

CONQUER
BLOCKS

PYTHON

VARIABLES, TIPOS DE DATOS
Y OPERACIONES BASICAS



QUE ES PYTHON

Lenguaje creado en los años 80 por Guido van Rossum en el Centro para las Matemáticas y la Informática CWI de los países bajos

Multiplataforma: Unix, Linux, MacOS, Windows

Multiparadigma (alto nivel) : permite programación orientada a objetos, programación estructurada y programación funcional.

Sintaxis compacta, sencilla e intuitiva, con una curva de aprendizaje mínima y una potente librería de funciones y clases.

Un programa de Python no se compila si no que se ejecuta directamente (usa un intérprete). Eso permite hacer cosas que son imposibles en otros lenguajes como ejecutar instrucciones de manera interactiva, crear funciones al vuelo mientras un programa se ejecuta, interpretar un string como código Python y ejecutarlo etc.



LINEA DE COMANDOS VS SCRIPTS

Linea de comandos de Python en la terminal

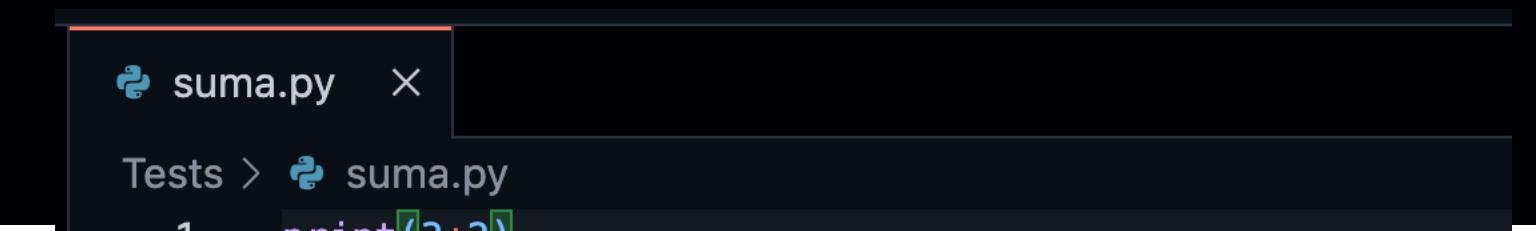
```
[base] MacBook-Pro-2:Python tu_nombre_de_usuario$ python
○ (ConquerB) MacBook-Pro-2:Python tu_nombre_de_usuario$ python
Python 3.9.15 (main, Nov 24 2022, 08:29:02)
[Clang 14.0.6 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 'Esto es una linea de comandos de python'
'Esto es una linea de comandos de python'
>>> 3+2
5
>>> exit()
```



LINEA DE COMANDOS VS SCRIPTS

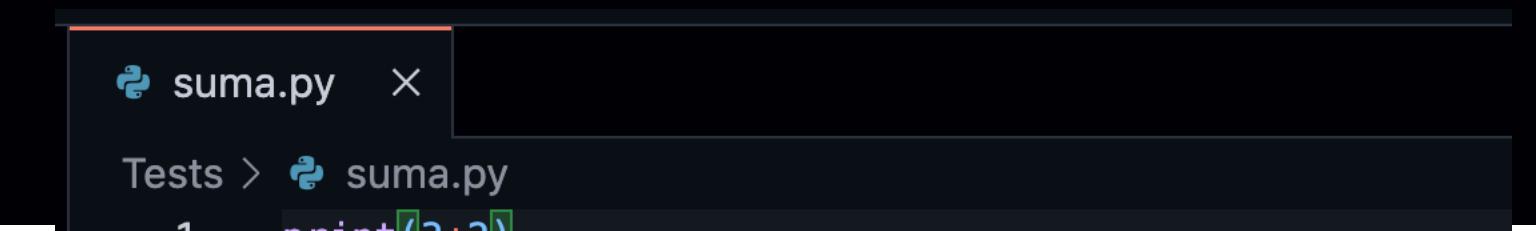
Línea de comandos de Python en la terminal

```
(base) MacBook-Pro-2:Python tu_nombre_de_usuario$ python
Python 3.9.15 (main, Nov 24 2022, 08:29:02)
[Clang 14.0.6 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 'Esto es una linea de comandos de python'
'Esto es una linea de comandos de python'
>>> 3+2
5
>>> exit()
```



The terminal window shows the Python interpreter running. It starts with the Python version and copyright information. Then it enters an interactive mode with three prompts ('>>>'). The first two prompts are strings: "'Esto es una linea de comandos de python'" repeated twice. The third prompt is a numerical expression '3+2', which is evaluated and printed as '5'. Finally, the user exits the session with 'exit()'. The terminal has a dark background with white text.

