

# Taller de Python

## Científico



## Conceptos y diseño



Departamento de

**Ciencias de la Atmósfera y los Océanos**

Facultad de Ciencias Exactas y Naturales - Universidad de Buenos Aires

# ¿Por qué Python?

- \* Fácil de aprender.

# ¿Por qué Python?

- \* Fácil de aprender.
- \* Gran cantidad de bibliotecas.



» Package Index

PACKAGE INDEX >>

[Browse packages](#)  
[Package submission](#)  
[List trove classifiers](#)  
[RSS \(latest 40 updates\)](#)  
[RSS \(newest 40 packages\)](#)  
[PyPI Tutorial](#)

## PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **85057** packages here.

To contact the PyPI admins, please use the [Support](#) or [Bug reports](#) links.

# ¿Por qué Python?

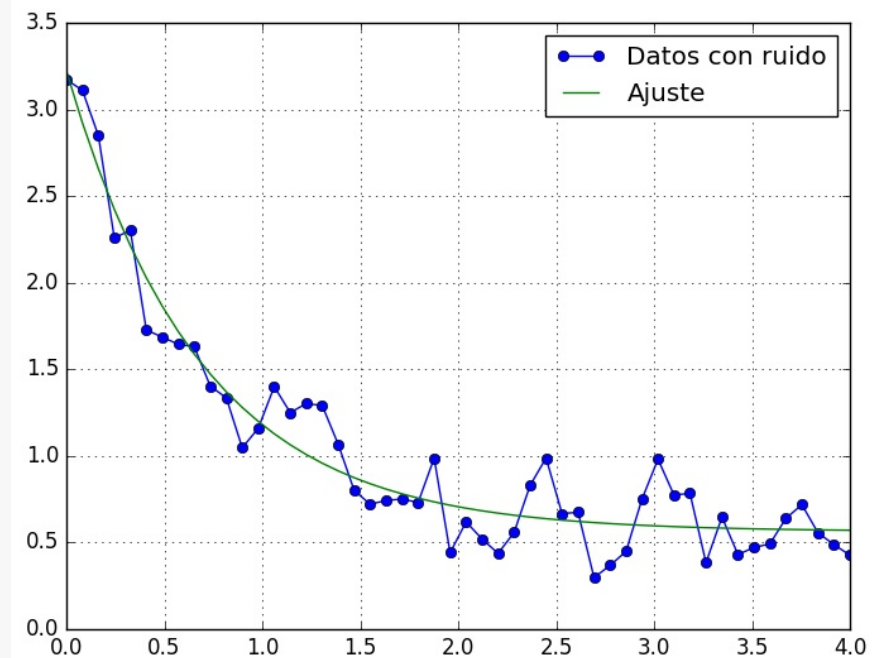
- \* Fácil de aprender.
- \* Gran cantidad de bibliotecas.
- \* Desarrollo de software bastante rápido.  
(Python viene con pilas).

```
import numpy as np
from scipy.optimize import curve_fit

# función a ajustar
def func(x, a, b, c):
    return a * np.exp(-b * x) + c

# datos con ruido
xdata = np.linspace(0, 4, 50)
y = func(xdata, 2.5, 1.3, 0.5)
ydata = y + 0.2 * np.random.normal(size=len(xdata))

#ajuste y plot
popt, pcov = curve_fit(func, xdata, ydata)
plt.plot(xdata, ydata, '-o')
plt.plot(xdata, func(xdata, *popt))
plt.legend(('Datos con ruido', 'Ajuste'))
plt.grid(True)
plt.show()
```



# ¿Por qué Python?

- \* Fácil de aprender.
- \* Gran cantidad de bibliotecas.
- \* Desarrollo de software bastante rápido.
- \* Una comunidad gigante a la cual consultar.
- \* Soporte científico excelente.



## Tagged Questions

info newest 28 featured frequent votes active unanswered

Python is a dynamic and strongly typed programming language designed to emphasize usability. Two similar but incompatible versions of Python are in widespread use (2 and 3). If you have a version-specific Python question consider using the python-2.7 or python-3.x tags in addition to the python tag.

