

# Introducción a Python

Objetivos:

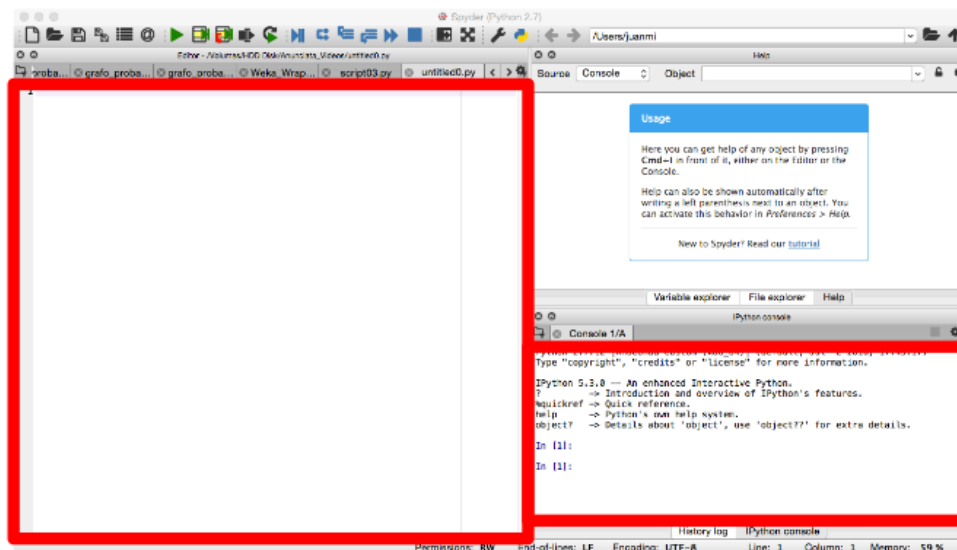
- Entorno Anaconda Spyder: Espacio de trabajo, Ventana de Comandos, Editor.
- Variables, Asignaciones.
- Operadores básicos, tipos de datos y funciones
- Scripts o programas

## Entorno Anaconda Spyder

Para programar en Python vamos a usar la distribución Anaconda<sup>1</sup>, empleada principalmente en los ámbito de ciencias de datos y aprendizaje automático.

**Nota importante:** En los ordenadores del laboratorio emplearemos Anaconda con Python 3.

Después de **entrar** en **Linux** (nombre y contraseña LDAP), arranca Anaconda desde una terminal tecleando **spyder**. Después del arranque del entorno de trabajo integrado (puede tardar unos segundos), se verá el siguiente interfaz:



**Consola IPython:** es la ventana principal del entorno de trabajo de Anaconda. Es un intérprete de lenguaje Python. Se usa para introducir datos y ejecutar funciones o programas. Lo veremos más adelante. Los comandos se escriben a la

<sup>1</sup><https://www.anaconda.com/products/individual>

derecha del solicitador de comando o *prompt* “In [1]”. Por ejemplo, si queremos saber el resultado de una suma:

a+b, si a=10 y b=23

escribimos lo siguiente a la derecha del *prompt* en la ventana de comandos:

```
>> a = 10
```

```
>> b = 23
```

```
>> a + b
```

dará como respuesta:

