Python Semestral Básico

Introducción e Historia

Octubre de 2017

PROTECO

Aspectos generales del curso

- Nombre del curso: Python básico
- Duración: 20 horas
- Horario: 10:00 - 14:00 (descanso 20 min)
- Fecha de inicio: 30 de Septiembre
- Fecha de término: 28 de Octubre

Instructores

-> **Chávez Delgado Jorge Luis (Titular)**

- Generación 33
- jorgechavez.proteco@gmail.com

Instructores

-> **Vargas Daniel (Adjunto)**

- Generación 33
- danielvc@comunidad.unam.mx

-> **Gutierrez Óscar (Adjunto)**

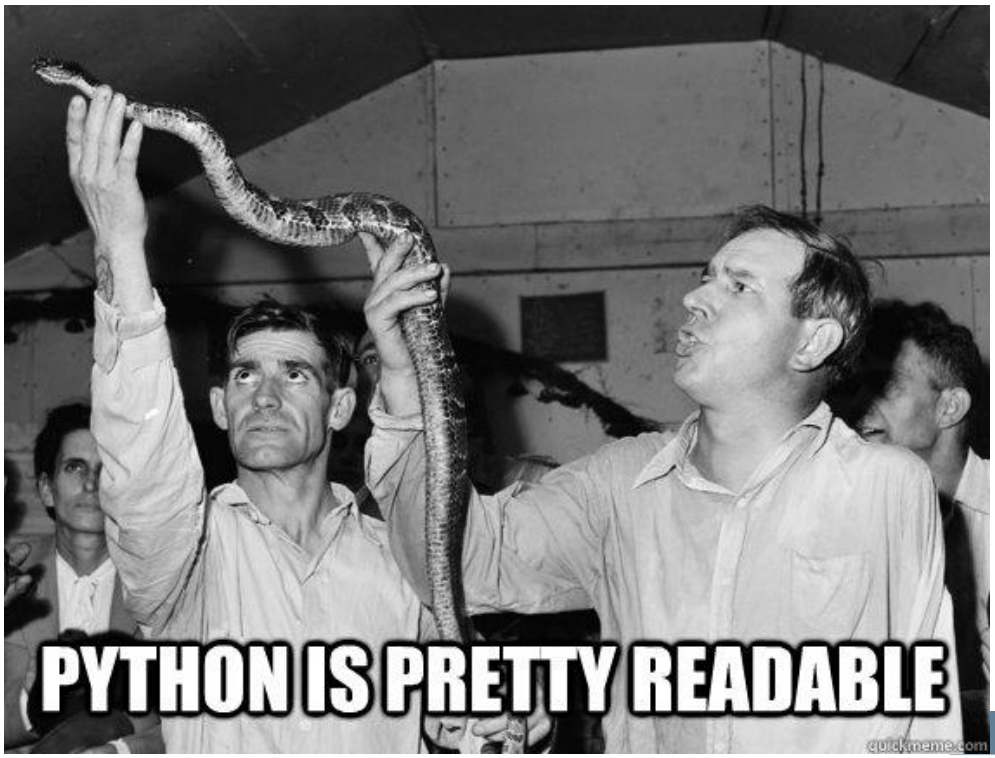
- Generación 33
- oscar.proteco@gmail.com

Presentación del curso

- Por favor menciona:
  - Nombre
  - Escuela o trabajo actual
  - Algún otro lenguaje(s) que manejes
  - ¿Por qué quieres aprender Python?
  - ¿Qué esperas de este curso?

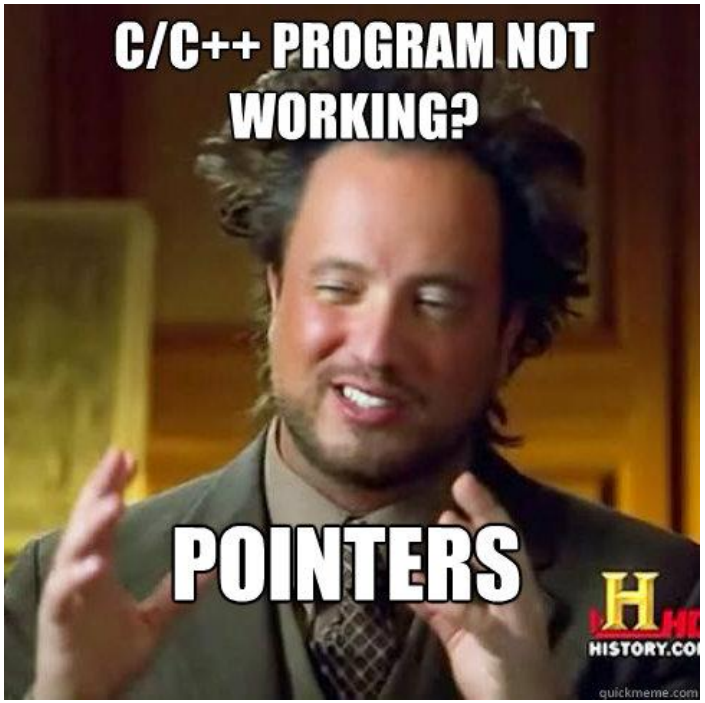
Introducción

- Es un lenguaje cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.



Introducción

- Python es un poderoso lenguaje de programación interpretado y fácil de aprender.

