

Introducción a



pythonTM

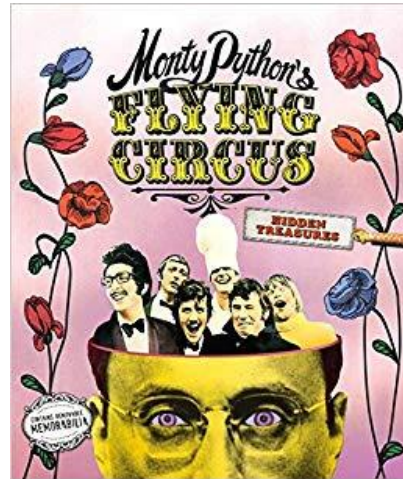
Indice

- Historia de Python
- Características principales
- Diferentes versiones
- Instalación de Python
- Tipos de datos
- Estructura de control de flujos



Historia de Python

- Creado por Guido van Rossum
- El lenguaje fue creado como sucesor del lenguaje de programación ABC en 1991
- Recibió este nombre gracias al grupo cómico "Monty **Python**'s Flying Circus"



Historia de Python

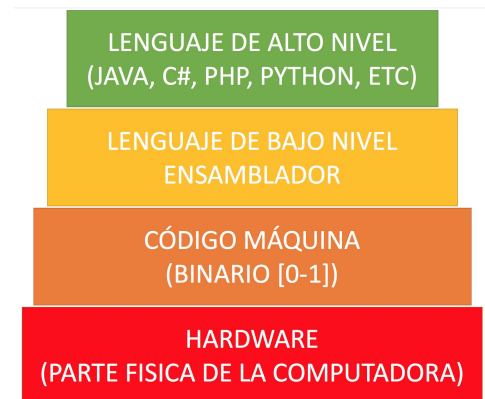
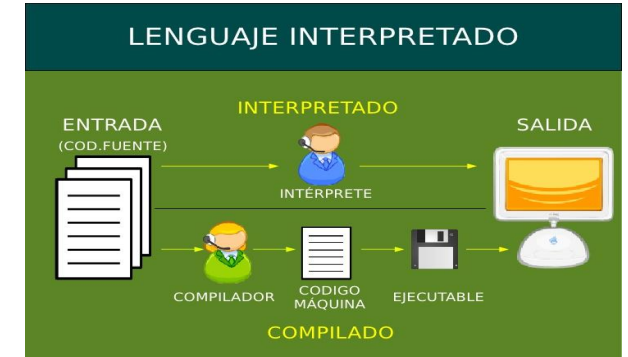
- Principales usos
 - Sistemas
 - Desarrollo web
 - Data science

Mar 2021	Mar 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	15.33%	-1.00%
2	1	▼	Java	10.45%	-7.33%
3	3		Python	10.31%	+0.20%
4	4		C++	6.52%	-0.27%
5	5		C#	4.97%	-0.35%
6	6		Visual Basic	4.85%	-0.40%
7	7		JavaScript	2.11%	+0.06%



Características

- Lenguaje interpretado
- Tipado dinámico
- Multiplataforma
- Multiparadigma (estructurado, orientado a objetos, funcional, ...)
- Código legible
- Versiones:
 - Python 2.7, Python 3.5. Actualmente 3.9



The Zen of Python

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

...

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.



PEP-8 Estilo de codificación

- Indentación: 4 espacios
- Se deben separar las funciones de nivel superior y las clases con dos líneas en blanco, mientras que los métodos dentro de clases los podemos separar con una sola línea.
- Tamaño de línea: Máximo 79 caracteres. Continuar con la línea con el símbolo “\” o incluir paréntesis:

```
Salida = campo1 + campo2 + campo3 \
      + campo4 + campo5
```

```
Salida = ( campo1 + campo2 + campo3
          + campo4 + campo5)
```



PEP-8 Estilo de codificación

- Las sentencias de import deben de estar generalmente separadas una en cada línea, por ejemplo:



```
import os, sys
import os
import sys
from urllib2 import urlopen, Request
```

- Usar espacios alrededor de los operadores aritméticos
- No usar espacios alrededor del signo igual cuando se encuentre en un listado de argumentos de una función

```
def suma(a = 0, b = 0):
def suma(a=0, b=0):
```



- No se deben comparar booleanos mediante ==:

```
if valido == True:
    pass
if valido:
    pass
```



PEP-8 Estilo de codificación

- Espacio después de “,” pero no después ni antes de (), [], {}.
- Comentarios:
 - De una línea: `# Esto es un comentario`
 - De varias líneas: `""" Esto es un comentario
De varias líneas """`
- Nombres:
 - Clases debe empezar por mayúscula `Persona`, `GeneradorViento`
 - Funciones en minúscula separados por `_`: `calcula_varianza()`
 - Constantes en mayúsculas separadas por `_`: `EARTH_RADIUS`



PEP-257 Estilo de documentación

- La especificación de Python tiene reservada la primera línea después de la definición de una función o clase para añadir la cadena de documentación.
- Se utilizan las triples comillas para indicar este tipo de comentarios

```
def suma(a, b):  
    """  
    Esta funcion va a calcular la suma de los dos valores recibidos  
    Argumentos:  
    a -- primer numero del sumatorio  
    b -- Segundo numero del sumatorio  
    """  
    return a + b
```

- Luego si indicamos: `help(nombre_funcion)` nos muestra el mensaje



Volcando información por pantalla: print

- Una de las funciones que más utilizaremos en Python es la función print, podemos imprimir no sólo cadenas sino cualquier tipo de dato

```
>>> print("Hola")
Hola
>>> print("Hola")
Hola
>>> print(['Hola', 'Adios'])
['Hola', 'Adios']
>>> print(", ".join(['Hola', 'Adios']))
Hola,Adios
>>> print("Yo me llamo %s y tengo %d años, mido %.2f" % ("Patricia", 34, 1.60))
Yo me llamo Patricia y tengo 34 años, mido 1,60
```

- Los programas que escribimos normalmente hacen uso de dos tipos de salidas por pantalla, la salida estándar y la salida de error.
- Print por defecto va a imprimir en la salida estándar, podemos también forzar a que vuelque por la salida de error

```
>>> print >> sys.stdout, "Esto va por la salida estandar"
>>> print >> sys.stderr, "Esto va por la salida de error"
```



El intérprete de Python

MAC:

- En los sistemas operativos Linux está instalado por defecto

```
sesion_python % python
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Aún así, usar la versión de la plataforma no es lo recomendado, es necesario instalar Python 3.8 desde la web

Windows:

- <https://www.python.org/downloads/>



Instalación intérprete de Python Windows




The image shows the Python.org homepage. At the top, there is a navigation bar with links to Python, PSF, Docs, PyPI, Jobs, and Community. Below this is a search bar and a 'Socialize' button. The main content area features the Python logo and a 'Download the latest version for Windows' section. A yellow button labeled 'Download Python 3.7.2' is prominent. Below this, there are links for other operating systems and pre-releases. A large illustration of two parachutes with boxes hanging from them is also visible.


Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.6.4	2017-12-19	Download	Release Notes
Python 3.6.3	2017-10-03	Download	Release Notes
Python 3.3.7	2017-09-19	Download	Release Notes
Python 2.7.14	2017-09-16	Download	Release Notes
Python 3.4.7	2017-08-09	Download	Release Notes
Python 3.5.4	2017-08-08	Download	Release Notes
Python 3.6.2	2017-07-17	Download	Release Notes

[View older releases](#)

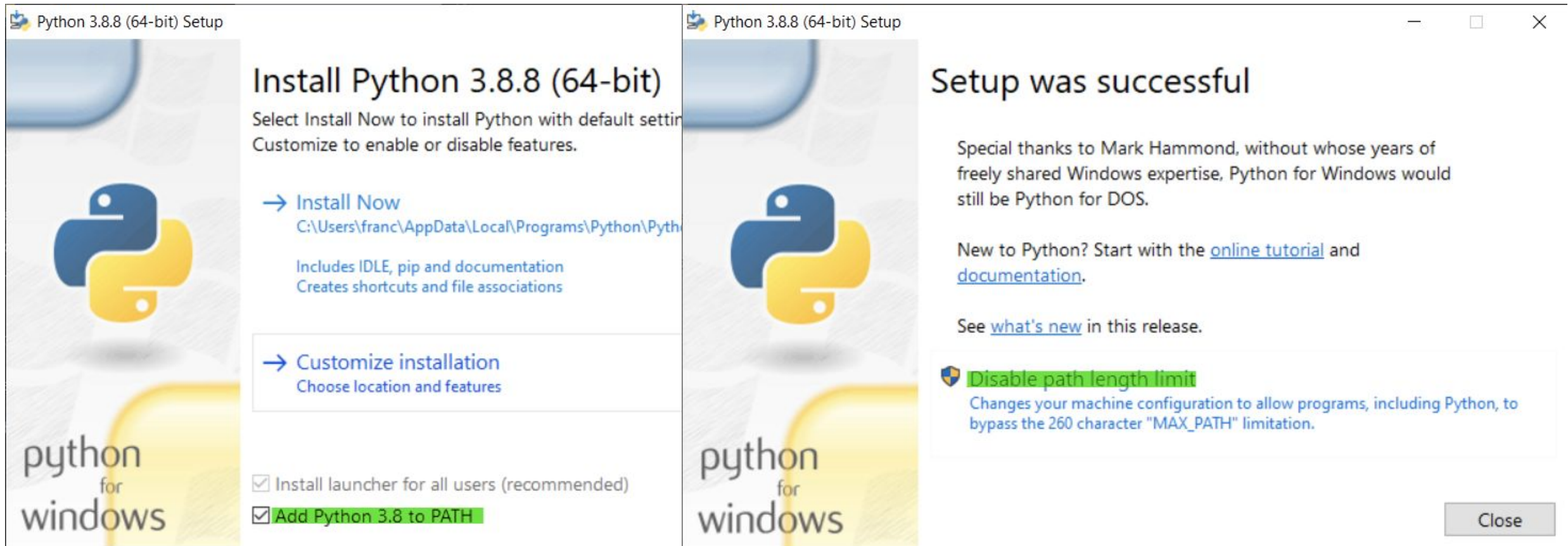
 python-2.7.14.amd64



The image shows the 'Python 2.7.9 Setup' window. It asks the user to 'Select whether to install Python 2.7.9 for all users of this computer.' There are two radio buttons: 'Install for all users' (selected) and 'Install just for me (not available on Windows Vista)'. The Python logo is on the left. At the bottom, there are 'Back', 'Next >', and 'Cancel' buttons. An orange arrow points to the 'Next >' button.



Instalación Python en Windows



Instalación intérprete de Python Mac

Python 3.8.8

Feb. 19, 2021

 Download

[Release Notes](#)

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		d3af3b87e134c01c7f054205703adda2	24483485	SIG
XZ compressed source tarball	Source release		23e6b769857233c1ac07b6be7442eff4	18271736	SIG
macOS 64-bit Intel installer	Mac OS X	for macOS 10.9 and later	3b039200febdd1fa54a8d724dee732bc	29819402	SIG
Windows embeddable package (32-bit)	Windows		b3e271ee4fafce0ba784bd1b84c253ae	7332875	SIG
Windows embeddable package (64-bit)	Windows		2096fb5e665c6d2e746da7ff5f31d5db	8193305	SIG
Windows help file	Windows		d30810feed2382840ad1fbc9fce97002	8592431	SIG
Windows installer (32-bit)	Windows		94773b062cc8da66e37ea8ba323eb56a	27141264	SIG
Windows installer (64-bit)	Windows	Recommended	77a54a14239b6d7d0dcbe2e3a507d2f0	28217976	SIG



Ejecución intérprete de Python

Mac

En aplicaciones tendremos un IDLE que podremos hacer click.

O desde la consola:

```
/Library/Frameworks/Python.framework/Versions/3.8/bin/python3
```

Windows

Desde CMD:

```
python.exe
```



Ejercicios Intérprete Python

Una vez instalado Python:

1. Ejecutar el intérprete
2. Imprimir por pantalla “Hola Mundo”
3. Incluir comentarios al código de python
4. Suma de 4 y 7 y el producto de 225 y 6
5. Ejecuta la sentencia “import this”



Solución Interprete Python

```
>>> 'hola mundo'
hola mundo
```

```
>>> 'hoy es viernes' #Ejercicio 2
hoy es viernes
```

```
>>> 4+7
11
```

```
>>> 225*6
1350
```

```
>>> import this
The Zen of Python, by Tim Peters
.....
.....
```



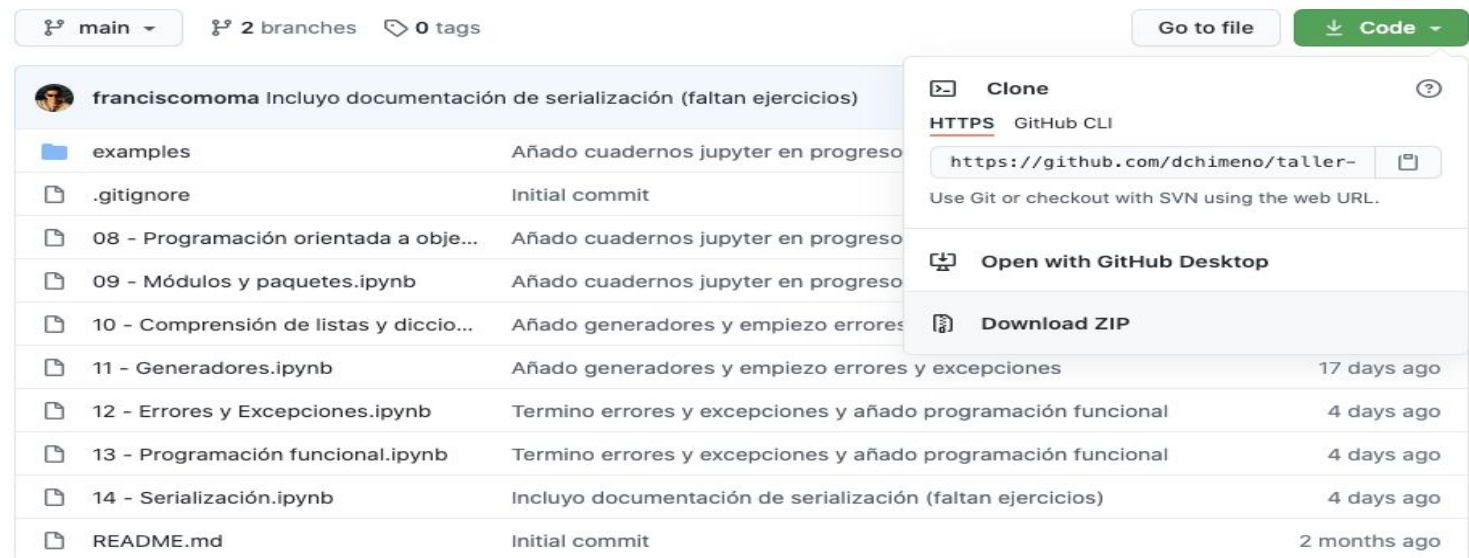
Descarga de archivos

<https://github.com/dchimeno/taller-python>

- Podéis descargar los archivos con git:

`git clone https://github.com/dchimeno/taller-python`

- O podéis descargarlo como ZIP



Virtualenvs

- Los virtualenvs son muy utilizados en Python para aislar las diferentes versiones de Python y los diferentes proyectos.
- Así, según el proyecto, podremos tener diferentes librerías y versiones para cada uno

En linux/mac

Una vez localizado donde se ha descargado Python. En mac con el instalador, en:

/Library/Frameworks/Python.framework/Versions/3.8/bin/python3

```
$ /Library/Frameworks/Python.framework/Versions/3.8/bin/python3 -m venv venv  
source venv/bin/activate  
python3
```

-> aquí ya estamos dentro de nuestro entorno virtual

Para salir del virtualenv:

deactivate

En Windows

Si hemos incluido el path

```
python -m venv venv  
venv\Scripts\activate  
deactivate
```



Instalación de dependencias

En linux/mac

Desde la carpeta de ejercicios y con el virtualenv activado, instalaremos las dependencias con (las necesitaremos después)