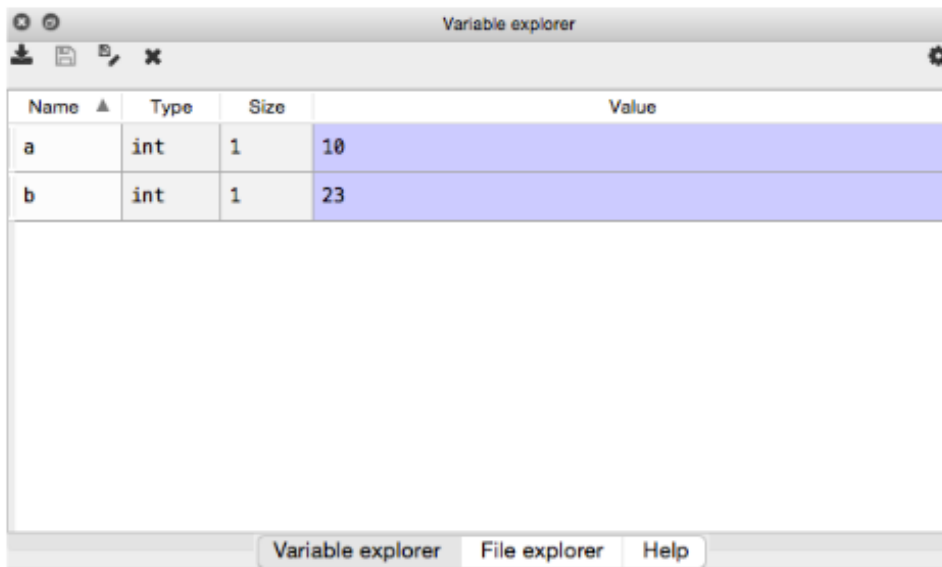
```
Out [3] = 33
```

**Nota:** Para ejecutar un comando, pulsa Intro.

**History log (historial de comandos):** registra todos los comandos escritos en la ventana de comandos. Nos permite saber qué comandos se han ejecutado previamente.

A screenshot of a window titled 'history.py' showing a list of commands entered in the IPython console. The commands include importing time, sleeping for 5 seconds, and some Selenium WebDriver code. At the bottom, the commands 'a = 10', 'b = 23', and 'a + b' are listed. The window has a 'History log' button at the bottom.

**Variable Explorer:** contiene las variables creadas desde que se ha abierto la sesión en Anaconda.

A screenshot of the 'Variable explorer' window in Anaconda. It shows a table with columns 'Name', 'Type', 'Size', and 'Value'. Two variables are listed: 'a' with type 'int' and size 1, and 'b' with type 'int' and size 1. The values are 10 and 23 respectively. The window has a toolbar at the top and tabs at the bottom for 'Variable explorer', 'File explorer', and 'Help'.

Name ▲	Type	Size	Value
a	int	1	10
b	int	1	23

Las variables creadas se pueden listar con el comando *who*. Además, podemos usar el comando *whos* para ver también los valores y dimensiones de cada variable.

Si escribimos el comando *reset* borraremos todas las variables. Para borrar una variable, se emplea el comando *del*, seguido del nombre de la variable.

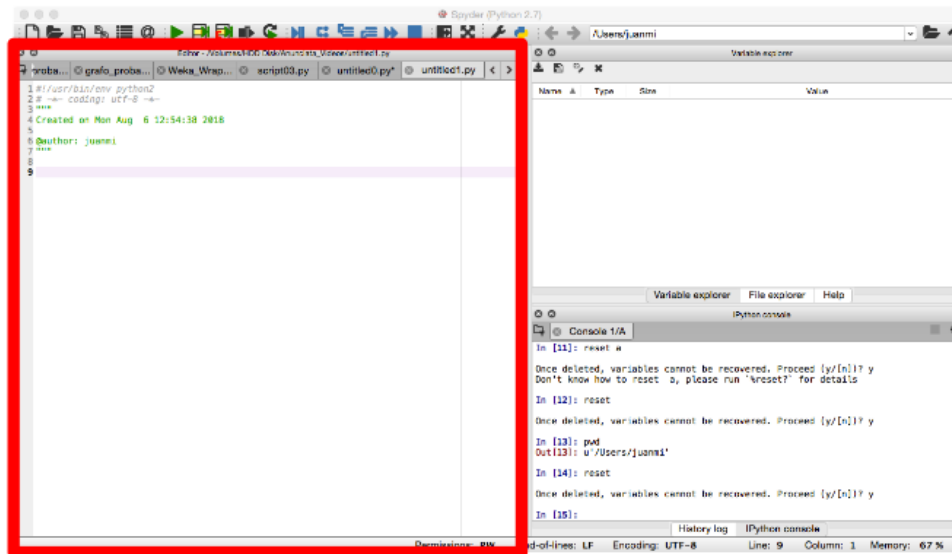
**Grabar los datos a fichero:** Las variables y datos las podemos grabar a un fichero con extensión *.spydata* para usarlas posteriormente, pulsando el botón izquierdo sobre “Save data as...” (Guardar como) en Variable Explorer.