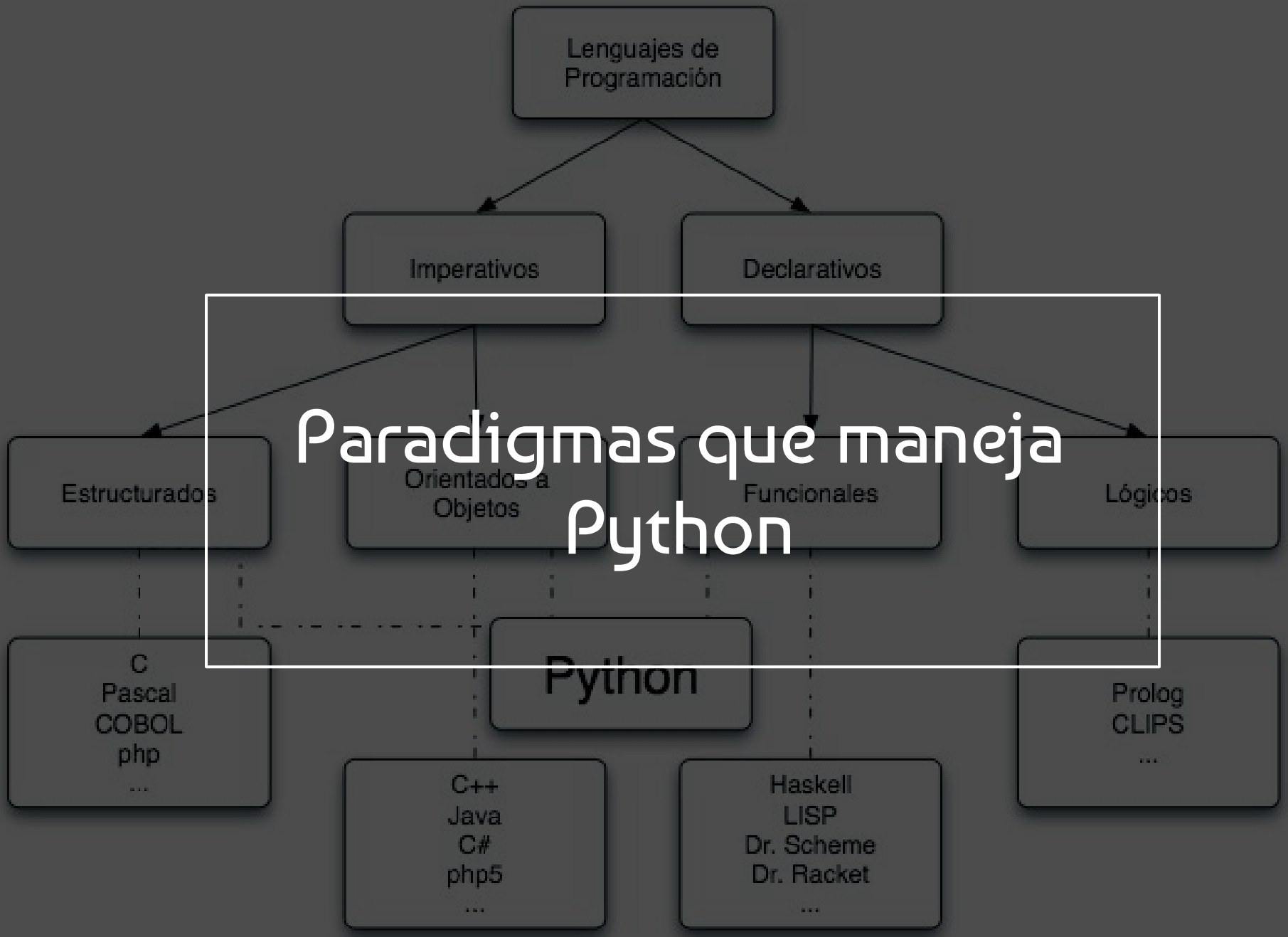


Introducción

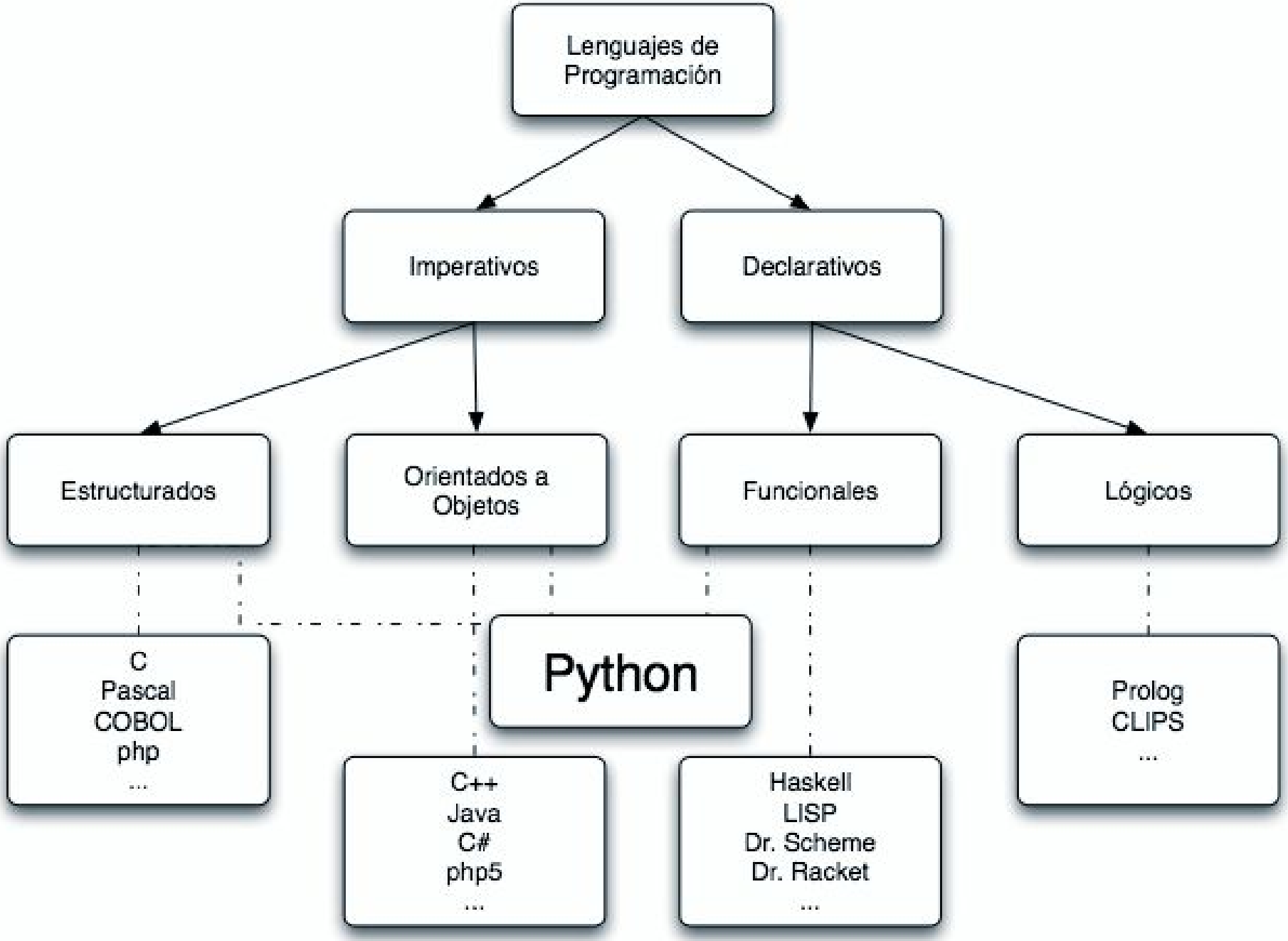
- Es un lenguaje multiparadigma, ya que soporta orientación objetos, programación estructurada y en menor medida, programación funcional.
- Un paradigma representa un enfoque particular o filosofía para diseñar soluciones.

Introducción

- Los paradigmas son diferentes en la forma de abstraer los elementos involucrados en un problema.
- Generalmente lo usamos porque nos gusta o se nos facilita entender problemas esa manera.



Paradigmas que maneja Python

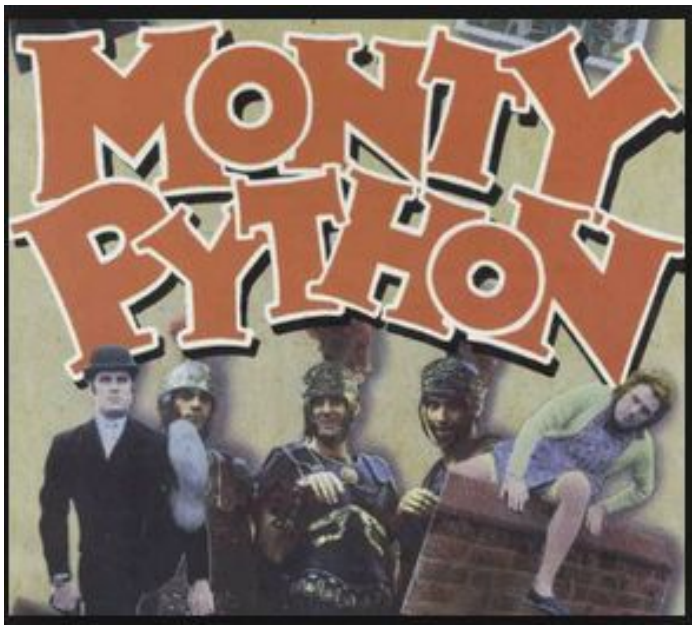

Lenguajes de Programación
Imperativos
Declarativos
Estructurados
Orientados a Objetos
Funcionales
Lógicos
C
Pascal
COBOL
php
...
Python
C++
Java
C#
php5
...
Haskell
LISP
Dr. Scheme
Dr. Racket
...
Prolog
CLIPS
...

