
Repaso de Python

Ciencias de Datos



Repaso Python

Instalación

Funciones

Strings

Listas y tuplas

Diccionarios

Ficheros

Modularidad

Clases

Python

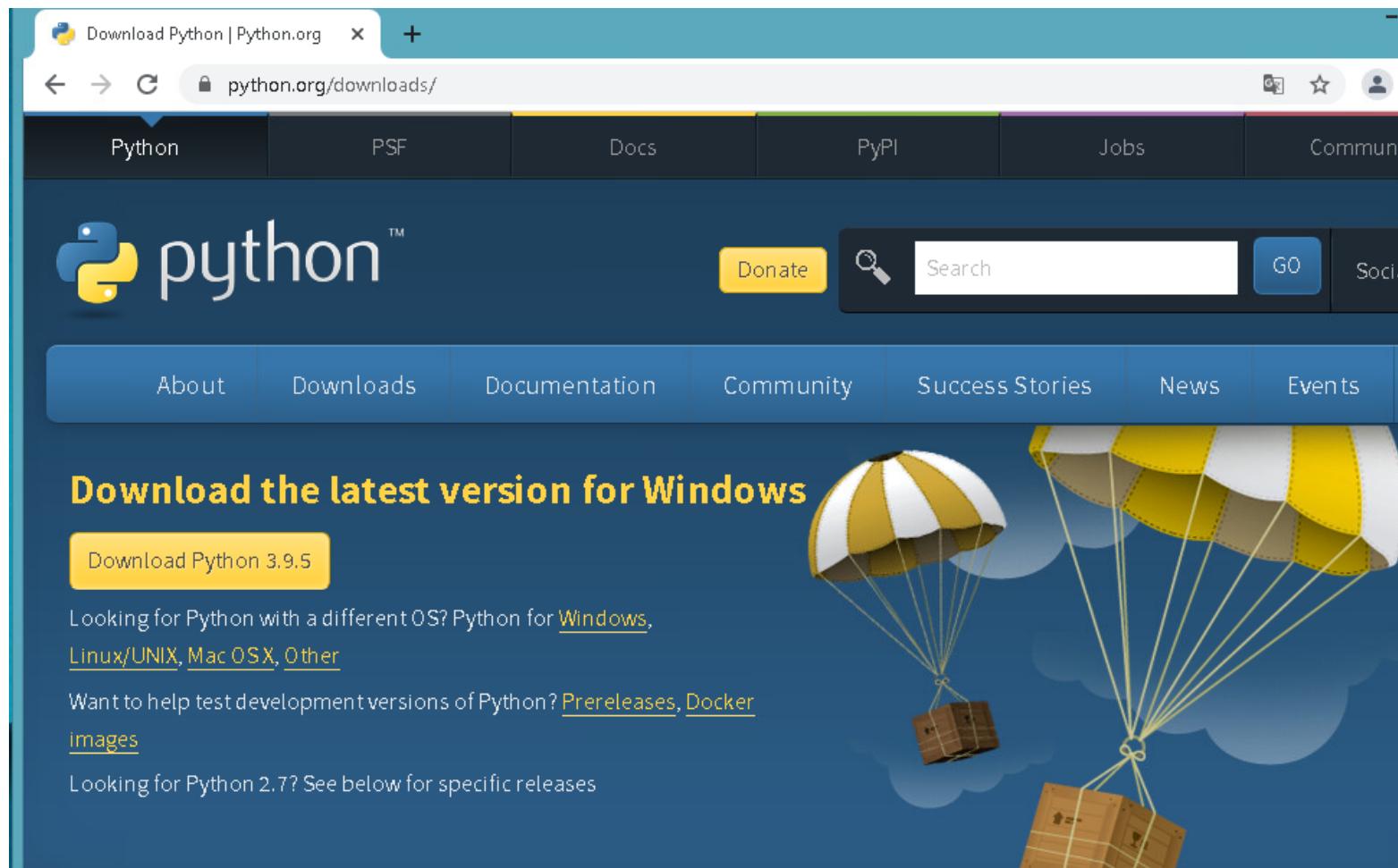
Python es un lenguaje de scripts

Python es un lenguaje de alto nivel sencillo de programar y ejecutar.

Python es un lenguaje de código abierto, ejecutable en sistemas operativos Windows, Linux, y Unix.

Descargar Python

www.python.org



Download Python | Python.org

python.org/downloads/

Python PSF Docs PyPI Jobs Community

python™

Donate Search GO Social

About Downloads Documentation Community Success Stories News Events

Download the latest version for Windows

[Download Python 3.9.5](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OSX](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#), [Docker images](#)

Looking for Python 2.7? See below for specific releases

A large graphic on the right side of the page shows two yellow and white striped parachutes descending from the sky, each carrying a wooden shipping crate.

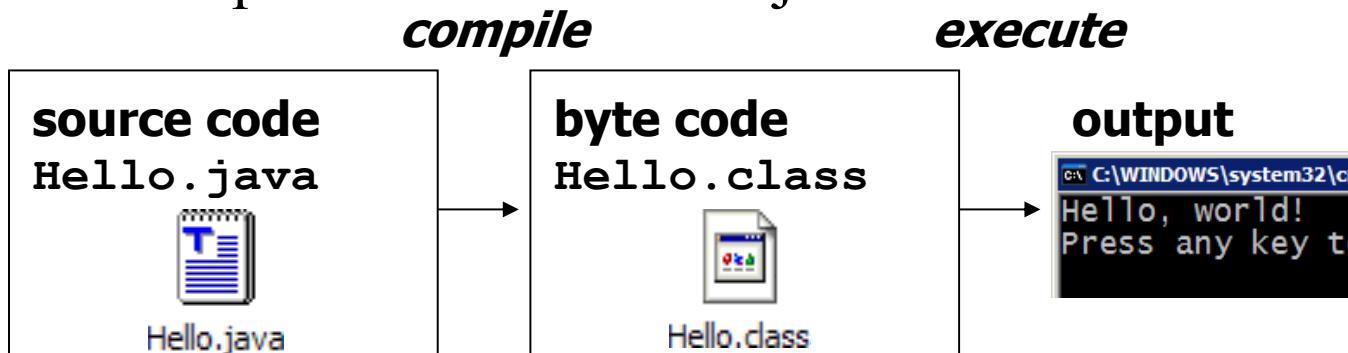
Instalar Python

Instalar Python y añadir python.exe al PATH del sistema

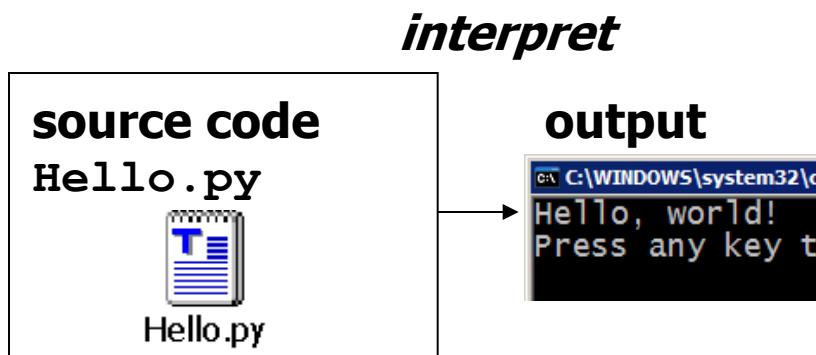


Compilado vs interpretado

- ◆ En muchos lenguajes de programación los programas deben ser compilados antes de ser ejecutados.



Python es directamente interpretado



Pero también puede ser compilado en packages

Python

- ◆ Python es un lenguaje interpretado, por lo que no necesita ser compilado ni linkado para ser ejecutado.
- ◆ El interprete puede usarse de manera interactiva o mediante scripts
- ◆ Python suele tener programas más cortos que otros lenguajes, como C++ o java por:
 - No tener que declarar el tipo de variables o argumentos
 - La agrupación (bloques de código) se hace con sangrado, sin necesidad de llaves.

Python interactivo

- ◆ Su ruta puede añadirse al PATH del sistema añadiéndose a dicha variable del sistema o en la actual sesión con

```
set path=%path%;C:\...\pythonxx;
```

- ◆ También se puede ejecutar desde línea de comandos con
`python -c command [arg]`

- ◆ O ejecutar scripts con

```
python -m module [arg] ...
```

Probar Python interactivo

```
C:\Windows\System32\cmd.exe - python
Microsoft Windows [Versión 10.0.17134.228]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3*4
14
>>> nombre = "Luis"
>>> nombre
'Luis'
>>> print("Hola ", nombre)
Hola Luis
>>>
```

Probar Python interactivo en pyspark

```
Welcome to
```

```
SPARK  
version 1.2.1
```

```
Using Python version 2.6.6 (r266:84292, Nov 21 2013 10:50:32)  
SparkContext available as sc.
```

```
>>> 2 + 3*4  
14  
>>> nombre = "Luis"  
>>> print(nombre)  
Luis  
>>> "Hola " + nombre  
'Hola Luis'  
>>> █
```

Python interactivo en Jupyter Notebook

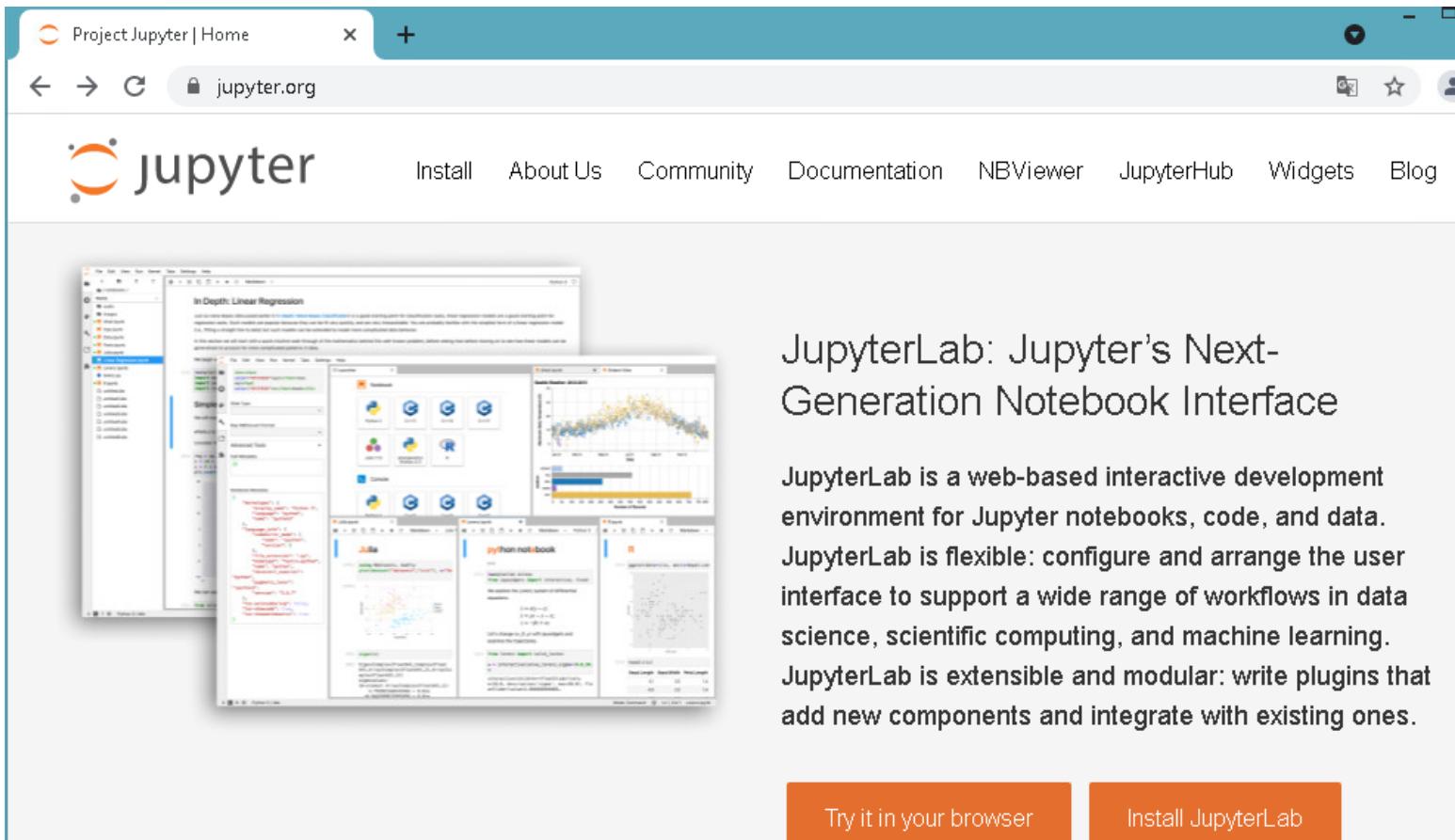
<https://jupyter.org/>

The screenshot shows the official Project Jupyter website at <https://jupyter.org>. The page features a large, central logo consisting of the word "jupyter" in a lowercase sans-serif font, surrounded by three overlapping orange semi-circles of increasing size from bottom-left to top-right. The background is white with a subtle, faint watermark-like pattern of various programming language icons (Python, JavaScript, R, etc.) scattered around the logo. At the top, there is a navigation bar with links for "Install", "About Us", "Community", "Documentation", "NBViewer", "JupyterHub", "Widgets", and "Blog". On the left side, there is a sidebar with the "jupyter" logo and a "Project Jupyter | Home" link. The main content area contains a brief description of the project's purpose.

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

Python interactivo en Jupyter Notebook

<https://jupyter.org/>

A screenshot of a web browser displaying the official Jupyter website at jupyter.org. The page features a header with the Jupyter logo, navigation links for Install, About Us, Community, Documentation, NBViewer, JupyterHub, Widgets, and Blog. Below the header, there is a large image showing multiple Jupyter notebooks and data visualizations, illustrating the platform's capabilities. The browser's address bar shows the URL jupyter.org.

JupyterLab: Jupyter's Next-Generation Notebook Interface

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones.