Python Scripts



suma.py

```
Tests > suma.py
1 print(3+2)
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit <https://support.apple.com/kb/HW20492>.

```
(base) MacBook-Pro-2:Python tu_nombre_de_usuario$ conda activate conquerblocks
(base) MacBook-Pro-2:Python tu_nombre_de_usuario$ /Users/tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/
(base) MacBook-Pro-2:Python tu_nombre_de_usuario$ 5
(base) MacBook-Pro-2:Python $
```

The code editor shows a single file named 'suma.py' with one line of code: 'print(3+2)'. Below the code editor are four tabs: 'PROBLEMAS', 'SALIDA', 'TERMINAL', and 'CONSOLA DE DEPURACIÓN'. The 'TERMINAL' tab is currently selected and displays the output of the Python script. It also includes a note about switching to the zsh shell. The terminal has a dark background with white text.



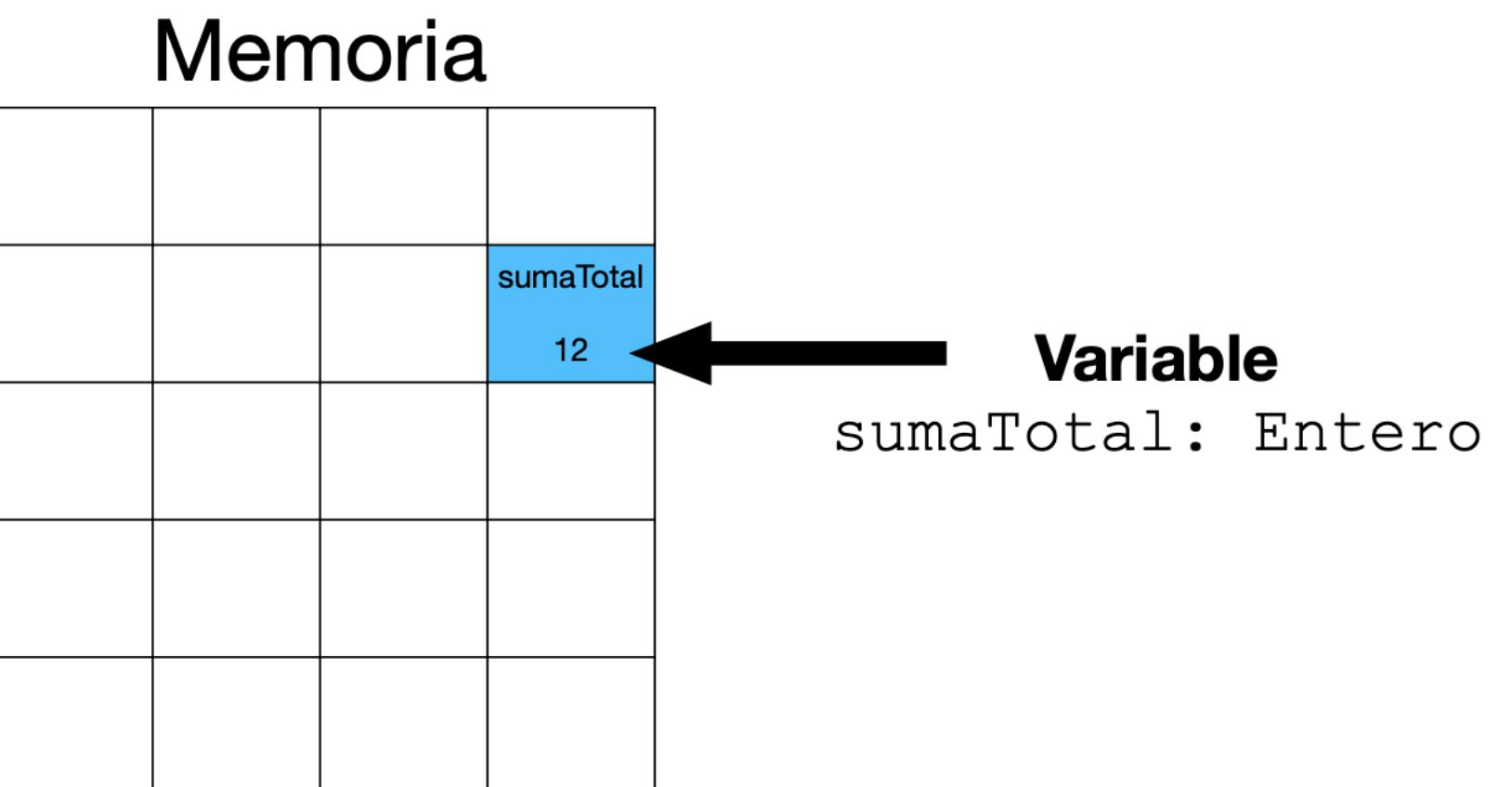
VARIABLES

DEFINICIÓN: Espacio reservado en memoria que tiene asignado un identificador



VARIABLES

DEFINICIÓN: Espacio reservado en memoria que tiene asignado un identificador





VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero  
Definir precio Como Real  
Definir nota Como Texto  
Definir terminado Como Logica
```

