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

Disclaimer

- Basado en mi experiencia
- No soy diseñador
- Preguntas (Excepto filosóficas)

Yago no tiene permitidas las preguntas.

Install it pls!

- Windows & MAC:

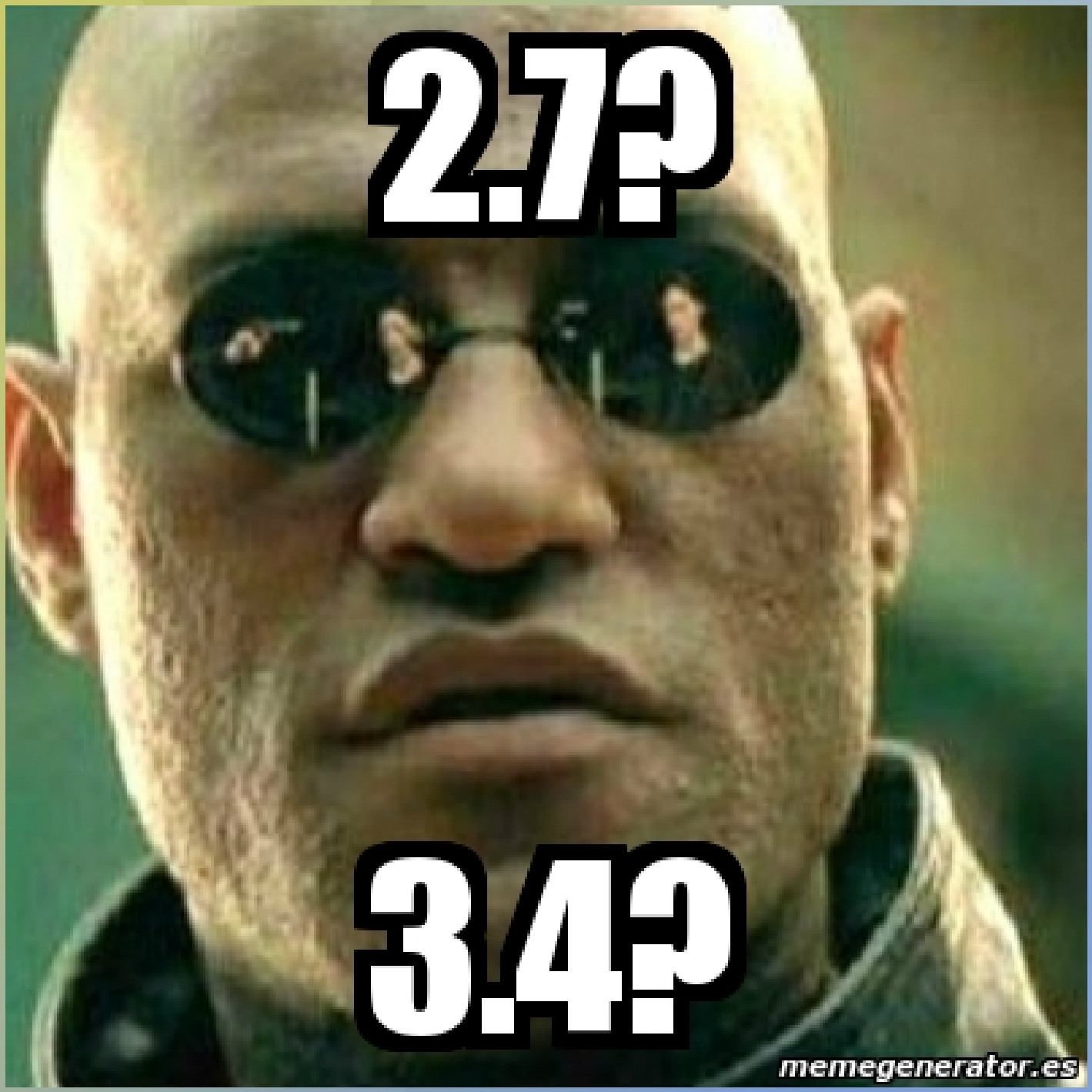
  http://www.python.org/download/releases/2.7.6/

  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:

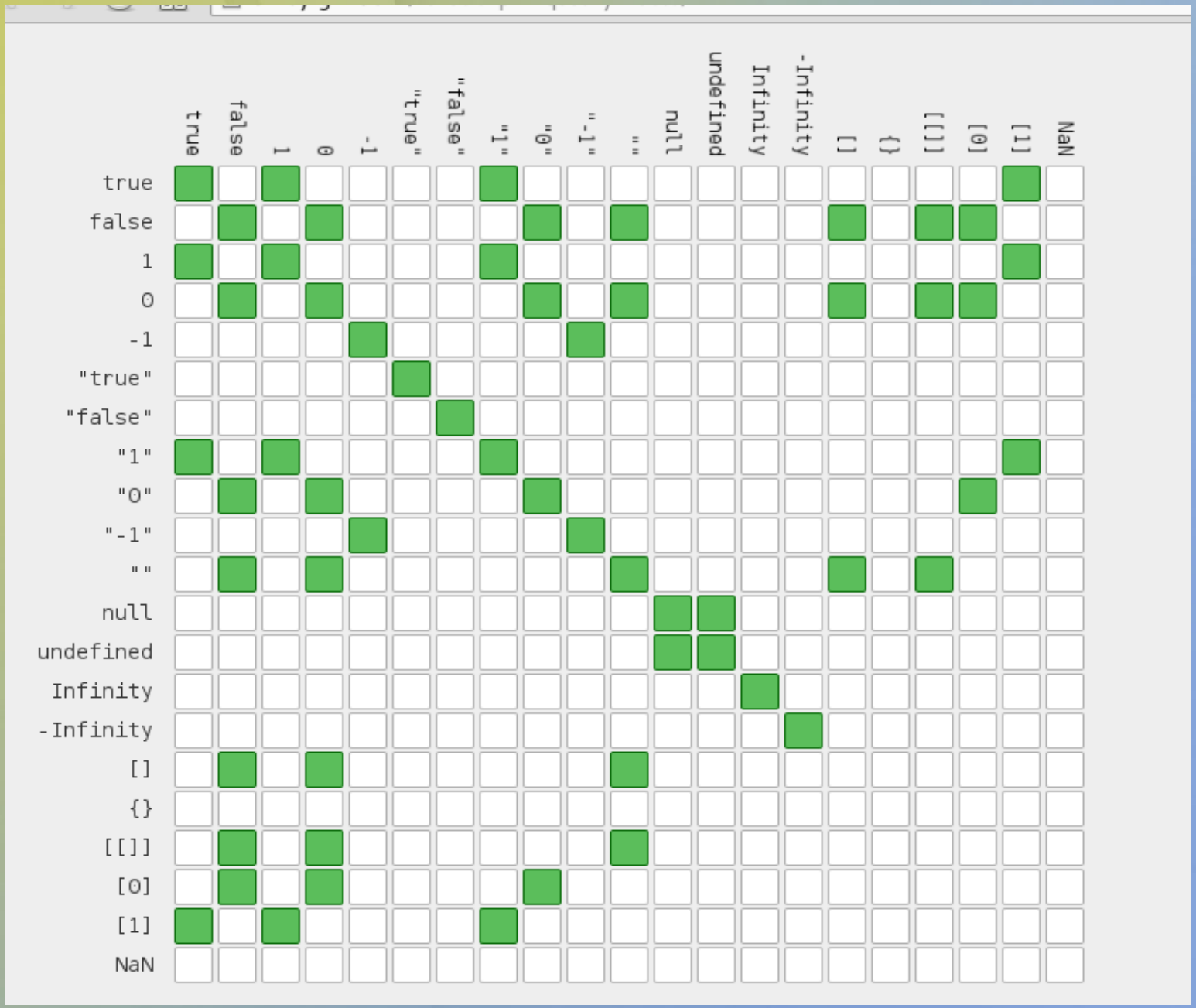
  a, b = [“hola”, 2]
- Desde str (from reddit):