Try it in your browser

Install JupyterLab

Python interactivo en Jupyter Notebook

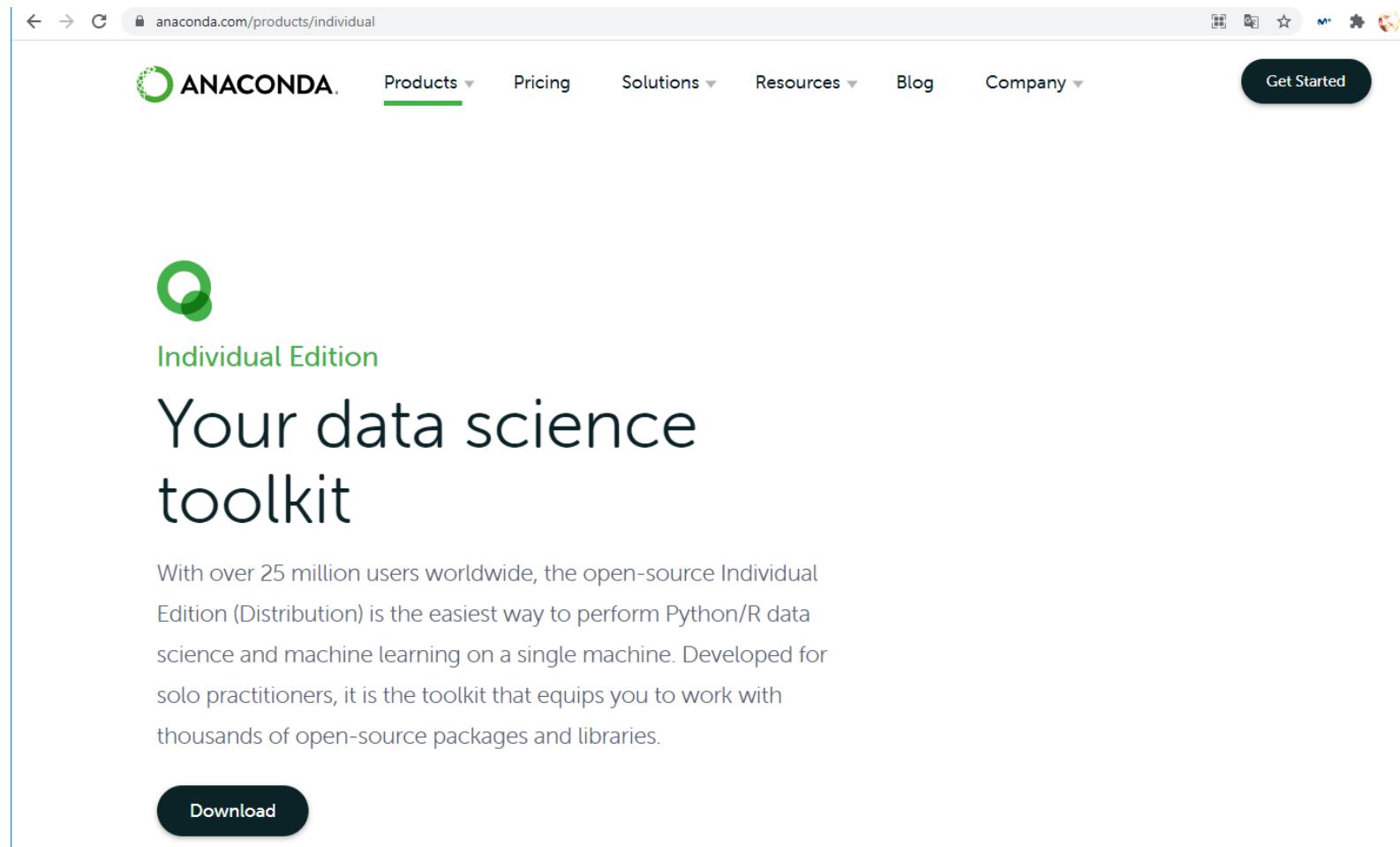
<https://jupyter.org/try>

The screenshot shows a web browser window with the URL jupyter.org/try in the address bar. The page title is "Try Jupyter". The main content area contains three blue rectangular boxes, each with a title, an icon, and a brief description.

- Try Classic Notebook**: Features the Python logo icon. Description: "A tutorial introducing basic features of Jupyter notebooks and the IPython kernel using the classic Jupyter Notebook interface."
- Try JupyterLab**: Features the Jupyter logo icon. Description: "JupyterLab is the new interface for Jupyter notebooks and is ready for general use. Give it a try!"
- Try Jupyter with Julia**: Features the Julia logo icon. Description: "A basic example of using Jupyter with Julia."

Install Anaconda

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



The screenshot shows the Anaconda website at the URL <https://www.anaconda.com/products/individual>. The page features a dark header with the Anaconda logo and navigation links for Products, Pricing, Solutions, Resources, Blog, and Company. A prominent green button labeled "Get Started" is located in the top right corner. Below the header, there's a large green "Q" icon followed by the text "Individual Edition". The main headline reads "Your data science toolkit". A descriptive paragraph explains that the Individual Edition is the easiest way to perform Python/R data science and machine learning on a single machine, developed for solo practitioners. At the bottom, a "Download" button is visible.

anaconda.com/products/individual

ANACONDA. Products ▾ Pricing Solutions ▾ Resources ▾ Blog Company ▾ Get Started

Individual Edition

Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Download

Anaconda Navigator

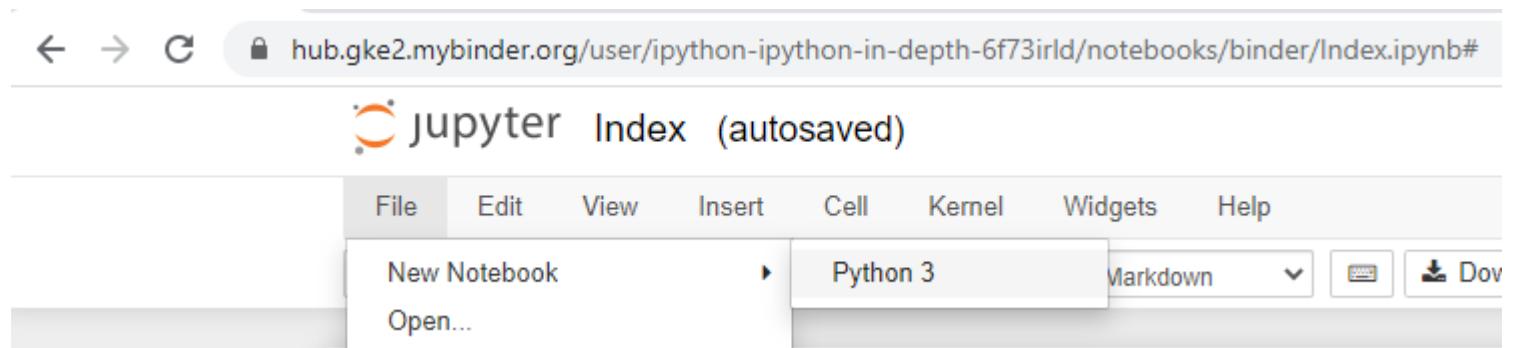
<https://www.anaconda.com/products/individual>

The screenshot shows the Anaconda Navigator application window. On the left is a sidebar with icons for Home, Environments, Learning, Community, Documentation, and Developer Blog. The main area displays a grid of eight applications:

- JupyterLab** (2.2.6): An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. Includes a "Launch" button.
- Notebook** (6.1.4): Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. Includes a "Launch" button.
- Glueviz** (1.0.0): Multidimensional data visualization across files. Explore relationships within and among related datasets. Includes an "Install" button.
- Orange 3** (3.26.0): Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox. Includes an "Install" button.
- Qt Console** (4.7.7): PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. Includes an "Install" button.
- RStudio** (1.1.456): A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks. Includes an "Install" button.
- Spyder** (4.1.5): Scientific PYthon Development EnvIRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features. Includes an "Install" button.
- VS Code** (1.49.3): Streamlined code editor with support for development operations like debugging, task running and version control. Includes an "Install" button.

At the top right are "Upgrade Now" and "Sign in to Anaconda Cloud" buttons, and a "Refresh" button. The top center shows the title "Anaconda Navigator".

Python interactivo en Jupyter Notebook



A screenshot of a Jupyter Notebook interface titled "Untitled". The title bar indicates "Last Checkpoint: hace un minuto (unsaved changes)". The top navigation bar is identical to the previous screenshot. Below the toolbar, there is a row of icons for file operations (New, Open, Save, etc.) and a "Run" button. The main area contains a code cell labeled "In [1]:" with the Python command `print("Hola Mundo")`. The output of the cell is "Hola Mundo". A second, empty code cell is visible below it, labeled "In []:".

```
In [1]: print("Hola Mundo")
Hola Mundo
```

```
In [ ]:
```

Instalación Jupyter Notebook: pip install notebook

```
C:\Users\Luis>pip install notebook
Collecting notebook
  Downloading https://files.pythonhosted.org/packages/39/b6/8135d31209691ce/notebook-6.4.0-py3-none-any.whl (9.5MB)
    100% |██████████| 9.5MB 1.0MB/s
Collecting nbformat (from notebook)
  Downloading https://files.pythonhosted.org/packages/e7/c7/dd50978c637a7af/nbformat-5.1.3-py3-none-any.whl (178kB)
    100% |██████████| 184kB 4.9MB/s
Collecting terminado>=0.8.3 (from notebook)
  Downloading https://files.pythonhosted.org/packages/07/ea/0b2b2a16748428e/terminado-0.10.0-py3-none-any.whl
Collecting prometheus-client (from notebook)
  Downloading https://files.pythonhosted.org/packages/22/f7/f6e1676521ce7e3/prometheus_client-0.10.1-py2.py3-none-any.whl (55kB)
    100% |██████████| 61kB 5.1MB/s
Collecting Send2Trash>=1.5.0 (from notebook)
```

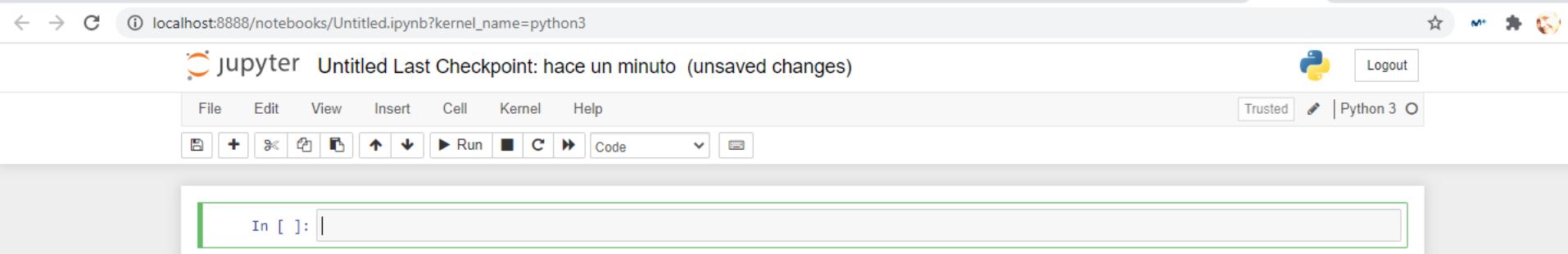
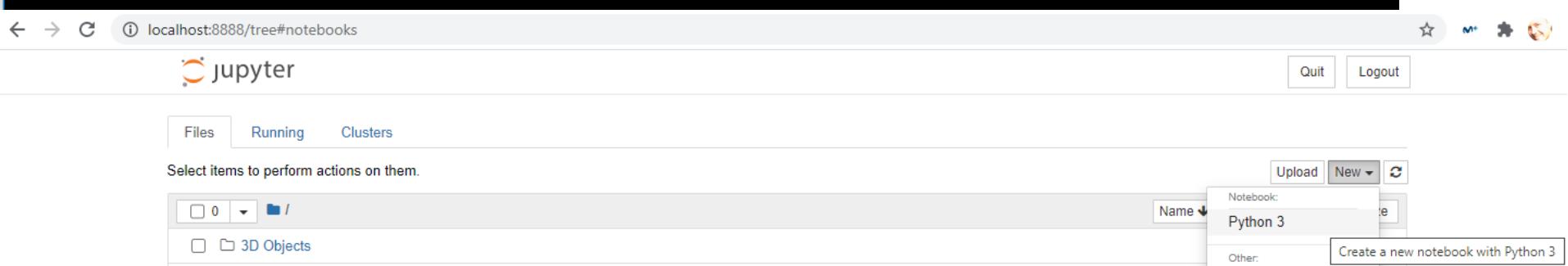
Instalar notebook (si falla instalación)

```
C:\Users\Luis>python -m pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/cd/82
/pip-21.1.2-py3-none-any.whl (1.5MB)
    100% |██████████| 1.6MB 3.3MB/s
```

```
C:\Users\Luis>pip install notebook
Collecting notebook
  Downloading https://files.pythonhosted.org/packages/39/b6/8135d31209691ce
/notebook-6.4.0-py3-none-any.whl (9.5MB)
    100% |██████████| 9.5MB 1.0MB/s
Collecting nbformat (from notebook)
  Downloading https://files.pythonhosted.org/packages/e7/c7/dd50978c637a7af
/nbformat-5.1.3-py3-none-any.whl (178kB)
    100% |██████████| 184kB 4.9MB/s
Collecting terminado>=0.8.3 (from notebook)
  Downloading https://files.pythonhosted.org/packages/07/ea/0b2b2a16748428e
/terminado-0.10.0-py3-none-any.whl
Collecting prometheus-client (from notebook)
  Downloading https://files.pythonhosted.org/packages/22/f7/f6e1676521ce7e3
/prometheus_client-0.10.1-py2.py3-none-any.whl (55kB)
    100% |██████████| 61kB 5.1MB/s
Collecting Send2Trash>=1.5.0 (from notebook)
```

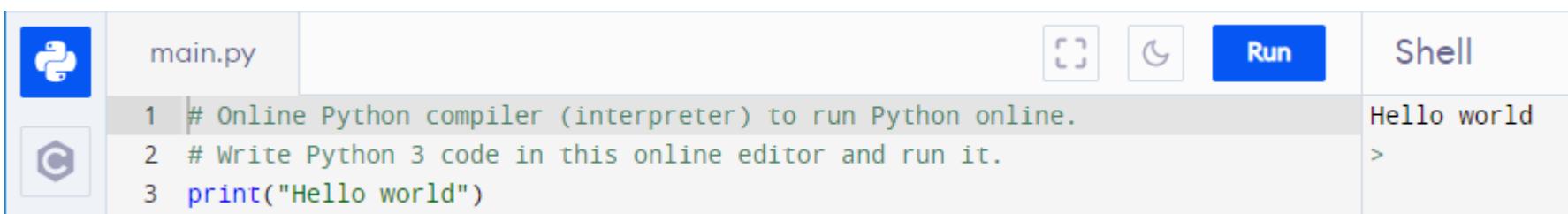
Run jupyter notebook: jupyter notebook

```
C:\Users\Luis>jupyter notebook
[I 01:30:45.658 NotebookApp] Writing notebook server cookie secret to C:\Users\Luis\AppData\Roaming\jupyter\runtime\notebook_cookie_secret
[I 01:30:46.639 NotebookApp] Serving notebooks from local directory: C:\Users\Luis
[I 01:30:46.640 NotebookApp] Jupyter Notebook 6.4.0 is running at:
[I 01:30:46.640 NotebookApp] http://localhost:8888/?token=6d3886649ae7a55220efd2b92aa15d2874f54c060df4862c
[I 01:30:46.640 NotebookApp] or http://127.0.0.1:8888/?token=6d3886649ae7a55220efd2b92aa15d2874f54c060df4862c
[I 01:30:46.641 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 01:30:46.730 NotebookApp]
```



Python web editor

<https://www.programiz.com/python-programming/online-compiler/>

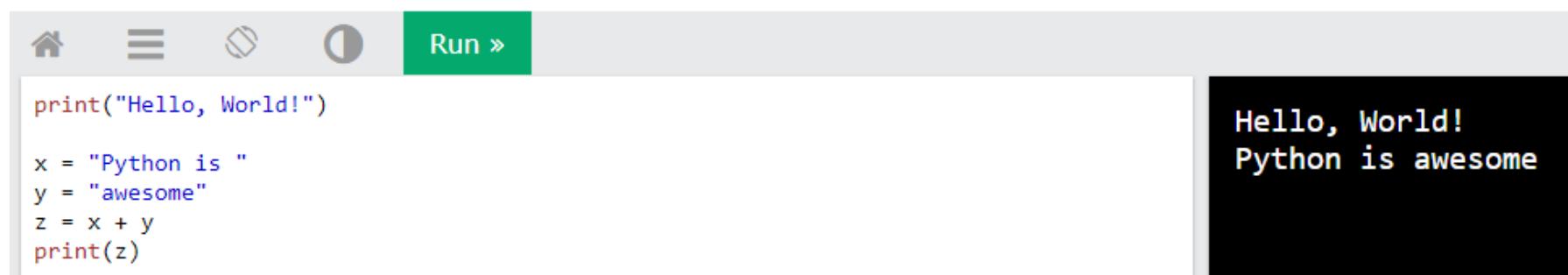


The screenshot shows a web-based Python editor. On the left, there are two icons: a blue Python logo and a grey C logo. The main area has a title bar with 'main.py' and tabs for 'Run' and 'Shell'. Below the title bar is a code editor with the following Python code:

```
1 # Online Python compiler (interpreter) to run Python online.
2 # Write Python 3 code in this online editor and run it.
3 print("Hello world")
```

To the right of the code editor is an output window showing the result of running the code: "Hello world".

https://www.w3schools.com/python/trypython.asp?filename=demo_compiler



The screenshot shows a web-based Python compiler from W3Schools. At the top, there are several icons: a house, three horizontal lines, a gear, and a circular arrow. Next to them is a green 'Run' button with the text 'Run >'. Below the icons is a code editor window containing the following Python code:

```
print("Hello, World!")

x = "Python is "
y = "awesome"
z = x + y
print(z)
```

On the right side, there is a results window with a black background and white text, displaying the output of the code: "Hello, World! Python is awesome".

