

# Python 101

Lucas Emmanuel Bais  
lucas.bais@gmail.com



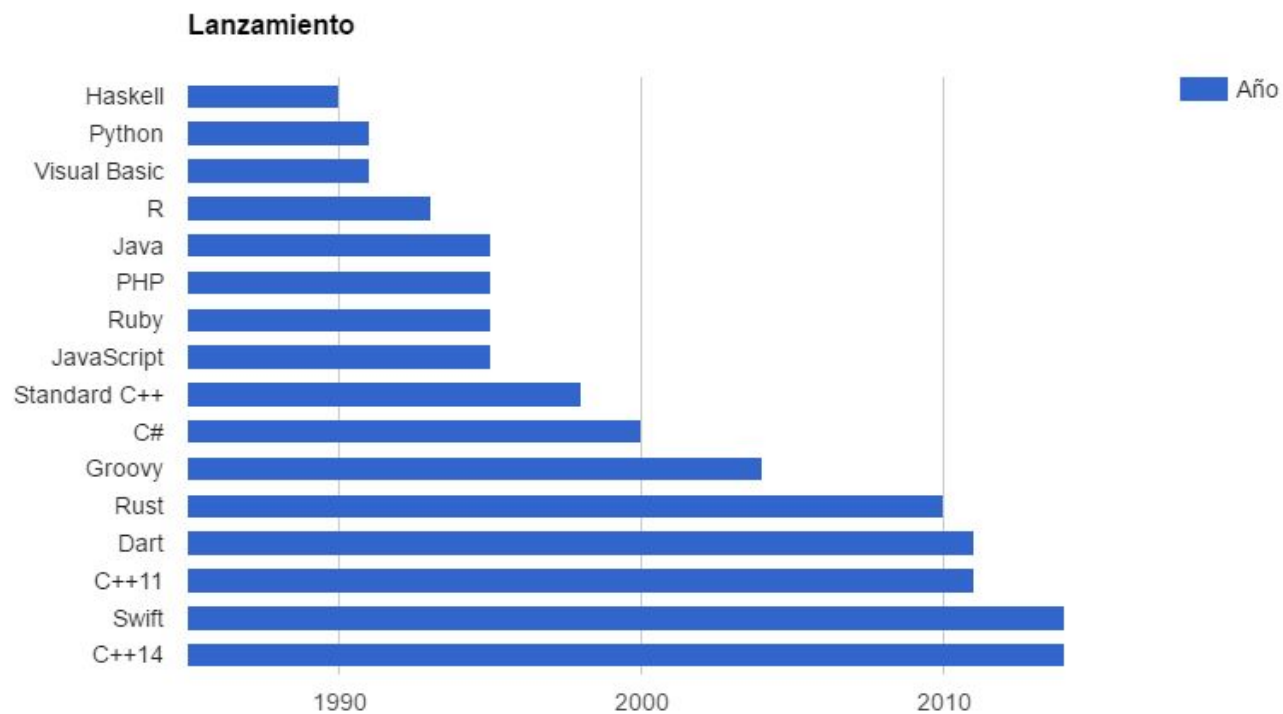
¿Qué es Python?

# Algo de historia

- En 1991, se publicó el código de la versión 0.9.0. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones y funciones.
- Python alcanzó la versión 1.0 en enero de 1994.
- Creador: Guido van Rossum
- Desarrolladores: Python Software Foundation (organización sin fines de lucro)

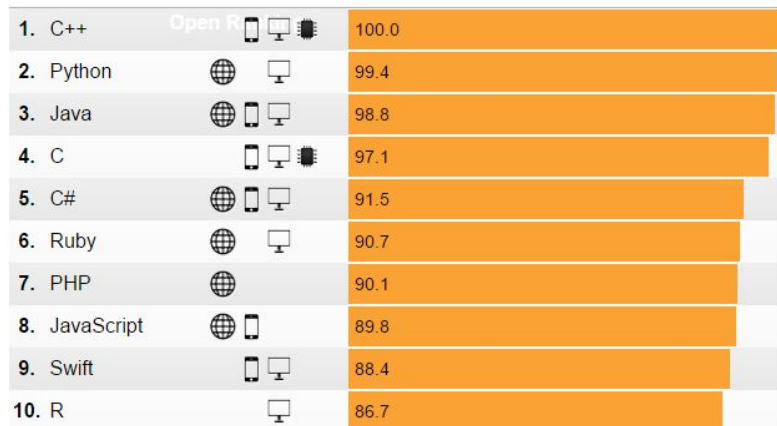


Los Monty Python, de allí su nombre



# Popularidad 2015

## Open source



## Trabajo



Fuente (IEEE): <http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015>

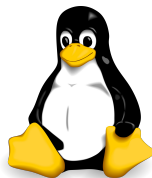
# Generalidades

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma.

