

# Python Academy

# Fundamentals

Clase 1

¡Bienvenidos!



# Temario

- ¿Qué es Python?
- ¿Por qué Python?
- El intérprete
  - Modo interactivo
  - Ejecutando archivos
- Operadores y su precedencia
- Variables y sus tipos
- Estructuras de control
  - Condicionales
  - Bucles
- La función *range()*
- Otras herramientas de control



# ¿Qué es Python?

Python es un lenguaje de programación interpretado cuya máxima es la legibilidad de su código.

Fue creado por el holandés Guido van Rossum, es open-source y administrado por la Python Software Foundation.

Es multiparadigma: Soporta la programación imperativa, programación orientada a objetos y la programación funcional.

Es multiplataforma: Se puede encontrar un intérprete de Python para los principales sistemas operativos (Windows, Linux, Mac OS, etc.). Además, se puede reutilizar el mismo código en cada una de las plataformas.

Es dinámico: El tipo de las variables se decide durante el tiempo de ejecución.

Es interpretado: El código no se compila a lenguaje máquina. Es traducido a medida que se lo requiere.



# ¿Por qué Python?

- Es open-source, de libre uso y distribución.
- Es uno de los lenguajes más versátiles que existen.
- Es fácil. Si ya sabés programar, más aún. Si no, es un excelente punto de partida.
- Por ser tan popular, cuenta con una enorme comunidad de desarrolladores y aficionados que viven encontrándole nuevos usos.
- Python Package Index (PyPI) cuenta con miles de módulos desarrollados por terceros, y vos podés escribir los tuyos para que los use quien quiera o reutilizarlos en otros proyectos propios.
- Su sintaxis es muy similar al lenguaje inglés. Leer código escrito en Python muchas veces se siente como estar leyendo pseudocódigo o un texto en inglés.
- Fue diseñado para ser rápido de aprender, usar y entender. Hace mucho énfasis en la escritura de código limpio y uniforme.



# Requerimientos

Para seguir el curso, se recomienda contar con las siguientes herramientas:

- Python >= 3.6
  - Se puede descargar desde <https://www.python.org/downloads/>
- Algún IDE o editor de código.

Ejemplos:

- Visual Studio Code
- Atom
- Notepad++
- Geany
- Thonny
- PyCharm



# El intérprete

El intérprete funciona de manera similar al shell de Unix:

- Cuando se lo llama con una entrada estándar conectada a una terminal, lee y ejecuta comandos de manera interactiva. Se dice entonces que el intérprete está en **modo interactivo**.
  - ✓ De mucha utilidad para ejecutar programas pequeños, realizar cálculos y leer la documentación del lenguaje.
  - ✓ Para acceder, solo hace falta ejecutar `python` en una terminal.
- Cuando se lo llama con un archivo como entrada estándar, lee y ejecuta un *script* desde ese archivo.  
Ejemplo:  
`> python mi_script.py`
- Algunos módulos de Python también son útiles como scripts. Estos pueden invocarse utilizando:  
`> python -m module [args] ...`



# Operadores y su precedencia

Nº	TIPO	SÍMBOLOS
1	acceso de miembro	expr.miembro
	llamadas a funciones/métodos	expr(...)
2	subíndices/segmentos	expr[...]
3	exponenciación	**
4	operadores unarios	+expr, -expr, ~expr
5	multiplicación, división	*, /, //, %
6	suma, resta	+, -
7	desplazamiento bit a bit	<<, >>
8	and bit a bit	&
9	xor bit a bit	^
10	or bit a bit	
	comparaciones	is, is not, ==, !=, <, <=, >, >=
11	contención	in, not in
12	not lógico	not expr
13	and lógico	and
14	or lógico	or
15	condicional	val1 if cond else val2
16	asignaciones	=, +=, -=, *=, etc.



# Variables

Una variable es una manera de identificar un dato almacenado en memoria.

Es una suerte de contenedor, cuyo contenido puede cambiar durante la ejecución del programa.

Permite acceder fácilmente a su contenido para ser leído, manipulado o transformado.

```
a = 1      # inicializamos <a> con el valor 1
b = 2      # inicializamos <b> con el valor 2
c = a + b    # <c> (3) es definida como el resultado de una sentencia

a, b, c = 1, 2, 3    # se puede asignar múltiples valores a múltiples variables en una sola línea
print(a, b, c)      # 1 2 3

a = b = c = 1      # y asignar un mismo valor a múltiples variables
print(a, b, c)      # 1 1 1

"""
Esto es un comentario mult-línea. Todo** lo que escriba entre pares de tres comillas dobles o simples es considerado
parte del comentario hasta que este no finalice.

**Todo incluye ", ' , #, print(), etc.
"""


```



# Tipos de variables

A diferencia de otros lenguajes de programación estáticos (los tipos de variables se definen de antemano y suelen ser inmutables), Python no posee demasiados tipos de variables.

