

Introducción a Python

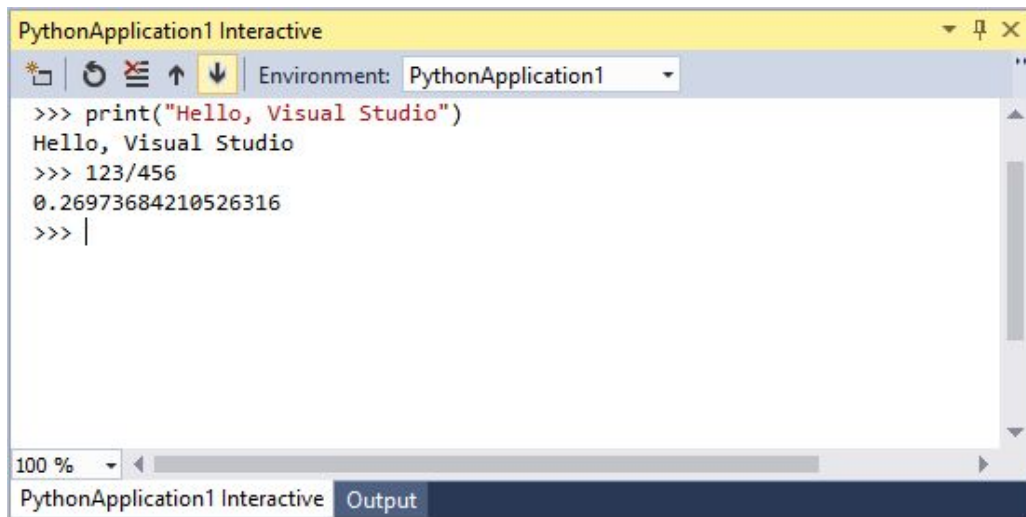
Módulo de Programación.
CFGS Desarrollo de Aplicaciones Web
IES Gran Capitán





Sesiones interactivas

Los lenguajes interpretados suelen ofrecer una herramienta de ejecución interactiva.



```
PythonApplication1 Interactive
>>> print("Hello, Visual Studio")
Hello, Visual Studio
>>> 123/456
0.26973684210526316
>>> |
```



Expresiones

Una *expresión* es una **combinación de valores y operaciones** que, al ser evaluados, entregan un **valor** de un **tipo** determinado.

Pueden formar parte de una expresión:

- Valores literales (como 2, "hola" o 5.7).
- Variables.
- Operadores.
- Llamadas a funciones.



Tipos de expresiones

- Aritméticas.
 - $a + 2 * b$
 - $(x + 2) / 5$
- Relacionales.
 - $a < 6$
 - $x \neq y$
- Lógicas.
 - $a < 6 \text{ and } b > 5$
 - `not es_primo`
- Alfanuméricas.
 - `"Hola" + " " + "mundo"`



Operadores aritméticos

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
División entera	//	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4



Excepciones

Los errores de ejecución se denominan *excepciones*.

```
>>> 1 + * 3↵
      File "<stdin>", line 1
        1 + * 3
            ^
SyntaxError: invalid syntax

>>> 2 + 3 %↵
      File "<stdin>", line 1
        2 + 3 %
            ^
SyntaxError: invalid syntax

>>> 1 / 0↵
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: int division or modulo by zero
```



Tipos de datos

- Numéricos:
 - Entero (no son infinitos, si me paso obtengo un overflow).
 - En Python sí son infinitos (salvo la limitación de la memoria)
 - Flotante.
 - Un tipo *flotante* no es *real*, es una aproximación.
 - Un número muy pequeño se puede confundir con cero (underflow).
 - Complejos.
- Booleanos.
- Cadenas de caracteres.
- Otros tipos: listas, tuplas, diccionarios, conjuntos...
- Los datos de un tipo son *mutables* o *inmutables*.



Tipo numérico (entero y flotante)

Se pueden mezclar en una expresión, pero el valor del tipo de esta será *flotante*.

```
>>> 1 + 2 + 3 + 4 + 5 + 6 + 0.5↵
21.5
>>> 1 + 2 + 3 + 4 + 5 + 6 + 0.0↵
21.0
```

Los *enteros* ocupan menos memoria que los *flotantes* y las operaciones con ellos son más rápidas.



Literales de entero

Python permite expresar números enteros, además de en base 10 (lo normal), en:

- base 2 (binario).

- Prefijo `0b`: `0b101` `0b11101`

- base 8 (octal).

- Prefijo `0o`: `0o177` `0o255`

- base 16 (hexadecimal).

- Prefijo `0x`: `0x1A9` `0xFF7`



Tipo booleano (lógico)

- Llamado *booleano* por ser propio del **Álgebra de Boole**.
- Solo puede presentar uno de dos valores: **True** o **False**.
- Operadores lógicos:
 - «y lógica» o conjunción (**and**).
 - «o lógica» o disyunción (**or**).
 - Existe un «o exclusivo» (xor) en algunos lenguajes. En Python podemos usarlo como función importando el módulo *operator*.
 - «no lógico» o negación (**not**).