# Generalidades

- Lenguaje de alto nivel (mucho nivel de abstracción)
- Python Software Foundation License (compatible GNU - GPL)
- De propósito general
- Orientado a objetos, imperativo y con características de lenguaje funcional
- Multiplataforma



# Generalidades

- Manejo de memoria dinámica (con garbage collector)
- Dynamic name resolution
- Sintaxis limpia - Indentado (whitespace) ¡Nos ahorramos llaves y puntos y coma!

C++ / ANSI C /

Java

```
unsigned int i =  
0;  
while(i<n){  
    hacerAlgo(i);  
    i++;  
}
```

Python

```
i = 0  
while(i<n):  
    hacerAlgo(i)  
    i +=1
```



# Aplicaciones

- Web (¡Servicios y más!)
- Apps Desktop
- Investigación
- Big data analysis
- Juegos
- Bash Scripting
- Y mucho más.



# Python en la vida real

Performance/Estabilidad de Python, ¿se la banca?

<http://www.python.org.ar/wiki/RendimientoPythonVsJavaVsNet>

¿Que aplicaciones (conocidas) están hechas en Python?

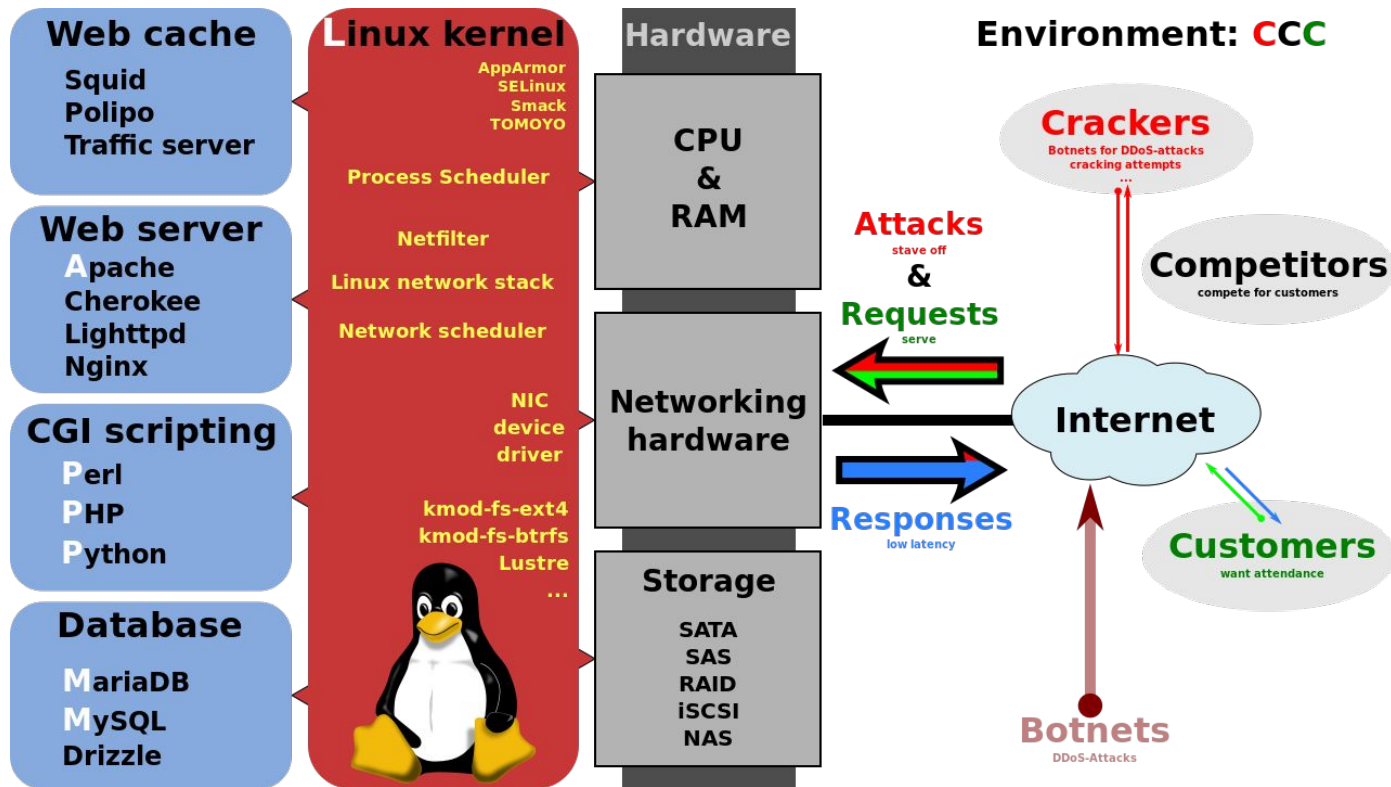
BitTorrent (original): programa para compartir archivos p2p (interfaz wx)