Python web editor

<https://replit.com/languages/python3>

A screenshot of the Replit Python web editor interface. It shows a dropdown menu set to 'Python'. Below it is a code editor with the following Python code:

```
1 print('Hello, world!')
```

The output window to the right displays the result:

```
Hello, world!
```

A screenshot of the Online Python web editor interface. The code editor contains the following Python script named 'main.py':

```
1 # Online Python - IDE, Editor, Compiler, Interpreter
2
3
4 def sum(a, b):
5     return (a + b)
6
7 a = int(input('Enter 1st number: '))
8
9 b = int(input('Enter 2nd number: '))
10
11 print(f'Sum of {a} and {b} is {sum(a, b)}')
```

The status bar at the bottom indicates 'Ln: 10, Col: 1'. Below the code editor are three buttons: 'Run' (highlighted), 'Share', and 'Command Line Arguments'. The terminal window at the bottom shows the interaction:

```
Enter 1st number:
3
Enter 2nd number:
5
Sum of 3 and 5 is 8
```

Python web editor

<https://trinket.io/features/python3>

The screenshot shows the Trinket Python web editor interface. On the left, there's a code editor with two tabs: 'main.py' and 'inflammation-01.csv'. The 'main.py' tab contains the following Python code:

```
1 import numpy
2 import matplotlib.pyplot
3
4 # Load data
5 data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
6 # Make Figure
7 fig = matplotlib.pyplot.figure(figsize=(5.0, 7.0))
8
9 # Create subplots in 3 rows and 1 column
10 axes1 = fig.add_subplot(3, 1, 1)
11 axes2 = fig.add_subplot(3, 1, 2)
12 axes3 = fig.add_subplot(3, 1, 3)
13
14 # Plot and label the average, max, and min of the data
15 axes1.set_ylabel('average')
16 axes1.plot(data.mean(axis=0))
17
18 axes2.set_ylabel('max')
19 axes2.plot(data.max(axis=0))
20
21 axes3.set_ylabel('min')
22 axes3.plot(data.min(axis=0))
23
24 fig.tight_layout()
25
26 matplotlib.pyplot.savefig("plot.png")
27
28 # Code adapted from Software Carpentry. Check out the full lesson here:
29 # http://swcarpentry.github.io/python-novice-inflammation/01-numnv.html
```

On the right, there are two plots generated by the code. The top plot, titled 'average', shows a line graph with data points at x-coordinates from 0 to 40. The y-axis ranges from 0.0 to 12.5. The data shows an overall upward trend with some fluctuations, peaking around x=20. The bottom plot, titled 'max', shows a line graph with data points at x-coordinates from 0 to 40. The y-axis ranges from 0 to 20. The data shows a single sharp peak at x=20.

Crear script de Python

Crear una carpeta c:/Python/scripts

Crear un documento HolaMundo.py

The screenshot shows the Notepad++ interface with the following details:

- Title Bar:** C:\Python\scripts\HolaMundo.py - Notepad++
- Menu Bar:** Archivo, Editar, Buscar, Vista, Codificación, Lenguaje, Cor
- Toolbar:** Includes icons for New, Open, Save, Print, Cut, Copy, Paste, Find, Replace, and others.
- Tab Bar:** Shows two tabs: "new 1" and "HolaMundo.py". The "HolaMundo.py" tab is active.
- Code Editor:** Displays the following Python code:

```
1 nombre = "Luis"
2 print ("Hola ", nombre)
```

The word "print" is highlighted in blue.

Ejecutar script de Python

Ir a c:/Python/scripts en una línea de comandos (cmd)

Interpretar el script con: python HolaMundo.py

 Símbolo del sistema

```
Microsoft Windows [Versión 10.0.17134.228]
(c) 2018 Microsoft Corporation. Todos los de

C:\Users\Luis>cd C:/Python/scripts

C:\Python\scripts>python HolaMundo.py
Hola Luis
```

Python IDE

El uso de IDEs (integrated development environment) como **PyCharm**, **PyDev para Eclipse**, **Spyder**, **Rodeo**, **Atom**, **Jupiter Notebook** facilita el uso de scripts en ficheros con extensión .py

Contiene:

Ventana interactiva

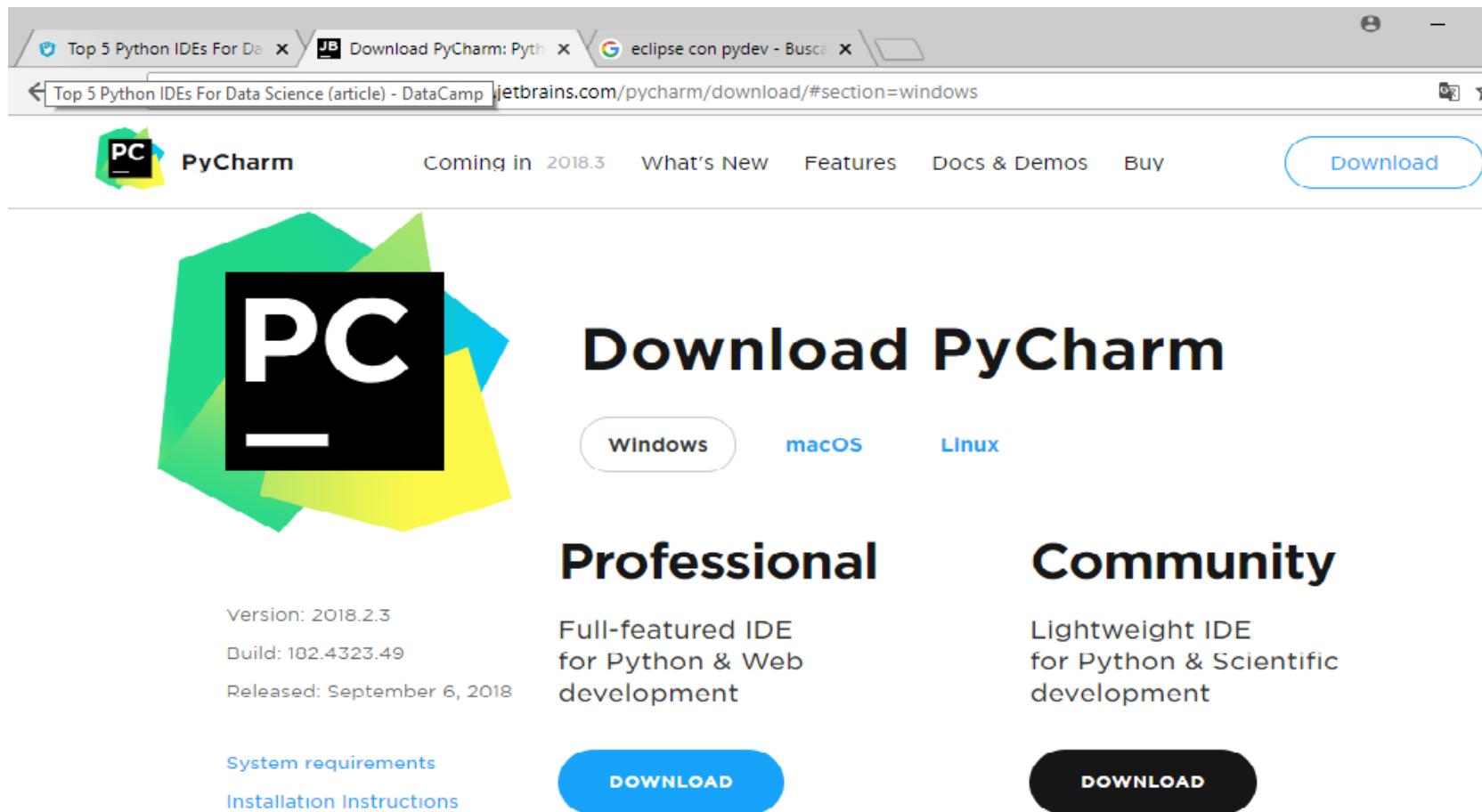
Herramientas de Edición

Comprobación de sintaxis

Debug

Instalar PyCharm

Descargar PyCharm Community



The screenshot shows a web browser window with three tabs open:

- Top 5 Python IDEs For Data Science - DataCamp
- JB Download PyCharm: Python
- eclipse con pydev - Busca

The main content area displays the PyCharm download page. The URL in the address bar is jetbrains.com/pycharm/download/#section=windows.

The page features the PyCharm logo (a stylized 'PC' inside a hexagon) and navigation links for "Coming in 2018.3", "What's New", "Features", "Docs & Demos", and "Buy". A prominent blue "Download" button is located on the right.

The central heading is "Download PyCharm". Below it are three buttons for "Windows", "macOS", and "Linux".

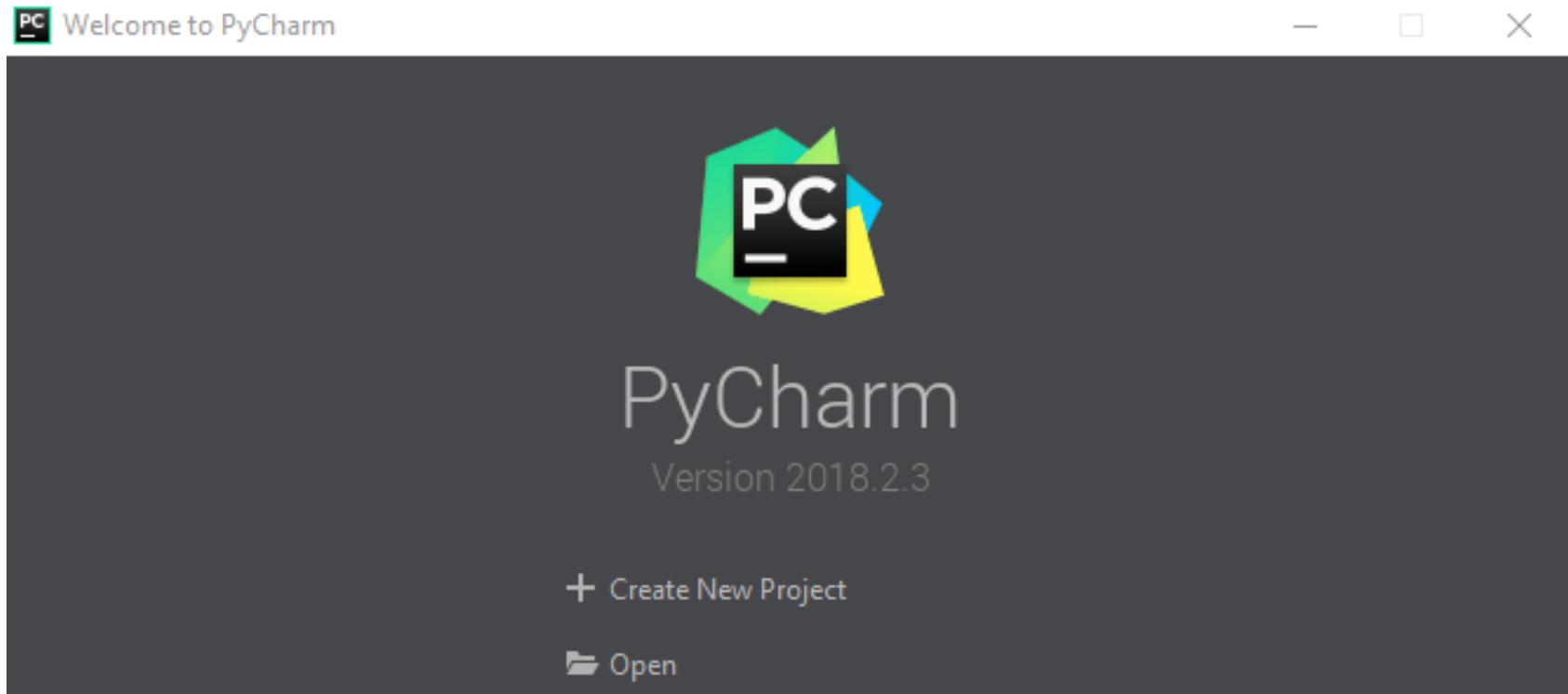
Two main product sections are shown:

- Professional**: Described as a "Full-featured IDE for Python & Web development". It includes a "System requirements" link and a blue "DOWNLOAD" button.
- Community**: Described as a "Lightweight IDE for Python & Scientific development". It includes a black "DOWNLOAD" button.

At the bottom left, there are links for "Version: 2018.2.3", "Build: 102.4323.49", and "Released: September 6, 2018".