**File Explorer:** es la ventana que muestra los ficheros del directorio de trabajo.

También podemos saber el directorio actual con el comando *pwd* (print working directory).

Para que Python encuentre los ficheros que te interesan, debes seleccionar ese directorio como el de trabajo o incluirlo en los caminos de búsqueda (el path).

**El Editor:** se usa para crear programas o scripts. Pulsa el botón “New File” (o File > New File) en la barra para que se muestre la ventana del editor.



## Variables y asignaciones

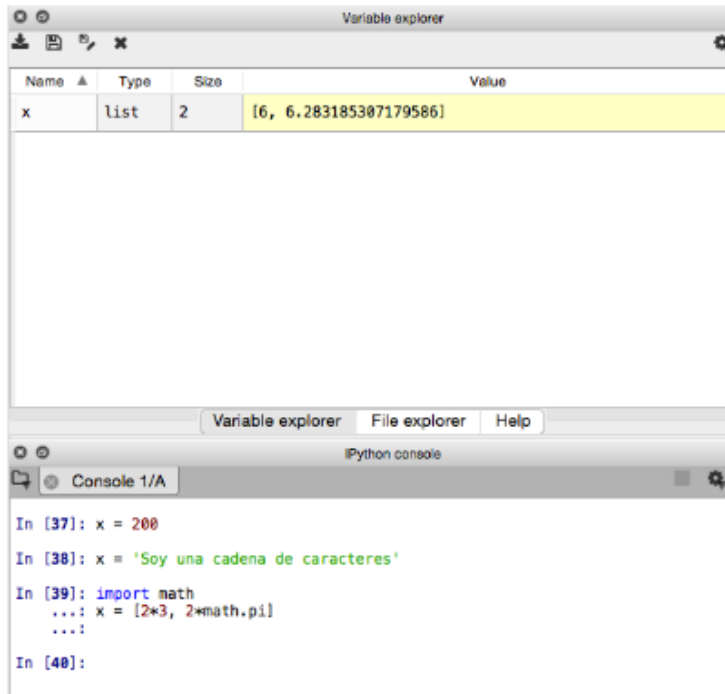
### Variables

Las variables almacenan datos (en este curso principalmente matrices), y se definen con el operador de asignación: “=”.

El tipo de datos de las variables en Python depende del valor que se les asigne, sin declarar su tipo (sea número entero, real, matriz, texto, etc.). El tipo puede variar dinámicamente si a una variable se le asigna un valor de otro tipo de datos diferente.

Cuando se le da nombre a una variable, es conveniente usar convenciones para especificar el contenido y su objetivo en el programa tanto para nuestro uso como para cualquier persona que vaya a examinar posteriormente el programa. Se puede consul-

tar una guía de convenciones en: <https://www.python.org/dev/peps/pep-0008/>



### Notas:

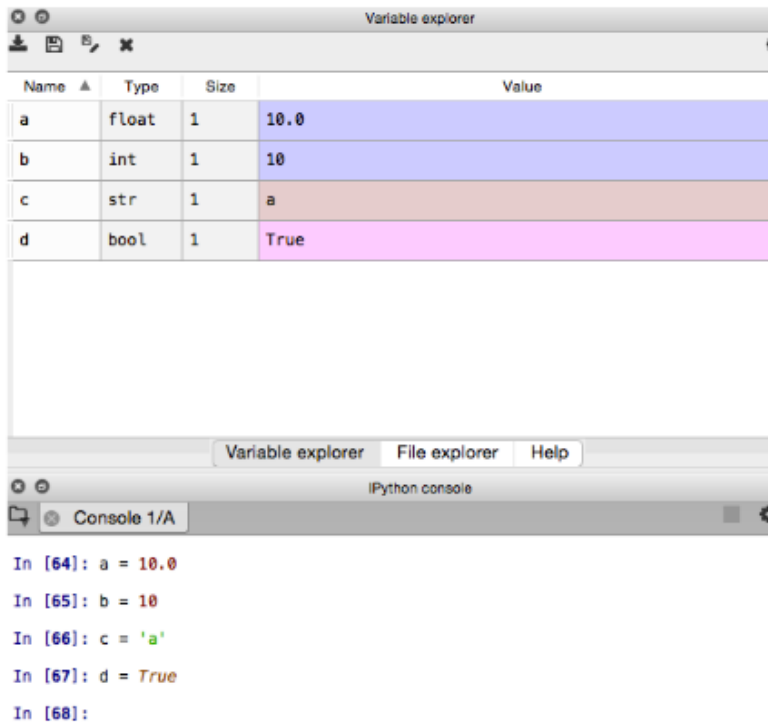
- Python distingue entre mayúsculas y minúsculas. Por ello, las variables x y X son distintas.
- Se pueden escribir varios comandos o instrucciones en una misma línea separándolos con punto coma (;).

