ÍNDICE

1. Introducción a la lógica de programación:

Aprender a programar es como empezar a leer una receta de comida; tienes que leer las instrucciones para que puedas llegar al objetivo. La lógica detrás de la programación consiste en detectar, analizar y ejecutar con éxito las instrucciones de un programa. Además, no solamente tienes que seguir la receta al pie de la letra, porque no siempre tendrás los utensilios o condimentos necesarios, y para eso es necesario improvisar. Pues con la programación es igual.

Conceptos básicos de la programación

Es necesario adentrarse al vocabulario de este mundo para poder entender un poco de su semántica.

Código es una secuencia de instrucciones que un programador escribe para decirle a un dispositivo (como una computadora) qué hacer. Básicamente vendría siendo un conjunto de palabras con el que podríamos ordenarle comunicarnos con la computadora.

Un **desarrollador** (también llamado **programador**) es aquella persona que analiza un problema e implementa una solución usando un conjunto de código. [1]

Un **programa** sería un conjunto de instrucciones o una serie de módulos o procedimientos que permiten realizar determinados tipos de operaciones informáticas. [2] Si el código son instrucciones, por lo tanto un programa sería un conjunto de código con un objetivo en específico.

Los **lenguajes de programación** son intermediarios entre los humanos y la máquina, facilitando la comunicación de instrucciones escritas por los desarrolladores y, a su vez, interpretables o compilables por las computadoras. Existen varios lenguajes hoy en día, pero la mayoría de ellos utilizan una sintaxis en común, por lo que una vez que seas experto en uno solo, podrás dominar los demás con menos dificultad. Estos se pueden dividir por niveles de abstracción, por paradigmas de la programación, etc; sin embargo, por niveles de abstracción es lo más común:

- Lenguajes de Bajo Nivel:
    - Lenguajes de Máquina: Consisten en instrucciones binarias (ceros y unos) directamente ejecutables por un procesador.
    - Lenguajes Ensambladores (Assembly): Es un pequeño paso por encima del lenguaje de máquina. Utiliza mnemónicos y símbolos para representar las instrucciones y los datos binarios del lenguaje de máquina, haciéndolo ligeramente más legible para los humanos.

- Lenguajes de alto nivel:
    - Más alejados del hardware y más cercanos al lenguaje humano. Incluyen Python, Java, C++, etc., y suelen ser más fáciles de aprender y usar. Estos tienen diferentes objetivos, como automatizar tareas de la computadora, servir como calculadora, e incluso crear páginas web.

En algún punto todos hemos sido programadores, ya sea directamente o indirectamente; para poder cocinar un platillo debiste de haber seguido una receta escrita o por memoria; para poder vestirte debiste de seguir una secuencia de pasos para estar vestido para alguna ocasión, sin importar el orden pues el resultado es el mismo; para poder hacer que una planta crezca deberás conocer entender qué tipo de planta es para darle su respectivo cuidado. Todos estos ejemplos están relacionados con la forma en la que se usa la programación.

Importancia de la lógica en el desarrollo de software

La programación trata de resolver problemas. La lógica ayuda a descomponer un problema en partes más pequeñas y comprensibles. Los desarrolladores utilizan el razonamiento lógico para analizar y abordar problemas de manera sistemática. Del mismo modo, ninguna persona nace con este tipo de lógica, sino la desarrollas conforme la experiencia a través del tiempo, es por eso que es importante tener perseverancia y paciencia hacia tu objetivo de convertirte en un buen programador.

Contenido adicional

- Qué son los lenguajes de programación y para qué sirven | Computación y programación
    - https://www.youtube.com/watch?v=pWw4UtQhdek&ab_channel=GCFAprendeLibre
- Para Qué se Usa Cada Lenguaje de Programación?
    - https://www.youtube.com/watch?v=pqfAvZC5clw&ab_channel=deivchoi

2. Lógica Booleana

Fundamentos de la lógica booleana

Gracias al matemático George Boole, tenemos un tipo de álgebra denominada como su apellido: “Álgebra booleana”, con el objetivo de facilitar la manipulación de valores de verdad, como lo son los verdaderos (1) o falsos (0). Esto se ha convertido en una parte fundamental en el diseño de sistemas digitales y programación de computadoras.

Por ejemplo, tenemos la afirmación: “El cielo es azul”, donde el valor de verdad llega a ser verdadero cuando el cielo realmente es azul, o falso cuando es de cualquier otro color.