[learn more...](#) [top users](#) [synonyms \(3\)](#) [python jobs](#)

0 votes

1 answer

20 views

### Combine result of two functions in python

The objective I'm trying to reach is to take a string, extract time as hours and minutes and return them as degrees on an analogue clock dial, the only way i could think of to do this is to create 2 ...

python

asked 6 mins ago



Anayac

13 4

0 votes

0

### Is there a way to block django views from serving multiple requests concurrently?

I have a django app, where I am using one of the views to fetch data from local filesystem and parse it and add it to my database. Now the thing is, I want to restrict this view from serving multiple ...

Questions Jobs Documentation Beta Tags Users

606,567 questions tagged

python about »

## Related Tags

django × 61262  
python-2.7 × 34748  
python-3.x × 26477  
numpy × 24724  
pandas × 23825  
list × 20228  
matplotlib × 15818  
regex × 14300  
dictionary × 14075



Ingresa o Registrate



Inicio



Nosotros



Lista



Wiki



Noticias



## Comunidad Python Argentina

Nuestro objetivo es nuclear a los usuarios de Python. Pretendemos llegar a personas y empresas, promover el uso de Python e intercambiar información.



### Aprendiendo Python

Recopilación de recursos que los miembros de PyAr consideramos útil para quienes se inician en este lenguaje.



### Lista de correo

Instrucciones para suscribirse a nuestra lista, nuestro principal canal de comunicación.



### Bolsa de Trabajo

Ofrece o busca empleos relacionados con Python.



### Quiénes usan Python

Información sobre organizaciones



### Nuestra wiki

El espacio de edición colaborativa de



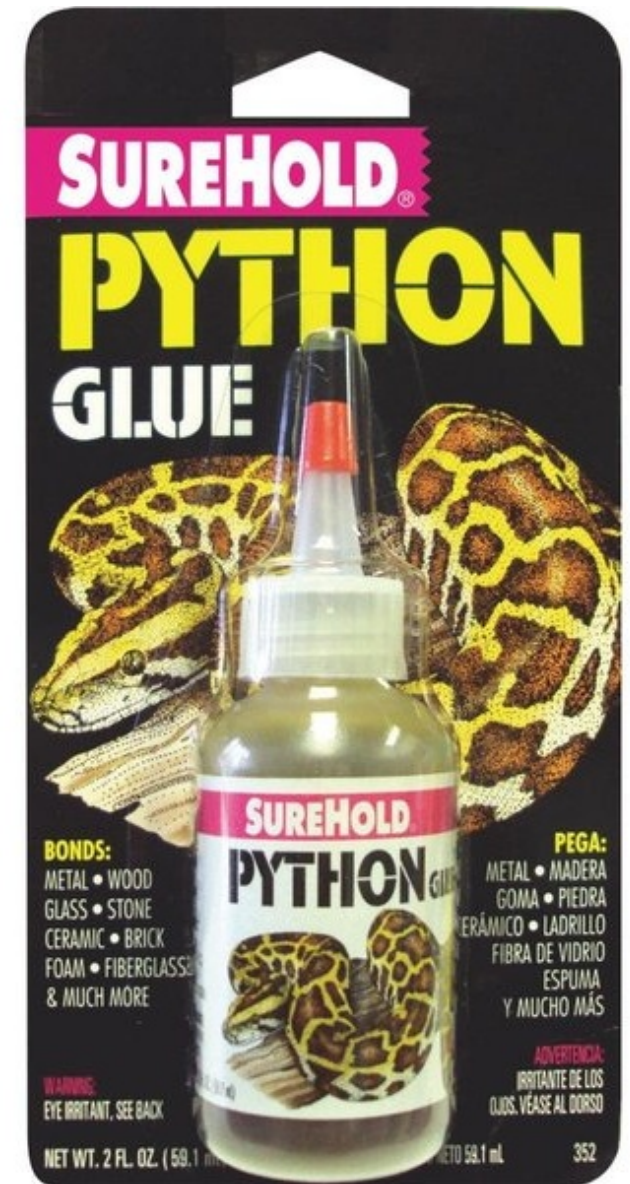
### Proyectos de la comunidad





# ¿Por qué Python?

- \* Fácil de aprender.
- \* Gran cantidad de bibliotecas.
- \* Desarrollo de software bastante rápido.
- \* Una comunidad gigante a la cual consultar.
- \* Soporte científico excelente.
- \* Es un lenguaje de pegamento.