```
python3 -m pip install -r requirements.txt
```

En Windows



Editores de Python: PyCharm

- Página para descarga de pycharm:

<https://www.jetbrains.com/pycharm/download/>

The image shows the PyCharm download page and the installation wizard. The download page has a navigation bar with links: Coming in 2017.1, What's New, 2016.3, Features, Docs & Demos, Buy, and Download. The main content area is titled 'Download PyCharm' and has tabs for macOS, Windows, and Linux. Under the Windows tab, there are two options: 'Community' (Lightweight IDE for Python & Scientific development, 179 MB) and 'Professional' (Full-featured IDE for Python & Web development, 232 MB). Both have 'DOWNLOAD' buttons. An orange arrow points to the 'Windows' tab, and another points to the 'Community' download button. Below the download buttons is a banner for 'pycharm-community-2016.3.2'. To the right, the 'PyCharm Community Edition Setup' window is shown. It has a title bar with standard window controls. The main area has the PyCharm logo and the text 'Welcome to the PyCharm Community Edition Setup Wizard'. It explains that the wizard will guide through the installation and recommends closing other applications. It says 'Click Next to continue.' and has 'Next >' and 'Cancel' buttons at the bottom. An orange arrow points to the 'Next >' button.

PyCharm

Coming in 2017.1 What's New 2016.3 Features Docs & Demos Buy **Download**

Download PyCharm

macOS **Windows** Linux

Community

Lightweight IDE for Python & Scientific development

DOWNLOAD

179 MB

Professional

Full-featured IDE for Python & Web development

DOWNLOAD

232 MB

pycharm-community-2016.3.2

PyCharm Community Edition Setup

Welcome to the PyCharm Community Edition Setup Wizard

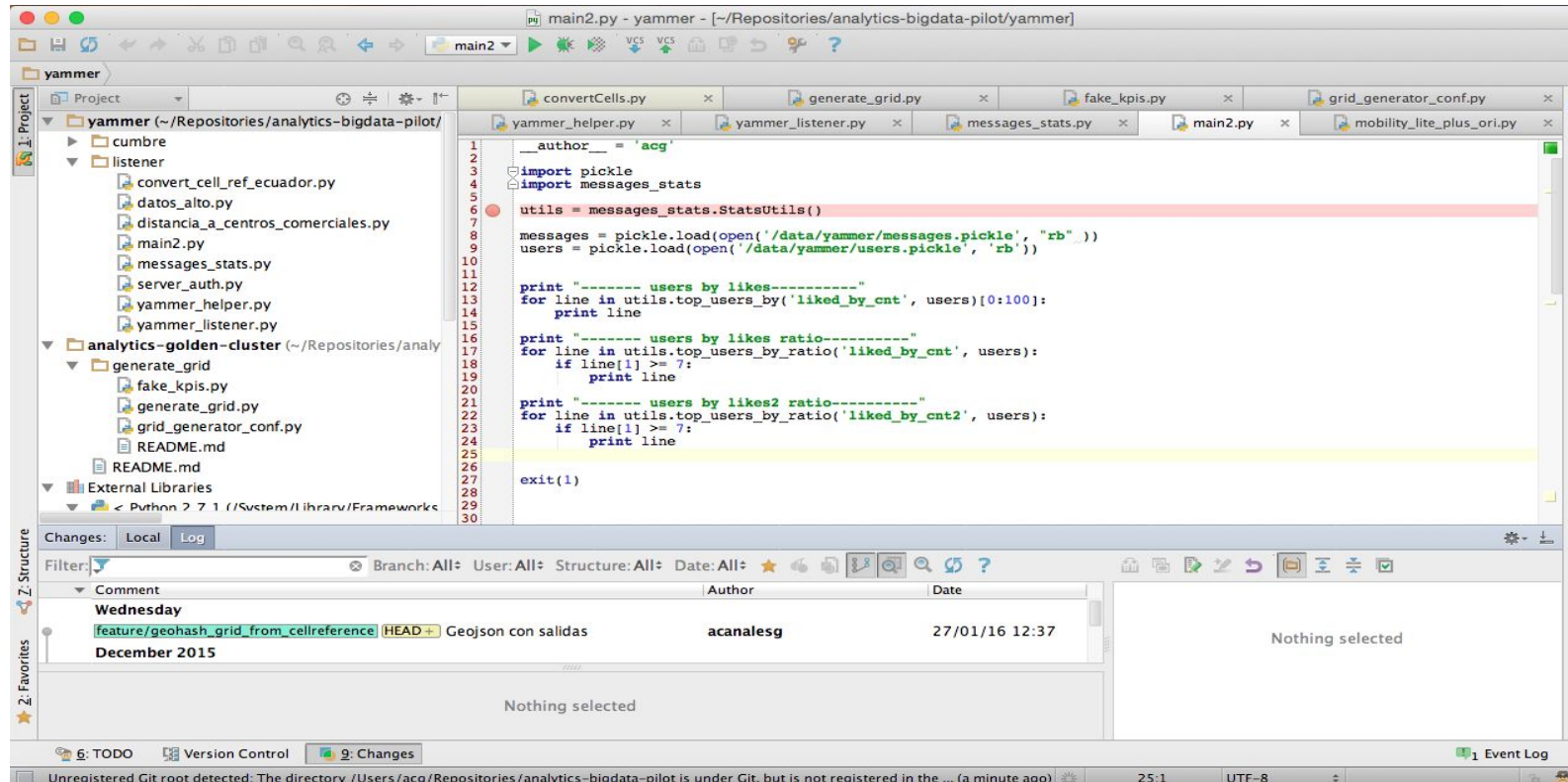
This wizard will guide you through the installation of PyCharm Community Edition.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Next > Cancel

Editores de Python: PyCharm

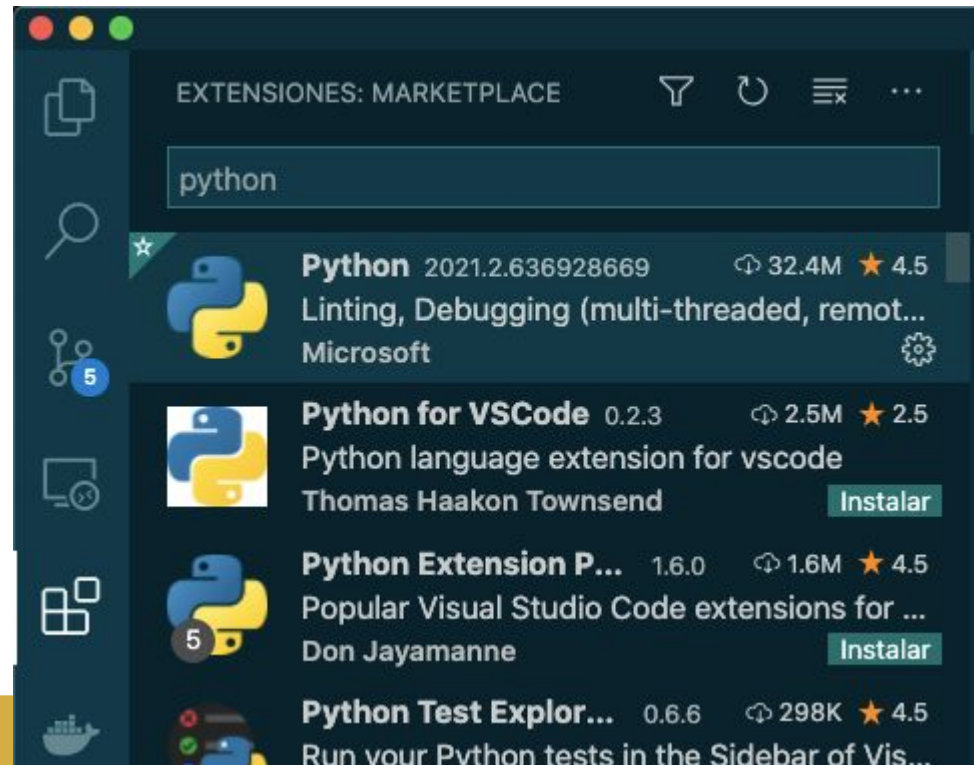


Editores de Python: VsCode

- Página para descarga de vscode:

<https://code.visualstudio.com/>

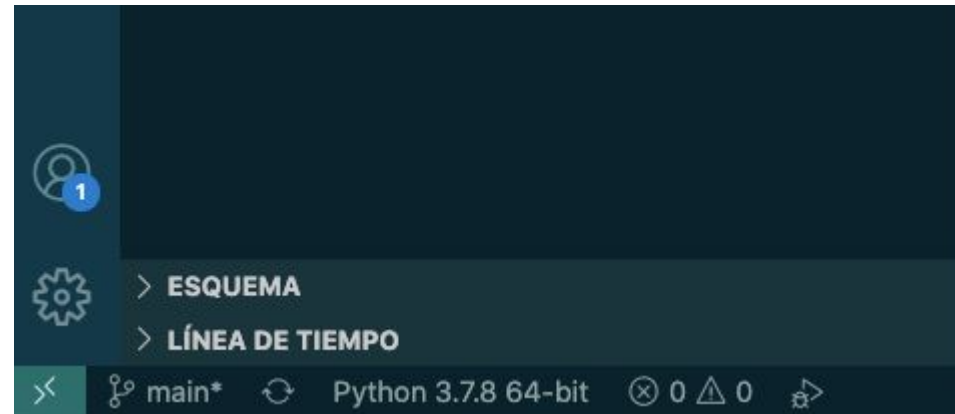
- Instalar el plugin de python



vsCode: Seleccionar venv

Debemos decirle a Visual Studio Code, con qué Python ejecutar nuestros archivos.

Si hemos creado un venv en la carpeta del proyecto, lo detectará automáticamente, si no, se lo podemos poner en:



Ejercicios

1. Abrir la carpeta de ejercicios con VsCode y añadir un fichero test.py
2. Añadir un fichero hola_mundo.py y escribir hola mundo.
3. Calcular en VsCode cuanto es la suma de 4 y 7, y el producto de 225 y 6
4. Hacer que muestre por pantalla el zen de Python
5. Ejecutar con el IDE.



Solución Ejercicios VsCode

```
test.py
1 print(4+7)
2
3
4 print(225*6)
5
6
7 import this
```

TERMINAL PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN

2: Python Debug Console ▾ + [] ✕ ^ ×