  a, b = “ab”

Operaciones relacionales (pa' comparar vamos)

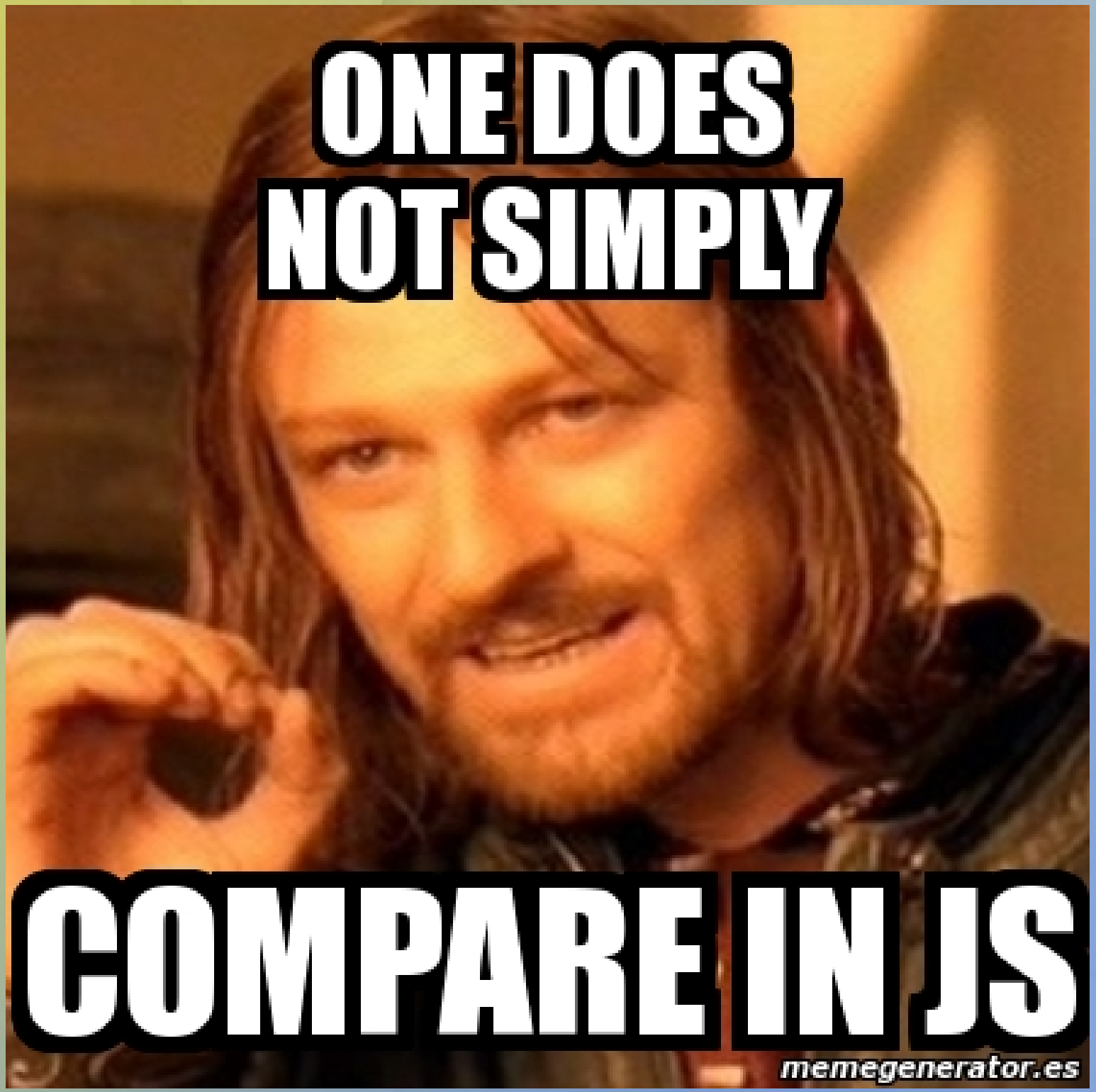
- Los típicos:
  - ==
  - !=
  - >
  - <
  - >=
  - <=
- El resultado es un booleano (True o False)

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>:

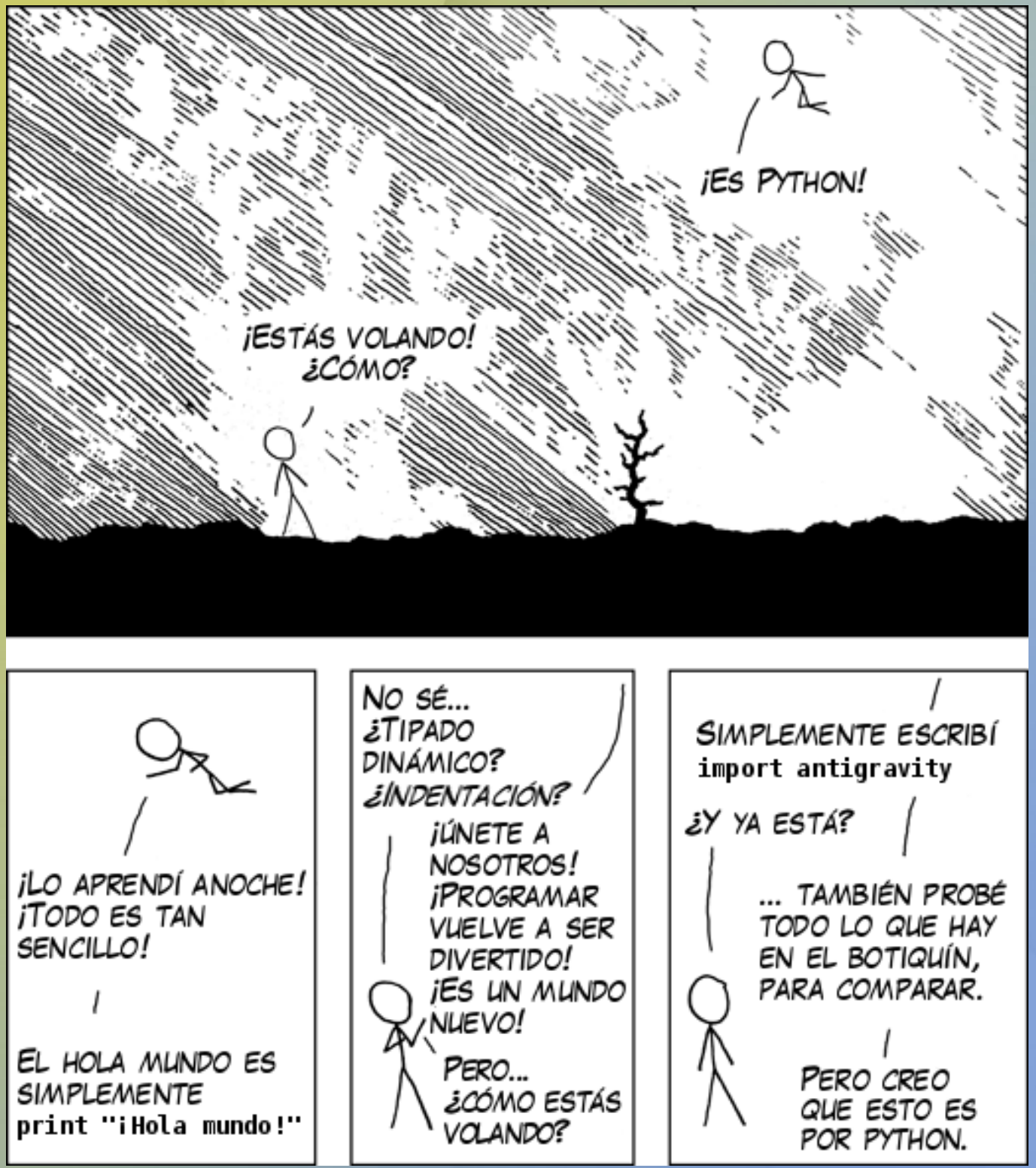
  #que hacer

- 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, ...>>):