Creador de python

- Python fue creado en 1991 por Guido van Rossum en el Centro para las Matemáticas y la Informática, como un sucesor del lenguaje de programación ABC. Guido nombró al lenguaje por su afición al programa Monty Python.

Origen del nombre

- Monty Python's Flying Circus es una serie de televisión británica, basada en sketches breves que en muchas ocasiones incluían una importante carga de crítica social, y la mayoría de las obras rozaban el absurdo total.

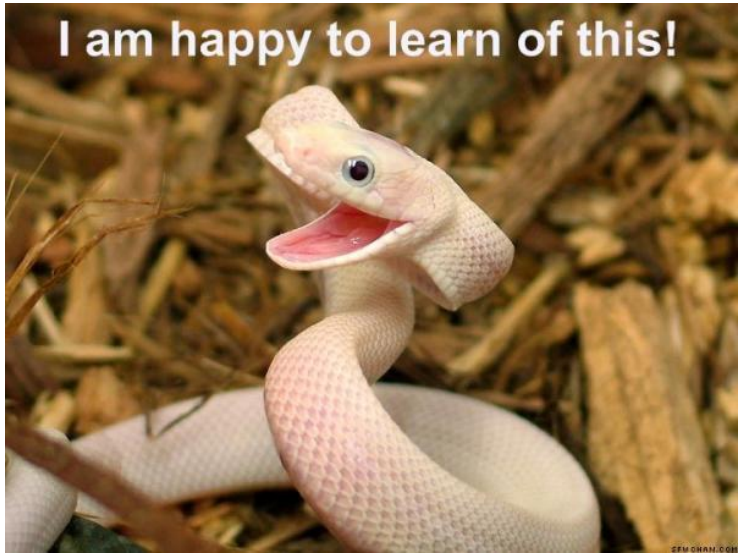


BDFL

- Benevolent Dictator for Life (BDFL) es un título informal que se otorga a ciertos individuos de la comunidad de desarrolladores de software de código abierto que tienen la tarea de asignar las directrices generales y, en ciertas situaciones, las decisiones finales dentro del ámbito de un proyecto.

Características

- Ideal para scripts multiplataforma por su naturaleza interpretada, gracias a su elegante sintaxis, tipado fuerte y dinámico y facilidad de aprendizaje fue el lenguaje más usado del 2015 y 2016.



Lenguaje de alto nivel

ANALOGÍA DEL AUTOMÓVIL

ALTA

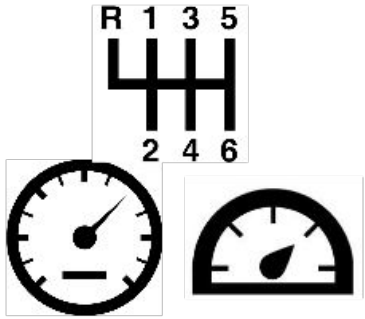
ABSTRACCIÓN

BAJA

ALTO NIVEL

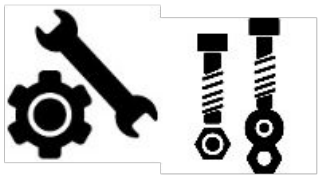
Desarrollo de aplicaciones en lenguajes como Java, Python o C#.

MEDIO NIVEL

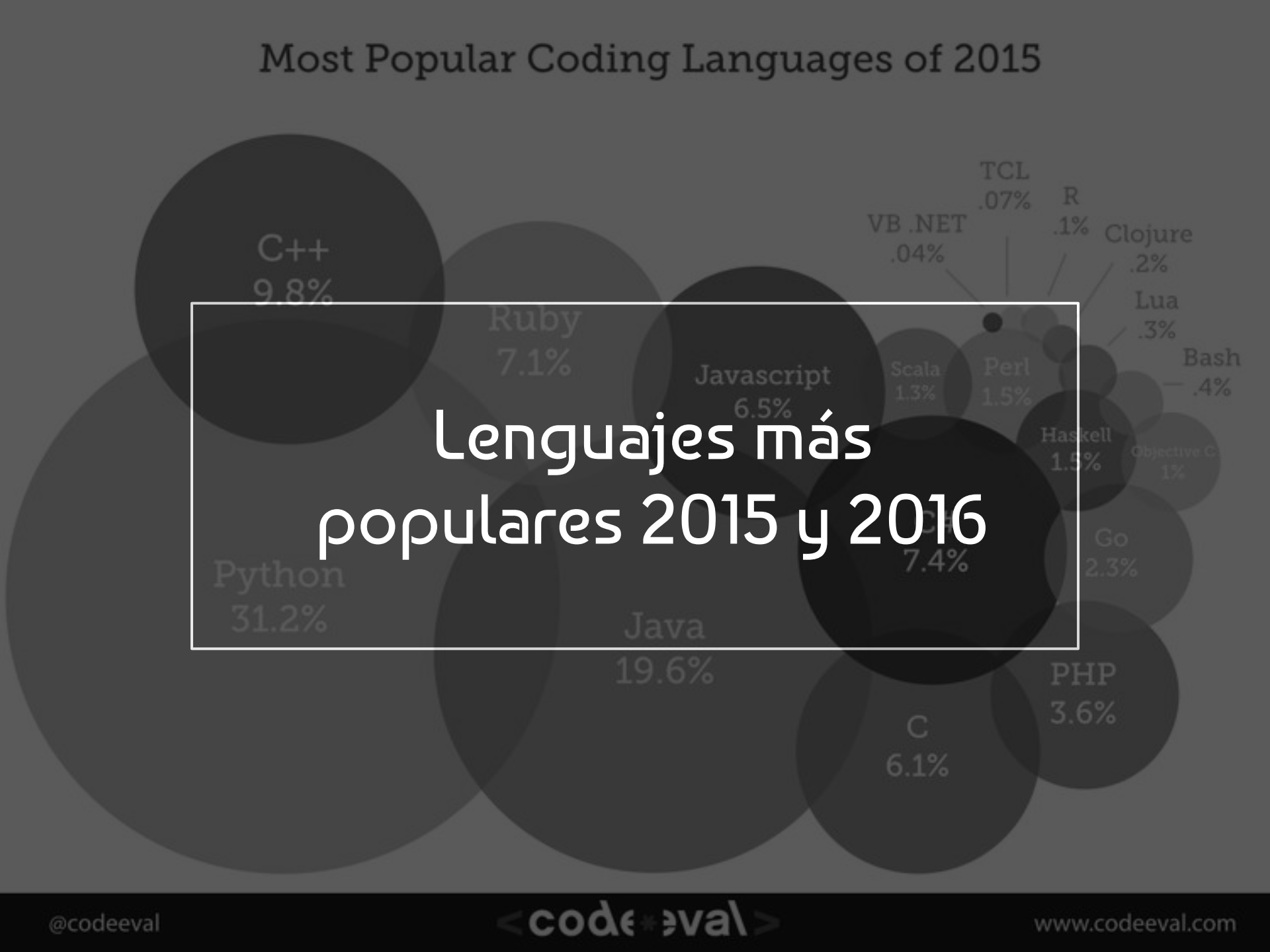


Desarrollo de sistemas, software crítico en lenguajes como C o C++.

