

Introducción a Python

Facundo Batista
[@facundobatista](#)



Arte gráfico: Diana Batista

Indice

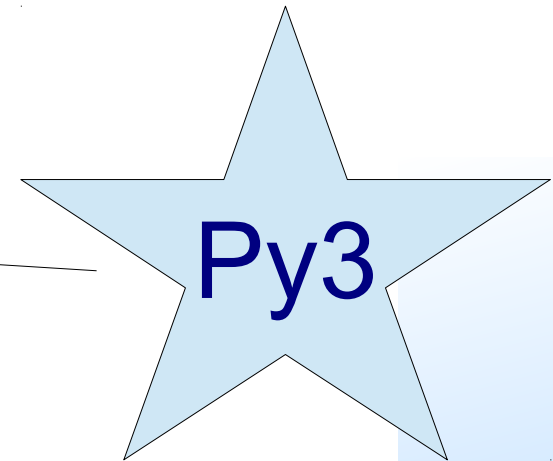
- ¿Qué es Python?
- Corriendo e interpretando
- Tipos de datos
- Controles de flujo
- Encapsulando código
- Tres detalles



¿Qué es Python?

Algunas características

- Gratis Y Libre
 - × Y Open Source, todo por el mismo precio: **cero**
- Maduro (+25 años)
 - × Diseño elegante y **robusto**
 - × Pero **evoluciona**
- **Fácil** de aprender
 - × Se lee como pseudo-código
 - × **Sintaxis** sencilla, lenguaje muy **ortogonal**
- Extremadamente **portable**
 - × Unix, Windows, Mac, BeOS, Win/CE
 - × DOS, OS/2, Amiga, VMS, Cray...



Propiedades del lenguaje

- Compila a bytecode interpretado
 - × La compilación es **implícita y automática**
 - × Tipado **dinámico**, pero **fuerte**
- **Multi-paradigma**
 - × Todo son objetos
 - × Pero puede usarse de manera procedural
- Módulos, clases, funciones, generadores
- Manejo moderno de **errores**
 - × Por **excepciones**
 - × Muy útil **detalle de error**

Más propiedades

- Tipos de datos de **alto nivel**
 - × **Enteros sin límites**, strings, flotantes, complejos
 - × Listas, **diccionarios**, conjuntos
- Intérprete interactivo
 - × Clave en el **bajo conteo de bugs**
 - × **Acelera** sorprendentemente el **tiempo de desarrollo**
 - × Permite **explorar**, **probar** e incluso ver la **doc**
- Viene con las **baterías incluidas**
 - × Extensa biblioteca estándar
 - × Clave en la **productividad** de Python

Las baterías incluídas

- La Biblioteca Estándar ayuda con...
 - × Servicios del sistema, fecha y hora, subprocessos, sockets, internacionalización y localización, base de datos, threads, formatos zip, bzip2, gzip, tar, expresiones regulares, XML (DOM y SAX), Unicode, SGML, HTML, XHTML, XML-RPC (cliente y servidor), email, manejo asincrónico de sockets, clientes HTTP, FTP, SMTP, NNTP, POP3, IMAP4, servidores HTTP, SMTP, herramientas MIME, interfaz con el garbage collector, serializador y deserializador de objetos, debugger, profiler, random, curses, logging, compilador, decompilador, CSV, análisis lexicográfico, interfaz gráfica incorporada, matemática real y compleja, criptografía (MD5 y SHA), introspección, unit testing, doc testing, etc., etc...

Le ponemos más pilas

- Bases de datos
 - × MySQL, PostgreSQL, MS SQL, Informix, DB/2, Sybase
- Interfaces gráficas
 - × Qt, GTK, win32, wxWidgets, Cairo
- Frameworks Web
 - × Django, Turbogears, Zope, Plone, webpy
- Y un montón más de temas...
 - × Pillow: para trabajar con imágenes
 - × PyGame: juegos, presentaciones, gráficos
 - × SymPy: matemática simbólica
 - × Numpy: calculos de alta performance