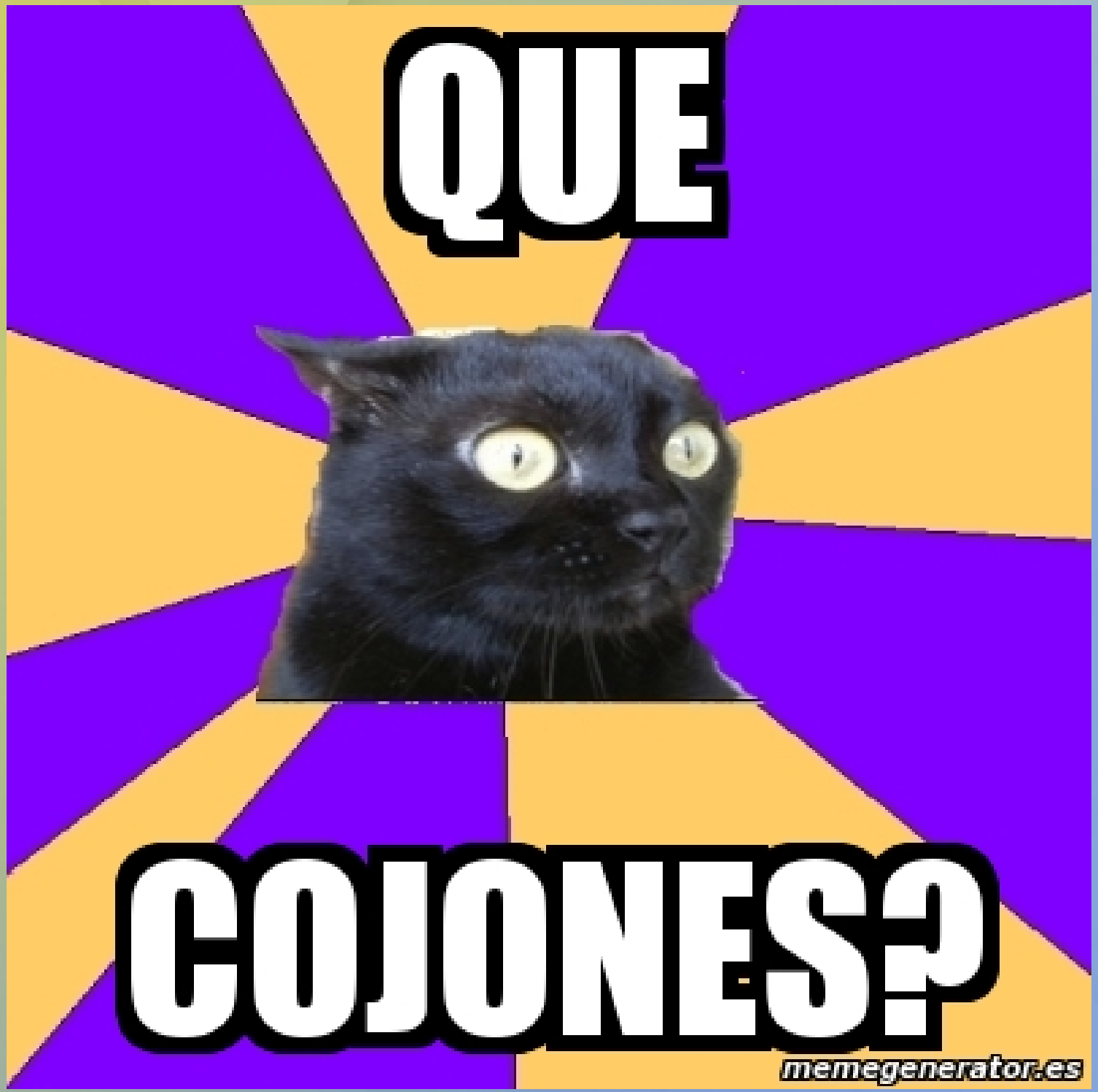
  #Cosas para hacer

  #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

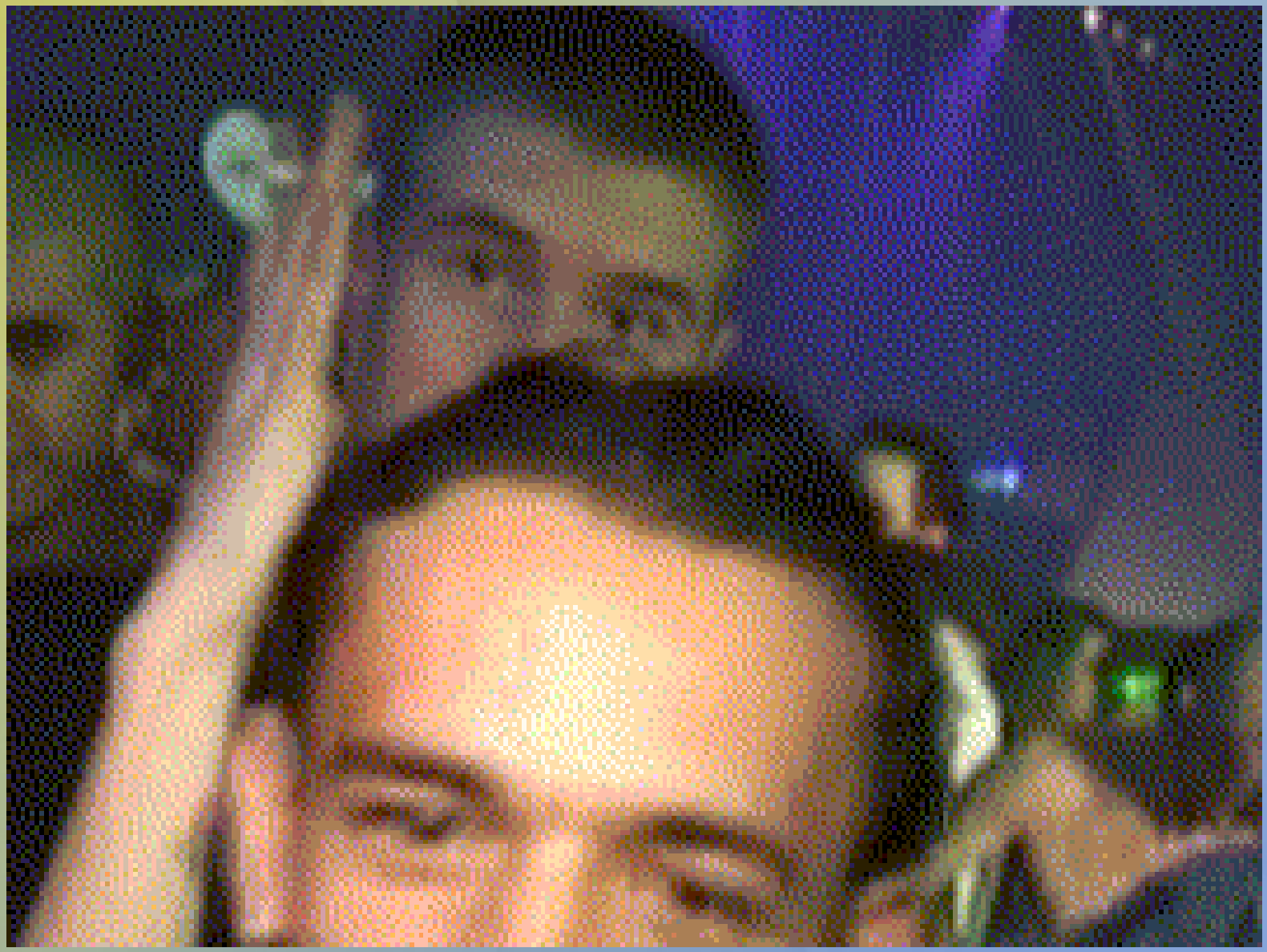
Moar funcs!

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

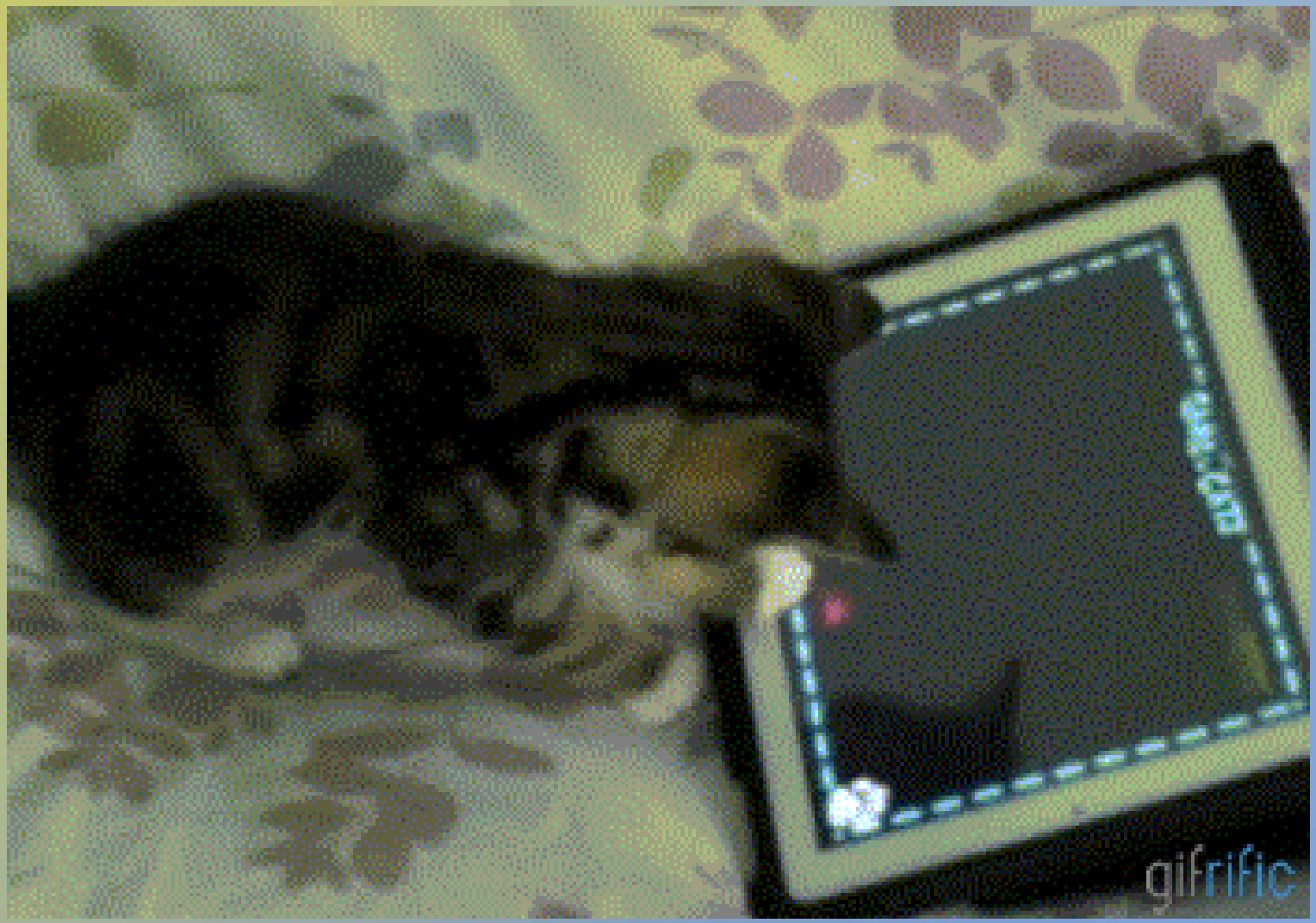
- Desenpaquetando parametros en llamada... (Wut?)
- locals(), globals()... Llamando a funciones de retorno de forma dinámica

EJEMPLOS!