```
11
1350
dchimeno@mac-516804 [ venv ] ~/Documents/master/taller-python [
[ main ] cd /Users/dchimeno/Documents/master/taller-python ; /usr/bin/env /Users/dchimeno/Documents/master/taller-python/venv/bin/python /Users/dchimeno/.vscode/extensions/ms-python.python-2021.2.636928669/pythonFiles/lib/python/debugpy/launcher 51063 -- /Users/dchimeno/Documents/master/taller-python/test.py
11
1350
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one— and preferably only one —obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
dchimeno@mac-516804 [ venv ] ~/Documents/master/taller-python [
```

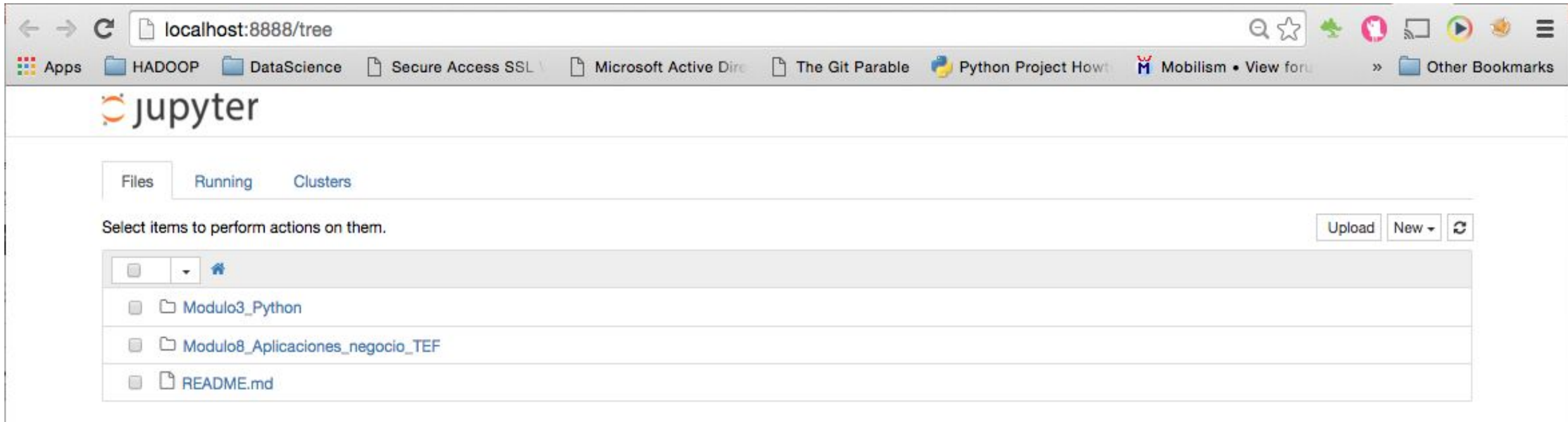


IPython y los notebooks

Desde la terminal con el entorno activado:

> jupyter notebook

Se abrirá el navegador con nuestros archivos



Ejercicio Notebooks Python

1. Arrancar el servidor de notebooks con prompt: “jupyter notebook”
2. Crear un notebook y guardarlo como “Primeros Pasos en Python”
3. Calcular en notebook cuanto la suma de 4 y 7, y el producto de 225 y 6
4. Crear otro fichero python zen.py que nos muestre por pantalla el zen de Python



Solución Ejercicios Ipython

The screenshot displays the JupyterLab web interface in a browser. The top navigation bar includes tabs for 'Files', 'Running', and 'Clusters'. Below this, a message states 'Select items to perform actions on them.' The file browser shows a directory structure with folders like 'analytics-data_normalisation' and 'Application Data'. A 'New' button is visible, which has been clicked to open a dropdown menu. This menu lists options: 'Text File', 'Folder', 'Terminals Unavailable', 'Notebooks', and 'Python 2'. An orange arrow points to the 'Notebooks' option. Below the file browser, a second browser window shows the JupyterLab interface for a notebook named 'primeros_pasos_python'. The 'File' menu is open, showing options like 'New Notebook', 'Open...', 'Make a Copy...', 'Rename...', 'Save and Checkpoint', 'Revert to Checkpoint', 'Print Preview', 'Download as', 'Trusted Notebook', and 'Close and Halt'. Two orange arrows point to the 'New Notebook' and 'Save and Checkpoint' options. The notebook's toolbar shows the 'Code' button selected. The bottom of the image features a yellow banner with the Python logo on the left and the number '30' on the right.

URM - OneDrive x analytics-data-nor x Home x edx-dataframes x hadoop - How to e x Home x Untitled x patricia

localhost:8888/tree?token=389edaac2bd138c9bd05642442019f71bb89445bcd9a359#

jupyter Logout

Files Running Clusters

Select items to perform actions on them.

Upload New

Text File
Folder
Terminals Unavailable
Notebooks
Python 2

Create a new notebook with

URM - OneDrive x analytics-da x Home x edx-datafra x hadoop - H x Home x

localhost:8888/notebooks/primeros_pasos_python.ipynb

jupyter primeros_pasos_python Last Checkpoint: a few seconds ago (autosaved)

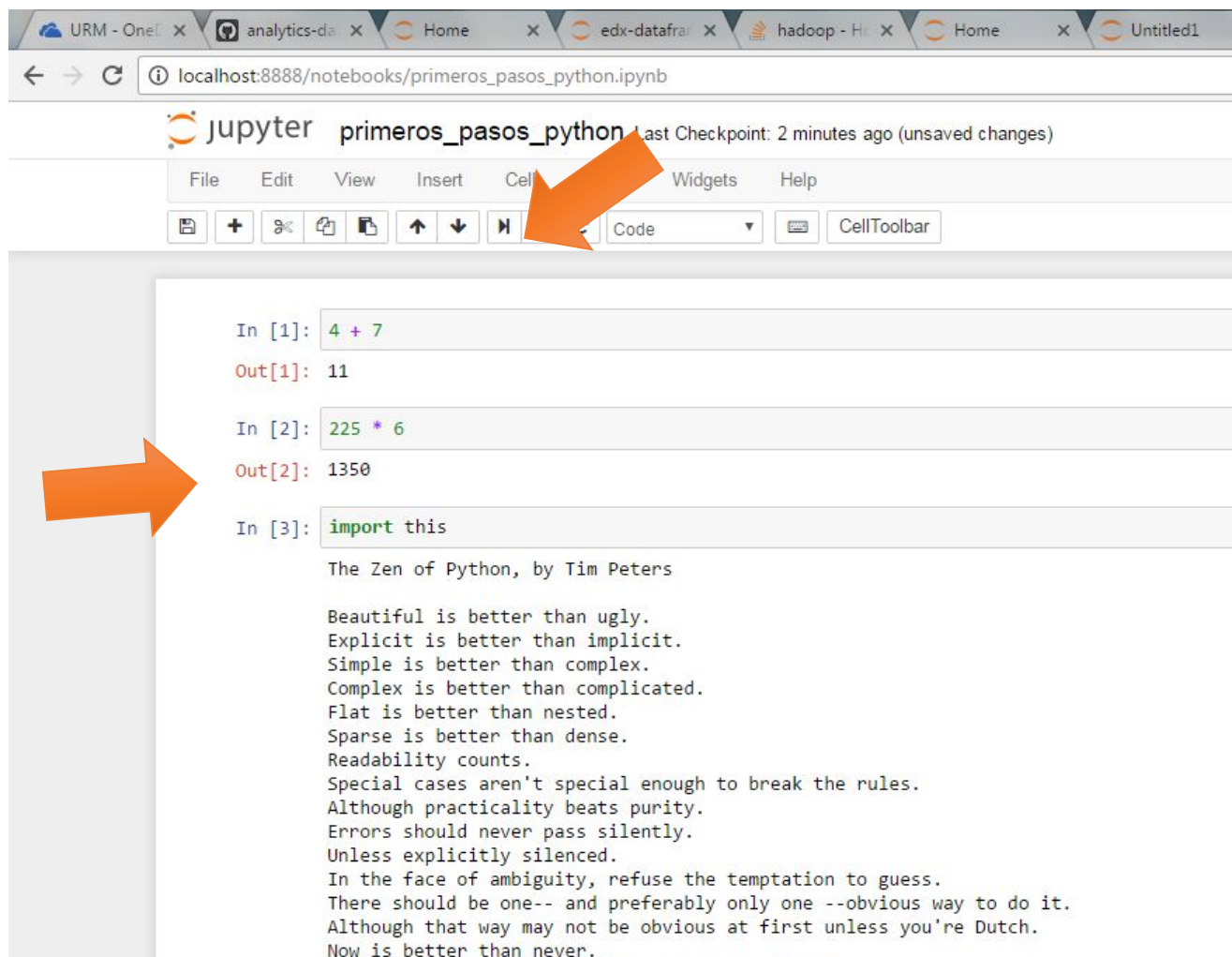
File Edit View Insert Cell Kernel Widgets Help

New Notebook
Open...
Make a Copy...
Rename...
Save and Checkpoint
Revert to Checkpoint
Print Preview
Download as
Trusted Notebook
Close and Halt

Code CellToolbar



Solución Ejercicios Ipython



URM - One X analytics-da X Home X edx-datafra X hadoop - H X Home X Untitled1 X

localhost:8888/notebooks/primeros_pasos_python.ipynb

Jupyter primeros_pasos_python Last Checkpoint: 2 minutes ago (unsaved changes)

File Edit View Insert Cell Widgets Help

Code CellToolbar

```
In [1]: 4 + 7
Out[1]: 11

In [2]: 225 * 6
Out[2]: 1350

In [3]: import this

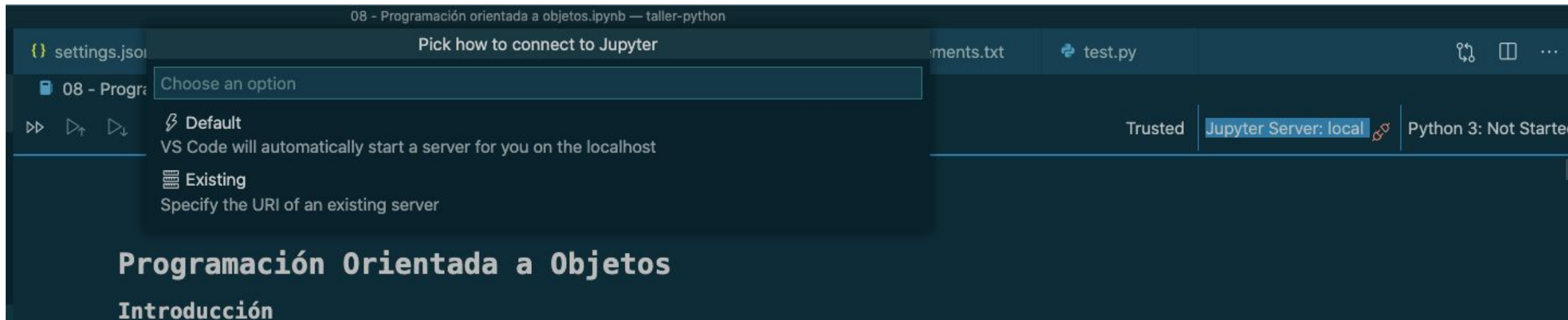
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
```



Jupyter VsCode

También es posible integrar VsCode con nuestros notebooks, y editarlos y ejecutarlos desde VsCode (necesita la extensión de Python instalada)



Palabras Reservadas en Python

- También llamadas *keywords*
- Son ciertos identificadores cuyo uso se prohíbe
- Los utiliza el lenguaje con propósitos particulares como parte de las instrucciones

Keywords in Python programming language				
False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

```
>> import keyword  
>> print(keyword.kwlist)
```



Tipo de datos

- Booleanos
- numéricos
- *cadena*
- *lista*
- *Tupla*
- *diccionario*



Tipos de Datos: Asignación

- Python es un lenguaje de tipado dinámico, no tenemos que declarar el tipo de variable al escribir el código, los tipos se asignan en tiempo de ejecución.
- La instrucción de asignación tiene esta forma:

identificador = expresión

- Para crear una variable, simplemente la asociamos un valor.