Entre los más comunes podemos destacar:

- **Números:**

```
"""
Enteros (integers)
"""

entero = 2 + 5      # <entero> es un integer de valor 7

type(entero)      # int

"""

Punto flotante (floats)
"""

real = 1.1 + 2.2    # <real> es un float de valor 3.3 (aprox.)

print(real)        # 3.3000000000000003

print(f'{real:.1f}')  # 3.3
```

```
"""
Complejos (complex)
"""

comp = 1 + 2j      # los números complejos consisten de una parte
                    # real y otra imaginaria

print(comp.real)   # 1.0

print(comp.imag)   # 2.0

"""

Python acepta operaciones aritméticas entre tipos distintos de
números, tal que...

"""

res = 2 + 3j + 5.7

print(res)         # (7.7+3j)
```



# Tipos de variables

- **Cadenas:** secuencias o cadenas de caracteres **inmutables** (strings).

```
a = 'Esta es una cadena'      # comillas simples  
  
b = "y esta es otra."        # comillas dobles  
  
c = a + " " + b      # se pueden concatenar como sumas  
  
print(c)    # 'Esta es una cadena y esta es otra'  
  
"""  
Existen distintos tipos de cadenas (f: format; r: raw; b: bytes; u: unicode).  
  
Dos de las más comunes:  
"""  
  
curso = 'Python Academy Fundamentals'  
  
print(f'Bienvenidos a {curso}')    # Bienvenidos a Python Academy Fundamentals  
  
dir_ = 'C:\\\\Users\\\\Python_Academy'    # tengo que "escapar" \ con otro \  
dir_ = r'C:\\Users\\Python_Academy'     # mismo resultado
```

```
"""  
Otras formas de formar cadenas a partir de otras cadenas:  
"""  
  
curso = 'Python Academy Fundamentals'  
  
print('Bienvenidos a {}'.format(curso))  
  
print('Bienvenidos a %s' % curso)      # deprecado
```



# Tipos de variables

- **Booleanas:** verdadero (True) y falso (False).

```
"""
El método bool()

"""

a = bool('abc')      # True
b = bool(0)          # False
c = bool(None)       # False
d = bool(dict)       # True
e = bool(dict())     # False

"""

Comparaciones

"""

a == b      # False
a == True    # True
0 > 1      # False
0 <= 0      # True
0 != 1      # True
```

```
"""
El operador 'not'

"""

not True      # False
not False     # True

"""

El operador 'and'

"""

True and True   # True
True and False  # False
False and False # False

"""

El operador 'or'

"""

True or False   # True
True or True    # True
False or False  # False
```

```
"""
El operador 'is'

"""

1 is 1        # True
False is bool(None)  # True
1 is not 0     # True

"""

El operador 'in'

"""

'tho' in 'python'  # True
a = [1, 3, 5]
2 in a        # False

"""

bool como número

"""

True + False    # 1
True + True     # 2
True - True     # 0
```

```
"""
Operadores concatenados
"""

0 < 1 < 2      # True
True and 0 > 1   # False
not False and not None  # True

a = 0
a is a < 1      # True
(a is a) < 1     # False
False is a       # False
False == a       # True
'abc' < 'abcd' < 'abce'  # True

"""

Identidad

"""

x = []
y = []
x is x      # True
x is y      # False
x == y      # True
```



# Tipos de variables

- **Listas:** secuencias **mutables** de valores.

```
a = [1, 'dos', int("3")]      # las listas están delimitadas por []
b = list((4, [5, 'seis']))    # y pueden ser inicializados con el método list()
c = a + b        # se pueden sumar
print(c)      # [1, 'dos', 3, 4, [5, 'seis']]
```

- **Tuplas:** secuencias **inmutables** de valores.

```
a = (1, 'dos', int("3"))      # las tuplas están delimitadas por ()
b = tuple([4, [5, 'seis']])    # y pueden ser inicializadas con el método tuple()

print(a, b)      # (1, 'dos', 3) (4, [5, 'seis'])
```



# Tipos de variables

- **Conjuntos:** secuencias de valores **únicos**.

```
a = {1, 2, 3}      # los conjuntos están delimitados por {}
b = set([1, 3])    # y pueden ser inicializados con el método set()
c = a - b         # se pueden restar
print(c)          # {2}
print(set(['python', 'academy', 'python']))    # {'academy', 'python'}
```

- **Diccionarios:** pares de claves **únicas** y valores.

```
a = {'uno': 1, 2: 'dos', 'tres': int("3")}      # los diccionarios están delimitados por {}
                                                # y sus claves y valores separados por :
b = a['uno']        # permiten acceder a un valor llamando a su respectiva clave
print(b)          # 1
c = dict(nombre='Python Academy', año=2021)    # y pueden ser inicializados con el método dict()
print(c)          # {'nombre': 'Python Academy', 'año': 2021}
```