Tablas de verdad

Es importante destacar que existen operaciones lógicas dentro de la álgebra booleana, cuya finalidad es manipular y analizar expresiones booleanas. Imagina que estás planificando un viaje por carretera. Antes de tomar cualquier ruta, necesitas evaluar si el tráfico está congestionado, si hay obras en la carretera o si hay alguna alerta meteorológica. Si todas estas condiciones son favorables (verdaderas), decides tomar la carretera. De lo contrario, buscas una ruta alternativa. Esta evaluación y toma de decisiones se asemeja a la lógica booleana, donde las condiciones son expresadas como valores booleanos y se combinan para tomar decisiones.

Una tabla de verdad es una herramienta fundamental en la lógica booleana que se utiliza para representar y analizar el comportamiento de una expresión booleana. Consiste en una tabla que enumera todas las posibles combinaciones de valores de verdad para las variables involucradas en la expresión y muestra el resultado de la expresión para cada combinación.

La siguiente es una tabla de verdad que simplifica el entendimiento del funcionamiento de la operación booleana básica “AND (&&)”:

A	**B**	**A && B**
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 1.0. Tabla de verdad: AND

Las condiciones, eventos o cualquier otro procedimiento que pueda ser evaluado como verdadero o falso, son representados con las letras A,B,C,D…,etc.; nota: estas pueden solamente tomar el valor verdadero (1) o falso (0). Básicamente lo que se demuestra en la Tabla 1.0, es la conjunción de dos elementos, A y B; literalmente se lee “A y B”. El resultado de la conjunción SIEMPRE tenderá a falso (0), siempre y cuando los dos elementos no sean verdaderos (1). Imagina una puerta de seguridad que requiere dos claves para abrirla. La primera clave representa la proposición "Tienes la tarjeta de acceso" y la segunda clave representa la proposición "Conoces el código de seguridad". La puerta se abrirá únicamente si tienes ambas claves; es decir, si ambas proposiciones son verdaderas.

La siguiente es una tabla de verdad que simplifica el entendimiento del funcionamiento de la operación booleana básica “OR (||)”:

A	**B**	**A || B**
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 1.1. Tabla de verdad: OR

La operación OR toma dos valores booleanos y devuelve verdadero (1) si al menos uno de ellos es verdadero, y falso (0) si ambos son falsos. En la tabla se muestra la disyunción de A con respecto con B (no importa el orden); se lee literalmente “A o B". Supón que estás planeando asistir a un concierto y estás buscando opciones de compra de entradas. Puedes decidir comprar entradas para la sección A (si A es verdadero), o para la sección B (si B es verdadero), o para la sección VIP (si C es verdadero). Cualquiera de estas opciones te permitirá disfrutar del concierto.

La siguiente es una tabla de verdad que simplifica el entendimiento del funcionamiento de la operación booleana básica “NOT (!)”:

A	**!A**
0	1
1	0

Tabla 1.2 Tabla de verdad: NOT

La operación NOT toma un valor booleano y devuelve el opuesto. Es decir, devuelve verdadero (1) si el valor de entrada es falso, y falso (0) si el valor de entrada es verdadero. Dentro de la tabla se demuestra como se “niega” un elemento al utilizar dicha operación; se lee literalmente “A negada”.

Ahora, estos ejemplos solo cuentan con dos operandos, pero ¿qué pasaría si tuviéramos múltiples condiciones combinados con múltiples operandos?
Un ejemplo es el siguiente: Supongamos que tenemos los siguientes operandos:

A = Verdadero (1)
B = Falso (0)
C = Verdadero (1)
D = Falso (0)

Y tenemos la siguiente operación: **A** && **B** && **C** && **D**

Lo usual que te pedirán hacer en un examen de la universidad es crear la tabla de verdad, pero dentro de la programación, es poco común el uso de dicha herramienta mientras realizas tu código. Es por eso que para poder ahorrar tiempo y entender a la primera deberás apegarte a las definiciones de cada una de las operaciones booleanas.

En este ejemplo, nos indica la utilización de solamente la operación “AND”, y esta explica que el resultado de dos elementos deberán ser verdadero para que el resultado sea verdadero. El truco es agarrar de dos en dos elementos de izquierda a derecha, de modo que vas aplicando una operación a la vez, como si fuera una simple operación matemática.

Primero convertimos cada variable en su valor verdadero (1 o 0) dependiendo el valor asignado, o convertirlos en el estado de verdad que le asignaron (verdadero o falso). En este caso, utilizaremos los unos y ceros, la operación quedaría de la siguiente manera:

1 && 0 && 1 && 0

Después, aplicaríamos la operación respectiva agarrando primero los dos primeros elementos de izquierda a derecha, cuyo resultado es falso (0), puesto que uno de ellos (B) es negativo. Quedando como resultado lo siguiente:

0 && 1 && 0

Luego, agarramos los dos primeros nuevamente, así sucesivamente hasta tener un solo valor de verdad. En esta operación, el resultado debe ser falso (cero).