Trac: sistema de gestión de proyectos (interfaz web)

YouTube (<http://youtube.com>)

Reddit (<http://reddit.com>)

# ¿Donde vive Python?



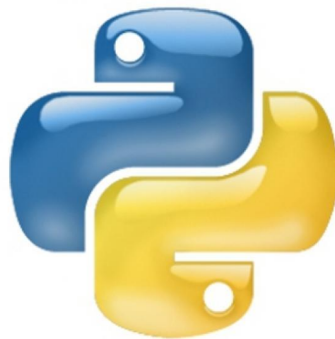
# Filosofía Python - too long, didn't read

- 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.
- Lo práctico gana a lo puro.
- 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.<sup>15</sup>
- 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 (namespaces) son una gran idea ¡Hagamos más de esas cosas!

# El intérprete

En ciencias de la computación, intérprete o interpretador es un programa informático capaz de analizar y ejecutar otros programas.

En la actualidad, uno de los entornos más comunes de uso de los intérpretes es en los navegadores web, debido a la posibilidad que estos tienen de ejecutarse independientemente de la plataforma.



# Intérprete vs Compilador



# ¿Varios tipos de Python?

No solo hay múltiples versiones de Python que contienen distintas funcionalidades (ya las veremos), sino que también existen distintas implementaciones de Python con diferentes intérpretes. Ya que Python en sí es un standard.

Algunos ejemplos: CPython, IronPython, Jython, MicroPython, PyPy, etc.

Nosotros utilizaremos CPython, que utiliza intérprete y core escrito en C.

# ¿Qué es el tipado dinámico?

Un lenguaje de programación es dinámicamente tipado si una misma variable puede tomar valores de distinto tipo en distintos momentos.

En ANSI C / C++/ Java:

```
unsigned int valor = 3;  
valor = "Jorge"; → !!!!Error!!!!  
Me dijiste que era un entero  
:(
```

En Python:

```
valor = 3  
valor = "Jorge"  
OK!
```





Entonces... ¿por qué python?

# Herramientas de trabajo

## Escritura:

Editores de texto: Sublime, Atom, vim, gedit, etc.

IDEs: Eclipse + pydev (plug-in), Netbeans, NINJA-IDE y muchos más...

## Python (intérprete):

Windows: <https://www.python.org/downloads/windows/>

Linux: `sudo apt-get install python` (en general, ya viene con 2.7 y 3.4-3.5)

Mac: <https://www.python.org/downloads/mac-osx/> (en general, ya viene con 2.7)

GIT / BitBucket (opcional) - <https://github.com/>

Slack (opcional) - <https://python-utn-fra.slack.com/>

# Nuestro primer “¡Hola mundo!”

prompt> python

Python 2.7.6 (default, Jun 22 2015, 17:58:13) → la versión de python

[GCC 4.8.2] on linux2 → compilador del interprete

```
>> print("¡Hola mundo!")
```

Desde archivo:

Generamos archivo de texto hola.py, escribimos el print y luego ejecutamos “python hola.py”

También podemos generar código ejecutable en linux: `chmod +x hola.py`



¡Ejecutemos una prueba simple en  
python y comparemos con C++!

# Biblioteca estándar - Constantes

False, True, None, Ellipsis (mismo que ...), \_\_debug\_\_(true, si se ejecuta con -O)

# Biblioteca estándar - Operadores booleanos

Operador	
<	or
<=	and
>	not
>=	...
==	
!=	
is	
is not	

Operador	Significado
x   y	or entre bits
x ^ y	or excluyente entre bits
x & y	and entre bits
x << n	x movido a la izquierda n bits
x >> n	x movido a la derecha n bits
~x	bits invertidos

# Type

```
>>> type("hola")  
<type 'str'>
```

```
>>> type("hola") == str  
True
```

¿Qué pasa con los números?

```
isinstance(valor, ....)
```

```
isinstance(True, int)
```

¿que pasa?

# Biblioteca estándar - Tipos

- *Numeric Types* — *int, float, complex*
- *Tipos texto* - *str* → “hola”
- *None* (*null, nil*)
- *class* **list**(*[iterable]*) → [1,2,3] , [“a”,”b”,”c”] , “abc”, [“a”,1,23]
- *class* **range**(*stop*)
- *class* **range**(*start, stop[, step]*)
- *class* **tuple**(*[iterable]*)
- *class* **dict**(*\*\*kwarg*) tel = {'jack': 4098, 'sape': 4139}
- *class* (soporte con clases) **Enum**
- *Class* **set**(*[iterable]*) basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
- (hay más)
- <https://docs.python.org/3/tutorial/datastructures.html>