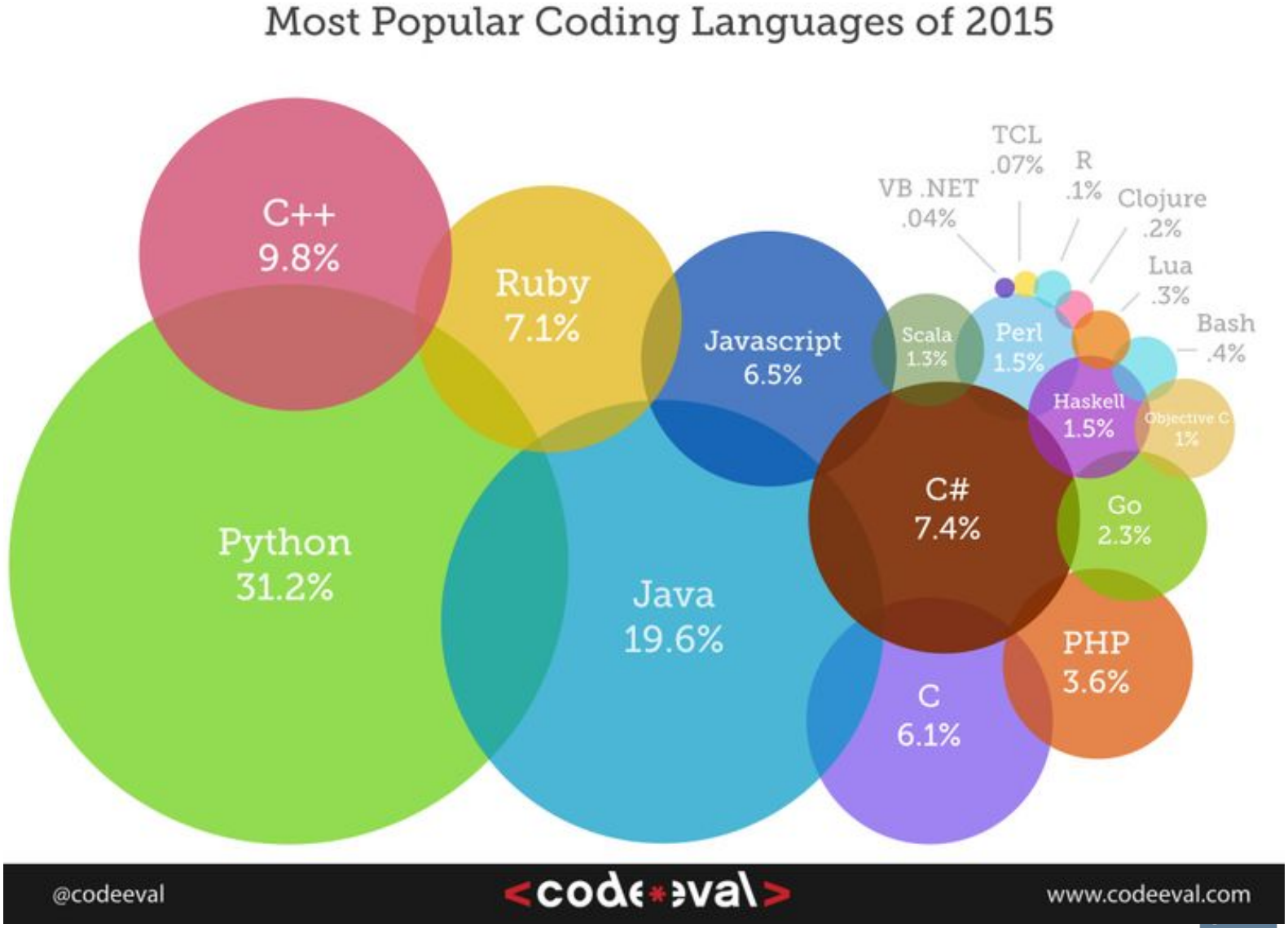
BAJO NIVEL

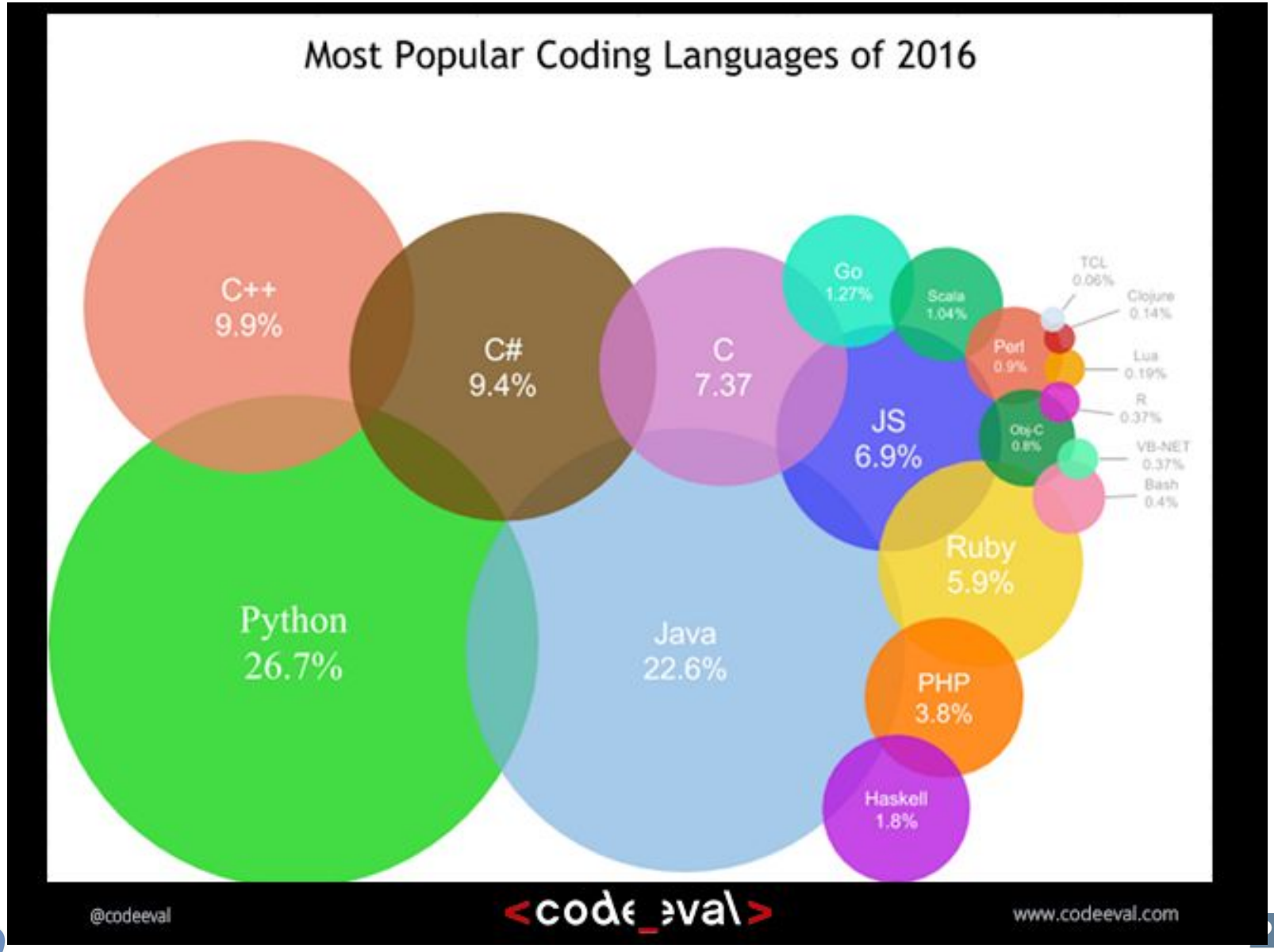


Lenguajes para la manipulación directa del hardware como ensamblador.