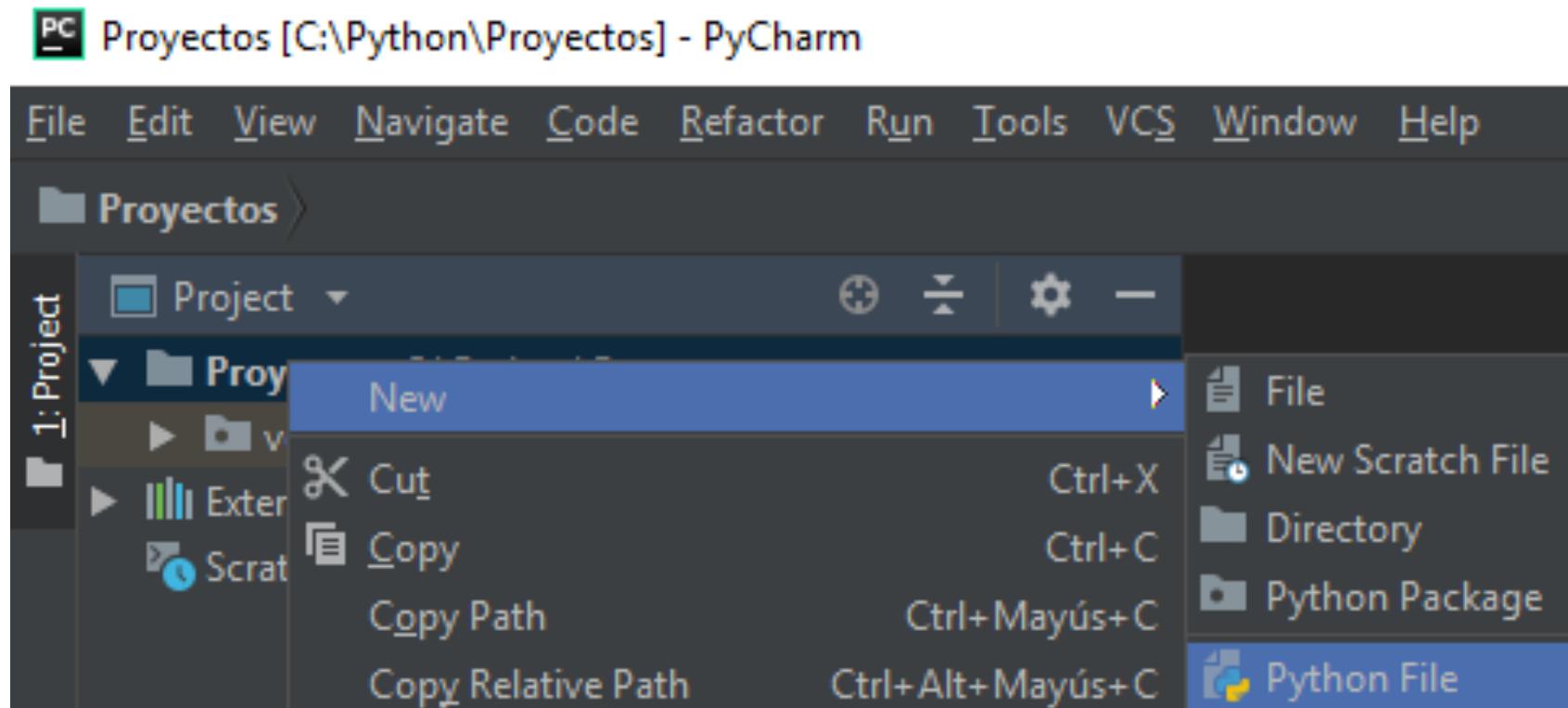
Crear un proyecto

Crear una proyecto (carpeta) en c:\Python\Proyecto

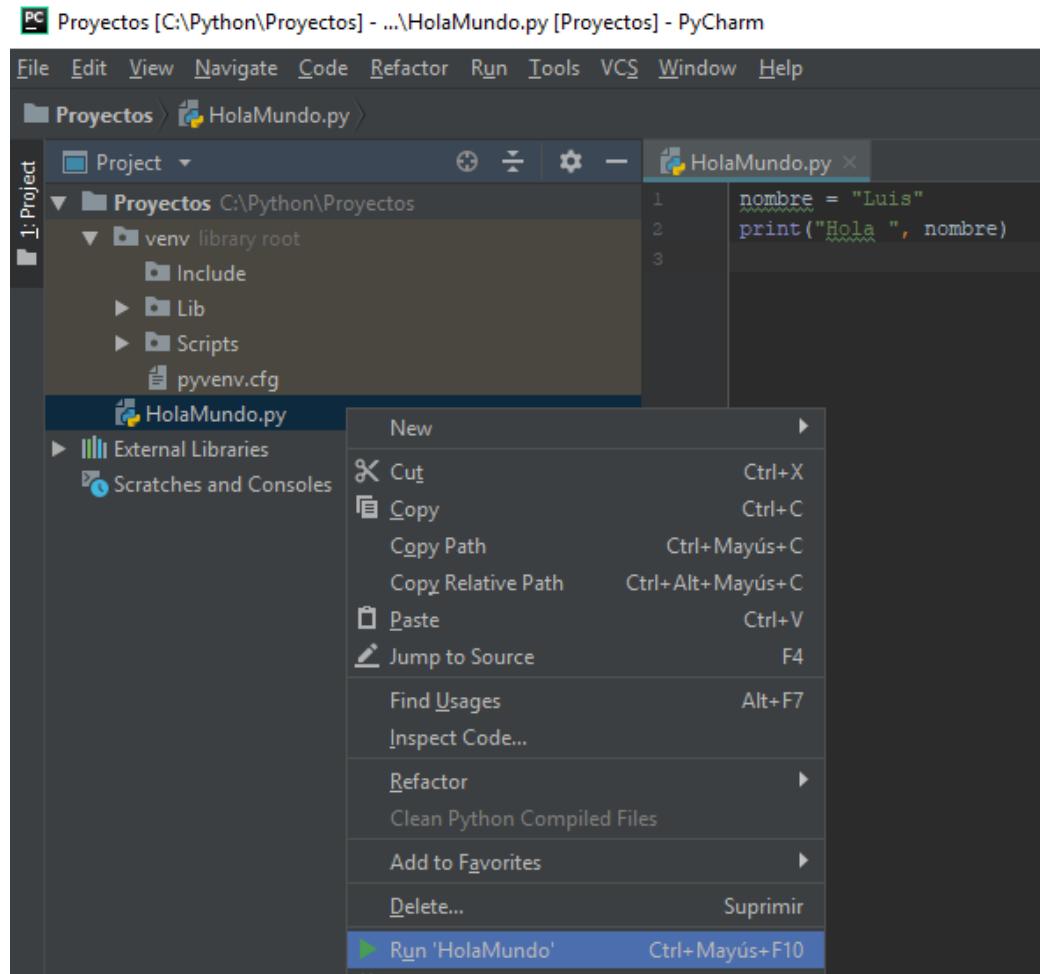


Crear un fichero en el proyecto

Crear un fichero Ejer1.1.py en el proyecto



Ejecutar un programa en el proyecto



Ver salida en la consola

The screenshot shows the PyCharm IDE interface. The title bar reads "Proyectos [C:\Python\Proyectos] - ...\\HolaMundo.py [Proyectos] - PyCharm". The menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The breadcrumb navigation shows "Proyectos > HolaMundo.py". The Project tool window on the left shows a file structure under "Proyectos": "venv library root" contains "Include", "Lib", "Scripts", and "pyvenv.cfg"; "HolaMundo.py" is selected and highlighted. The Editor tab "HolaMundo.py" displays the following code:

```
1 nombre = "Luis"
2 print("Hola ", nombre)
3
```

The Run tool window at the bottom shows the command "C:\Python\Proyectos\venv\Scripts\python.exe C:/Python/Proyectos/HolaMundo.py" and the output "Hola Luis". Below the output, it says "Process finished with exit code 0".

Spyder

The screenshot shows the Spyder IDE interface with the following components:

- Code Editor:** On the left, the file `ejemplo.py` is open, displaying Python code. The code includes functions for calculating distances between colors and printing distances for a given image.
- Plots:** In the center-right, there is a plot pane showing a 2D heatmap or image. The x-axis ranges from -0.5 to 1.5, and the y-axis ranges from -0.5 to 1.5. The plot is divided into four quadrants: top-left (blue), top-right (orange), bottom-left (green), and bottom-right (dark blue). To the right of the plot, there is a vertical stack of smaller versions of the same image.
- Console:** At the bottom, the IPython Console shows the following output:

```
In [2]: runfile('/Users/luisgarmendia/Downloads/ejemplo.py', wdir='/Users/luisgarmendia/Downloads')
Reloaded modules: image
File "unknowno", line 51
    print (range(2,1))
^
IndentationError: unexpected indent

In [3]: runfile('/Users/luisgarmendia/Downloads/ejemplo.py', wdir='/Users/luisgarmendia/Downloads')
[32288, 32288]
[60500, 60500]

[164, 164]
[200, 200]

[74064, 99664]
[168200, 158400]
range(2, 1)

In [4]: runfile('/Users/luisgarmendia/Downloads/ejemplo.py', wdir='/Users/luisgarmendia/Downloads')
[32288, 32288]
```
- Status Bar:** At the bottom right, the status bar displays: Spyder. Update available internal (Python 3.9.14) Completions: Internal ✓ LSP: Python Line 52, Col 22 UTF-8 LF RW Mem 61%.

Python tutor (debugging)

<https://www.pythontutor.com>

Python 2.7

```
1 def listSum(numbers):
2     if not numbers:
3         return 0
4     else:
5         (f, rest) = numbers
6         return f + listSum(rest)
7
8 myList = (1, (2, (3, None)))
9 total = listSum(myList)
```

[Edit code](#)

→ line that has just executed

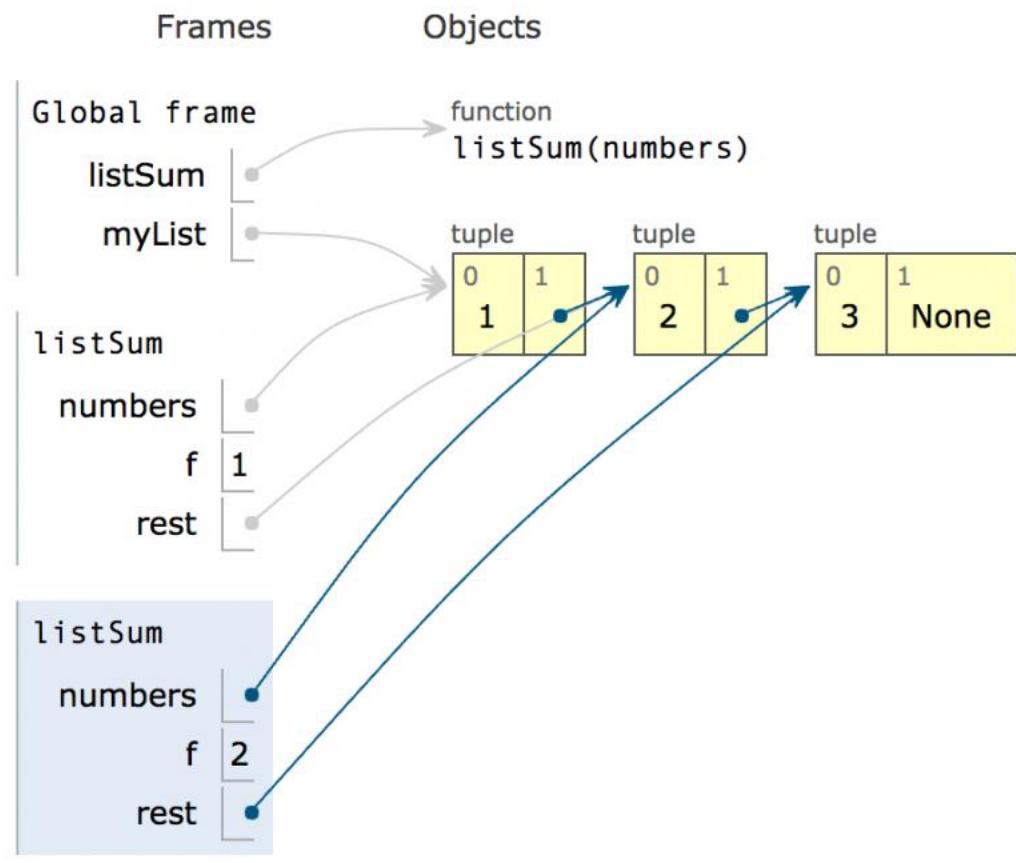
→ next line to execute

< Back

Step 11 of 22

Forward >

Visualized using [Python Tutor](#) by Philip Guo



Python 2 vs Python 3

La sentencia **print** pasa a ser función:

```
print(args, end="\n", sep=" ")
```

Ejemplos:

Python 2.x

```
print "La respuesta es", 2*2
print x,
print()
print >>sys.stderr, "fatal error"
```

Pyyhon 3.x

```
print("La respuesta es", 2*2)
print(x, end=" ") # (no saltar)
print
print("fatal error", file=sys.stderr)
```

El lenguaje Python: Sintaxis

- Tipos de datos básicos
- Tipos compuestos, secuencias
- Operadores
- Condicionales
- Control de flujo
- Bucles
- Funciones

Sintaxis de Python

Python es sensible a mayúsculas-minusculas (case-sensitive), tanto para variables (no es lo mismo ‘x’ que ‘X’) como para palabras reservadas (no es lo mismo ‘print’ que ‘Print’).

Se termina una sentencia pulsando Enter, o empezando una nueva línea.

Para sentencias largas, el carácter de continuación de línea es (\). Una excepción es si hay paréntesis o corchetes abiertos, entonces Python entiende que no es final de sentencia

Se requiere un sangrado de 4 espacios en bloques de bucles, sentencias if/then o sentencias try/except.

Una línea se desinterpreta con ‘#’ al principio de una línea para hacer comentarios.

Nombres de variables

Los nombres de las variables:

- ◆ Son sensibles a mayúsculas-minúsculas
- ◆ No pueden contener espacios
- ◆ No pueden empezar por número
- ◆ Convención: empezar con minúscula.
 - Ejemplo: miVariable
- ◆ No puedes ser palabras reservadas como ‘import’ o ‘print’.

Tipos de variables

Otros lenguajes de programación como C++ o java son fuertemente tipados, lo que obliga a declarar el tipo de la variable:

```
int x = 2;
```

```
Bool b=true
```

```
String s = "Hola Mundo"
```

Python no es un leguaje fuertemente tipado. A una variable se le pueden asignar objetos de distintos tipos.

Tipos de Variables

Escribir en una shell

```
>>>x=5  
>>>type(x)  
<class 'int'>  
>>>x="this is text"  
>>>type(x)  
<class 'str'>  
>>>x=5.0  
>>>type(x)  
<class 'float'>
```

Tipos Numéricos

- ◆ Numéricos (integer, long integer, floating-point, and complex)

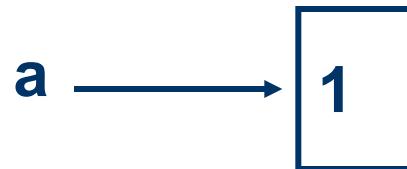
```
>>> x = 4  
>>> int(x)  
4  
>>> long(x)  
4L  
>>> float(x)  
4.0  
>>> complex(4, .2)  
(4+0.2j)
```

Operadores aritméticos

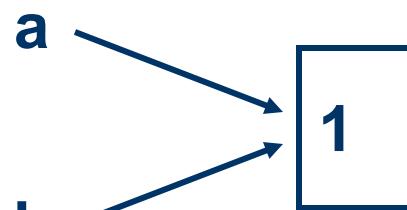
- + suma
- - resta
- / division
- * producto
- ** potencia
- % módulo (resto de la división entera)

Las variables son objetos

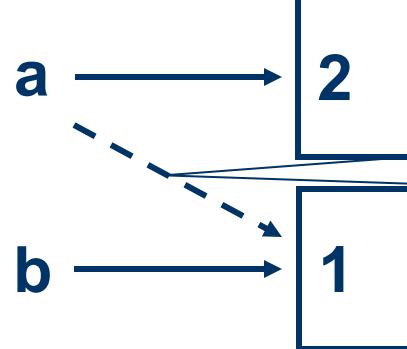
a = 1



b = a



a = a+1



new int object created by add operator (1+1)

old reference deleted by assignment (a=...)

