

Curso de Python: Nivel Inicial

Daniel Molina Cabrera

Curso de Python (Abril 2018)



CC BY-SA

Contenido del tema

- 1 Introducción sobre Python ¿Por qué Python?
- 2 Sobre el curso
- 3 Instalando Python y Entornos
- 4 Empezando con Python
- 5 Listas

Contenido del tema

- 6 Tipos cadena
- 7 Bucles e iteradores
- 8 Funciones y clases
- 9 Clases
- 10 Librerías
- 11 Uso de ficheros

Sobre mí



Sobre mí

- Daniel Molina Cabrera.
- Email: dmolina@decsai.ugr.es
- Profesor de Informática en Ceuta-UGR.
- Feliz programador de Python desde hace más de 15 años.
- Desarrollador oficial de varios paquetes Python en Pypi.
- Impartido varias asignaturas usando Python.
- Defensor a ultranza del software libre.

Índice

- 1 Introducción sobre Python ¿Por qué Python?
- 2 Sobre el curso
- 3 Instalando Python y Entornos
- 4 Empezando con Python
- 5 Listas

¿Qué es Python?



¿Qué es Python?



python

¿Qué es Python?



python

Es un lenguaje de programación

- Es un lenguaje para programar todo tipo de aplicaciones.
- Se diseñó para que fuese **fácil de usar** y **divertido**.

¿Quién lo inventó?

Python



- Inventado por Guido Van Rossum.
- Creado en 1989 en vacaciones de navidad.
- Pensado para enseñar programación a niños.
- Muy bien aceptado por la comunidad.
- No dependiente del autor: dilema del autobús.
- Influyente: Ruby, ...

¿Por qué Python? Es popular

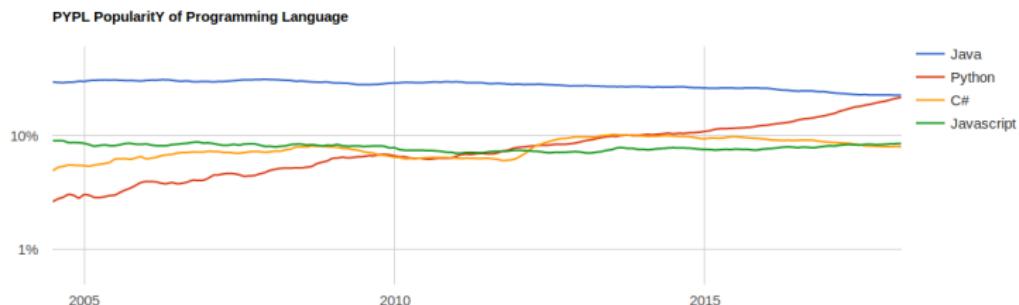


Figure: consulta de tutoriales (fuente: PYPL (Google Trends))

Es popular

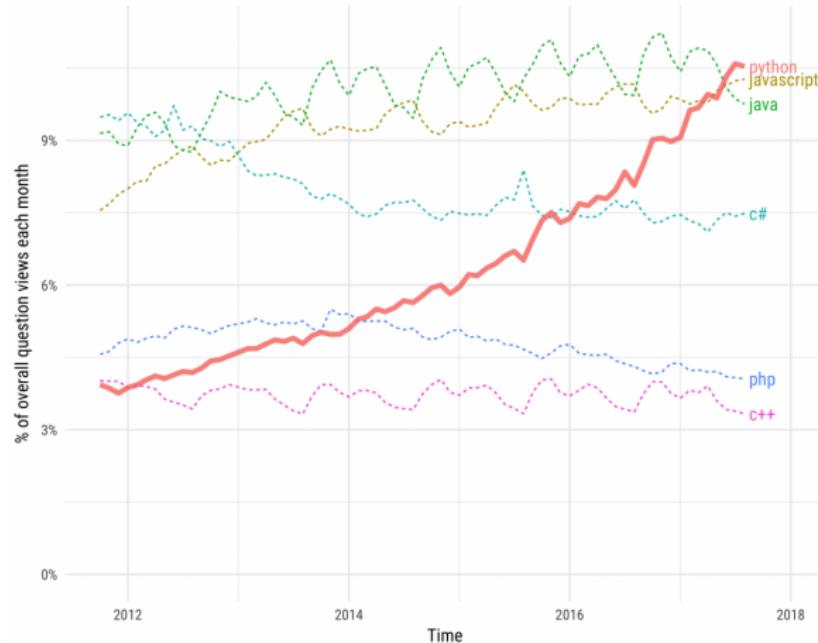
¿Por qué Python? Es popular

Table: evolución en TIOBE

Marzo 2018	Marzo 2017	Lenguaje	Porcentaje	Evolución
1	1	Java	14.941%	-1.44%
2	2	C	12.760%	+5.02%
3	3	C++	6.452%	+1.27%
4	5	Python	5.869%	+1.95%
5	4	C#	5.067%	+0.66%

Tengo unas gráficas más¹

Growth of major programming languages
Based on Stack Overflow question views in World Bank high-income countries

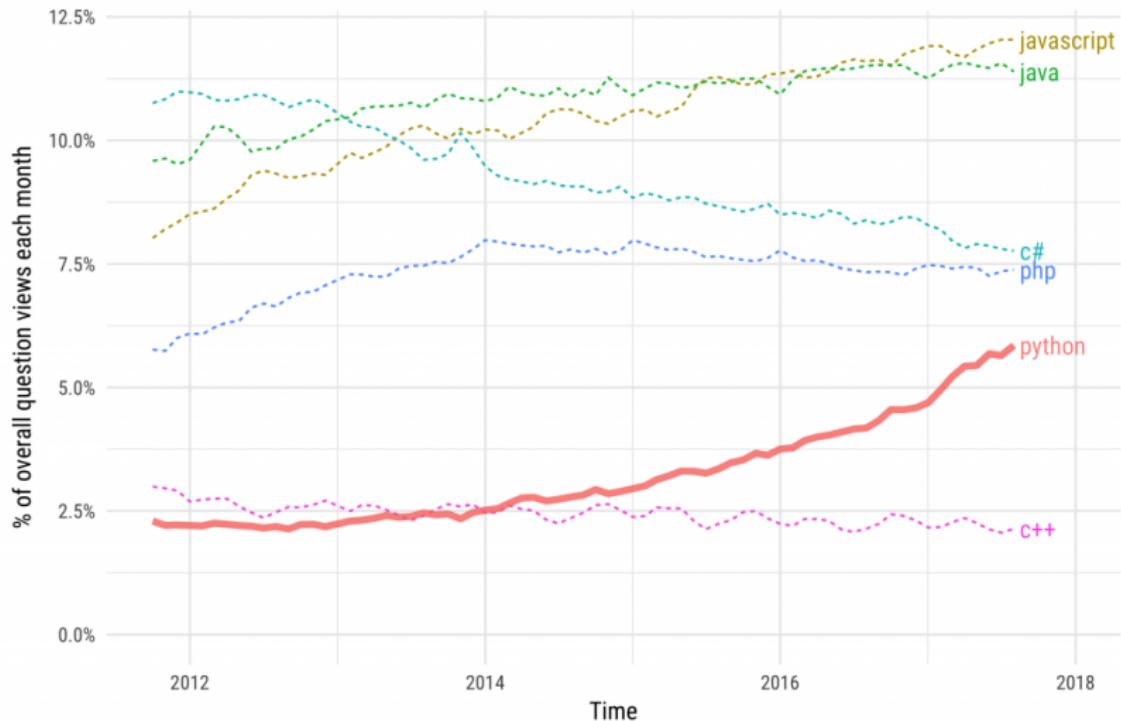


¹source: <https://bit.ly/2f54vYk>

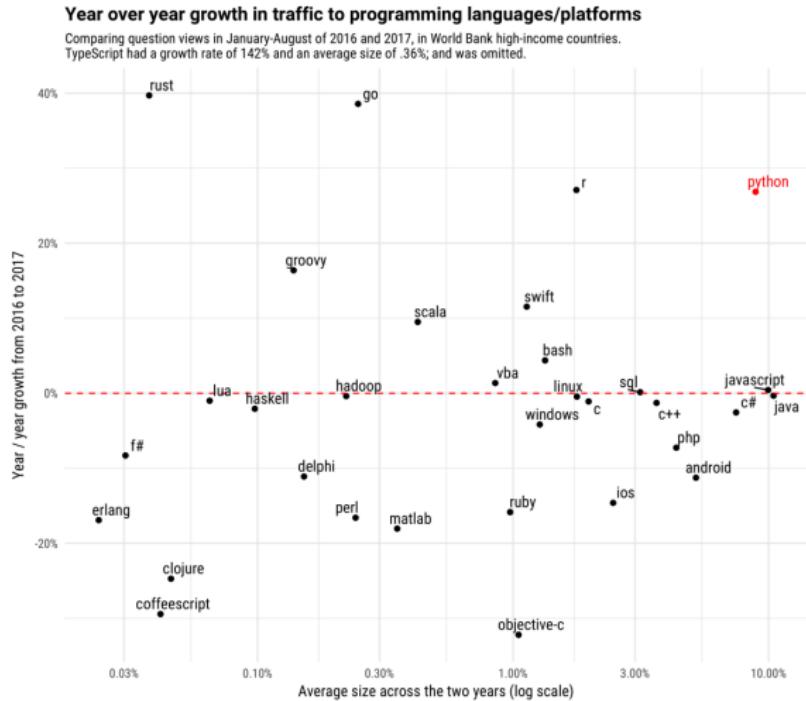
Muchas más gráficas ¹

Growth of major programming languages in non-high-income countries

Based on Stack Overflow question views in countries not classified as high-income by the World Bank.



