



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:

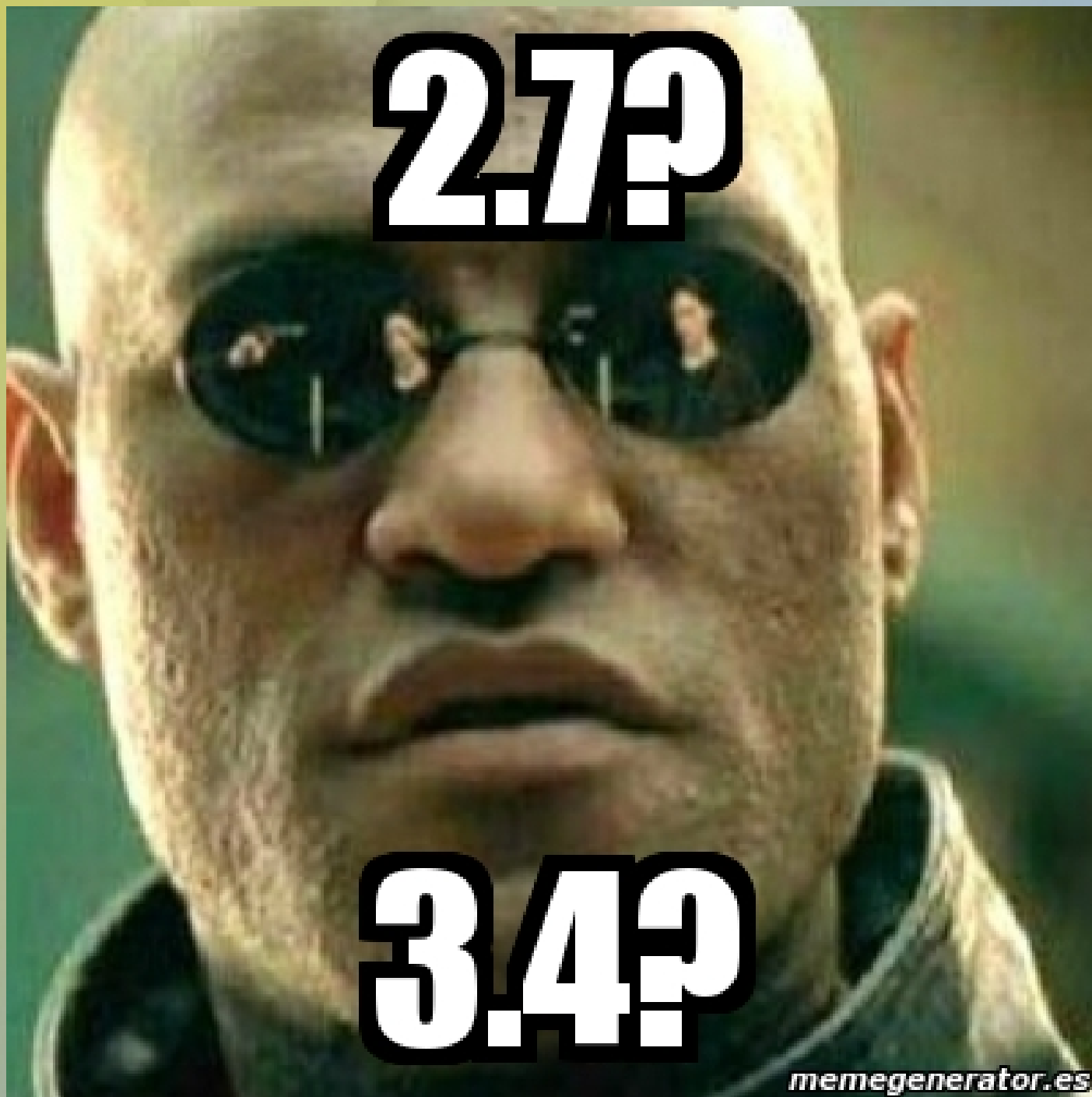
<http://www.python.org/download/releases/2.7.6/>

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

- Linux:

`pacman -S python2.7` (Arch)

[Much Google]



¿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

- Código en un fichero (.py preferiblemente)
- Desde terminal:
- `python <nombreDelArchivo> [parametros]`
- Primera línea: `#!/usr/bin/python` (o variante)
- `chmod +x <nombreDelArchivo>`
- `./<nombreDelArchivo>`

Comentarios

- Línea: #
 - #Esto es un comentario
- Bloque: `""" blah blah blah """`
 - `"""`
Esto es un
comentario en
bloque
`"""`

Variables y Constantes

- Sin tipado escrito (tipado dinámico)
 - `>>>nombre_var = "hola!"`
 - `>>>nombre_var = 8`
 - `>>>PI = 3.1415`
- Variables minúsculas y `_` de separación
- Constantes mayúsculas

¡¡EJEMPLOS!!

