Un micro tutorial de Python

Números

```
>>> 2+2
4
>>> (50 - 5*6) / 4
5.0
>>> 2309874209847209 * 120894739
279251639722309561933451
>>> 3 * 3.75 / 1.5
7.5
>>> (2+3j) * (8-4j)
(28+16j)
```

Cadenas

```
>>> 'Secuencia de caracteres'
'Secuencia de caracteres'
>>> "Hola" + " mundo"
'Hola mundo'
>>> "Eco " * 4
'Eco Eco Eco Eco '
>>> saludo = 'Hola mundo'
>>> saludo[0], saludo[-2]
('H', 'd')
>>> saludo[2:5]
'la '
```

Listas

```
>>> a = [100, 'huevos', 'sal']
>>> a
[100, 'huevos', 'sal']
>>> a[0]
100
>>> a[-2:]
['huevos', 'sal']
>>> a + ['oro', 9]
[100, 'huevos', 'sal', 'oro', 9]
>>> a[0] = "manteca"
>>> a
['manteca', 'huevos', 'sal']
```

Conjuntos

```
>>> f = set("abracadabra")
>>> f
{'b', 'a', 'r', 'c', 'd'}
>>> f & set(['a','e','i','o','u'])
{'a'}
```

Diccionarios

```
>>> dias = {"Ene": 31, "Jul": 30}
>>> dias
{'Jul': 30, 'Ene': 31}
>>> dias["Ene"]
31
>>> dias["Ago"] = 31
>>> dias["Jul"] = 31
>>> dias
{'Ago': 31, 'Jul': 31, 'Ene': 31}
>>> "Mar" in dias
False
>>> dias.keys()
dict_keys(['Ago', 'Jul', 'Ene'])
>>> dias.values()
dict_values([31, 31, 31])
```

if

```
if <expresión>:
    <código>
elif <expresión>:
    <código>
else:
    <código>
```

Una <expresión> es algo que evalúa siempre a Verdadero o Falso
Operadores lógicos: `or`, `and`, `not`
Comparadores: `< > == != in is`
<código> es un bloque de código, de una o más lineas, delimitado por la sangría.

while

```
while <expresión>:
    <código>
```

for

```
>>> bichos = ["pulgas", "piojos"]
>>> for bicho in bichos:
...     print("Mata-" + bicho)
...
Mata-pulgas
Mata-piojos
```

List comprehensions

```
>>> vec = [3, 7, 12, 0, 3, -13]
>>> [x**2 for x in vec]
[9, 49, 144, 0, 9, 169]
>>> [x**2 for x in vec if x <= 7]
[9, 49, 0, 9, 169]
```

Excepciones

```
>>> 5 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in ...
ZeroDivisionError: div... by zero

>>> try:
...     5 / 0
... except ZeroDivisionError:
...     print("oops!")
...
oops!

try:
    <código>
except [Excepcion1, ...]:
    <código>
finally:
    <código>
else:
    <código>
```

Si hay una excepción en el <código> del *try*, se ejecuta el <código> del *except*. Si no hubo ninguna excepción, se ejecuta el <código> del *else*. Y siempre se ejecuta el <código> del *finally*.

```
>>> raise ValueError("Ejemplo!")
Traceback (most recent call last):
  File "<stdin>", line 1, in ...
ValueError: Ejemplo!
```

Funciones

```
>>> def alcuadrado(n):
...     resultado = n ** 2
...     return resultado
...
>>> alcuadrado(3)
9
```

```
>>> def funcion(a, b=0, c=7):
...     return a, b, c
...
>>> funcion(1)
(1, 0, 7)
>>> funcion(1, 3)
(1, 3, 7)
>>> funcion(1, 3, 9)
(1, 3, 9)
>>> funcion(1, c=9)
(1, 0, 9)
>>> funcion(b=2, a=-3)
(-3, 2, 7)
```

Clases

```
>>> import math
>>> class Posicion:
...   def __init__(self, x, y):
...     self.x = x
...     self.y = y
...   def distancia(self):
...     x = self.x**2 + self.y**2
..      return math.sqrt(x)
...
>>> p1 = Posicion(3, 4)
>>> p1.x
3
>>> p1.distancia()
5.0
>>> p2 = Posicion(7, 9)
>>> p2.y
9
>>> p1.y
4
```

Módulos

- Funciones, clases, y/o código suelto, todo en un archivo
- Es un *.py* normal, sólo lo importamos y directamente lo usamos
- Fácil, rápido, ¡y funciona!

Armamos un *pos.py* que contiene la clase definida arriba:

```
>>> import pos
>>> p = pos.Posicion(2, 3)
>>> p.x
2
```

El Zen de Python

por Tim Peters

Bello es mejor que feo.
Explícito es mejor que implícito.
Simple es mejor que complejo.
Complejo es mejor que complicado.
Plano es mejor que anidado.
Disperso es mejor que denso.
La legibilidad cuenta.
Los casos especiales no son tan especiales como para quebrantar las reglas.
Aunque lo práctico gana a la pureza.
Los errores nunca deberían dejarse pasar silenciosamente.
A menos que hayan sido silenciados explícitamente.
Frente a la ambigüedad, rechaza la tentación de adivinar.
Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
Ahora es mejor que nunca.
Aunque nunca es a menudo mejor que *ya mismo*.
Si la implementación es difícil de explicar, es una mala idea.
Si la implementación es fácil de explicar, puede que sea una buena idea.
Los espacios de nombres son una gran idea
- ¡Hagamos más de esas cosas!

python™

PyAr

http://es.wikipedia.org/wiki/Python

¿Por dónde empezar?

http://argentinaenpython.com.ar/quiero-aprender-python/

Comunidades

Pertenecer a una comunidad de Software Libre o de un lenguaje de programación, como Python, te va a permitir no estar solo a la hora de tener una duda o estar bloqueado con algo que no funciona como uno espera.

Python Argentina

PyAr

Sitio Oficial
http://python.org.ar/

Ayuda instantánea (chat)
http://python.org.ar/irc/

Lista de Correo
http://python.org.ar/lista/

Eventos
Meetup, Sprint, PyDay, PyCamp y PyConAr

Python Bolivia
https://twitter.com/PythonBolivia/

Python Paraguay
http://www.meetup.com/Python-Paraguay/

Python Peru
http://python.pe/

Proyecto Argentina en Python

por Manuel Kaufmann

argentina

Sitio Oficial
http://argentinaenpython.com.ar/