# ¿Por qué Python?

- \* Fácil de aprender.
- \* Gran cantidad de bibliotecas.
- \* Desarrollo de software bastante rápido.
- \* Una comunidad gigante a la cual consultar.
- \* Soporte científico excelente.
- \* Es un lenguaje de pegamento.
- \* Posee una licencia de código abierto.

# Un poco de historia

- \* Comenzó a desarrollarse en 1989 por Guido van Rossum (Benevolent Dictator For Life!)
- \* Python 2.0: Octubre 2000 (actual: 2.7.12)
- \* Python 3.0: Diciembre 2008 (actual 3.5.2)



# ¿Que versión uso?

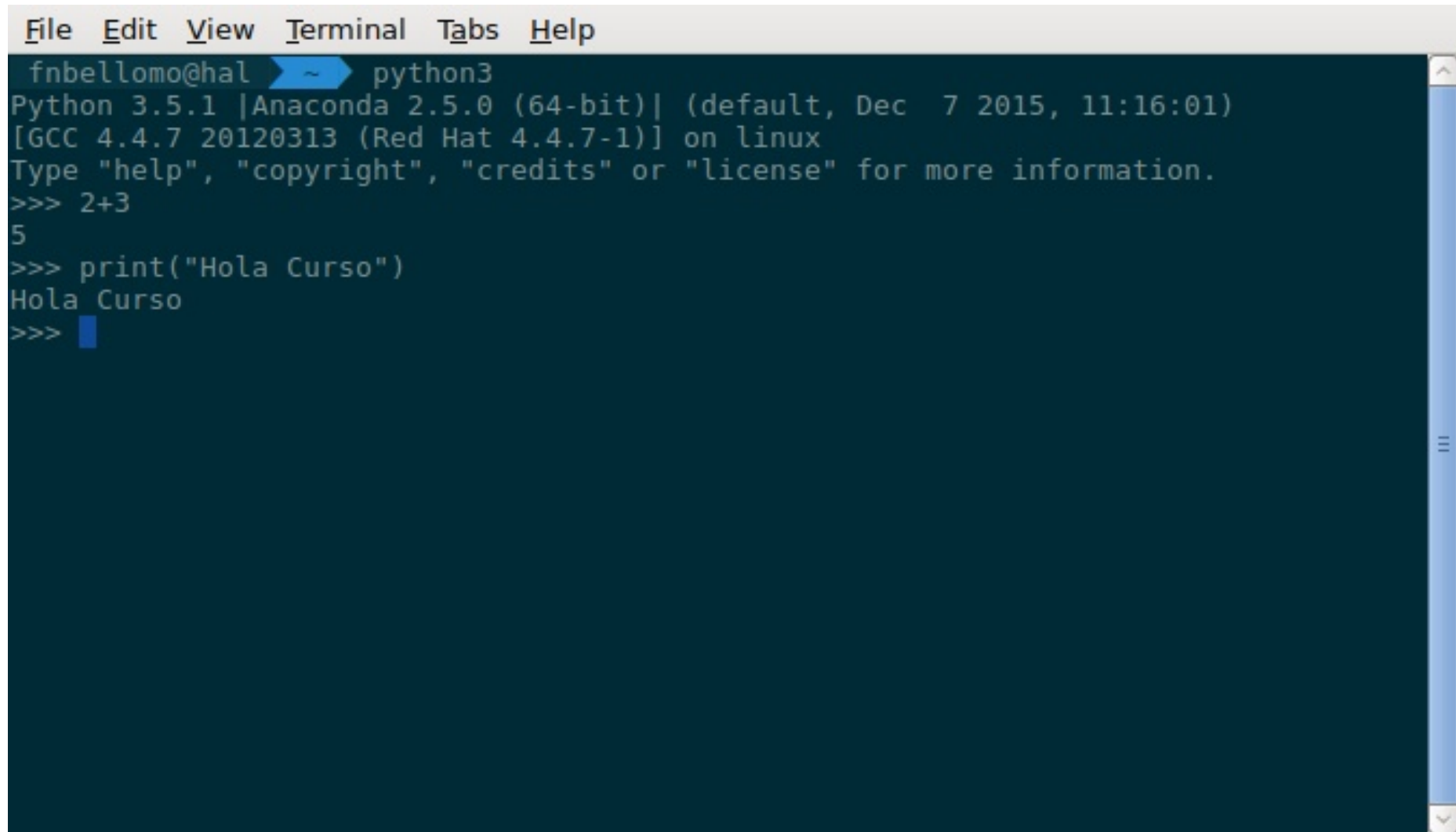
- \* Python 2.x es historia.
- \* Todas las grandes bibliotecas migraron a Python 3.x
- \* No va a haber más 2.x releases
- \* Existen herramientas de conversión: 2to3 3to2
- \* El futuro es de Python 3.x