Como programador, debes siempre tomar en cuenta todas las excepciones en todo caso de estudio o problema. ¿Qué pasaría si en la operación cuenta agrupaciones, como los paréntesis?

Un ejemplo es el siguiente: Supongamos que tenemos los siguientes operandos:

A = Falso (0)
B = Verdadero (1)
C = Falso (0)
D = Verdadero (1)

Y tenemos la siguiente operación: !(**B** && (!**A** && **C** || !**D**))

Al encontrarnos con agrupaciones, tenemos siempre que resolverlo siempre de adentro hacia afuera, hasta eliminar todas los paréntesis, y así realizar la forma de solucionarlo ya antes mencionada.

Entonces, se debe de solucionar primero la operación que utiliza los operandos **A**, **C** y **D**. **A** se convierte en verdadero (1) porque fue negado con la operación NOT, al juntarlo con **C** mediante la operación AND su resultado será falso (0), para después juntarlo con **D** negado (falso) mediante la operación OR, dando como resultado falso. Una vez tengamos el producto de la agrupación más profunda, el siguiente paso es juntarla con las operaciones externas. En este caso, sería juntar **B** (verdadero) con el resultado (falso), cuyo producto por la operación AND es falso (0), pero al ser sometido por una operación NOT, se convierte en verdadero (1).

Sección de ejercicios

- ¿Cuál es el valor de verdad resultante de las siguientes operaciones?.

1.
```
(true && false) || (true || false)
```

2.
```
!(true && false) || (false && true)
```

3.
```
!true && (false || true)
```

4.
```
!(false || false) || (true && false)
```

5.
```
!(false || true) || (true && true)
```

6.
```
(false || true) && (!(true && true) || (false && true))
```

7.
```
((true && true) || !(false || true)) && ((true || true) && false)
```

8.
```
((true || false) && !(true && false)) || (!(false && true) && true)
```

9.
```
(!(true || false) && (true && true)) || ((false || false) && !true)
```

10.
```
((true && false) || (true || false)) && !(true && true)
```

Contenido adicional

- Reglas de Boole. Ejercicios.
    - https://www.youtube.com/watch?v=bZI2gdOS3RQ&ab_channel=Electr%C3%B3nicaFP
- Mapas de Karnaugh para Simplificar Circuitos
    - https://www.youtube.com/watch?v=8WgEmX0ExaY&ab_channel=ElprofeGarc%C3%ADa

3. Tipos de datos más utilizados en la programación

Las palabras como “int”, “float”, etc, son ejemplos de palabras reservadas de los lenguajes de programación que le hacen saber a la computadora qué tipo de dato el programador va a utilizar en su código. Generalmente se le adjudica un tipo de datos a las variables dentro de un código; las variables son contenedores de información que pueden tener un valor que puede cambiar durante la ejecución del programa. Una variable y una constante es que el valor de una variable es mutable (puede cambiar en la ejecución del programa), mientras que una constante es inmutable (que no cambia en la ejecución del programa). Cabe recalcar que las palabras reservadas varían de acuerdo al lenguaje de programación, incluso en python no se necesitan, basta con solo nombrar el nombre de la variable y asignarle cualquier dato, como se podrían mostrar en los ejercicios puestos en este estudio.

Los siguientes tipos de datos son los más utilizados en la programación con sus nombres más comunes:

Entero (int)

Como su nombre lo explica, esta palabra reservada sirve para declarar variables que contengan números enteros. Por ejemplo: -1, 5, 1000. Pero existe una limitación, estos datos no pueden superar los 32 bits, es decir, no rebasar los 2 millones positivos ni negativos.

Ejemplo:

```
int edad = 25
```

Flotante (float,double)

Es un tipo de dato numérico que se utiliza para representar números con parte fraccionaria. A diferencia de los datos enteros, que representan números sin parte fraccionaria, los datos de punto flotante pueden representar números reales que incluyen tanto la parte entera como la parte decimal. La dependencia del uso de los números

decimales es por el lenguaje de programación, pero normalmente es de 15-16 decimales significativos, después de esos los redondea o pierde precisión .

Ejemplo:

```
float altura = 18.5
float num1 = 12345678901234567890.12345678901234567890
float num2 = 0.00000000000000000000123456789
```

Caracteres (char)

El tipo de dato de caracteres (o "carácter" en singular) se utilizan para representar símbolos individuales, como letras del alfabeto (mayúsculas y minúsculas), dígitos numéricos (0-9), signos de puntuación, símbolos especiales y caracteres de control (como el espacio en blanco o el retorno de carro).