Lenguajes más populares 2015 y 2016


Most Popular Coding Languages of 2015
C++
9.8%
Ruby
7.1%
Javascript
6.5%
TCL
.07%
R
.1%
VB .NET
.04%
Clojure
.2%
Lua
.3%
Bash
.4%
Scala
1.3%
Perl
1.5%
Haskell
1.5%
Objective C
1%
C#
7.4%
Go
2.3%
Python
31.2%
Java
19.6%
PHP
3.6%
C
6.1%
@codeeval
<code*eval>
www.codeeval.com


Most Popular Coding Languages of 2016
C++
9.9%
C#
9.4%
C
7.37
Go
1.27%
Scala
1.04%
TCL
0.06%
Clojure
0.14%
Perl
0.9%
Lua
0.19%
R
0.37%
Obj-C
0.8%
VB-NET
0.37%
Bash
0.4%
JS
6.9%
Ruby
5.9%
Python
26.7%
Java
22.6%
PHP
3.8%
Haskell
1.8%
@codeeval
<code_eval>
www.codeeval.com

Ventajas

- Fácil de usar
  - Los tipos se asocian a objetos no a variables
  - Opera en un muy alto nivel de abstracción
  - Las reglas sintácticas son muy sencillas

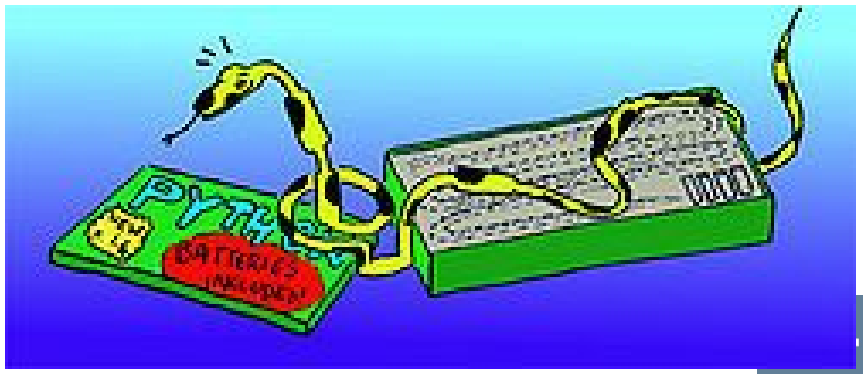
Python es muy conveniente para el desarrollo rápido de aplicaciones.

Ventajas

- Expresividad
  - Python es un lenguaje muy expresivo
    - Una línea en Python puede hacer más que una línea en cualquier otro lenguaje.
    - Menor líneas de código -> Menor tiempo en escribir y ejecutar.
    - Menor líneas de código -> Mayor facilidad para mantener y depurar los programas.

Ventajas

- Legibilidad
  - Facilidad de lectura
- Filosofía de Baterías incluidas
  - La librería estándar de Python es muy amplia



Ventajas

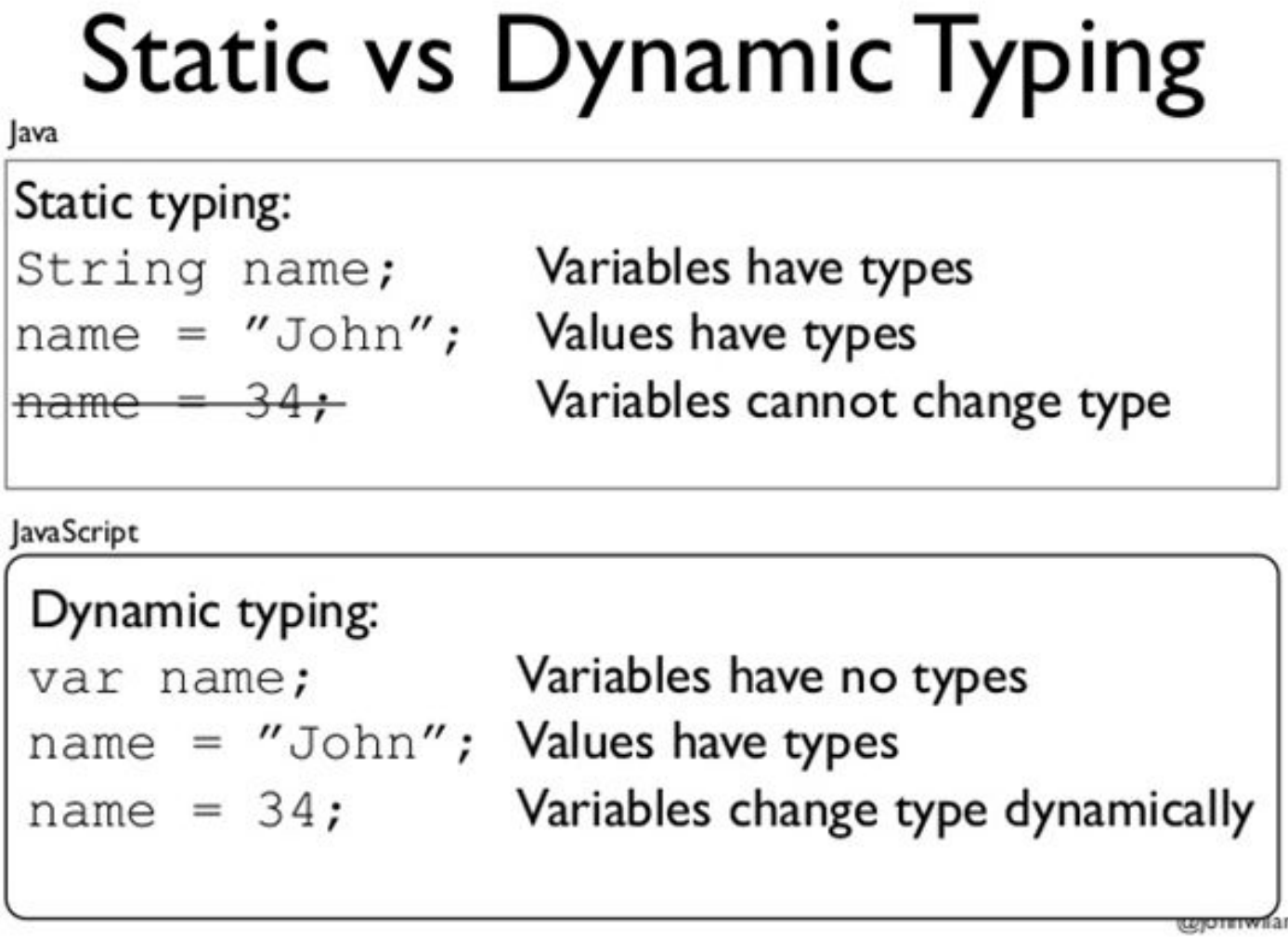
- Multiplataforma
  - Dado que es interpretado, el mismo código puede ejecutarse en cualquier plataforma que tenga instalado un intérprete de Python.