Visión general de crecimiento¹



¿Quién lo usa?

Who uses Python?



Confidence Comes Standard.™



Massachusetts
Institute of
Technology



NOKIA
Connecting People

YAHOO!



CANONICAL

LexisNexis®

Consumer
Notebook



eBay



Firefox®

Google™



AstraZeneca



reddit

Quora

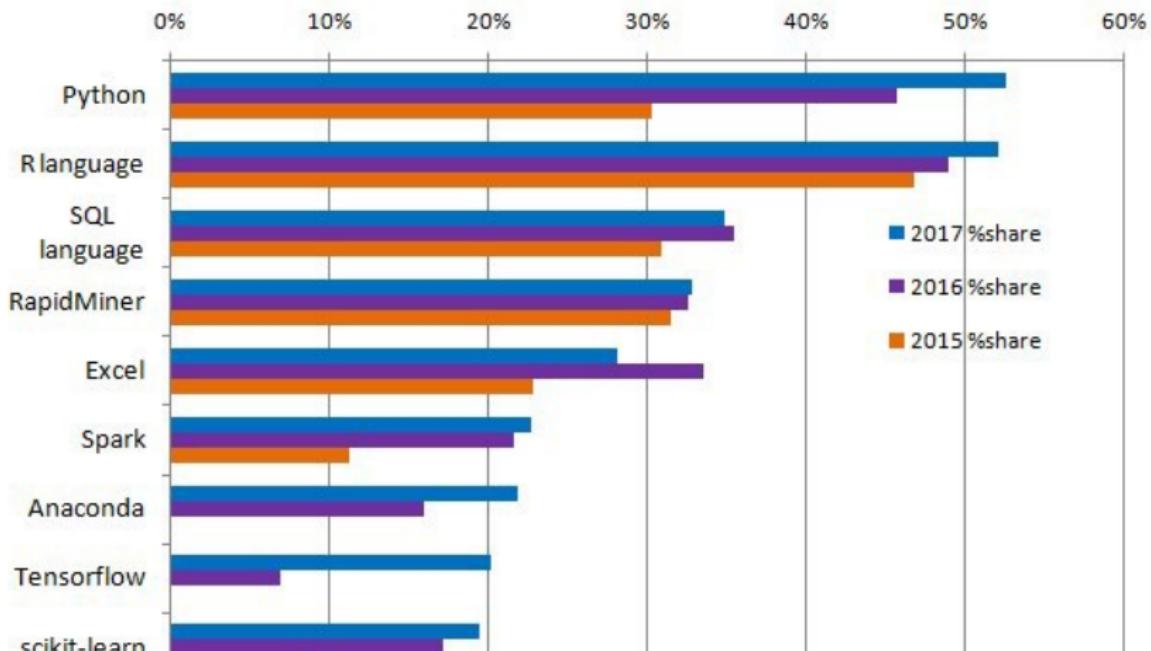
TRUECar.
Know the Real Price™



INDUSTRIAL
LIGHT & MAGIC
A Lucasfilm COMPANY

¿Y en temas actuales?

KDnuggets Analytics, Data Science, Machine Learning Software Poll, top tools share, 2015-2017



Pero eso no es lo importante



Ventajas de Python: Portabilidad

#1 Run it anywhere

Linux	Windows	
FreeBSD	Mac OS X	JVM
OpenBSD	Solaris	.NET
NetBSD	HP-UX	Android OS
BSD	OS/2	

http://en.wikipedia.org/wiki/CPython#Supported_platforms



Portabilidad

Es un lenguaje interpretado

- No se compila a ejecutable.
- Se necesita Python para ejecutar.
 - Disponible en distintos sistemas operativos.

Ejemplo de uso

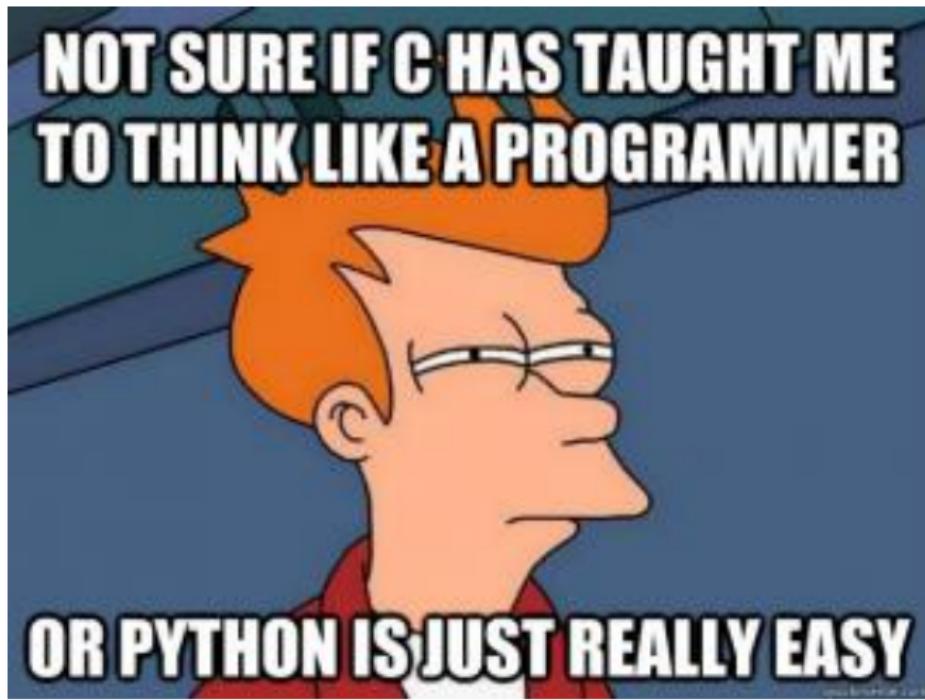
```
python hello.py
```

Otra opción (Linux)

```
#!/usr/bin/env python3
print("hola a todos")
```

```
chmod a+x hello.py
./hello.py
```

Ventajas de Python: Legibilidad



Ventajas de Python: Legibilidad

C/C++

```
#include <iostream>

int main(void) {
    std::cout <<"Hola a todos desde C++" <<std::endl;
}
```

Java

```
class Main {
    public static void main(String[] args) {
        System.out.println("Hola a todos desde Java");
    }
}
```

Python

```
print("Hola a todos desde Python\n")
```

Ventajas de Python: Legibilidad, Alto nivel

Admite variables alto nivel

```
list = ["fruta", "cereales", "berenjena"]

for item in list:
    print(item)
```

Ejemplo: Implementar programa grep

```
from sys import argv

def main(fname, word):
    with open(fname, "r") as file:
        for line in file:
            if word in line:
                print(line)
```

Ventajas de Python: Legibilidad, Alto nivel

"You can't just copy-pase pseudocode
into a program and expect it to work"



El Zen de Python

Zen de Python, de 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.

Criterios de diseño de Python

- Diseñado para simplicidad.

Ventajas de Python: Comunidad

Lenguaje *open source*

- Dirigido por la comunidad.
- Todo tipo de librerías.
- Repositorio de librerías disponibles.

Versátil

Aplicaciones escritorio PyQt,

automatización Fabric, celery, ...

Aplicaciones webs Django, Flask.

Software científico Numpy, Pandas, Matplotlib.

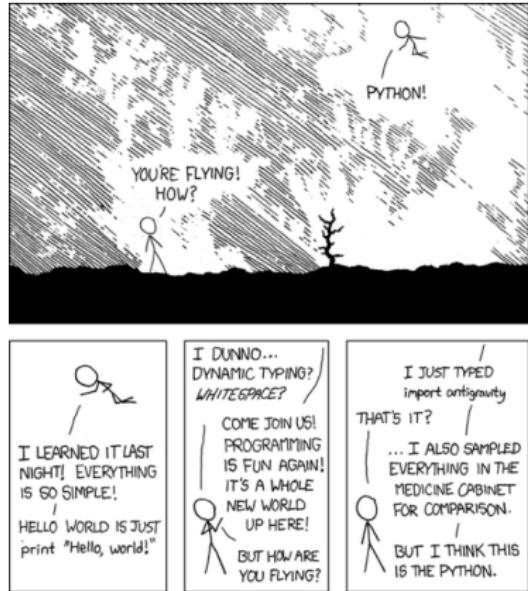
Machine Learning NLTK, TensorFlow, Scikit-learn.

Ventajas de Python: Comunidad



Figure: Librerías de software científico

Pero sobre todo



Índice

- 1 Introducción sobre Python ¿Por qué Python?
- 2 Sobre el curso
- 3 Instalando Python y Entornos
- 4 Empezando con Python
- 5 Listas

¿Qué necesito?



Requisitos

- Un ordenador.
- Da igual el SO: Windows, Linux, MacOS.
- Instalaremos Python3 usando Anaconda (opcional si ya está instalado).
- Ocupa espacio, pasare un *pendrive*.

¿Cómo será?



Interactivo

- Trabajaremos en Python.
- Estilo taller.

Sin conocimientos previos

- De otros lenguajes.
 - Siempre útil.
- Estilo *Pythonico*.

¿Qué veremos?