Ejemplo:

```
char letra = 'A'
```

Menudo, los caracteres se utilizan como componentes individuales en cadenas de texto más largas. Las cadenas de texto son secuencias de caracteres concatenados y se utilizan ampliamente para representar palabras, frases, párrafos, nombres de archivos y más en programas de computadora. Realmente, el límite de la definición de las variables depende específicamente de la memoria RAM de la computadora. Para poder declararlas es por medio de la palabra reservada “string”.

Ejemplo:

```
string hello = "My name is Marco Antonio!"
```

Booleano (bool)

Este tipo de dato puede tener solo dos posibles valores: verdadero (true) o falso (false). Es utilizado para realizar evaluaciones condicionales y controlar el flujo de un programa basado en condiciones lógicas. Los valores booleanos se utilizan para controlar el flujo de un programa mediante estructuras de control condicional, como instrucciones if, else-if y while.

Ejemplos:

```
bool valor_estado1 = True
bool valor_estado2 = False
```

Sección de ejercicios:

- Identificar el tipo de dato que corresponde al valor mostrado, ya sea entero, flotante,

11. `"59.96"`
12. `-50.1`
13. `599999`
14. `0.15-ae8`
15. `'¿'`
16. `"0.16-0"`
17. `false`
18. `true`
19. `falsetrue`

Contenido adicional

- Programación en C++ || Tipos de datos básicos en C++
  - https://www.youtube.com/watch?v=xBOpQN8jR54&ab_channel=Programaci%C3%B3nATS
- TIPOS DE DATOS | Introducción a los ALGORITMOS y la PROGRAMACIÓN
  - https://www.youtube.com/watch?v=_INtSsEcnwc&ab_channel=TodoCode

4. Estructuras de control

Estas declaraciones son esenciales para la programación y deben ser utilizadas dependiendo el tipo de condición que quieres agregar a tu código.

Cada programa de software está compuesto por un conjunto de instrucciones que indican a la computadora qué operaciones debe realizar y en qué orden. Estas instrucciones están escritas en lenguajes de programación comprensibles por humanos, y luego se traducen a un lenguaje que la computadora puede ejecutar.

Estas estructuras de control son las más utilizadas en el campo de la programación para solucionar problemas comunes:

Condicionales

- if

Este se utiliza para ejecutar una serie de instrucciones en dado caso que la condición sea VERDADERA. En caso contrario, el bloque de código no se ejecutará.

La sintaxis es la siguiente:

```
if condicion:
    # Código a ejecutar si la condición es verdadera
```

- elif (else if)

Este se utiliza cuando se quiere evaluar o realizar múltiples condiciones en secuencia y ejecutar el bloque de código correspondiente a la primera condición verdadera encontrada. Si la condición1 es verdadera, se sale automáticamente de la estructura, sino, lee la condición2 para saber si es verdadera, así sucesivamente hasta encontrar una condición verdadera.

La sintaxis es la siguiente:

```
if condicion1:
    # Código a ejecutar si la condición1 es verdadera
elif condicion2:
    # Código a ejecutar si la condición2 es verdadera
# Puedes tener más elif si es necesario
```

- else

Se ejecuta si ninguna de las condiciones siguientes ANTERIORES es verdadera. Dicho esto, es necesario que para que funcione debe tener alguna de las dos condiciones, ya sea “if” o “elif” , antes de poner “else”. Es decir, todas las condiciones anteriores a “else” tuvieron que ser falsas para ejecutarlo.

La sintaxis es la siguiente:

```
if condicion:
    # Código a ejecutar si la condición es verdadera
else:
    # Código a ejecutar si la condición no es verdadera
```

La siguiente sintaxis puede ser un ejemplo muy común:

```
if condicion1:
    # Código a ejecutar si la condición1 es verdadera
elif condicion2:
    # Código a ejecutar si la condición2 es verdadera
else:
    # Código a ejecutar si la condición no es verdadera
```

De esa manera ya puede combinar todos los condicionales vistos, SIEMPRE tendrán esa estructura, es decir, una condicional “elif” no puede estar sin antes haber puesto un “if”; una condicional “else” no puede estar sin antes haber puesto o un “if” o un “elif”. Cabe destacar que, como toda lista de instrucciones, las líneas de código se deben escribir siempre de arriba hacia abajo para que la computadora pueda interpretarlo.

Sección de ejercicios:

- ¿Qué imprimirá (print()) los siguientes ejercicios de estructura de control de acuerdo a los valores de las variables? Nota: La función “print()” se utiliza en programación para mostrar mensajes en la pantalla para imprimir valores; todo valor entre los paréntesis dentro de la función es todo lo que proyectará en la pantalla, ya sean variables previamente definidas o cadenas de caracteres (dicha función puede cambiar de nombre según el lenguaje de programación, asegúrate de encontrar bien su sintaxis).