- Open Source
  - Está desarrollado con el modelo open source y está disponible libremente bajo una licencia pública general de GNU .

Características

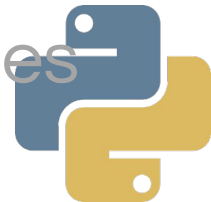
- Tipado Dinámico
  - La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, Python para todos sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.

Dinámico VS Estático



Características

- Tipado Fuerte
  - No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Por ejemplo, si tenemos una variable que contiene un texto (variable de tipo cadena o string) no podremos tratarla como un número (sumar la cadena “9” y el número 8). En otros lenguajes el tipo de la variable cambiaría para adaptarse al comportamiento esperado, aunque esto es más propenso a errores.



Fuerte VS Débil

Strong vs Weak Typing

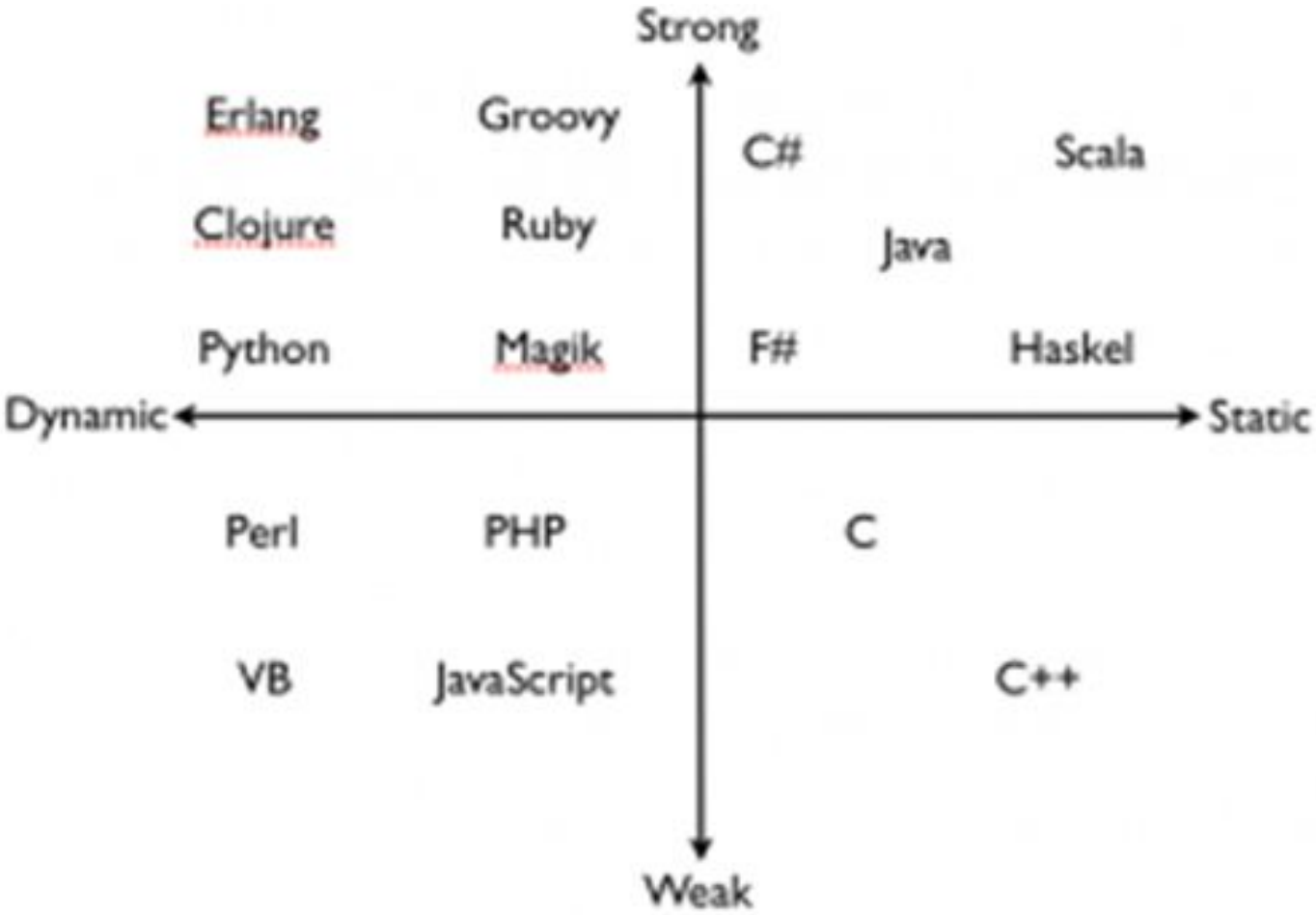
Java

`int intVar = 4;`	Pretty strong typing
`"" + intVar;`	Implicit type conversion
`intVar + 0.0;`	Implicit, widening type conv.
`intVar + (int)0.0;`	Explicit type conversion
~~`intVar + null;`~~	Not allowed

JavaScript

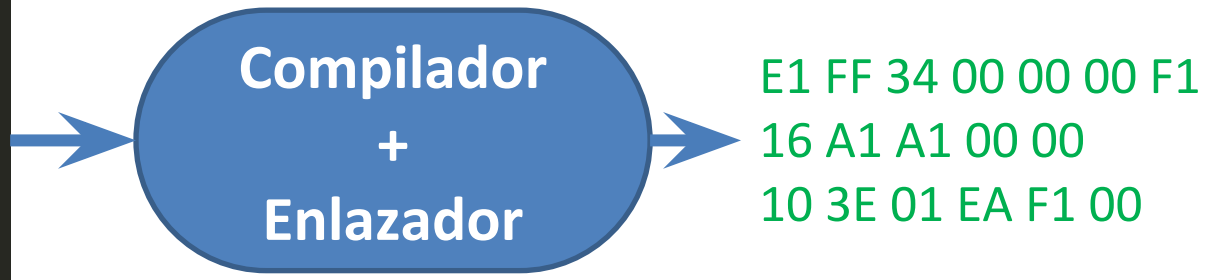
`var dynVar = 4;`	Very weak typing
`dynVar + "";`	Implicit type conversion
`dynVar + null;`	Implicit type conversion
`dynVar + [];`	Implicit type conversion
`dynVar + [0];`	Implicit type conversion

Clasificación



¿Cómo funciona Python?