Operadores lógicos

Expresiones lógicas

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

Expresiones lógicas combinadas con operadores lógicos

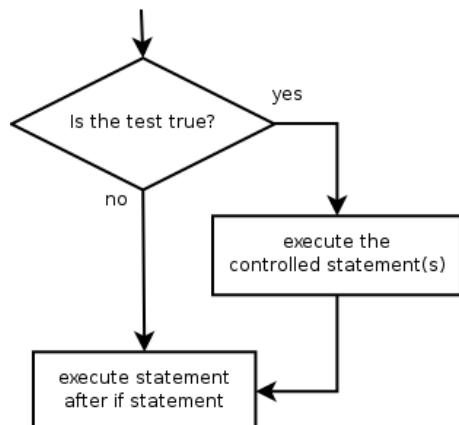
Operator	Example	Result
<code>and</code>	<code>9 != 6 and 2 < 3</code>	True
<code>or</code>	<code>2 == 3 or -1 < 5</code>	True
<code>not</code>	<code>not 7 > 0</code>	False

IF

```
if <condition>:  
    <statements>
```

```
x = 5
```

```
if x > 4:  
    print("x is greater than 4")
```



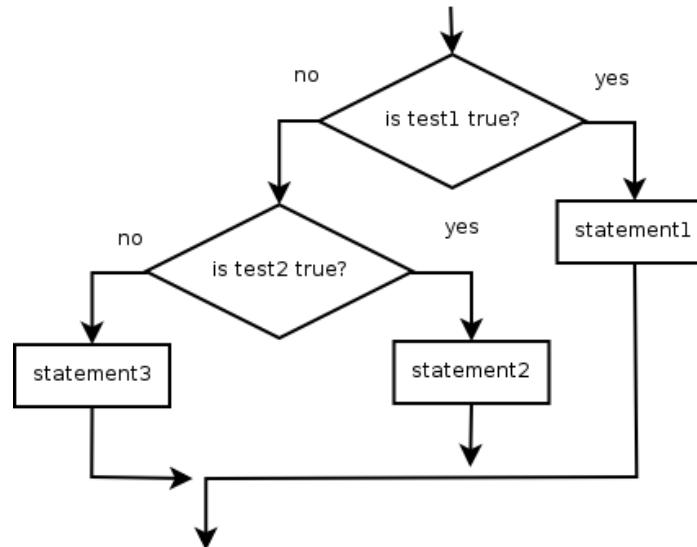
Bloques por tabulación

- ◆ Python usa tabulación (o espaciado) para mostrar estructura de bloques
 - Tabula una vez para indicar comienzo de bloque
 - Des-tabula para indicar el final del bloque

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

ELIF

```
if condition:  
    statements  
elif condition:  
    statements  
else:  
    statements
```



FOR

```
for variableName in groupOfValues:  
    <statements>
```

Ejemplo:

```
for x in range(1, 6):  
    print ("El cuadrado de ", x, " es ", x * x)
```

Iteradores

- ◆ Iteradores sobre colecciones:

```
for x in L:
```

- ◆ Iteradores sobre diccionarios:

```
for k in mydict.keys( ):
```

- ◆ Iteradores explícitos:

```
for x in [1, 1, 2, 3, 5, 8, 13]:
```

- ◆ Iteradores numéricos

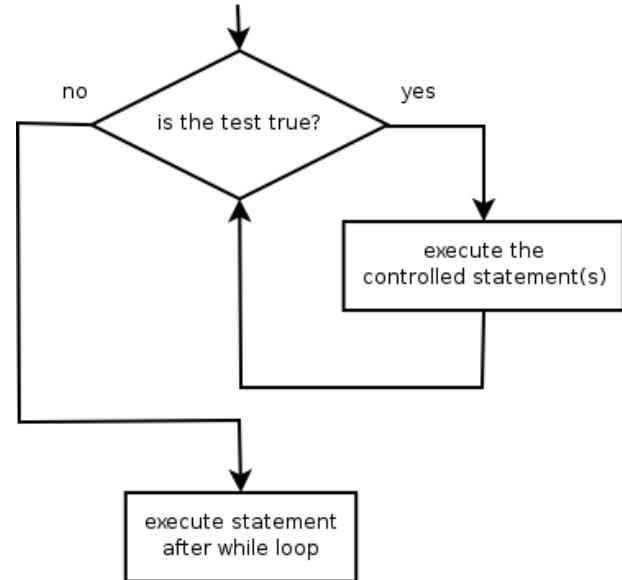
```
for x in range (1,1001):
```

WHILE

```
while <condition>:  
    <statements>
```

◆ Ejemplo

```
number = 1  
  
while number < 200:  
    print (number)  
    number = number * 2
```



1 2 4 8 16 32 64 128

Ejercicio

- ◆ Escribe un programa ejer25WHILE.py que reciba un número del teclado y escribe las potencias de 2 menores que el número dado, usando WHILE

```
number = 1
print("Escribe un numero entero: ")
n = int(input())
print("Las potencias de dos menores que " + str(n) + " son:")
while number < n:
    print (number)
    number = number * 2
```

BREAK y CONTINUE

- ◆ Ejemplo: ejer23BREAK.py

```
while True:  
    s = input ()  
    if s == "quit" :  
        break  
    if len(s) > 3:  
        continue  
    print ("Tiene que ser mas largo")  
el programa continua
```

Funciones

- ◆ Una función se declara usando la palabra clave def

```
>>>def suma(a,b):  
    sum = a + b  
    return sum  
>>>print (suma(5,6))  
11
```

- ◆ A una función se le pueden asignar parámetros por defecto:

```
def suma(a=4,b=6):  
    sum = a + b  
    return sum  
  
>>>print (suma())  
10  
>>>print (suma(b=8))  
12
```

Ejercicio

- ◆ Escribe un programa ejerSuma.py que defina una función que suma dos argumentos

```
def suma(a,b):  
    sum = a + b  
    return sum  
  
print(suma(5,6))
```

Programación modular

Probar las funciones anteriores desde otro script:

```
from script import función
```

```
from script import *
```

Ejercicio 2.7

- ◆ Escribe un programa que defina una función que recibe una función que escribe las potencias de dos menores que el argumento.
- ◆ El programa debe pedir un número por teclado y llamar a dicha función hasta que se escriba un cero

Ejercicio

- ◆ Escribe un programa ejerMCD.py que defina una función que recibe dos números y escribe su mcd.

```
def menor(x, y):  
    if x < y:  
        return x  
    return y  
def MCD(a, b):  
    divisor = menor(a,b)  
    while( divisor > 0      ):  
        #print(a%divisor, b%divisor, divisor)  
        if (a%divisor==0  and b%divisor==0):  
            return divisor  
        divisor-=1  #divisor = divisor -1
```

Recursividad

Ejemplo2:

Factorial recursivo

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n*factorial(n-1)
```

Factorial iterativo

```
def factorial2(n):  
    fact=1  
    for i in range(n,1,-1):  
        fact=fact *i  
    return fact
```

Menú de Opciones

Escribe un programa menu.py que muestre una menú de opciones para llamar repetidamente a las funciones anteriores, incluyendo una opción de salir. Según la función elegida pedirá uno o dos números por teclado. Por ejemplo:

Menú:

1-Potencias de dos menores que un número

2-MCD

3-Fibonacci

0-Salir

Opcion:

Nota: se pueden importar funciones de otros scripts, por ejemplo:

```
from potencias import potenciasDeDos
```

Strings

Funciones de Strings

Operaciones con strings: concatenación

```
>>> x = "Hola Mundo" #x es alfanumérico  
>>> x  
'Hola Mundo'
```

Las cadenas de caracteres se pueden concatenar con ‘+’

```
>>> string1 = "Me gusta "  
>>> string2 = "el Python"  
>>> print (string1 + string2)  
Me gusta el Python
```

Operaciones con strings: multiplicación

Las cadenas de caracteres también se pueden “multiplicar” con el operador ‘*’, concatenándose n veces.

```
>>> x = "Hola " *3
```

```
>>> x
```

```
'Hola Hola Hola '
```

```
>>> x = "Hola " + "Hola " + "Hola "
```

```
>>> x
```

```
'Hola Hola Hola '
```

Strings como lista de chars

El operador corchete selecciona un carácter suelto de una cadena.

```
>>> fruta = "banana"  
>>> letra = fruta[1]  
>>> print letra  
  
a
```

La primera letra se empieza a contar desde la casilla cero

```
>>> letra = fruta[0]  
>>> print letra  
  
b
```

Índices de un String

- Ejemplo

name = "L. Pérez"

index	0	1	2	3	4	5	6	7
character	L	.		P	é	r	e	z

- Ejemplo:

```
print (name, " empieza por ", name[0])
```

Salida:

L. Pérez empieza por L

Longitud de un string

La función **len** devuelve el número de caracteres de una cadena:

```
>>> fruta = "banana"  
>>> len(fruta)
```

```
6
```

Para obtener la última letra de un string:

```
longitud = len(fruta)  
ultima = fruta[longitud-1]
```

Recorriendo strings con for

Ejemplo:

```
for car in fruta:  
    print car
```

Recorriendo strings con for

Ejemplo:

```
prefijos = "JKLMNOPQ"  
sufijo = "ack"  
for letra in prefijos:  
    print (letra + sufijo)
```

La salida del programa es:

Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack

Porciones de cadenas

Llamamos porción a un segmento de una cadena. La selección de una porción es similar a la selección de un carácter:

```
>>> s = "Pedro, Pablo, y María"
```

```
>>> print s[0:5]
```

Pedro

```
>>> print s[7:12]
```

Pablo

```
>>> print s[16:21]
```

María

El operador [n:m] devuelve la parte de la cadena desde el enésimo carácter hasta el emésimo, incluyendo el primero pero excluyendo el último.

Ejemplo

```
>>> smiles = "C(=N) (N) N.C(=O) (O) O"  
>>> smiles[0]  
'C'  
>>> smiles[1]  
' ('  
>>> smiles[-1]  
'O'  
>>> smiles[1:5]  
'(=N)'  
>>> smiles[10:-4]  
'C(=O) '
```

Comparando cadenas

Ejemplo:

```
if palabra < "banana":  
    print "Tu palabra," + palabra +  
        ", va antes de banana."  
  
elif palabra > "banana":  
    print "Tu palabra," + palabra +  
        ", va después de banana."
```

Las cadenas son inmutables

Las cadenas son inmutables, lo que significa que no puede cambiar una cadena existente. Lo más que puede hacer es crear una nueva cadena que sea una variación de la original:

Ejemplo:

```
>>>saludo = "Hola, mundo"  
>>>saludo[0] = 'M'           # ERROR!
```

```
>>>saludo = 'M' + saludo[1:]  
>>>saludo  
'Mola, mundo'
```

Módulo string

El módulo `string` contiene funciones útiles para manipular cadenas.

Ejemplo de uso de la función `find`

```
>>> import string  
>>> fruta = "banana"  
>>> indice = string.find(fruta, "a") #Python2  
>>> indice = fruta.find("a")          #Python 3  
>>> print (indice)  
1
```

Ejemplo: find y split

```
smiles = "C (=N) (N) N.C (=O) (O) O"  
>>> smiles.find("(O)")  
15  
>>> smiles.find(".")  
9  
>>> smiles.find(".", 10)  
-1  
>>> smiles.split(".")  
['C (=N) (N) N', 'C (=O) (O) O']  
>>>
```

Ejemplo: in, not in

```
if "Br" in "Brother":  
    print "contains brother"
```

```
email_address = "luis"  
if "@" not in email_address:  
    email_address += "@ucm.es"
```

Ejemplo: strip

Borra espacios o determinados caracteres

```
>>> line = " # This is a comment line \n"  
>>> line.strip("\n")  
'# This is a comment line'
```

Otros métodos de String

```
email.startswith("c")      endswith("u")
True/False
```

```
>>> names = ["Ben", "Chen", "Yaqin"]
>>> ", ".join(names)
'Ben, Chen, Yaqin'
```

```
>>> "chen".upper()
'CHEN'
```

Otros métodos de String

Ejercicio:

Ver métodos de String desde IDE, a partir de un objeto.

```
1 nombre= "Luis"
2 nombre.|
3     m capitalize(self)                      str
4     m casefold(self)                       str
5     m center(self, width, fillchar)         str
6     m count(self, x, __start, __end)        str
7     m encode(self, encoding, errors)        str
8     m endswith(self, suffix, start, end)    str
9     m expandtabs(self, tabsize)             str
10    m find(self, sub, __start, __end)       str
11    m format(self, args, kwargs)            str
12    m format_map(self, map)                str
13    m index(self, sub, __start, __end)      str
14    m isalnum(self)                        str
15    m isalpha(self)                         str
16    m isdecimal(self)                      str
17
18
```

\ para caracteres especiales

```
\n -> newline  
\t -> tab  
\\" -> backslash
```

...

Pero windows usa backslash para directorios...

```
filename = "C:\python\ejer1.py" # Peligro!  
filename = "C:\\python\\\\ejer1.py" " # Mejor!  
filename = "C:/python/ejer1.py" # Funciona
```

Encontrando minúsculas

Podemos usar estas constantes y find para determinar si un carácter es minúscula

Ejemplos:

```
def esMinuscula(c) :  
    return find(string.lowercase, c) != -1
```

Alternativamente, podemos aprovecharnos del operador **in**, que determina si un carácter aparece en una cadena:

```
def esMinuscula(c) :  
    return c in string.lowercase
```

O bien

```
def esMinuscula(c) :  
    return 'a' <= c <= 'z'
```

Ejercicio

Escribir un programa que pida una frase y aparezca un menú de opciones similar a :

1. Mostrar la longitud de la frase
2. Mostrar el número de minúsculas
3. Mostrar el número de mayúsculas
4. Mostrar el número de dígitos (0..9)
5. Mostrar el número de palabras
6. Pedir una letra y mostrar el número de ocurrencias de dicha letra
7. Mostar la frase al revés
8. Indicar si la frase es palíndromo

Secuencias, Listas, Diccionarios

- Operaciones básicas con Listas, Tuplas, Diccionarios y Conjuntos
- Búsqueda, ordenación
- Iteradores
- Generadores
- Lambdas

Secuencias

Las listas y las cadenas, y otras cosas que se comportan como conjuntos ordenados, se llaman **secuencias**.

Ejemplo:

[10, 20, 30, 40]

["spam", "elástico", "golondrina"]

Listas

Una **lista** es un **conjunto ordenado** de valores, en el cual cada valor va identificado por un **índice**.

Los valores que constituyen una lista son sus **elementos**.

Las listas son similares a las cadenas de texto (strings), que son conjuntos ordenados de caracteres, excepto en que los elementos de una lista pueden ser de cualquier tipo.

Listas

Los elementos de una lista no tienen por qué ser del mismo tipo.

Ejemplo:

```
["hola", 2.0, 5, [10, 20]]
```

Se dice que una lista dentro de otra lista está **anidada**.

Listas de enteros: range

Python proporciona una manera sencilla de crear las listas que contienen números enteros consecutivos:

```
>>> range(1,5)  
[1, 2, 3, 4]
```

Hay dos formas alternativas para range:

- Con un solo argumento, crea una lista que empieza desde 0:

```
>>> range(10)  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Con un tercer argumento, especificando el el espacio (step) entre dos valores sucesivos;

```
>>> range(1, 10, 2)  
[1, 3, 5, 7, 9]
```

Listas vacías

Para terminar, hay una lista especial que no contiene elementos.

Se la llama lista vacía y se representa [].

```
vacio = []
```

Listas: acceso a los elementos

La sintaxis para acceder a los elementos de una lista es la misma que para acceder a los caracteres de una cadena: el operador corchetes [].