20.

```
nota = 75
if nota >= 60:
    print("El estudiante aprobó el examen.")
else:
    print("El estudiante no aprobó el examen.")
```

21.

```
tipo_usuario = "admin1"
if tipo_usuario == "admin":
    print("El usuario tiene acceso de administrador.")
else:
    print("El usuario no tiene acceso de administrador.")
```

22.

```
stock = 10
if stock > 0:
    print("El artículo está en stock.")
else:
    print("El artículo no está en stock.")
```

23.

```
edad = 20
if edad >= 18:
    print("El usuario es mayor de edad.")
else:
    print("El usuario es menor de edad.")
```

24.

```
ingresos = 45000
if ingresos < 30000:
    print("Los ingresos de la persona son bajos.")
elif ingresos < 60000:
    print("Los ingresos de la persona son medios.")
else:
    print("Los ingresos de la persona son altos.")
```

25.

```
indice_contaminacion = 90
if indice_contaminacion < 50:
    print("La calidad del aire es buena.")
elif indice_contaminacion < 70:
    print("La calidad del aire es aceptable.")
else:
    print("La calidad del aire es mala.")
```

Contenido adicional

- Estructuras de programación (qué es secuencia, condicional, ciclo) | programación
  - https://www.youtube.com/watch?v=rNY5eWogI18&ab_channel=GCFAprendeLibre
- Estructuras CONDICIONALES ✏️ | SI SINO | Introducción a los ALGORITMOS y la PROGRAMACIÓN
  - https://www.youtube.com/watch?v=5m9xSRVfEYM&ab_channel=TodoCode

5. Operadores aritméticos

Los operadores aritméticos en la programación sirven para realizar operaciones matemáticas en los programas. Estas operaciones pueden ser tan simples como sumar dos números o tan complejas como calcular una fórmula algebraica.

- Suma (+):

  Se lee: El resultado es igual a la suma de *a* más *b*.
  Ejemplo:

```
a = 5
b = 3
resultado = a + b
# resultado será 8
```

- Resta (-):

  Se lee: El resultado es igual a la resta de *a* menos *b*.
  Ejemplo:

```
a = 10
b = 7
resultado = a - b
# resultado será 3
```

- Multiplicación (*):

  Se lee: El resultado es la multiplicación de *a* por *b*.
  Ejemplo:

```
a = 4
b = 6
resultado = a * b
# resultado será 24
```

- Potencia (**):

  Se lee: *a* a la potencia del exponente *b*. El operando de potencia (**), también conocido como operador de exponenciación, se utiliza en programación para calcular la potencia de un número. Su función es elevar un número a una potencia específica.
  Ejemplo:

```
resultado = 2 ** 3
print(resultado)  # Esto imprimirá 8
```

- División decimal (/):

Se lee: El resultado es la división decimal de *a* entre *b*. Quiere decir que mostrará el cociente junto con sus decimales pertinentes.

Ejemplo:

```
a = 15
b = 3
resultado = a / b
# resultado será 5.0 (división decimal)
```

Ejemplo:

```
a = 16
b = 3
resultado = a / b
# resultado será 5.333333333333333 (división decimal)
```

- División entera (//)

Se lee: El resultado es la división entera de *a* entre *b*. Quiere decir que mostrará únicamente el cociente de la división.

Ejemplo:

```
a = 15
b = 3
resultado = a // b
# resultado será 5 (división entera)
```

- Módulo (%):

Se lee: El resultado es igual a *a* módulo *b*. Quiere decir que estamos calculando el residuo de la división del primer número entre el segundo número, de izquierda a derecha.

Ejemplo:

```
a = 17
b = 5
resultado = a % b
# resultado será 2 (17 dividido por 5 da 3, con residuo 2)
```

Sección de ejercicios:

Como en los ejercicios anteriores, las operaciones aritméticas se leen de izquierda a derecha, tomando en cuenta primero aquellas operaciones que están dentro de paréntesis.

- ¿Qué valor imprimirá la variable final de la ecuación aritmética?

26. Valor final de la variable *total_segundos*

```
horas = 3
minutos = 15
segundos = 30
total_segundos = (horas * 3600) + (minutos * 60) + segundos
print("El total de segundos es:", total_segundos)
```

27. Valor final de la variable *fahrenheit*

```
celsius = 85
fahrenheit = (celsius * 9/5) + 32
print("85 grados Celsius equivalen a", fahrenheit, "grados
Fahrenheit.")
```

28. Valor final de la variable *promedio*

```
calificacion1 = 85
calificacion2 = 90
calificacion3 = 88
calificacion4 = 92
promedio = (calificacion1 + calificacion2 + calificacion3 +
calificacion4) / 4
print("El promedio de las calificaciones es:", promedio)
```

29. Valor final de la variable *area*

```
base = 10
altura = 8
area = (base * altura) / 2
print("El área del triángulo es:", area)
```

