python

From n00b to h4x0r!

A practical and empirical guide with full examples to become a master from the beginning

(At least try it)

Indice

- Cosas basicas*
- Cosas no tan basicas*
- Cosas intermedias*
- Librerias utiles*
- Cosas avanzadas*
- Links

- *Bastantes gilipolleces recurrentes

¿Quien soy?

- Miguel García, a.k.a. Rock
- Estudiante de la FI, ETSIINF,...
- Miembro de ACM FI
- Amante de la Seguridad & Python
- miguelglafuente@gmail.com
- @BinaryRock
- http://rockneurotiko.github.io

Install it pls!

- Windows & MAC:

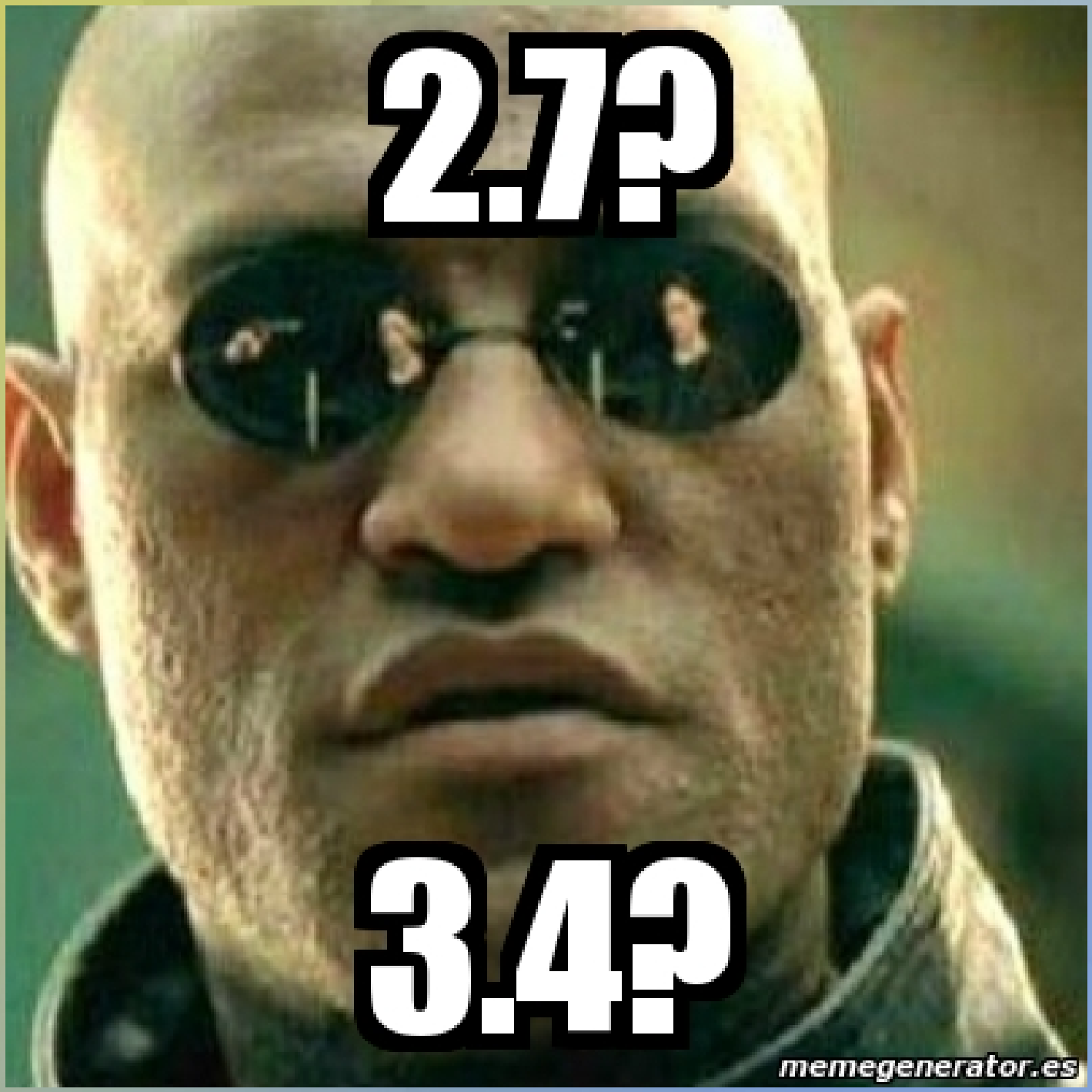
  http://www.python.org/download/releases/2.7.6/

  http://www.python.org/download/releases/3.4.0

- Linux:

  pacman -S python2.7 (Arch)

  [Much Google]


2.7?
3.4?
memegenerator.es

¿Qué es?