Memoria

sumaTotal			
	precio		terminado
	nota		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero  
Definir precio Como Real  
Definir nota Como Texto  
Definir terminado Como Logica
```

Memoria

sumaTotal			
	precio		terminado
	nota		

INICIALIZACIÓN

```
sumaTotal = 12  
precio = 20.5  
nota = "Hola"  
terminado = Falso
```

Memoria

sumaTotal			
12			
	precio		terminado
	20.5		Falso
	nota		
	"Hola"		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero  
Definir precio Como Real  
Definir nota Como Texto  
Definir terminado Como Logica
```

Memoria

sumaTotal			
	precio		terminado
	nota		

En PYTHON **no declaramos** las variables.
Las variables se inicializan directamente

INICIALIZACIÓN

```
sumaTotal = 12  
precio = 20.5  
nota = "Hola"  
terminado = Falso
```

Memoria

sumaTotal			
12			
	precio		terminado
	20.5		Falso
	nota		
	"Hola"		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero
Definir precio Como Real
Definir nota Como Texto
Definir terminado Como Logica
```

Memoria

sumaTotal			
	precio		terminado
	nota		

En PYTHON **no declaramos** las variables.
Las variables se inicializan directamente

```
numero_entero = 42
numero_decimal = 12.5
texto = 'hola'
variable_logica = True
```

INICIALIZACIÓN

```
sumaTotal = 12
precio = 20.5
nota = "Hola"
terminado = Falso
```

Memoria

sumaTotal			
12			
	precio		terminado
	20.5		Falso
	nota		
	"Hola"		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero
Definir precio Como Real
Definir nota Como Texto
Definir terminado Como Logica
```

Memoria

sumaTotal			
	precio		terminado
	nota		

En PYTHON **no declaramos** las variables.
Las variables se inicializan directamente

```
numero_entero = 42
numero_decimal = 12.5
texto = 'hola'
variable_logica = True
```

INICIALIZACIÓN

```
sumaTotal = 12
precio = 20.5
nota = "Hola"
terminado = Falso
```

Memoria

sumaTotal	12		
	precio		terminado
	20.5		Falso
	nota		
	"Hola"		

INICIALIZACIÓN EXPLÍCITA

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```



VARIABLES

MODIFICACIÓN

```
sumaTotal = 12  
  
precio = 20.5  
  
nota = "Hola"  
  
terminado = Falso
```

Memoria

sumaTotal			
12			
	precio		terminado
	20.5		Falso
	nota		
	"Hola"		

MODIFICACION EN PYTHON

```
numero_entero = 42  
numero_decimal = 12.5  
texto = 'hola'  
variable_logica = True
```

```
sumaTotal = sumaTotal + 4  
  
precio = 3.4 + 4.6  
  
nota = "Adios"  
  
terminado = Verdadero
```

Memoria

sumaTotal			
16			
	precio		terminado
	8.0		Verdadero
	nota		
	"Adios"		

```
numero_entero = numero_entero + 4  
numero_decimal = 12.5 + 4.6  
texto = 'adios'  
variable_logica = False
```