```
>>> edad = 24  
>>> a = 3 + 7
```

- Para averiguar el tipo de una variable podemos usar la función ***type()***

```
>>> type(24)  
<type 'int'>
```



Ejemplos de Asignación

- Se pueden asignar varias variables el mismo valor:

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

- Sería equivalente a poner:

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

- También se puede asignar los valores en paralelo:

```
>>> a, b = 24, 26  
>>> a  
24  
>> b  
26
```



Tipos de Datos: Booleanos

- Variables binarias que pueden tomar valor VERDADERO (True o 1) o FALSO (False o 0)
- Tenemos 3 operadores lógicos para trabajar con booleanos:

Operación	Resultado	Notas
x or y	Si el valor de x es falso devuelve el valor de y, sino devuelve el valor lógico de x.	Solo se evalúa el segundo argumento si el primero es False.
x and y	Si el valor de x es falso devuelve el valor de x, sino devuelve el valor de y.	Solo se evalúa el segundo argumento si el primero es True.
not y	Si el valor de x es falso, devuelve True, sino devuelve False	

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

```
>>> bool(1)
True
>>> bool(0)
False
```

```
>>> not(False)
True
>>> not(True)
False
```



Tipos de Datos: Booleanos

- Podemos generar booleanos a partir de otras variables mediante funciones de comparación:

- > Mayor que; < Menor que;

```
>>> 3 > 2
True
>>> 3 < 2
False
```

- >= Mayor o igual que; <= Menor o igual que;

```
>>> 2 >= 1 + 1
True
>>> 5 - 2 <= 2
False
```

- == Igual que; != Distinto de;

```
>>> 5 == 4 + 1
True
>>> 8 / 2 != 4
False
```

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

Aunque hay reglas de precedencia lo más aconsejable para evitar comportamientos inesperados es utilizar paréntesis para especificar nuestras comparaciones complejas



Ejemplos: Booleanos

```
>>> valor = 4 < 5
>>> valor
True
>>> valor = 4 < 5 < 4
>>> valor
False
>>> 'ol' in 'Hola'
True
>>> 'A' not in 'Hola'
True
>>> not 'A' not in 'Hola'
False
>>> a, b, c = True, True, False
>>> a or b and c
True
>>> (a or b) and c
False
```



Tipos de Datos: Números

- Varios tipos de números:

CLASE	TIPO	DESCRIPCION	EJEMPLO
Int	Numérico	Número entero.	entero = 53
Float	Numérico	Numero reales con decimales	Real = 2.59
complex	Numérico	Parte real y parte imaginaria	complejo = $8.2 + 4.8j$



Tipos de Datos: Números

- En Python podemos trabajar con diferentes tipos de datos numéricos:

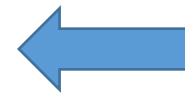
```
>>> a = 24  
>>> b = 2.565  
>>> c = 0.1e-7  
>>> d = 12345678901234567890
```

¿Qué tipo de dato son a, b, c y d?

- Se podría forzar el tipo de dato utilizando las funciones int(), float() o long().

```
>>> a = int(24)  
>>> b = float(2.565)  
>>> c = float(0.1e-7)  
>>> d = long(12345678901234567890)
```

¿Qué tipo de dato son a, b, c y d?



Tipos de Datos: Números

- Las operaciones que están definidas para todos los tipos de datos numéricos son las siguientes:

Operation	Result
$x + y$	Suma x e y
$x - y$	Resta x e y
$x * y$	Producto x e y
x / y	Cociente de x e y
$x // y$	Parte entera de x e y
$x \% y$	Resto de x e y
$-x$	x cambiado de signo
$+x$	X sin cambios
$\text{abs}(x)$	Valor absolute de x
$x ** y$	<i>X elevado a la potencia de y</i>
$\text{min}(\text{Arg}, \text{Arg}, \dots)$	Devuelve el valor minimo
$\text{max}(\text{Arg}, \text{Arg}, \dots)$	Devuelve el valor maximo



Tipos de Datos: Números

- Las operaciones que están definidas para todos los tipos de datos reales (entero, entero largo y punto flotante) son las siguientes:

Operation	Result
<code>math.trunc(x)</code>	x truncado a la parte entera
<code>round(x[, n])</code>	x redondeado a n dígitos, redondeando hacia arriba. Si se omite n se considera 0.
<code>math.floor(x)</code>	La parte entera menor o igual de x.
<code>math.ceil(x)</code>	La parte entera más grande de x.

- NOTA: Hay que importar la librería “math” para poder utilizar estas operaciones



Ejercicios Tipos de Datos: Números

- Averigua los valores que se obtienen al realizar las siguientes operaciones:

$a + b$

b / c

- Averigua cuánto es 659 dividido entre 4
- Truncar el valor b
- Redondear a dos decimales el valor b
- Obtener la parte entera del valor b
- Obtener la parte mayor del valor b

```
>>> a = 24  
>>> b = 2.565  
>>> c = 0.1e-7  
>>> d = 12345678901234567890
```



Solución Tipos de Datos: Números

```
>>> a = 24
>>> b = 2.565
>>> c = 0.1e-7
>>> d =
12345678901234567890

>>> a+b
26.565
>>> b/c
256500000.0

>>> 659/4
164
```

```
>>> import math
>>> math.trunc(b)
2

>>> round(b,2)
2.56
>>> math.floor(b)
2.0
>>> math.ceil(b)
3.0
```



Tipos de Datos: Cadenas

- Para trabajar con cadenas, simplemente las incluiremos entre comillas (simples o dobles)

```
>>> saludo = "Hola"  
>>> type(saludo)  
>>> str
```

- Para acceder a partes de la cadena utilizaremos el operador [], en Python el indexado empieza en 0 (no en 1).

```
>>> saludo[0]  
'H'  
>>> saludo[2]  
'l'  
>>> saludo[1:]  
'ola'
```



Tipos de Datos: Cadenas

- Las cadenas tienen las siguientes operaciones:
 - Suma (concatenación) de dos variables o más:

```
>>> a, b = "hola",  
"mundo"  
>>> a + b  
>>> "hola mundo"
```

- Multiplicación de cadenas:

```
>>> a = "todos"  
>>> a * 3  
>>> "todostodostodos"
```

```
>>> "Hola " + "Pepe"  
'Hola Pepe'  
>>> b = "hola"  
>>> c = 1  
>>> cadena = b + str(c)  
>>> cadena  
'hola1'
```



Ejercicios Tipos de Datos: Cadenas

- Ejercicio:

1. Crear una cadena con el abecedario
2. ¿Cuál es la letra número 5 del alfabeto?
3. ¿Cuál es la penúltima?



Solución Tipos de Datos: Cadenas

Ejercicio:

1. Crear una cadena con el abecedario
2. ¿Cuál es la letra numero 5 del alfabeto?
3. ¿Cuál es la penúltima?

```
import string
print(string.ascii_lowercase)
'abcdefghijklmnopqrstuvwxyz'
```

```
>>> abecedario =
"abcdefghijklmnopqrstuvwxyz"
>>> abecedario[5-1]
'e'
>>> len(abecedario)
26
>>> abecedario[26-2]
'y'
>>> abecedario[-2]
'y'
```



Tipos de Datos: Cadenas

- Las operaciones que están definidas para todos los tipos de datos string son las siguientes: `variable_string.operation()`

Operation	Result
<code>upper()</code>	Se convierte a mayúsculas
<code>lower()</code>	Se convierte a minúsculas
<code>replace(old, new)</code>	Reemplaza las apariciones de old en la cadena por new
<code>capitalize()</code>	Hace mayúsculas la primera letra
<code>index(cadena2)</code>	Busca la posición de cadena 2 en nuestra cadena
<code>isdigit()</code>	True si la cadena es numérica
<code>split(separador)</code>	Devuelve una lista de las subcadenas usando el separador <i>separador</i>
<code>format()</code>	Permite construir cadenas en base a parámetros
<code>Find(cadena2)</code>	Busca la posición de cadena 2 en nuestra cadena
<code>center(n, Simbolo)</code>	Devuelve una cadena de longitud n con la cadena centrada en ella
<code>len(str)</code>	Saber la longitud de la cadena



Tipos de Datos: Cadenas

- Un ejemplo de cómo se utiliza la función format:

```
>>> cadena = "Hola {0}. Como estas {0}? Yo me llamo {1}. Hoy es dia {2}"
>>> cadena.format('Ramon', 'Patricia', 10)
'Hola Ramon. Como estas Ramon?. Yo me llamo Patricia. Hoy es dia 10'

>>> type(cadena)
<type 'str'>
```

- Para ver todas las funciones de la cadena se puede ver:

```
>>> help(str)
```



Tipos de Datos: Cadenas

- Desde Python 3.6 tenemos f-strings

```
>>> nombre_1, nombre = "Ramón", "Patricia"  
cadena = f"Hola {nombre_1}. Como estas {nombre_2}?"  
>>> print(cadena)  
'Hola Ramon. Como estas Patricia?'  
  
>>> type(cadena)  
<type 'str'>
```

- Para ver todas las funciones de la cadena se puede ver:

```
>>> help(str)
```



Ejercicios Tipos de Datos: Cadenas

- Ejercicio: (01. Parte 1)
 1. Rellenar la cadena del abecedario con el símbolo # hasta completar 30 caracteres.
 2. Mirar la longitud de la cadena abecedario.
 3. Poner en mayúsculas y en minúsculas la cadena.
 4. Buscar en la cadena abecedario la letra R y saber su posición
 5. Crea otra cadena con las 5 letras que van después de la n
 6. Reemplazar la letra Z por Hola.
 7. Obtener una lista de la cadena abecedario y separarla por la letra i
 8. Unir la lista anterior con el símbolo “-” usando ‘-’.join(lista).



Solución Ejercicios Tipos de Datos: Cadenas

- Ejercicio:

1. Rellenar la cadena del abecedario con el símbolo ## hasta completar 30 caracteres.
2. Mirar la longitud de la cadena abecedario.
3. Poner en mayúsculas y en minúsculas la cadena.
4. Buscar en la cadena abecedario la letra R
5. Crea otra cadena con las 5 letras que van después de la n
6. Reemplazar la letra Z por Hola.
7. Obtener una lista de la cadena abecedario y separarla por la letra i
8. Unir la lista anterior con el símbolo “-”

```
>>> abecedario="abcdefghijklmnopqrstuvwxyz"
>>> abecedario.center(30,'#')
'##abcdefghijklmnopqrstuvwxyz##'
>>> len(abecedario)
26
>>> abecedario.upper()
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> abecedario.lower()
'abcdefghijklmnopqrstuvwxyz'
>>> abecedario.index('r')    o >>> abecedario.find('r')
17
>>> abecedario.index('n')
13
>>> abecedario[13+1:13+6]
'opqrs'
>>> abecedario.replace("z","hola")
'abcdefghijklmnopqrstuvwxyzhola'
>>> abecedario.split("i")
['abcdefgh', 'jklmnopqrstuvwxyz']
>>> "-".join(abecedario.split("i"))
'abcdefgh-jklmnopqrstuvwxyz'
```



Tipos de Datos: Listas

- Para almacenar valores en Python usaremos listas

```
>>> mi_lista = [1,2,3,4]
>>> mi_lista = list([1,12])
>>> mi_lista[1]
2
```

- Al igual que con las cadenas utilizaremos el operador [] para acceder a los distintos elementos.
- Si tenemos una cadena, podemos usar el comando .split(sep) para construir una lista conforme al separador indicado.