Parte 1: Python y Sintaxis

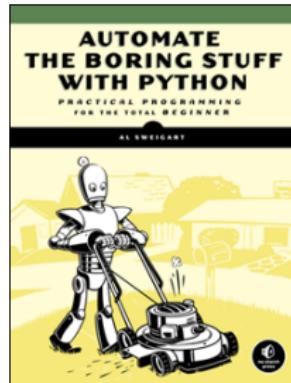
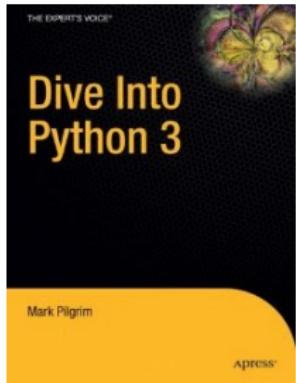
- Qué es y qué ofrece Python (Visto)
- Instalación de Python y entornos.
- Sintaxis de Python:
 - Tipos de datos.
 - Condicionales y bloques.
 - Bucles.
 - Manejo de listas e iteradores.
 - Definiendo funciones y clases.
 - Instalando y usando librerías.
 - Trabajando con ficheros

¿Qué veremos?

Parte 2: Uso de Python en distintos entornos

- Python en ciencia.
- Gráficos con Python.
- Páginas web con Python.

Bibliografía



Referencias

- <http://www.diveintopython.net/>
- <https://automatetheboringstuff.com/>

Índice

- 1 Introducción sobre Python ¿Por qué Python?
- 2 Sobre el curso
- 3 Instalando Python y Entornos
- 4 Empezando con Python
- 5 Listas

Instalando Python



Instalación

Disponible en

<https://docs.anaconda.com/anaconda/install/>

- Windows.
- Linux.
- MacOS.

Instalando Python

En pendrive

Windows .exe (<https://www.anaconda.com/download/#windows>):

//www.anaconda.com/download/#windows).

Linux .sh (<https://www.anaconda.com/download/#linux>):

//www.anaconda.com/download/#linux).

MacOS .pkg (<https://www.anaconda.com/download/#macos>):

//www.anaconda.com/download/#macos)

Instalando en Windows

Anaconda3 5.1.0 (64-bit) Setup



Choose Install Location

Choose the folder in which to install Anaconda3 5.1.0 (64-bit).

Setup will install Anaconda3 5.1.0 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\Users\anaconda\Anaconda3

[Browse...](#)

Space required: 2.5GB

Space available: 13.6GB

Anaconda, Inc.

< Back

[Next >](#)

Cancel

Instalando en Windows

Anaconda3 5.1.0 (64-bit) Setup



Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

Add Anaconda to my PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

Register Anaconda as my default Python 3.6

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.6 on the system.

Anaconda, Inc.

< Back

Install

Cancel

Instalando en Windows



Instalando en Windows

Anaconda3 5.1.0 (64-bit) Setup



ANACONDA.

Thanks for installing Anaconda3!

Anaconda is the most popular Python data science platform.

Share your notebooks, packages, projects and environments on Anaconda Cloud!

Learn more about Anaconda Cloud

Learn how to get started with Anaconda

< Back

Finish

Cancel

Instalando en MacOS

Install Anaconda3

Standard Install on "Macintosh HD"

- Introduction
- Read Me
- License
- Destination Select
- Installation Type**
- Installation
- Summary

This will take 1.44 GB of space on your computer.

Click **Install** to perform a standard installation of this software in your home folder. Only the current user of this computer will be able to use this software.

Change Install Location...

Customize **Go Back** **Install**



ANACONDA

Instalando en MacOS

Install Anaconda3

Select a Destination

- Introduction
- Read Me
- License
- Destination Select**
- Installation Type
- Installation
- Summary

How do you want to install this software?

-  Install for all users of this computer
-  **Install for me only**
-  Install on a specific disk...

Installing this software requires 1.44 GB of space.
You have chosen to install this software in your home folder.
Only the current user will be able to use this software.

Go Back Continue



ANACONDA

Instalando en MacOS

Install Anaconda3

Microsoft Visual Studio Code

- Introduction
- Read Me
- License
- Destination Select
- Installation Type
- Installation
- Microsoft VSCode**
- Summary

Anaconda has partnered with Microsoft to bring you Visual Studio Code. Visual Studio Code is a free, open source, streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more.

To install VS Code, you will require Internet Connectivity.

Click the button below to install Visual Studio Code

[Visual Studio Code License](#)

Install Microsoft VSCode

Go Back Continue



ANACONDA®

Instalando en MacOS

Install Anaconda3

The installation was completed successfully.

Anaconda is the leading open data science platform powered by Python.

Share your notebooks and packages on Anaconda Cloud!
[Sign up for free](#)

- Introduction
- Read Me
- License
- Destination Select
- Installation Type
- Installation
- Summary

 ANACONDA

Go Back Close

Instalando en Linux

Usando Anaconda

```
bash ~/Downloads/Anaconda3-5.1.0-Linux-x86_64.sh
```

Desde el sistema de paquetes

```
sudo apt install python3
python3 -m pip install --upgrade pip
python3 -m pip install jupyter
```

Entornos

Formato interactivo

`python` Línea de forma interativa.

`ipython/jupyter` interfaz con esteroides (autocompletado, . . .).

`ipython/jupyter notebook` Interfaz web.

Notebook

- Entorno desde el navegador.
- Fácil para pruebas rápidas (usaremos los primeros días).
- Formato de ficheros `.ipyb` aceptado por Github.

Editores Específicos de Python

`Tonny` Editor para aprendizaje.

`Spyder` Disponible en Anaconda, integrado con consola.

Ejemplo de entornos (Python por defecto)

```
[daniel@ubuntu:~/Descargas/opt/screen$ python3
Python 3.6.1 |Anaconda custom (64-bit)| (default, May 11 2017, 13:09:58)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> a = numpy.random.rand(30)
>>> for x in a:
...     print(x)
...
0.499233852424
0.9603781513162
0.17979869278
0.421647299513
0.869444245156
0.073945665169
0.788476969445
0.797646443367
0.101946844059
0.44467458971
0.41946447909
0.40539583659
0.285587840242
0.454929511648
0.151935443775
0.259959559369
0.692344940673
0.384080715125
0.882240175938
0.702248028962
0.0994603536405
0.14245217421
0.2624017151
0.859998137479
0.81760093835
0.170895118696
0.551435218441
0.346895972828
0.441309069151
0.341333841852
>>> print(a)
[ 0.49923385  0.96037815  0.17979869  0.4216473   0.86944425  0.07394567
  0.78847696  0.79764644  0.10194684  0.44467459  0.41946448  0.40539584
  0.28558784  0.45492951  0.15193544  0.25995955  0.69234494  0.38408072
  0.88224018  0.17089512  0.09946035  0.33081421  0.20242205  0.35999834
  0.81176009  0.17089512  0.55143522  0.34689597  0.44130907  0.34133384]
```

Figure: consola por defecto de python

Ejemplo de entornos (IPython/Jupyter)

The screenshot shows a terminal window titled "IPython: home/daniel". It displays the following session:

```
In [1]: import numpy
In [2]: str?
Init signature: str(self, /, *args, **kwargs)
Docstring:
    Create a new string object from the given object. If encoding or
    errors is specified, then the object must expose a data buffer
    that will be decoded using the given encoding and error handler.
    Otherwise, returns the result of object.__str__() (if defined)
    or repr(object).
    encoding defaults to sys.getdefaultencoding().
    errors defaults to 'strict'.
    Type:           type
In [3]: a = numpy.random.rand
```

A tooltip is visible over the command "a = numpy.random.rand", listing the available methods for the "random" module:

numpy.random.rand	numpy.random.random
numpy.random.randint	numpy.random.random_integers
numpy.random.randn	numpy.random.random_sample

Figure: consola de ipython/jupyter

Ejemplo de entornos (IPython/Jupyter notebook)

The screenshot shows a window titled "IP[y]: Notebook IPython tutorial (autosaved)". The window has a toolbar with File, Edit, View, Insert, Cell, Kernel, Help, and a "Notebook saved" button. Below the toolbar is a toolbar with various icons. The main area contains two code cells.

In [1]:

```
x = 1 + 1
print x
```

In [1]:

```
y = 2 * x
print y
```

NameError Traceback (most recent call last)
<ipython-input-1-1894ee302034> in <module>()
----> 1 y = 2 * x
 2 print y
NameError: name 'x' is not defined

Ejemplo de entornos (IPython/Jupyter notebook)

IP[y]: Notebook Untitled0 (autosaved)