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

my_variable_1

_my_variable_1

1_my_variable



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras

my_variable_1 ✓

_my_variable_1 ✓

1_my_variable ✗

my variable ✗



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras

No se deben usar palabras que están asociadas ya a funciones internas de python

my_variable_1 ✓

_my_variable_1 ✓

1_my_variable ✗

my variable ✗

por ejemplo: print ✗



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓

1_my_variable ✗

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras my variable ✗

No se deben usar palabras que están asociadas ya a funciones internas de python

por ejemplo: print ✗

Los nombres de variables deben ser cortos pero descriptivos

nombre >> n

nombre_estudiante >> n_e

tamaño_nombre >> tamaño_del_nombre_de_las_personas



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓

1_my_variable ✗

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras

my variable ✗

No se deben usar palabras que están asociadas ya a funciones internas de python

por ejemplo: print ✗

Los nombres de variables deben ser cortos pero descriptivos

nombre >> n

nombre_estudiante >> n_e

tamaño_nombre >> tamaño_del_nombre_de_las_personas

Cuidado al usar la ele minúscula ‘l’, la i mayúscula ‘I’ y la letra o mayúscula ‘O’. Al leer el código el usuario puede confundirlos con unos y ceros ‘1’, ‘0’



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
print(x,y,z)
```

10 10 10



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
  
print(x,y,z)  
  
10 10 10
```

```
x, y, z = 10,20,30  
  
print(x,y,z)  
2] ✓ 0.1s  
  
.. 10 20 30
```



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
print(x,y,z)
```

10 10 10

```
x, y, z = 'texto 1', 'texto 2', 'texto2'  
print(x,y,z)
```

✓ 0.2s

texto 1 texto 2 texto2

```
x, y, z = 10,20,30  
print(x,y,z)
```

[2] ✓ 0.1s

.. 10 20 30



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
print(x,y,z)
```

10 10 10

```
x, y, z = 'texto 1', 'texto 2', 'texto2'  
print(x,y,z)  
✓ 0.2s
```

texto 1 texto 2 texto2

```
x, y, z = 10,20,30  
print(x,y,z)  
[2] ✓ 0.1s
```

10 20 30

```
x, y, z = 'texto 1', 120.3, 42  
print(x,y,z)  
✓ 0.2s
```

texto 1 120.3 42



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegro de conocerte', nombre)
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegro de conocerte', nombre)
```

```
¿Cómo te llamas?
Elena
Me alegro de conocerte Elena
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegro de conocerte', nombre)
```

```
¿Cómo te llamas?
Elena
Me alegro de conocerte Elena
```

```
nombre = input('¿Cómo te llamas? ')
print('Me alegro de conocerte', nombre)
```

```
¿Cómo te llamas? Elena
Me alegro de conocerte Elena
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegro de conocerte', nombre)
```

OUTPUT

```
¿Cómo te llamas?
Elena
Me alegro de conocerte Elena
```

```
nombre = input('¿Cómo te llamas? ')
print('Me alegro de conocerte', nombre)
```

```
¿Cómo te llamas? Elena
Me alegro de conocerte Elena
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
numero = input('¿Cuántos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cuántos años tienes? 25
Traceback (most recent call last):
  File "/Users/tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = float(input('¿Cuántos años tienes? '))
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cuántos años tienes? 25
Entonces has vivido aproximadamente 9125.0 dias
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = float(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Entonces has vivido aproximadamente 9125.0 dias
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = int(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Entonces has vivido aproximadamente 9125 dias
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = int(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365*numero, 'dias')
```

```
¿Cúantos años tienes? 25.5 
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 17, in <module>
    numero = int(input('¿Cúantos años tienes? '))
ValueError: invalid literal for int() with base 10: '25.5'
```



VARIABLES – PEDIR VALORES

La función `input()` permite obtener texto escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = int(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365*numero, 'dias')
```

```
¿Cúantos años tienes? 25.5
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 17, in <module>
    numero = int(input('¿Cúantos años tienes? '))
ValueError: invalid literal for int() with base 10: '25.5'
```



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```

```
'> numero_entero = 42
     numero_decimal = float(numero_entero)
     print(numero_decimal)

✓ 0.1s
```

42.0



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```

La función `type()` nos devuelve el tipo de dato con el que estamos trabajando

```
numero_entero = 42
numero_decimal = float(numero_entero)
numero_texto = str(numero_entero)
print(type(numero_entero), type(numero_decimal), type(numero_texto))

✓ 0.3s

<class 'int'> <class 'float'> <class 'str'>
```

```
numero_entero = 42
numero_decimal = float(numero_entero)
print(numero_decimal)
```