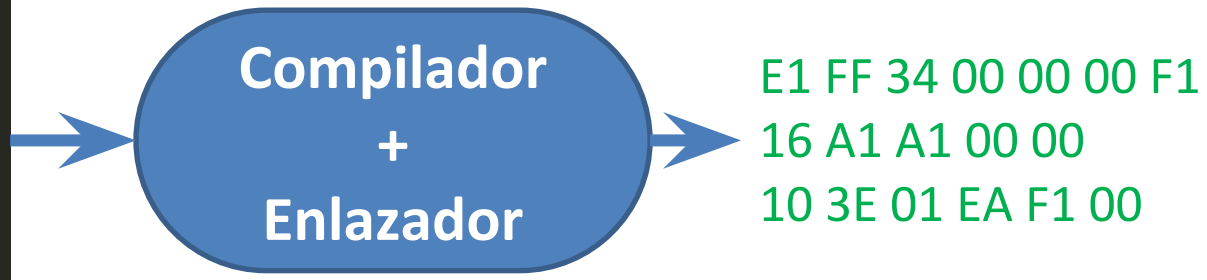
```c
/*Funciones*/
void error(void){
    perror("error: insuficiente espacio de memoria");
    exit(1);
}
nodo *NuevoNodo(){
    nodo *q = (nodo *)malloc(sizeof(nodo));
    if(!q) error();
    return (q);
}
void buscar(int, nodo **);
void visualizar_arbol(nodo *,int);
void borrar_arbol(nodo *a);
void borrar(int, nodo **);
void borrar_nodo(nodo **, nodo **);

int main(){
    nodo *raiz = NULL;
    int k;

    printf("Introducir claves. Finalizar con eof\n\n");
    printf("clave: ");

    while (scanf ("%d", &k) != EOF){
        buscar(k, &raiz);
        printf("clave: ");
```



- En un compilador el código completo se analiza y se reportan todos los errores y advertencias. El compilador genera el código máquina mientras que el enlazador se encarga de poner las funciones externas para generar un ejecutable específico para ese sistema.

¿Cómo funciona Python?

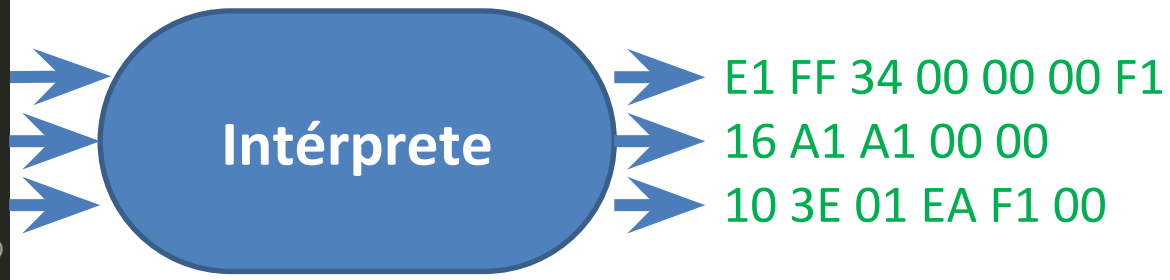
```c
/*Funciones*/
void error(void){
    perror("error: insuficiente espacio de memoria");
    exit(1);
}
nodo *NuevoNodo(){
    nodo *q = (nodo *)malloc(sizeof(nodo));
    if(!q) error();
    return (q);
}
void buscar(int, nodo **);
void visualizar_arbol(nodo *,int);
void borrar_arbol(nodo *a);
void borrar(int, nodo **);
void borrar_nodo(nodo **, nodo **);

int main(){
    nodo *raiz = NULL;
    int k;

    printf("Introducir claves. Finalizar con eof\n\n");
    printf("clave: ");

    while (scanf ("%d", &k) != EOF){
        buscar(k, &raiz);
        printf("clave: ");
```



- En un compilador el código completo se analiza y se reportan todos los errores y advertencias. El compilador genera el código máquina mientras que el enlazador se encarga de poner las funciones externas para generar un ejecutable específico para ese sistema.

¿Cómo funciona Python?

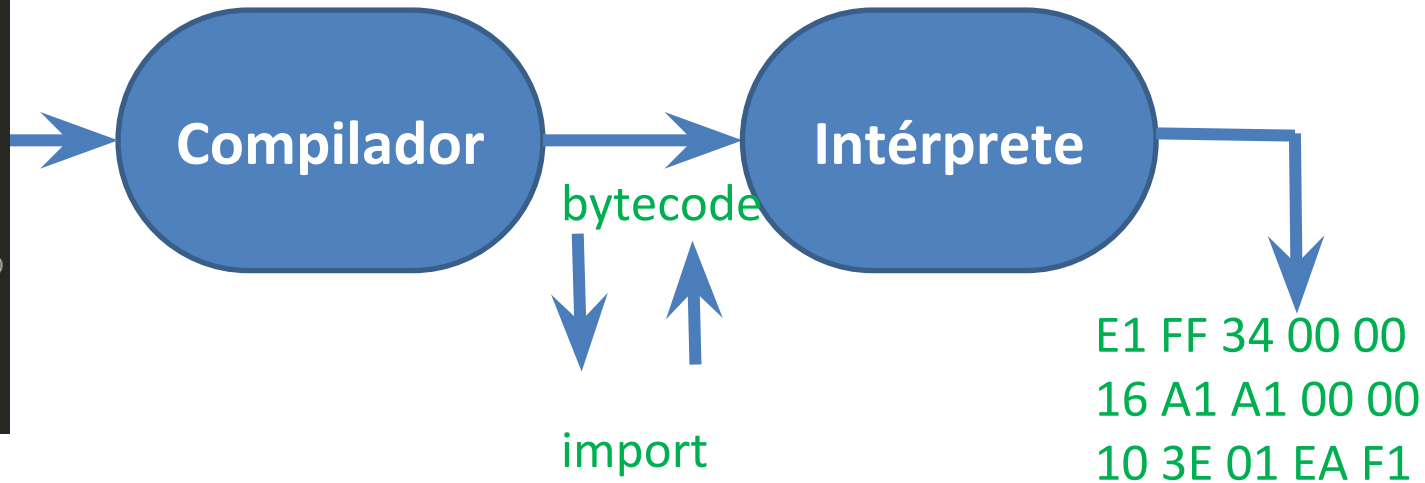
```python
class Animal():
    def __init__(self,especie,tipo,nombre):
        self.especie=especie
        self._tipo=tipo
        self.__nombre=nombre

    def setEspecie(self,especie):
        self.especie=especie

    def getEspecie(self):
        return self.especie

    def setTipo(self,tipo):
        self.tipo=tipo

    def getTipo(self):
        return self.tipo



    def reproducir(self):
        print(self._nombre, "se esta reproduciendo")


    def moverse(self):
        if self._tipo=="acuatico":
            print("Yo nado")
        elif self._tipo=="terreste":
            print("Yo camino o me arrastro")
```



- En un intérprete código se analiza y ejecuta línea por línea, por lo regular compila a un lenguaje intermedio (bytecode) y después lo traduce al código máquina. Si alguna línea falla no interrumpe la operación y sigue ejecutando las demás.

