Resumen Python:
Diego Markiewicz – Gaspar Oddovero

- **Nombre corto, único y misterioso:** Guido van Rossum buscaba un nombre así para su lenguaje de programación.
- **Inspiración:** El programa de comedia "Monty Python's Flying Circus" era su favorito.
- **Homenaje:** Nombró el lenguaje en honor al programa.

```
#Comentarios de una línea comienzan con una almohadilla (o signo gato)

""" Strings multilinea pueden escribirse
    usando tres "'s, y comunmente son usados
    como comentarios.
"""
```

`print` función que muestra información. Ejemplo: print("Hola Mundo")

Variables en Python

Podemos crear una nueva variable de Python asignando un valor a una etiqueta, usando el operador de asignación =.

En este ejemplo asignamos una cadena con el valor "Gaspar" a nombre: `nombre = "Gaspar"`

ejemplo con un número: `edad = 8`

Python detecta automáticamente el tipo a partir del tipo de valor. También puedes crear una variable de un tipo específico utilizando el constructor de clase, pasando un valor literal o un nombre de variable:

```
nombre = str("Flavio")
otroNombre = str(nombre)
```

También puedes convertir de un tipo a otro utilizando el constructor de clases. Python intentará determinar el valor correcto, por ejemplo, extrayendo un número de una cadena:

```
edad = int("20")
print(edad) #20
```

---

tipos de datos en Python:

Tipos básicos:

- **Números:** Enteros (ej: 10), decimales (ej: 3.14), complejos (ej: 1+2j).
- **Cadenas:** Secuencias de texto (ej: "Hola mundo").
- **Booleanos:** True o False (ej: activo = True).

```
# Operadores aritméticos. Python tiene varios operadores aritméticos: +,-
,*,/,%(resto),**(exponenciación) y // (división)
1 + 1 #2
2 - 1 #1
2 * 2 #4
```

```
4 / 2 #2
4 % 3 #1
4 ** 2 #16
4 // 2 #2
# La división por defecto retorna un número 'float' (número de coma flotante)
35 / 5  # 7.0
# Cuando usas un float, los resultados son floats
3 * 2.0 # 6.0
# Refuerza la precedencia con paréntesis
(1 + 3) * 2  #  8
```

```
# Valores 'boolean' (booleanos) son primitivos
True - False
# Niega con 'not'
not True  # = False  not False  # = True

# Igualdad es ==
1 == 1 # = True     2 == 1 # = False
# Desigualdad es !=
1!= 1  # = False     2!= 1  # = True
# Más comparaciones
1 < 10  # = True
1 > 10  # = False
2 <= 2  # = True
2 >= 2  # = True
# ¡Las comparaciones pueden ser concatenadas!
1 < 2 < 3  # = True
2 < 3 < 2  # = False
```

```
# + también se usa para concatenar valores de cadena:
nombre = "Aquiles"
nombre += " era un buen perro"
print(nombre) #Aquiles era un buen perro
# Podemos combinar el operador de asignación con operadores aritméticos:
+=    -=    *=     /=     %=
Ejemplo:
edad = 8
edad += 1 #edad es ahora 9
```

Tipos avanzados:

- **Listas:** Colecciones ordenadas y mutables (ej: ["elemento1", "elemento2"]).
- **Tuplas:** Colecciones ordenadas e inmutables (ej: (1, 2, 3)).
- **Diccionarios:** Colecciones no ordenadas de clave-valor (ej: {"nombre": "Juan", "edad": 30}).
- **Conjuntos:** Colecciones no ordenadas sin duplicados (ej: {1, 2, 3}).

```
# Strings se crean con " o '
"Esto es un string."
'Esto también es un string'
# ¡Strings también pueden ser sumados!
"Hola " + "mundo!" #= "Hola mundo!"
# Un string puede ser tratado como una lista de caracteres
"Esto es un string"[0] #= 'E'
```

- También puedes interpolar cadenas usando variables en el contexto

```
nombre = 'Bob'
comida = 'Lasaña'
print(f'{nombre} quiere comer {comida}')  #= "Bob quiere comer lasaña"

Una cadena tiene un conjunto de métodos integrados, como:
isalpha() para comprobar si una cadena contiene solo caracteres y no está vacía
isalnum() para comprobar si una cadena contiene caracteres o dígitos y no está
vacía
isDigit() para comprobar si una cadena contiene dígitos y no está vacía
lower() para obtener una versión en minúsculas de una cadena
islower() para comprobar si una cadena está en minúsculas
upper() para obtener una versión en mayúsculas de una cadena
isupper() para comprobar si una cadena está en mayúsculas
title() para obtener una versión capitalizada de una cadena
startsswith() para comprobar si la cadena comienza con una subcadena específica
endswith() para comprobar si la cadena termina con una subcadena específica
replace() para reemplazar una parte de una cadena
split() para dividir una cadena en un separador de caracteres específico
strip() para recortar el espacio en blanco de una cadena
find() para encontrar la posición de una subcadena
```