✓ 0.1s

42.0



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```

La función `type()` nos devuelve el tipo de dato con el que estamos trabajando

```
numero_entero = 42
numero_decimal = float(numero_entero)
numero_texto = str(numero_entero)
print(type(numero_entero), type(numero_decimal), type(numero_texto))

✓ 0.3s

<class 'int'> <class 'float'> <class 'str'>
```

```
✓
numero_entero = 42
numero_decimal = float(numero_entero)
print(numero_decimal)

✓ 0.1s

42.0
```

```
✓
boolean_0 = bool(0)
boolean_1 = bool(1)
boolean_42 = bool(42)
boolean_float = bool(45.3)
boolean_texto = bool('hola')
boolean_texto_vacio = bool('')
print(boolean_0, boolean_1, boolean_42, boolean_float, boolean_texto)
print(boolean_texto, boolean_texto_vacio)

✓ 0.3s

False True True True True
True False
```



VARIABLES - ERRORES TIPICOS

```
variable = 'esta es mi variable'
print(varable)
] ⑧ 0.6s
```

```
NameError                                 Traceback (most recent call last)
Cell In[12], line 2
      1 variable = 'esta es mi variable'
----> 2 print(varable)

NameError: name 'varable' is not defined
```



CONSTANTES

CONSTANTE = VARIABLE

En algunos lenguajes de programación estas variables son inmutables

En Python las constantes **no existen como tal**:

Una constante será una variable que no variaremos a lo largo de nuestro código



STRINGS

VARIABLES DE TIPO TEXTO

Son útiles para muchísimos propósitos:

- representar nombres de usuario y contraseñas
- direcciones de email
- mensajes de error
- links

...



STRINGS O CADENAS DE CARACTERES

VARIABLES DE TIPO TEXTO

```
string1 = "Esto es un texto"
string2 = 'Esto tambien es un texto'
string3 = 'El otro dia le dije a mi amigo, "Python es mi lenguaje favorito"'
print(string1)
print(string2)
print(string3)
```

✓ 0.2s

```
Esto es un texto
Esto tambien es un texto
El otro dia le dije a mi amigo, "Python es mi lenguaje favorito"
```



STRINGS O CADENAS DE CARACTERES

VARIABLES DE TIPO TEXTO

```
string1 = "Esto es un texto"
string2 = 'Esto tambien es un texto'
string3 = 'El otro dia le dije a mi amigo, "Python es mi lenguaje favorito"'
print(string1)
print(string2)
print(string3)
✓ 0.2s
```

```
Esto es un texto
Esto tambien es un texto
El otro dia le dije a mi amigo, "Python es mi lenguaje favorito"
```

```
string5 = esto pretende ser un texto
print(string5)
```

✗ 0.5s

Cell In[26], line 1
string5 = esto pretende ser un texto
^
SyntaxError: invalid syntax



STRINGS

VARIABLES DE TIPO TEXTO

```
string1 = "Esto es un texto"
string2 = 'Esto tambien es un texto'
string3 = 'El otro dia le dije a mi amigo, "Python es mi lenguaje"'
print(string1)
print(string2)
print(string3)
```

✓ 0.2s

Esto es un texto

Esto tambien es un texto

El otro dia le dije a mi amigo, "Python es mi lenguaje"

```
string4 = ''
Este texto es completamente inventado:
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
...
print(string4)
```

✓ 0.1s

Este texto es completamente inventado:

Lorem ipsum dolor sit amet, consectetur adipiscing elit,
 sed do eiusmod tempor incididunt ut labore et dolore magna
 aliqua. Ut enim ad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo consequat.



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.

Cambio de mayúscula a minúscula:

title()

```
nombre = 'juan gomez'
print(nombre.title())
✓ 0.6s
Juan Gomez
```



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.

Cambio de mayúscula a minúscula:

title()

```
nombre = 'juan gomez'  
print(nombre.title())  
✓ 0.6s  
Juan Gomez
```

upper()

```
nombre = 'juan gomez'  
print(nombre.upper())  
✓ 0.6s  
JUAN GOMEZ
```



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.