Intérprete de Python

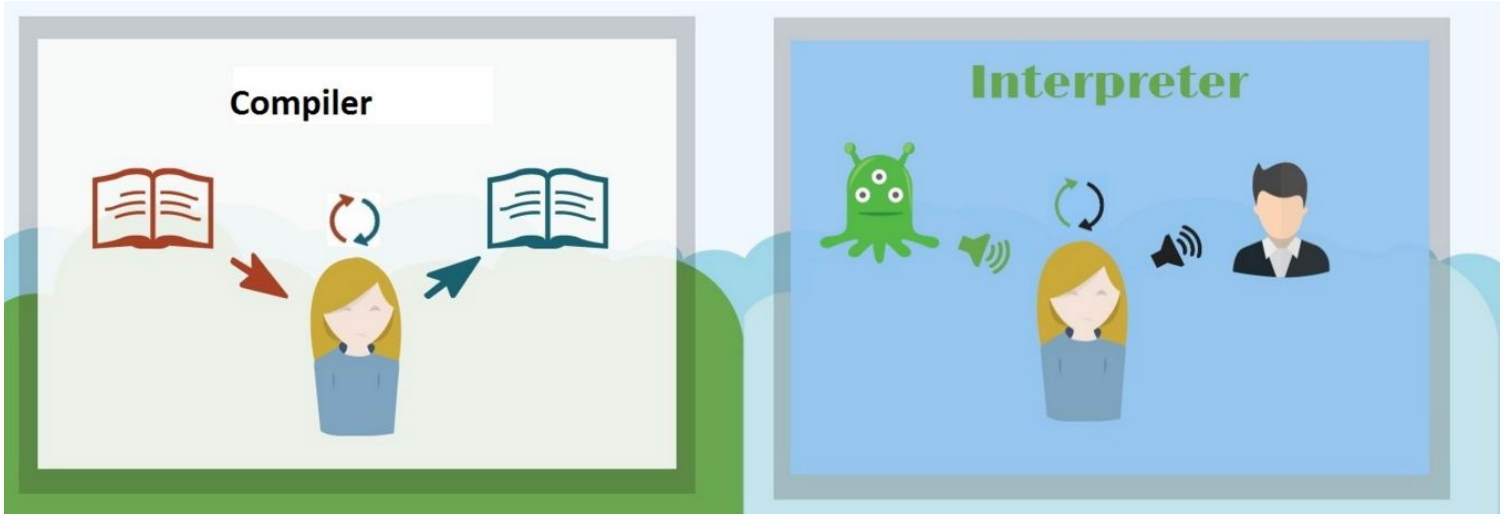
```
class Animal():
    def __init__(self,especie,tipo,nombre):
        self.especie=especie
        self._tipo=tipo
        self.__nombre=nombre

    def setEspecie(self,especie):
        self.especie=especie

    def getEspecie(self):
        return self.especie

    def setTipo(self,tipo):
        self.tipo=tipo

    def getTipo(self):
        return self.tipo



    def reproducir(self):
        print(self._nombre, "se esta reproduciendo")


    def moverse(self):
        if self._tipo=="acuatico":
            print("Yo nado")
        elif self._tipo=="terreste":
            print("Yo camino o me arrastro")
```



- Un programa en python se pasa a bytecode primero, si hay alguna referencia que no esté definida el compilador lo indicará, luego la máquina virtual de ese sistema ejecuta el bytecode con las dependencias ya resueltas directamente en código máquina

En Resumen

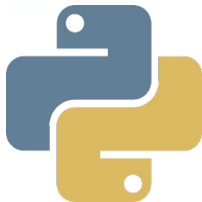
- Un compilador traduce el código completo y si no está bien escrito no puede completar la traducción, en cambio un intérprete sólo traduce lo que entiende y se va de corrido



Desventajas

- No es el lenguaje más rápido (por su naturaleza interpretada)
- No posee las librerías más extensas (no es Matlab)
- No tiene revisión de tipos (desperdicia m memoria)

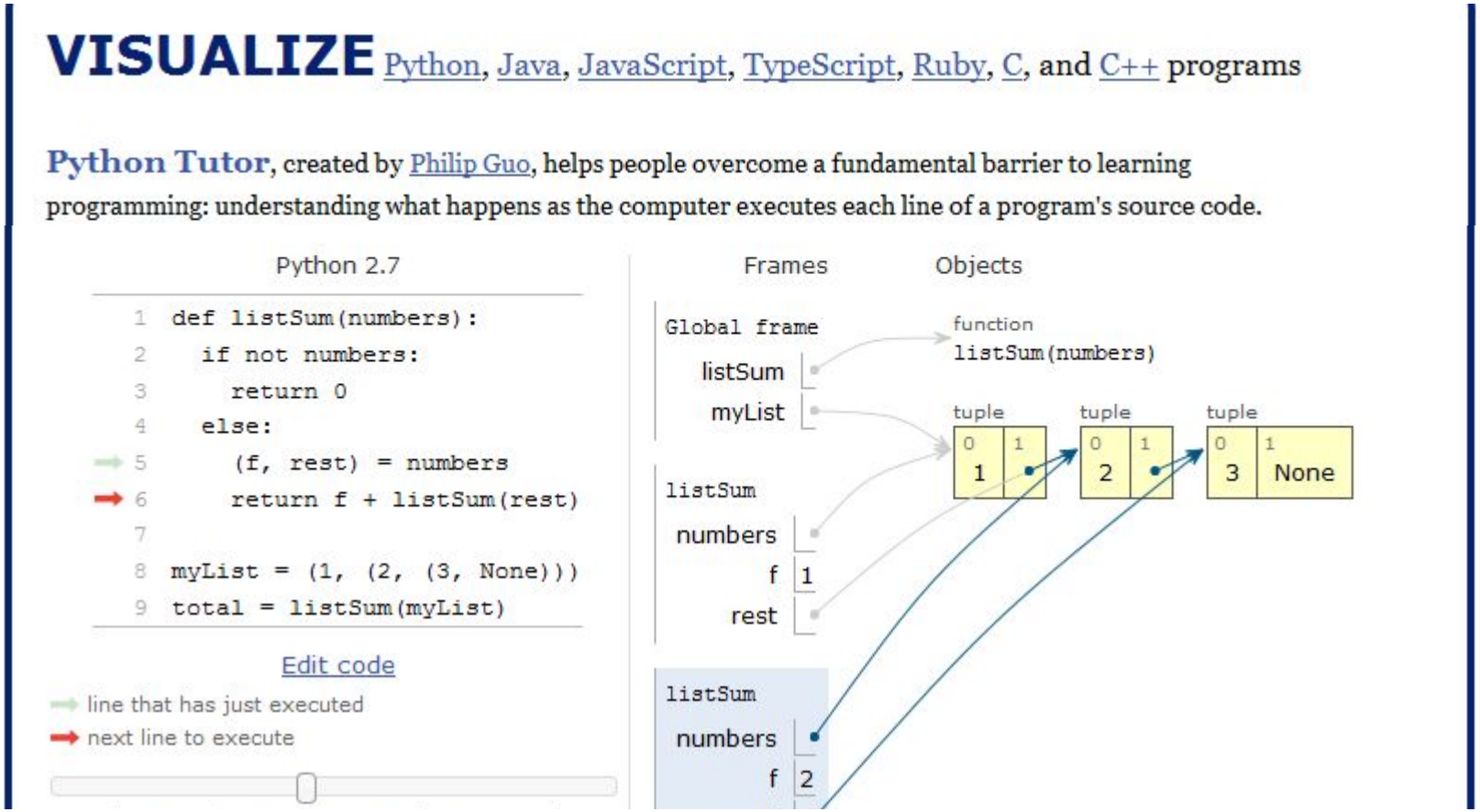
¿Quién usa Python?



¿Quién usa Python?

- http://www.python.org/about/apps
- http://www.python.org/about/success

Visualización de código



http://pythontutor.com/

Enlaces de ayuda

- Ejercicios y un poco de teoría
  https://learnpythonthehardway.org

- Ayuda específica (pero en inglés)
  http://stackoverflow.com/

- Documentación oficial (un poco complicada)
  https://docs.python.org/3/tutorial/index.html

Tarea

- Investigar en Internet (o fuentes verificables) algún módulo o paquete de python que les sea interesante o les gustaría que viéramos en el curso.
- Investigar si tiene una página con documentación y con qué versión de python se ejecuta.
- Algunas carácteristicas de lo que hace o podemos hacer con ese paquete