

# El lenguaje python

Antonio Espín Herranz

# ¿Qué es Python?

- **Python** es un lenguaje de programación creado por **Guido van Rossum** a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”.
- Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible.
- Se trata de un **lenguaje interpretado** o de script, con **tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos**.

# Características

- **Tipado dinámico:**
  - El tipo de una variable se determina con el valor asignado.
  - No es necesario declarar las variables.
- **Lenguaje fuertemente tipado:**
  - Una vez se asignado a la variable su tipo hay que convertirla de forma explícita utilizando una función de conversión.
  - Cuando queremos imprimir números convertirlos a string.
- **Interpretado:**
  - Se genera un código intermedio de la misma forma que en java que luego se ejecuta.
  - Archivos compilados tienen extensión pyc.
- **Multiplataforma:**
  - Python se encuentra disponible para multitud de plataformas: UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc
- **Distingue entre mayúsculas y minúsculas (Case sensitive).**
- **Programación Orientada a Objetos.**
  - En Python todo son objetos.

# Más características

- **Sintaxis clara** y sencilla.
- **Elimina elementos innecesarios** de la sintaxis.
- Enfocado a **programas de alto nivel** para bajo nivel mejor utilizar C.
- **Minimalista**: todo aquello innecesario no hay que escribirlo (;, {, }, '\n')
- Muy denso: **poco código hace mucho**
- Soporta objetos y estructuras de datos de alto nivel:
  - **strings, listas, diccionarios**, etc.
- Múltiples niveles de organizar código: **funciones, clases, módulos, y paquetes**.

# Sintaxis

- Utiliza la tabulación para determinar el inicio y fin de un bloque:

Código en C/Java	Código en Python
<pre>if (x) {     if (y) {         f1();     }     f2(); }</pre>	<pre>if x:     if y:         f1()     f2()</pre>

# Cuando **NO** usar python

- **Python no es el lenguaje perfecto, no es bueno para:**
  - Programación de bajo nivel (system-programming), como programación de drivers y kernels
  - Python es de demasiado alto nivel, no hay control directo sobre memoria y otras tareas de bajo nivel.
  - Aplicaciones que requieren alta capacidad de computo
  - No hay nada mejor para este tipo de aplicaciones que el viejo C
- **Python es ideal:**
  - Como lenguaje "pegamento" para combinar varios componentes juntos
  - Para llevar a cabo prototipos de sistema
  - Para la elaboración de aplicaciones cliente
  - Para desarrollo web y de sistemas distribuidos
  - Para el desarrollo de tareas científicas, en los que hay que simular y prototipar rápidamente

# Implementaciones de Python

- Existen varias implementaciones de Python:
- **CPython:**
  - Es la más utilizada, la más rápida y la más madura. Cuando la gente habla de Python normalmente se refiere a esta implementación. En este caso tanto el intérprete como los módulos están escritos en C.
- **Jython:**
  - Es la implementación en Java de Python, mientras que **IronPython** es su contrapartida en C# (.NET). Su interés estriba en que utilizando estas implementaciones se pueden utilizar todas las librerías disponibles para los programadores de Java y .NET.
- **PyPy:**
  - Se trata de una implementación en Python de Python.

# Instalar Python 3 en Windows

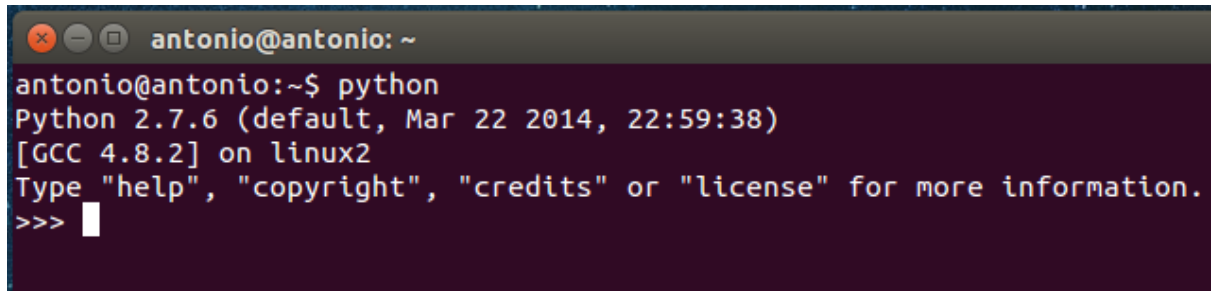
- **En Windows:**
  - Podemos descargar el instalador de:
    - <https://www.python.org/downloads/>
    - Actualmente vigentes la versión 2.x y la 3.x
    - Seleccionar versión 32 o 64 Bits.
  - La instalación se realiza en:
    - C:\Users\<<usuario>>\AppData\Local\Programs\Python\Python35