- Lenguaje fácil, de alto nivel y multipropósito.
- Filosofía: Legibilidad
- Multiparadigma (POO, imperativa, func.)
- Interpretado! (Con Byte-code .pyc)
- www.python.org/dev/peps/pep-0008/

Python is not Java

- Flat is better than nested.
- Simple is better than complex.
- KISS
- Readability counts.
- XML es basura
  - "Some people, when confronted with a problem, think "I know, I'll use XML." Now they have two problems.”
- Getters y setters son el demonio. No lo hagas.

Java is not Python either.

- Eclipse es un “must use”

Un momento...

Eso no es bueno...

```
this.getPresentation().getSlides().setActualSlide(
    this.getPresentation().getSlides().getActualSlide() + 1 );
```

Guido van Rossum says:

- This emphasis on readability is no accident. As an object-oriented language, Python aims to encourage the creation of reusable code. Even if we all wrote perfect documentation all of the time, code can hardly be considered reusable if it’s not readable. Many of Python’s features, in addition to its use of indentation, conspire to make Python code highly readable

Warnings!

- No hay {}, se usan tabulaciones:

```
Parte1:
    Parte2:
        Parte3:
            EstoEsDeLaParte3
        EstoEsDeLaParte2
    EstoEsDeLaParte1
```

Intérprete!

- En una consola (Meta+R, cmd):
  - python (o python2, python3,...)
  - >>>
- Salir del interprete:
  - Crt+D
  - Exit()
- ipython: shell “mejorado”
  - Aunque en 3.4 el interprete mola mucho!

Beneficios del intérprete

- Testear rapido fragmentos
  - ¿Funcionará 'x'?... Abrir el interprete y probarlo
  - ¿Cómo se hacía 'y'?... “”
- ¡Calculadora! =D
- import this

Sin intérprete

- Codigo en un fichero (.py preferiblemente)

- Desde terminal:
- python <nombreDelArchivo> [parametros]

- Primera linea: #!/usr/bin/python (o variante)
- Chmod +x <nombreDelArchivo>
- ./<nombreDelArchivo>

Comentarios

- Linea: #
  - #Esto es un comentario

- Bloque: “”” blah blah blah “””
  - “””

    Esto en un

    comentario en

    bloque

    “””

Variables y Constantes

- Sin tipado escrito (tipado dinámico)
  - >>>>nombre_var = “hola!”
  - >>>>nombre_var = 8
  - >>>>PI = 3.1415
- Variables minusculas y _ de separacion
- Constantes mayusculas

¡¡EJEMPLOS!!

Tipos

- Numero
  - Entero: a = 2; b = 0o10 [octal] ;c = 0x23 [Hex]
  - Long: a = 45696678615198764 3L (2.x)
  - Real: d = 3.34
  - Complejo: e = (4.5 + 3j)
- String y Unicode: e = “Hola”; e=u'Hola'
- Boolean: True/False
- Listas/Tuplas/Diccionarios
- Objetos

Operadores

- Suma/resta: + - (3+2-1)
- Multiplicacion/Division: * / (4*2/3)
- Exponente: ** (2**3)
- Division entera: // (5.0 // 2) [=2]
- Modulo: % (4%2)

Tuplas y listas

- Tupla: almacen de datos, pero inmutable (parecido a los arrays, ...)
- tupla1 = (“a”, 2, 3.4)
- print(tupla1[0])
- Lista: almacen de datos... mutable
- lista1 = [“a”,2,3.4]
- print(lista1[0])

¡¡¡EJEMPLOS!!!