FIN DIA 1 ^^



DIA 2 =D



Antes de empezar

- Más lento pls!
- 4 espacios para indentacion

  (editores ayudan)
- Recomendacion: picar mucho codigo para aprender a programar mejor (cualquier lenguaje)
- Slide de arrays/listas con salto:

  lista[::-1]
- Parar si hace falta
- http://pili.la/githubrock

Objetos y esas cosas raras

- En Python TODO es un objeto
- POO muy beneficiada, pero no necesaria
- Clase: (el modelo de objeto)

```
class NombreClase:
    #cosas
```

- Propiedades: (de instancia)

```
class NombreClase:
    prop1 = 1
    prop2 = 2
```

- Métodos: (Funciones de una clase)

```
class NombeClase:
    def metodo(self):
        #Cosas para hacer
```

- Objeto: (La instanciacion de la clase)

```
class NombreClase:
    a=1
var = NombreClase()
print var.a
var.a = “=D”
print var.a
```

- Herencia: (Soporta herencia multiple)

```
class Clase1(object):
    valor=”a”
class Clase2(Clase1):
    valor2=”b”
a = Clase2()
print a.valor
print a.valor2
```

- object es la clase basica, es recomendable hacer que si no hereda de nada, herede de esta. (en 2.x)

2 more things

- El metodo __init__(self [, args]): es el que se ejecuta al iniciar la clase (“constructor”): inicializar
- self? → self es la convencion para referirse a la instancia.
  - Propiedades accesibles: self.nombre
  - Todos los metodos self primer argumento
- En Python no hay encapsulacion (propiedades privadas) → Convención: _nombre == “Tu sabras lo que haces si tocas esto”

EJEMPLOS!

Strings y sus cosicas(1)

- capitalize() → Primera letra mayusculas
- lower() → Todo en minusculas
- upper() → Todo en mayusculas
- swapcase() → Cambia may/min
- title() → Primera letra cada palabra May.
- center(n,[relleno]) → centra y rellena a los lados
- ljust, rjust, igual que center (izq. y der.)

Strings y sus cosicas(2)

- count(subcadena) → apariciones de sub
- find(subcadena) → busca y devuelve la posicion de inicio (si no, -1)
- startswith, endswith (subcadena)
- isalpha, isdigit, isalnum, isnumeric ()
- islower, isupper ()
- replace(cad_buscar,cad_reemplazar)
- strip([caracter]) → Quita “caracter”, espacio por defecto.

Strings y sus cosicas(3)

- Tenemos un string para formatear:

  a = “Hola {0} que tal tu {1}?”
- a.format(“Miguel”, “perro”)

  >>>”Hola Miguel que tal tu perro?”
- Tambien con claves:

  a=”Hola {nombre} que tal tu {cosa}?”
- a.format(cosa=”perro”, nombre=”Miguel”)

Strings y sus cosicas(y 4 :)

- split([separador]) → Devuelve una lista de la cadena separada por la subcadena

  a = “Hola que tal”

  a.split(“ “) → [“Hola”,”que”,”tal”]
- splitlines() → Devuelve una lista con las lineas

  a = “Ey\nQue tal\nman?”

  a.splitlines() → [“Ey”,”Que tal”,”man?”]
- len(cadena) → Tamaño de la cadena

Ejercicios?

Crear un módulo para validación de nombres de usuarios.

Dicho módulo deberá cumplir con los siguientes criterios de aceptación

- El nombre de usuario debe contener un mínimo de 6 caracteres y un máximo de 12
- El nombre de usuario debe ser alfanumérico
- Nombre de usuario con menos de 6 caracteres, imprime el mensaje “El nombre de usuario debe contener al menos 6 caracteres”
- Nombre de usuario con más de 12 caracteres, imprime el mensaje “El nombre de usuario no puede contener más de 12 caracteres”
- Nombre de usuario con caracteres distintos a los alfanuméricos, imprime el mensaje “El nombre de usuario puede contener solo letras y números”
- Nombre de usuario válido, retorna True, si no, False

Ejercicios?

Crear un módulo para validación de contraseñas.

Dicho módulo deberá cumplir con los siguientes criterios de aceptación

- La contraseña debe contener un mínimo de 8 caracteres
- Una contraseña debe contener letras minúsculas, mayúsculas, números y al menos 1 carácter no alfanumérico
- La contraseña no puede contener espacios en blanco
- Contraseña válida, retorna True
- Contraseña no válida, imprime el mensaje “La contraseña elegida no es segura” y retorna False

Listas, listas y moar listas(1)

- append(elem) → añade al final el elem
- extend(lista) → añade la lista al final
- insert(n,elem) →añade el elem en la pos n
- pop() → Saca el ultimo elemento
- pop(n) → Saca el elemento en pos n

  Notese que si se usa append + pop() = Pila

  y append + pop(0) = Cola
- remove(elem) → Elimina el elemento

Listas, listas y moar listas(2)

- reverse() → Da la vuelta
- sort() → ordena
- sort(reverse=True) → Ordena al reves
- count(elem)
- index(elem[, pos_in, pos_fin])

- tuple(lista) y list(tupla)
- max(lista/tupla) y min(lista/tupla)

Comprehension de listas

- En lugar de especificar elementos, decir la regla para ello.
- Normal:

```
lista=[]
for i in range(10):
    lista.append(i**2)
```

- Comprehension:

```
lista = [x**2 for x in range(10)]
```

- lista = [(x,y) for x in [1,2,3] for y in [3,1,4] if x != y]
- array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

```
list = [num for elem in array for num in elem]
```

Ejemplo práctico

- Yagomix!
- Necesidad:
  - Listar ficheros de un path, no ocultos, .mp3 y que no estuviesen ya reproducidos.

```
def getSongs(path, filesPlayed):
    return [f for f in listdir(path)
    if isfile(join(path, f))
    and f[0] != "."
    and f[-4:] == ".mp3"
    and f not in filesPlayed]
```

Some about dics

- update(dic2) → Une diccionarios
- get(clave) → Devuelve el valor de la clave
  - Si no esta, devuelve None
- keys(),values(),items() devuelven “views” en Python 3 (en 2.x listas)
- Recorrer todo:

```
for i, j in diccionario.items():
    print(i,j)
```

Comprehension sic!

