

# Introducción a Python 3



# ¿QUE ES PYHTON?

Python es un lenguaje de programación de alto nivel:

- **Simple:** es un lenguaje con sintaxis sencilla.
- **Interpretado:** Python se procesa en tiempo de ejecución por el intérprete.
- **Interactivo:** Se puede interactuar directamente con el intérprete para escribir programas.
- **Software Libre**
- **Orientado a objetos:** Python soporta la orientación a objetos.
- **Ampliable:** Permite combinar fragmentos con otros lenguajes de programación.
- **Incrustable:** Permite insertar código en otros lenguajes para dar facilidades de scripting.
- **Librerías Extendidas:** existen múltiples librerías que permiten ampliar su funcionalidad.

# HISTORIA DE PYTHON

- Python lo diseñó **Guido Van Rossum** al final de los ochenta.
- Rossum publicó la primera versión de Python (0.9.0) en Febrero de **1991** en el CWI (Centrum Wiskunde & Informatica) de Holanda, Amsterdam.
- Python surge del lenguaje de programación **ABC** desarrollado en el CWI.
- Rossum escogió el nombre de "**Python**" porque era un gran fan de los Monty Python.
- Python lo mantiene un grupo de desarrollo en CWI dirigido por Rossum



[https://en.wikipedia.org/wiki/Guido\\_van\\_Rossum#/media/File:Guido\\_van\\_Rossum\\_OSCON\\_2006.jpg](https://en.wikipedia.org/wiki/Guido_van_Rossum#/media/File:Guido_van_Rossum_OSCON_2006.jpg)

# OBJETOS

El mundo está lleno de objetos.



Los objetos se pueden agrupar.

Grupo Coches



Grupo Sillas



# CLASES

Una clase es un prototipo de como son los objetos de un grupo y contiene:

## ATRIBUTOS

Son características que definen al objeto: datos.

## METODOS

Son acciones, operaciones que sirven para modificar los datos que están relacionados con el objeto.

Se construye definiendo (por ejemplo, en el caso de coches):

## ATRIBUTOS

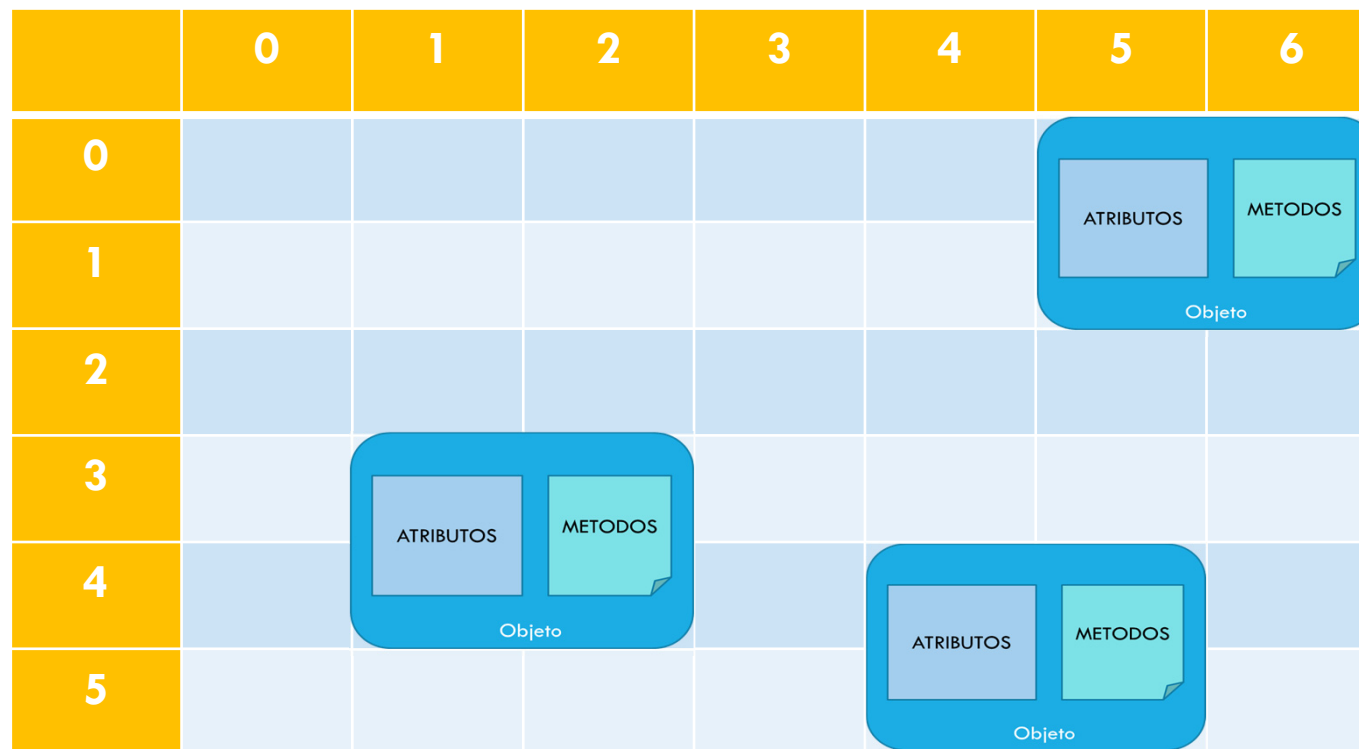
Marca  
Color  
Número de plazas  
Descapotable si/no  
Motor  
Tipo de combustible  
...