Resumen listas + tuplas

- Acceder posicion: lista[n]
- Porcion: lista[n:n2]
- Sumar listas: lista1 + lista2
- Contenido de lista “n” veces: lista * n
- Añadir a lista: lista.append(elem)
- Sacar de la lista el ultimo: lista.pop()
- Sacar elemento “n”: lista.pop(n)

Diccionarios

- Mutables
- Par de elementos: clave → valor
- La clave puede ser: String, Int, Float, Tupla (Aunque se suele usar String o Int)
- dicc = {“clave1” : “valor1”, “clave2” : “valor2”}
- dicc[“clave1”]

Resumen Diccs

- Recuperar valor clave n: dicc[n]
- Añadir par: dicc[“claveNoExiste”] = n
- Eliminar par: del dicc[“claveExiste”]
- Tip (construccion dinámica):

  dicc = dict([(“clave1”,”valor1”), (“clave2”,”valor2”)])

Trucos de asignacion

- Asignacion multiple:

  a, b, c = “hola”, 2, [1,2,3]
- Asignacion desde tupla:

  a, b = (“hola”, 2)
- Asignacion Desde lista:

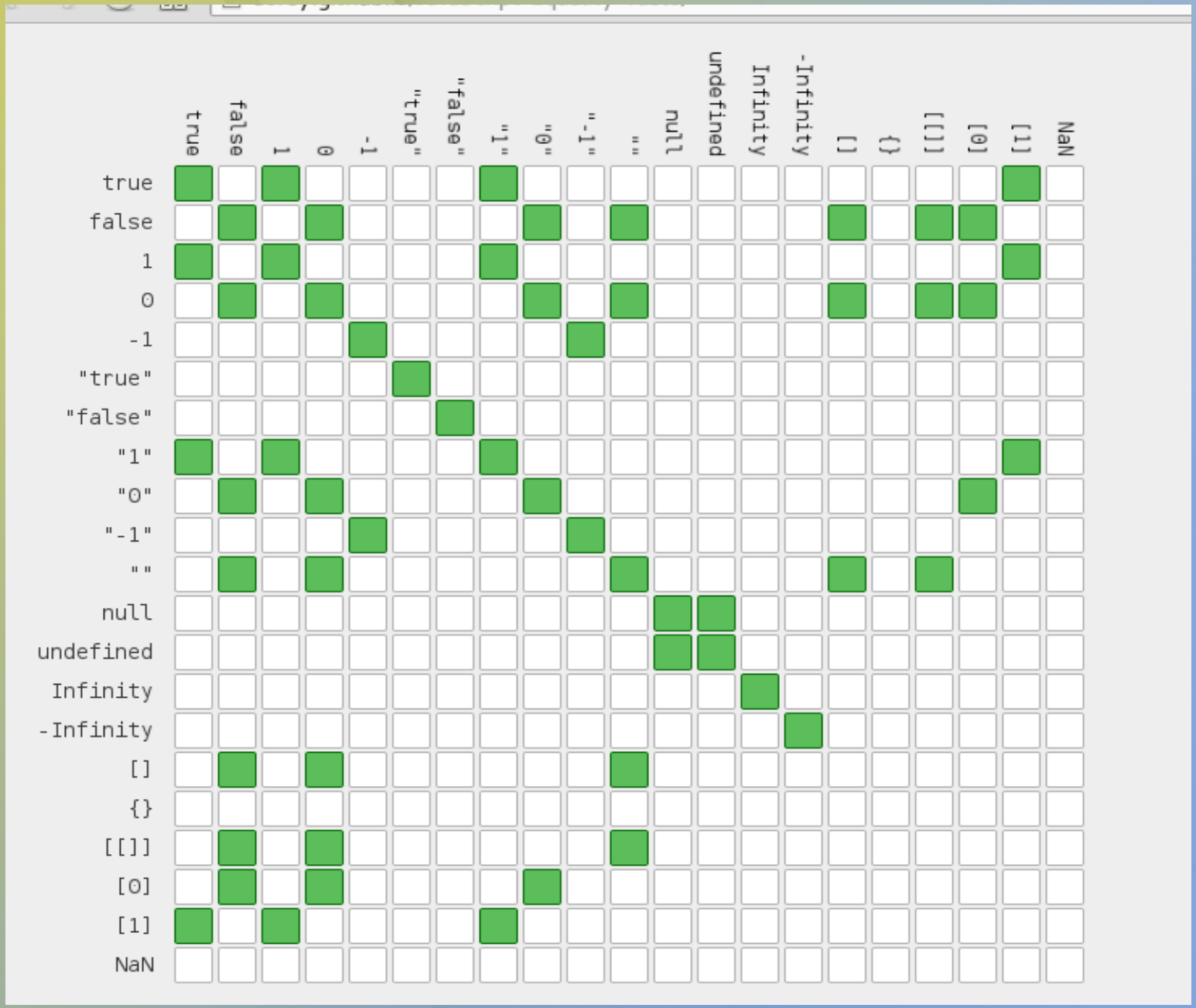
  a, b = [“hola”, 2]
- Desde str (from reddit):