File Edit View Insert Cell Kernel Help

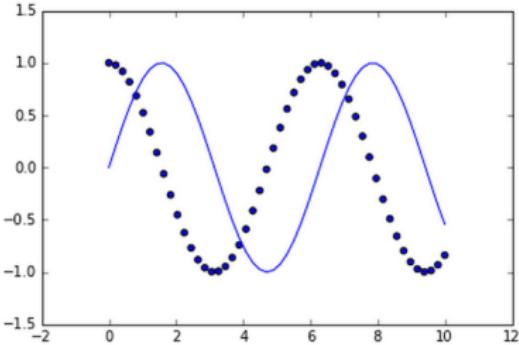
Cell Toolbar: None

In [1]: `%pylab inline`

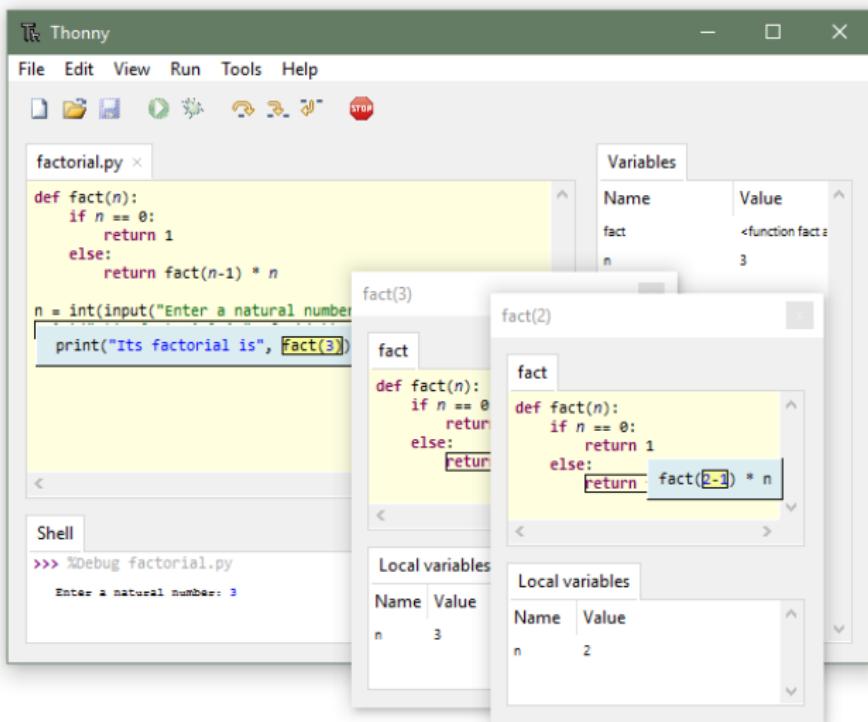
Populating the interactive namespace from numpy and matplotlib

In [6]: `x = np.linspace(0, 10, 50)
y = np.sin(x)
z = np.cos(x)
plot(x, y)
scatter(x, z)`

Out[6]: <matplotlib.collections.PathCollection at 0x10660a1d0>



Ejemplo de entornos (Tonny)



Ejemplo de entornos (Spyder)

The screenshot shows the Spyder Python IDE interface. The main window has a toolbar at the top with various icons for file operations, search, and tools. Below the toolbar, there are tabs for 'Editor' (containing 'Interpolation.py' and 'montecarlo_pi.py'), 'Object inspector', 'Source' (set to 'Console'), 'Object' (set to 'mean'), and 'Options'. The 'Object' tab displays the documentation for the 'mean' function from the numpy module.

mean(a, axis=None, dtype=None, out=None)
Function of numpy.core.fromnumeric module

Compute the arithmetic mean along the specified axis.

Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs.

Parameters

a : array_like
Array containing numbers whose mean is desired. If a is not an array, a conversion is attempted.

Object inspector Variable explorer File explorer

Console

In [2]: sin([1,2,3])
Out[2]: array([0.84147098, 0.90929743, 0.14112001])

In [3]: |

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 16 Column: 1

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
'''Simple generation of pi via MonteCarlo integration.
Taken from the Py4Science Workbook.
'''
import math
import random
import numpy as np
from scipy import weave
def v1(n = 100000):
    '''Approximate pi via monte carlo integration'''
    rand = random.random
    sqrt = math.sqrt
    sm = 0.0
    for i in xrange(n):
        sm += sqrt(1.0-rand())**2
    return 4.0*sm/n
def v2(n = 100000):
    '''Implement v1 above using weave for the C call'''
    support = '#include <stdlib.h>'
    code = '''
    double sm;
    float rnd;
    srand(1); // seed random number generator
    sm = 0.0;
    for(int i=0;i<n;++i) {
        rnd = rand()/(RAND_MAX+1.0);
        sm += sqrt(1.0-rnd*rand());
    }
    return sm;
    '''
    print code
    result = weave.inline(code, support=support, type_converters=weave.converters.blitz)
    return result
```

Editores e IDE



Editores Extensibles

Atom Editor de Github, muchos módulos.

SublimeText Editor no gratuito extensible, muy popular.

NeoVim Fork de vim.

Spacemacs Emacs preconfigurado.

IDE completos

PyCharm Versión community, módulos de pago.

PyDev Módulo de eclipse.

Ejemplo: PyCharm

Versiones

- Community (y Educativa).
- Comercial.

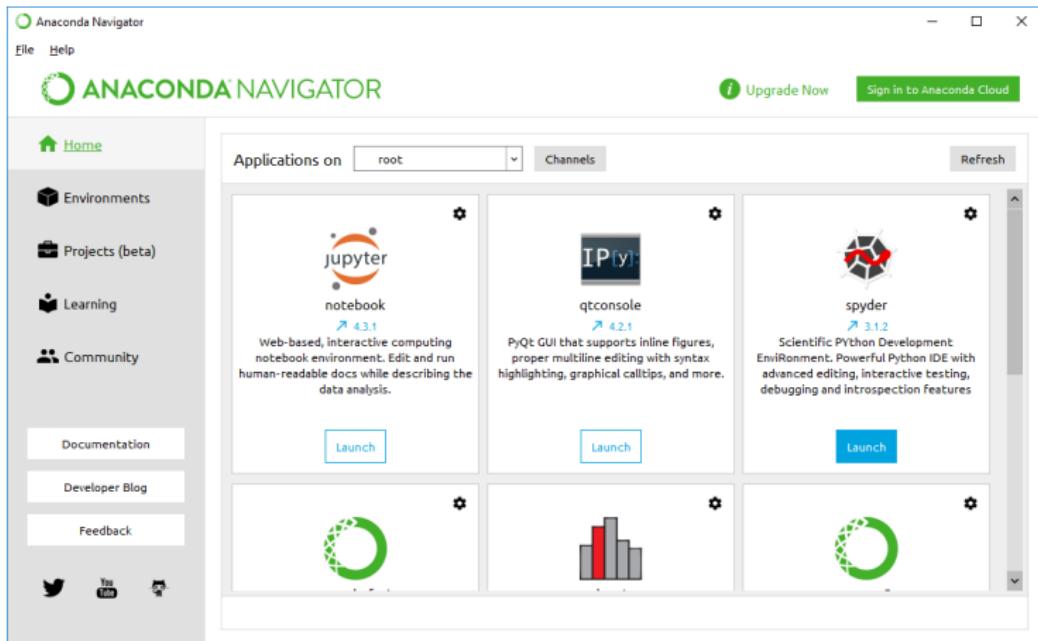
```
28     response = self.client.get(reverse('polls:index'))
29     self.assertEqual(response.status_code, 200)
30     self.assertContains(response, 'No polls are available.')
31     self.assertQuerysetEqual(response.context['latest_question_list'], [])
32
33     # Create a question.
34     def test_index_view_with_a_future_question(self):
35         """If a question has a pub_date in the future, it won't be displayed on
36         the index page.
37
38         """
39         question = create_question(question_text="Future question.", days=30)
40         response = self.client.get(reverse('polls:index'))
41         self.assertEqual(response.status_code, 200)
42         self.assertContains(response, 'No polls are available.')
43
44         #assertEqual(response.context['latest_question_list'], [])
45
46     def test_index_view_with_no_questions(self):
47         """If no questions exist, an appropriate message is displayed."""
48
49         response = self.client.get(reverse('polls:index'))
50         self.assertEqual(response.status_code, 200)
51         self.assertContains(response, "No polls are available.")
52
53     def test_index_view_with_past_question(self):
54         """Even if both past and future questions exist, only past questions
55         should be displayed.
56
57         """
58         question = create_question(question_text="Past question.", days=-30)
59         question = create_question(question_text="Future question.", days=30)
60         response = self.client.get(reverse('polls:index'))
61         self.assertEqual(response.status_code, 200)
62         self.assertContains(response, "Past question")
63
64     def test_index_view_with_two_past_questions(self):
```

25.18 LF: UTF-8: Cn: master: ip

Índice

- 1 Introducción sobre Python ¿Por qué Python?
- 2 Sobre el curso
- 3 Instalando Python y Entornos
- 4 Empezando con Python
- 5 Listas

Empezando con Python



Pasos

- 1 Lanzar "Anaconda Navigator".
- 2 Seleccionar Jupyter Notebook.

Variables

```
msg = "Hola a todos"  
print(msg)
```

Hola a todos

Hola a todos

Variables

- Las variables permiten guardar datos.
- Se accede a los valores usando el nombre de la variable.
- Se puede modificar los valores durante la ejecución.

```
a = 3  
b = 4  
print(a+b)  
a = a+1
```

Tipos de variables

Las variables pueden guardar distinto tipo de datos

Números entero number = 3

Números real number_real = 3.2

Cadena msg = "Hola"

Listas lista = [1, 2, 3]

Tabla hash datos = {'c': "cerrar", 'd': "delete"}

Diferencia

- No hay que definir el tipo de una variable.
- Una misma variable pueden tomar valores de distinto tipo (no recomendado).

```
variable = 4
```

```
variable = "hola"
```

Ejemplo de uso

Ejemplo de uso

```
print(number+1)
print(number_real-1)
print(lista)
print(datos)
```

```
4
2.2
[1, 2, 3]
{'c': 'cerrar', 'd': 'delete'}
```

Aviso

Python tiene tipos, no se permiten operaciones entre tipos.

```
print(number+lista)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    
```

Asignación de tipos en Python3

En Python3 se incorporó definir tipos en variables

```
count: int = 4
```

Vamos a meter un error

```
count: int = 4
```

```
count += 1.5
```

El analizador (mypy) detecta errores en tipos

```
python3 ej_typing.py
```

5.5

```
python3 -m mypy ej_typing.py # or myp ej_typing.py
```

```
ej_typing.py:2: error: Incompatible types in assignment (expression has type "float",  
variable has type "int")
```

Usando variables

Tipos entero

```
number = 3  
print(number+3)  
print(number/2)
```

6
1.5

División

- Dividir números enteros devuelve un número real.
- La división entera es otra operación: `//`.

