Python 101

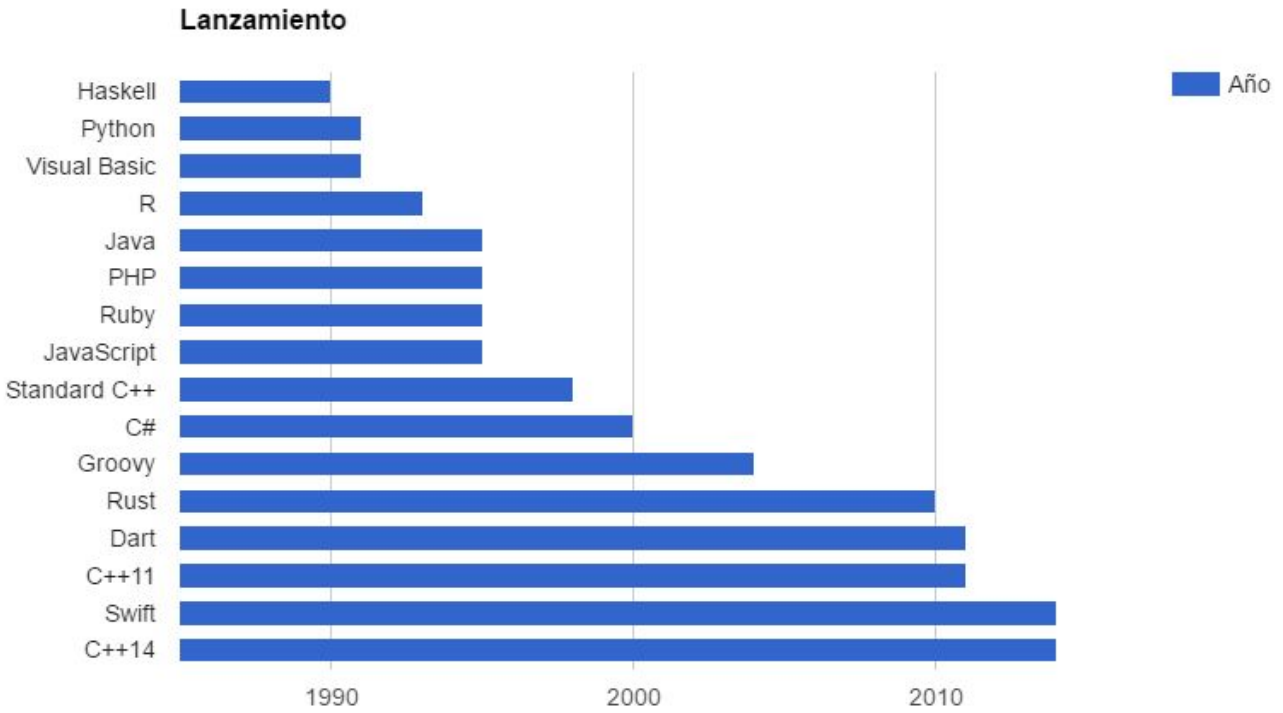
Lucas Emmanuel Bais
lucas.bais@gmail.com
(enviar correo, con asunto python utn-fra)

¿Qué es Python?

Algo de historia

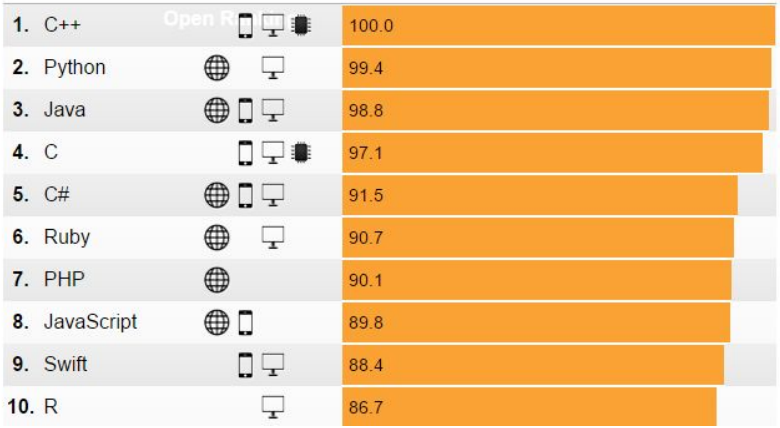
- En 1991, se publicó el código de la versión 0.9.0. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones y funciones.
- Python alcanzó la versión 1.0 en enero de 1994.
- Creador: Guido van Rossum
- Desarrolladores: Python Software Foundation (organización sin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

Los Monty Python, de allí su nombre


Lanzamiento
Haskell
Python
Visual Basic
R
Java
PHP
Ruby
JavaScript
Standard C++
C#
Groovy
Rust
Dart
C++11
Swift
C++14
1990
2000
2010
Año

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!
- Comparativa de sintaxis python vs c++
- Comparativa contra otros lenguajes

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?