La expresión dentro de los corchetes especifica el índice, comenzando en cero.

```
>>> numeros = range(10)  
>>> numeros  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> print numeros[1]  
1
```

Listas: acceso a los elementos

También se pueden asignar elementos a una lista:

```
>>> numeros[1]=5  
>>> numeros  
[0, 5, 2, 3, 4, 5, 6, 7, 8, 9]
```

Listas: acceso a los elementos

Si se da un índice negativo, se cuenta hacia atrás desde el final de la lista.

```
>>> numeros  
[0, 5, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> numeros [-1]  
9  
>>> numeros [-2]  
8
```

Listas: len

La función ‘len’ toma una lista y devuelve su tamaño.

```
>>> numeros  
[0, 5, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> len(numeros)  
10
```

Es una buena idea usar este valor como límite superior de un bucle, en lugar de una constante.

Listas: recorrido con FOR

En Python **for** se utiliza como una forma genérica de iterar sobre una secuencia

La sintaxis generalizada de un bucle for es:

```
for VARIABLE in LISTA:  
    CUERPO
```

Ejemplo

```
secuencia = ["uno", "dos", "tres"]  
for elemento in secuencia:  
    print "número "+elemento
```

Listas: operaciones

El operador `+` concatena listas:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

De forma similar, el operador `*` repite una lista un número dado de veces:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Listas: slices

Las operaciones de porciones funcionan sobre las listas:

```
>>> lista = ['a', 'b', 'c', 'd', 'e', 'f']
>>> lista[1:3]
['b', 'c']
>>> lista[:4]
['a', 'b', 'c', 'd']
>>> lista[3:]
['d', 'e', 'f']
>>> lista[:]
['a', 'b', 'c', 'd', 'e', 'f']
```

Listas: borrado con DEL

Las listas son Mutables (los Strings no)

“**del**” elimina un elemento de una list

```
>>> a = ['uno', 'dos', 'tres']
```

```
>>> del a[1]
```

```
>>> a
```

```
['uno', 'tres']
```

```
>>> lista = ['a', 'b', 'c', 'd', 'e', 'f']
```

```
>>> del lista[1:5]
```

```
>>> print lista
```

```
['a', 'f']
```

Métodos de los objetos lista

list.append(x). Agrega un ítem al final de la lista; equivale a `a[len(a):] = [x]`.

list.extend(L). Extiende la lista agregándole todos los ítems de la lista dada; equivale a `a[len(a):] = L`.

list.insert(i, x). Inserta un ítem en una posición dada.

list.remove(x). Quita el primer ítem de la lista cuyo valor sea x.

list.pop([i]). Quita el ítem en la posición dada de la lista, y lo devuelve.

Métodos de los objetos lista

list.index(x). Devuelve el índice en la lista del primer ítem cuyo valor sea x. Es un error si no existe tal ítem.

list.count(x). Devuelve el número de veces que x aparece en la lista.

list.sort(). Ordena los ítems de la lista, in situ.

list.reverse(). Invierte los elementos de la lista, in situ.

Métodos de los objetos lista

Ver los métodos de List en un IDE

The screenshot shows a code editor with the following code:

```
lista = []
lista.
```

A method completion dropdown is open, listing the following methods for the `list` class:

Método	Tipo de retorno
<code>append(self, object)</code>	<code>list</code>
<code>clear(self)</code>	<code>list</code>
<code>copy(self)</code>	<code>list</code>
<code>count(self, object)</code>	<code>list</code>
<code>extend(self, iterable)</code>	<code>list</code>
<code>index(self, object, start, stop)</code>	<code>list</code>
<code>insert(self, index, object)</code>	<code>list</code>
<code>pop(self, index)</code>	<code>list</code>
<code>remove(self, object)</code>	<code>list</code>
<code>reverse(self)</code>	<code>list</code>
<code>sort(self, key, reverse)</code>	<code>list</code>
<code>__add__(self, x)</code>	<code>list</code>

Listas como argumento

Cuando se pasa una lista como argumento, en realidad se pasa una referencia a ella, no una copia de la lista.

```
def cabeza(lista):  
    return lista[0]  
  
numeros = [1,2,3]  
cabeza(numeros)  
1
```

Listas como argumento

Cuando se pasa una lista como argumento, en realidad se pasa una referencia a ella, no una copia de la lista.

```
def cabeza(lista):  
    return lista[0]
```

```
>>> numeros = [1,2,3]  
>>> cabeza(numeros)  
>>> 1
```

El parámetro `lista` y la variable “`numeros`” son alias de un mismo objeto.

Listas como argumento

Si la función modifica una lista pasada como parámetro, el que hizo la llamada verá el cambio.

```
def borra_cabeza(lista):  
    del lista[0]  
  
>>> numeros = [1,2,3]  
>>> borra_cabeza(numeros)  
>>> print numeros  
[2, 3]
```

Listas como argumento

Si una función devuelve una lista, devuelve una referencia a la lista.

```
def cola(lista):  
    return lista[1:]
```

```
>>> numeros = [1,2,3]  
>>> resto = cola(numeros)  
>>> print resto  
>>> [2, 3]
```

Ejercicio

Escribe un script Ejer41.py con las siguientes funciones que reciban una lista de enteros:

imprime(lista) -imprime los valores de una lista

máximo(lista) -devuelve el valor máximo

mínimo(lista) -devuelve el valor mínimo

suma(lista) -devuelve la suma de los valores

longitud(lista) -devuelve la longitud de la lista

media(lista) -devuelve la media de los valore

Incluye un programa que pida números y los guarde en una lista hasta introducir el -1 y muestre un menú para preguntar por cada función de la lista introducida, incluyendo una opción 0 para salir.

Listas: Objetos

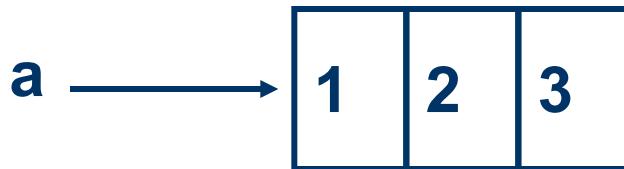
Cada objeto tiene un identificador único, que podemos obtener por medio de la función id. Imprimiendo los identificadores de a y b podemos saber si apuntan al mismo objeto.

Como las cadenas de texto son inmutables:

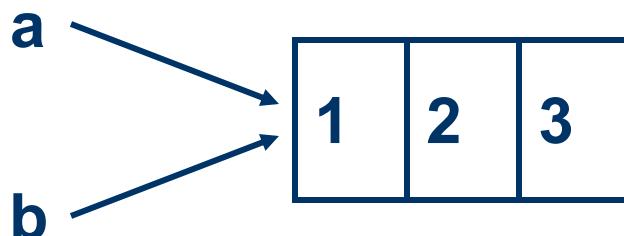
```
>>> a="Hola"  
>>> b="Hola"  
>>> print id(a), id(b)  
107295808 107295808  
>>> b="que tal"  
>>> print id(a), id(b)  
107295808 107295968
```

Las listas son objetos

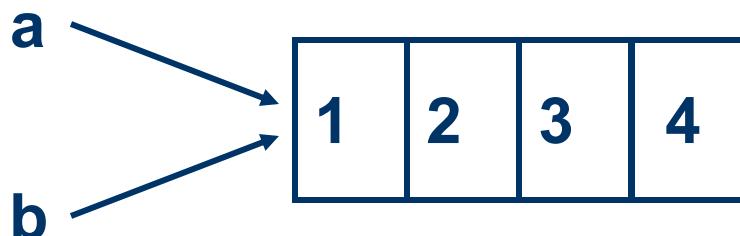
a = [1, 2, 3]



b = a



a.append(4)



Listas: Objetos

Curiosamente, las listas se comportan de otra manera. Cuando crea dos listas,
obtiene dos objetos:

```
>>> l1=[1,2,3]
>>> l2=[1,2,3]
>>> print id(l1),id(l2)
321677672 321677632
>>> l1==l2
True
```

Listas: alias de Objetos

Si asigna una variable a otra, ambas variables se refieren al mismo objeto:

```
>>> l1=[1,2,3]
>>> l3=l1
>>> l3
[1, 2, 3]
>>> print id(l1),id(l3)
321677672 321677672
>>> l3[2]=5
>>> l1
[1, 2, 5]
```

Listas: clones

Si queremos modificar una lista y mantener una copia del original, necesitaremos ser capaces de hacer una copia de la lista, no sólo de su referencia. Este proceso a veces se denomina clonado.

```
>>> a = [1, 2, 3]
>>> b = []
>>> b[:] = a[:]
>>> print b
[1, 2, 3]
```

Listas anidadas

```
>>> lista = ["hola", 2.0, 5, [10, 20]]
```

Para extraer los elementos de la lista anidada, podemos proceder en dos pasos:

```
>>> elt = lista[3]
```

```
>>> elt[0]
```

```
10
```

O podemos combinarlos:

```
>>> lista[3][1]
```

```
20
```

matrices

```
>>> matriz = [[1, 2, 3], [4, 5, 6],  
             [7, 8, 9]]
```

Podemos elegir una fila entera de la matriz :

```
>>> matriz[1]  
[4, 5, 6]
```

O tomar sólo un elemento de la matriz usando doble índice:

```
>>> matriz[1][1]  
5
```

Cadenas y listas: split

La función “**split**” divide una cadena en una lista de palabras.
Por defecto, cualquier número de caracteres de espacio en blanco
se considera un límite de palabra:

```
>>> import string  
>>> cancion = "La lluvia en Sevilla..."  
>>> string.split(cancion)  
['La', 'lluvia', 'en', 'Sevilla...']
```

Cadenas y listas: split

Se puede usar un argumento opcional llamado **delimitador** para especificar qué caracteres se usarán como límites de palabra.

```
>>> import string  
>>> cancion = "La lluvia en Sevilla..."  
>>> string.split(cancion, 'll')  
['La ', 'uvia en Sevi', 'a...']
```

Cadenas y listas: join

La función join es la inversa de split.

Toma una lista de cadenas y concatena los elementos con un espacio entre cada par:

```
>>> import string  
>>> lista = ['La', 'lluvia', 'en',  
           'Sevilla...']  
>>> string.join(lista)  
'La lluvia en Sevilla...'
```

Ejercicio

Escribe un script Ejer.py con las siguientes funciones:

imprime(string)	- Imprime el string
numPalabras(string)	- Devuelve el número de palabras
imprimePalabras(string)	- Imprime las palabras en distintas líneas
imprimePalabrasOrd(string)	- Imprime las palabras ordenadas
imprimePalabras(string, n)	- Imprime las palabras de longitud n
maxLong(string)	- Da la longitud de la palabra más larga
minLong(string)	- Da la longitud de la palabra más corta
imprimePalabrasOrdLong(string)	- Imprime las palabras ordenadas por su longitud y orden lexicográfico

Incluye un programa que pida una frase y muestre un menú para preguntar por cada función de la lista introducida, incluyendo una opción 0 para salir.

RANGE

Ejemplo:

Suma los números positivos menores que 10

```
>>> n=0  
>>> for i in range(10):  
...     n=n+i  
...  
>>> n  
45
```

```
>>> sum(i for i in range(10))  
45
```

RANGE

Ejemplo:

Suma los cuadrados de los números positivos menores que 10

```
>>> sum(i*i for i in range(10))
```

```
285
```

O bien:

```
>>> sum(i**2 for i in range(10))
```

```
285
```

Herramientas de programación funcional

Hay tres funciones integradas que son muy útiles cuando se usan con listas: filter(), map(), y reduce().

filter(funcion, secuencia) devuelve una secuencia con aquellos ítems de la secuencia para los cuales **funcion(item)** es verdadero.

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0  
...  
>>> filter(f, range(2, 25))  
[5, 7, 11, 13, 17, 19, 23]
```

Herramientas de programación funcional

map(funcion, secuencia) llama a `funcion(item)` por cada uno de los ítems de la secuencia y devuelve una lista de los valores retornados. Por ejemplo, para calcular unos cubos:

```
>>> def cubo(x): return x*x*x  
>>> map(cubo, range(1, 11))  
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

reduce(funcion, secuencia) devuelve un único valor que se construye llamando a la función binaria `funcion` con los primeros dos ítems de la secuencia, entonces con el resultado y el siguiente ítem, y así sucesivamente.

```
>>> def sumar(x, y): return x+y  
>>> reduce(sumar, range(1, 11))
```

Tuplas

En Python una tupla que es similar a una lista salvo en que es inmutable.

Sintácticamente, una tupla es una lista de valores separados por comas:

```
>>> tupla = 'a', 'b', 'c', 'd', 'e'
```

Aunque no es necesario, se debe encerrar las tuplas entre paréntesis:

```
>>> tupla = ('a', 'b', 'c', 'd', 'e')
```

Tuplas

Para crear una tupla con un sólo elemento, debemos incluir una coma al final:

```
>>> t1 = ('a',)  
>>> type(t1)  
<type 'tuple'>
```

Sin la coma sería un string

```
>>> t2 = ('a')  
>>> type(t2)  
<type 'string'>
```

Tuplas

Las operaciones sobre las tuplas son las mismas que sobre las listas.

```
>>> tupla = ('a', 'b', 'c', 'd', 'e')  
>>> tupla[0]  
'a'  
>>> tupla[1:3]  
('b', 'c')
```

Tuplas

Pero si intentamos modificar uno de los elementos de la tupla provocaremos un error:

```
>>> tupla[0] = 'A'
```

```
TypeError: object doesn't support item assignment
```

Aunque no podamos modificar los elementos de una tupla, podemos sustituir una tupla por otra diferente:

```
>>> tupla = ('A',) + tupla[1:]
```

```
>>> tupla
```

```
('A', 'b', 'c', 'd', 'e')
```

Asignación de tuplas

Para intercambiar los valores de dos variables se puede usar una variable temporal:

```
>>> temp = a  
>>> a = b  
>>> b = temp
```

O con asignación de tuplas:

```
>>> a, b = b, a
```

Tuplas como valor de retorno

Las funciones pueden devolver tuplas como valor de retorno. Por ejemplo, una función que intercambie dos parámetros:

```
def intercambio(x, y):  
    return y, x  
a, b = intercambio(a, b)
```

Peligroso error:

```
def intercambio(x, y): # versión incorrecta  
    x, y = y, x
```

No funciona porque los argumentos se pasan por valor, no por referencia, y lo que se cambian son variables locales.

Diccionarios

Los diccionarios son similares a otros tipos compuestos (cadenas, listas y tuplas) excepto en que pueden usar como índice cualquier tipo inmutable.

Ejemplo: diccionario que traduzca inglés al español.

```
>>> ingEsp = {}  
>>> ingEsp['one'] = 'uno'  
>>> ingEsp['two'] = 'dos'  
>>> print ingEsp  
{'one': 'uno', 'two': 'dos'}
```

Diccionarios

```
>>> ingEsp = { 'one': 'uno',  
              'two': 'dos',  
              'three': 'tres'}  
  
>>> print ingEsp  
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Los pares clave-valor no están en orden, ya que los elementos de un diccionario nunca se indexan con índices enteros.

En lugar de eso, usamos las claves para buscar los valores correspondientes:

```
>>> print ingEsp['two']  
'dos'
```

Métodos KEYS y VALUES

El método “**keys**” acepta un diccionario y devuelve una lista con las claves que aparecen

```
>>> ingEsp.keys()          #similar a keys(ingEsp)  
['one', 'three', 'two']
```

El método “**values**” devuelve una lista de los valores del diccionario:

```
>>> ingEsp.values()  
['uno', 'tres', 'dos']
```

Métodos ITEMA y HAS_KEY

El método “**items**” devuelve una lista de tuplas con los pares clave-valor del diccionario:

```
>>> ingEsp.items()  
[ ('one', 'uno'), ('three', 'tres'), ('two',  
'dos') ]
```

El método “**has_key**” acepta una clave y devuelve verdadero (1) si la clave aparece en el diccionario:

```
>>> ingEsp.has_key('one')
```

1

Conjuntos

Python también incluye un tipo de dato para **conjuntos**. Un conjunto es una colección **no ordenada y sin elementos repetidos**.

