

Módulo 1
Clase 2

Instrucciones básicas del Lenguaje Python

Objetivos



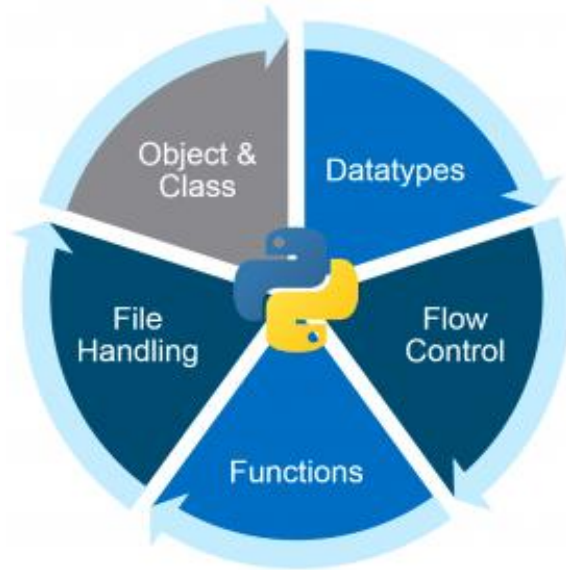
- Aprender Instrucciones básicas de Python.
- Conocer sobre tipos y estructuras de datos, operadores y expresiones.
- Conocer sobre flujos de control.
- Codificar un programa creando funciones.

Contenido

1. Variables y tipos de datos
2. Entrada y salida de datos
3. Operadores y Expresiones
4. Control de Flujos
5. Funciones
6. Módulos



Fundamentos de Python



FUNDAMENTALS

Datatypes	Flow Control	Functions	File Handling	Object & Class
Numbers Strings Lists Dictionaries	If Else For While Continue	Definition Function Call Docstring Return	Reading Writing Editing	Variables Functions

A continuación, los 5 fundamentos para dominar en Python:

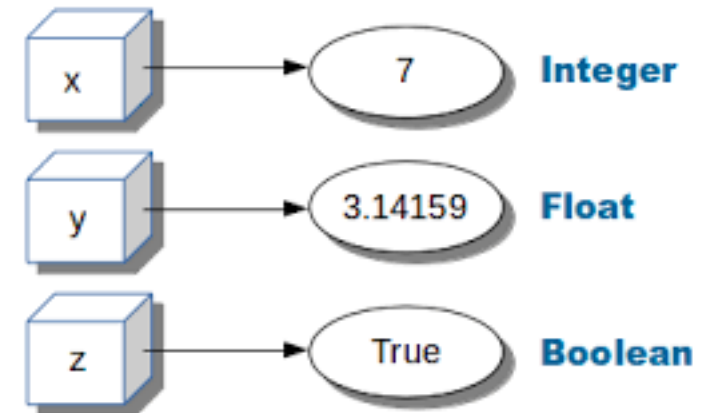
- Tipos de datos
- Control de Flujos
- Funciones
- Manejo de documentos
- Objetos y clases

Variables y Tipos de Datos

¿Qué es una Variable?

Variables

- Espacio de memoria que recibe un identificador y contiene información.
- En Python no se necesita declarar antes de usarla, como en otros lenguajes.



```
In [13]: #  
         name_of_var = 2
```

```
In [14]: x = 2  
         y = 3
```

```
In [15]: z = x + y
```

```
In [16]: z
```

```
Out[16]: 5
```

Variables

- Siempre primero se debe indicar el nombre de la variable y luego, su contenido:

```
num = 17
```

Un mal ejemplo de la declaración de una variable en Python:

```
17 = num
```

Lo siguiente produciría un error Syntax Error.

Reglas para nombrar una variable:

Inicio:

Letras desde A-Z, a-z o un guión bajo(_)

Resto:

Letras desde A-Z, a-z , guión bajo(_) o dígitos(0-9)

Existen palabras reservadas que no se pueden usar como nombre de variable porque Python las usa para otras cosas. Por ejemplo el nombre de una variable no puede ser `*print*`.

Palabras reservadas:

and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield.