Tipos

- Numero
 - Entero: `a = 2; b = 0o10 [octal] ;c = 0x23 [Hex]`
 - Long: `a = 456966786151987643L (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 ;-)

[illegible]

Y en Python! =D

[illegible]



**ONE DOES
NOT SIMPLY**

COMPARE IN JS

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! (Wiiiiiii!)

(Notese la indentacion)

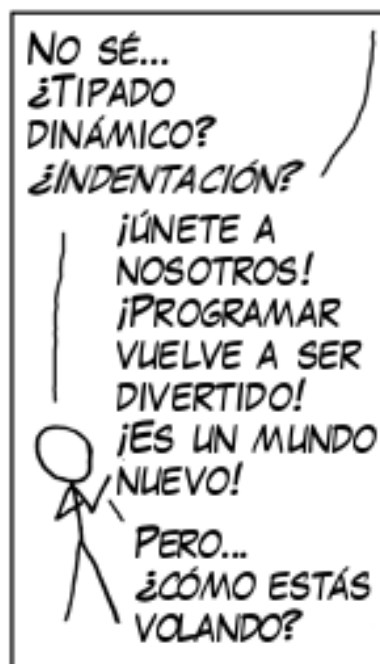
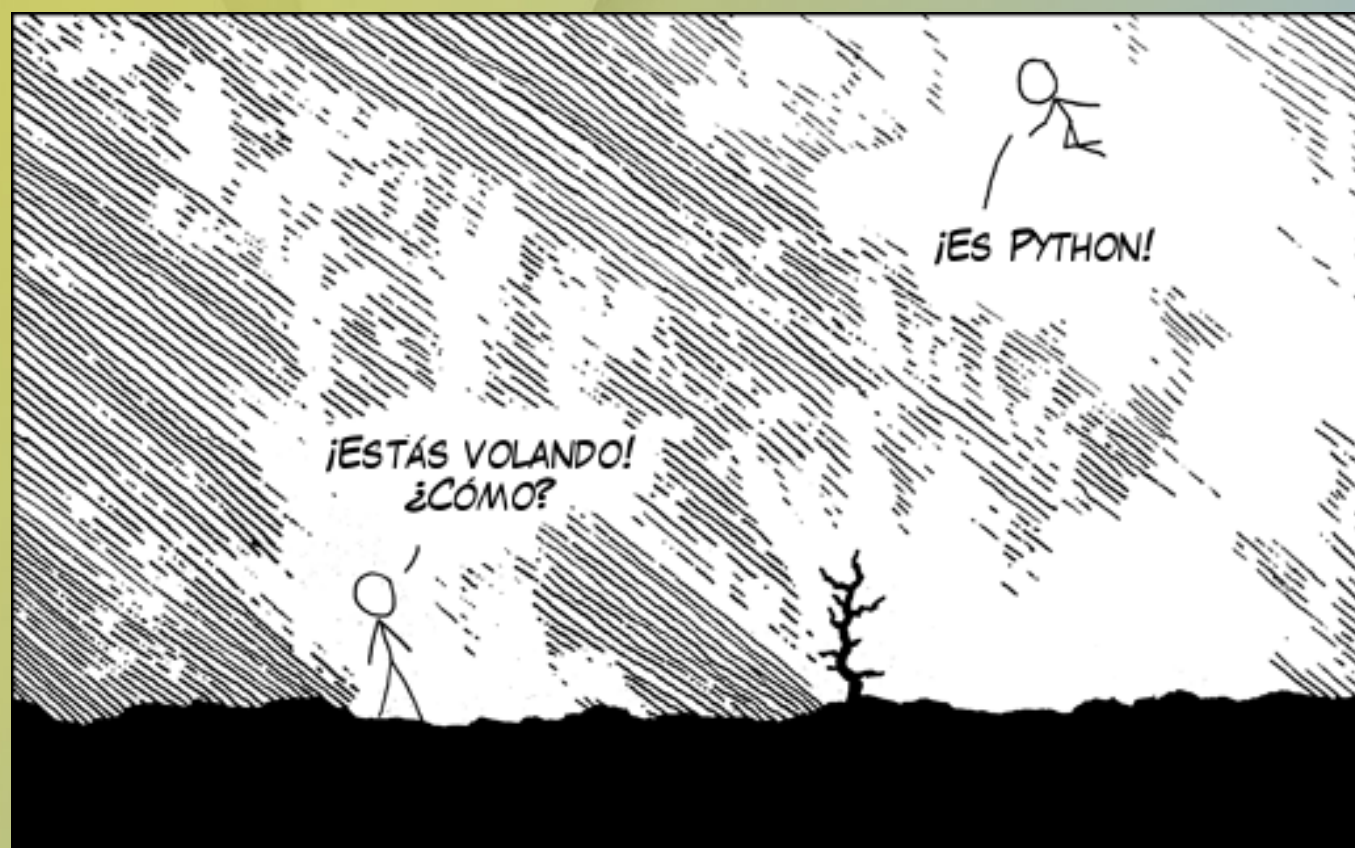
- if <<condicion1>>:
 hacer cuando cond1
- elif <<cond2>>:
 hacer cuando con2 y no cond1
- else:
 hacer en otros casos

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!