Python Argentina



- ¿Quienes somos?
 - × Grupo de **entusiastas** de Python
 - × Referencia para la aplicación y **difusión** del lenguaje
- ¿Cómo **participar**?
 - × Suscribiéndose a la **Lista de Correo** (somos **~1550**)
 - × También en el **canal** de IRC (**#pyar**, en Freenode)
 - × **Asistiendo** a las **reuniones** y eventos
 - × Más info en la página: **www.python.org.ar**
- **PyAr es federal**
 - × Se organizan **reuniones en otras provincias**
 - × No hay que pedir permiso, **sólo coordinarlas**

Corriendo e interpretando

- Menos charla y más acción
 - × Python es interpretado
 - × No hace falta compilar
 - × Ciclo corto de pruebas
 - × Y encima tenemos el **Intérprete Interactivo**
- **Go! Go! Go!**
 - × Acá es donde vamos **a la realidad**, :)
 - × ¡Juro que antes andaba!
 - × **Go!**



Tipos de datos

Haciendo números

Enteros

```
>>> 2 + 2
```

```
4
```

```
>>> (50 - 5 * 6) / 4
```

```
5.0
```

```
>>> 7 / 2
```

```
3.5
```

```
>>> 7 % 3
```

```
1
```

```
>>> 23477 ** 13
```

```
658193818783243403439870082773641292484261074235533  
098117
```

Floats

```
>>> 3 * 3.75 / 1.5
```

```
7.5
```

```
>>> 7 / 2.3
```

```
3.0434782608695654
```

Más números

Complejos, decimales, fracciones

```
>>> (3-4j) ** 2.1
(-10.797386682316887-27.308377455385106j)
>>> Decimal(2).sqrt()
Decimal('1.414213562373095048801688724')
>>> Fraction(2, 3) ** 3
Fraction(8, 27)
```

Otras bases

```
>>> 0xf4
244
>>> 0b100010101
277
>>> hex(5566), bin(5566)
('0x15be', '0b1010110111110')
```

Cadenas

Comillas, apóstrofos, triples

```
>>> 'Una cadena es una secuencia de caracteres'
'Una cadena es una secuencia de caracteres'
>>> "Ella dijo: 'si'"
"Ella dijo: 'si'"
>>> """Una linea
... y la otra"""
'Una linea\ny la otra'
```

Algunas operaciones

```
>>> "Hola" + " mundo"
'Hola mundo'
>>> "Eco " * 4
'Eco Eco Eco Eco '
>>> len("Hola mundo")
10
>>> "moño".encode("utf8")
b'mo\xc3\xb1o'
```

Accediendo a las cadenas

Por posición

```
>>> saludo = 'Hola mundo'
>>> saludo[0]
'H'
>>> saludo[3]
'a'
>>> saludo[-2]
'd'
```

Rebanando

```
>>> saludo[2:5]
'la '
>>> saludo[2:8]
'la mun'
>>> saludo[:4]
'Hola'
>>> saludo[-2:]
'do'
```

Listas

Corchetes, varios tipos de elementos

```
>>> a = ['harina', 100, 'huevos', 'manteca']  
>>> a  
['harina', 100, 'huevos', 'manteca']
```

Accedemos como cualquier **secuencia**

```
>>> a[0]  
'harina'  
>>> a[-2:]  
['huevos', 'manteca']
```

Concatenamos, reemplazamos

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


Y dale con las listas

Pueden tener incluso **otras listas**

```
>>> a
['sal', 100, 'huevos', 'manteca']
>>> a[1] = ["Hola", 7]
>>> a
['sal', ['Hola', 7], 'huevos', 'manteca']
```

Borramos elementos

```
>>> del a[1]
>>> a
['sal', 'huevos', 'manteca']
```

Tenemos otros **métodos**

```
>>> a.index("huevos")
1
>>> a.sort()
>>> a
['huevos', 'manteca', 'sal']
```

Conjuntos

Definimos con llaves, poniendo valores

```
>>> nros = {1, 2, 1, 3, 1, 4, 1, 5}
>>> nros
set([1, 2, 3, 4, 5])
>>> otros = {4, 5, 6, 7}
>>> otros.update([6, 7, 8])
>>> otros
set([8, 4, 5, 6, 7])
```

Operamos

```
>>> nros - otros
set([1, 2, 3])
>>> nros & otros
set([4, 5])
>>> nros | otros
set([1, 2, 3, 4, 5, 6, 7, 8])
```