## Operadores básicos, tipos de datos y funciones

Hay varios tipos básicos para las variables: números enteros, números de coma flotante, números complejos, cadenas de caracteres y booleanos.

Operaciones aritméticas básicas que incluye Python: +, -, \*, /, % (resto división entera), \*\* (Exponencial).

El siguiente ejemplo muestra cuatro variables con diferentes tipos de datos.



Python también incluye una serie de funciones matemáticas habituales que se pueden usar en comandos y programas. Están incluidas en el paquete *math* (salvo *abs()*), que debe ser importado para poder emplearlas (`import math`):

Expresión matemática	Expresión en Python
logaritmo neperiano	<code>log()</code>
logaritmo en base 10	<code>log10()</code>
raíz cuadrada	<code>sqrt()</code>
$e^x$	<code>exp(x)</code>
valor absoluto	<code>abs()</code>

### Notas:

- Se pueden añadir comentarios en el código con el símbolo “#”. Por ejemplo, “Número de elementos” en la siguiente línea es un comentario:

```
# Número de elementos:
a = 3
```

- *type()* se emplea para saber el tipo de datos de una variable (`type(nombre_variable)`).

## Scripts

Tal y como se ha mencionado anteriormente, el Editor se usa para crear programas o scripts.

## Crear nuevo script

Pulsa el botón “New File” (o File > New File) en la barra para que se muestre la ventana del editor.

Antes de empezar a escribir código, en caso de que sea necesario, se ha de emplear la instrucción *import* para importar las clases y los métodos Python que necesitemos.

En dicho script se escribirán las líneas de código en Python para solucionar cada problema planteado durante el curso.

## Grabar

Se pueden guardar los programas (scripts) en el disco duro con la opción de menú “File > Save as”, que pedirá el nombre del fichero en el que grabar el código. Como es habitual en este tipo de entornos, también podremos abrir un programa previamente guardado con la opción “File > Open”.

**Nota importante:** En caso de querer ejecutar sólo parte del script en el Editor, se puede **seleccionar** la parte en cuestión y mediante la combinación **Ctrl + Enter** se ejecutará en el intérprete. En caso de querer ejecutar el **script entero**, se usará el botón **Run File** (tecla **F5**).

## Código Python

### Listas y listas de listas (arrays y matrices)

Una lista es una colección de elementos ordenada e intercambiable en Python que permite la existencia de elementos duplicados.

El constructor para crear una lista es *list()*. Para añadir un elemento se emplea *append()*. Para eliminar un elemento en concreto, *remove()*. *len()* devuelve la longitud de la lista.

El acceso a los elementos de una lista se puede hacer mediante un índice *i*, donde *i* representa la posición del elemento de 0 a la longitud de la lista−1.

A la hora de recorrer una lista, éste se puede efectuar mediante su rango, empleando la función *range(inicio, fin)* especificando como inicio 0 y como final la longitud de la lista. También puede recorrerse mediante un iterador en la forma: *for elem in elems;* donde *elems* es el nombre de la lista y *elem* corresponde a cada elemento desde el inicio hasta el final cada vez que se itera.