```
number = 3  
print(number//2)
```

Usando variables

Tipo real

```
number = 3  
number_real = number_real + number  
print(number_real)
```

9.2

Tipo cadena

```
msg = "holo"  
print("a" in msg)  
msg = msg + " adios"  
print(msg)
```

True

holo adios

Usando variables

Tipo List

```
lista2 = lista + [6]
print(lista)
print(lista2)
print(4 in lista)
print("El primer valor es ", lista2[0])
print("Los dos siguientes son", lista2[1:3])
print("Los siguientes son", lista2[1:])
```

```
[1, 2, 3]
[1, 2, 3, 6]
False
El primer valor es 1
Los dos siguientes son [2, 3]
Los siguientes son [2, 3, 6]
```

Bloques y Condicionales

Condicionales

- Aplicar el mismo código siempre igual no es *divertido*.
- Un programa puede ejecutar código según una condición.

Ejemplo

```
print("Dime un numero")
numero = int(input())

if numero == 7:
    print("Has acertado!!!\n")
else:
    print("Has fallado, mas suerte para otra\n")
```

Bloques y condicionales

Definiendo los bloques

- Los bloques se marcan en otros lenguajes usando { . . . }.
- Por legibilidad se debe tabular.
- Python se guía de la tabulación, es necesario un editor adecuado.

Ejemplo

```
cantidad = int(input())

if cantidad < 1000:
    if cantidad < 100:
        print("Eso es una misería")
    else:
        print("Eso es poco")
else:
    print("Eso es mucho")
```

Condicionales

Ejemplo

```
if number > 5:  
    print("mayor que 5")
```

Formato

- Tras palabra **if** se indica una condición y un **:**.
- El código tabulado se ejecuta sólo si la condición se cumple.
- Puede ponerse un **else**, se ejecuta si no se cumple.

Con else

```
if number > 5:  
    print("mayor que 5")  
else:  
    print("menor o igual que 5")
```

¿Y el switch?

No tiene switch

- Tiene muchas limitaciones en otros lenguajes.
- tiene el **elif**.

Ejemplo

```
if number > 0:  
    print("mayor que 0")  
elif number > 5:  
    print("mayor que 5")  
else:  
    print("menor o igual que 5")
```

Índice

- 1 Introducción sobre Python ¿Por qué Python?
- 2 Sobre el curso
- 3 Instalando Python y Entornos
- 4 Empezando con Python
- 5 Listas

Listas

Listas

- Permiten guardar varios valores en una variable.
- Pueden ser de tipos distintos (no recomendable).
- Se pueden acceder mediante corchetes y posición:
 - 0 ⇒ primer elemento.
 - 1 ⇒ segundo elemento.
 - ...

Ejemplos

```
lista1 = ['monty', 'python', 42]
lista2 = [1, 2, 3, 4, 5]
print(lista1[1], lista1[2], lista2[3])
```

```
python 42 4
```

Modificando elementos

Cambiando el valor

Directamente `lista[posicion] = nuevo valor.`

Añadiendo elementos

Al final Método `append` (lo más eficiente).

Varios al final Operador `+` (ambos deben ser listas).

En medio Método `insert` (`insert(posicion, valor)`).

Borrando elementos

Todos los elementos Método `clean`.

Elemento concreto `del lista[posicion]`.

Modificando listas

Ejemplos

```
lista = [1, 2, 3, 4, 5]
print(lista)
lista[2] *= 2
print(lista)
lista.append(9)
print(lista)
lista.insert(0, 0)
print(lista)
del lista[4]
print(lista)
lista = lista + [10, 11]
print(lista)
lista.clear()
print(lista)
```

Salida

```
[1, 2, 3, 4, 5]
[1, 2, 6, 4, 5]
[1, 2, 6, 4, 5, 9]
[0, 1, 2, 6, 4, 5, 9]
[0, 1, 2, 6, 5, 9]
[0, 1, 2, 6, 5, 9, 10, 11]
[]
```

Accediendo elementos

Accediendo elementos

Único elemento `lista[posicion]`

Rango de elementos entre `[inicio, fin]` `lista[inicio: fin]`

Rango de elementos con salto `lista[inicio: fin : salto]`

Rango antes de una posición `lista[: posicion]`

Rango desde de una posición `lista[posicion :]`

Rango desde de una posición `lista[: posicion]`

¿Tamaño?

`len()` El número de elementos, vale para muchos tipos.

Accediendo elementos

```
lista = [1, 2, 3, 4, 5, 6]
print(lista[3])
print(lista[1:3])
print(lista[0:3])
print(lista[:3])
print(lista[3:])
print(lista[0:6:2])
print(lista[::-2])
print(lista[:])
print(lista[::-1])
print(lista[::-2])
```

Salida

```
4
[2, 3]
[1, 2, 3]
[1, 2, 3]
[4, 5, 6]
[1, 3, 5]
[1, 3, 5]
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[6, 5, 4, 3, 2, 1]
```

Las listas son referencias

Cuidado con variables

```
lista = [1, 2, 3, 4, 5, 6]
lista2 = lista
lista2[2] = 0
print("Lista2: ", lista2)
print("Lista original: ", lista)
```

Salida

```
Lista2: [1, 2, 0, 4, 5, 6]
Lista original: [1, 2, 0, 4, 5, 6]
```

Cuidado con las variables

- Ambas variables contienen la misma lista.
- Modificando una variable se modifica el valor de la otra.

Solución: hacer copias

Cuidado con variables

```
lista = [1, 2, 3, 4, 5, 6]
lista2 = lista[:]
lista2[2] = 0
print("Lista2: ", lista2)
print("Lista original: ", lista)
```

Salida

```
Lista2: [1, 2, 0, 4, 5, 6]
Lista original: [1, 2, 3, 4, 5, 6]
```

Índice

- 6 Tipos cadena
- 7 Bucles e iteradores
- 8 Funciones y clases
- 9 Clases
- 10 Librerías
- 11 Uso de ficheros

Cadenas

Son listas

```
msg = "hola"  
print(len(msg))  
print(msg[1])  
print("Caracteres: ")  
  
for c in msg:  
    print(c)
```

Salida

```
4  
o  
Caracteres:  
h  
o  
l  
a
```

Índice

- 6 Tipos cadena
- 7 Bucles e iteradores
- 8 Funciones y clases
- 9 Clases
- 10 Librerías
- 11 Uso de ficheros

Tipos de bucle: while

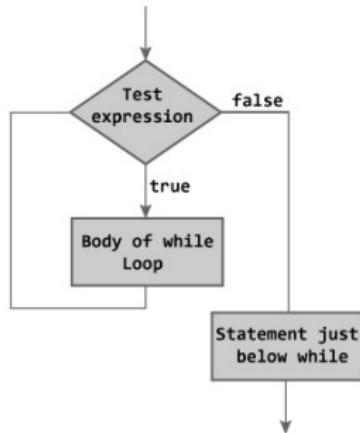


Figure: Flowchart of while Loop

Ejemplo

```
num = 1
```

```
while num <= 5:  
    print(num)  
    num = num + 1
```

```
1  
2  
3  
4  
5
```

Bucles while

Ejecuta el código tabulado mientras se cumpla la condición.

Tipos de bucle: for

Ejemplo

```
lista = ['a', 'b', 'c', 'd',  
  
for num in lista:  
    print(num)
```

Salida

```
a  
b  
c  
d  
e
```

Bucles for

Formato for <variable> in <lista>: bloque

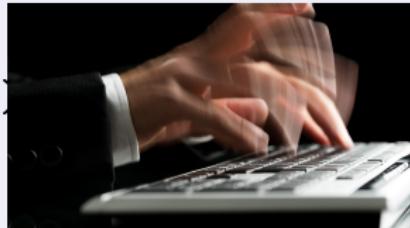
Significado

- Por cada elemento de la lista:
 - Asigna su valor en la variable.
 - Ejecuta el bloque de código tabulado.

Tipos de bucle: for

Programador acostumbrado a otros lenguajes

```
for i in range(0, len(lista))  
    print(lista[i])
```



Programador *pythonico*

```
for elem in lista:  
    print(elem)
```



Tipos de bucle: for

¿Y si necesito el índice?

```
for i, elem in enumerate(lista):  
    print("El elemento", i, "vale", elem, "==" , lista[:]
```

```
El elemento 0 vale 1 == 1  
El elemento 1 vale 2 == 2  
El elemento 2 vale 3 == 3  
El elemento 3 vale 4 == 4  
El elemento 4 vale 5 == 5  
El elemento 5 vale 6 == 6
```

Función enumerate

Recibe una lista, devuelve además de cada elemento de la lista, la posición (empezando por cero).

Tipos de bucle: for

¿Y si necesito recorrer varias listas a la vez?