# Instalar Python en Linux

- **En Linux / Unix:** suele venir ya instalada. Para comprobarlo abrir un terminal y teclear **python**.
- Si fuera necesario instalarlo en Ubuntu:

A terminal window with a dark purple background. The title bar shows 'antonio@antonio: ~'. The terminal text shows the command 'python' being executed, resulting in 'Python 2.7.6 (default, Mar 22 2014, 22:59:38) [GCC 4.8.2] on linux2'. It then prompts the user to type 'help', 'copyright', 'credits', or 'license' for more information, followed by a prompt '>>>' and a cursor.

```
antonio@antonio: ~  
antonio@antonio:~$ python  
Python 2.7.6 (default, Mar 22 2014, 22:59:38)  
[GCC 4.8.2] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 
```

- Escribir el siguiente comando para **instalar python en linux** en la versión 3.x:

**\$ sudo apt-get install python3.5**

# Ejecutar código en python

- Desde **Windows**:

- Se puede ejecutar en modo consola o editar el fichero con el **IDLE** (editor de python).
- Dentro del editor con F5 ejecuta el fichero.

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: L:\antonio\TRABAJO\CURSOS\PYTHON\python 3.x\pruebas\hello_world.py
hello world
>>>
```

- Desde **Linux**:

- Se puede utilizar en modo comando desde el terminal:

```
>>> 5+7
12
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> █
```

# Ejecutar código en python II

- **Ejecución de programas:**
  - Desde **Windows**:
    - En **consola**: *python fichero.py*
  - Desde **Linux** (igual):
    - En **consola**: *python fichero.py*
- **Editores** para python:
  - La distribución de python viene con su propio editor.
  - **Geany**
  - Eclipse

# **Primer programa en Python**

# Ejemplo

- **Programa.py**  
# primer programa en python  
print("hola mundo")  
input()
- Las sentencias se escribe una por línea, sin punto y coma pero si queremos escribir varias sentencias en la misma línea se separan por un ;
  - print("hola"); print(4)
- Resultado:
  - Imprime y espera a que pulsemos una tecla.
- **Comentarios:**  
# Línea comentada, precedida de una #  
  
"""  
Comentario en Bloque. Triple comilla  
"""
- **Las sentencias acaban en nueva línea, no en ;**
- Los **bloques** son **indicados por tabulación** que sigue a una sentencia acabada en ':'. Después de un if.

# Variables

- No hace falta declararlas ni asignarlas el tipo.
- `pi = 3.141516`
- Una vez que las hemos dado valor (equivalente a declararlas) se pueden utilizar:
- `B = 2 * pi.`
- `print(B)` # Se pueden imprimir tal cual.
- Se pueden evaluar polinomios dando valor a la variable `x`:

$$x^{**4} + x^{**3} + 2 * x^{**2} - x.$$

Si las variables no se inicializan da error cuando las utilizamos.  
**Al inicializarlas se crean!!**

# Palabras reservadas

- Son palabras que tienen un significado especial dentro del lenguaje.
- and, 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, yield.

# Tipos básicos



# Tipos

- Tipos de datos numéricos: **int**, **float** y **complex**.
- Las operaciones se adjuntan al tipo de los operandos. Si uno de los dos es real, el resultado será real.
- Enteros: 22, 102, 123, etc.
- Reales: **30.03**, **2e3**  $\Rightarrow 2 \cdot 10^3$ ,  $2 \cdot 10^{-3}$
- Complejos =  $2.1 + 7.8j$

# Constructores de tipos

- Tenemos un constructor para cada tipo:
  - Enteros: **int()**
  - Float: **float()**
  - Complejos: **complex()**
- Se pueden utilizar para hacer conversiones cuando leemos de teclado:  
X = int(input())  
Y = float(input())  
Z = complex(input())

# Tipos

- Los enteros no tienen límite de tamaño.
- El límite de los número viene marcado por la memoria de la máquina.
- Podemos realizar este tipo de operaciones:  
X = 34  
Y = 1000  
print(X \*\* Y)

# Tipos

- El literal que se asigna a la variable también se puede expresar como un **octal**, anteponiendo un cero:

#027 octal = 23 en base 10

entero = **0o27**

- O bien en hexadecimal, anteponiendo un 0x:

# 0x17 **hexadecimal** = 23 en base 10

entero = **0x17**

# Formatear

- También disponemos de los constructores: `bin()`, `oct()` y `hex()` para cambiar de formato un número a binario, octal y hexadecimal.