El operador *in* devuelve si un elemento se encuentra dentro de la lista o no. Por ejemplo: dado *a = [1, 2, 3, 4]*, *3 in a* devolverá **True**.

Respecto a las matrices, la implementación básica de Python se realiza mediante lista de listas. Es decir, se emplea una lista en la que cada posición corresponde a su vez a otra lista con los valores. Por ejemplo:

```
a = [['Alicia',80,75,85,90,95],['Carolina',80,80,80,90,95]]
```

A la hora de iterar se pueden emplear índices (i, j) para acceder a posiciones específicas, o bien emplear iteradores para realizar la misma función.

## Instrucciones básicas

### Assert

Instrucción empleada para verificar que una condición se cumple en un momento determinado de la ejecución de un programa.

Ejemplo de uso:

```
assert a == b
```

En caso de que la condición sea correcta, la ejecución del programa sigue con normalidad, mientras que en caso de ser incorrecta el programa parará dando un error de aserción “AssertionError”.

Esta instrucción se emplea principalmente para encontrar errores (testing).

### Condiciones (if)

Evalúa una expresión booleana, para ejecutar una serie de sentencias en función de que dicha expresión sea cierta.

La cláusula **Else** sirve para ejecutar una serie de sentencias en el caso de que la expresión devuelva False.

### Operadores lógicos

Empleados habitualmente en condiciones son:

and, or, not, >, <, >=, <=, ==, !=

### Iteraciones (for/while)

For con rango o iterador.

### Nota importante:

En Python existe la estructura denominada “List Comprehension”, que sirve para crear listas de una manera concisa. Es especialmente útil si se quiere crear otra lista en base a operaciones sobre valores de una lista original mediante operaciones con iteraciones. Por ejemplo, suponiendo declarada una lista `old_list` y definidas las funciones `filter()` y `expression()`, si se quiere realizar la siguiente acción:

```
new_list = [ ]  
for elem in old_list:
```

```

if filter(elem):
    new_list.append(expression(elem))

```

podremos hacerlo usando la estructura “List Comprehension” de la siguiente forma:

```
new_list = [expression(elem) for elem in old_list if filter(elem)]
```

La sintaxis básica es la siguiente:

```
[ expression for item in list if conditional ]
```

Otro ejemplo:

```

x = [i for i in range(10)]
print(x)

```

Escribiré la lista [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

## Tabulación

El lenguaje de programación Python obliga a tabular las líneas de código que formen parte del mismo bloque de código. Es la propia tabulación la que determina el inicio y el fin de cada bloque de código. Las instrucciones que pueden tener bloques de código incluyen: **if**, **for**, **while**, definiciones de funciones (**def**). Todas ellas terminan en : para indicar que a continuación viene un bloque de código.

## Mensajes por pantalla

Para sacar un mensaje por pantalla se emplea el comando print. Dicho comando puede emplearse para mostrar por pantalla valores de variables, indicando mediante % el tipo de la variable (%d para números enteros) en el string de salida y mediante % seguido de las variables, los valores que deben ir en cada hueco reservado en el string. En el siguiente caso se muestra por pantalla el resultado de la suma entre las variables a y b, guardado en c:

```

In [19]: a = 10
...: b = 10
...: c = a + b
...: print('%d + %d = %d' % (a, b, c))
...:
10 + 10 = 20

```

Es recomendable usarlo sólo en programas principales y no para buscar errores (para buscar errores de código es recomendable emplear el debugger).

## Funciones

Hay cuatro pasos a la hora de definir una función de usuario en Python:

1. Emplear la palabra reservada **def** para declarar una función, seguida del nombre de la función.



2. Añadir parámetros a la función. Deben ir entre paréntesis. Finalizar la línea con dos puntos (:).
3. Añadir los comandos que la función debe ejecutar.
4. Finalizar la función con el comando **return**. En caso de no hacerlo, la función devolverá un objeto *None*.