  a, b = “ab”

Operaciones relacionales (pa' comparar vamos)

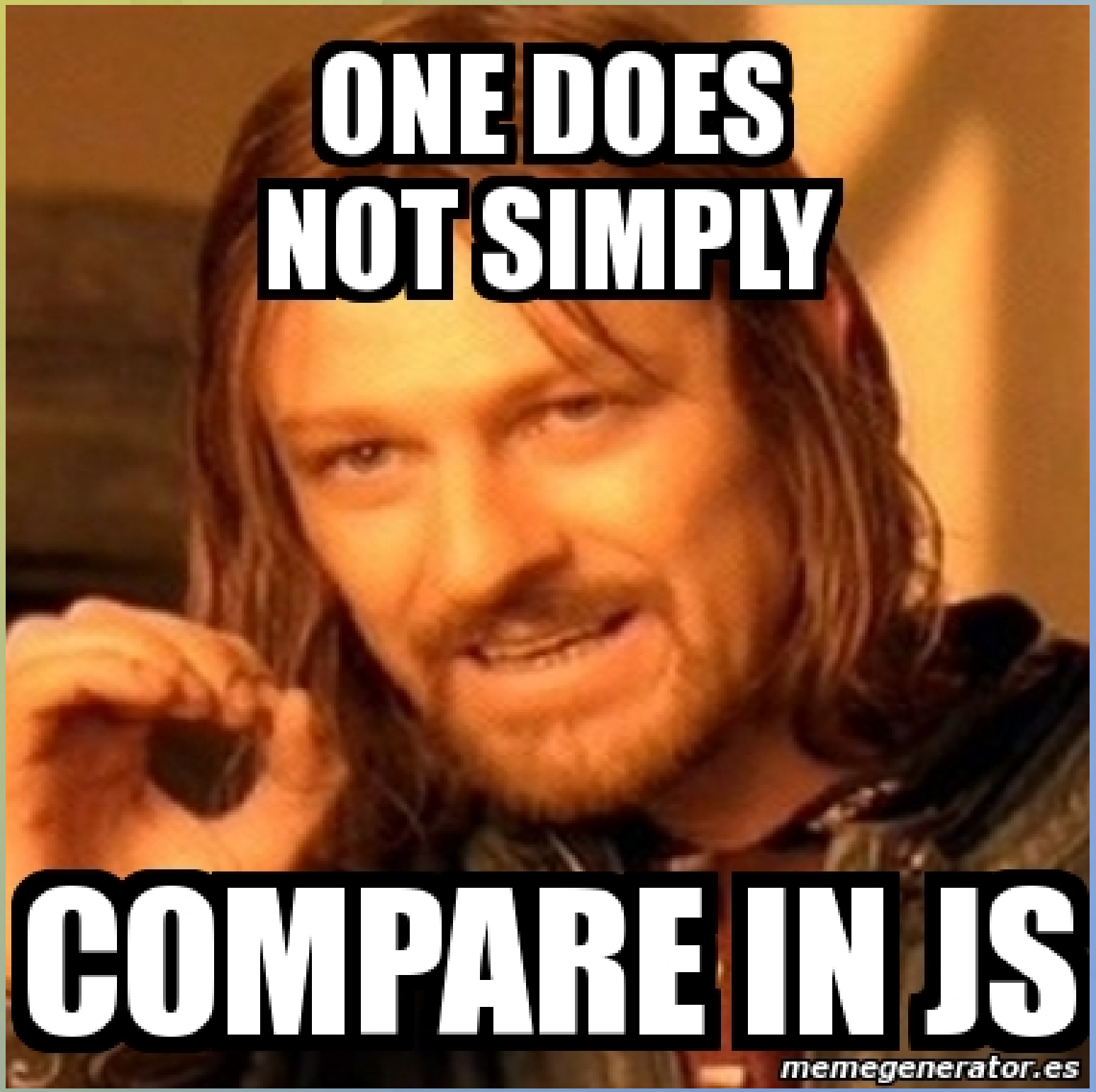
- Los típicos:
  - ==
  - !=
  - >
  - <
  - >=
  - <=
- El resultado es un booleano (True o False)

