfica: se usa un lenguaje de programación, o de manera textual.

Metodología para crear un algoritmo:

Definir el problema: El para qué.

Analizar el problema: Analizar cada uno de los detalles que lo componen, que metodología se aplicara

Diseñar el algoritmo: Iniciamos a escribirlo, a marcar todos aquellos pasos necesarios.

Prueba de escritorio: Tenemos entradas de prueba, para las cuales tenemos salidas esperadas

ESQUEMA

Para desarrollar tu lógica primero tenemos que conocer el esquema de la programación.

OPERACIONES: Son sencillamente operaciones matemáticas, (no te asustes) mayormente utilizaremos las operaciones básicas.

Ejemplo: En la programación si escribimos $2+2$ el computador realizará el cálculo y nos dará la respuesta que será 4

```
2+2
// 4
```

pero si queremos manipular este resultado o la operación misma para poder reutilizarla posteriormente ¿Qué debemos hacer?

Asignarle un nombre, una variable para que pueda ser identificada y utilizada constantemente

```
let sumar = 2+2;
```

VARIABLES

Como su nombre lo explica, son un objeto cuyo contenido Varía.

Ejemplo:

Jarra = Variable

Jugo = Contenido

```
let jarra = "Jugo de Naranja"
```



Por ejemplo, si pido el contenido de la variable jarra, el resultado será “Jugo de Naranja”

¿Qué valor nos devuelve si declaramos esto?

Sumar = Variable

2+2= Contenido

```
let sumar = 2+2;
```

Identifica las variables según los requerimientos de negocio

Es momento de identificar qué variables necesitarás para resolver las siguientes situaciones.

Cajero electrónico

Qué variables necesito para llevar a cabo el funcionamiento de un cajero automático en el cual solo realizaré la acción de sacar dinero en efectivo.

Usar un chat

Qué variables debo tener en cuenta para hablar con un amigo a través de una aplicación de mensajería instantánea, teniendo en cuenta que solo le puedo enviar mensajes si está conectado a la aplicación.

Pagar con tarjeta de crédito

Estás en la caja de un supermercado y necesitas realizar el pago de tus productos usando una tarjeta de crédito porque es el único medio de pago que reciben. ¿Qué variables necesitas?



Lavar la ropa

Necesitas lavar tu ropa, pero debes tener en cuenta los diferentes factores que necesitarás para llevar a cabo esta tarea. ¿Tienes lavadora? ¿Lo harás a mano? ¿Qué necesitas?

Hablar por teléfono

Es momento de llamar a un amigo por teléfono, debes tener en cuenta varios puntos importantes: ¿Tienes teléfono? ¿Está tu amigo disponible? ¿Qué necesitas para realizar esta llamada?

TIPOS DE DATOS

Este contenido son TIPOS DE DATOS más comunes que Podemos guardar son:

(STRINGS = TEXTO)

En este caso guardamos el nombre un libro.

```
let libro = "La Potencia del Talento No Mirado";
```

Strings (texto) y concatenación

Yo soy una cadena de texto".

Una cadena de texto es la unión de diferentes caracteres que funcionan como eslabones y al unirse forman una cadena o frase.

En el caso del ejemplo cada uno de los elementos de esta frase es un caracter, cada letra y cada espacio al ser concatenado nos da un sentido lógico de valor que al traducirlo es un string

Char: es un caracter.

String: es un conjunto de caracteres.

(NUMBER = NUMERO)



Un número (Pueden ser entero o decimal)

```
let enteros = 2020;  
let decimales = 1.9872;
```

Números y operaciones matemáticas básicas

Tipos de datos:

Int: los enteros son los que no tienen decimales como por ejemplo el 3, 5, 10, 12

Float: son lo contrario a los int más bien, tienen un número decimal como por ejemplo 2.4, 5.3...

Short: manejan 2 bytes.

Long: manejan 8 bytes.

Operadores para operaciones matemáticas

(+) Para realizar sumas

(-) Para realizar restas

(*) Para realizar multiplicaciones

(/) Para realizar divisiones

💡 Debemos tener cuidado con el tipo de dato y las operaciones porque podemos obtener resultados distintos. Debemos hacer una correcta conversión.