```
>>> "Uno,dos,tres,cuatro,cinco".split(",")
['Uno', 'dos', 'tres', 'cuatro', 'cinco']
```



Tipos de Datos: Listas

<https://docs.python.org/3/tutorial/datastructures.html>

List operation	Explanation	Example
<code>[]</code>	Creates an empty list	<code>x = []</code>
<code>len</code>	Returns the length of a list	<code>len(x)</code>
<code>append</code>	Adds a single element to the end of a list	<code>x.append('y')</code>
<code>insert</code>	Inserts a new element at a given position in the list	<code>x.insert(0, 'y')</code>
<code>del</code>	Removes a list element or slice	<code>del(x[0])</code>
<code>remove</code>	Searches for and removes a given value from a list	<code>x.remove('y')</code>
<code>reverse</code>	Reverses a list in place	<code>x.reverse()</code>
<code>sort</code>	Sorts a list in place	<code>x.sort()</code>
<code>+</code>	Adds two lists together	<code>x1 + x2</code>
<code>*</code>	Replicates a list	<code>x = ['y'] * 3</code>
<code>min</code>	Returns the smallest element in a list	<code>min(x)</code>
<code>max</code>	Returns the largest element in a list	<code>max(x)</code>
<code>index</code>	Returns the position of a value in a list	<code>x.index('y')</code>
<code>count</code>	Counts the number of times a value occurs in a list	<code>x.count('y')</code>
<code>in</code>	Returns whether an item is in a list	<code>'y' in x</code>

Table 5.1 List operations



Ejercicios Tipos de Datos: Listas

01. Parte 1 - 2

1. Crear una lista con los meses del año
2. Selecciona el mes noveno, y el penúltimo
3. Crea una lista con los meses de verano (septimo a noveno)
4. Crea una lista con los meses de verano en orden inverso



Ejercicios Tipos de Datos: Listas

```
>>> meses=['ENERO','FEBRERO','MARZO','ABRIL','MAYO','JUNIO',  
'JULIO','AGOSTO','SEPTIEMBRE','OCTUBRE','NOVIEMBRE',  
'DICIEMBRE']  
>>>  
>>> meses[8]  
'SEPTIEMBRE'  
>>> meses[-1]  
'DICIEMBRE'  
>>> verano = meses[6:8]  
>>> print verano  
['JULIO', 'AGOSTO']  
>>> verano.reverse()  
>>>  
>>> print verano  
['AGOSTO', 'JULIO']  
>>>
```



Extra: Calendar

Python tiene un rico sistema de librerías estándar.

“With batteries included”

```
import calendar  
calendar.monthcalendar(2021, 3)
```

```
[[1, 2, 3, 4, 5, 6, 7],  
 [8, 9, 10, 11, 12, 13, 14],  
 [15, 16, 17, 18, 19, 20, 21],  
 [22, 23, 24, 25, 26, 27, 28],  
 [29, 30, 31, 0, 0, 0, 0]]
```

[Documentación calendar](#)



TIPOS DE DATOS: Tuplas

- Una tupla al igual que una lista es una estructura para almacenar secuencias de valores, en vez de [] las tuplas se usan (), la diferencia fundamental con las listas es que las tuplas son inmutables, es decir, no se pueden modificar.
- Aunque se use con (), el constructor real de una tupla es ,

```
>>> mi_tupla = (1,2,3,4)
>>> mi_tupla = 1,2,3,4
>>> mi_tupla = tuple([1,12])
```

- Podemos crear tuplas a partir de listas, y también listas a partir de tuplas.
- Las operaciones para acceder, concatenar, dividir tuplas son las mismas que las indicadas para listas. Nótese que aquellas operaciones que modifican una lista (ej. append())no van a ser aplicables a una tupla



Ejercicios Tipos de Datos: Tuplas

1. Crear una tupla con los meses del año
2. Selecciona el mes noveno, y el penúltimo
3. Selecciona todos los meses después de Agosto (no incluido)



Ejercicios Tipos de Datos: Tuplas

```
>>> meses=('ENERO','FEBRERO',  
'MARZO','ABRIL','MAYO','JUNIO',  
'JULIO','AGOSTO','SEPTIEMBRE',  
'OCTUBRE','NOVIEMBRE','DICIEMBRE')  
>>>  
>>> meses[8]  
'SEPTIEMBRE'  
>>> meses[-2]  
'NOVIEMBRE'  
  
>> meses[8:]
```



TIPOS DE DATOS: DICCIONARIOS

- También llamados hash tables, se trata de una colección no ordenada de pares clave-valor.
- Para acceder a los datos del diccionario, en vez del índice numérico usaremos la clave.
- Las claves deben ser únicas dentro del diccionario, los valores no necesariamente
- Las claves deben de ser de tipo inmutable
- Optimizados para reducir el tiempo de acceso
- Nota: Desde Python 3.7 los diccionarios son (oficialmente) ordenados por inserción. Es decir, podemos asegurar que el orden en el que se crean las clave:valor, es el mismo.
 - Esto es una optimización, aunque hay veces que podemos aprovecharnos de ello.

```
>>> my_dict = {'Enero': 31, 'Febrero': 28}
>>> my_dict.get('Enero')
31
>>> my_dict['Enero']
31
>>> my_dict['Marzo'] = 31
```



TIPOS DE DATOS: DICCIONARIOS

- Podemos iterar sobre un diccionario mediante un bucle for:

```
>>> for clave in mi_diccionario:  
...     print(clave, mi_diccionario[clave])
```

- O también incluyendo los valores:

```
>>> for clave, valor in mi_diccionario.items():  
...     print(clave, valor)
```

- Podemos utilizar “in/not in” para ver si un elemento está contenido en un diccionario (como clave)

```
>>> 'Enero' in my_dict  
True
```



TIPOS DE DATOS: DICCIONARIOS

Los diccionarios contienen los siguiente métodos:

Operation	Result
<code>len(d)</code>	Devuelve el número de elementos del diccionario <i>d</i> .
<code>d[key]</code>	Devuelve el valor asociado a la clave <i>key</i> .
<code>d[key] = value</code>	Añade al diccionario la clave <i>key</i> con el valor <i>value</i> , si la clave ya existe modifica el valor
<code>del d[key]</code>	Elimina la clave <i>key</i> .
<code>d.clear()</code>	Elimina todos los elementos del diccionario
<code>d.copy()</code>	Devuelve una copia superficial del diccionario <i>d</i> .
<code>d.get(key)</code>	Devuelve el valor asociado a la clave <i>key</i> .
<code>d.items()</code>	Devuelve una lista de tuplas compuestas por los pares clave valor que forman el diccionario.
<code>d.keys()</code>	Devuelve una lista con las claves del diccionario.
<code>d.values()</code>	Devuelve una lista con los valores del diccionario



Ejercicios Tipos de Datos: Diccionario

1. Crea un diccionario con todos los meses del año, donde la key sea el mes y el valor los días del mes.
2. Mostrar los días que tiene el mes de Marzo
3. Crea una tupla con los keys del diccionario que se ha creado.
4. Crea una tupla con los values del diccionario que se ha creado.
5. Tamaño del diccionario
6. Borrar del diccionario los meses de Julio y Agosto
7. Recorrer el diccionario y mostrar por pantalla mes con los días del mes.



Ejercicios Tipos de Datos: Diccionario

```
>>> meses={'ENERO':31,'FEBRERO':28,
'MARZO':31,'ABRIL':30,'MAYO':31,'JUNIO':30,
'JULIO':31,'AGOSTO':31,'SEPTIEMBRE':30,
'OCTUBRE':31,'NOVIEMBRE':30,
'DICIEMBRE':31}
>>> meses['Marzo']
31
>>> meses.keys()
['SEPTIEMBRE', 'NOVIEMBRE', 'JUNIO',
'AGOSTO', 'ENERO', 'OCTUBRE', 'JULIO',
'MARZO', 'ABRIL', 'FEBRERO', 'DICIEMBRE',
'MAYO']
>>> meses.values()
[30, 30, 30, 31, 31, 31, 31, 31, 30, 28, 31, 31]
>>> len(meses)
12
>>> del meses['JULIO']
>>> meses
{'SEPTIEMBRE': 30, 'NOVIEMBRE': 30, 'JUNIO':
30, 'AGOSTO': 31, 'ENERO': 31, 'OCTUBRE': 31,
'MARZO': 31, 'ABRIL': 30, 'FEBRERO': 28,
'DICIEMBRE': 31, 'MAYO': 31}
>>> del meses['AGOSTO']
```