Comparacion == en JS ;-)



Y en Python! =D

	True	False	1	0	-1	""	"True"	"False"	"1"	"0"	"-1"	None	[]	{}	[[]]	[0]	[1]
True	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
False	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
"True"	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
"False"	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
"1"	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
"0"	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
"-1"	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
""	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
None	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
[]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
{}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
[[]]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
[0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1


ONE DOES
NOT SIMPLY
COMPARE IN JS
memegenerator.es

Operadores lógicos

- AND = and:
  - True and False
- OR = or:
  - True or True
- XOR = ^ (si eres osado !=):
  - True ^ True ;; False != True
- El resultado es un booleano (tx Mr.Obvius)

Python drunk's game!

- En Python 2.x True y False son globales
- Así que se pueden cambiar =D

  >>True = False

  >>False = (1==1)

- Uno hace una operación larga booleana:

  (True and False ^ False) ^ (True and True or False or True and False)

- “x” tiempo para decir el resultado, si esta mal, bebe!
- Si hay más gente, se sigue la expresión, el que acierte hace una nueva.

Estructuras de flujo condicionales! (Wiiiiii!)

(Notese la indentacion)

- if <<condicion1>>:

    hacer cuando cond1

  elif <<cond2>>:

    hacer cuando con2 y no cond1

  else:

    hacer en otros casos

EJEMPLOS!

Estructuras de control iterativas

- while <condicion>:

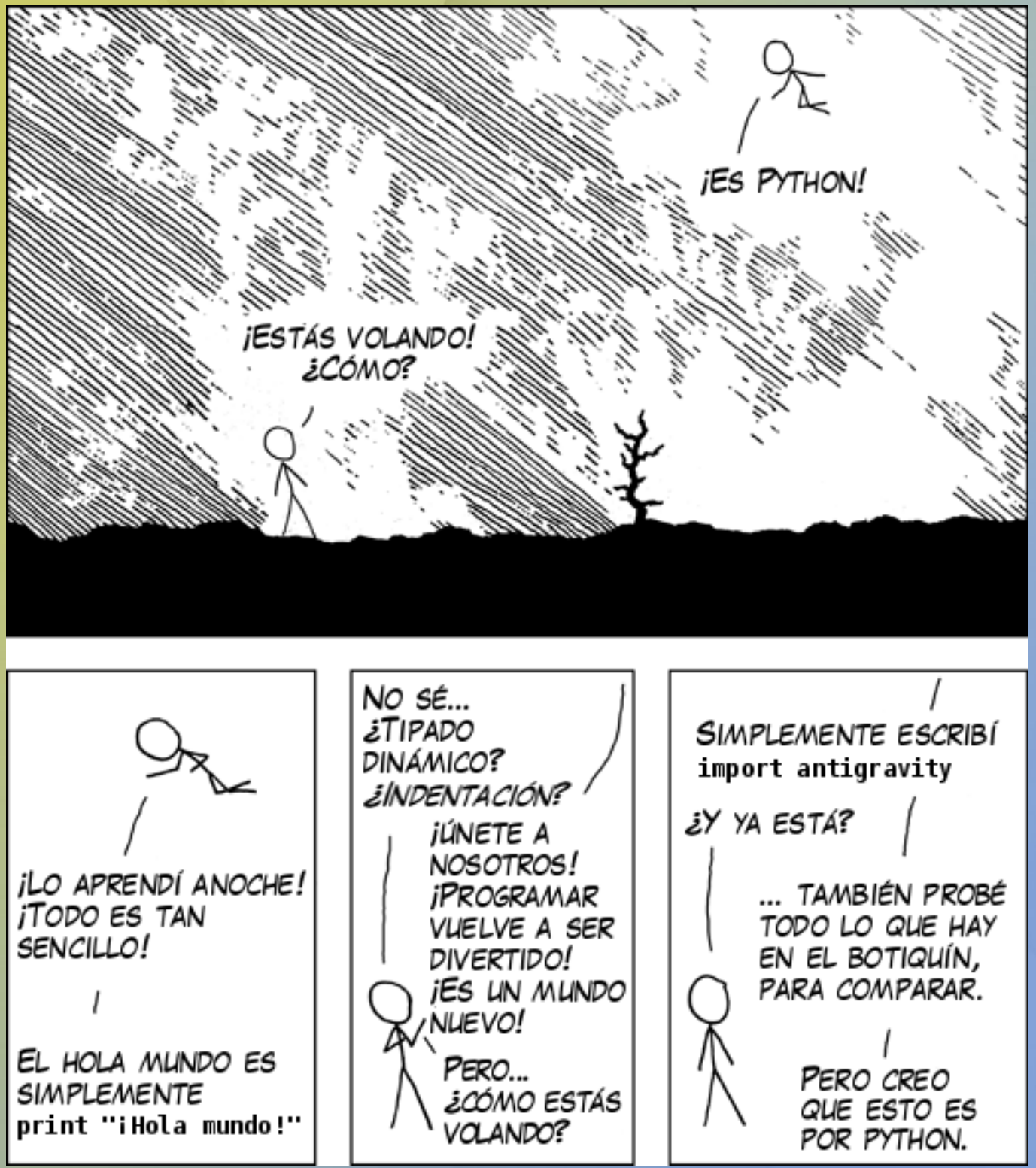
  #que hacer

- for <nombre_var> in <estructura>:

  #que hacer

- Recorre listas y tuplas, diccionarios recorre las keys

EJEMPLOS!


¡ES PYTHON!
¡ESTÁS VOLANDO! ¿CÓMO?
¡LO APRENDÍ ANOCHE! ¡TODO ES TAN SENCILLO!
EL HOLA MUNDO ES SIMPLEMENTE
print "¡Hola mundo!"
NO SÉ... ¿TIPADO DINÁMICO? ¿INDENTACIÓN?
¡ÚNETE A NOSOTROS! ¡PROGRAMAR VUELVE A SER DIVERTIDO! ¡ES UN MUNDO NUEVO!
PERO... ¿CÓMO ESTÁS VOLANDO?
SIMPLEMENTE ESCRIBÍ
import antigravity
¿Y YA ESTÁ?
... TAMBIÉN PROBÉ TODO LO QUE HAY EN EL BOTIQUÍN, PARA COMPARAR.
PERO CREO QUE ESTO ES POR PYTHON.

¡Modulos y paquetes!

- 1ª regla: KISS!
- 2ª regla: No re-inventes, tardarás más y casi seguro será peor.
- Python tiene librerías para casi todo lo imaginable, merece la pena buscar un poco en tito Google antes.

Hacer tus propios paquetes

- Organizar proyectos en subcarpetas
- Eclipse(netbeans,...) te lo hace solo...

programPrinc.py

subcarp/

__init__.py

doAll.py

subcarp2/

__init__.py

doNothing.py

¿Factor común de las subcarpetas?

Usar modulos/paquetes

- Varios modos:
  - **import** modulo ← Acceso: nombre.loquesea
  - **from** modulo **import** algo, algo2, algo3

    Se usan con el nombre tal cual (algo, algo2, ...)
  - **import** modulo.submodulo.submodulo2 [...]

    Se puede importar solo un submodulo
  - **import** modulo **as** m ← Acceso: m.loquesea
  - **from** modulo **import** algo **as** a, algo2 **as** a2
  - **from** modulo **import** * ← ¡¡Intentar evitar!!

EJEMPLOS!