## METODOS

Abrir-puerta  
Encender-motor  
Poner-radio  
Quitar-capota  
Girar-volante  
...

# CREAR OBJETOS

Cuando se crea un objeto se reserva una porción de memoria donde se guarda una copia de la plantilla general de la clase a la que pertenece el objeto (se dice que el objeto es una instancia de la clase). Después se pueden rellenar los datos particulares de ese objeto concreto.



# OBJETO:

Los diferentes objetos de una clase, tienen los mismos campos; es decir que el número, y los nombres de los campos de una misma clase son los mismos, mientras que el valor o incluso el tipo de un campo particular de cada objeto puede ser distinto.

Los objetos de clases diferentes pueden tener diferentes campos.

Ejemplo: Un círculo tiene un campo “diámetro”, mientras que un triángulo tiene los campos “ancho” y “alto”.

# ACCEDER A CONTENIDO DE OBJETOS

Imaginad que tengo un objeto que se llama **micoche** que tiene los atributos que aparecen en la ficha.

ATRIBUTOS
Marca: <b>BMW</b>
Color: <b>gris</b>
Número de plazas: <b>4</b>
Descapotable: <b>no</b>
Motor: <b>170CV</b>
Tipo de combustible: <b>Diesel</b>
...

Si quiero imprimir la marca de mi coche puedo escribir:

```
>>>print(micoche.Marca)
```

Y si quiero imprimir el motor que lleva el coche:

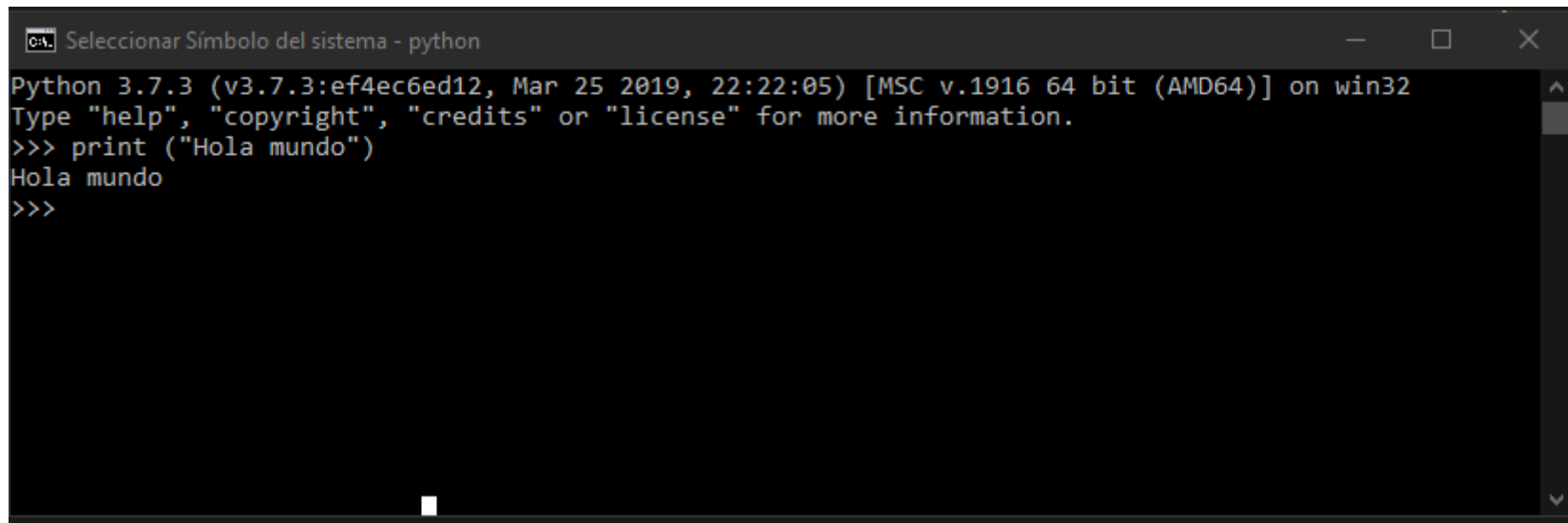
```
>>>print(micoche.Motor)
```

Y si quiero que se abra la puerta, tendré que recurrir al método correspondiente:

```
>>> micoche.Abrir-puerta
```



# APLICACIÓN BÁSICA “HOLA MUNDO”



```
Seleccionar Símbolo del sistema - python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hola mundo")
Hola mundo
>>>
```

# EJEMPLO BÁSICO

```
x = 34-23          # Comentario
y = "Hello"        # Otro
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World" # Concatenación de cadenas
print (x)
print (y)
```