El operador "in" te permite verificar si una cadena contiene una subcadena:

```
nombre = "Aquiles"
print("Aqu" in nombre) #True
```

```
El método .find() se utiliza para encontrar la primera aparición de una subcadena
dentro de una cadena de texto. Es parte del módulo estándar de cadenas de Python.

cadena.find(subcadena, [inicio, fin])

Ejemplo:
cadena = "Hola, mundo!" # Buscar "mundo"
posicion = cadena.find("mundo")
print(f"La subcadena 'mundo' se encuentra en la posición {posicion}") # Salida: 7
```

Dada una cadena, puedes obtener sus caracteres usando corchetes cuadrados para obtener un elemento específico, dado su índice, comenzando desde cero [0]:

```
nombre = "Aquiles"
nombre[0] #'A'
nombre[1] #'q'
nombre[2] #'u'
```

También puedes usar un rango, usando lo que llamamos **rebanar**(slicing):

```
nombre = "Roger"
nombre[0:2] #"Ro"
nombre[:2] #"Ro"
nombre[2:] #"ger"
```

```
# Examina el largo de una lista con 'len' y nos muestra el tamaño de caracteres
Print(len("Diego")) #= 5
```

- **if** controla el flujo del programa según una condición. Ejemplo:

```
edad = 18
if edad >= 18:
        print("Eres mayor de edad")
else:
         print("Eres menor de edad")
```

Ejecuta código **solamente si se cumple una condición**.
¿Cómo funciona? Se escribe `if` seguido de una condición entre paréntesis, luego dos puntos y el código a ejecutar si la condición es `True`.

```
#Está una declaración de un 'if'. ¡La indentación es significativa en Python!
# imprime "una_variable es menor que 10"
if una_variable > 10:
    print("una_variable es completamente mas grande que 10.")
elif una_variable < 10:    # Este condición 'elif' es opcional.
    print("una_variable es mas chica que 10.")
else:           # Esto también es opcional.
    print("una_variable es de hecho 10.")
```

Booleanos en Python

Python proporciona el tipo `bool`, que puede tener dos valores: `True` (Verdadero) y `False` (Falso) (*capitalizado*).

```
hecho = False
```

```
hecho = True
```

Los booleanos son especialmente útiles con estructuras de control condicionales como declaraciones `if`:

```
hecho = True
if hecho:
    # Ejecuta algun codigo
else:
    # Ejecuta algun otro codigo
```

Al evaluar un valor para `True` o `False`, si el valor no es un `bool`, tenemos algunas reglas dependiendo del tipo que estemos verificando:

- los números son siempre `True` excepto el número `0`,
- las cadenas son `False` solo cuando están vacías `''`,
- las listas, tuplas, conjuntos y diccionarios son `False` solo cuando están vacíos.

Puedes verificar si un valor es booleano de esta manera:

```
hecho = True
type(hecho) == bool #True
```

Aceptar la entrada del usuario, usando `input()`:

```
edad = input('¿Cual es tu edad?:')
print('Tu edad es ' + edad)
```

```
##* SABER SI SOLO INGRESO NUMEROS
Ejemplo
entrada = input("Ingrese un número: ")
print (type(entrada)) #ver que tipo de datos es la variable entrada
print (entrada.isdigit())
if  entrada.isdigit()  :
    print(f"el numero ingresado es {entrada}.")
```

```
else:
    print(f"Error: Debe ingresar un número válido. fijate lo que pusiste!{entrada}")
    exit
```

SABER SI SOLO INGRESO TEXTO

Ejemplo :

```
entrada = input("Ingrese texto: ")
if entrada.isalpha() or entrada.isspace():
    print(f"el text ingresado es {entrada}.")
else:
    print("Error: Debe ingresar solo texto (letras o espacios).")
```

Acumulador: Suma valores para obtener un total.

```
Ejemplo:
# Inicializamos el acumulador en 0
total_ventas = 0
# Simulamos la venta de productos
precio_producto = float(input("Ingrese el precio del producto: $"))
cantidad = int(input("Ingrese la cantidad vendida: "))
total_ventas += precio_producto * cantidad  # Acumulamos el total de ventas
print(f"El total de ventas es de ${total_ventas}.")
-Se inicializa en 0.
-Se incrementa con el resultado de la multiplicación de precio y cantidad.
-Muestra el valor final del acumulador (total de ventas).
```

- **Contador:** Cuenta el número de veces que ocurre un evento.

```
Ejemplo:
contador = 0
contador +=1 # va contando 1 a 1
```

Resumen rápido de `import` en Python:

¿Qué es `import`?

- Una palabra clave para **incluir código de un módulo en otro**.
- Útil para **organizar el código** y **reutilizarlo**.

¿Qué es un módulo?

- Un **archivo .py** que contiene código Python.
- Puede incluir **funciones, clases, variables** y código ejecutable.
- Sirve para **agrupar código relacionado** y **facilitar la reutilización**.

El módulo random en Python proporciona funciones para generar números aleatorios. ejemplo:

```
Ejemplo:
import random
numero_aleatorio = random.randint(1, 10)
print(f"Número aleatorio entre 1 y 10: {numero_aleatorio}")
```