Tipos de Datos en Python

```
[ ]: # tipos de variables
```

```
[2]: type(1)
```

ENTERO

NUMERO COMPLEJO

```
[2]: int
```

```
[5]: type(0.1)
```

DECIMAL

```
[5]: float
```

```
[3]: type('texto')
```

CADENA DE
CARACTERES

```
[3]: str
```

(Comilla simple o doble)

```
[4]: type(True)
```

BOOLEANO

```
[4]: bool
```

```
In [22]: # complex numbers: note the use of `j` to specify the imaginary part
x = 1.0 - 1.0j
type(x)
```

Out[22]: complex

```
In [23]: print(x)
```

(1-1j)

```
In [24]: print(x.real, x.imag)
```

(1.0, -1.0)



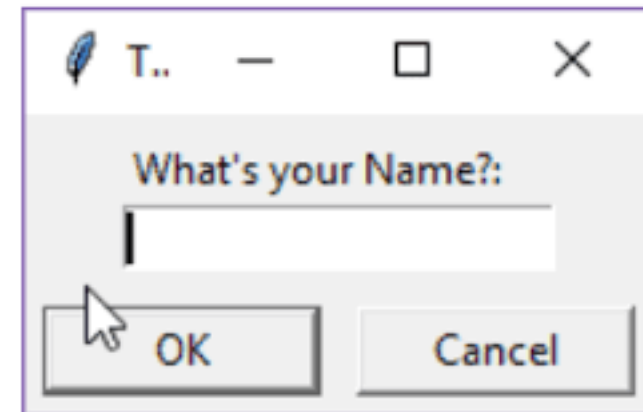
Valores booleanos tienen formato capitalizado (T y F mayúscula)

Entrada y Salida de Datos

Input (Ingreso de Información)

Es la manera de recibir información del entorno externo al programa. Puede ser un usuario, un archivo, otro programa, etc.

Sintaxis: `input("texto a mostrar:")`



```
input("Ingrese su nombre:")
```

Ingrese su nombre:

Input (Ingreso de Información)

- El output de un programa es la manera de transmitir información hacia el entorno. Puede escribir en la consola de la computadora.
- En Jupyter notebook veremos el output debajo de la celda donde fue ejecutado.

Sintaxis: print()

```
print(123)  
print('Hola, Mundo!')
```

```
123  
Hola, Mundo!
```



Imprimir en la Consola

```
In [20]: x = 'hello'
```

```
In [22]: print(x)  
hello
```

Además, podemos formatear la salida con variables:

```
In [23]: num = 12  
         name = 'Sam'
```

```
In [24]: print('My number is: {one}, and my name is: {two}'.format(one=num,two=name))  
My number is: 12, and my name is: Sam
```

```
In [25]: print('My number is: {}, and my name is: {}'.format(num,name))  
My number is: 12, and my name is: Sam
```

Este es el formateo F-string:

```
In [3]: print(f'Hola soy {nombre} y tengo {edad} años')  
Hola soy Miguel y tengo 35 años
```


Operadores y Expresiones

Operadores Matemáticos

```
In [6]: 1 + 1
```

```
Out[6]: 2
```

```
In [7]: 1 * 3
```

```
Out[7]: 3
```

```
In [8]: 1 / 2
```

```
Out[8]: 0.5
```

```
In [9]: 2 ** 4
```

```
Out[9]: 16
```

```
In [10]: 4 % 2
```

```
Out[10]: 0
```

```
In [11]: 5 % 2
```

```
Out[11]: 1
```

```
In [12]: (2 + 3) * (5 + 5)
```

```
Out[12]: 50
```

Los **operadores** son símbolos matemáticos que llevan a cabo una operación específica, entre los operandos tienen una función en específico y pueden recibir operandos variables.

Los operandos serían aquellos argumentos que reciben los operadores para realizar su función.

Operadores de Asignación

OPERADOR	FUNCIÓN	EJEMPLOS	RESULTADO
"="	Asigna un valor a un elemento. Puede ser variable, lista, diccionario, tupla, etc.	a = 2	"a" vale 4
"+="	El primer elemento es igual a la suma del primer elemento con el segundo. Se suele utilizar como contador.	b += 1	b = b + 1 Cada vez que se ejecute esta instrucción se le sumará 1 a "b"
"-="	El primer elemento es igual a la resta del primer elemento con el segundo. Se suele utilizar como contador decreciente.	b -= 1	b = b - 1 Cada vez que se ejecute esta instrucción se le restará 1 a "b"
"*="	El primer elemento es igual a la multiplicación del primer elemento con el segundo.	b *= 2	b = b * 2 Cada vez que se ejecute esta instrucción se multiplicará por dos 2 a "b"
"/="	El primer elemento es igual a la división del primer elemento con el segundo.	b /= 2	b = b / 2 Cada vez que se ejecute esta instrucción se dividirá por 2 a "b"
"%="	El primer elemento es igual al Módulo: resto de la división del primer elemento con el segundo.	b %= 2	b = b % 2 Cada vez que se ejecute esta instrucción "b" se dividirá por 2 y se le asignará el valor del resultado (resto)
"**="	El primer elemento es igual al resultado del exponente del primer elemento con el segundo.	b **= 2	b = b ** 2 Cada vez que se ejecute esta instrucción "b" se expondrá por dos y se le asignará el valor del resultado (exponencial)