# ENTENDIENDO EL CÓDIGO

- **La indentación es parte del lenguaje:**
  - Sirve para definir los bloques.
- **Las variables se crean al asignarles un valor**
  - Los tipos de las variables no se tienen que declarar. Python se encarga de saber de que tipo son.
- **La asignación usa `=` y se compara con `==`.**
- **Los símbolos `+` `-` `*` `/` `%` corresponden a operadores matemáticos.**
  - Se concatena con `+`.
  - Se usa de forma especial `%` para darle formato a las cadenas (como el printf de C)
- **Los operadores lógicos son palabras (`and`, `or`, `not`)**

# OPERADORES

## Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
-	Negativo
*	Multiplicación
**	Exponente
/	División
//	División entera
%	Residuo

## Operadores relacionales

Operador	Evalúa
==	a == b ¿a igual a b?
!=	a != b ¿a distinta de b?
>	a > b ¿a mayor que b?
<	a < b ¿a menor que b?
>=	a >= b ¿a mayor o igual que b?
<=	a <= b ¿a menor o igual que b?

## Operadores lógicos

Operador	Evalúa
or	a or b ¿Se cumplen a o b?
and	a and b ¿Se cumple a y b?
not	not x Contrario a x

# TIPOS BÁSICOS DE DATOS

## Enteros (por defecto para números)

`z = 5 / 2` # El resultado es 2.5.

## Números reales (llamados en punto flotante)

`x = 3.456`

## Cadenas

- Para indicarlas se puede usar `"` o `'`  
`"abc"` `'abc'` (Son lo mismo.)
- En caso de conflicto se usan ambas.  
`"matt's"`
- Se usan triple doble comillas para múltiples párrafos o para incluir comillas y apóstrofes:  
`"""a'b'c"""`

# ESPACIO EN BLANCO

## El espacio en blanco tiene significado en Python:

- En especial la indentación y los saltos de línea.
- Utiliza un salto de línea para terminar una línea de código.  
Se utiliza un \ para que el salto de línea no se considere.

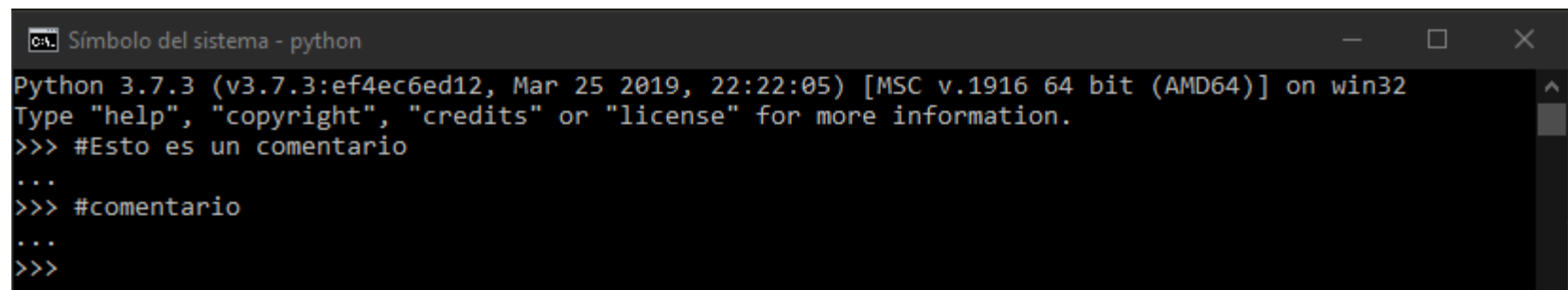
## Se utiliza **indentación consistente** para marcar los bloques.

- Los elementos del mismo bloque tienen la misma indentación.
- El bloque empieza en el primer elemento con esa indentación y termina cuando aparece una indentación menor.

```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
    print ("False") → Error!
```

# COMENTARIOS

Se inician con `#`. Los comentarios NO se ejecutan y son ignorados por el interprete de Python.

A screenshot of a Windows command prompt window titled 'Símbolo del sistema - python'. The window shows the Python 3.7.3 interpreter prompt. The user has entered three lines of code, each starting with a hash symbol followed by a comment. The first line is '>>> #Esto es un comentario', followed by an ellipsis '...'. The second line is '>>> #comentario', followed by an ellipsis '...'. The third line is '>>>' followed by an ellipsis '...'. The window's title bar includes standard Windows window controls (minimize, maximize, close) and the system clock shows 12:00 PM on 10/10/2019.

```
Símbolo del sistema - python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> #Esto es un comentario
...
>>> #comentario
...
>>>
```

Comentarios de múltiples líneas entre triples comillas.

```
'''
print("We are in a comment")
print ("We are still in a comment")
'''
```