30. Valor final de la variable *resto*

```
resto = 17 % 5
print("El resto de la división es:", resto)
```

31. Impresión de la condición

```
numero = 14
if numero % 2 == 0:
    print("El número es par.")
else:
    print("El número es impar.")
```

32. Valor final de la variable *volumen*

```
arista = 4
volumen = arista ** 3
print("El volumen del cubo es:", volumen)
```

Contenido adicional

- OPERADORES (Operadores aritméticos) - LENGUAJE C
  - https://www.youtube.com/watch?v=6cprJILuARw&ab_channel=Pasosporingenier%C3%ADa
- Programación en Python | Operadores Aritméticos
  - https://www.youtube.com/watch?v=PMOWXusLr9g&ab_channel=Programaci%C3%B3nATS

6. Operadores de comparación:

A diferencia de los operadores anteriores, estos SIEMPRE regresarán algún valor de tipo BOOLEANO, es decir, verdadero o falso.

- Igual (==):

Se lee: ¿a es igual a b? Este operador se utiliza para comprobar si dos valores son iguales. Si los valores son iguales, devuelve true; de lo contrario, devuelve false.

Ejemplo:

```
numero1 = 10
numero2 = 10

if numero1 == numero2:
    print("Los números son iguales.") #<--imprimirá esto
else:
    print("Los números son diferentes.")
```

- Mayor que (>):

Se lee: ¿a es mayor que b? Este operador verifica si el primer valor es mayor que el segundo. Devuelve true si el primer valor es mayor que el segundo; de lo contrario, devuelve false.

Ejemplo:

```
numero1 = 10
numero2 = 5

if numero1 > numero2:
    print("numero1 es mayor que numero2.")
else:
    print("numero1 no es mayor que numero2.") #<--imprimirá esto
```

- Mayor igual que (>=):

Se lee: ¿a es mayor igual que b? Este operador verifica si el primer valor es mayor o igual que el segundo. Devuelve true si el primer valor es mayor o igual que el segundo; de lo contrario, devuelve false.

Ejemplo:

```
numero1 = 10
numero2 = 10

if numero1 >= numero2:
    print("numero1 es mayor o igual que numero2.") #<--imprimirá esto
else:
    print("numero1 no es mayor o igual que numero2.")
```

- Menor que (<):

Se lee: ¿a es menor que b? Este operador verifica si el primer valor es menor que el segundo. Devuelve true si el primer valor es menor que el segundo; de lo contrario, devuelve false.

Ejemplo:

```
numero1 = 5
numero2 = 10

if numero1 < numero2:
    print("numero1 es menor que numero2.") #<--imprimirá esto
else:
    print("numero1 no es menor que numero2.")
```

- Menor igual que (<=):

Se lee: ¿a es menor igual que b? Este operador verifica si el primer valor es menor o igual que el segundo. Devuelve true si el primer valor es menor o igual que el segundo; de lo contrario, devuelve false.

Ejemplo:

```
numero1 = 5
numero2 = 10

if numero1 <= numero2:
    print("numero1 es menor o igual que numero2.") #<--imprimirá esto
else:
    print("numero1 no es menor o igual que numero2.")
```

- Distinto que (!=)

Se lee: ¿a es distinto que b? Este operador verifica si dos valores son diferentes. Si los valores son diferentes, devuelve true; de lo contrario, devuelve false. Al igual que el operador de igualdad, este operador no distingue entre tipos de datos.

Ejemplo:

```
a = 5
b = 10

if a != b:
    print("a y b son diferentes.") #<--imprimirá esto
else:
    print("a y b son iguales.")
```

Sección de ejercicios:

Estos operadores de comparación son esenciales para la toma de decisiones en la programación y se utilizan frecuentemente en estructuras de control como condicionales. Permiten a los programadores comparar valores y tomar acciones basadas en esas comparaciones. Es por eso que los siguientes ejercicios son una combinación de lo aprendido anteriormente.

-¿Qué imprimirá (print()) los siguientes ejercicios de acuerdo a los valores de las variables?

