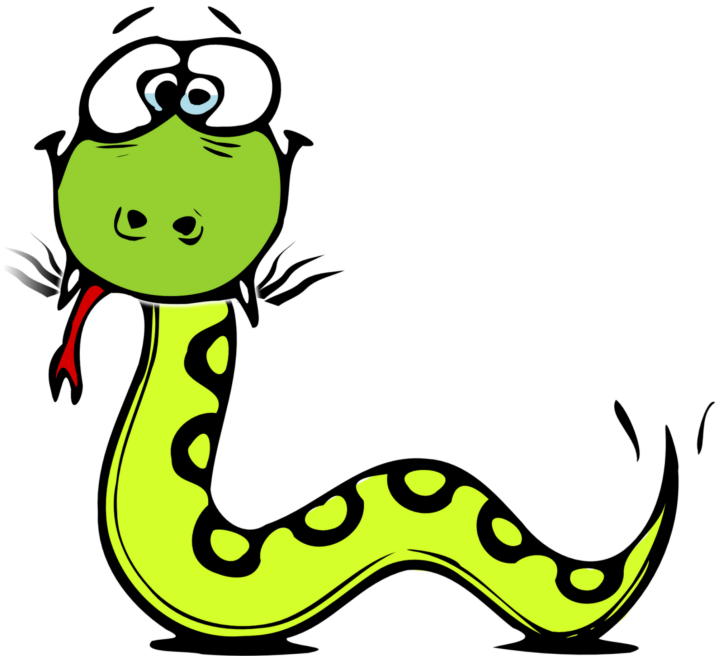
R.VAN DER GRAAFF
G.DISCABBIO E.FROST



PYTHON

para estudiantes de Inteligencia Artificial

PYTHON

Para Estudiantes de Inteligencia Artificial

Robert Van Der Graaff

Giuseppe DiScabbio

Edward Frost

Derechos de autor

Índice general

Introducción a Python

1. Introducción

Un programa en Python es una serie de comandos escritos de acuerdo a ciertas reglas que la computadora lee y ejecuta. Estos comandos deben seguir ciertas reglas conocidas como **sintaxis**.
La información que maneja un programa en Python se conoce como **datos**. Los datos de un programa en Python pueden ser de diferentes tipos, según la información que contengan.

2. Objetivos

- **Objetivo General:**
  - Introducir a los estudiantes al lenguaje de programación Python, su sintaxis básica y los tipos de datos fundamentales.
- **Objetivos Específicos:**
  - Comprender la estructura básica de un programa en Python.
  - Declarar y utilizar variables.
  - Conocer y trabajar con los tipos de datos básicos en Python: enteros, flotantes, cadenas de texto y booleanos.

3. ¿Qué es PYTHON?

Python es un lenguaje de programación de alto nivel, interpretado,

interactivo y de propósito general. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python es conocido por su sintaxis sencilla y legible, lo que lo convierte en un lenguaje ideal tanto para principiantes como para programadores experimentados.

Una de las características más destacadas de Python es su enfoque en la legibilidad del código. Esto facilita el aprendizaje del lenguaje, ya que permite a los desarrolladores expresar conceptos en pocas líneas de código. Además,

Python soporta múltiples paradigmas de programación, incluidos el imperativo, el orientado a objetos y el funcional.

Python cuenta con una amplia colección de bibliotecas estándar que cubren diversas áreas, como el procesamiento de datos, el análisis numérico, la inteligencia artificial, la automatización, el desarrollo web y mucho más. Gracias a su comunidad activa, Python se ha convertido en uno de los lenguajes más populares y versátiles en la actualidad.

3.Características de Python

- **Sintaxis sencilla**: Python es fácil de leer y escribir, lo que permite que los programadores se centren en resolver problemas en lugar de preocuparse por detalles complejos del lenguaje.
- **Multiparadigma**: Soporta programación orientada a objetos, funcional y procedimental.
- **Extensa librería estándar**: Python tiene una gran cantidad de bibliotecas que permiten realizar tareas complejas con poca cantidad de código.
- **Portabilidad**: Python es compatible con múltiples sistemas operativos, como Windows, Linux y macOS.
- **Comunidad activa**: La gran comunidad de desarrolladores y contribuyentes asegura una continua evolución y soporte del lenguaje.