# VARIABLES

- En Python no se necesita declarar variables, esto es, no es necesario indicar de que tipo son.
- La declaración ocurre automáticamente cuando se asigna un valor a la variable
- Las variables pueden cambiar de tipo simplemente con asignarles otro valor de diferente tipo..
- Se puede asignar un valor a múltiples variables simultáneamente.
- También se pueden asignar múltiples objetos a múltiples variables al mismo tiempo.

```
counter = 100      # An integer assignment  
miles   = 1000.0   # A floating point  
name    = "John"   # A string  
z       = None     # A null value
```

```
x = 1  
x = "string value"
```

```
a = b = c = 1
```

```
a, b, c = 1, 2, "john"
```



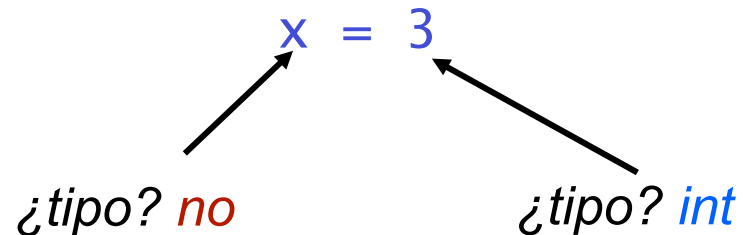
# VARIABLES: ASIGNACIÓN - BINDING

- La asignación de una variable significa que se asigna a un **nombre** una **referencia** a cierto **objeto**

**¡¡La asignación crea referencias, no copias!!**

- Creas el nombre la primera vez que aparece a la izquierda de una expresión.
- Los **nombres** en Python no tienen un tipo propio. Los **objetos** si.

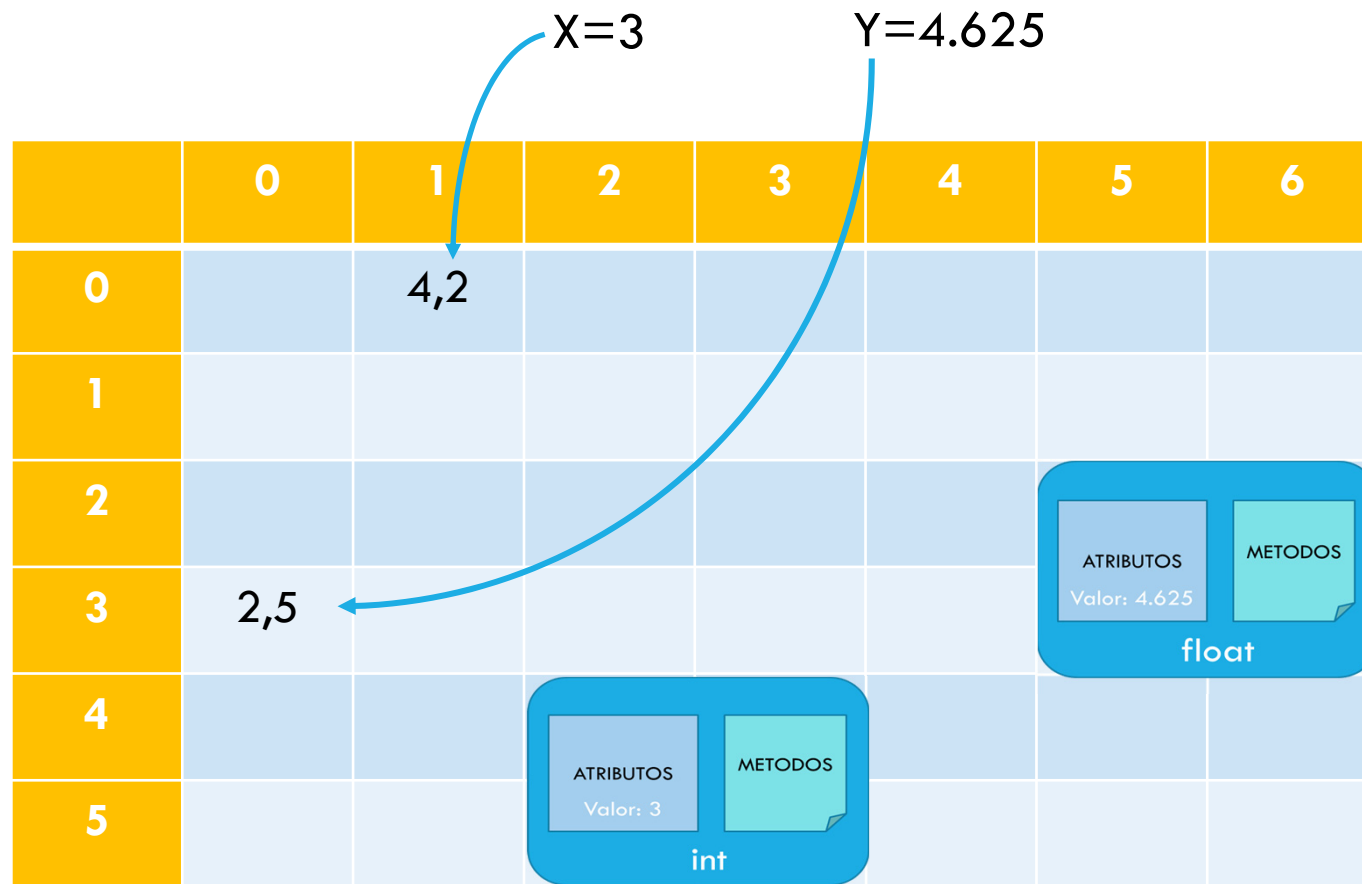
Python determina el tipo de la referencia de forma automática, dependiendo del tipo de objeto que se asigne.