```
nombres = ['Daniel', 'Amalia', 'Carlos', 'Rosa']
pies = [43, 41, 44, 42]
```

Option 1: estilo C

```
for i in range(len(nombres)):
    print("Usuario", nombres[i], "tiene pie", pies[i])
```

Option 2: Recorrido con enumerate

```
for i, nombre in enumerate(nombres):
    print("Usuario", nombre, "tiene pie", pies[i])
```

Option 3: uso de zip

```
for nombre, pie in zip(nombres, pies):
    print("Usuario", nombre, "tiene pie", pie)
```

Recibe varias listas, devuelve cada elemento de todas.

Formato inline de for

Notación más matemática

```
lista = [1, 2, 3, 4, 5]
doble = [2*x for x in lista]
print(doble)
doble_par = [2*x for x in lista if x % 2 == 0]
print(doble_par)
```

```
[2, 4, 6, 8, 10]
[4, 8]
```

Formato

- [expresion **for** variable **in** lista]
- [expresion **for** variable **in** lista **if** condicion]

Concepto de iterador

Iterador:

- Función que devuelve una serie de elementos.
- Se usan en los bucles for.
- Se generan los elementos en cada ejecución del bucle, ahorra memoria.

Ejemplo: range()

```
ran = range(3)
# No muestra la lista
print(ran)
# Se puede recorrer ahora
for x in ran:
    print(x)
# O directamente
for x in range(3):
```

Salida

```
range(0, 3)
0
1
2
0
1
2
```

Otros iteradores

Métodos iteradores clásicos

`map` Aplica una función a cada elemento de una lista(secuencia).

`filter` Filtra los elementos de una lista.

Ejemplo

```
pares = filter(espar, lista)  
for x in pares:  
    print(x)
```

espar

```
def espar(x):  
    return x % 2 == 0
```

Salida

```
2  
4  
6
```

Posibles problemas con iteradores

Range es seguro

- Otros métodos: map, filter, ... no lo son.
- Eso implica que sólo se pueden ejecutar una vez.

Ejemplo

```
pares = filter(espar, lista)
print("Primera vez")

for x in pares:
    print(x)

print("Repetimos")

for x in pares:
    print(x)
```

Salida

```
Primera vez
2
4
6
Repetimos
```

Possible solución

```
pares = list(pares)
```

Índice

- 6 Tipos cadena
- 7 Bucles e iteradores
- 8 Funciones y clases
- 9 Clases
- 10 Librerías
- 11 Uso de ficheros

Funciones

Funciones

- Permiten no repetir el mismo código una y otra vez.
- Se hace una única vez y se repite.

Ejemplo

```
def suma(lista):
    sum = 0

    for item in lista:
        sum += item

    return sum

print(suma([1, 3, 5]))
# Suma de 10 a 20
```

Salida

```
9
165
286
```

Sintaxis de las funciones

Formato:

```
def nombreFuncion(parametros):  
    # Código  
    # return salida
```

Bloque

La tabulación limita el código dentro de la función.

Estructura

- La función puede recibir parámetros.
- Puede devolver un valor mediante `return`, pero no es obligatorio.

Ejemplo de funciones

Máximo y mínimo

```
def mymax(value1, value2):  
    if value1 >= value2:  
        return value1  
    else:  
        return value2
```

```
print(mymax(3, 5))  
print(mymax(4, 2))  
print(mymax(0, -2))
```

5
4
0

Funciones que devuelven valores

Funciones

Las más comunes son las que devuelven (*return algo*).

Función que devuelve

```
def add_vector(vector1, vector2):
    result = []

    for item1, item2 in zip(vector1, vector2):
        result.append(item1+item2)

    return result

print(add_vector([1, 2, 3], [4, 2, 3]))
[5, 4, 6]
```

Funciones que no devuelven valores

Función que no devuelve

```
def add_vector(vector1, vector2, result):
    result.clear()

    for item1, item2 in zip(vector1, vector2):
        result.append(item1+item2)

result = []
print("Imprimimos lo que devuelve la funcion")
print(add_vector([1, 2, 3], [4, 2, 3], result))
print("Ahora la variable de salida")
print(result)
```

Imprimimos lo que devuelve la funcion

None

Ahora la variable de salida

[5, 4, 6]

Paso de parámetros

Ejemplo con un entero

```
def fun1(param):  
    print(param)  
    param = 4  
    print(param)
```

```
variable = 3  
fun1(variable)  
print(variable)
```

3

4

3

Ejemplo con lista

```
def fun2(param):  
    print(param)  
    param.clear()  
    print(param)
```

```
variable = [3, 5, 7, 10]  
fun2(variable)  
print(variable)
```

[3, 5, 7, 10]

[]

[]

Diferencia

- El parámetro *name* apunta al mismo dato que la variable que se pasa en la llamada (*variable*).

Parámetrosopcionales

Parámetrosopcionales

- Las funciones pueden tener parámetros por defecto.

```
def increm(value, increm=10):  
    return value+increm  
  
print(increm(3, 2))  
print(increm(3, 10))  
print(increm(3))
```

5

13

13

Posición de los parámetros por defecto

- Deben aparecer al final.
- Si no fuese así, no quedaría claro cómo interpretarlo si

Parámetros con nombre

Parámetros con nombre

No es necesario poner los parámetros en orden si se sabe su nombre.

Ejemplo: copia de vector

```
def mycopyvec(source, dest):
    dest.clear()

    # or dest.extend(source)
    for elem in source:
        dest.append(elem)

list1, list2 = [3, 4, 5], [2, 9, 7]
mycopyvec(list1, list2)
print("Lista1: ", list1)
print("Lista2: ", list2)
```

Salida

```
Lista1: [3, 4, 5]
Lista2: [3, 4, 5]
Lista1: [2, 9, 7]
Lista2: [2, 9, 7]
```

Índice

6 Tipos cadena

7 Bucles e iteradores

8 Funciones y clases

9 Clases

10 Librerías

11 Uso de ficheros

Clases

Concepto de clases

- En ciertos dominios requieren cierta información (Board).
- Se realiza una serie de operaciones con la información.
- Por comodidad se puede agrupar juntas.

Si tenemos esto

```
x = ...
y = ...
draw_board_battleship(x, y, size, fname_background, ...
x += 1
draw_board_battleship(x, y, size, fname_background, ...)
```

Errores

- Para un simple concepto puede haber muchas variables.
- Las funciones requieren muchos parámetros.

Clases

Es la forma que Python agrupa información

- Agrupa variables y funciones sobre dichos datos.
- Una instancia de la clase (variable) guarda esos datos como atributos.
- No es necesario pasarle los datos, lo recoge de la variable.

Ejemplo

```
board = Board(x, y, size)
board.setBackground(fname_background)
board.draw()
board.move(inc_x=1)
board.draw()
```

Clases

Python posee clases

- Python tiene **clases** como C++ o Java.
- No posee estructuras.
- No exige su uso como Java.
- Se combina con enfoque estructural cuando conviene.

Modelo de clases de Python

- Ofrece la misma funcionalidad que otros.
- Tiene sus particulares.
 - No restricción de permisos.
 - Convenios de nomenclatura.
 - Propiedades.
 - Métodos especiales: str, init, ...

Lo vemos por comparación

Java

```
public class Point {  
    private int xCoord;  
    private int yCoord;  
  
    public Point() {  
        xCoord = yCoord = 0;  
    }  
    public Point( int x, int y ) {  
        xCoord = x;  
        yCoord = y;  
    }  
    public String toString() {  
        return "(" + xCoord + ", " + yCoord + ")";  
    }  
    public int getX() { return xCoord; }  
}
```

Lo vemos por comparación

C++

```
class Point {  
private:  
    int xCoord;  
    int yCoord;  
  
public:  
    Point() {  
        xCoord = yCoord = 0;  
    }  
    Point(int x, int y) {  
        xCoord = x;  
        yCoord = y;  
    }  
    int getX() { return xCoord; }  
    int getY() { return yCoord; }  
}
```

Lo vemos por comparación

Python

```
class Point:

    def __init__(self, x = 0, y = 0):
        self.xCoord = x
        self.yCoord = y

    def __str__(self):
        return "({},{})".format(self.xCoord, self.yCoord)

    def getX(self):
        return self.xCoord

    def getY(self):
        return self.yCoord
```

Peculiaridades de clases

No límites de acceso

- No hay límites de acceso.
- Convenio: si empieza por "__" no es público, no se debe acceder.

Definir atributos

- No sección especial.
- Se hace con `self.variable = ...` en el constructor.

Métodos especiales:

`__init__` Constructor.

`__str__` Método para mostrar valores (conversión a cadena, ...).

Métodos en las clases

Métodos:

- Son funciones normales.
- Recibe como primer parámetro el propio objeto (`self`).
- El objeto `self` se usa para acceder a los atributos/métodos.
- Todo método siempre tiene algún parámetro.

Ejemplo de uso

```
point = Point()  
print(point)  
point2 = Point(2, 3)  
print(point2)  
print(point2.getX())  
point2.shift(1, -1)  
print(point2)  
# Acceso directo, no recomendado
```

Salida

```
(0,0)  
(2,3)  
2  
(3,2)  
(4,2)
```

Herencia

Clase base

```
class Polygon:  
    def __init__(self, no_of_sides):  
        self.n = no_of_sides  
        self.sides = [Point(0,0) for i in range(self.n)]  
  
        def setPoint(self, i, x, y):  
            assert i >= 0 and i < self.n  
            self.sides[i] = Point(x, y)  
  
        def __str__(self):  
            return ", ".join([str(t) for t in self.sides])  
  
a = Polygon(2)  
a.setPoint(0, x=1, y=2)  
a.setPoint(1, x=1, y=1)
```

Herencia

Clase heredada

```
def copy(point, incX=0, incY=0):
    return Point(point.getX()+incX, point.getY()+incY)

class Rectangle(Polygon):
    def __init__(self, init, size):
        Polygon.__init__(self, 5)
        self.sides[0] = copy(init)
        self.sides[1] = copy(init, incY=size)
        self.sides[2] = copy(init, incX=size, incY=size)
        self.sides[3] = copy(init, incX=size)
        self.sides[4] = copy(init)

a = Rectangle(Point(0, 0), 3)
print(a)
a = Rectangle(Point(2, 2), 1)
```

Propiedades

Acceso a atributos

- Python no suele usar métodos getX, setX.
- Se puede acceder directamente a los métodos.
- ¿Y si se desea limitar?

Propiedades

- Permite asignar método de asignación y consulta.
- Es transparente.

Propiedades

Ejemplo