Operaciones matemáticas compuestas: paréntesis y orden de evaluación

Las operaciones matemáticas compuestas son aquellas que contienen varias operaciones en una sola.

Para resolver este tipo de operaciones tenemos que tener en cuenta la jerarquía de las operaciones la cual sigue el siguiente orden de solución.

Paréntesis.

Potencias y raíces.

Multiplicación y división

Sumas y restas.

BOOLEANS = BOOLEANOS

```
let libroDisponible = true;
```

True y false: booleanos y tablas de la verdad

Estos datos hacen referencia a lógica. Nos darán un resultado de verdadero o falso de acuerdo a su valor.



Los usaremos cuando vayamos a crear funciones, condiciones y estemos llevando nuestra lógica un escalón más arriba, pues son quienes definirán si algo está sucediendo o definitivamente no está pasando.

Tablas de verdad: Son las tablas que nos indican el resultado de acuerdo a la conexión que hagamos entre ellas.

Composición

Dado por dos proposiciones en este orden. Son cada uno de los valores que vamos a combinar para que nos den un resultado. El resultado será lo que la conectiva lógica defina entre cada uno de ellos.

(ARRAY = ARREGLO / MATRIZ)

Una lista puede contener varios datos de diferente tipo
Pueden contener: strings, numbers, booleans u otro array

```
let fundadores = ["Paula Cardenau", "Emiliano Fazio", Federico  
Seineldin", 3, true, ["Otro Array"]]
```

Arrays y funciones sobre arrays

Qué son los arrays?

Son un conjunto de elementos del mismo tipo ordenados en fila.

Su primera posición (index) será siempre 0.

También puedes almacenar números, caracteres y strings.

Su tamaño puede variar.

Puedes ordenarlo de la forma que tú lo requieras.

Puedes recorrerlo. Es decir, podemos pasar uno a uno sobre cada una de sus posiciones y operarlas.

Puedes acceder a una posición específica.

También podemos guardar este tipo de dato que puede ser Verdadero (true) o Falso (false):

FUNCIONES

Las funciones son muy importantes ya que aquí pondremos toda nuestra lógica de programación.

Son utilizadas para ingresemos datos, los procesemos y los saquemos, todo lo que entra va a salir, pero de una forma diferente.



Realizamos un conjunto de instrucciones que realizan tareas o calculan un valor, pero para que un procedimiento califique como función, debe tomar alguna entrada y devolver una salida donde hay alguna relación obvia entre la entrada y la salida. Para usar una función, debes definirla en algún lugar del ámbito desde el que deseas llamar

Por ejemplo: Podemos crear una función que se llame *sumar* en la que vamos a ingresar dos números (dos datos) y de resultado vamos a dar la suma

```
function sumar (a,b){  
  return a + b;  
}  
sumar (6+6); // 12
```

CONDICIONALES

Las «instrucciones **condicionales**» se usan para realizar las diferentes acciones según una condición

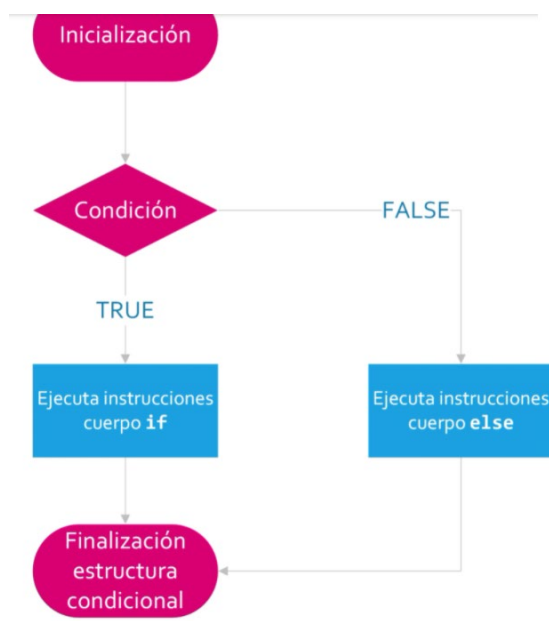
If y Else: condicionales y comparaciones para proteger tus algoritmos

Estructuras de control:

Herramientas sobre las cuales se construye un algoritmo. Nos ayudan a construir el flujo de nuestras tareas.