¡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 módulos/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 código 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?

Moar funcs!

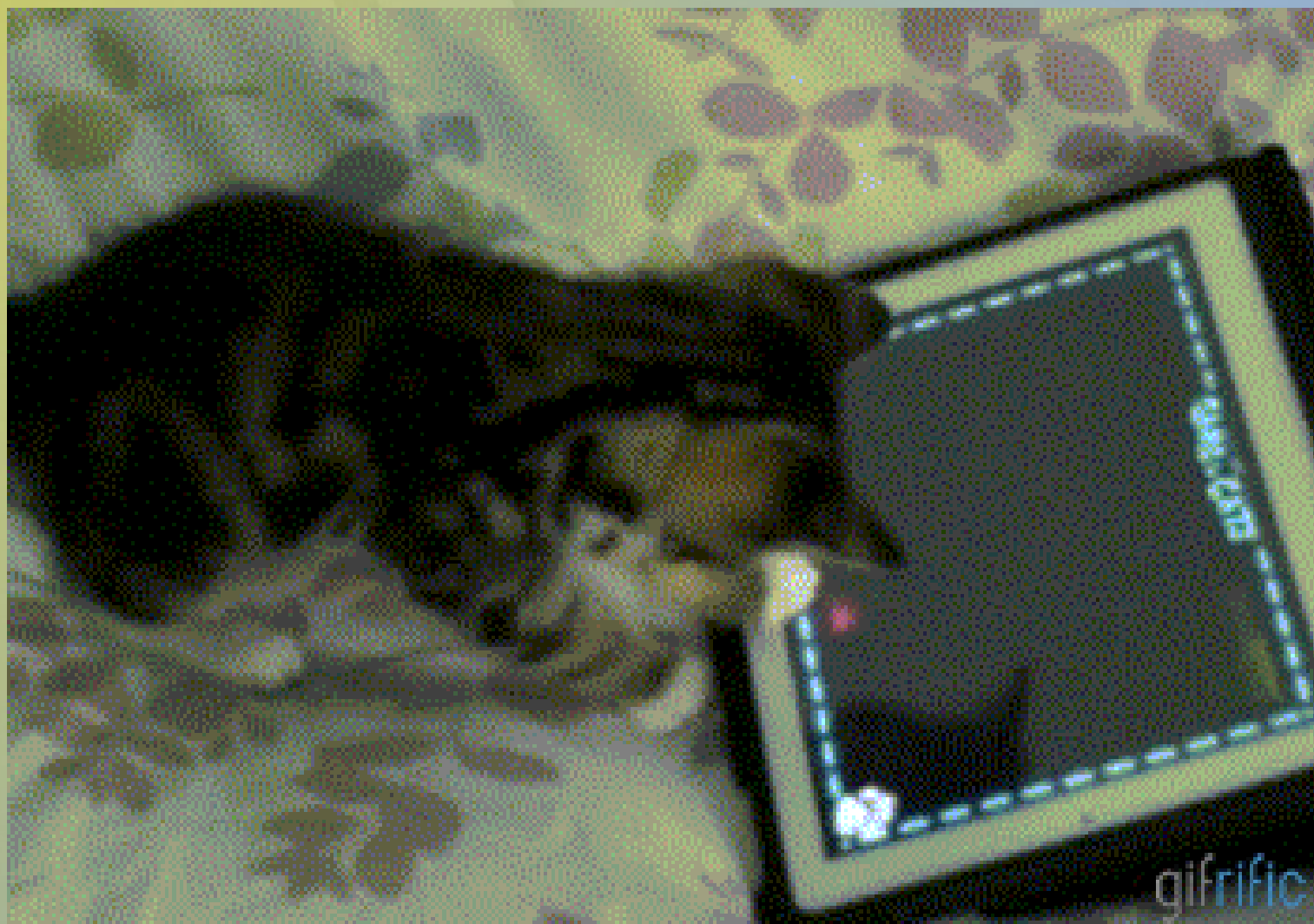
- `def func(arg_fijo, *arg_variable, **k_arg)`
- Los variables se recorren como lista
- Los variables “k” (keyword) se recorren como diccionario
- Desenpaquetando parametros en llamada... (Wut?)
- `locals()`, `globals()`... Llamando a funciones de retorno de forma dinámica

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/githubbrock>

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 NombreClase:
```

```
    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([character])` → Quita “character”, 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 líneas
`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 revés
- `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) ← n° pares`
- `for i in range(2,8): [N° 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

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` → lista `DicReader` → 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 código = D

Fin dia 2!