# ASIGNACIÓN

Tabla Asignación Memoria

X	0,1
Y	3,0



# ASIGNACIÓN

Si tratas de utilizar un nombre antes de que sea creado, saldrá un error:

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

```
3
```

```
>>> y
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#16>", line 1, in -toplevel-
```

```
    y
```

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

```
>>> y = 3
```

```
>>> y
```

```
3
```

# NOMBRES VÁLIDOS

- Python distingue entre minúsculas y mayúsculas.
- Los nombres no pueden empezar con número.
- Pueden contener letras, números y sub\_guiones.  
`bob Bob _bob _2_bob_ bob_2 BoB`
- Existen **palabras reservadas** (no pueden usarse como nombres). El IDE os las indicará con un color si las usáis. algunos ejemplos son:  
`and, continue, elif, else, for, if, import, in, not, or, print, return, while, etc...`

# VARIABLES

**Tipo de una variable:** type()

```
>>> variable=1+2j
>>> type(variable)
<type 'complex'>
```

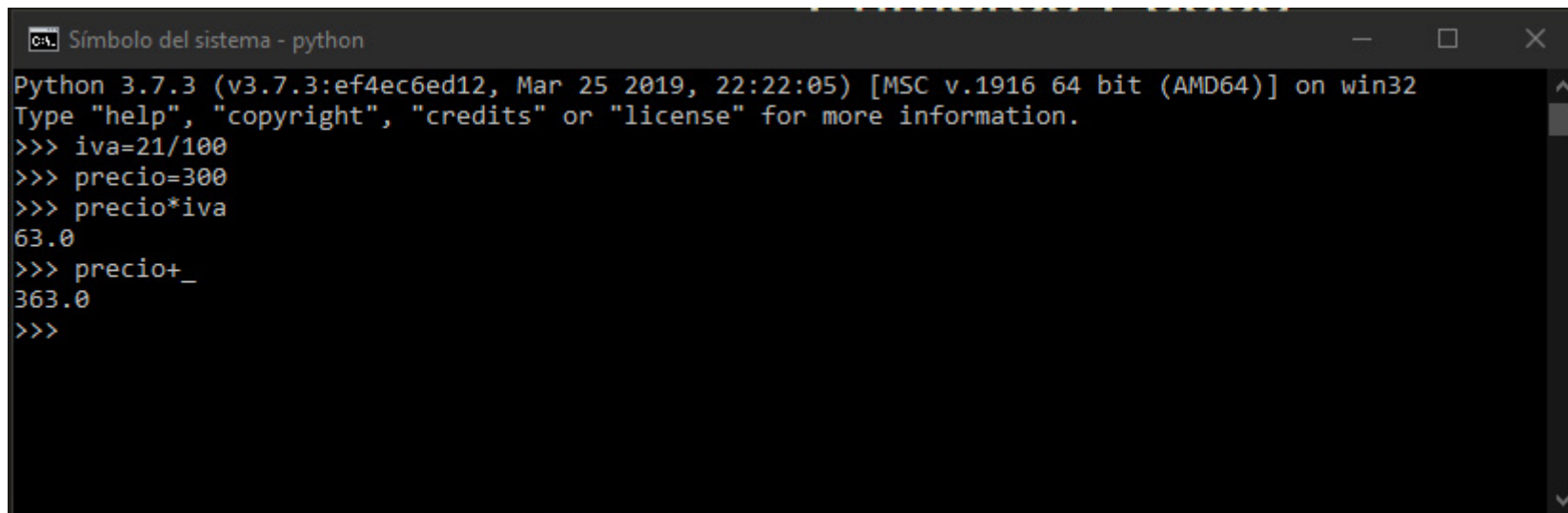
**Identidad de un objeto:** id() – Un entero constante y único asignado a un objeto mientras exista

```
>>> a=5
>>> b=6
>>> c=7
>>> id(a), id(b), id(c)
(505894368, 505894384, 505894400)
>>> a=b=c
>>> id(a), id(b), id(c)
(505894400, 505894400, 505894400)
>>>
```

# MODO INTERACTIVO

En **modo interactivo**, la última expresión impresa se asigna a la variable `_`.