## Zen de Python (una parte), por Tim Peters

- \* Bello es mejor que feo.
- \* Explícito es mejor que implícito.
- \* Simple es mejor que complejo.
- \* Complejo es mejor que complicado.
- \* Debería haber una - y preferiblemente sólo una - manera obvia de hacerlo.
- \* La legibilidad cuenta.
- \* Si la implementación es difícil de explicar, es una mala idea.

# Python es interactivo

Simplemente abrimos una consola y probamos

A screenshot of a terminal window with a dark blue background and light blue text. The window has a menu bar at the top with 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The prompt 'fnbellomo@hal' is followed by a blue arrow pointing to the right, then '~' and 'python3'. The output shows the Python version 'Python 3.5.1 |Anaconda 2.5.0 (64-bit)| (default, Dec 7 2015, 11:16:01)', the compiler '[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux', and a message to type 'help', 'copyright', 'credits' or 'license' for more information. The user enters '>>> 2+3' and the output is '5'. Then the user enters '>>> print("Hola Curso")' and the output is 'Hola Curso'. The prompt '>>>' is followed by a blue cursor.

```
File Edit View Terminal Tabs Help
fnbellomo@hal ~ python3
Python 3.5.1 |Anaconda 2.5.0 (64-bit)| (default, Dec 7 2015, 11:16:01)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> print("Hola Curso")
Hola Curso
>>> 
```

# Variables

Son definidas dinámicamente, no se tiene que especificar cuál es su tipo de antemano

```
a = 'spam'
b = a*2
a = 10**2
c = a*a
```

```
def foo(a, b):
    return a+b

a, b = "hola", "mundo"
foo(a, b)

a, b = 2, 8
foo(a, b)
```

“Duck Typing”

Las llamadas a funciones aceptan cualquier tipo de argumento.

Error de ejecución si no se ajusta.

# Tipos de datos

De la librería estándar.

Tipo	Clase	Notas	Ejemplo
<code>str</code>	Cadena	Inmutable	<code>'Cadena'</code>
<code>unicode</code>	Cadena	Versión <a href="#">Unicode</a> de <code>str</code>	<code>u'Cadena'</code>
<code>list</code>	Secuencia	Mutable, puede contener objetos de diversos tipos	<code>[4.0, 'Cadena', True]</code>
<code>tuple</code>	Secuencia	Inmutable, puede contener objetos de diversos tipos	<code>(4.0, 'Cadena', True)</code>
<code>set</code>	Conjunto	Mutable, sin orden, no contiene duplicados	<code>set([4.0, 'Cadena', True])</code>
<code>frozenset</code>	Conjunto	Inmutable, sin orden, no contiene duplicados	<code>frozenset([4.0, 'Cadena', True])</code>
<code>dict</code>	Mapping	Grupo de pares clave:valor	<code>{'key1': 1.0, 'key2': False}</code>
<code>int</code>	Número entero	Precisión fija, convertido en <i>long</i> en caso de overflow.	<code>42</code>
<code>long</code>	Número entero	Precisión arbitraria	<code>42L</code> ó <code>456966786151987643L</code>
<code>float</code>	Número decimal	Coma flotante de doble precisión	<code>3.1415927</code>
<code>complex</code>	Número complejo	Parte real y parte imaginaria <i>j</i> .	<code>(4.5 + 3j)</code>
<code>bool</code>	Booleano	Valor booleano verdadero o falso	<code>True</code> o <code>False</code>