¿Que aplicaciones (conocidas) están hechas en Python?

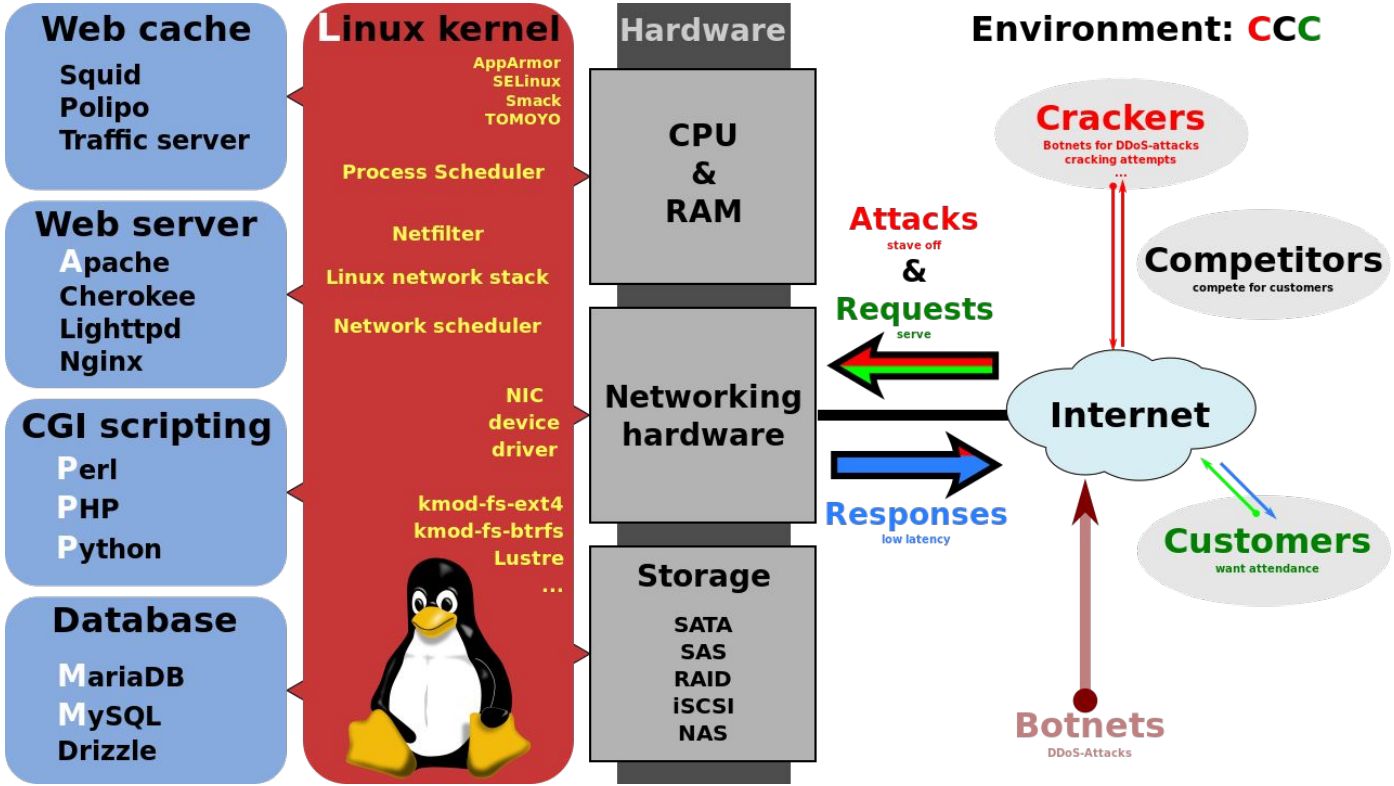
BitTorrent (original): programa para compartir archivos p2p (interfaz wx)

YouTube

Reddit

Trac: sistema de gestión de proyectos (interfaz web)

¿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.15
- 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 3.4-3.5-3.6)

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/

Básico para movernos en la terminal de linux/macos

python <opciones> <nombre de archivo> : ejecuta script de python indicado.

pwd: nos dice el directorio donde estamos parados.

ls o ls -l: nos lista los archivos en el directorio donde estamos.

mkdir <nombre>: genera una carpeta con “nombre”.

touch <nombre - extensión>: genera archivo de nombre/extensión determinada.

Tail <archivo>: nos muestra las últimas líneas del archivo.

Nuestro primer “¡Hola mundo!”

Desde la terminal:

bash#> 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

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

http://stackoverflow.com/a/231855 yield explanation

Biblioteca estándar - funciones

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

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

Generando .pyc

Archivos “compilados” para PVM se generan cuando corremos por primera vez el programa escrito en python, ejecutando el archivo .py

También pueden generarse ejecutando:
python -m compileall .

Donde -m mod : run library module as a script (terminates option list)

“.” es el directorio donde estamos parados.

Intérprete de archivo ejecutable:

#!/usr/bin/python3.5


That's all Folks!