Diccionarios

Definimos con **llaves**, poniendo **pares**

```
>>> días = {"enero": 31, "junio": 30, "julio": 30}
>>> días
{'enero': 31, 'julio': 30, 'junio': 30}
>>> días["enero"]
31
>>> días["agosto"] = 31
>>> días["julio"] = 31
>>> días
{'agosto': 31, 'enero': 31, 'julio': 31, 'junio': 30}
>>> cualquiercosa = {34: [2, 3], (2, 3): {3: 4}}
```

Borrando

```
>>> del días["julio"]
>>> días
{'agosto': 31, 'enero': 31, 'junio': 30}
```

Más diccionarios

Viendo qué hay

```
>>> "marzo" in días
```

```
False
```

```
>>> días.keys()
```

```
dict_keys(['enero', 'agosto', 'junio'])
```

```
>>> días.values()
```

```
dict_values([31, 31, 30])
```

Otros métodos

```
>>> días.get("agosto", "No tenemos ese mes")
```

```
31
```

```
>>> días.get("mayo", "No tenemos ese mes")
```

```
'No tenemos ese mes'
```

```
>>> días.pop("agosto")
```

```
31
```

```
>>> días
```

```
{'enero': 31, 'junio': 30}
```



Controles de flujo

Si tal cosa o la otra

Estructura del `if`

```
a = ...  
  
if a == 0:  
    print("Ojo con el valor de b")  
    b = 0  
  
elif a > 100 or a < 0:  
    print("Error en el valor de a")  
    b = 0  
  
else:  
    b = c / a  
print(b)
```

Eso que hay `después` del `if`:

- or, and, not
- < > == != <= >= in is
- **Todo** evalúa a Falso o Verdadero

Por cada uno

Estructura del `for`

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

Si queremos la **secuencia de números**

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> for i in range(2, 10, 3):
...     print(i ** 2)
...
4
25
64
```

Mientras tanto...

Estructura del `while`

```
>>> a = 0
>>> while a < 1000:
...     print(a ** 5)
...     a += 3
0
243
7776
...
980159361278976
995009990004999
```

Al igual que el `for`, tiene:

- `continue`: Vuelve a empezar al principio del loop
- `break`: Corta el loop y sale
- `else`: Lo ejecuta si no cortamos con el `break`

Excepciones

Sucedan cuando algo se **escapa de lo normal**

```
>>> 14 / 2
```

```
7.0
```

```
>>> 14 / 0
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ZeroDivisionError: division by zero
```

Podemos **capturarlas**

```
>>> try:
```

```
...     print(14 / 0)
```

```
... except ZeroDivisionError:
```

```
...     print("error!")
```

```
...
```

```
error!
```

Manejando lo excepcional

Es muy versátil

- `try`: Acá va el bloque de código que queremos supervisar
- `except`: Atrapa todo, o sólo lo que se le especifique
- `else`: Si **no hubo** una excepción, se ejecuta esto
- `finally`: Lo que esta acá se ejecuta **siempre**
- Se pueden **combinar** de cualquier manera

Y podemos **generar** excepciones

```
>>> raise ValueError("Acá contamos que pasó")
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ValueError: Aquí contamos que pasó
```

Encapsulando código

Funciones

Estructura básica

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

Las funciones son **objetos**

```
>>> alcuadrado  
<function alcuadrado at 0xb7c30b54>  
>>> f = alcuadrado  
>>> f(5)  
25
```

Más funciones

Mucha **flexibilidad** con los **argumentos**

```
>>> def func(a, b=0, c=7):  
...     return a, b, c  
...
```

```
>>> func(1)
```

```
(1, 0, 7)
```

```
>>> func(1, 3)
```

```
(1, 3, 7)
```

```
>>> func(1, 3, 9)
```

```
(1, 3, 9)
```

```
>>> func(1, c=9)
```

```
(1, 0, 9)
```

```
>>> func(b=2, a=-3)
```

```
(-3, 2, 7)
```

Clases

Armando una clase

```
>>> class MiClase:
...     x = 3
...     def f(self):
...         return 'Hola mundo'
...
>>> c = MiClase()
>>> c.x
3
>>> c.f()
'Hola mundo'
```

Heredando