Funciones

- def nombre (<<arg1, arg2, ...>>):

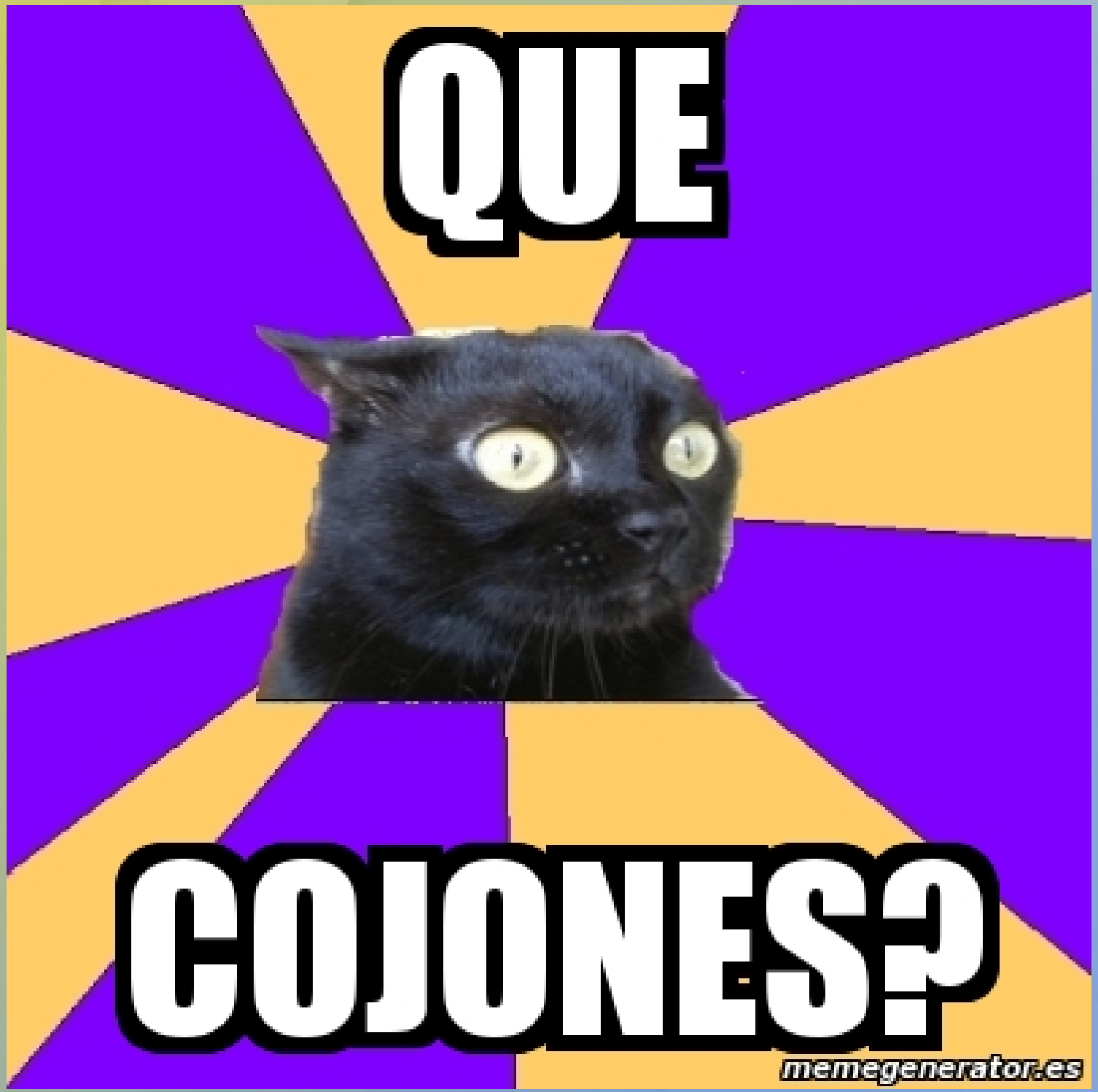
  #Cosas para hacer

  #return optativo

- Mejor codigo y ejemplos =D

Argumentos por defecto =D

```
def funcion(arg = []):
    arg.append(0)
    return arg
print(funcion([]))
print(funcion([]))
print(funcion())
print(funcion())
print(funcion())
...
```


QUE
COJONES?
memegenerator.es

Moar funcs!

- def func(arg_fijo, *arg_variable, **k_arg)
- Los variables se recorren como lista
- Los variables “k” (keyword) se recorren como diccionario

- Desenpaquetando parametros en llamada... (Wut?)
- locals(), globals()... Llamando a funciones de retorno de forma dinámica

EJEMPLOS!