```
{'SEPTIEMBRE': 30,
'NOVIEMBRE': 30, 'JUNIO': 30,
'ENERO': 31, 'OCTUBRE': 31,
'MARZO': 31, 'ABRIL': 30,
'FEBRERO': 28, 'DICIEMBRE': 31,
'MAYO': 31}
>>> for mes,dias in meses.items():
...     print(f"{mes}:{dias}")
SEPTIEMBRE:30
NOVIEMBRE:30
JUNIO:30
ENERO:31
OCTUBRE:31
MARZO:31
ABRIL:30
FEBRERO:28
DICIEMBRE:31
MAYO:31
```

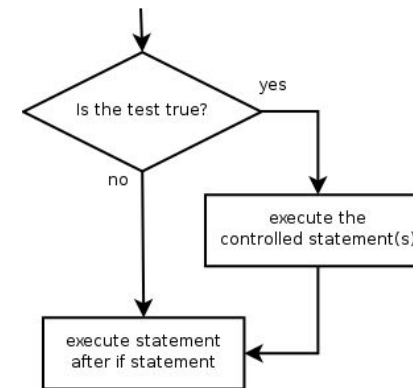
- Estructura de control de flujo:
 - IF
 - *FOR*
 - *WHILE*
 - *Funciones*



Sentencias Condicionales: IF

- Un elemento básico en los lenguajes de programación son las estructuras condicionales, en Python la sintaxis de los bloques IF es muy sencilla:

```
if <sentencia>:  
    do this  
    and this
```



- No hay llaves alrededor de los distintos bloques.
- Ejemplo:

```
constante = 3.4
```

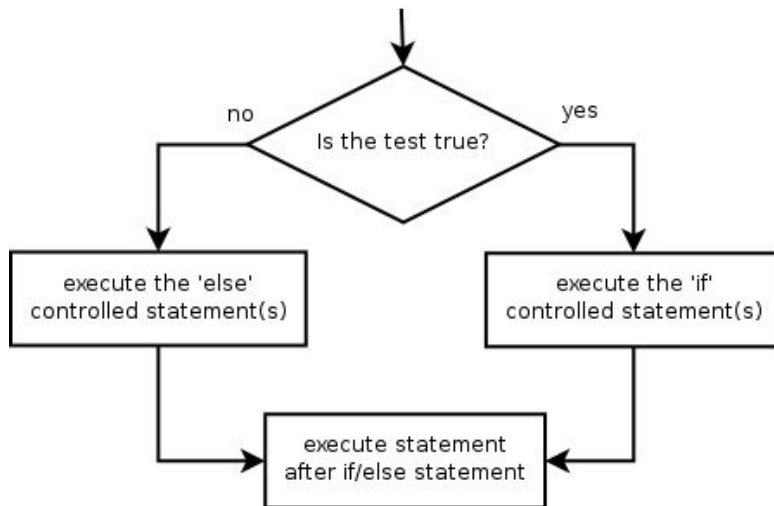
```
if constante > 2.0:
```

```
    print("Es correcto")
```

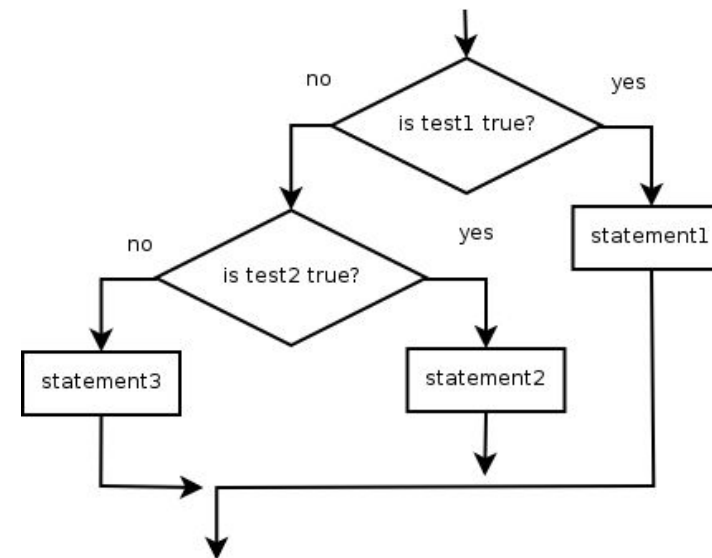


Sentencias Condicionales: IF

```
if <sentencia>:  
    do this  
    and this  
else:  
    do that
```



```
if <sentencia>:  
    do this  
elif <sentencia2>:  
    do this other thing  
else:  
    do that
```



Sentencias Condicionales: IF

Muchas expresiones lógicas utilizan operadores relacionales:

Operator	Meaning	Example	Result
==	equals	<code>1 + 1 == 2</code>	True
!=	does not equal	<code>3.2 != 2.5</code>	True
<	less than	<code>10 < 5</code>	False
>	greater than	<code>10 > 5</code>	True
<=	less than or equal to	<code>126 <= 100</code>	False
>=	greater than or equal to	<code>5.0 >= 5.0</code>	True

Operator	Example	Result
and	<code>9 != 6 and 2 < 3</code>	True
or	<code>2 == 3 or -1 < 5</code>	True
not	<code>not 7 > 0</code>	False

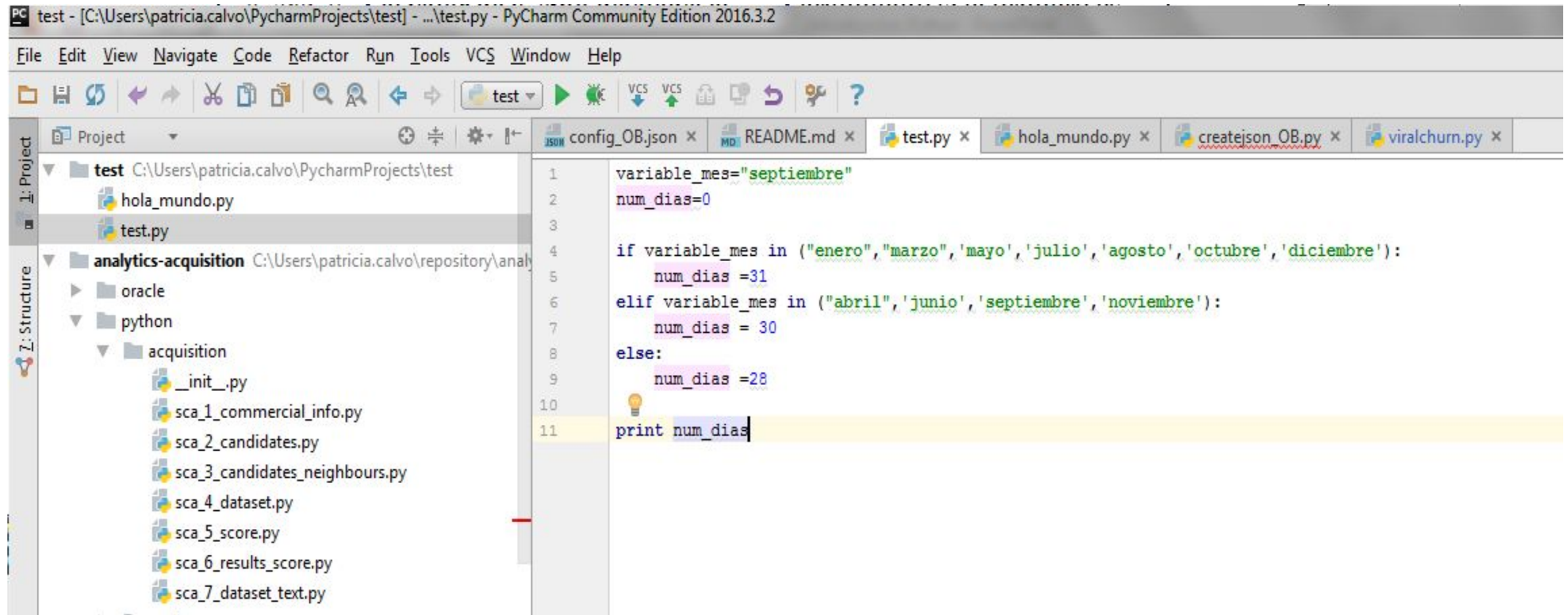


Ejercicio Sentencias Condicionales: IF

- Ejercicio 5 (Utilizar bloque condicional):
 1. Dependiendo del valor que se le asigne a una variable llamada mes nos tendrá que dar el número de días que tiene ese mes en otra variable llamada num_dias.



Solución Sentencias Condicionales: IF



The screenshot shows the PyCharm IDE interface. The left sidebar displays the project structure with folders 'test' and 'analytics-acquisition'. The main editor window shows the file 'test.py' with the following code:

```
1 variable_mes="septiembre"
2 num_dias=0
3
4 if variable_mes in ("enero","marzo","mayo","julio","agosto","octubre","diciembre"):
5     num_dias =31
6 elif variable_mes in ("abril","junio","septiembre","noviembre"):
7     num_dias = 30
8 else:
9     num_dias =28
10
11 print num_dias
```



Funciones en Python

- Para declarar una función utilizaremos la palabra `def`, a continuación especificaremos el número de parámetros

```
def mi_funcion(arg1, arg2):  
    haz esto  
    haz aquello  
    return resultado
```

- De nuevo la indentación de 4 espacios
- Utilizamos `return` para devolver el resultado de la ejecución de la función
- Nótese que en ningún momento indicamos el tipo de dato que devolvemos, ni tampoco el de los argumentos, en función de la ejecución de la función podría incluso ocurrir que devolvamos tipos distintos de datos
- En versiones recientes de Python 3 se han incluido anotaciones, que, de manera **opcional**, nos pueden ayudar a establecer tipos para nuestros datos



Ejercicio de Funciones Python

- Ejercicio:

1. Crear una función que calcule el número de días de un mes (ejercicio anterior).
2. Crear una función que reciba un argumento, y si se trata de un cadena retorne “Es un string: nombre”. Si el argumento es erróneo debe devolver “Error”
3. Crea una función que reciba un número y devuelva el mes en esa posición



Solución de Funciones Python