If/Else:

Si se cumple determinada condición se ejecuta acción/código anidada en el if, si no es así se ejecutará la anidada en el else. Esta es muy usada para validar información





Por ejemplo: Programar para verificar, si una persona es mayor de edad tenga autorización a ingresar el sitio, de lo contrario No

```
if (edad >= 18) {  
    autorizar ();  
}  
else {  
    noAutorizar();  
};
```

Podemos hacer más comparaciones, por ejemplo.

```
if (color === "verde") {  
    dibujarCirculo ();  
}  
else if (color === "amarillo") {  
    dibujarCuadrado ();  
}  
else {  
    dibujarTriangulo ();  
}
```

Switch y Case: condicionales en forma de casos

Qué es Switch y Case

Es una estructura de control, que nos permite evaluar múltiples casos que puede llegar a cumplir una variable y realizar una acción en esa situación.

Estructura del condicional switch.

```
switch(numero) {  
  case 1:  
    "El número es 1"  
    break;  
  case 2:  
    "El número es 2"  
    break;  
  case 3:  
    "El número es 3"  
    break;  
  default:  
    "El número es mayor a 3"  
}
```




CICLOS / BUCLES

¿Qué es un ciclo? While, For y Do While

¿Qué es un ciclo?

Es una estructura de control que ejecuta un bloque de instrucciones de manera repetida.

¿Cuándo utilizar un ciclo for, while o do while?

For: cuando sabes (o puedes saber) las veces repetirás el ciclo. Ejemplos: “5 veces”, “la cantidad de elementos que tiene un arreglo”.

While: Cuando no sabes las veces que se repetirá un ciclo. Ejemplos: “reintentar conectarme a una base de datos si falló al hacerlo”

Do While: Cuando no sabes las veces que se repetirá un ciclo y necesitas que se realice por lo menos una vez. Ejemplos: “Conectarme a la base de datos, si falló, repetir hasta que me pueda conectar”

Nos permiten procesar datos uno por uno especialmente en Arrays.

Por ejemplo: Si tenemos un array con lenguajes de programación y quiero imprimirlos en pantalla utilizamos el siguiente Bucle.

```
let lista = ["javascript", "python", "Java"];

for (let lenguaje of lista){
  console.log(lenguaje);
}

// "javascript"
// "python"
// "Java"
```

Otro ejemplo: si tengo una lista de números y queremos sumarle 5 puedo hacerlo con un Bucle.

```
let lista = [10, 20, 30];
```



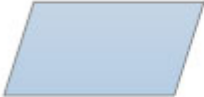


```
for (let numero of lista) {  
  console.log(numero+5);  
}  
// 15  
// 25  
// 35
```

¿Cómo diseñar algoritmos con diagramas de flujo?

Qué es un diagrama de flujo

Un diagrama de flujo es una representación gráfica de un proceso. Cada paso del proceso es representado por un símbolo diferente que contiene una breve descripción de la etapa de proceso. Los símbolos gráficos del flujo del proceso están unidos entre sí con flechas que indican la dirección de flujo del proceso.

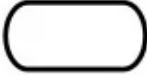







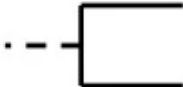


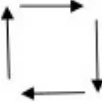




Elementos principales de un diagrama de flujo:

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Linea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