Cambio de mayúscula a minúscula:

title()

```
nombre = 'juan gomez'  
print(nombre.title())  
✓ 0.6s  
Juan Gomez
```

upper()

```
nombre = 'juan gomez'  
print(nombre.upper())  
✓ 0.6s  
JUAN GOMEZ
```

lower()

```
nombre = 'jUAn goMeZ'  
print(nombre.lower())  
✓ 0.2s  
juan gomez
```



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '
>>> nombre
'python '
>>> nombre.rstrip()
'python'
>>> nombre
'python '
>>> █
```



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '
>>> nombre
'python '
>>> nombre.rstrip()
'python'
>>> nombre
'python '
>>> █
```

```
>>> nombre = nombre.rstrip()
>>> nombre
'python'
>>> █
```



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '
>>> nombre
'python '
>>> nombre.rstrip()
'python'
>>> nombre
'python '
>>> █
```

left strip - lstrip():

```
>>> nombre = ' python'
>>> nombre.lstrip()
'python'
>>> █
```

```
>>> nombre = nombre.rstrip()
>>> nombre
'python'
>>> █
```



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '
>>> nombre
'python '
>>> nombre.rstrip()
'python'
>>> nombre
'python '
>>> 
```

```
>>> nombre = nombre.rstrip()
>>> nombre
'python'
>>> 
```

left strip - lstrip():

```
>>> nombre = ' python'
>>> nombre.lstrip()
'python'
>>> 
```

strip():

```
>>> nombre = ' python '
>>> nombre
' python '
>>> nombre.strip()
'python'
```



STRINGS

Sustituir partes del string:

replace():

```
>>> string = 'Hola.Mundo'  
>>> print(string.replace(".", " "))  
Hola Mundo  
>>> █
```



STRINGS

Sustituir partes del string:

replace():

```
>>> string = 'Hola.Mundo'  
>>> print(string.replace(".", " "))  
Hola Mundo  
>>> █
```

Encontrar un string dentro de otro string:

find():

```
>>> string = 'Hola Mundo'  
>>> print(string.find('Hol'))  
0  
>>> print(string.find('do'))  
8  
>>> print(string.find('hey'))  
-1  
>>> █
```



STRINGS

Comprueba que el string empieza de cierta forma :

startswith():

```
>>> string = 'Hola Mundo'  
>>> print(string.startswith('Hol'))  
True  
>>> print(string.startswith('Mun'))  
False  
>>> █
```

Comprueba que el string termina de cierta forma :

endswith():

```
>>> string = 'Hola Mundo'  
>>> print(string.endswith('do'))  
True  
>>> print(string.endswith('Ho'))  
False  
>>> █
```



STRINGS

Combinar y concatenar texto:

```
✓
nombre = 'juan'
apellido = 'gomez'
nombre_completo = nombre + apellido
print(nombre_completo)

] ✓ 0.3s

juangomez
```



STRINGS

Combinar y concatenar texto:

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print(nombre_completo)  
  
✓ 0.3s  
  
juan gomez
```



STRINGS

Combinar y concatenar texto:

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print(nombre_completo)
```

✓ 0.3s

juan gomez

```
'  
    nombre = 'juan'  
    apellido = 'gomez'  
    nombre_completo = nombre + " " + apellido  
    print("¡Hola, " + nombre_completo.title() + "!")
```

✓ 0.6s

¡Hola, Juan Gomez!



STRINGS

Combinar y concatenar texto:

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print(nombre_completo)
```

✓ 0.3s

juan gomez

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print("¡Hola, " + nombre_completo.title() + "!")
```

✓ 0.6s

¡Hola, Juan Gomez!

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
mensaje = "¡Hola, " + nombre_completo.title() + "!"  
print(mensaje)
```

✓ 0.2s

¡Hola, Juan Gomez!



STRINGS

Tabs y saltos de linea:

Tab - \t :

```
>>> print("Python")
Python
>>> print("\tPython")
      Python
```



STRINGS

Tabs y saltos de linea:

Tab - \t :

```
>>> print("Python")
Python
>>> print("\tPython")
      Python
```

Salto de linea - \n :

```
>>> print("Lenguajes:\nPython\nJavaScript\nSolidity")
Lenguajes:
Python
JavaScript
Solidity
>>> █
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> █
```

Los índices comienzan en **0**



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> █
```

Los índices comienzan en **0**

O extraer partes del mismo:

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> █
```

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> █
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> █
```

Los índices comienzan en **0**

O extraer partes del mismo:

de **0** a **4**

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> █
```

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> █
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> █
```

Los índices comienzan en **0**

O extraer partes del mismo:

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> █
```

de **0** a **4**

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> █
```

de **5** a **8**



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> 
```

Los índices comienzan en **0**

O extraer partes del mismo:

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> 
```

de **0** a **4**

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> 
```

de **5** a **8**

Revertir un string:

```
>>> cadena = 'abcde'  
>>> print(cadena[::-1])  
edcba  
>>> 
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> 
```

Los índices comienzan en **0**

O extraer partes del mismo:

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> 
```

de **0** a **4**

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> 
```

de **5** a **8**

Revertir un string:

```
>>> cadena = 'abcde'  
>>> print(cadena[::-1])  
edcba  
>>> 
```

Consultar el tamaño de un string:

```
>>> cadena = 'abcde'  
>>> print(len(cadena))  
5  
>>> 
```



STRINGS - ERRORES TÍPICOS

```
>>> mensaje = "Jean le Rond d'Alembert fue un gran matematico"
>>> print(mensaje)
Jean le Rond d'Alembert fue un gran matematico
>>>
```

```
>>> mensaje = 'Jean le Rond d'Alembert fue un gran matematico'
      File "<stdin>", line 1
        mensaje = 'Jean le Rond d'Alembert fue un gran matematico'
                           ^
SyntaxError: invalid syntax
```

Syntax Error significa que el interprete no reconoce esta parte del código como código válido de python



NÚMEROS

Enteros o Integers:

```
>>> 2 + 3  
5  
>>> 3 - 2  
1  
>>> 2 * 3  
6  
>>> 3 / 2  
1.5  
>>> █
```

Suma (+), resta(-),
multiplicación (*),
división (/)



NÚMEROS

Enteros o Integers:

```
>>> 2 + 3  
5  
>>> 3 - 2  
1  
>>> 2 * 3  
6  
>>> 3 / 2  
1.5  
>>> █
```

3 ↑ 2

```
>>> 3**2  
9  
>>> 3**3  
27  
>>> 10**6  
1000000  
>>> █
```

Suma (+), resta(-),
multiplicación (*),
división (/)

potencia (**)



NÚMEROS

Enteros o Integers:

```
>>> 2 + 3  
5  
>>> 3 - 2  
1  
>>> 2 * 3  
6  
>>> 3 / 2  
1.5  
>>> [REDACTED]
```

Suma (+), resta(-),
multiplicación (*),
división (/)

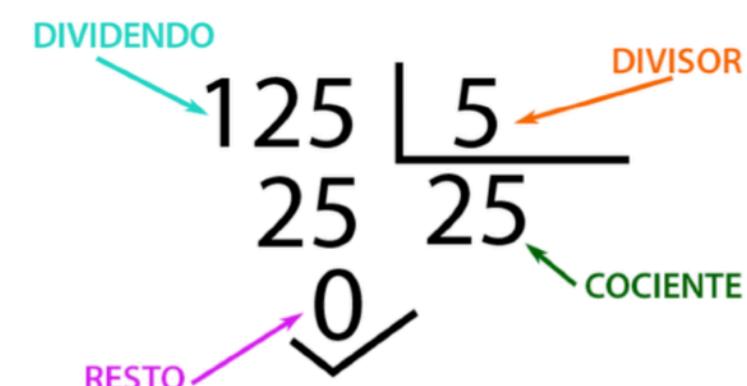
```
3 ↑ 2
```

```
>>> 3**2  
9  
>>> 3**3  
27  
>>> 10**6  
1000000  
>>> [REDACTED]
```

potencia (**)

```
>>> 4 % 3  
1  
>>> 5 % 3  
2  
>>> 6 % 3  
0  
>>> [REDACTED]
```

modulo o resto (%)





NÚMEROS

Enteros o Integers:

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
>>> [REDACTED]
```

Suma (+), resta(-), multiplicación (*), división (/)

3 ↑ 2

```
>>> 3**2
9
>>> 3**3
27
>>> 10**6
1000000
>>> [REDACTED]
```

potencia (**)