33.

```
numero = -5

if numero >= 0:
    print("El número es positivo.")
else:
    print("El número es negativo.")
```

34.

```
base = 5
altura = 5
area = base * altura

if base == altura:
    print("El área es", area, "y es un cuadrado.")
else:
    print("El área es", area, "y no es un cuadrado.")
```

35.

```
anio = 2024

if (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0):
    print(anio, "es un año bisiesto.")
else:
    print(anio, "no es un año bisiesto.")
```

36.

```
precio_producto = 50
cantidad = 3
descuento = 10

precio_total = (precio_producto * cantidad) - (precio_producto *
cantidad * descuento / 100)
print("El precio total con descuento es:", precio_total)
```

37.

```
numero = 15

if numero % 2 == 0:
    print(numero, "es un número par.")
else:
    print(numero, "es un número impar.")

if numero % 3 == 0:
    print(numero, "es divisible por 3.")
else:
    print(numero, "no es divisible por 3.")
```

Contenido adicional

- OPERADORES LÓGICOS y de COMPARACIÓN | Iniciándose en la Programación
  - https://www.youtube.com/watch?v=rUhdqG_1vEg&ab_channel=TecnoBinaria
- Operadores de comparación | Curso Python
  - https://www.youtube.com/watch?v=9HoKXY8uKVA&ab_channel=IEEEITBA

7. Bucles y repeticiones

También conocidos como estructuras de repetición o ciclos, son elementos fundamentales en la programación que permiten ejecutar un bloque de código múltiples veces, basado en una condición o un conjunto de valores. Son una herramienta poderosa para automatizar tareas repetitivas. Pueden iterar sobre una colección de datos (flotantes, enteros, caracteres, string, etc), realizar cálculos repetidos o simplemente ejecutar un bloque de código un número específico de veces.

• Bucle While

Se utiliza para repetir un bloque de código mientras se cumpla una condición específica. La condición se evalúa antes de cada iteración y, si es verdadera, el bloque de código se ejecuta. Si la condición es falsa, el bucle se detiene y la ejecución continúa con la siguiente instrucción después del bucle.
Estructura:

```
while condicion:
    # Bloque de código a ejecutar mientras la condición sea verdadera
    # Se ejecutará repetidamente hasta que la condición sea falsa
```

Ejemplo:

Suponiendo que la tarea es imprimir los números del 1 al 5 utilizando un bucle, se haría de esta manera:

```
numero = 1



while numero <= 5:
    print(numero)
    numero = numero + 1
```

Resultado:

```
1
2
3
4
5
```

- ## Bucle For

Se utiliza para repetir un bloque de código un número específico de veces.se utiliza cuando se sabe cuántas veces se desea iterar o cuando se quiere iterar sobre una secuencia, mientras que el bucle while se utiliza cuando se quiere repetir un bloque de código mientras se cumpla una condición específica.

Estructura:

```
for elemento in secuencia:
    # Bloque de código a ejecutar para cada elemento de la secuencia
    # Se ejecutará una vez para cada elemento de la secuencia
```

Nota: El *elemento* es una variable que toma el valor de cada elemento de la *secuencia* en cada iteración del bucle.=, mientras que la secuencia es la secuencia sobre la cual se va a iterar, puede ser una lista, tupla, conjunto, rango y otro tipo de estructura de datos iterable. Dicho bloque de código se ejecutará UNA vez por cada elemento de la secuencia; si tenemos 5 elementos dentro de una lista, se repetirá 5 veces, una por cada elemento.

Ejemplo: (Utilizando un rango, del 1 al 10, incluyendo el 1 pero excluyendo el 11)

```
suma = 0

for numero in range(1, 11):
    suma = suma + numero #el guardar el resultado de la suma en la
misma variable suma, es lo que hace que se acumulen los valores

print("La suma de los números del 1 al 10 es:", suma) #<-- suma = 55
```

Ejemplo: (Utilizando arreglos -estructura de datos más popular para almacenar datos de cualquier tipo-)

```
arreglo = [10, 20, 30, 40, 50]

print("Elementos del arreglo:")
for elemento in arreglo:
    print(elemento)
```

Resultado:

```
10
20
30
40
50
```

La ventaja de estos ciclos o bucles, así como en toda la programación, es que puedes utilizar cualquier en la mayoría de los casos (tomando en cuenta que las necesidades y los datos que tenemos); es decir, si te encuentras en la posición de utilizar cualquier de los dos, puedes hacerlo sin ningún problema.

Sección de ejercicios

Existen bucles anidados que ayudan a iterar entre colecciones de datos más avanzados, como lo son matrices, listas anidadas, árboles binarios, etc. Los siguientes ejercicios te ayudarán a desarrollar esencialmente los elementos fundamentales que existen en la programación y la resolución de problemas.

- Realiza el código correspondiente para realizar los problemas a continuación, toma en cuenta todo lo visto anteriormente. Puedes utilizar cualquier bucle de tu preferencia, de acuerdo a tus necesidades y objetivo.

38. Calcular e imprimir (utilizando print()) los números pares que hay entre los números 1 y 10, tomando en cuenta los mismos números.