Esto significa que, cuando se usa Python como calculadora, se facilita continuar los cálculos, por ejemplo:



```
Símbolo del sistema - python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> iva=21/100
>>> precio=300
>>> precio*iva
63.0
>>> precio+_
363.0
>>>
```

# NÚMEROS

- Los números son objetos **Inmutables** en Python cuyo valor no puede cambiar.
- Hay tres tipos de números predefinidos en Python 3:
  - Enteros (int)
  - Números en punto flotante (float)
  - Números complejos:  $\langle \text{parte real} \rangle + \langle \text{parte imaginaria} \rangle j$
- **Funciones numéricas comunes**

Function	Description
<b>int</b> (x)	to convert x to an integer
<b>float</b> (x)	to convert x to a floating-point number
<b>abs</b> (x)	The absolute value of x
<b>exp</b> (x)	The exponential of x: $e^x$
<b>log</b> (x)	The natural logarithm of x, for $x > 0$
<b>pow</b> (x,y)	The value of $x^{**}y$
<b>sqrt</b> (x)	The square root of x for $x > 0$

# CONVERSIONES

```
>>> variable='45'
>>> variable+1

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    variable+1
TypeError: cannot concatenate 'str' and 'int' objects
>>> int(variable)
45
>>> variable+1

Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    variable+1
TypeError: cannot concatenate 'str' and 'int' objects
>>> int(variable)+1
46
```

```
>>> variable='34.5'
>>> variable-1.5

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    variable-1.5
TypeError: unsupported operand type(s) for -: 'str' and 'float'
>>> float(variable)-1.5
33.0
```



# INTRODUCCIÓN DE DATOS POR TECLADO

- `input()` Permite la introducción de datos desde el teclado en Python.
- Cuando se llama a esta función, el programa se detendrá hasta que el usuario haya escrito algo en el teclado y pulsado la tecla return.
- `input()` Devuelve lo que ha introducido el usuario como una **cadena de caracteres**. Para no perderlo, habrá que guardarlo en una variable. Si queremos que sea un número, habrá que convertirlo de cadena a número.
- Puede llevar un parámetro opcional que se imprimirá en pantalla.

```
nombre = input("Como te llamas? ")
```

# EXPRESIONES LÓGICAS

***True* y *False*** son constantes en Python.

**Otros valores equivalentes a *True* o *False*:**

*False*: cero, *None*, contenedores u objetos vacíos.

*True*: números distintos a cero, objetos no vacíos.

**Operadores de comparación: `==`, `!=`, `<`, `<=`, etc.**

X y Y tienen el mismo valor: `X == Y`

Si comparas `X is Y`:

X y Y son dos variables que hacen referencia al mismo objeto.

**Se pueden combinar expresiones booleanas.**

Utilizando *and*, *or* y *not*.

Para evitar ambigüedad se necesitan paréntesis.

# CONDICIONES IF

```
if x == 3:
    print ("x vale 3.")
elif x == 2:
    print ("x vale 2.")
else:
    print ("x vale otra cosa.")
print ("Esto ya está fuera del 'if'.")
```

Fíjate:

- El uso de bloques indentados.
- Dos puntos (:) de la expresión booleana.

# EXPRESIONES CONDICIONALES

```
x = valor_verdadero if condición else valor_falso
```

Utiliza también evaluación **Lazy**:

**Primero, se evalúa condición**

**Si devuelve *True*, valor\_verdadero se evalúa y acaba.**

**Si devuelve *False*, valor\_falso se evalúa y acaba.**

# BUCLES WHILE

```
>>> x = 3
>>> while x < 5:
    print(x, "dentro del bucle")
    x = x + 1
3 dentro del bucle
4 dentro del bucle
>>> x = 6
>>> while x < 5:
    print(x, "dentro del bucle")

>>>
```

# BREAK Y CONTINUE

Puedes utilizar la palabra reservada *break* para salir del bucle *while* completamente.

Puedes utilizar la palabra reservada *continue* dentro de un bucle, para detener el procesamiento de la iteración actual para ir inmediatamente a la siguiente.

# BUCLES FOR

- Un bucle for recorre cada uno de los elementos de una colección, o cualquier objeto “iterable”

```
for <elemento> in <colección>:  
    <sentencias>
```

- Si <colección> es un iterador, el for recorre cada elemento de la iteración.
- Si <colección> es una cadena, entonces el ciclo recorre cada carácter de la cadena.

```
for caracter in "Hello World":  
    print (caracter)
```

# LA FUNCIÓN RANGE()

- A menudo queremos iterar sobre una secuencia de enteros.
- La función `range()` es un **Iterador** toma un entero como parámetro y devuelve una secuencia de números del cero a uno antes del número que recibió.

`range(5)` devuelve el iterador `range(0,5)`, lo que implica que va a recorrer los números 0,1,2,3,4

Para imprimir los números uno a uno:

```
for x in range (5): # itera del 0 al 4 de uno en uno
    print (x)
```

```
for x in range (2, 5): # itera del 2 al 4 de uno en uno
    print (x)
```

```
for x in range (3, 10, 2): # itera del 3 al 9 de 2 en 2
    print (x)
```