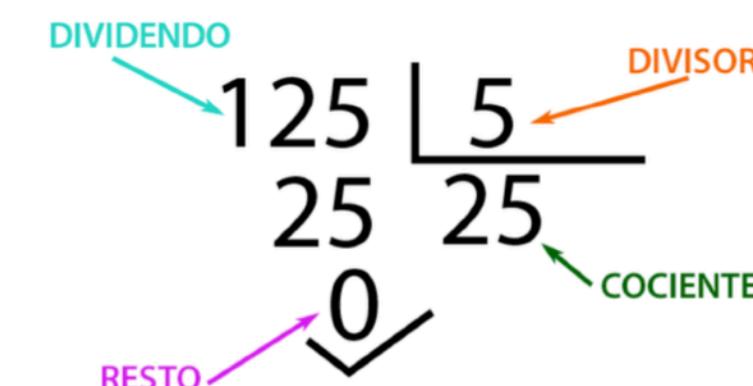
```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 6 % 3
0
>>> [REDACTED]
```

modulo o resto (%)

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
>>> [REDACTED]
```

Python sigue el orden matemático de las operaciones:

1. contenido de los paréntesis
2. exponentes
3. multiplicación y división
4. suma y resta





NÚMEROS

Floats o decimales:

```
>>> 0.2 + 0.2  
0.4  
>>> 2 * 0.1  
0.2  
>>> 2 * 0.2  
0.4  
>>> 0.2 + 0.5  
0.7
```



NÚMEROS

Floats o decimales:

```
>>> 0.2 + 0.2  
0.4  
>>> 2 * 0.1  
0.2  
>>> 2 * 0.2  
0.4  
>>> 0.2 + 0.5  
0.7
```

```
>>> 0.2 + 0.1  
0.3000000000000004  
>>> 3 * 0.1  
0.3000000000000004  
>>> █
```



NÚMEROS

Floats o decimales:

```
>>> 0.2 + 0.2  
0.4  
>>> 2 * 0.1  
0.2  
>>> 2 * 0.2  
0.4  
>>> 0.2 + 0.5  
0.7
```

```
>>> 0.2 + 0.1  
0.3000000000000004  
>>> 3 * 0.1  
0.3000000000000004  
>>> |
```



Python intenta darte tantos valores tras la coma como le es posible. Intenta trabajar con la mayor precisión posible.

El origen de esta imprecisión esta en como los ordenadores están forzados a representar los números de manera interna.

Ocurre en todos los lenguajes de programación.



COMBINAR NUMEROS Y STRINGS

```
>>> numero_dias = 365
>>> mensaje = 'El año tiene ' + numero_dias + 'dias'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```



COMBINAR NUMEROS Y STRINGS

```
>>> numero_dias = 365
>>> mensaje = 'El año tiene ' + numero_dias + 'dias'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

```
>>> numero_dias = 365
>>> mensaje = 'El año tiene ' + str(numero_dias) + ' dias'
>>> print(mensaje)
El año tiene 365 dias
>>> █
```



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN

OBJETIVO:

- Explicar que debe hacer el código
- Explicar cómo funciona sus distintos segmentos



Especialmente importante en trabajos colaborativos

o al reutilizar código antiguo



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN

OBJETIVO:

- Explicar que debe hacer el código
- Explicar cómo funciona sus distintos segmentos



Especialmente importante en trabajos colaborativos

o al reutilizar código antiguo

SINTAXIS:

la almohadilla # indica que esa linea es un comentario

```
# Esto es un comentario  
# Puedo escribir lo que quiera aqui  
# y el interprete lo ignorará  
print('hey!')  
  
✓ 0.7s  
  
hey!
```



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN

TRUCO:

Los strings que no están asociados a una variable son ignorados por el intérprete.

```
"Esto es un string"
'Esto también es un string'
...
Y esto
también
lo
es
...
print('hey!')
]
✓ 0.3s

hey!
```

SINTAXIS:

la almohadilla # indica que esa linea es un comentario

```
# Esto es un comentario
# Puedo escribir lo que quiera aqui
# y el interprete lo ignorará
print('hey!')

✓ 0.7s

hey!
```



RESUMEN

1. El uso de variables y su nomenclatura
2. Strings y como trabajar con sus funciones y métodos
3. Como trabajar con números y operaciones aritméticas
4. Combinar números y strings
5. Leer valores de entrada
6. Conversión entre tipos de datos
7. Uso de los comentarios

CONQUER
BLOCKS