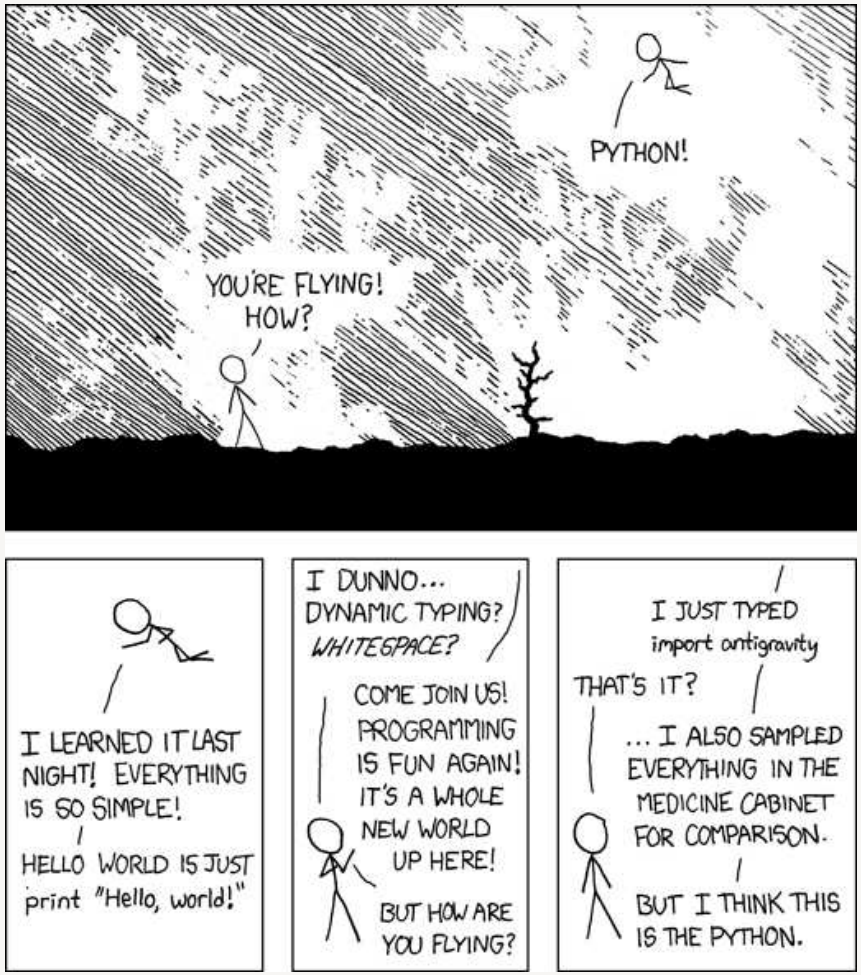
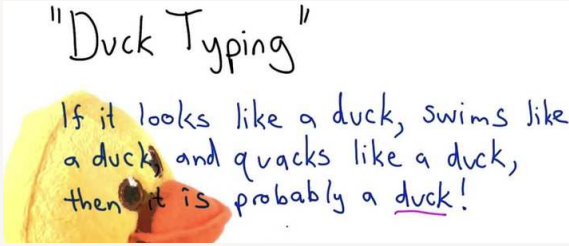
Python



https://www.python.org/

- Librerías ++
- Comunidad ++
- Curva de aprendizaje ++
- Free + Open Source
- Interpretado, multiparadigma, fuertemente tipado, tipado dinámico
- Frameworks, environments, shells



- Zen de Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
...
Readability counts.
Special cases aren't special enough to break the rules.
...

- Style Guide

https://www.python.org/dev/peps/pep-0008/

Tipos de datos, objetos, duck typing, ID

```
[6]  x = 3

     print (x)
     print (type (x))
     print (id(x))
```

```
3
<class 'int'>
94341805677120
```

```
[7]
     x = 1.2
     print (x)
     print (type (x))
     print (id(x))
```

```
1.2
<class 'float'>
140438647160752
```

id(*object*)

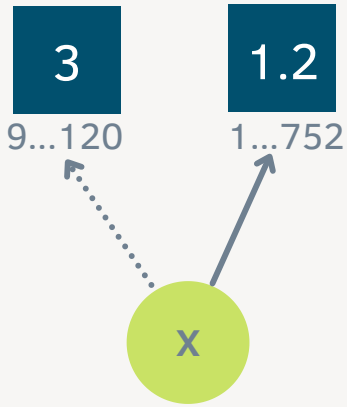
Return the “identity” of an object.

- Tiene sentido que haya cambiado el ID, son dos objetos diferentes de diferente tipo!