```
>>> class MiClase(ClasePadre):
>>> class MiClase(ClasePadre, ClaseTio):
```

Otra clase sobre clases

```
>>> class Posicion:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...     def distancia(self):
...         dist = math.sqrt(self.x ** 2 + self.y
... ** 2)
...         return dist
...
>>>
>>> pos1 = Posicion(3, 4)
>>> pos1.x
3
>>> pos1.distancia()
5.0
>>> pos2 = Posicion(7, 9)
>>> pos2.y
9
>>> pos1.y
4
```

El módulo más paquete

- Módulos

- × Funciones, o clases, o lo que sea en un archivo
- × Es un `.py normal`, sólo que lo importamos y usamos
- × Fácil, rápido, funciona

Tengo un `pos.py`, con la clase de la filmina anterior:

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

- Paquetes

- × Cuando tenemos muchos módulos juntos
- × Usamos directorios, e incluso subdirectorios



Tres detalles

Generadores

Ejemplo: Función que devuelve una cantidad de algos

```
>>> def fibonacci(cant):  
...     valores = []  
...     a, b = 0, 1  
...     while len(valores) < cant:  
...         valores.append(b)  
...         a, b = b, a+b  
...     return valores  
...  
>>> fibonacci  
<function fibonacci at 0xb7c30b54>  
>>> fibonacci(10)  
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]  
>>> procesar(fibonacci(10))  
'Ok'  
>>> procesar(fibonacci(9999999999999999999999)) #  
ouch!!!!
```

Seguimos generando

Somos vagos, vamos devolviendo **valor por valor**

```
>>> def fibonacci(cant):  
...     a, b, i = 0, 1, 0  
...     while i < cant:  
...         yield b  
...         a, b = b, a+b  
...         i += 1  
...  
>>> fibonacci  
<function fibonacci at 0xb7c30bfc>  
>>> fibonacci(10)  
<generator object at 0xb7c294ac>  
  
>>> procesar(fibonacci(10))  
'Ok'  
  
>>> procesar(fibonacci(9999999999999999999999))  
'Ok'
```

Entendiendo de listas

Queremos **procesar elementos** de una lista

```
>>> vec = [3, 7, 12, 0, 3, -13, 45]
>>> result = []
>>> for x in vec:
...     result.append(x ** 2)
...
>>> result
[9, 49, 144, 0, 9, 169, 2025]
```

Quizás **no todos**

```
>>> vec = [3, 7, 12, 0, 3, -13, 45]
>>> result = []
>>> for x in vec:
...     if x <= 7:
...         result.append(x ** 2)
...
>>> result
[9, 49, 0, 9, 169]
```

Entendiendo de listas

List comprehensions

```
>>> vec = [3, 7, 12, 0, 3, -13, 45]
```

```
>>> [x ** 2 for x in vec]
```

```
[9, 49, 144, 0, 9, 169, 2025]
```

```
>>> [x ** 2 for x in vec if x <= 7]
```

```
[9, 49, 0, 9, 169]
```

Generator comprehensions!

```
>>> sum(x ** 2 for x in range(1000))
```

```
332833500
```

```
>>> len(x for x in range(1000) if (x ** 2) % 2 ==  
    0)
```

```
500
```

Una gran idea

- Hay varios espacios de nombres
 - × Básicos: local y global
 - × Los tienen las funciones, clases, módulos

¡El mismo ejemplo que antes!

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

- Más útiles de lo que parecen
 - × Simplifican la estructura a mentalizar
 - × Prolijidad, legibilidad, traceabilidad
 - × Pruébenlos, los van a extrañar cuando no los tengan

¿Preguntas?

¿Sugerencias?

Espero que lo hayan disfrutado
Tanto como yo, :)

¡Muchas gracias!

Facundo Batista

facundo@taniquetil.com.ar

www.taniquetil.com.ar



Licencia: Creative Commons [@facundobatista](http://creativecommons.org/licenses/by-nc-sa/2.5/deed.es_AR)
Atribución-NoComercial-CompartirDerivadasIgual 2.5 Argentina
http://creativecommons.org/licenses/by-nc-sa/2.5/deed.es_AR