# Estructuras de control: condicionales

- Sentencias **if**: ejecutan un bloque de código si la condición es verdadera (True).

```
# ejemplo 1
# imprime si <num> es igual, mayor o menor que 0

num = 0

if num == 0:
    print('<num> es igual a 0.')    # esto es lo que se va a ejecutar

elif num > 0:
    print('<num> es mayor que 0.')

else:
    print('<num> es menor que 0.')
```



```
# ejemplo 2
# imprime el valor de <cadena> si bool(cadena) == True
# o que <cadena> está vacía si bool(cadena) == False

cadena = ""

if cadena:
    print(f'El valor de <cadena> es "{cadena}".')

else:
    print('<cadena> está vacía.')    # esto es lo que se va a ejecutar
```



# Estructuras de control: bucles

- Sentencias **while**: ejecutan un bucle mientras la condición sea verdadera (True).

```
# ejemplo 1
# imprime números hasta llegar al 10 (sin contar), arrancando por 0

n = 0

while n < 10:    # mientras que <n> sea menor que 10

    print(n)

    n += 1      # equivalente a decir 'n = n + 1'
```

```
# ejemplo 2
# imprime números indefinidamente, esperando 0.5 segundos entre cada iteración

import time

n = 0

while True:      # True siempre es verdadero, por lo que el bucle corre indefinidamente

    print(n)

    n += 1

    time.sleep(0.5)    # llamamos al método 'sleep' de 'time' y le pasamos segundos
```



# Estructuras de control: bucles

- Sentencias **for**: iteran sobre los valores de cualquier secuencia (listas, cadenas, etc.) en su orden original.

```
# ejemplo 1
# imprime si los números en <enteros> son pares o impares

enteros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for n in enteros:      # por cada <n> en <enteros>

    if n % 2:      # si el resto != 0

        print(f'{n} es impar.')

    else:      # si el resto == 0

        print(f'{n} es par.')
```

```
# ejemplo 2
# imprime la cantidad de letras en cada palabra en <palabras>

palabras = ['python', 'academy', 'fundamentals']

for p in palabras:

    print(f'{p} tiene {len(p)} letras.')
```

```
# ejemplo 3
# agrega los valores en <temperaturas> menores que 20 a <menos_que_20>

temperaturas = [27, 31, 13, 9, 0, 34]

menos_que_20 = []

for t in temperaturas:

    if t < 20:

        menos_que_20.append(t)
```



# Estructuras de control

- La función **range**: genera progresiones aritméticas.

```
# ejemplo 1
# imprime si los números entre el 0 y el 9 (incluyéndolos) son pares o impares

for n in range(10):    # equivalente a range(0, 10) (no incluye al número final)

    if n % 2:

        print(f'{n} es impar')

    else:

        print(f'{n} es par')
```

```
# ejemplo 2

sum(range(4))      # 6 (0 + 1 + 2 + 3)

list(range(0, 10, 3))  # [0, 3, 6, 9]
```

```
# ejemplo 3
# imprime el número de índice de cada objeto en una lista junto al objeto

# lista[n] es la manera de llamar al valor en esa posición

lista = ['python', 'academy', 'fundamentals']

for i in range(len(lista)):

    print(i, lista[i])
```



# Estructuras de control

- Las sentencias **break** y **continue**:

```
# usamos 'break' para salir de un bucle

num = 7

opcion = int(input('Ingresá un número: '))      # convertimos al valor en entero

while True:

    if opcion == num:

        print('¡Ganaste!')

        break      # si se cumple la condición, el bucle termina

    print('Número incorrecto.')

    opcion = int(input('Ingresá un número: '))      # se reemplaza a <opcion> para
                                                    # volver a comparar
```

```
# usamos 'continue' para proseguir a la siguiente iteración

palabras = ['python', 'academy', 'fundamentals', 'ibm', 'kyndryl']

for p in palabras:

    if len(p) < 4:      # si len(p) es menor que 4

        continue

    if p.startswith('f'):      # si <p> empieza con 'f'

        continue

    print(p)
```



# Estructuras de control

- Las sentencias **pass** y **else** en bucles:

```
# usamos 'pass' para seguir de largo sin hacer nada

# ejemplo 1
# imprime sólo los números pares en <enteros>

enteros = list(range(10))

for n in enteros:

    if n % 2 == 0:

        print(n)

    else:

        pass      # todavía no sé lo que quiero hacer acá
```

```
# ejemplo 2
# 'while True' como busy-wait hasta que se
# ejecute Ctrl + C

while True:

    pass
```

```
# usamos 'else' en bucles para ejecutar sentencias en ausencia de 'break'

# ejemplo 1
# imprime <num> y avisa cuando vale más que 6

num = 1

while num < 6:      # mientras <num> sea menor que 6

    print(num)

    num += 1

else:    # si <num> es igual o mayor que 6

    print('<num> ya no vale menos que seis.')
```

```
# ejemplo 2
# imprime los números en range(6) y avisa cuando termina

for num in range(6):

    print(num)

else:    # cuando el bucle haya terminado sin 'break'

    print("¡Terminé!")
```



# Python Academy

# Fundamentals

Clase 1

¡Gracias!