- Comprehension de diccionarios:

  dicc = {i:str(i) for i in range(5)}

  >>> {0: '0', 1: '1', 2: '2', 3: '3', 4: '4'}

- Set comprehension

  lista = [1, 2, 2, 3, 4, 5, 5, 7, 8]

  mi_set = set(lista)

  <<que en el fondo es>>

  mi_set = {x for x in lista}

  >>>{1, 2, 3, 4, 5, 7, 8}

Data pls!!! (1) (Python 2.x)

- input([mensaje])
- input → Objetos existentes, normalmente usado para numeros (int,float,complex) puede ser cualquiera, incluso...

  a = “hola”

  b = input() → Cuando pida, pones a

  print(b) → ¿Que deberia pasar?

Data pls!!! (2) (Python 2.x)

- raw_input([mensaje])
- Se va a “tragar” todo → como un String
- Coger numeros y que un str no pete la ejecucion
- a = raw_input(“Hola!: “)

  print a
- En el mensaje que da bien el “: “

Data pls!!! (3) (Python 3.4)

input() para todo! =D

Error? That's a feature!

- try, except y finally

```
try:
    #Acciones que pueden dar error
except:
    #Que hacer si salta la excepcion
else:
    #Que hacer si NO salta la excepcion
finally:
    #Que hacer despues de todo, vaya bien o mal
```

Finally means finally!

```
Def test():
    try:
        print(“Dentro de try”)
        return “Saliendo desde try”
    finally:
        print(“Dentro de finally”)
        return “Saliendo desde finally”
print(test())
```

Excepciones concretas

- except puede (y debe) llevar el tipo de error a capturar

```
try:
    #blablabla
except TypeError as e:
    #Que hacer en TypeError (e contiene la traza)
except IOError as e:
    #Que hacer en error entrada/salida
except:
    #Excepcion general
```

Raise it! (Python 2.x)

- Si queremos levantar el error:
- raise [excepcion [, mensaje de error(str)]]

```
except TypeError:
    raise TypeError, “Error de tipos!”
except:
    raise Exception, “Error general”
```

Raise it! (Python 3.4)

- Si queremos levantar el error:
- raise excepcion(mensaje)

```
except TypeError:
    raise TypeError(“Error de tipos!”)
except:
    raise Exception(“Error general”)
```

Funciones lambda

- Funciones anonimas (en tiempo de ejecucion)

- def incremento(n): return lambda x: x + n

  a = incremento(5)

  print(a(50)) >>> 55

  print(incremento(2)(68)) >>> 70

Usos en Python – filter (2.x)

- filter(): filtra una lista/tupla con una regla
- a=range(50)

- filter(lambda x: x%2==0, a) ← nº pares

- for i in range(2,8): [Nº primos]

    a = filter(lambda x: x == i or x%i, a)

Usos en Python – map (2.x)

- map(func, lista): aplica funcion a lista → lista
- frase = “Vamos a testear map con lambda”

```
palabras = frase.split()
```

```
>>>['Vamos', 'a', 'testear', 'map', 'con', 'lambda']
```

```
longitudes = map(lambda palabra: len(palabra), palabras)
```

```
>>>[5, 1, 7, 3, 3, 6]
```

...

- En python 3 las funciones map, filter [y zip] han cambiado.
- Listas nunca mas, ahora iteradores!

  [De los cuales hablaré luego =D]
- Para conseguir la lista, lo mismo que antes pero haciendo list(<expresion>)
- Ej:

  longitudes = list(map(lambda palabra: len(palabra), palabras))

Usos reales:

- Buscar en google:
- <nombre proyecto grande> source code lambda

- P.e: django source code lambda
- Muchos usos reales y utiles!

2 en 1 – An Unix tip & lambda use!

from subprocess import getoutput

(Py2.x): from commands import getoutput

mount = getoutput(“mount -v”)

lines = mount.splitlines()

p = list(map(lambda line: line.split()[2], lines))

>>>['/proc', '/sys', '/dev', '/run', '/', '/sys/kernel/security', '/dev/shm', '/dev/pts', '/sys/fs/cgroup', (...)]

- =DD Puntos de montaje en lista!
- One more example!

Give me fuel, give me file!

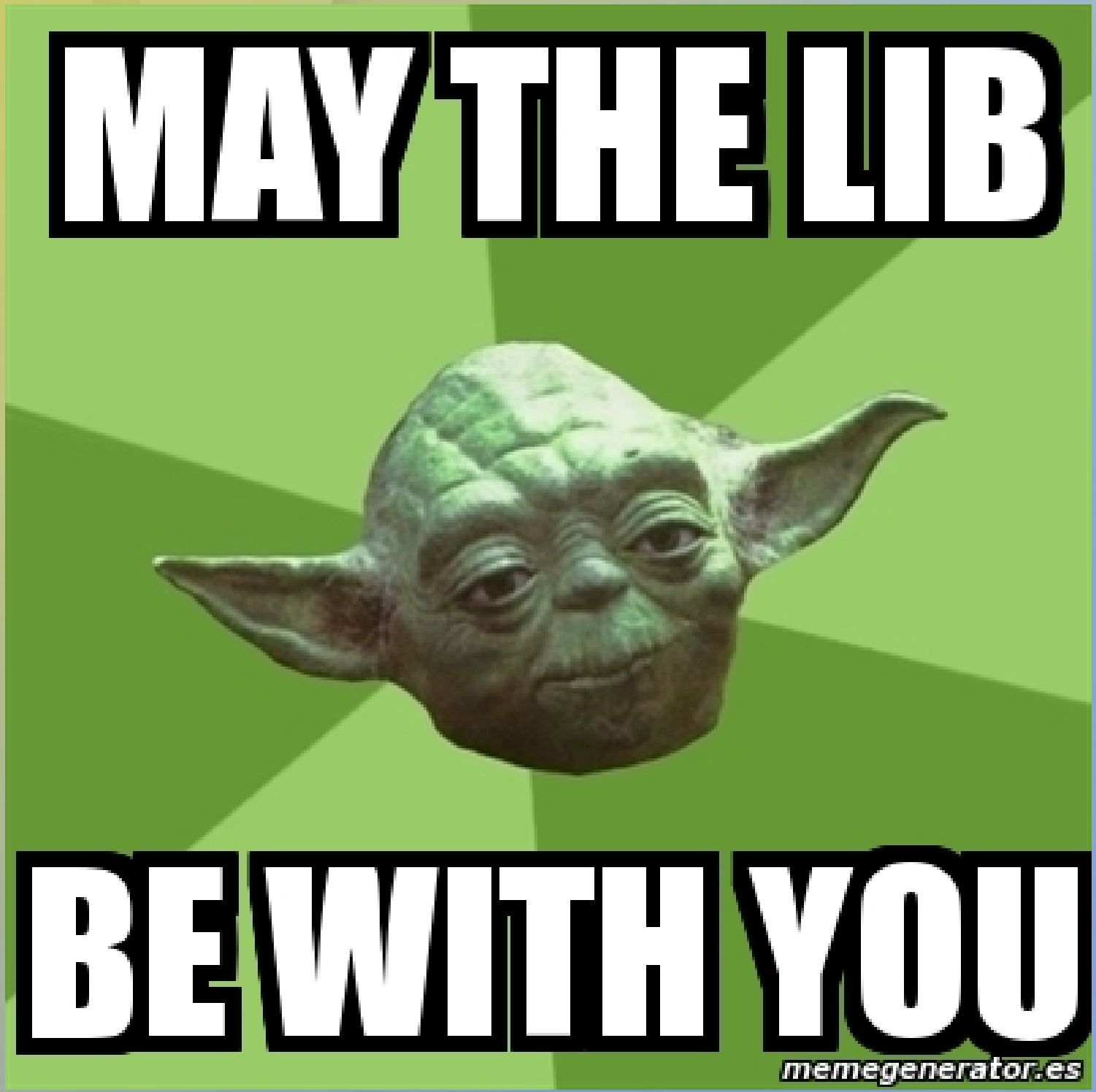
- Abrir un archivo: f = open(path, modo)
- Path puede ser relativo o absoluto
- Modo: [”r”, ”w”, ”a”, “r+”, “rb”, “wb”,...]
- Al dejar de usarlo f.close()
- Para prevenir los olvidos, mucho mejor:

```
with open(path,modo) as f:
    #Durante todo el with, f sera el archivo
    #y al acabar lo cerrara auto-magicamente!
```