En el siguiente ejemplo se define y invoca una función que calcula la suma de tres valores numéricos:

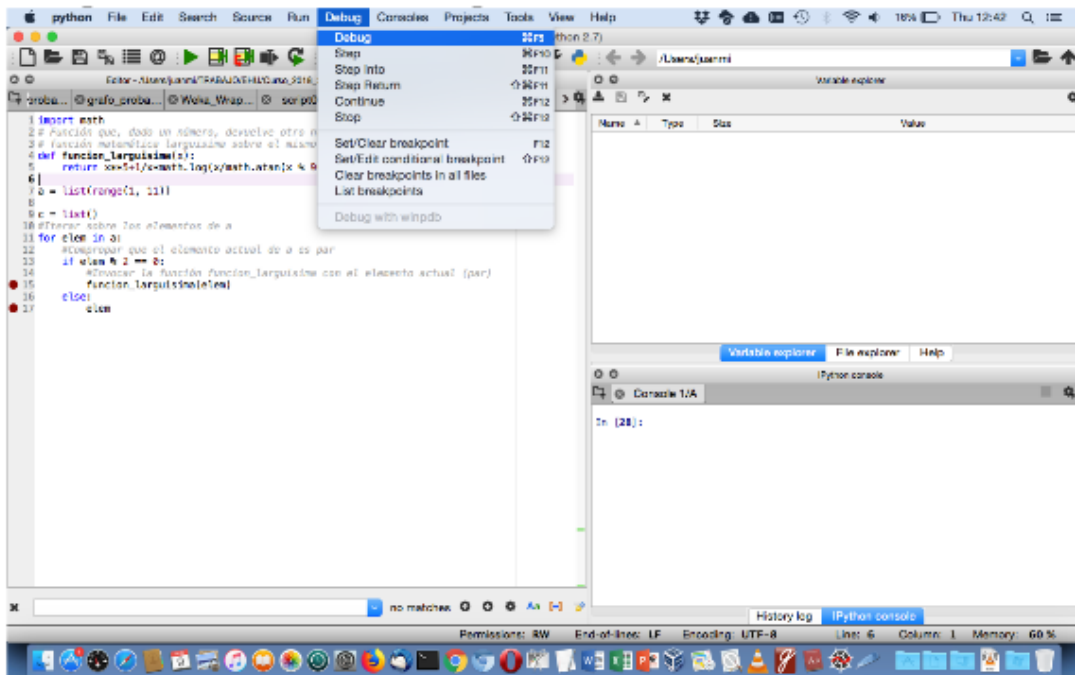
```
#Función que devuelve la suma de los 3 parámetros
def suma_3_valores(a, b, c):
    return a + b + c

v1, v2, v3 = 10, 20, 23

#Llamada a la función para calcular la suma de los 3 valores
result = suma_3_valores(v1, v2, v3)
print('%d + %d + %d= %d' % (v1, v2, v3, result))
```

## Debugger

El debugger integrado en Spyder permite depurar errores en el código parando la ejecución del mismo en las líneas que consideremos oportunas. En el siguiente código se ha detectado que *c* es una lista vacía, con lo que se introducen dos puntos de interrupción donde se tendrían que añadir valores a dicha lista.



Una vez añadidos, dándole a Debug > Debug, el debugger se pondrá en marcha, ejecutará el programa y lo parará en la línea correspondiente al primer punto

de ruptura (breakpoint) definido en el programa. Para moverse por la ejecución del programa se puede bien ejecutar la línea de código actual (Ctrl + F10), entrar dentro del método, si lo hubiera (Ctrl + F11), ejecutar el método, si lo hubiera, sin entrar en él (Ctrl + Shift + F11) o bien saltar hasta el siguiente punto de interrupción (Ctrl + F12).

En el siguiente pantallazo se muestra cómo hemos llegado hasta el elemento con valor 4 de la lista a, pero no hay valores en c, con lo que se determina que el problema es que los valores no se añaden a la lista, a pesar de ser calculados correctamente. La línea marcada en morado en el Editor es la actual en la ejecución del código, también marcada en la consola Python, mientras que en el Variable Explorer se ven los valores de las variables en el programa. El valor de c no aparece por estar la lista vacía.