Operadores de Comparación

Se utilizan para comparar valores y nos devolverá *True/False* como resultado de la condición.

```
1 > 2
```

False

```
1 < 2
```

True

```
1 >= 2
```

False

```
1 <= 2
```

True

```
a = 1  
b = 2
```

```
a == b
```

False

```
a != b
```

False

Operadores de Comparación

Son and (y) or (o) not (no) y sirven para comprobar si dos o más operandos son ciertos (True) o falsos (false) y nos devolverá como resultado True o False.

```
In [38]: True and False
```

```
Out[38]: False
```

```
In [39]: not False
```

```
Out[39]: True
```

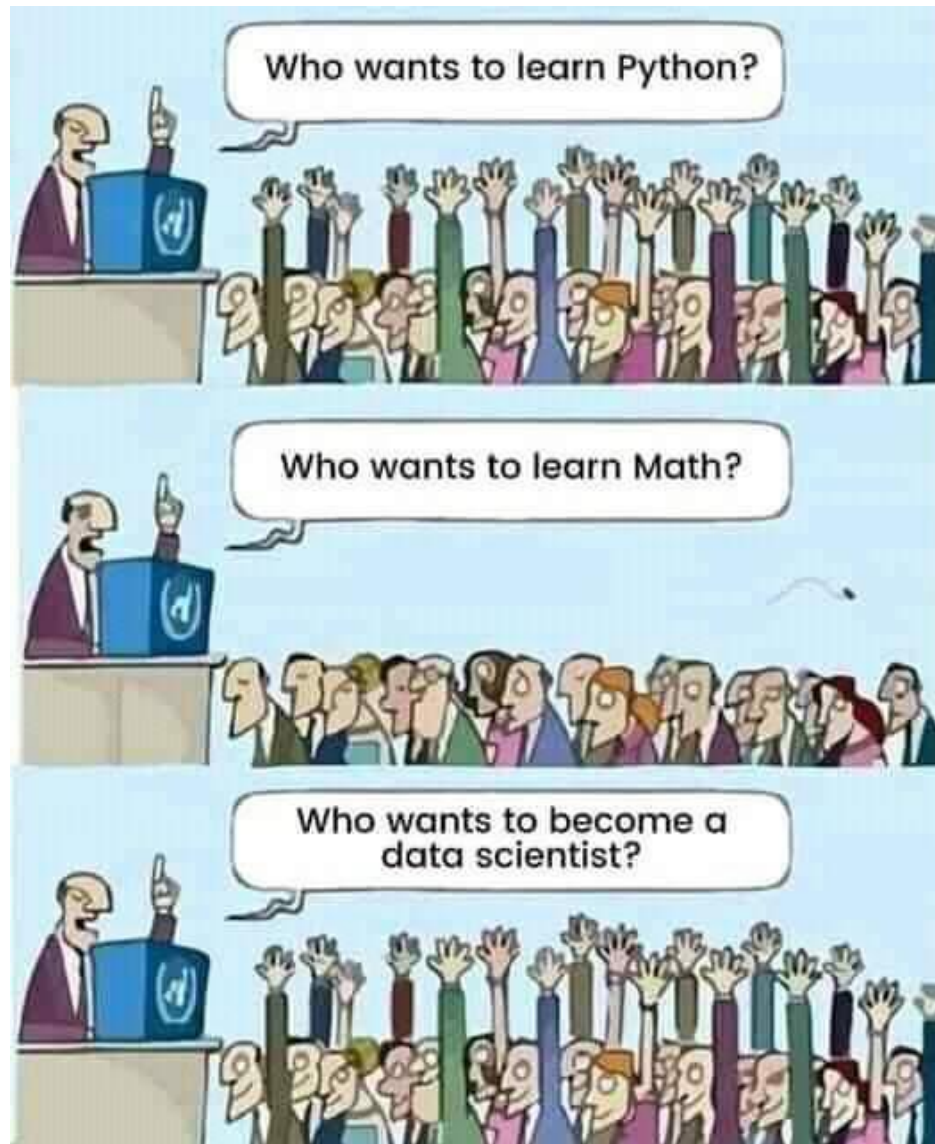
```
In [40]: True or False
```

```
Out[40]: True
```