- No es necesario declarar las variables (duck typing)
- Si controla tipo de datos (fuertemente tipado)

```
y = 'Hola'
# Se puede usar ' o " para strings
x - y
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-bbd1dd82f272> in <module>()
      1 y = 'Hola'
      2 # Se puede usar ' o " para strings
----> 3 x - y
      4

TypeError: unsupported operand type(s) for -: 'float' and 'str'
```




```
print (id(1.2))
```

```
140438577165520
```

Objetos inmutables

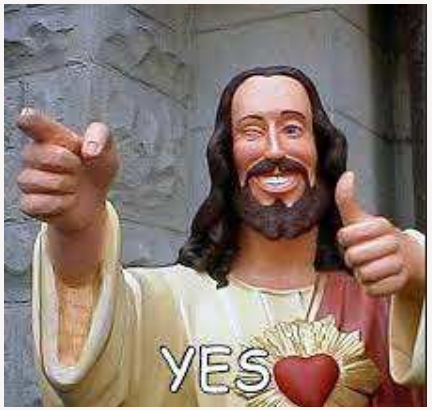
```
a = 5
b = a
print (a,b)
print (id(a),id(b))
```

```
5 5
94341805677184 94341805677184
```

```
b = b + 2
print (a,b)
print (id(a), id(b))
```

```
5 7
94341805677184 94341805677248
```

- Objeto con un valor fijo (números, strings y tuplas). El objeto no puede ser alterado. Se tiene que crear un nuevo objeto si es necesario almacenar un nuevo valor.



```
lista = ['a', 1, 1]
print (lista)
print (type(lista))
print (id(lista))
```

```
['a', 1, 1]
<class 'list'>
140438507970960
```

```
[18] lista.append(9)
     print (lista)
     print (lista [0])
     print (lista[-1])
     print (lista*2)
     print (lista[0:1])
```

```
['a', 1, 1, 9]
a
9
['a', 1, 1, 9, 'a', 1, 1, 9]
['a']
```

Listas

- Acceso a elementos
- Slicing
- Indices negativos
- List comprehension (forma elegante de definir listas).
  Tbien se puede tener comprensión en caso de set y dict

Objetos mutables

- Objeto que puede cambiar su valor pero conservan su ID

```
l1 = [1,2,3]
print (l1)
l2 = l1
print (l2)
print (id(l1), id(l2))
```

```
[1, 2, 3]
[1, 2, 3]
140438507543040 140438507543040
```

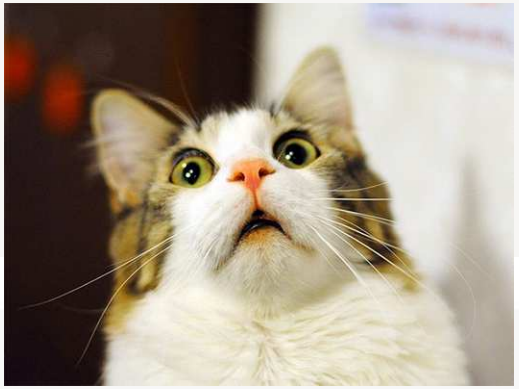
- l1 y l2 son el mismo objeto

```
l2.insert(1,'hola')
l1.insert(1, 'chau')
print (l1,l2)
```

```
[1, 'chau', 'hola', 2, 3] [1, 'chau', 'hola', 2, 3]
```

```
print(id(l1),id(l2))
```

```
140438507611552 140438507611552
```



```
[26] l3 = [l1,l2]
     print(l3)
     print(id(l3))
```

```
[[1, 'chau', 'hola', 2, 3], [1, 'chau', 'hola', 2, 3]]
140438507542160
```

```
l1.insert(0,9999)
print(l2)
```

```
[9999, 1, 'chau', 'hola', 2, 3]
```

- https://www.pythoncheatsheet.org/

Iterables

Objeto que:

- Puede incluir cero, uno o muchos elementos,
- Tiene la habilidad de retornar un elemento a la vez (se puede iterar entre sus elementos)

range () es una función iterable

```
for indice in range(3):
  print (indice)
```

```
0
1
2
```

```
texto = 'iterable'
for caracter in texto:
  print(caracter)
```

```
i
t
e
r
a
b
l
e
```

```
[38] colores = ['rojo', 'verde', 'azul']
     iter_colores = iter(colores)
     print (iter_colores)
```

```
<list_iterator object at 0x7fba63629710>
```