```
>>> x = 1234
```

```
>>> bin(x) '0b10011010010'
```

```
>>> oct(x) '0o2322'
```

```
>>> hex(x) '0x4d2'
```

```
>>>
```

# Formatear

- Alternativamente, podemos utilizar la función `format`: para formatear la salida a consola:

```
>>> format(x, 'b') '10011010010'
```

```
>>> format(x, 'o') '2322'
```

```
>>> format(x, 'x') '4d2'
```

- También con los constructores anteriores: `int()`
- Se puede indicar la base a la que convertir el número:  

```
print(int('4d2', 16))  
print(int('10011010010', 2))
```

# Tipos

- **Reales** representados por **float**.
- Internamente se almacena en el tipo **double de C**.

```
ff = 5.678
```

```
type(ff)
```

```
float
```

```
>>> ff = 5.777
>>> type(ff)
<type 'float'>
>>> cc = 2.1 + 7j
>>> type(cc)
<type 'complex'>
>>>
```

- Tipo **complejo** representado por **complex**.
  - Uso científico y matemático.

# Tipos

- **Lógicos:** Tipo **bool** . Valores: **True** / **False**.
- Tipo **None**, para indicar nada ..., si algo es None y preguntamos en un if será False.
- **Cadenas** (representadas por el tipo **str**)
  - Las cadenas no son más que texto encerrado entre comillas simples ('cadena') o dobles ("cadena"). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con \, como \n, el carácter de nueva línea, o \t, el de tabulación.
  - Los caracteres especiales "\\..." se interpretan igual en cadenas con comillas simples y dobles.



# Tipos

- Una cadena puede estar precedida por el carácter **u** o el carácter **r**, los cuales indican, respectivamente, que se trata de una cadena que utiliza codificación **Unicode** y una cadena **raw** (*del inglés, cruda*).
- Las cadenas raw se distinguen de las normales en que los caracteres escapados mediante la barra invertida (\) no se sustituyen por sus contrapartidas. Esto es especialmente útil, por ejemplo, para las expresiones regulares.
- **En python 3 las cadenas son Unicode y no es necesario poner una u delante de la cadena.**
- Ejemplo:
  - `unicode = "äóè"`
  - `raw = r"\n"`

# Tipos

- Las cadenas permiten comillas triples.
- En este caso se mantienen los saltos de línea:  
triple = """primera linea  
esto se vera en otra linea"""
- Soportan el operador + y \*
  - a = “uno”
  - b = “dos”
  - c = a + b # c es “unodos”
  - c = a \* 3 # c es “unounouno”

# Operadores

- Operadores **aritméticos**:

Operador	Descripción	Ejemplo
+	Suma	<code>r = 3 + 2</code> # r es 5
-	Resta	<code>r = 4 - 7</code> # r es -3
-	Negación	<code>r = -7</code> # r es -7
*	Multiplicación	<code>r = 2 * 6</code> # r es 12
**	Exponente	<code>r = 2 ** 6</code> # r es 64
/	División	<code>r = 3.5 / 2</code> # r es 1.75
//	División entera	<code>r = 3.5 // 2</code> # r es 1.0
%	Módulo	<code>r = 7 % 2</code> # r es 1

# Operadores

- A nivel de bits:

Operador	Descripción	Ejemplo
&	and	<code>r = 3 &amp; 2 # r es 2</code>
	or	<code>r = 3   2 # r es 3</code>
^	xor	<code>r = 3 ^ 2 # r es 1</code>
~	not	<code>r = ~3 # r es -4</code>
<<	Desplazamiento izq.	<code>r = 3 &lt;&lt; 1 # r es 6</code>
>>	Desplazamiento der.	<code>r = 3 &gt;&gt; 1 # r es 1</code>

# Operadores sobre bits

- & and sobre bits.
- | or sobre bits.
- ^ xor sobre bits.
- ~ complemento a 1. Cambia ceros por unos.
- >>: rotación a la derecha. Equivalente a dividir por  $2^n$ . Siendo n el número de bits a rotar.
- <<: rotación a la izquierda. Igual que la anterior pero multiplicando.

# Operadores

- Lógicos:

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	<code>r = True and False # r es False</code>
or	¿se cumple a o b?	<code>r = True or False # r es True</code>
not	No a	<code>r = not True # r es False</code>

# Operadores

- Relacionales:

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
!=	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<	¿es a menor que b?	<code>r = 5 &lt; 3 # r es False</code>
>	¿es a mayor que b?	<code>r = 5 &gt; 3 # r es True</code>
<=	¿es a menor o igual que b?	<code>r = 5 &lt;= 5 # r es True</code>
>=	¿es a mayor o igual que b?	<code>r = 5 &gt;= 3 # r es True</code>