Not sirve para indicar el contrario de un booleano como True/False

Operadores Especiales

OPERADOR	FUNCIÓN	EJEMPLOS	RESULTADO
"in"	El operador "in" (en) devuelve True si un elemento se encuentra dentro de otro.	a = [3,4] 3 in a	True Porque "3" se encuentra en "a".
"not in"	El operador "not in" (no en) devuelve True si un elemento NO se encuentra dentro de otro.	a = [3,4] 5 in a	True Porque "5" NO se encuentra en "a".
"is"	El operador "is" (es) devuelve True si los elementos son exactamente iguales.	x = 10 y = 10 x is y	True Porque ambas variables tienen el mismo valor, son iguales.
"Not is"	El operador "not is" (no es) devuelve True si los elementos NO son exactamente iguales.	x = 10 y = 11 x not is y	True Porque ambas variables NO tienen el mismo valor, son diferentes.

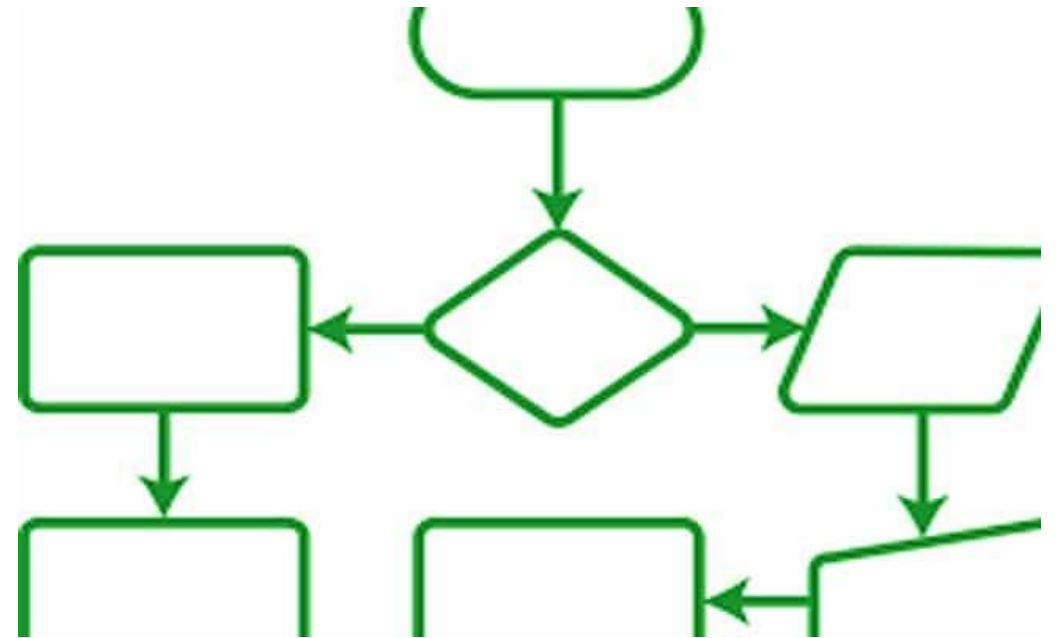


Control de Flujos

Control del Flujo



El control del flujo es la manera que poseen los lenguajes de programación de provocar que el flujo de la ejecución del programa avance y se ramifique en función de los cambios de estado de los datos.