39. Calcular e imprimir la suma de los primeros 100 números naturales.

40. Calcular e imprimir la tabla de multiplicar del 7 hasta el décimo múltiplo.

Contenido adicional

- Árbol binario en Python - Estructura de datos en programación
    - https://www.youtube.com/watch?v=d0ibZK_6Q7g&ab_channel=LuisCabreraBenito
- Curso Python 3 desde cero | Listas anidadas
    - https://www.youtube.com/watch?v=8tXEia5ZcCE&ab_channel=LaGeekipediaDeErnesto

8. Programación estructurada

Un paradigma es un conjunto de reglas que guían la forma en que se diseña, desarrolla y estructura el software. Utiliza la siguiente analogía para entender un poco más: el caminar se puede hacer llamar como un paradigma de transporte, puesto que existen varias maneras de llegar a un destino, y varios atajos en el camino, pero de una u otra forma llegas al destino. La programación estructurada es un paradigma de programación que se basa en el principio de dividir un programa en partes más pequeñas y manejables, utilizando estructuras de control como secuencias, selecciones (condicionales) y bucles para controlar el flujo de ejecución del programa. Este enfoque se centra en escribir código claro, legible y fácil de entender, lo que facilita la comprensión, el mantenimiento y la depuración del software.

Solo tienes que entender que mientras estás escribiendo código, tienes que tomar en cuenta ciertas reglas para hacer el texto más entendible y “limpio”. La razón es por que al estar trabajando en equipo en proyectos, el código que vayas escribiendo será leído, observado y/o retroalimentado, y si no logran entender la más mínima idea de lo que has creado, no podrán ayudarte en solucionar algún problema o mencionar en qué podrías mejorar.

9. Herramientas que te ayudarían

Si son como yo, leer puede ser una de las maneras más tardadas y laboriosas de aprender, esto dependerá del poder de la imaginación y concentración de la persona. Es por eso que el contenido visual es una gran herramienta que puede ayudarte a avanzar un poco más rápido sin desviar tu atención de manera frecuente, como lo sería leer o interpretar texto pesado.

Los videos adjuntos son contenido que me ayudaron a entender temas difíciles que me enfrenté cuando empecé a estudiar programación. Cabe mencionar que, como cualquier otra carrera, nunca terminarás de estudiar, y en el mundo de la programación no es la excepción. Los consejos que doy personalmente cuando me preguntan qué necesito para “sobrevivir” o “querer” alguna carrera relacionada a la tecnología son ser persistente, creativo y paciente; es decir, terco, inventivo y aguantador. Para mí esas son las cualidades más importantes que un programador debe de tener.

10. Sección de respuestas

1. True. 2. True. 3. False. 4. True. 5. True. 6. False. 7. False. 8. True. 9. False. 10. False. 11.String. 12. Flotante (float). 13. Entero (int). 14. No existe tipo de dato. 15. Carácter. 16.String. 17. Booleano. 18. Booleano. 19. No existe tipo de dato. 20. “El estudiante aprobó el examen.” 21. “El usuario no tiene acceso de administrador.”. 22. “El artículo está en stock.”. 23. “El usuario es mayor de edad.”. 24. “Los ingresos de la persona son medios.”. 25. “La calidad del aire es mala.”. 26. “El total de segundos es: 11,730”. 27. “85 grados Celsius equivalen a 185.0 grados Fahrenheit.”. 28. “El promedio de las calificaciones es: 88.75”. 29. “El área del triángulo es: 40”. 30. “El resto de la división es: 2”. 31. “El número es par." .32. “El volumen del cubo es: 64”. 33. “El número es negativo.”. 34. “El área es 25 y es un cuadrado.”. 35. “2024 es un año bisiesto.”. 36. “El precio total con descuento es: 135.0”. 37. “15 es un número impar.”, “15 es divisible por 3.”.

38.
Ejercicio:

```
contador = 0
numero = 0

while contador < 10:
    if numero % 2 == 0:
        print(numero)
        contador += 1
    numero += 1
```

Resultado:

```
0
2
4
6
8
10
12
14
16
18
```

39.
Ejercicio:

```
suma = 0

for numero in range(1, 101):
    suma += numero

print("La suma de los primeros 100 números naturales es:", suma)
```

Resultado:
La suma de los primeros 100 números naturales es: 5050

40.
Ejercicio:

```
numero = 7

for i in range(1, 11):
    print(numero, "x", i, "=", numero * i)
```

Resultado:

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

11. Bibliografía

[1] Estefania Cassingena Navone. (2023). freeCodeCamp. ¿Qué es programación? Manual para principiantes. https://www.freecodecamp.org/espanol/news/que-es-programacion-manual-para-principiantes/#:~:text=Es%20el%20proceso%20de%20escribir,las%20aplicaciones%20que%20te%20encantan.

[2] Euroinnova. (2018). EuroInnova International Online Education. que es un programa de software. https://www.euroinnova.mx/blog/que-es-un-programa-de-software