Operador booleano **and**

El operador **and** da como resultado *True* si y solo si son *True* sus dos operandos. Esta es su *tabla de verdad*:

and		
operandos		resultado
izquierdo	derecho	
True	True	True
True	False	False
False	True	False
False	False	False



Operador booleano **or**

El operador **or** proporciona *True* si cualquiera de sus operandos es *True*, y *False* solo cuando ambos operandos son *False*. Esta es su tabla de verdad:

or		
operandos		resultado
izquierdo	derecho	
True	True	True
True	False	True
False	True	True
False	False	False



Operador booleano **not**

El operador **not** es unario, y proporciona el valor *True* si su operando es *False* y viceversa. He aquí su tabla de verdad:

not	
operando	resultado
True	False
False	True



Precedencia de los operadores lógicos

Operación	Operador	Aridad	Asociatividad	Precedencia
Negación	not	Unario	—	alta
Conjunción	and	Binario	Por la izquierda	media
Disyunción	or	Binario	Por la izquierda	baja

```
>>> True and False↵
```

```
False
```

```
>>> not True↵
```

```
False
```

```
>>> (True and False) or True↵
```

```
True
```

```
>>> True and True or False↵
```

```
True
```

```
>>> False and True or True↵
```

```
True
```

```
>>> False and True or False↵
```

```
False
```



Operadores relaciones (comparación)

operador	comparación
!=	es distinto de
==	es igual que
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

```
>>> 5 >= 5↵
```

```
True
```

```
>>> 1 != 0↵
```

```
True
```

```
>>> 1 != 1↵
```

```
False
```

```
>>> -2 <= 2↵
```

```
True
```



Asociatividad de los operadores relacionales

Normalmente los operadores relacionales no son asociativos por la izquierda (como los aritméticos).

$2 < 3 < 4$ equivale a $(2 < 3)$ and $(3 < 4)$.

Sin embargo, Python sí la permite, es una “rareza” en el mundo de los lenguajes de programación.



Tabla completa de operadores

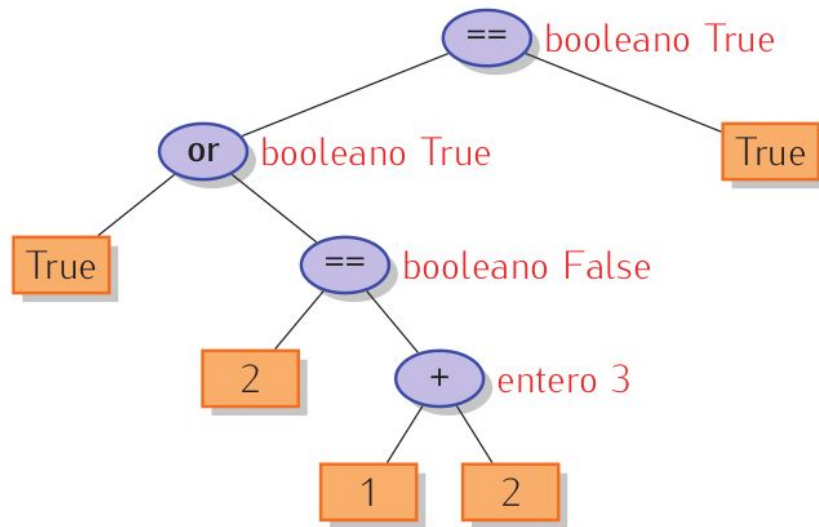
Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
División entera	//	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4
Igual que	==	Binario	—	5
Distinto de	!=	Binario	—	5
Menor que	<	Binario	—	5
Menor o igual que	<=	Binario	—	5
Mayor que	>	Binario	—	5
Mayor o Igual que	>=	Binario	—	5
Negación	not	Unario	—	6
Conjunción	and	Binario	Por la izquierda	7
Disyunción	or	Binario	Por la izquierda	8



Combinación de operadores

Es posible combinar en una misma expresión operadores aritméticos, lógicos y relacionales.

```
>>> (True or (2 == 1 + 2)) == True  
True
```





El tipo de datos cadena (string)

Una cadena es una **secuencia de caracteres** (letras, números, espacios, marcas de puntuación, etc) y en Python se distingue porque va encerrada entre comillas simples o dobles.

```
>>> nombre = "Fulanito"
>>> apellidos = "de Tal..."
>>> nombre_completo = nombre + ' ' + apellidos
>>> nombre_completo
'Fulanito de Tal...'
>>> nombre * 5
'FulanitoFulanitoFulanitoFulanitoFulanito'
```



Variables

- Creamos variables en Python **inicializándolas**.
 - Se reserva un espacio en la memoria.
 - Se almacena ahí el valor (id).
 - Se asocia la dirección en memoria con el *identificador* (nombre).
- El tipo de dato de la variable será el del valor asignado.
 - Si le asignamos otro de tipo distinto, el tipo de la variable cambia.
- Ojo!!! Las asignaciones NO son ecuaciones.

```
>>> x = 3
```

```
>>> x = x + 1
```

```
>>> x
```

```
4
```



Asignaciones con operador

```
>>> z = 1↵
>>> z += 2↵
>>> z *= 2↵
>>> z /= 2↵
>>> z -= 2↵
>>> z %= 2↵
>>> z **= 2↵
>>> z /= 2↵
>>> z↵
0.5
```



Funciones predefinidas

float

bin

str

len

int

oct

abs

ord

bool

hex

round

chr



Vitaminando con más funciones importando **módulos**.

Python proporciona más funciones a través de **módulos** (math, sys...) que hay que **importar**.

- *from math import sin*
- *from math import sin, cos*
- *from math import **
 - No aconsejado.
- **import** math



Métodos

Los datos de ciertos tipos (incluso los tipos) permiten invocar **métodos**, que podemos considerar funciones “especiales”.

```
>>> mi_cadena = "Hola, ¿qué tal?"
>>> mi_cadena.upper()
'HOLA, ¿QUÉ TAL?'
>>> mi_cadena.lower()
'hola, ¿qué tal?'
>>> mi_cadena.replace("Hola", "HOLA")
'HOLA, ¿qué tal?'
>>> len(mi_cadena) ← Función
15
_
```