Podemos usar paginas como

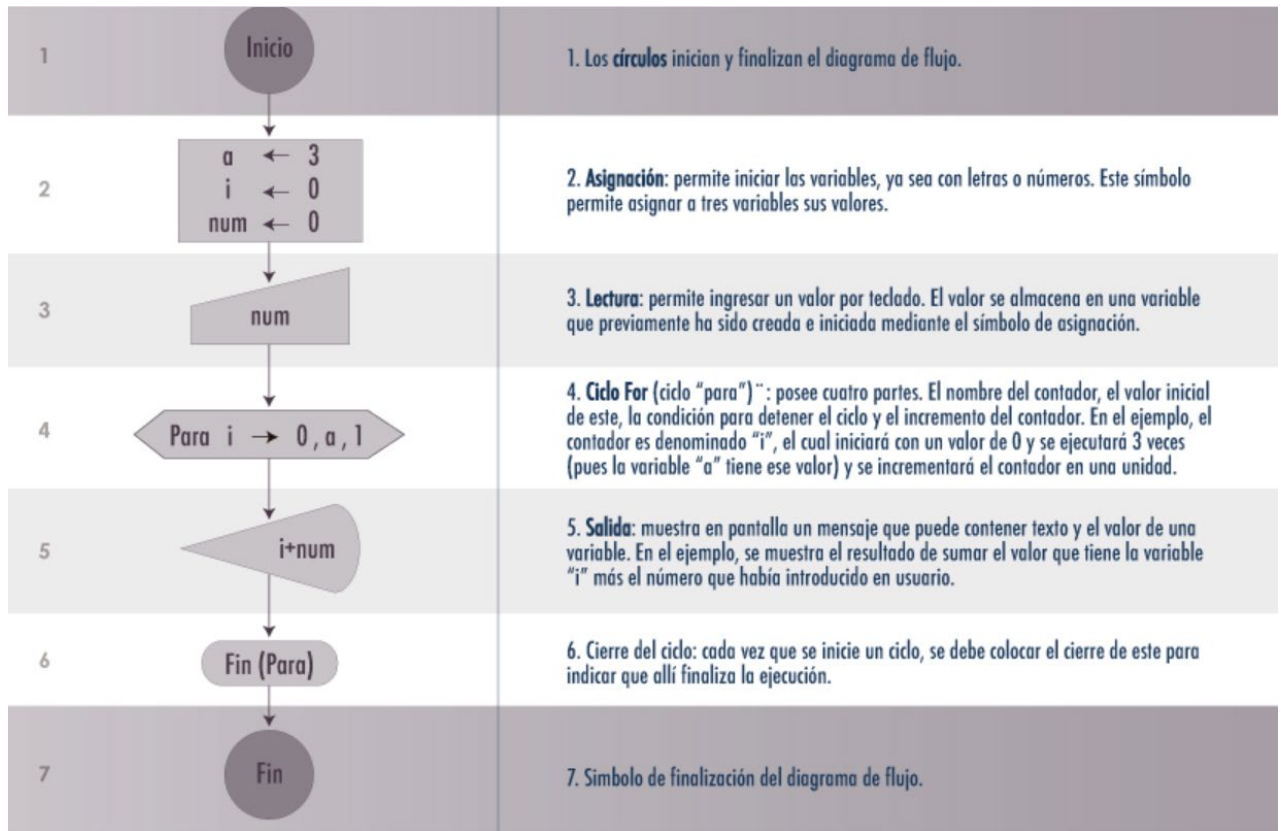
Diagrama de flujo con condicionales

Tabla de simbologías de un diagrama de flujo.