Gracias a estas características, Python es ampliamente utilizado en ciencia de datos, inteligencia artificial, desarrollo web, automatización y muchas otras áreas.

4.Sintaxis básica:

- Importancia de la indentación en Python.
- Uso de comentarios: `# comentario`.

4.1.Variables y asignación:

Explicación de cómo declarar variables y asignarles valores. Tipos de variables: enteros, flotantes, cadenas y booleanos.

4.2.Tipos de datos:

- **Enteros:** `int`, ejemplo: `x = 5`.
- **Flotantes:** `float`, ejemplo: `y = 3.14`.

- **Cadenas:** `str`, ejemplo: `name = "Juan"`.
- **Booleanos:** `bool`, ejemplo: `is_active = True`.

5. Ejemplos

Ejemplo 1: Declaración y uso de variables numéricas:

```
x = 10  # Asignacion de un entero
y = 3.14  # Asignacion de un flotante
print(x + y)  # Imprime la suma de un entero y un
    flotante
```

Explicación: En este ejemplo, asignamos un valor entero a `x` y un valor flotante a `y`. Luego, mostramos cómo sumar estos valores.

Ejemplo 2: Trabajo con cadenas:

```
name = "Maria"
greeting = "Hola, " + name
print(greeting)
```

Explicación: Aquí concatenamos una cadena con otra, mostrando cómo trabajar con texto.

Ejemplo 3: Uso de variables booleanas:

```
is_active = True
if is_active:
    print("El programa está activo.")
```

Explicación: Utilizamos una variable booleana en una estructura condicional `if` para mostrar cómo se trabaja con valores lógicos.

Ejemplo 4: Operaciones aritméticas:

```
a = 8
b = 2
result = a / b
print("El resultado de la división es:", result)
```

Explicación: En este caso, mostramos cómo realizar una operación aritmética simple con variables numéricas.

Ejemplo 5: Entrada de datos con `input()`:

```
age = input(" Cu ántos años tienes? ")
print("Tienes " + age + " años.")
```

Explicación: Usamos `input()` para obtener datos del usuario y luego los mostramos.

5. Ejercicios para casa con resolución completa:

Ejercicio 1:

Enunciado: Crea una variable `nombre` que contenga tu nombre y una variable `edad` con tu edad. Luego, imprime una frase que diga "Hola, [nombre]. Tienes [edad] años."

Resolución:

```
nombre = "Juan"
edad = 25
print("Hola, " + nombre + ". Tienes " + str(edad) + " años.")
```

Ejercicio 2:

Enunciado: Crea dos variables `a` y `b` de tipo flotante. Realiza la suma, resta, multiplicación y división entre ellas, y muestra los resultados por pantalla.

Resolución:

```
a = 3.5
b = 1.5
print("Suma:", a + b)
print("Resta:", a - b)
print("Multiplicación:", a * b)
print("División:", a / b)
```

Ejercicio 3:

Enunciado: Escribe un programa que pida al usuario su edad y calcule cuántos años tiene dentro de 5 años.

Resolución:

```
edad = int(input(" Cu ántos años tienes? "))
print("En 5 años tendrás", edad + 5, "años.")
```

Ejercicio 4:

Enunciado: Crea una variable booleana `is_sunny` que indique si está soleado o no. Usa una estructura condicional `if` para imprimir .Es un día soleado.o "No está soleado".

Resolución:

```
is_sunny = True
if is_sunny:
    print("Es un día soleado.")
else:
    print("No está soleado.")
```

Ejercicio 5:

Enunciado: Crea un programa que pida al usuario su nombre y lo imprima en mayúsculas.

Resolución:

```
nombre = input(" Cu ál es tu nombre? ")
print("Tu nombre en mayúsculas es:", nombre.upper())
```

6. Conclusión:

En esta clase, hemos aprendido los conceptos básicos de Python, desde la sintaxis básica hasta cómo trabajar con variables y tipos de datos fundamentales como enteros, flotantes, cadenas y booleanos. Estos son los bloques de construcción esenciales para cualquier programa en Python. Con estos conocimientos, los estudiantes estarán listos para abordar estructuras de control y operaciones más complejas en las próximas clases.