# CADENAS DE CARACTERES - STRINGS

- Las cadenas de caracteres en Python son objetos **Immutable**.

```
>>> str= "strings are immutable!"
>>> str[0]="S"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

- Se puede modificar una cadena reasignando la variable a una nueva cadena.
- En Python un carácter es una cadena de longitud uno.
- Python acepta ('), doble comillas (") and triple comillas (""" or """) para denotar cadenas.
- Los índices de las cadenas empiezan en 0 al principio de la cadena o en -1 al final.

P	y	t	h	o	n
0	1	2	3	4	5

P	y	t	h	o	n
-6	-5	-4	-3	-2	-1

# STRINGS

## ■ Operadores de cadenas de caracteres

Si asumimos que la cadena **a** contiene 'Hello' y **b** contiene 'Python'

Operator	Description	Example
+	<b>Concatenación</b> – Junta los valores a cada lado del operador	a + b resulta HelloPython
*	<b>Repetición</b> – Crea una nueva cadena concatenando múltiples copias de la misma cadena	a*2 resulta HelloHello
[ ]	<b>Slice</b> – Proporciona el carácter con el índice que se indica	a[1] devolverá e a[-1] devolverá o
[ : ]	<b>Slice de rango</b> - Proporciona los caracteres en un rango	a[1:4] devolverá ell
in	<b>Pertenencia</b> - Devuelve true si un carácter pertenece a una cadena	'H' in a devolverá True

# STRINGS

## ■ Algunas funciones de cadenas de caracteres

**str(x)**: convierte x a cadena de caracteres

**len(string)**: devuelve la longitud total de la cadena

## ■ Algunos métodos de cadenas de caracteres

Method	Description
<b>str.count</b> (sub, beg=0, end=len(str))	Counts how many times sub occurs in string or in a substring of string if starting index beg and ending index end are given.
<b>str.isalpha</b> ()	Returns True if string has at least 1 character and all characters are alphanumeric and False otherwise.
<b>str.isdigit</b> ()	Returns True if string contains only digits and False otherwise.
<b>str.lower</b> ()	Converts all uppercase letters in string to lowercase.
<b>str.upper</b> ()	Converts lowercase letters in string to uppercase.
<b>str.replace</b> (old, new)	Replaces all occurrences of old in string with new.
<b>str.split</b> (str=' ')	Splits string according to delimiter str (space if not provided) and returns list of substrings.
<b>str.strip</b> ()	Removes all leading and trailing whitespace of string.
<b>str.title</b> ()	Returns "titlecased" version of string.

# OPERACIONES CON CADENAS

La clase string tiene varios métodos que son muy útiles para dar formato a las cadenas de texto.

```
>>> "hello".upper() 'HELLO'
```

En la documentación podrás encontrar muchas más.

Nota: usa `<string>.strip()` para eliminar los saltos de línea de los archivos de texto.

# FORMATEO DE CADENAS

Para formatear cadenas se utilizan cadenas de control que empiezan por un \ y se siguen de otro símbolo.

```
>>> print ("Lenguaje: \n\tPython")
```

Lenguaje:

Python

## Cuidado con la diferencia entre:

- Retorno de carro (\r): vuelve al principio de una línea
- Nueva línea (\n): va al principio de la siguiente línea

Secuencia	Significado
\<nueva línea>	El final de línea se ignora
\\	Backslash
\'	Comilla Simple
\"	Comilla Doble
\n	Nueva línea
\r	Retorno de carro
\t	Tabulador

# FORMATO DE CADENAS CON f"{}"

```
>>> nombre = Pepe
```

```
>>> apellido = Prieto
```

```
>>> print(f"Hola {nombre} de la familia {apellido}.")
```

## CON UNA VARIABLE

```
>>> nombre = Pepe
```

```
>>> apellido = Prieto
```

```
>>> mensaje = f"Hola {nombre} de la familia {apellido}."
```

```
>>> print(mensaje)
```

# OPERADOR % PARA FORMATO DE CADENAS

- El operador % te permite construir cadenas a partir de diferentes tipos de datos con formato. Por ejemplo podemos indicar cuantos decimales pueden imprimirse o indicar cuando queremos una nueva línea.
- Muy parecido al comando printf de C.