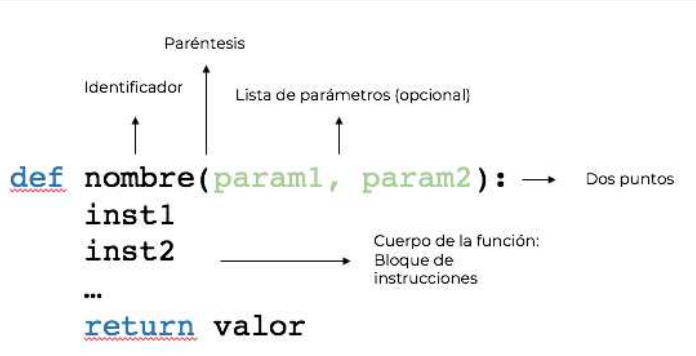
```
[39] color=next(iter_colores)
     print (color)
```

```
rojo
```

```
print(next(iter_colores))
print(next(iter_colores))
print(next(iter_colores))
```

```
verde
azul
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-40-9ed1af144445> in <module>()
      1 print(next(iter_colores))
      2 print(next(iter_colores))
----> 3 print(next(iter_colores))
      4

StopIteration:
```

Funciones




```
def mi_otra_fun (a, b ):
  a=a+2
  b+b*2
  return(a,b)

primer_arg =20
seg_arg = [1,2]
print(mi_otra_fun(a,b))
```

```
(7, 7)
```

```
respuesta = lambda n1,n2: n1 + n2
print (respuesta(2,3))
```

```
5
```

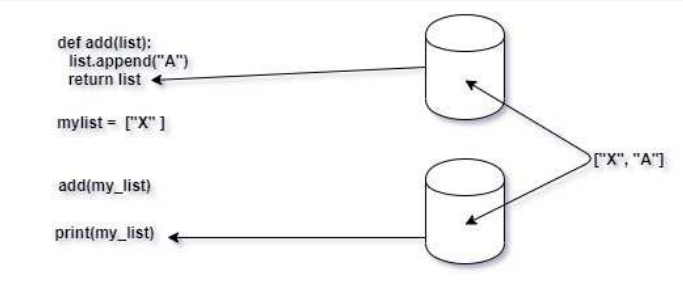
```
def mi_fun (lista, alfa=[99] ):
  print('Dentro de mi_fun')
  print(id(lista))
  lista = lista + alfa
  print(id(lista))
  return(lista)

mi_lista=[1,2,3]
print(id(mi_lista))
print(mi_fun(mi_lista))
print(mi_lista)
```

```
140438507251744
Dentro de mi_fun
140438507251744
140438507001488
[1, 2, 3, 99]
[1, 2, 3]
```

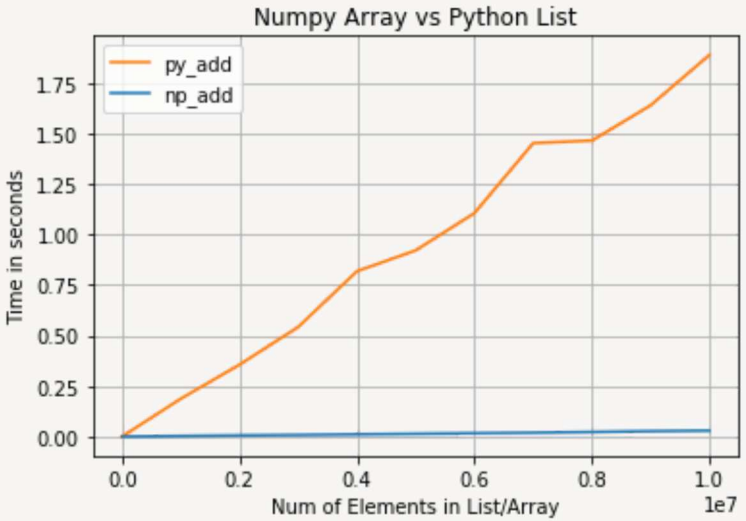
```
def agregar(lista):
  lista.append('B')
  print(id(lista))
  return (lista)

mi_lista = [1,2]
print (id(mi_lista))
agregar(mi_lista)
print(mi_lista)
```

```
140438507313024
140438507313024
[1, 2, 'B']
```

Pasaje por referencia de objeto (no el objeto)



Arreglos



Se compara la suma de 2 listas de hasta 10.000.000 elementos, con la suma de 2 arreglos con la misma cantidad de elementos

https://medium.com/@gough.cory/performance-of-numpy-array-vs-python-list-194c8e283b65

Listas	Array (Numpy)
• Pueden tener elementos de diferente tipo	• Solo pueden tener elementos de igual tipo
• No nec importar un modulo	• Nec importar un modulo
• No hace operaciones aritméticas directamente	• Hace operaciones aritméticas directamente
• Pensadas para poco elementos	• Pensadas para muchos elementos
• + flexibilidad	• < flexibilidad (op por elementos)
• Se puede mostrar sin nec de un loop	• Nec de un loop para el print
• Consume > memoria	• Consume < memoria que las listas

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(10)
y = 2*x
print(y)
plt.plot(x,y)
```

```
[ 0  2  4  6  8 10 12 14 16 18]
[<matplotlib.lines.Line2D at 0x7fba635333d0>]
```