Estructuras de control I: CONDICIONALES

Introducción:

El **control de flujo en Python** se refiere a la forma en que se ejecutan las instrucciones de un programa, determinando el orden y la lógica con la que se procesan las líneas de código.

- **Objetivo General:** Aprender cómo utilizar estructuras de control de flujo en Python, específicamente las condicionales, para tomar decisiones dentro de un programa.
- **Objetivos Específicos**
  - Entender y usar la estructura condicional básica `if`.
  - Implementar condicionales con `else` y `elif`.
  - Crear programas que tomen decisiones basadas en condiciones lógicas.

Control de flujo en Python:

Las estructuras de control en Python permiten definir el flujo de ejecución de un programa, organizando la manera en que se toman decisiones, se repiten instrucciones o se manejan errores. Se dividen en tres tipos principales:

- **Estructuras Condicionales (Decisiones)**

  Estas estructuras permiten que un programa tome decisiones basadas en una condición lógica. Son útiles cuando se quiere ejecutar diferentes acciones dependiendo de si una condición es verdadera o falsa.

  Características:

  - Evalúan una condición que puede ser verdadera o falsa.
  - Ejecutan un bloque de código si la condición es verdadera y otro si es falsa.

- Permiten múltiples condiciones encadenadas para escenarios más complejos.

Ejemplo práctico: Se utilizan cuando hay que verificar si un usuario tiene acceso a un sistema, si un número es positivo o negativo, o para aplicar descuentos según el monto de una compra.

- **Estructuras Iterativas (Bucles o Repeticiones)**

  Las estructuras iterativas permiten repetir un bloque de código varias veces hasta que se cumpla una condición o hasta recorrer una serie de elementos.

  Características:

  - Se ejecutan un número determinado o indeterminado de veces según la condición.
  - Pueden recorrer elementos de una lista, diccionario o conjunto de datos.
  - Es importante controlar la condición de salida para evitar repeticiones infinitas.

    Ejemplo práctico: Se usan para procesar listas de elementos, repetir cálculos hasta obtener un resultado específico o automatizar tareas repetitivas como el envío de correos en masa.

- **Manejo de Excepciones (Errores)** Esta estructura permite manejar errores inesperados que pueden ocurrir durante la ejecución de un programa, evitando que se detenga abruptamente.
  Características:

  - Permiten detectar errores sin interrumpir el programa.
  - Definen un plan alternativo en caso de que ocurra un error.
  - Mejoran la estabilidad y la experiencia del usuario.

  Ejemplo práctico: Se aplican cuando se espera que un usuario ingrese datos en un formato específico, al leer archivos que podrían no existir o al manejar errores de conexión en una aplicación web.

Condicionales en Python

Las estructuras condicionales permiten que un programa ejecute diferentes bloques de código en función de si una condición es verdadera o falsa.

Estructura básica `if`:

La sintaxis básica de una estructura `if` es:

```
if condición:
    # Bloque de código a ejecutar si la condición es
       verdadera
```

Ejemplo:

```
x = 10
if x > 5:
    print("x es mayor que 5")
```

Estructura `if-else`:

La estructura `if-else` permite ejecutar un bloque de código cuando la condición es verdadera y otro bloque cuando es falsa:

```
if condición:
    # Bloque de código si la condición es verdadera
else:
    # Bloque de código si la condición es falsa
```

Ejemplo:

```
x = 3
if x > 5:
    print("x es mayor que 5")
else:
    print("x es menor o igual a 5")
```

Estructura `if-elif-else`:

Si tienes múltiples condiciones, puedes usar `elif` para evaluar condiciones adicionales:

```
if condición1:
    # Bloque de código si condición1 es verdadera
elif condición2:
    # Bloque de código si condición2 es verdadera
else:
    # Bloque de código si ninguna condición es verdadera
```

Ejemplo:

```
x = 8
if x > 10:
    print("x es mayor que 10")
elif x == 8:
    print("x es igual a 8")
else:
    print("x es menor que 8")
```

Operadores de comparación:

Los operadores de comparación permiten realizar evaluaciones lógicas:

- `==`: igual a
- `!=`: diferente a
- `>: mayor que`
- `<`: menor que
- `>=`: mayor o igual que
- `<=`: menor o igual que

Ejemplos (5 ejemplos explicados):

Ejemplo 1: Uso de `if` para verificar si un número es positivo:

```
x = 10
if x > 0:
    print("El número es positivo.")
```

Explicación: Este programa verifica si un número es mayor que cero y, si es cierto, imprime un mensaje indicando que es positivo.