```
class Temperature:  
    def __init__(self, temperature):  
        self.celsius = temperature
```

```
@property  
def farenheit(self):  
    return self.celsius*1.8+32
```

```
@farenheit.setter  
def farenheit(self, value):  
    self.celsius = (value-32)/1.8
```

```
temp = Temperature(30)  
print(temp.celsius)  
print(temp.farenheit)
```

Salida

```
30  
86.0  
40  
104.0  
100.0  
37.77777777777778
```

Índice

- 6 Tipos cadena
- 7 Bucles e iteradores
- 8 Funciones y clases
- 9 Clases
- 10 Librerías
- 11 Uso de ficheros

Importando paquetes



No tienes que hacer todo el código

- Reutilizar código externo, librerías/paquetes.
 - Funciones o clases que podemos usar.

Estructurando el código

Dividiendo el programa

- Podemos poner varias funciones en ficheros distintos.
 - Ej: utils.py, game.py, board.py, ...
- Queremos usar esas funciones dentro de otros ficheros.

Paquetes

- Un paquete es un fichero (o varios) con funciones/clases.
- Puede ser usado en otros ficheros.

Distinto caso

Librería externa Nombre único del paquete, identifica la librería.

Fichero local El nombre del paquete es el del fichero (sin .py).

Conflictos de nombres

Conflictos de nombres

- Una misma función puede existir en varios paquetes.

Import

```
import package  
...  
package.fun(...)
```

Sin usar import

```
print(sqrt(9))
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'sqrt' is not defined
```

Con import

```
import math  
print(math.sqrt(9))
```

3.0

Otras opciones

Indicar las funciones usadas de cada paquete

```
from package import fun1, fun2, ..., funN
```

Ejemplo

```
from math import sqrt  
print(sqrt(9))
```

¿Y si hay muchas funciones?

- Podría ser mejor usar paquete.

```
import numpy as np
```

```
a = np.zeros(5)  
print(a)
```

Otras opciones

Indicar las funciones usadas de cada paquete

```
from package import fun1, fun2, ..., funN
```

Ejemplo

```
from math import sqrt  
print(sqrt(9))
```

¿Y si hay muchas funciones?

- Podría ser mejor usar paquete. ⇒ ¿Y si no me gusta?

```
import numpy as np
```

```
a = np.zeros(5)  
print(a)
```

Otras opciones

Indicar las funciones usadas de cada paquete

```
from package import fun1, fun2, ..., funN
```

Ejemplo

```
from math import sqrt  
print(sqrt(9))
```

¿Y si hay muchas funciones?

- Podría ser mejor usar paquete. ⇒ ¿Y si no me gusta?

Uso de alias

```
import numpy as np
```

```
a = np.zeros(5)  
print(a)
```

Salida

```
[ 0.  0.  0.  0.  0.]
```

Ejemplo local

Fichero utils.py

```
def print_hello():
    print("Hola")

def print_adios():
    print("Adios")
```

Fichero main.py (mismo directorio)

```
import utils

utils.print_hello()
print("Bla Bla Bla")
utils.print_adios()
```

Hola
Bla Bla Bla
Adios

Programa main

Al hacer import se ejecuta el fichero

- No es adecuado poner código fuera de las funciones.
- Es conveniente una función **main**.

Función main

```
def main():
    print("Hola a todos")

if __name__ == "__main__":
    main()
```

Hola a todos

Comentarios

- La función puede tener cualquier nombre.
- Se garantiza que sólo se ejecuta como programa, no por

Instalar paquetes



Instalación

- Python permite instalar muy fácilmente paquetes.
- Existe un repositorio oficial de paquetes: PyPI.

Pip programa para buscar/installar/borrar paquetes (y sus dependencias).

Uso de Pip

Buscar: search

```
pip search cec2013lsgo
```

```
cec2013lsgo (2.1) - Package for benchmark for the Real ...
Evolutionary Computation CEC'2013
INSTALLED: 2.0
LATEST: 2.1
```

Información: show

```
pip show cec2013lsgo
```

```
Name: cec2013lsgo
Version: 2.0
Summary: Package for benchmark for the Real ...
Home-page: https://github.com/dmolina/cec2013lsgo
Author: Daniel Molina
Author-email: dmolina@decsai.ugr.es
License: GPL V3
Location: /mnt/home/daniel/anaconda3/lib/python3.6/site-packages/...
Requires: cython, numpy
```

Uso de Pip

Instalar: install

```
pip install numpy
```

```
Requirement already satisfied: numpy in ...
```

Borrar: uninstall

```
pip uninstall cec2013lsgo
```

```
Successfully uninstalled cec2013lsgo-2.0
```

Uso de pip y permisos

¿Y si no tengo permisos?

Se puede instalar en el \$HOME del usuario con:

```
pip install numpy --user
```

Aviso

Es recomendable usarlo y no hacer "sudo pip".

Cargando un módulo

Librerías y ejecutables

- Hay librerías que ofrecen programas ejecutables.
- También la librería puede contener una función main asociada.

Cargando un módulo

- Con la opción "-m" se puede cargar un módulo.

Ejemplo: Carga de módulo

Detección estática

```
python -m mypy check.py
```

Otro ejemplo: servidor web

```
python3 -m http.server
```

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ... 127.0.0.1:5000
[19/Apr/2018 18:57:45] "GET / HTTP/1.1" 200 -
```

Conflictos entre paquetes

Possible situación

- Paquete1 ⇒ v2.0 de Paquete4
- Paquete3 ⇒ v3.0 de Paquete4

¿Cómo se puede resolver?

- Se puede instalar las librerías en directorios distintos.
 - Paquete1 y v2.0 de Paquete4 en uno.
 - Paquete3 y v3.0 de Paquete4 en otro.
- Activar el entorno que queremos (usar las librerías de un directorio u otro).
- Así además si la librería incluye ejecutable lo tendremos configurado.

Herramientas de entornos

Conda

- Específico de Anaconda.
- No compatible con otros.

pipenv

- Muy reciente (1 año y poco).
- Considerado el estándar.

Otros (pyenv, venv)

- Anteriores a pipenv.

Conflictos de paquetes: uso de conda

conda

Ver los entornos `conda env list`. El actual está marcado con *.

Crear un entorno `conda create -n nombre [opciones]`.

Activar entorno `source activate nombre`.

Desactivar entorno actual `source deactivate`.

Ejemplo de uso de conda

Lista

```
conda env list
```

Salida

```
# conda environments:  
#  
base          * /mnt/home/daniel/anaconda3  
IA            /mnt/home/daniel/anaconda3/envs/IA  
curso        /mnt/home/daniel/anaconda3/envs/curso  
wcci2018     /mnt/home/daniel/anaconda3/envs/wcci2018
```

Uso de entorno

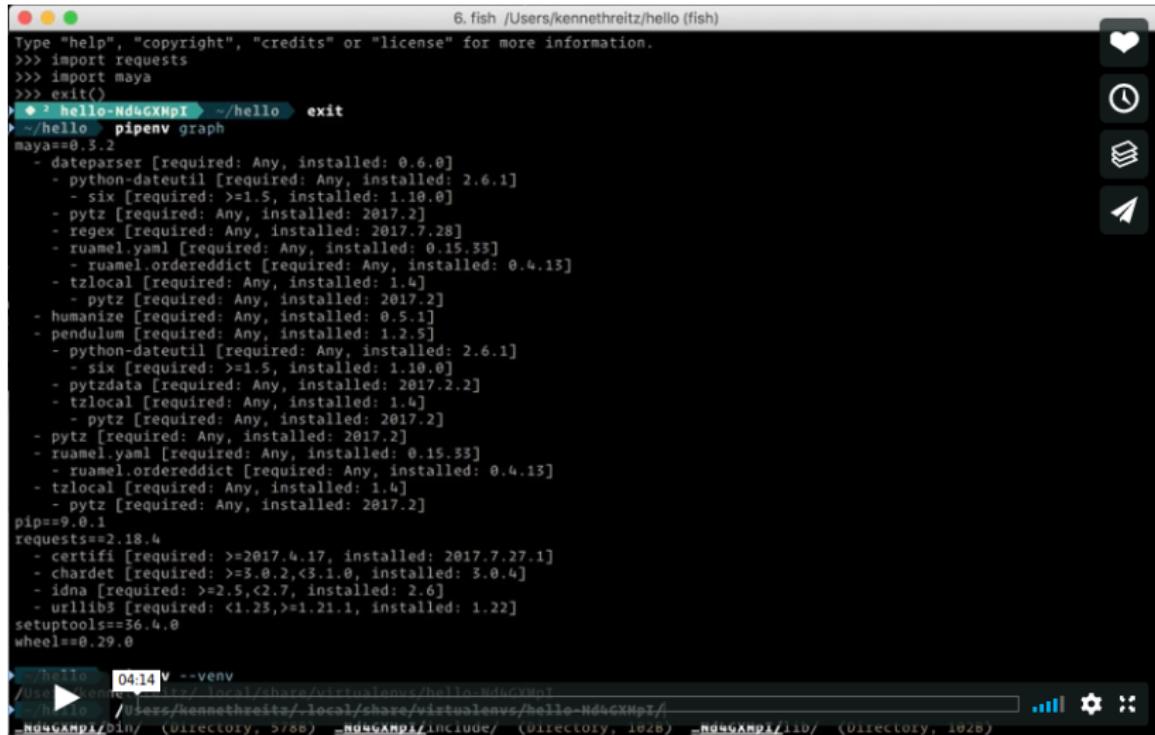
Configurar paquetes (se indica en la shell)

```
daniel@ubuntu:~/working$ source activate IA  
(IA) daniel@ubuntu:~/working$ pip install ...  
(IA) daniel@ubuntu:~/working$ source deactivate  
daniel@ubuntu:~/working$
```