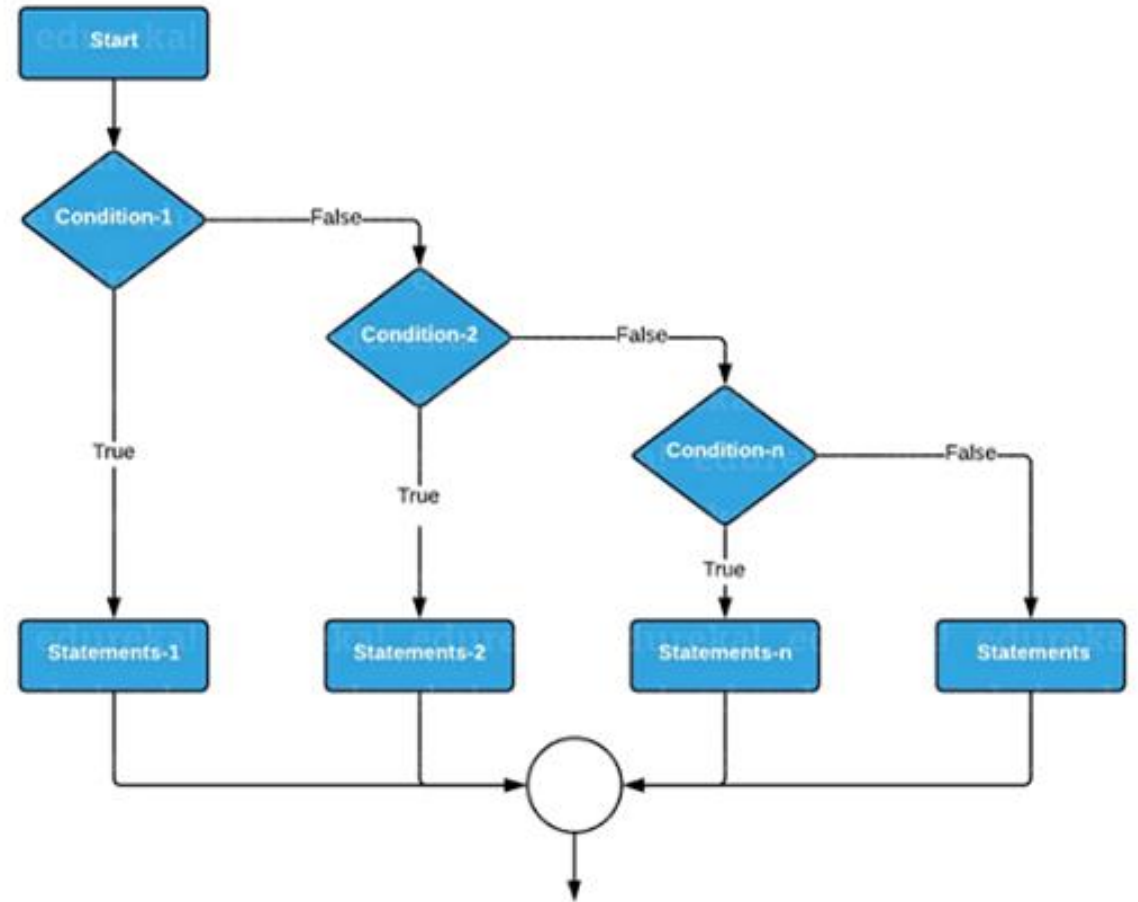
Control del Flujo

- En Python, al igual que en TODOS los lenguajes de programación de alto nivel, tenemos distintos tipos de sentencias o construcciones para controlar el flujo de la ejecución del programa.
 - **Condicionales:** permiten decidir por un camino, entre dos o más posibles, dada una condición.
- Python: if e if-else
 - **Cíclicos o iterativos:** permiten ejecutar un bloque de instrucciones un número dado o un número de veces hasta que una condición se cumpla.
- Python: for y while

Control de Flujo



¿Por qué? Para agregar lógica y repetir ciertas declaraciones para obtener una solución con menos código y de manera inteligente.



Sentencias Condicionales

Las sentencias condicionales son: **if, elif, else**

```
edad = 35
```

```
if edad >= 18:  
    print('Eres Mayor de Edad')  
print('Gracias por Preguntar')
```

Eres Mayor de Edad
Gracias por Preguntar

```
if edad >= 18:  
    print('Eres Mayor de Edad')  
else:  
    print('Eres Menor de Edad')  
  
print('Gracias por Preguntar')
```