```
>>> def numero_mes(mes):
...     if mes in ['ENERO',
... 'MARZO', 'MAYO', 'JULIO', 'AGOSTO',
... 'OCTUBRE', 'DICIEMBRE']:
...         return 31
...     elif mes in ['ABRIL',
... 'JUNIO', 'SEPTIEMBRE', 'NOVIEMBRE']:
...         return 30
...     else:
...         return 28
...
>>> numero_mes('ENERO')
31
>>> numero_mes('MAYO')
31
>>> numero_mes('FEBRERO')
28
>>> numero_mes('NOVIEMBRE')
30
```

```
>>> def nombre(name):
...     if type(name) == str:
...         return 'Es un string: ' + name
...     else:
...         return 'Error'
>>> nombre('hola')
Es un string: hola'
>>> nombre(1)
'Error'
>>> >>> def mes_posicion(posicion):
...     meses=['ENERO', 'FEBRERO',
... 'MARZO', 'ABRIL', 'MAYO', 'JUNIO',
... 'JULIO', 'AGOSTO', 'SEPTIEMBRE',
... 'OCTUBRE', 'NOVIEMBRE',
... 'DICIEMBRE']
...     return meses[posicion-1]
...
>>> mes_posicion(6)
'JUNIO'
>>> mes_posicion(1)
'ENERO'
```



Bucles: FOR

- Los bucles son los elementos de un lenguaje de programación que nos permiten repetir la ejecución de bloques de instrucciones, en Python contamos con “for” y con “while”.
- Utilizaremos un for para repetir un número determinado de veces una serie de instrucciones:

```
sumatorio = 0  
  
for x in range(10):  
    sumatorio += x
```

- Es decir, for lo que nos va a hacer es ir iterando sobre los distintos elementos de la lista que le pasemos, esa variable después de “in” es cualquier tipo de dato sobre el que se pueda iterar: lista, tupla, cadena ...



Ejercicio Bucles: FOR

1. Crea una función que devuelva una lista con los números múltiplos de 3 entre el 0 y el 90.
2. Crea una función que recorra los meses y devuelva aquellos que contienen una letra 'E'.



Solución Bucles: FOR

```
>>> inicio=0
>>> fin=90
>>> def multiples(inicio, fin):
...     vec=[]
...     for i in range(inicio, fin):
...         resto = i % 3
...         if resto == 0:
...             vec.append(i)
...     return vec
...
>>> vec=[]
>>> multiples(0,90)
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33,
36, 39, 42, 45, 48, 51, 54, 57, 60, 63,
66, 69, 72, 75, 78, 81, 84, 87]
```

```
>>> meses=['ENERO','FEBRERO',
'MARZO','ABRIL','MAYO','JUNIO',
'JULIO','AGOSTO','SEPTIEMBRE',
'OCTUBRE','NOVIEMBRE',
'DICIEMBRE']
>>> def meses_con_E(meses):
...     vec=[]
...     for mes in meses:
...         if 'E' in mes:
...             vec.append(mes)
...     return vec
...
>>> meses_con_E(meses)
['ENERO','FEBRERO','SEPTIEMBRE',
'OCTUBRE','NOVIEMBRE',
'DICIEMBRE']
>>>
```



Ejercicio Bucle FOR + diccionario

A partir de una lista de tuplas con el siguiente formato:

```
[ (Alvaro, Salamanca), (Mario, Madrid), (Marta, Salamanca), (Juan, Avila),  
(Pedro, Madrid), (Susana, Soria), (Martin, Valladolid), (Mario, Valladolid), (Jorge,  
Palencia) ]
```

Crea un programa que genere un diccionario de ciudades, y que para cada ciudad almacene los nombres de las distintas personas asociadas a esa ciudad



Solución Ejercicio Bucle FOR + diccionario

```
>>> a=[('Alvaro', 'Salamanca'),('Mario', 'Madrid'),('Marta', 'Salamanca'),('Juan', 'Avila'), ('Pedro', 'Madrid'),('Susana', 'Soria'),('Martin', 'Valladolid'),('Mario', 'Valladolid'),('Jorge', 'Palencia')]
```

```
>>> dic={}
```

```
>>> for nombre, ciudad in a:
```

```
    if ciudad not in dic:
```

```
        dic[ciudad] = nombre
```

```
    else:
```

```
        vec = []
```

```
        vec.append(dic[ciudad])
```

```
        vec.append(nombre)
```

```
        dic[ciudad] = vec
```

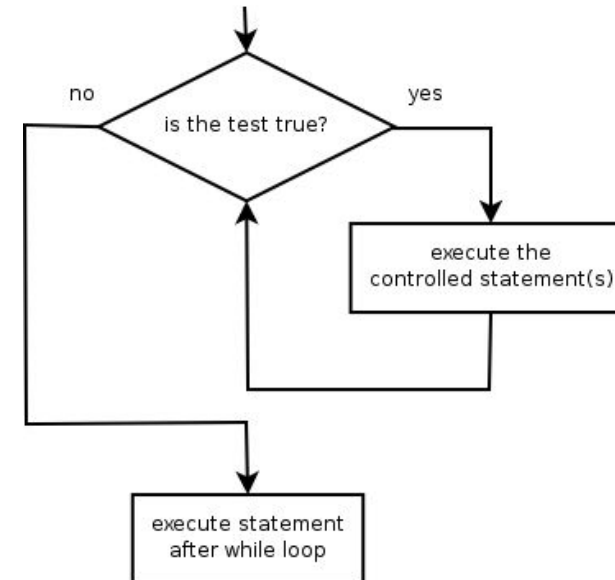
```
>>>> {'Madrid': ['Mario', 'Pedro'], 'Valladolid': ['Martin', 'Mario'], 'Avila': 'Juan', 'Salamanca':  
['Alvaro', 'Marta'], 'Soria': 'Susana'}{'Salamanca': ['Alvaro', 'Marta'], 'Palencia': 'Jorge', 'Madrid':  
['Mario', 'Pedro'], 'Avila': 'Juan', 'Soria': 'Susana', 'Valladolid': ['Martin', 'Mario']}
```



Bucles: While

- Utilizaremos la sentencia while para definir un bucle condicional, se ejecutará mientras se cumpla una determinada condición.

```
continuar = True  
  
while continuar:  
    do this  
    if esto:  
        continuar = False
```



Ejercicio Bucles: While

1. Crea una función que devuelva una lista con los números múltiplos de 3 entre el 0 y el 90 utilizando un bucle while.



Solución Bucles: While

```
inicio=0
fin=90
def multiples(inicio, fin):
    dd=[]
    i = inicio
    while i < fin:
        resto = i % 3
        if resto == 0:
            dd.append(i)
        i += 1
    return dd

>> multiples(inicio, fin)
```



Pidiendo información: raw_input()

- Al igual que podemos mostrar información, también podemos solicitar información por pantalla, para ello utilizaremos el comando `input()`

```
>>> nombre = input("Como te llamas?: ")
Como te llamas?: Pepe
>>> print "Hola " + nombre
Hola Pepe
```



PEP-257 Estilo de documentación

- La especificación de Python tiene reservada la primera línea después de la definición de una función o clase para añadir la cadena de documentación.
- Se utilizan las triples comillas para indicar este tipo de comentarios

```
def suma(a, b):  
    """  
    Esta funcion va a calcular la suma de los dos valores recibidos  
    Argumentos:  
    a -- primer numero del sumatorio  
    b -- Segundo numero del sumatorio  
    """  
    return a + b
```



Ejercicio PEP-257 Estilo de documentación

- Ejercicio:
 1. Crea una función producto, con resultado la multiplicación de dos números y con docstring
 2. Ejecuta
 3. Ejecuta help sobre cada una de ellas `help(producto)`



Solución PEP-257 Estilo de documentación

```
>>> def producto(a, b):  
...     """  
...     Esta funcion va a calcular el producto de los dos  
...     valores recibidos  
...     Argumentos:  
...     a primer numero del producto  
...     b Segundo numero del producto  
...     """  
...     return a * b  
...  
>>> def new_producto(a, b):  
...     return a * b  
...  
>>> producto(225,6)  
1350  
>>> new_producto(225,6)  
1350
```

```
>>> help(producto)  
Help on function producto in module __main__:  
  
producto(a, b)  
    Esta funcion va a calcular el producto de los dos  
    valores recibidos  
    Argumentos:  
    a primer numero del producto  
    b Segundo numero del producto  
  
>>> help(new_producto)  
Help on function new_producto in module __main__:  
  
new_producto(a, b)  
  
>>>
```



Manejo de Ficheros

Lectura de un fichero

- Se utiliza la función `open()`.

`open(name[, mode[, buffering]])`

siendo:

- *name*: El nombre del archivo,
- *mode*: El modo de apertura. (r, w, a)

- Para recorrer un fichero se utiliza un bucle for.
- Cuando se termina de utilizar el fichero se utiliza el método `close()`



Manejo de Ficheros

Lectura de un fichero

Dos maneras de recorrer un fichero:

```
>>> mi_archivo = open('/tmp/archivo')
>>> for linea in mi_archivo:
    .   print(linea)
>>> mi_archivo.close()
```

Más común con “Context Managers” (with)

```
>>> with open('/tmp/archivo') as mi_archivo:
    for linea in archivo:
        print(linea)
```



Manejo de Ficheros

Escritura en un fichero

- Se abre el fichero en modo lectura ('w', 'a').
- Existen el para escribir write().

```
>>> f = open('prueba.dat', 'w')  
>>> f.write("hola")  
>>> f.close()
```

```
>>> f = open('prueba.dat', 'w')  
>>> print >> f, "hola"  
>>> f.close()
```

- También se puede escribir mediante el comando print.



Ejercicio

Dado el archivo quijote.txt

1. Leer el contenido del fichero
2. Imprimir por pantalla la palabra más usada