# Expresiones y operaciones

Similares a otros lenguajes

Adición: +

Sustracción: -

Multiplicación: \*

Potencia: \*\*

División entera: //

División: /

Módulo: %

Operaciones lógicas

Igual: ==

Distinto: !=

Mayor: >

Menor: <

Mayor igual: >=

Menor igual: <=



# División en Python 2 y 3

Cuidado con la división en Python 2!

```
#Python 2  
>>> 5/3  
1
```

```
#Python 3  
>>> 5/3  
1.6666666666666667
```

Se puede "solucionar" con la siguiente línea al principio del script de Python 2

```
from __future__ import division
```

# Strings

Delimitadores de strings: se puede usar " o ' indistintamente.

```
a = "spam, spam, spam"  
b = 'El me dijo "hola" a mi'
```

```
"""Se puede construir strings  
de muchas líneas  
usando triple comilla"""
```

# Strings formatting

Dos formas

```
>>> "Pi con %d digitos %.3f" % (3, 3.141592653589793)
'Pi con 3 digitos 3.142'
```

```
>>> "Pi con {1} digitos {0:.3f}".format(3.141592653589793, 3)
'Pi con 3 digitos 3.142'
```

La segunda forma es más flexible y potente

```
>>> text = "I ate {num} {food} today. Yes, really {num}."
>>> answer = text.format(num=12, food="apples")
'I ate 12 apples today. Yes, really 12'
>>> answer = text.format(num=15, food="pie")
'I ate 15 pie today. Yes, really 12'
```

# Conjuntos

Colección desordenada en la cual cada elemento aparece una sola vez (conjunto matemático).

```
>>> {1+2, 3, "a"}  
{'a', 3}  
>>> {2, 1, 3}  
{1, 2, 3}  
>>> {'a', 'b', 'c', 'a', 'a'}  
{'a', 'b', 'c'}  
>>> a = set( ) #conjunto vacío
```

# Conjuntos (operaciones)

Algunas operaciones básicas de conjuntos:

```
>>> S = {1,2,3}
>>> len(S) # cardinal
3
>>> sum(S)
6
```

```
>>> 2 in S
True
>>> 4 in S
False
>>> 4 not in S
True
```

```
>>> {1,2,3} | {2,3,4} # unión
{1, 2, 3, 4}
>>> {1,2,3} & {2,3,4} # intersección
{2, 3}
```

# Listas

Secuencia ordenada de elementos. Los elementos se pueden repetir y pueden ser int, string, set, lists or dictionaries.

```
>>> [1,1+1,3,2,3]
[1, 2, 3, 2, 3]
>>> [[1,1+1,4-1],{2*2,5,6}, 'yo']
[[1, 2, 3], {4, 5, 6}, 'yo']
```

Las operaciones son las mismas que para los conjuntos.



# Listas (index / slice access)

El slice genera una nueva lista a partir de otra. La sintaxis es:  
[comienzo:final:paso]

```
>>> L = [0,10,20,30,40,50,60,70,80,90]
>>> L[:5]
[0, 10, 20, 30, 40]
>>> L[5:]
[50, 60, 70, 80, 90]
>>> L[2::2]
[20, 40, 60, 80]
>>> L[-3]
70
>>> L[::-1]
[90, 80, 70, 60, 50, 40, 30, 20, 10, 0]
```

# Diccionarios

Conjunto de pares key-value desordenados. Las key pueden ser int, floats o string. La idea es similar a una guía telefónica.

```
>>> {2+1:'thr'+ 'ee', 2*2:'fo'+ 'ur'}  
{3: 'three', 4: 'four'}  
>>> {0:'zero', 0:'nothing'}  
{0: 'nothing'}  
>>> mydict = {'Neo':'Keanu', 'Morpheus':'Laurence'}  
>>> mydict['Neo']  
'Keanu'
```

Las operaciones son las mismas que para los conjuntos.