OK, y ahora que?

- read() → Devuelve TODO el archivo
- read(num) → Devuelve “num” Bytes
- readline() → Devuelve una linea, la siguiente vez que se invoque, la siguiente
- readlines() → Lista con las lineas
- for line in f: → Dentro del bucle, cada it. es una linea :) #Mas usado eveh!
- seek(num) → va a la pos. num (en Bytes)
- write(algo) → escribe (Depende del modo)

- write en modo “w” escribe desde inicio.
  En modo “a” desde el final
- seek(n) → mueve el puntero al Byte n
- tell() → devuelve el byte donde esta el *
- readable() → ¿Te puedo leer?
- writable() → ¿Te puedo escribir?
- seekable() → ¿Puedo moverme por ahi?


MAY THE LIB
BE WITH YOU
memegenerator.es

CSV - lectura

import csv ;-)

EJEMPLO!!

- csv.reader(file, delimiter=”;”,...)
- csv.DictReader(file, delimiter=”;”,...)
- Toma un fichero abierto para lectura

  (“rb”, “rU” o “r” depende del caso)
- Muchas opciones, delimiter la mas usada (por defecto usa la coma)
- Devuelven un iterable (mas adelante hablaremos de esto)
- Reader → iterable de lista
- DicReader → iterable de diccionario

CSV – escritura (==)

import csv

EJEMPLO!!

- csv.writer(file, delimiter=”;”,...)
- csv.DictReader(file, fieldnames=fields,...)
- Toma un fichero abierto para esc. (“w”)
- Devuelven una clase con metodos para escr.
- Writer:

  writer.writerow(row) ← lista con cada elem
- DictWriter:

  dictwrit.writeheader() ← cabecera

  dictwrit.writerow(row) ← diccionario

CSV – Algo mas...

- Tanto writer como DictWriter tienen un metodo llamado writerows() que toma un iterable con los datos (en lugar de usar un for)
- Se puede crear Dialects para versiones raras/antiguas de csv [No idea, but exist!]

ConfigParser

- 2.x: ConfigParser, 3.x: configparser
- Archivos de configuracion de forma facil!

  [Seccion]

  Opcion = valor
- Permite leer/modificar/crear archivos =D
- Existe ConfigObj (mas flexible) pero no estandar
- Let's see code!

Logging

- Para archivos simples muy sencillo
- Para multiples modulos y diferenciarlos hay que hacer magia!
  - “hardcodeando” en codigo
  - Con archivos de configuracion
  - Con diccionarios “embebidos” (No recomend)

- http://docs.python.org/howto/logging.html#configuring-logging

eseculito (sqlite3)

- conn = sqlite3.connect(“mydb.db”)
  - Se puede poner “:memory:”
- cursor = conn.cursor()
- cursor.execute(SQL)
- Se puede hacer todo, crear, insertar, modificar, eliminar, pedir,...
- Mejor codigo =D

Fin dia 2!

Dia 3 =D

Antes de seguir

- ¿Preguntas/dudas?
- pythong.org
-
-
- De nuevo:

  http://rockneurotiko.github.io

  http://pili.la/githubrock

Un paseo por “os”

- os.name → plataforma
  - 'posix', 'nt', 'os2', 'ce', 'java', 'riscos'
- os.environ, os.getenv()
  - environ → diccionario accesible (error si no E)
  - getenv(valor) → devuelve el resultado o None
- os.getcwd(), os.chdir(path)
  - os.getcwd() → path de ejecucion actual
  - os.chdir(path) → cambia el path de ejecucion
- listdir(path) → lista con fich y dir

- os.mkdir() os.makedirs()
  - mkdir(p) → crea un directorio (como mkdir)
  - makedirs(p) → crea TODOS los directorios que hagan falta
- os.remove(p) → elim. un archivo (o error)
- os.rmdir(p) → igual que rmdir de linux
- os.removedirs(p) → borra directorio con directorios vacios (ningun fichero)
- os.rename(p, p2) → “mv” de toda la vida
- for root, dirs, files in os.walk(path)
  - Iterables para recorrer de forma recursiva
  - ejemplo

from os.path import *

- basename(path) → nombre del archivo
  - Ej: basename(“/ola/k/ase.py”) → “ase.py”
- dirname(path) → inversa de basename XD
- exists(path) → True/False (Socrates de directorios, cuestionando su existencia)
- isdir(path) / isfile(path) → True/False...
- join(*paths) → junta con el separador adecuado
- split(path) → tupla: (dir, file)

“os” in linux rulez!

- Cosas muy interesantes, merece la pena investigar. P.E:

```
from os import fork

pid = fork()

if pid==0:

    #Hijo

    #Para salir hay que usar os._exit(int) en lugar de exit(int)

else:

    #Padre
```

Subprocess - call

- .call(str) → ejecuta el string

  ej: subprocess.call(“gnome-terminal”)

  util para ejecutar aplicaciones, o codigo en otros lenguajes. Espera al final y coge el valor de terminacion (int)

- .call([str, str,...]) → programa + param.

  Ej: subprocess.call([“ping”, “google.com”])

Subprocess - Popen

- process = subprocess.Popen(str/list) → instancia de clase Popen

  ej: p = subprocess.Popen([“ls”, “-l”])

  ejecuta en segundo plano, no espera al fin.
- Popen tiene muchos metodos utiles:

  kill(), pid, terminate(), wait(), communicate()
- Para saber el valor de terminacion hay que hacer:

  codigo = p.wait()

Subprocess – mas Popen

- ¿Si queremos coger la salida? PIPE!

  args = [“ls”, “-l”]

  p = Popen(args, stdout=subprocess.PIPE)

  data = p.communicate() → tupla: (stdout, stderr)

sys

- Sys.argv → lista con los arg. del program.