Eres Mayor de Edad
Gracias por Preguntar

```
edad = 35
```

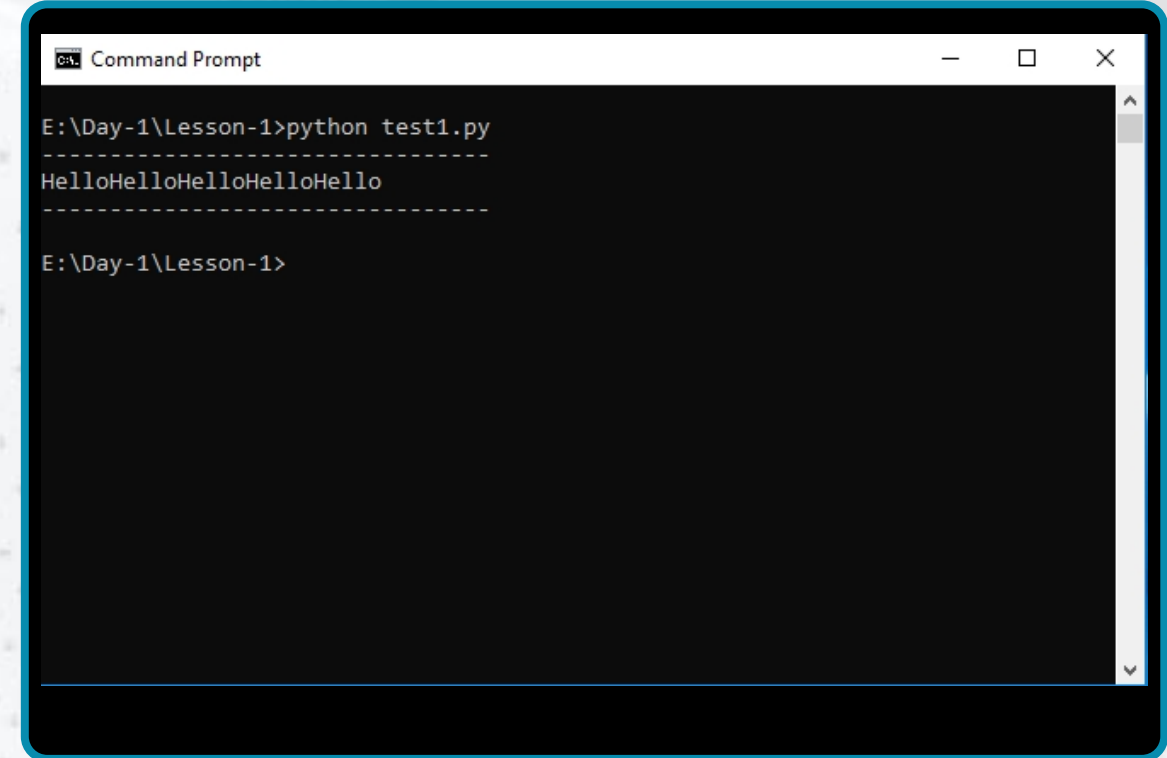
```
if (edad >= 18) and (edad < 30):  
    print('Joven')  
elif (edad >= 30) and (edad < 40):  
    print('Adulto Joven')  
elif (edad >= 40) and (edad < 60):  
    print('Adulto')  
elif edad >= 60:  
    print('Adulto Mayor')  
else:  
    print('Menor de Edad')
```

Adulto Joven

Funciones

Creando un script en Python

- Crear un script
- Ejecutar un script
- Ejecución con errores



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command prompt is open at the directory "E:\Day-1\Lesson-1". The user has entered the command "python test1.py". The output of the script is displayed as "HelloHelloHelloHelloHello", which is centered between two lines of dashed lines. The command prompt is currently at the "E:\Day-1\Lesson-1>" prompt.

```
Command Prompt
E:\Day-1\Lesson-1>python test1.py
-----
HelloHelloHelloHelloHello
-----
E:\Day-1\Lesson-1>
```

Script con Parámetros

- Rescatando parámetros de ejecución en el programa.
- Ejecutando con parámetros.

```
import sys  
  
print("This argument was passed to the script:", sys.argv[1])
```

```
python test2.py foobar
```

The background of the slide is a grayscale image of a book cover. The cover features a repeating pattern of stylized, overlapping leaf or feather shapes. A solid green rectangular banner is positioned horizontally across the middle of the image, partially obscuring the book cover pattern.

Dudas y consultas

**CUANDO EL PROFE TE DEJA DE TAREA
CREAR UN VIDEOJUEGO CON PYTHON**



DIOS, SOY YO DE NUEVO...

¡¡¡A practicar!!!