# Sintaxis

Los espacios en blanco son importantes. Definen los alcances (scope) de las funciones, estructura de control, clases, etc.

```
if 2<1:  
    foo()  
    bar()  
baz()
```

```
if 2<1:  
    foo()  
bar()  
baz()
```

```
if 2<1:  
    foo()  
        bar()  
baz()
```

# Sintaxis

Los espacios en blanco son importantes. Definen los alcances (scope) de las funciones, estructura de control, clases, etc.

```
if 2<1:  
    foo()  
    bar()  
baz()
```

Se llama a la  
función baz()

```
if 2<1:  
    foo()  
bar()  
baz()
```

Se llama a las  
funciones baz()  
y baz()

```
if 2<1:  
    foo()  
        bar()  
baz()
```

Error de sintaxis

# Estructuras de control

```
for i in list:  
    baz(i)
```

```
if a > b:  
    foo( )  
elif a != c:  
    bar( )  
else:  
    baz( )
```

```
while a > b:  
    foo( )
```

```
pass
```

```
break  
continue
```

Similar a otros lenguajes, salvo por el for que recorre listas.

# Estructuras de control

Un ejemplo simple

```
cnt = 0
suma = 0

while True:
    random = np.random.random()
    print(random)

    if 0.2 < random < 0.8:
        cnt += 1

    if cnt == 5:
        break
```



# Listas por compresión

Es una forma compacta y Pythonica de generar listas (o conjuntos) a partir de estructuras de control (for e if).

```
>>> [ 2*x for x in [2,1,3,4,5] ]  
[4, 2, 6, 8, 10]  
>>> [ x*y for x in [1,2,3] for y in [10,20,30] ]  
[10, 20, 30, 20, 40, 60, 30, 60, 90]  
>>> [ 2*x for x in [2,1,3,4,5,6] if x%2 == 0 ]  
[4, 8, 12]  
>>> [[x, y] for x in [1,2,3] for y in [10,20,30] if (x+y) % 3 != 0 ]  
[[1, 10], [1, 30], [2, 20], [2, 30], [3, 10], [3, 20]]
```

# Funciones incorporadas

`range()`: secuencia inmutable de números

```
>>> [x**2 for x in range(10)]  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
>>> [x for x in range(0, 100, 10)]  
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

`enumerate()`: enumera un objeto iterable

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']  
>>> list(enumerate(seasons))  
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]  
>>> list(enumerate(seasons, start=1))  
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

# Abriendo archivos

```
f = open('data.txt', 'r')  
...  
f.close() #No olvidar cerrar el archivo!
```

- 'r' open for reading (default)
- 'w' open for writing, truncating the file first
- 'x' open for exclusive creation, failing if the file already exists
- 'a' open for writing, appending to the end of the file if it exists
- 'b' binary mode
- 't' text mode (default)
- '+' open a disk file for updating (reading and writing)

# Recorriendo archivos

La expresión "with" cierra el archivo cuando termina (la forma pythonica de hacerlo).

```
with open('data.txt', 'r') as f:  
    # Se lee el archivo de a una línea  
    for i, line in enumerate(f):  
        print('Line #{0}: {1}'.format(i, line))
```

# Excepciones (Hay que usarlas)

Un ejemplo simple

```
try:  
    a = read_my_data()  
except:  
    print("Corrupted data")
```

Es siempre preferible a tener que definir funciones adicionales

```
if consistent_data():  
    a = read_my_data()  
else:  
    print("Corrupted data")
```

# Funciones

```
>>> def operation(a, b, c):  
    return (a + b) * c
```

# Argumentos posicionales

```
>>> operation(2, 4, 3)  
18
```

# Argumento por palabra

```
>>> operation(c=3, a=2, b=4)  
18
```

```
>>> def operation(a, b, c=3):  
    return (a + b) * c
```

# Argumentos opcionales

```
>>> operation(2, 4)  
18
```

```
>>> operation(2, 4, 5)  
30
```



# Instalando bibliotecas

Existen 3 formas básicas de instalar bibliotecas Python en nuestro sistema:

- \* Sistema de paquetes (apt, yum, etc)  
Versiones de bibliotecas viejas pero, resuelve dependencias  
IPython version: 2.4 (debian testing)
- \* **Pip**  
Versiones nuevas pero no resuelve dependencias  
(unicamente python)  
IPython version: 5.0
- \* **Conda**  
Versiones nuevas e implementa un sistema de paquetes capaz de resolver algunas dependencias.  
IPython version: 5.0

# Links importantes

Documentación oficial de python:

<https://docs.python.org/3/>

O simplemente:

[google.com](https://www.google.com)