Ejemplo 2: Uso de `if-else` para comprobar si un número es mayor o menor que 10:

```
x = 7
if x > 10:
    print("El número es mayor que 10.")
else:
    print("El número es menor o igual a 10.")
```

Explicación: Aquí comparamos si el valor de `x` es mayor que 10; de ser así, se imprime un mensaje; si no, se imprime el mensaje del bloque `else`.

Ejemplo 3: Uso de `if-elif-else` para determinar si un número es negativo, cero o positivo:

```
x = -5
if x > 0:
    print("El número es positivo.")
elif x == 0:
    print("El número es cero.")
else:
    print("El número es negativo.")
```

Explicación: En este ejemplo usamos múltiples condiciones con `elif` para diferenciar entre números positivos, cero y negativos.

Ejemplo 4: Comprobación de igualdad con ==:

```
nombre = "Juan"
if nombre == "Juan":
    print(" Hola , Juan!")
else:
    print("No eres Juan.")
```

Explicación: Comparamos la variable `nombre` con el valor `"Juan"` utilizando el operador de igualdad `==`.

Ejemplo 5: Comprobación de desigualdad con !=:

```
edad = 18
if edad != 21:
    print("No tienes 21 años.")
else:
    print(" Feliz  21 cumpleaños!")
```

Explicación: Aquí utilizamos el operador de desigualdad `!=` para verificar si la edad no es 21 y mostrar un mensaje en consecuencia.

Ejercicios para casa con resolución completa:

Ejercicio 1:

Enunciado: Crea un programa que pida un número al usuario y diga si es par o impar.

Resolución:

```
numero = int(input("Introduce un número: "))
if numero % 2 == 0:
    print("El número es par.")
else:
    print("El número es impar.")
```

Ejercicio 2:

Enunciado: Crea un programa que pida la edad del usuario y determine si es mayor de edad (18 años o más).

Resolución:

```
edad = int(input(" Cu ántos años tienes? "))
if edad >= 18:
    print("Eres mayor de edad.")
```

```
4 else:
5     print("Eres menor de edad.")
```

Ejercicio 3:

Enunciado: Escribe un programa que pida dos números y diga cuál es el mayor.

Resolución:

```
1 numero1 = int(input("Introduce el primer número: "))
2 numero2 = int(input("Introduce el segundo número: "))
3 if numero1 > numero2:
4     print("El primer número es mayor.")
5 elif numero2 > numero1:
6     print("El segundo número es mayor.")
7 else:
8     print("Ambos números son iguales.")
```

Ejercicio 4:

Enunciado: Crea un programa que pida un número y determine si es divisible por 3 y 5.

Resolución:

```
1 numero = int(input("Introduce un número: "))
2 if numero % 3 == 0 and numero % 5 == 0:
3     print("El número es divisible por 3 y 5.")
4 else:
5     print("El número no es divisible por 3 y 5.")
```

Ejercicio 5:

Enunciado: Crea un programa que pida una palabra y diga si es "python".

Resolución:

```
1 palabra = input("Introduce una palabra: ")
2 if palabra.lower() == "python":
3     print(" La  palabra es python!")
4 else:
5     print("La palabra no es python.")
```

Conclusión:

En esta clase, hemos aprendido a utilizar las estructuras condicionales `if`, `if-else` y `if-elif-else` para tomar decisiones dentro de un programa en Python. Estas estructuras son fundamentales para permitir que los programas respondan

a diferentes situaciones basadas en condiciones lógicas, lo que nos permite crear software dinámico y adaptable.

Estructuras de control II: Bucles

3.1. Objetivos

En este capítulo, aprenderemos sobre los bucles en Python, específicamente `for` y `while`. Comprenderemos su sintaxis, funcionamiento y aplicación en la resolución de problemas computacionales.

3.2. Introducción

Los bucles son estructuras de control que permiten ejecutar un bloque de código varias veces. En Python, existen dos tipos principales de bucles:

- **Bucle `for`**: Se utiliza para iterar sobre secuencias como listas, tuplas o rangos de números.
- **Bucle `while`**: Ejecuta un bloque de código mientras se cumpla una condición específica.