  Ej: python test.py a b c

  sys.argv → [“test.py”, “a”, “b”, “c”]
- Sys.executable → path del interprete
- Sys.exit(int) → termina el prog con $int
- Sys.path → lista con el PYTHONPATH
- Sys.platform → identificador de S.O.
- Sys.stdin/stdout/stderr → objetos “File” mapeados a las salidas/entrada

threading (y algo mas)

- Thread → clase de la que heredar y sobre-escribir el metodo run(self) **#EJEMPLOS**
- Usando Queue para evitar cosas malas
- El problema de GIL (Global Interpreter Lock)
- Mejorando velocidad y legibilidad (para mi) gracias a multiprocessing

pdb

```
import pdb
pdb.set_trace()
```

pip

- www.pip-installer.org/en/latest/
- apt-get install pip | pip3 | pip2
- pacman -S pip
- pip install -U pip → actualizar pip
- pip install $package
- pip install -r requirements.txt

virtualenv

- pip install virtualenv
- cd $carpeta
- virtualenv $nombre
- source $nombre/bin/activate

    Cambia $PATH

Librerias “unicas”

- sh → interfaz de subprocess.

  Llama a cualquier comando como funcion o.O

  http://pypi.python.org/pypi/sh
- Black-magic → Utilidad para decoradores

  https://pypi.python.org/pypi/black-magic
- Toolz → Utilidades para programacion funcional

  https://pypi.python.org/pypi/toolz/
- Requests → Utilidad para peticiones web

Infinitas mas!

- Necesidad → buscar :)

(Y si no, en reddit.com/r/Python suelen poner mucha mierda)

Metodos magicos :)

Qué son

- Métodos internos de objetos para hacer cosas malignas

- def __new__(cls,...)

  Lo primerísimo

  No es muy util, solo para subclases de tipos inmutables
- def __init__(self, ...)

  El constructor
- def __del__(self)

  Pocos usos, cerrar archivos, conexiones,...

  aunque mejor programar bien :)

Comparar dos instancias!

- instancia.equals(instancia2)
- Instancia == instancia
- ¿Cual mejor?

- __cmp__(self, other)

  No es recomendable
- __eq__(self, other) # ==
- __ne__(self, other) # !=
- __lt__(self, other) # <
- __gt__(self, other) # >
- __le__(self, other) # <=
- __ge__(self, other) # >=

  EJEMPLO: Palabra subclase de str que compare con len

Aritmetica!

- instancia.add(instancia2)
- Instancia + instancia2
- ¿Cual mejor?

- __add__(self, other) # +
- __sub__(self, other) # -
- __mul__(self, other) # *
- __floordiv__(self, other) # //
- __div__(self, other) # /
- __mod__(self, other) # %
- __pow__(self, other) # **

EJEMPLO: Vectores

Conversion de tipos

- __int__(self)
- __long__(self)
- __float__(self)
- __hex__(self)
- ...

Otros

- __str__(self)
- __unicode__(self)
- __format__(self, formatstr)

Saber mas:

- www.rafekettler.com/magicmethods.html


I love magic.

Iterables

- Literalmente, algo que se puede recorrer.
- Los tipicos son:

  Listas, tuplas, strings, archivos,...
- Se puede recorrer con un for, o acceder por indice, porque TODO el contenido lo guarda en RAM

Iteradores

- También se pueden recorrer, pero:
- NO se guarda en memoria → no se puede acceder por indice.
- Sin embargo, se puede recorrer.
- ¿Porqué?
- Contiene un metodo __next__ que es una regla para dar el siguiente elemento.
- Sólo se puede recorrer una vez.

Generadores

- Son módos de generar iteradores de forma sencilla.
- La sintaxis == comprehension de listas pero en lugar de [] son ()
- Ej: elem = (x**2 for x in range(10))
- NO se podría acceder: elem[0] →TypeError
- Se podría recorrer, tanto:

  for i in elem. → En cada iteracion i

  next(elem) → A cada llamada da el siguiente

Yield

- Se usa para hacer generadores/iteradores más complejos.
- Se usa en una funcion, reemplazando la funcion la return
- Creas una instancia de la funcion, y puedes recorrerla.
- Cada next() (o iteracion de for) se ejecuta la funcion y para en yield, y devuelve el valor.
- EJEMPLO!

itertools

- Import itertools
- Muchísimas funciones utiles para hacer iteradores.
- Ej:

  lista = [1,2,3,4,5]

  itertools.permutations(lista)

- dir(itertools)

Algo de meta-programacion

- Creación de clases de forma dinámica
- En tiempo de ejecución, según el programa necesite
- Un campo muy amplio, mejor leer ;-)
- Algunos ejemplos!
- http://python-3-patterns-idioms-test.readthedocs.org/en/latest/Metaprogramming.html

Decoradores en 11 pasos

1) Funciones

- Def foo():
- 	Return 1

2) Scope

- Una funcion accede a su scope y a las variables globales

3) Resolucion

- Podemos acceder a las globales, siempre que no se re-asignen

4) Tiempo de vida de variables

- Si una variable se define dentro de una funcion, fuera no se lee.
- Def foo():

  x=1

  print(x)

  → Error

5) Argumentos

- Los argumentos son parte del scope local :)

6) Funception

- Se pueden hacer funciones dentro de funciones, y la primera funcion llama a la la de dentro
- EJ()

7) Funciones son tipos

- Def foo():
- Pass
- a = foo()
- type(a)
- → function

- Por lo que se puede devolver! =D
- EJ()

8) Cierre

- Una funcion va a ver sus variables locales durante todo su tiempo de vida, y todas sus cosas internas veran todo
- EJ()

9) Decoradores!

- Basicamente podemos recibir una funcion como parametro, hacer cosas antes y/o despues de la ejecucion de la misma.
-
- Mejor codigo :)

10) Simbolo @

- El simbolo @ es un modo de llamar a decoradores.
- Normal seria:

```
Def foo():
    Pass
a = decorador(foo)
a()
```

10) @

- Pero python permite hacer algo asi:

```
@decorador
def foo():
    pass
foo()
```

11) General!!

- Usando *args y **kargs
- Codigo :)

Ejemplo de Flask para hacer una REST API sencilla en mi github

¿Donde hacer ejercicios?

- http://exercism.io/
- http://checkio.org/
- http://pythonchallenge.com/
- http://coj.uci.cu/index.xhtml

¿Algunos cursos?

- Udacity:
  - http://udacity.com/course/cs101
- Coursera:
  - http://coursera.org/course/interactivepython
- http://rockneurotiko.github.io XDDD

Links

- http://rockneurotiko.github.io
- http://mirror7.meh.or.id/Programming/GrayHatPython.pdf
- http://guide.python-distribute.org/
- http://codecondo.com/coding-challenges/
- http://http://pyvideo.org
-

Fin dia 3 =D