```
>>> x = "abc"  
>>> y = 34  
>>> "%s xyz %d" % (x, y)  
'abc xyz 34'
```

- La tupla después del operador % se utiliza para llenar los espacios marcados por %s y %d.
- Debes revisar la documentación para ver otros códigos de formato.

# STR()

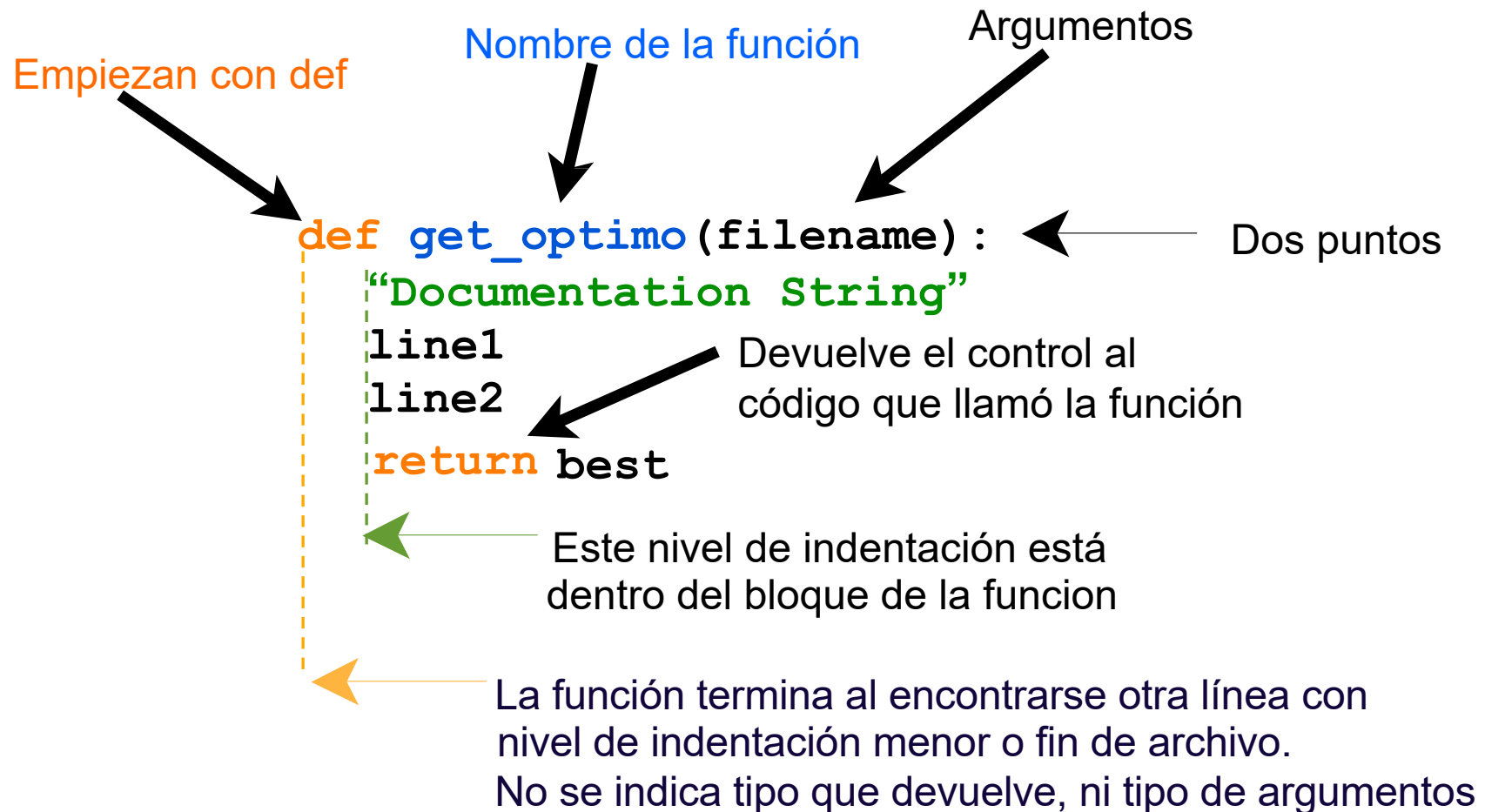
**La función built-in str() puede convertir cualquier tipo de dato a una cadena.**

Puedes definir como será este comportamiento para los tipos de datos definidos por el usuario, o redefinir el de muchos tipos.

```
>>> "Hello " + str(2)  
"Hello 2"
```



# DEFINIENDO FUNCIONES



# LLAMANDO A LAS FUNCIONES

- La sintaxis para llamar una función es:

```
>>> def myfun(x, y):  
        return x * y
```

```
>>> myfun(3, 4)  
12
```

- Los parámetros en Python se llaman “Call By-Sharing”

Los parámetros son referencias a las variables enviadas.

# PARÁMETROS \*

- Las funciones pueden recibir un número arbitrario de argumentos

```
>>> def print_args(*args):  
    print(args)
```

```
>>> print_args(3, 4, 10, 'hey')  
(3, 4, 10, 'hey')
```

- Como vemos los argumentos se reciben en tuplas.
- Si se requiere, se pueden indicar argumentos posicionales. Deben preceder a los parámetros arbitrarios.

```
>>> def print_args(pos1, pos2, *args):  
    print(pos1, pos2, args)
```

## \* EN LA LLAMADA A FUNCIONES

Al llamar funciones se puede indicar con un asterisco que la secuencia pasada debe tratarse como si fueran parámetros enviados por posición.

```
>>> def print_args(a,b,c):  
        print (a,b,c)
```

```
>>> print_args(3, 4, 10)  
(3, 4, 10)
```

```
>>> a = (3, 4, 10)
```

```
>>> print_args(*a)  
(3, 4, 10)
```

# FUNCIONES

```
#Declaracion de la funcion vacia
def suma() :

    a=input('Introduzca un dato: ')
    b=input('Introduzca otro dato: ')
    #Toca convertir los datos antes de sumarlos
    print(int(a)+int(b))

#Declaracion Funcion con argumentos

def resta(a,b) :

    print(int(a)-int(b))

#Programa y llamado de funciones

suma()
u=input('Introduzca un dato: ')
v=input('Introduzca otro dato: ')
resta(u,v)
```