3.3. Desarrollo del Tema

3.3.1. Bucle `for`

El bucle `for` permite recorrer elementos de una secuencia sin necesidad de un contador manual.

Sintaxis

```
for variable in secuencia:
    # Código a ejecutar
```

Ejemplo básico

```
for i in range(5):
    print("Iteración", i)
```

Ejemplo con listas

```
frutas = ["manzana", "banana", "cereza"]
for fruta in frutas:
    print("Me gusta", fruta)
```

3.3.2. Bucle `while`

El bucle `while` se usa cuando no se conoce de antemano cuántas veces se ejecutará el código, sino que depende de una condición.

Sintaxis

```
while condición:
    # Código a ejecutar
```

Ejemplo básico

```
contador = 0
while contador < 5:
    print("Contador:", contador)
    contador += 1
```

3.3.3. Control de flujo en los bucles

Se pueden modificar los bucles con `break` y `continue`.

Uso de `break`

```
for num in range(10):
    if num == 5:
        break  # Sale del bucle cuando num es 5
    print(num)
```

Uso de `continue`

```
for num in range(5):
    if num == 2:
        continue  # Salta la iteración cuando num es 2
    print(num)
```

3.4. Ejercicios Resueltos

1. **Números pares:** Imprimir los números pares del 1 al 10 usando un bucle `for`.

```
for num in range(1, 11):
    if num % 2 == 0:
        print(num)
```

2. **Suma de números naturales:** Calcular la suma de los primeros 10 números naturales usando un bucle `while`.

```
suma = 0
num = 1
while num <= 10:
    suma += num
    num += 1
print("Suma:", suma)
```

3. **Factorial de un número:** Calcular el factorial de un número ingresado por el usuario.

```
num = int(input("Ingrese un número: "))
factorial = 1
for i in range(1, num + 1):
    factorial *= i
print("Factorial:", factorial)
```

4. **Imprimir los primeros 10 múltiplos de 3.**

```
for i in range(1, 11):
    print(i * 3)
```

5. **Suma de los primeros 100 números.**

```
suma = sum(range(1, 101))
print("Suma:", suma)
```

6. **Contar vocales en una cadena.**

```
cadena = "Python es increíble"
vocales = "aeiouAEIOU"
contador = sum(1 for c in cadena if c in vocales)
print("Número de vocales:", contador)
```

7. **Mostrar una tabla de multiplicar.**

```python
num = int(input("Ingrese un número: "))
for i in range(1, 11):
    print(num, "x", i, "=", num * i)
```

8. **Adivinar un número con `while`.**

```python
import random
numero = random.randint(1, 10)
intento = 0
while intento != numero:
    intento = int(input("Adivina el número (1-10): ")
        )
print("  Correcto  !")
```

9. **Invertir una cadena con un bucle.**

```python
cadena = "Python"
invertida = ""
for c in cadena:
    invertida = c + invertida
print("Cadena invertida:", invertida)
```

10. **Imprimir la secuencia de Fibonacci.**

```python
a, b = 0, 1
for _ in range(10):
    print(a, end=" ")
    a, b = b, a + b
```

3.5. Ejercicios Propuestos

1. Contar cuántas veces aparece una letra en una cadena de texto.
2. Solicitar números al usuario hasta que ingrese 0 y mostrar la suma total.
3. Imprimir los primeros 20 números impares.
4. Mostrar los primeros 15 términos de la serie de Fibonacci.
5. Escribir un programa que pida una palabra y la invierta.
6. Calcular el promedio de una lista de números ingresados por el usuario.
7. Imprimir una tabla de multiplicar hasta el 12.
8. Determinar si un número ingresado es primo.
9. Generar un patrón de estrellas en pantalla.
10. Contar la cantidad de dígitos en un número entero ingresado por el usuario.

3.6. Conclusión

Los bucles `for` y `while` permiten repetir bloques de código de manera eficiente. Mientras que el bucle `for` es ideal para recorrer secuencias, el `while` se utiliza cuando la cantidad de iteraciones no es fija. Además, las sentencias `break` y `continue` permiten controlar el flujo de ejecución de los bucles.