```
>>> canasta = ['manzana', 'naranja',
   'manzana', 'pera', 'naranja', 'banana']
>>> fruta = set(canasta) # crea un conjunto
>>> fruta
set(['pera', 'manzana', 'banana', 'naranja'])
```

Los usos básicos de éstos incluyen verificación de pertenencia y eliminación de entradas duplicadas.

```
>>> 'naranja' in fruta          #
verificación de pertenencia rápida
True
>>> 'yerba' in fruta
False
```

Conjuntos

Los conjuntos también soportan operaciones matemáticas como la unión, intersección, diferencia, y diferencia simétrica.

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                               # letras únicas en a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                           # letras en a pero no en b
set(['r', 'b', 'd'])
>>> a | b                           # letras en a o en b
set(['a', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])
>>> a & b                           # letras en a y en b
set(['a', 'c'])
>>> a ^ b                           # letras en a o b pero no en ambos
set(['b', 'd', 'm', 'l', 'r', 'z'])
```

Ejercicio (Opcional)

Escribe un programa para la gestión de las notas de una asignatura. El programa debe pedir el nombre y apellido del alumno y su nota, que debe ser entre cero y 10 para que se almacene. Cuando se introduce un alumno de longitud cero se termina la introducción de datos. Una vez introducidas las notas el programa debe mostrar un menú con las siguientes opciones.

- 1- Mostrar notas
- 2- Mostrar número de alumnos
- 3- Mostrar nota media
- 4- Mostrar el nombre del alumno con mejor nota
- 5- Mostrar el nombre y nota de los suspensos
- 6- Mostar el nombre y nota de los alumnos aprobados ordenados por nota de mayor a menor
- 0- Salir

Ficheros en Python

- Lectura y escritura en ficheros
- Formatos

Ficheros

Trabajar con archivos se parece mucho a trabajar con libros.

Para usar un libro, hay que abrirlo. Cuando se ha terminado, hay que cerrarlo. Mientras el libro está abierto, puede escribir en él (“w”) o leer de él (“r”).

```
>>> f = open("test.dat", "w")
>>> f.write("Hooooola")
>>> f.close()
>>> file=open("test.dat", "r")
>>> m=file.read()
>>> print m
Hooooola
>>> file.close()
```

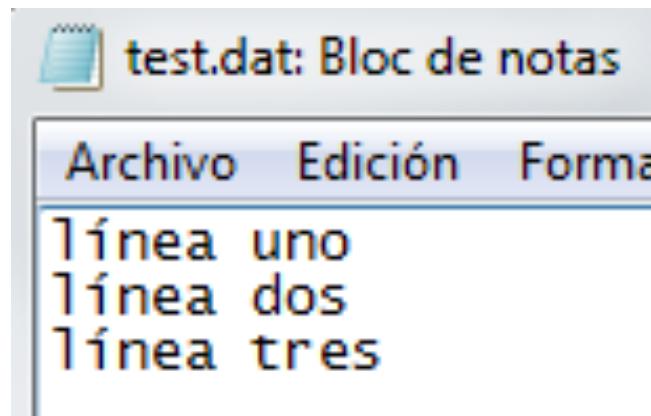
Ficheros

La función “**read**” admite el número de caracteres siguientes a leer.

```
>>> file=open("test.dat","r")
>>> print file.read(3)
Hoo
>>> print file.read(2)
oo
>>> file.read(3)
'la'
>>> file.close()
```

Ejercicio 5.1

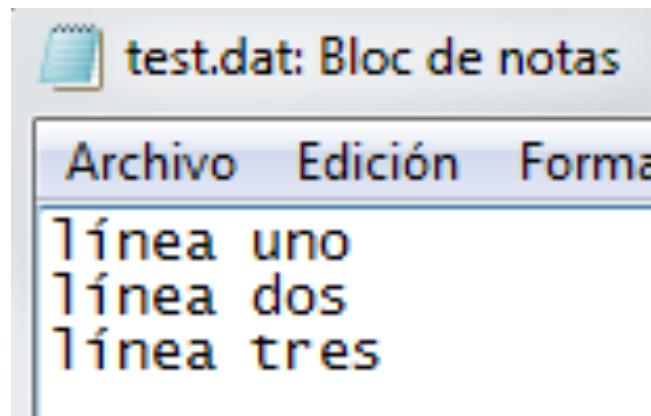
Escribir un programa que escriba tres líneas de texto en un fichero llamado test.dat



Ejercicio 5.1

Escribir un programa que escriba tres líneas de texto en un fichero llamado test.dat

```
f = open("test.dat", "w")
f.write("línea uno\nlínea dos\nlínea
tres\n")
f.close()
```



Ficheros

El método “**readline**“ lee todos los caracteres hasta e inclusive el siguiente salto de línea:

```
>>> f = open("test.dat", "r")
>>> print f.readline()
línea uno
```

“**readlines**” devuelve todas las líneas que queden como una lista de cadenas:

```
>>> print f.readlines()
['línea dos\n', 'línea tres\n']
>>> print f.readlines()
[]
>>> f.close()
```

Ejercicio

Define una función que copia un archivo, leyendo y escribiendo los caracteres de cincuenta en cincuenta.

```
def copiaArchivo(archViejo, archNuevo):  
    f1 = open(archViejo, "r")  
    f2 = open(archNuevo, "w")  
    while 1:  
        texto = f1.read(50)  
        if texto == "":  
            break  
        f2.write(texto)  
    f1.close()  
    f2.close()
```

Nota: La sentencia “**break**” interrumpe el bucle while

Ejercicio

Define una función que copia un archivo omitiendo las líneas que comienzan por #

```
def filtraArchivo(archViejo, archNuevo):  
    f1 = open(archViejo, "r")  
    f2 = open(archNuevo, "w")  
    while 1:  
        texto = f1.readline()  
        if texto == "":  
            break  
        if texto[0] == '#':  
            continue  
        f2.write(texto)  
    f1.close()  
    f2.close()
```

Nota: La sentencia “**continue**” termina la iteración actual del bucle, pero sigue haciendo bucles.

Operador de formato %.

Cuando aplica a enteros, % es el operador de módulo.

Pero cuando el primer operando es una cadena, % es el operador de formato.

El primer operando es la cadena de formato, y el segundo operando es una tupla de expresiones.

La secuencia de formato %d (d de decimal) significa que la primera expresión de la tupla debe formatearse como un entero.

```
>>> motos = 52  
>>> "%d" % motos  
'52'
```

El resultado es la cadena '52', que no debe confundirse con el valor entero 52.

Operador de formato %.

%f formatea el siguiente elemento de la tupla como un número en coma flotante (f de float)

%s formatea el siguiente elemento como una cadena (s de string)

```
>>> "En %d dias ingresamos %f millones de %s."\
... % (34, 6.1, 'dolares')
...
'En 34 dias ingresamos 6.100000 millones de dolares.'
```

Por defecto, el formato de coma flotante imprime seis decimales.

Operador de formato %.

Podemos detallar el número de dígitos como parte de la secuencia de formato:

```
>>> "%6d" % 62  
'       62 '  
>>> "%12f" % 6.1  
' 6.100000 '
```

Si el valor necesita menos dígitos, se añaden espacios en blanco delante del número. Si el número de espacios es negativo, se añaden los espacios tras el número:

```
>>> "%-6d" % 62  
' 62      '
```

Operador de formato %.

Podemos especificar el número de decimales para los números en coma flotante:

```
>>> "%12.2f" % 6.1  
' 6.10 '
```

Ejercicio

Escribir una función que reciba un diccionario, por ejemplo de notas, y lo escriba línea a línea ordenado por orden alfabético.

```
def informe (notas) :  
    estudiantes = notas.keys()  
    estudiantes.sort()  
    for estudiante in estudiantes :  
        print "%-20s %12.02f" % (estudiante,  
                                notas[estudiante])
```

```
>>> notasFeb = {'Eva': 6.23, 'Miriam': 9.45, 'Luis':  
7.25}
```

```
>>> informe(notasFeb)
```

Eva	6.23
Luis	7.25
Miriam	9.45

Modularidad

- Módulos standard
- Nuestros propios módulos
- Packages

Scripts en Python

Si se sale del intérprete de Python y se entra de nuevo, las definiciones (**funciones, variables, clases...**) se pierden.

Un guión, o script es un programa escrito con un editor de texto (en Python tiene extensión .py) para preparar la entrada para el interprete y ejecutarlo con ese archivo como entrada.

También se usan scripts para definir funciones o clases que **queramos usar desde otros programas.**

Módulos en Python

Un módulo es un archivo (en Python también tiene **extensión .py**) que contiene definiciones y declaraciones y puede ser usado en un script o en una instancia interactiva del intérprete.

Las definiciones de un módulo **pueden ser importadas** a otros módulos o al módulo principal.

Tenemos una colección de **módulos estándar** que podemos importar para usar, y también podemos crear nuestros propios módulos.

Módulos: import

- ◆ import hace que un módulo y su contenido sean disponibles para su uso.
- ◆ Algunas formas de uso son:

import test

- Importa modulo test. Referir a x en test con "test.x".

from test import x

- Importa x de test. Referir a x en test con "x".

from test import *

- Importa todos los objetos de test. Referir a x en test con "x".

import test as theTest

- Importa test; lo hace disponible como theTest. Referir a objeto x como "theTest.x".

Importación de un módulo

- ◆ Un módulo es una colección de métodos en un archivo que acaba en .py. El nombre del archivo determina el nombre del módulo en la mayoría de los casos.
- ◆ E.j. modulo.py:

```
def one(a):  
    print "in one"  
def two (c):  
    print "in two"
```

- ◆ Uso de un módulo:

```
>>> import modulo  
>>> dir(modulo) # lista contenidos módulo  
['__builtins__', '__doc__', '__file__', '__name__',  
'one', 'two']  
>>> modulo.one(2)  
in one
```

Función dir

La función **dir** nos devuelve una lista las funciones de un módulo.

```
>>> import math  
>>> dir(math)  
['__doc__', '__name__', '__package__', 'acos',  
'acosh', 'asin', 'asinh', 'atan', 'atan2',  
'atanh', 'ceil', 'copysign', 'cos', 'cosh',  
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1',  
'fabs', 'factorial', 'floor', 'fmod', 'frexp',  
'fsum', 'gamma', 'hypot', 'isinf', 'isnan',  
'ldexp', 'lgamma', 'log', 'log10', 'log1p',  
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',  
'sqrt', 'tan', 'tanh', 'trunc']
```

Función help

La función **help** nos describe las funciones de un módulo.

```
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available. It provides access to
    the
        mathematical functions defined by the C standard.

FUNCTIONS
    acos(....)
       acos(x)

        Return the arc cosine (measured in radians) of x.

    ...
DATA
    e = 2.718281828459045
    pi = 3.141592653589793
```

Biblioteca estándar

Python incluye automáticamente una serie de módulos de funciones y clases, como:

- ◆ sys
- ◆ math
- ◆ os
- ◆ ...

Librería de funciones math

Librería math

A screenshot of a Python IDE showing the documentation for the `math` module. The window title is "Python". In the code editor, the user has typed `>>> math.`. A dropdown menu lists several mathematical functions: `acos`, `acosh`, `asin`, `asinh`, `atan`, `atan2`, `atanh`, `ceil`, and `copysign`. To the right of the code editor, the module's documentation is displayed:

```
math.  
This module is always available. It provides  
access to the mathematical functions defined by  
the C standard.
```

import

Las librerías deben ser importadas para ser usadas.

Ejemplo:

```
>>> import sys  
>>> print(sys.version)
```

```
>>> import os  
>>> filename = os.environ.get('PYTHONSTARTUP')
```

From lib import *

Podemos importar de manera que no sea necesario calificar los atributos y funciones

Ejemplo:

```
>>> from sys import *
>>> print (version)
2.7.3 (default, Apr 10 2012, 23:31:26)
```

```
>>> from math import sqrt
>>> sqrt(4)
2.0
```

Librería sys

Librería sys

Python

```
>>> sys.version
'2.7.3 (default, Apr 10
2012, 23:31:26) [MSC v.1500 32 bit (Intel)]'
>>> sys.

    ▾ api_version
    ▾ argv
    ▾ builtin_module_names
    ▾ byteorder
    ▾ call_tracing
    ▾ callstats
    ▾ copyright
    ▾ displayhook
    ▾ dllhandle
```

sys.
This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter.
Dynamic objects:
argv -- command line arguments; argv[0] is the script pathname if known
path -- module search path; path[0] is the script directory, else ''
modules -- dictionary of loaded modules

Módulo random

El módulo **random** provee herramientas para realizar selecciones al azar:

```
>>> import random  
>>>random.choice( ['manzana', 'pera', 'banana'] )  
'manzana'  
>>> random.sample(xrange(100), 10)  
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]  
>>> random.random()      # un float al azar  
0.17970987693706186  
>>> random.randrange(6)  
4
```

Módulo datetime

El módulo datetime ofrece clases para manejar fechas y tiempos tanto de manera simple como compleja.

El módulo también soporta objetos que son conscientes de la zona horaria.

```
>>> from datetime import date  
>>> hoy = date.today()  
>>> print hoy  
2013-01-08  
>>> nacimiento = date(2004, 9, 22)  
>>> edad = hoy - nacimiento  
>>> edad  
datetime.timedelta(3030)  
>>> edad.days  
3030
```

Otros módulos

El módulo **glob** provee una función para hacer listas de archivos a partir de búsquedas con comodines.

El módulo **re** provee herramientas de expresiones regulares para un procesamiento avanzado de cadenas.

Hay varios módulos para acceder a internet y procesar sus protocolos. Dos de los más simples son **urllib2** para traer data de URLs y **smtplib** para mandar correos.

Los formatos para archivar y comprimir datos se soportan directamente con los módulos: **zlib**, **gzip**, **bz2**, **zipfile** y **tarfile**.

El módulo **threading** puede ejecutar tareas en segundo plano

Librerías para ficheros, bases de datos, GUIs, etc.

