From n00b to h4x0r!

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

¿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

Indice

Install it pls!

- Windows & MAC:

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

- 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)
- “Enpaquetable” en ejecutables (.exe)
- www.python.org/dev/peps/pep-0008/

Warnings!

- No hay {}, se usan tabulaciones:

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

Intérprete!

- En una consola (Meta+R, cmd):
  - python (o python2.7 si es arch)
  - >>>
- Salir del interprete:
  - Crt+D
  - Exit()
- ipython: shell “mejorado”

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]

Comentarios

- Linea: #
  - #Esto es un comentario

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

    Esto en un

    comentario en

    bloque

    “””

Variables y Constantes

- Sin tipado (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 = 010 [octal] ;c = 0x23 [Hex]
  - Long: a = 45696678615198764 3L
  - 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, listas,...)
- 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]

Operaciones relacionales (pa' comparar vamos)

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

Operadores lógicos

- AND = and:
  - True and False
- OR = or:
  - True or True
- XOR = xor:
  - False xor True
- El resultado es un booleano (tx Mr.Obvius)

Estructuras de flujo condicionales! (Wiiiiii!)

- (Notese la indentacion)
- if <<condicion1>>:

    hacer cuando cond1

  elif <<cond2>>:

    hacer cuando con2

  else:

    hacer en otros casos

Estructuras de control iterativas

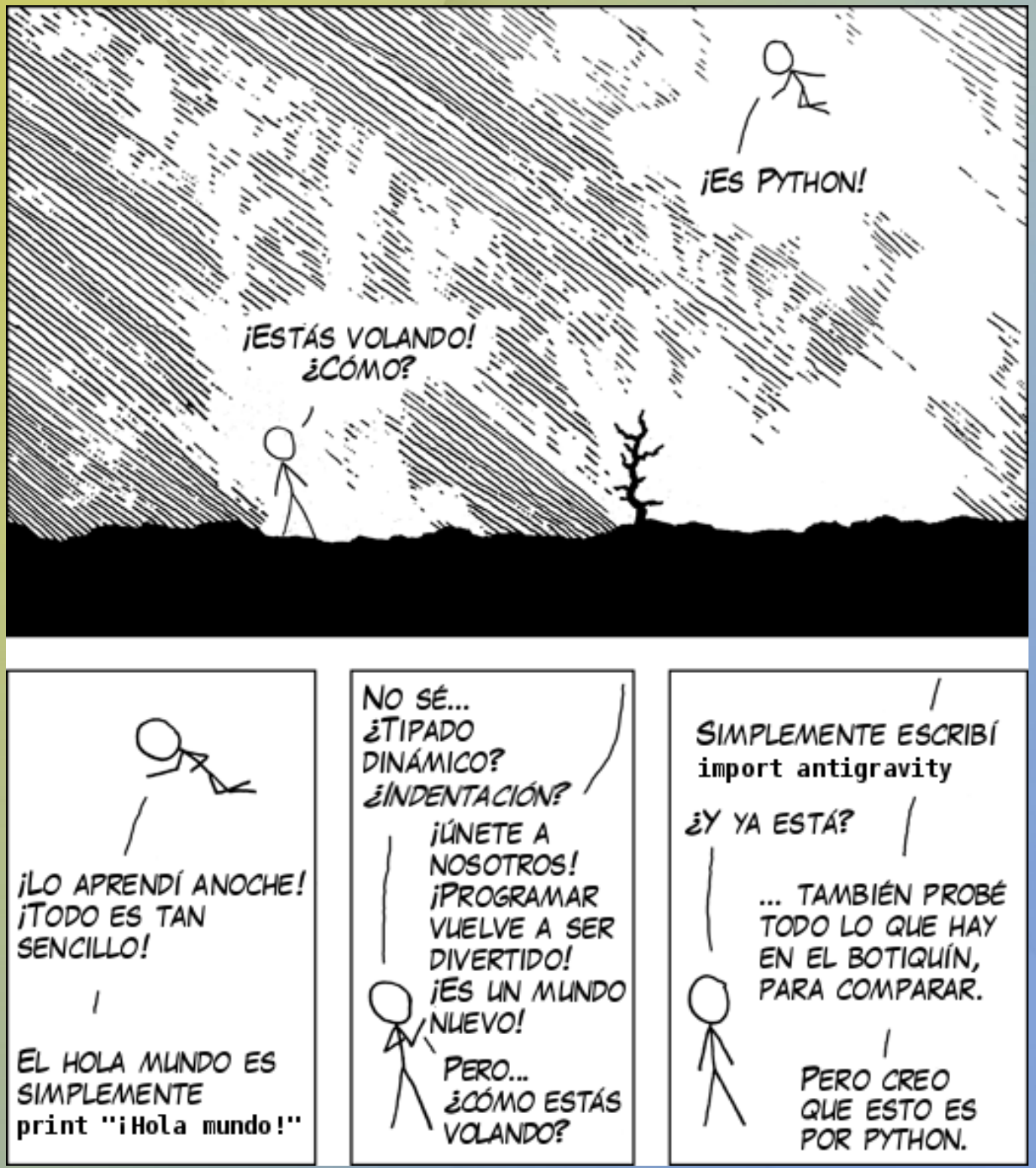
- while <condicion>:

  #que hacer

- for <nombre_var> in <estructura>:

  #que hacer

- Recorre listas y tuplas, diccionarios recorre las keys


¡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!

- 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 modulos

- 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

- 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!!

Funciones

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

  #Cosas para hacer

  #return optativo

- Mejor codigo y ejemplos =D

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

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: (del objeto)

```
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.

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_metodo(self) == “Tu sabras lo que haces si tocas esto”

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)
- isalnum, isalpha, isdigit ()
- 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, retorna el mensaje “El nombre de usuario debe contener al menos 6 caracteres”
- Nombre de usuario con más de 12 caracteres, retorna el mensaje “El nombre de usuario no puede contener más de 12 caracteres”
- Nombre de usuario con caracteres distintos a los alfanuméricos, retorna el mensaje “El nombre de usuario puede contener solo letras y números”
- Nombre de usuario válido, retorna True

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, retorna el mensaje “La contraseña elegida no es segura”

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]
```

Some about dics

- update(dic2) → Une diccionarios
- get(clave) → Devuelve el valor de la clave
- has_key(clave) → True o False
- Iteritems() → iterador con los pares

```
for clave, valor in dic.iteritems():
    print “Clave: “, clave, “Valor: “, valor
```

- keys(), values()

Data pls!!! (1)

- 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?
- WTF?

Data pls!!! (2)

- raw_input([mensaje])
- Se va a “tragar” todo → como un String
- Se suele usar para coger strings, o numeros y despues castear (aunque para eso input con try_except)

  a = raw_input(“Hola!: “)

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

Error? That's a feature!

- try, except y finally

  try:

      #Acciones que pueden dar error

  except:

      #Que hacer si salta la excepcion

  finally:

      #Que hacer despues de todo, vaya bien o mal

Excepciones concretas

- except puede llevar el tipo de error a capturar

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

Raise it!

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

```
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

- 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, criba erastotenes]

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

Usos en Python – map

- map(): devuelve una 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]

2 en 1 – An Unix tip & lambda use!

```
from commands import getoutput

mount = getoutput(“mount -v”)

lines = mount.splitlines()

p = 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”]
- 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
- for line in f: → Dentro del bucle, cada it. es una linea :) #Mas usado eveh!
- seek(num) → va a la pos. num (en Bytes)

A little about CTypes

- from ctypes import * (o import ctypes)
- libc = CDLL(“libc.so.6”)

  libc.printf(“Hola mundo!\n”)
- p = “\x00\x00\x00”

  memory = create_string_buffer(p,len(p))

  sc = cast(memory,CFUNCTYPE(c_void_p))

  sc()
- “Gray Hat Python” ← En la referencia

- http://mirror7.meh.or.id/Programming/GrayHatPython.pdf
-