SÍMBOLO	REPRESENTA	SÍMBOLO	REPRESENTA
	Terminal. Indica el inicio o la terminación del flujo, puede ser acción o lugar; además se usa para indicar una unidad administrativa o persona que recibe o proporciona información.		Documento. Representa cualquier tipo de documento que entra, se utilice, se genere o salga del procedimiento.
	Disparador. Indica el inicio de un procedimiento, contiene el nombre de éste o el nombre de la unidad administrativa donde se da inicio		Archivo. Representa un archivo común y corriente de oficina.
	Operación. Representa la realización de una operación o actividad relativas a un procedimiento.		Conector. Representa una conexión o enlace de una parte del diagrama de flujo con otra parte lejana del mismo.
	Decisión o alternativa. Indica un punto dentro del flujo en que son posibles varios caminos alternativos.		Conector de página. Representa una conexión o enlace con otra hoja diferente, en la que continúa el diagrama de flujo.
	Nota aclaratoria. No forma parte del diagrama de flujo, es un elemento que se adiciona a una operación o actividad para dar una explicación.		Línea de comunicación. Proporciona la transmisión de información de un lugar a otro mediante?
SÍMBOLO	REPRESENTA	SÍMBOLO	REPRESENTA
	Operación con teclado. Representa una operación en que se utiliza una perforadora o verificadora de tarjeta.		Dirección de flujo o línea de unión. Conecta los símbolos señalando el orden en que se deben realizar las distintas operaciones.
	Tarjeta perforadora. Representa cualquier tipo de tarjeta perforada que se utilice en el procedimiento.		Cinta magnética. Representa cualquier tipo de cinta magnética que se utilice en el procedimiento.
	Cinta perforada. Representa cualquier tipo de cinta perforada que se utilice en el procedimiento.		Teclado en línea. Representa el uso de un dispositivo en línea para proporcionar información a una computadora electrónica u obtenerla de ello.

NOTA: Los símbolos marcados con * son utilizados en combinación con el resto cuando se está elaborando un diagrama de flujo de un procedimiento en el cual interviene algún equipo de procesamiento electrónico.

Diagrama de flujo con ciclos



Modularización de código

¿Qué es la modularización?

La modularización es el proceso por el cual seleccionamos y agrupamos instrucciones de programación que cumplen una función específica.

Ventajas de la modularización de código.

La modularización permite subdividir una aplicación en partes más pequeñas (llamadas módulos)

Necesitas tener tu código modularizado, te ayudará a armar estructuras en tu Código y tendrás mayor optimización de tu aplicación.

Deja que cada bloque haga una tarea particular.

La modularización permitirá que nuestro código sea escalable.

Reutiliza y dinamiza, Nos permite reutilizar valores que podemos utilizar en diferentes contextos.

En funciones y archivos (también podemos modularizar archivos)

Resumen



- 1.- Necesitas tener tu código modularizado.
- 2.- Deja que cada bloque haga una tarea en particular.
- 3.- Esto permitirá que sea escalable.
- 4.- Seguramente estará optimizado.
- 5.- Reutiliza y dinamiza.
- 6.- A nivel de funciones y archivos (también podemos modularizar archivos).

Nomenclaturas de programación: camelCase, PascalCase, snake_case

Las nomenclaturas son formas de llamar a elementos, sentencias o acciones más específicas. En programación tenemos diferentes formas que a veces son aplicadas como “reglas” sobre las cuales llevamos uniformidad en el código y un estándar de trabajo especialmente dentro de equipos de desarrollo de software.

camelCase o CamelCase

Es una práctica de escritura que consiste en la unión de dos o más palabras sin espacios entre ellas, pero las diferencian una letra mayúscula inicial a partir de la segunda palabra, por ejemplo: miNombreEs.

Este nombre está dado porque forman con las letras mayúsculas iniciales la estructura de un camello que sube cuando hay un inicio de palabra y baja durante su definición.

PascalCase

Es similar a camelCase con una variación bastante simple: desde la primera letra de la frase se inicia con mayúscula cada palabra, sin embargo se mantiene la práctica de no tener espacios entre palabras y cada palabra siempre empieza con su primera letra mayúscula. Por ejemplo: MiNombreEs.

snake_case

La nomenclatura “serpiente” es definida de esta forma porque siempre la usamos sobre el piso, esto quiere decir que las letras siempre las minúsculas y las diferentes palabras que compongan el nombre o la definición que se desea dar están separadas por un guion bajo de esta forma: mi_nombre_es.

SINTAXYS

Es la diferencia entre la forma de escribir en cada lenguaje de programación.

La forma de escribir Código.

Por ejemplo: Escribimos Hola Mundo en varios lenguajes



JavaScript

```
document.write('Hola Mundo!');
```



Python

```
print "Hola Mundo!"
```

Java

```
public class Programa {  
    public static void main(String[] args) {  
        System.out.println("Hola mundo!");  
    }  
}
```