Módulos propios

Los módulos y scripts .py pueden importarse desde otros módulos o scripts, siempre que la carpeta esté incluida en el directorio actual, la variable de entorno PYTHONPATH o en **sys.path**

```
>>> sys.path
['C:\\Windows\\system32\\python27.zip', u'c:\\program
 files (x86) \\arcgis\\desktop10.1\\ arcpy',
'C:\\Python27\\ArcGIS10.1\\Lib',
'C:\\Python27\\ArcGIS10.1\\DLLs',
'C:\\Python27\\ArcGIS10.1\\Lib\\lib-tk',
'C:\\Python27\\ArcGIS10.1',
'C:\\Python27\\ArcGIS10.1\\lib\\site-packages',
'C:\\Program Files
(x86) \\ArcGIS\\Desktop10.1\\ arcpy',
'C:\\ProgramFiles(x86) \\ArcGIS\\Desktop10.1\\ArcToolb
ox\\Scripts',
'C:\\Program Files (x86) \\ArcGIS\\Desktop10.1\\bin',
]
```

Módulos propios

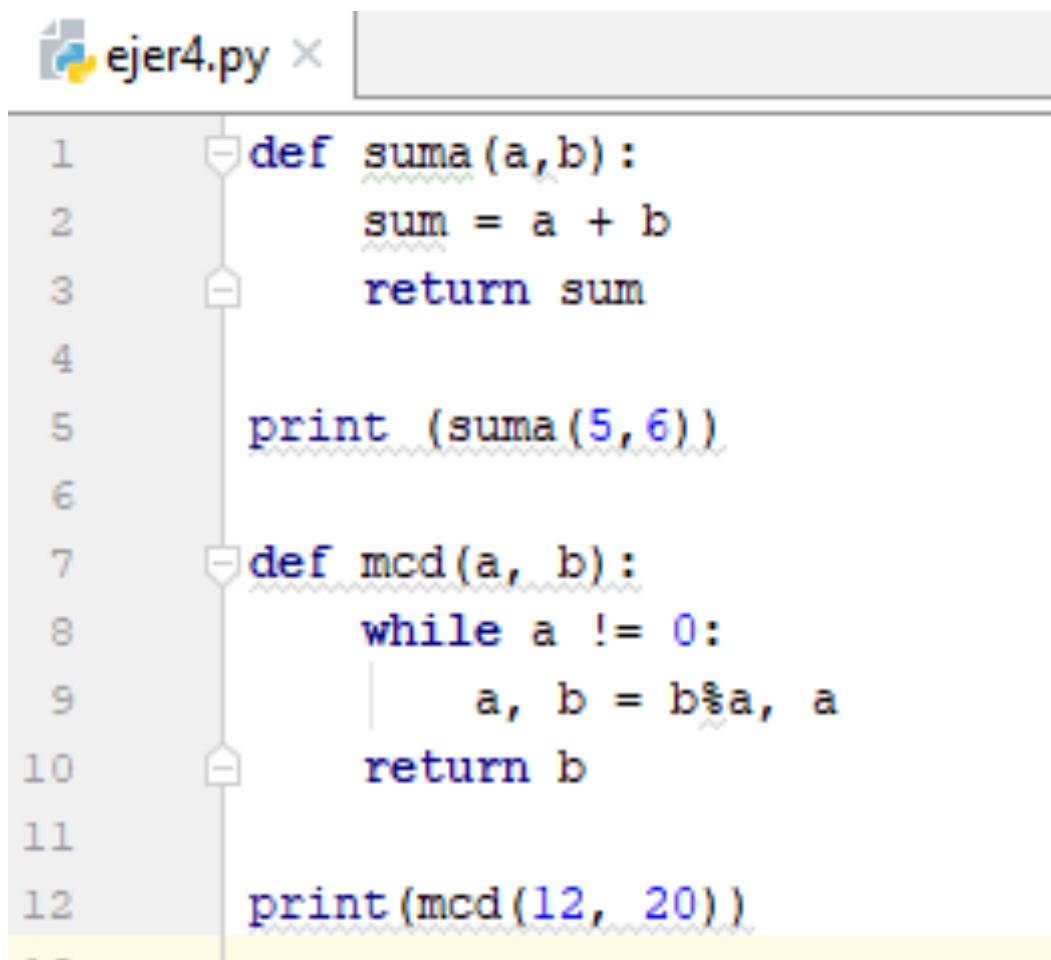
Un módulo puede contener tanto **declaraciones ejecutables como definiciones de funciones**. Estas declaraciones están pensadas para inicializar el módulo. Se ejecutan solamente la primera vez que el módulo se importa en algún lado.

Los módulos **pueden importar otros módulos**.

Es costumbre pero no obligatorio el ubicar todas las declaraciones import al principio del módulo (o script, para el caso).

Ejercicio

- ◆ Crear en tu proyecto, añadiendo las funciones suma y mcd (de otro módulo) y alguna instrucción



```
1 def suma(a,b):
2     sum = a + b
3     return sum
4
5 print(suma(5,6))
6
7 def mcd(a, b):
8     while a != 0:
9         a, b = b%a, a
10    return b
11
12 print(mcd(12, 20))
```

Ejercicio

- ◆ Llama a las funciones con namespace

The screenshot shows a code editor with two tabs:

- ejer4.py**:
The code in this file imports functions from the `ejer4` module and prints their results:

```
1 import ejer4
2 print(ejer4.suma(5, 7))
3
4 print(ejer4.mcd(24, 32))
```
- ejer42.py**:
This file contains the command to run the script and its output:

```
C:\Python\Proyecto\venv\Scripts\python.exe
12
8
```

¿Hay que calificar las funciones? –espacio para nombres-

Ejercicio 6.2

- ◆ Llama a las funciones –sin usar el namespace-

The screenshot shows a code editor with two tabs at the top: "ejer4.py" and "ejer42.py". The "ejer42.py" tab is active, displaying the following Python code:

```
1 from ejer4 import *
2 print (suma (5, 3))
3
4 print (mcd(24, 32))
```

The code imports all functions from "ejer4.py" and calls them directly without using the module name as a prefix.

¿Dónde se buscan los módulos?

1. Variable **sys.path**, automáticamente inicializada al directorio actual (¡no es lo mismo que el directorio del script!)
2. En las ubicaciones descritas en la variable de entorno **PYTHONPATH**
3. Ubicación dependiente de la instalación del intérprete (en SO Unix esta es, usualmente, `./:/usr/local/lib/python`)

Añadir ruta en sys.path

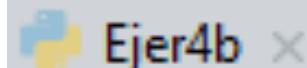
Si la carpeta de scripts no está incluida en el sys.path, debe incluirse.

```
>>>sys.path.append('C:\\Python\\Proyecto')
>>> import ejer4
>>> ejer.suma(7, 5)
12
```

Ejercicio

Crear un proyecto llamado Proyecto2 y utilizar la función ejer4.suma del Proyecto1 (sin usar packages)

```
import sys  
  
sys.path.append('C:\\Python\\Proyecto')  
  
import ejer4  
  
print(ejer4.suma(2,3))
```



```
C:\Python\Proyecto2\venv\Scripts\python.exe  
5
```

Paquetes

Un paquete es una manera de **organizar un conjunto de módulos como una unidad**.

Los paquetes pueden a su vez contener otros paquetes.

Para aprender como crear un paquete consideremos el siguiente contenido de un paquete:

```
package_example/  
package_example/__init__.py  
package_example/module1.py  
package_example/module2.py
```

Y estos serían sus contenidos:

```
# __init__.py  
# Exponer definiciones de módulos en este  
paquete.  
from module1 import class1  
from module2 import class2
```

Ejemplo Paquetes

- # module1.py
class class1:
 def __init__(self):
 self.description = 'class #1'
 def show(self):
 print self.description
- # module2.py
class class2:
 def __init__(self):
 self.description = 'class #2'
 def show(self):
 print self.description

Ejemplo Paquetes

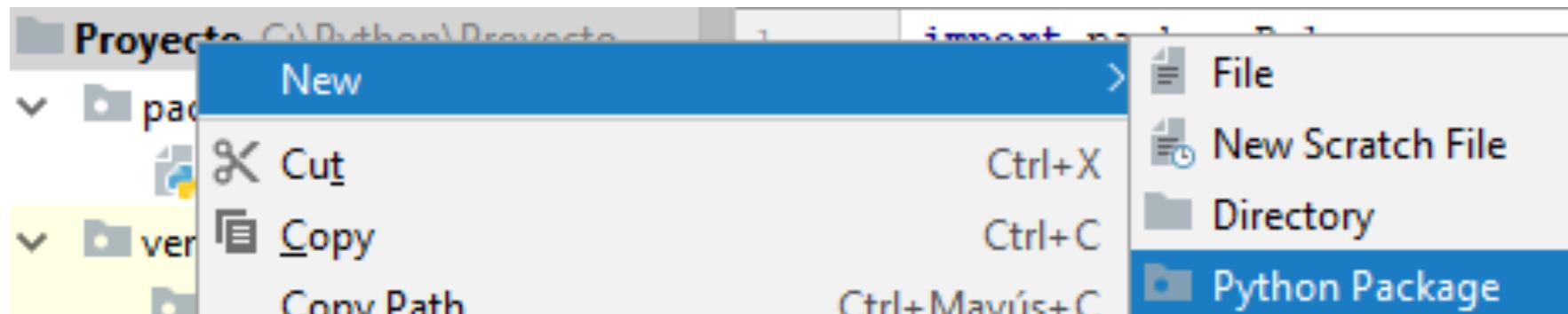
- ◆

```
# testpackage.py
import package_example
c1 = package_example.class1()
c1.show()
c2 = package_example.class2()
c2.show()
```
- ◆ Visualizaría:

```
class #1
class #2
```
- ◆ La localización de los paquetes debe especificarse o bien a través de la variable de entorno PYTHONPATH o en código del script mediante sys.path

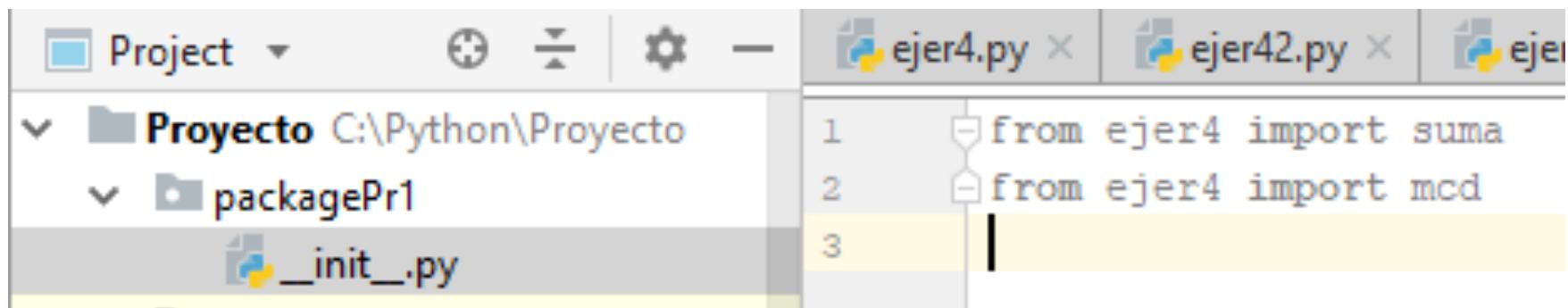
Ejercicio

Crear un package packagePr1 incluyendo las funciones suma y mcd del módulo ejer61



Ejercicio 6.5

Incluir las funciones suma y mcd del módulo ejer61 en `__init__.py`



```
from ejer4 import suma
from ejer4 import mcd
```

Ejercicio 6.5

Crear un módulo ejer65pckge que utilice las funciones de packagePr1

The screenshot shows a Python code editor interface. At the top, there is a tab labeled "ejer45pckge.py". Below the tabs, the code editor displays two lines of Python code:

```
1 import packagePr1  
2 print(packagePr1.suma(3, 4))
```

Below the code editor is a "Run" toolbar. It features a green play button icon, an upward arrow icon, and a dropdown menu currently set to "ejer45pckge". To the right of the toolbar, the output window shows the command "C:\Python\Proyecto\venv\Scripts\" followed by the number "7", indicating the result of the script execution.

Ejercicio 6.5

Uso las funciones de packagePr1 desde línea de comandos

```
>>> import sys  
>>> sys.path.append('C:\\Python\\Proyecto')  
>>> import packagePr1  
>>> print(packagePr1.suma(3,4))  
7
```

Exportación de paquetes a .zip

- ◆ Como en Java el código de un paquete puede recogerse en un .zip:

```
>>> import zipfile  
>>> a=zipfile.PyZipFile('mipackage.zip', 'w', zipfile.ZIP_DEFLATED)  
>>> a.writepy('package_example')  
>>> a.close()  
>>> ^Z # ^D depende de SO
```

- ◆ Luego lo puedes importar y usar insertando su path en sys.path o alternativamente añadiendo a la variable de entorno PYTHONPATH una referencia al nuevo .zip [creado](#):

```
$ mkdir prueba; cp mipackage.zip prueba  
$ export PYTHONPATH=/home/dipina/examples/prueba/mipackage.zip  
>>> import sys # esta y la siguiente no hacen falta si se ha inicializado  
      PYTHONPATH  
>>> sys.path.insert(0, '/home/dipina/examples/prueba/mipackage.zip')  
>>> import package_example  
>>> class1 = package_example.module1.class1()  
>>> class1.show()  
class #1  
>>> ^Z
```

Ejercicio

Exportar en un zip el package packagePr1 y el módulo ejer61

The screenshot shows a Python development environment with two tabs open: "ejer46export.py" and "Ejer4b.py". The "ejer46export.py" tab contains the following code:

```
1 import zipfile  
2 a=zipfile.PyZipFile('packagePr1.zip', 'w', zipfile.ZIP_DEFLATED)  
3 a.writepy('packagePr1')  
4 a.writepy('ejer4.py')  
5 a.close()
```

The "Run" tab shows the command being run: "C:\Python\Proyecto\venv\Scripts\python.exe C:/Python/Proyecto/ejer46export.py". The output of the run is: "Process finished with exit code 0".

Ejercicio

Ver el zip creado con Python compilado

The screenshot shows the WinRAR application window. At the top, there's a menu bar with Archivo, Órdenes, Herramientas, Favoritos, Opciones, and Ayuda. Below the menu is a toolbar with icons for Añadir (Add), Extraer en (Extract to), Comprobar (Check), Ver (View), Eliminar (Delete), Buscar (Search), Asistente (Assistant), Información (Information), and Buscar virus (Scan for viruses). The main area displays the contents of a ZIP file named "packagePr1.zip". The file list shows:

Nombre	Tamaño	Comprimido	Tipo	Modificado	CRC32
..			Carpeta de archivos		
packagePr1			Carpeta de archivos		
ejer4.pyc	398	266	Compiled Python ...	23/09/2018 13:24	D17CB2C7

Nombre	Tamaño	Comprimido	Tipo
..			Carpeta de archivos
__init__.pyc	188	154	Compiled Python ...

Ejercicio

Utilizar el package desde el zip en otro proyecto

The screenshot shows the PyCharm IDE interface. The title bar reads "Proyecto [C:\Python\Proyecto] - ...2\ejer45usopackage.py [Proyecto2] - PyCharm". The menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The toolbars include Project, Run, and Stop. The left sidebar shows the "Project" view with "packagePr1.zip" and "Proyecto2 C:\Python\Proyecto" expanded. Inside "Proyecto2", there are "venv library root", "Ejer4b.py", "ejer45pckage.py", and "ejer45usopackage.py". The main editor window displays the code for "ejer45usopackage.py":

```
1 import sys
2 sys.path.insert(0, 'C:\\\\Python\\\\Proyecto\\\\packagePr1.zip')
3 import packagePr1
4 print(packagePr1.suma(5, 7))
```

The "Run" toolbar at the bottom shows the command: "C:\Python\Proyecto2\venv\Scripts\python.exe C:/Python/Proyecto2/ejer45usopackage.py". The status bar at the bottom right shows the number "12".

Librerías existentes

- Expresiones regulares
- XML / JSON
- Sockets
- Internet
- Numpy
- Interfaces gráficas con Tkinter