# Biblioteca estándar - funciones

<a href="#"><u>abs()</u></a>	<a href="#"><u>dict()</u></a>	<a href="#"><u>help()</u></a>	<a href="#"><u>min()</u></a>	<a href="#"><u>setattr()</u></a>
<a href="#"><u>all()</u></a>	<a href="#"><u>dir()</u></a>	<a href="#"><u>hex()</u></a>	<a href="#"><u>next()</u></a>	<a href="#"><u>slice()</u></a>
<a href="#"><u>any()</u></a>	<a href="#"><u>divmod()</u></a>	<a href="#"><u>id()</u></a>	<a href="#"><u>object()</u></a>	<a href="#"><u>sorted()</u></a>
<a href="#"><u>ascii()</u></a>	<a href="#"><u>enumerate()</u></a>	<a href="#"><u>input()</u></a>	<a href="#"><u>oct()</u></a>	<a href="#"><u>staticmethod()</u></a>
<a href="#"><u>bin()</u></a>	<a href="#"><u>eval()</u></a>	<a href="#"><u>int()</u></a>	<a href="#"><u>open()</u></a>	<a href="#"><u>str()</u></a>
<a href="#"><u>bool()</u></a>	<a href="#"><u>exec()</u></a>	<a href="#"><u>isinstance()</u></a>	<a href="#"><u>ord()</u></a>	<a href="#"><u>sum()</u></a>
<a href="#"><u>bytearray()</u></a>	<a href="#"><u>filter()</u></a>	<a href="#"><u>issubclass()</u></a>	<a href="#"><u>pow()</u></a>	<a href="#"><u>super()</u></a>
<a href="#"><u>bytes()</u></a>	<a href="#"><u>float()</u></a>	<a href="#"><u>iter()</u></a>	<a href="#"><u>print()</u></a>	<a href="#"><u>tuple()</u></a>
<a href="#"><u>callable()</u></a>	<a href="#"><u>format()</u></a>	<a href="#"><u>len()</u></a>	<a href="#"><u>property()</u></a>	<a href="#"><u>type()</u></a>
<a href="#"><u>chr()</u></a>	<a href="#"><u>frozenset()</u></a>	<a href="#"><u>list()</u></a>	<a href="#"><u>range()</u></a>	<a href="#"><u>vars()</u></a>
<a href="#"><u>classmethod()</u></a>	<a href="#"><u>getattr()</u></a>	<a href="#"><u>locals()</u></a>	<a href="#"><u>repr()</u></a>	<a href="#"><u>zip()</u></a>
<a href="#"><u>compile()</u></a>	<a href="#"><u>globals()</u></a>	<a href="#"><u>map()</u></a>	<a href="#"><u>reversed()</u></a>	<a href="#"><u>__import__()</u></a>
<a href="#"><u>complex()</u></a>	<a href="#"><u>hasattr()</u></a>	<a href="#"><u>max()</u></a>	<a href="#"><u>round()</u></a>	
<a href="#"><u>delattr()</u></a>	<a href="#"><u>hash()</u></a>	<a href="#"><u>memoryview()</u></a>	<a href="#"><u>set()</u></a>	



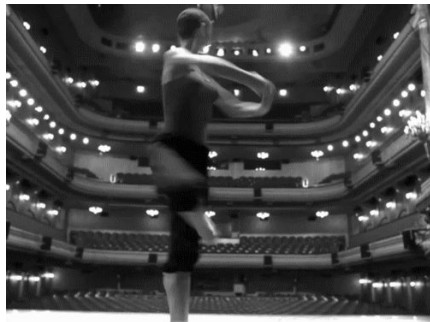
# Estructuras de control - For

```
for i in range(10):  
    print("Estoy por el...",i)
```

# Estructuras de control - while

```
i = 0  
while i<10:  
    print("Estoy por el...",i)  
    i +=1
```

('Estoy por el...', 0)  
('Estoy por el...', 1)  
('Estoy por el...', 2)  
('Estoy por el...', 3)  
('Estoy por el...', 4)  
('Estoy por el...', 5)  
('Estoy por el...', 6)  
('Estoy por el...', 7)  
('Estoy por el...', 8)  
('Estoy por el...', 9)



# Estructuras de control - If, elif, else

```
for i in range(3, 0, -1):  
    if i == 0 :  
        print("Soy el 0!")  
    elif i % 2 == 0:  
        print("Soy Par :)!")  
    else:  
        print("Soy Impar :(!")
```

Soy Impar :(!  
Soy Par :)!  
Soy Impar :(!

¿Cómo hago para que figure el 0?

```
if None:  
    print("se ejecuta?")
```

# Más estructuras de control

Continue, return, break, yield, pass, raise

# Generando .pyc

```
python -m compileall .
```





*That's all Folks!*