Cargar programa usando el entorno (se indica en la shell)

```
daniel@ubuntu:~/working$ source activate IA  
(IA) daniel@ubuntu:~/working$ python ...  
(IA) daniel@ubuntu:~/working$ source deactivate  
daniel@ubuntu:~/working$
```

Herramienta de entorno: pipenv



6. fish ./Users/kennethreitz/hello (fish)
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> import maya
>>> exit()
* hello-Nd4GXmpI ~/hello exit
~/hello > pipenv graph
maya==0.3.2
- dateparser [required: Any, installed: 0.6.0]
- python-dateutil [required: Any, installed: 2.6.1]
- six [required: >=1.5, installed: 1.10.0]
- pytz [required: Any, installed: 2017.2]
- regex [required: Any, installed: 2017.7.28]
- ruamel.yaml [required: Any, installed: 0.15.33]
- ruamel.ordereddict [required: Any, installed: 0.4.13]
- tzlocal [required: Any, installed: 1.4]
- pytz [required: Any, installed: 2017.2]
- humanize [required: Any, installed: 0.5.1]
- pendulum [required: Any, installed: 1.2.5]
- python-dateutil [required: Any, installed: 2.6.1]
- six [required: >=1.5, installed: 1.10.0]
- pytzdata [required: Any, installed: 2017.2.2]
- tzlocal [required: Any, installed: 1.4]
- pytz [required: Any, installed: 2017.2]
- pytz [required: Any, installed: 2017.2]
- ruamel.yaml [required: Any, installed: 0.15.33]
- ruamel.ordereddict [required: Any, installed: 0.4.13]
- tzlocal [required: Any, installed: 1.4]
- pytz [required: Any, installed: 2017.2]
pip==9.0.1
requests==2.18.4
- certifi [required: >=2017.4.17, installed: 2017.7.27.1]
- chardet [required: >=3.0.2,<3.1.0, installed: 3.0.4]
- idna [required: >=2.5,<2.7, installed: 2.6]
- urllib3 [required: <1.25,>=1.21.1, installed: 1.22]
setuptools==36.4.0
wheel==0.29.0
~/hello 04:14 V --venv
/use/kennethreitz/.local/share/virtualenvs/hello-Nd4GXmpI/
hello /Users/kennethreitz/.local/share/virtualenvs/hello-Nd4GXmpI/
_Nd4GXmpI/bin/ (Directory, >60) _Nd4GXmpI/include/ (Directory, 1028) _Nd4GXmpI/lib/ (Directory, 1028)

Índice

6 Tipos cadena

7 Bucles e iteradores

8 Funciones y clases

9 Clases

10 Librerías

11 Uso de ficheros

Uso de ficheros

Un programa no está solo en el sistema

- Tiene que comunicarse.
- La forma primordial es usando ficheros.

Dos formas de interactuar

- Leer y escribir ficheros.
- Conocer la estructura de ficheros/directorios del ordenador.

Uso de ficheros con Python

Sentencia with

- Uso de **with** para abrir ficheros.
- Permite agrupar el código y no preocuparse de cerrar fichero.

Mayor sencillez

- Alternativa librería os.path ⇒ libpath.
- Uso de print para escribir en ficheros.

Sentencia with

Sentencia with

- Uso de **with** para abrir ficheros.
- Permite agrupar el código y no preocuparse de cerrar fichero.

Código

```
# No es necesario cerrarlo, file es local para el bloque
with open("salida.txt", "w") as file:
    print("Hola a todos", file=file)
    print("Hasta luego", file=file)
```

¿Qué hace?

- ① Crea la variable *file* asociada al nuevo fichero "salida.txt".
- ② Escribe un mensaje usando la variable *file*.
- ③ Al terminar el bloque el fichero se cierra.

Cierre automático del fichero

Ejemplo incorrecto

```
# No es necesario cerrarlo, file es local para el bloque
with open("salida.txt", "w") as file:
    print("Hola a todos", file=file)

    print("Otro mensaje", file=file)
```

Da error

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/tmp/babel-1017136Z/python-10171jGE", line 5, in <module>
    print("Otro mensaje", file=file)
ValueError: I/O operation on closed file.
```

Modos de open

Puede contener como parámetro una cadena "*modo*" o "*modotipo*", en donde:

Modos de fichero

r (por defecto) Abrirlo para leer, si no existe lanza excepción.

w Abrirlo para escribir, si existe lo borra antes.

a Abrirlo para escribir, si existe añade al final.

Tipo de fichero

t (Por defecto) Fichero de texto.

b Fichero binario.

Ejemplo

```
open(fname, "r") # Abre para leer
```

```
open(fname, "wb") # Escribe fichero binario
```

```
open(fname, "a") # Concatena uno de texto
```

Escribiendo fichero binario

Leyendo: `read(longitud)`

```
with open("completo.pdf", "rb") as file:  
    head = file.read(22)  
    print(head)  
    second = file.read(5)  
    print(second)  
  
b'%PDF-1.5\n%\xd0\xd4\xc5\xd8\n17 0 ob'  
b'j\n<<\n'
```

Escribiendo: `write(buffer)`

```
with open("completo.pdf", "rb") as finput:  
    with open("copia.pdf", "wb") as foutput:  
  
        buffer = finput.read(1024)
```

Fichero de texto: leer

readline: Lee una línea

```
with open("salida.txt") as file:  
    line = file.readline()  
    print("Linea 1:", line)  
    print("Linea 1:", line.rstrip())  
    line = file.readline()  
    print("Linea 2:", line)
```

Línea 1: Hola a todos

Línea 1: Hola a todos

Línea 2: Hasta luego

readlines: Lee todo el fichero de golpe

```
with open("salida.txt") as file:  
    lines = file.readlines()  
    print("Lineas:", lines)
```

Fichero de texto: iteración

Iterar un fichero da las líneas

```
with open("salida.txt") as file:  
    for line in file:  
        print(line)
```

Hola a todos

Hasta luego

Cuidado con los retornos de carro

```
with open("salida.txt") as file:  
    for line in file:  
        print(line.rstrip())
```

Hola a todos

Hasta luego

Fichero de texto: escribir

Usar write

```
with open("salida.txt", "a") as fout:  
    fout.write("Nueva linea\n")
```

Uso de print

- print puede escribir.

```
with open("salida.txt", "a") as fout:  
    print("Otra linea", file=fout)
```

Trabajando con ficheros

No todo es with

- No todo es escribir y leer ficheros

Es necesario trabajar con ficheros

- Listas ficheros.
- Gestionar directorios.
- Gestionar ficheros: borrar, renombrar, ...

Trabajando con ficheros

os.system

Permite ejecutar una orden del SO.

Ejemplo con system

```
import os  
os.system("ls *.pdf")
```

```
completo.pdf  copia.pdf  usage.pdf
```

Cuidado

- os.system no es *portable* entre SOs.
- Hay otras alternativas *portables*.

Listar ficheros

Listar

`glob(pattern)` Permite buscar ficheros usando *.

`os.listdir(dir)` Lista un directorio.

Ejemplo

```
import os, glob  
pdfs = glob.glob("*.pdf")  
pdfs2 = [dir for dir in os.listdir('.') if ".pdf" in dir]  
print(pdfs)  
print(pdfs2)  
  
['copia.pdf', 'completo.pdf', 'usage.pdf']  
['copia.pdf', 'completo.pdf', 'usage.pdf']
```

Trabajar con directorios

Editar directorios

`os.mkdir(dir)` Crear directorio.

`os.chdir(dir)` Cambiar de directorio.

`os.curdir` Directorio actual.

`os.rmdir(dir)` Borrar directorio.

Editar ficheros

`os.remove(fname)` Borra fichero.

`os.rename(fold, fnew)` Cambiar el nombre de directorio.

`shutil.copy(origen, destino)` Copiar fichero o directorio.

`shutil.rmtree(dir)` Borra directorio (cuidado).

Librería pathlib

Nuevo interfaz

- Clase Path con muchos métodos.
- Totalmente independiente del SO.

Ejemplo

```
import pathlib
dir = pathlib.Path('.')
files = [file.name for file in dir.glob("*.pdf")]
print(files)

['copia.pdf', 'completo.pdf', 'usage.pdf']
```

Acceso a ficheros desde libpath

Ejemplo

```
from pathlib import Path
dir = Path('.')
file = dir / "usage.pdf"

if file.exists():
    if file.is_dir():
        print(file.name, "es un directorio")
    else:
        print(file.name, "es un fichero")
        file.

usage.pdf es un fichero
```

Trabajando con libpath

Cambiando jpg -> jpeg

```
import pathlib
dir = pathlib.Path('.')
subdirs = [x for x in dir.iterdir() if x.is_dir()]
print(subdirs)

dir = Path('..') / 'borra'
print(dir.resolve())

if dir.exists():
    files = list(dir.glob('*jpeg'))

        for file in files:
new_fname = str(file).replace('.jpeg', '.jpg')
file.rename(new fname)
```

License

Se puede obtener este curso (y otro material) de

https://github.com/dmolina/curso_python

Material disponible bajo licencia

Creative Commons Attribution-ShareAlike 4.0 International
License

Licencia