

From Nothing to Python

Contacto y grupos interesantes



t.me/cyberh99

DevUco Group and projects :-)

<https://t.me/joinchat/EWwrOk34IP-Qit7TLHoutg>

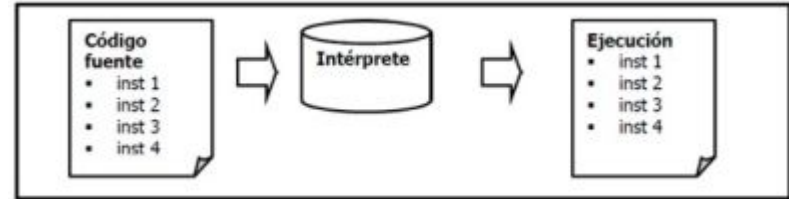
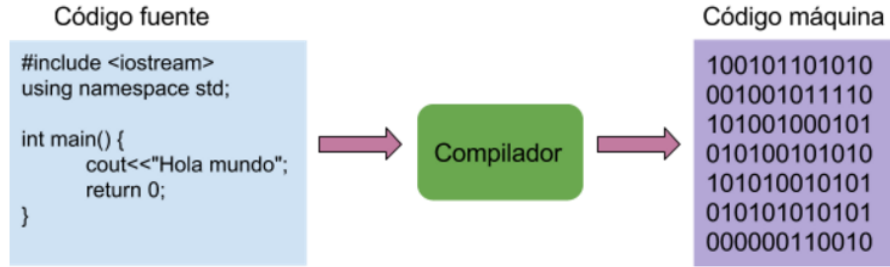
Aula Software Libre

<https://t.me/AulaSoftwareLibreUCO>

Our friends



Lenguajes interpretados vs Lenguajes compilados



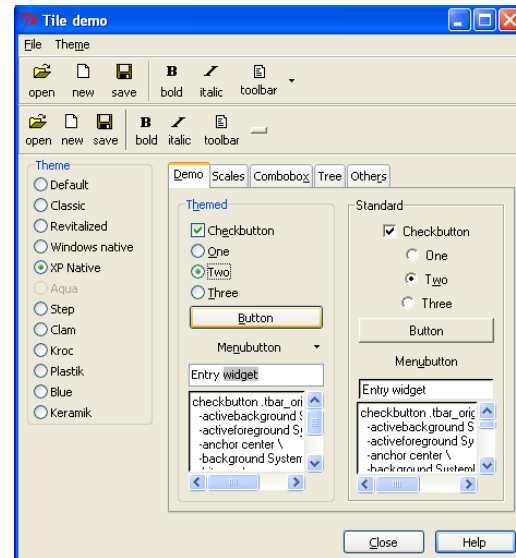
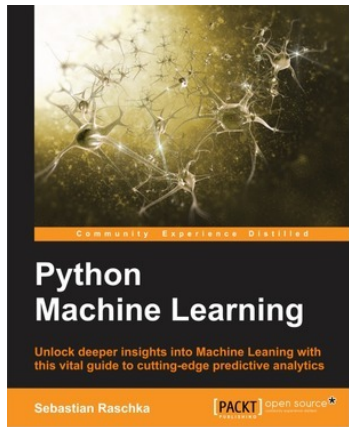
Ventajas de lenguajes interpretados

- Pueden ser ejecutados en cualquier plataforma.
- Ocupan menos espacio en la memoria.
- El framework es el que se encarga de que el hardware ejecute las instrucciones.
- Las variables de datos son dinámicas y no se restringen a un solo tipo.
- Son más utilizados en desarrollo web y en electrónica.

Desventajas de lenguajes interpretados

- Su ejecución es más lenta con respecto a los lenguajes compilados.
- Son más difíciles de debuggear.
- Requieren de una máquina virtual que sirva como intérprete las instrucciones y el procesador.
- No todas las máquinas virtuales están disponibles para todas las plataformas.

Python for all



Instalando Python

From terminal (Linux system):

- `apt install python`

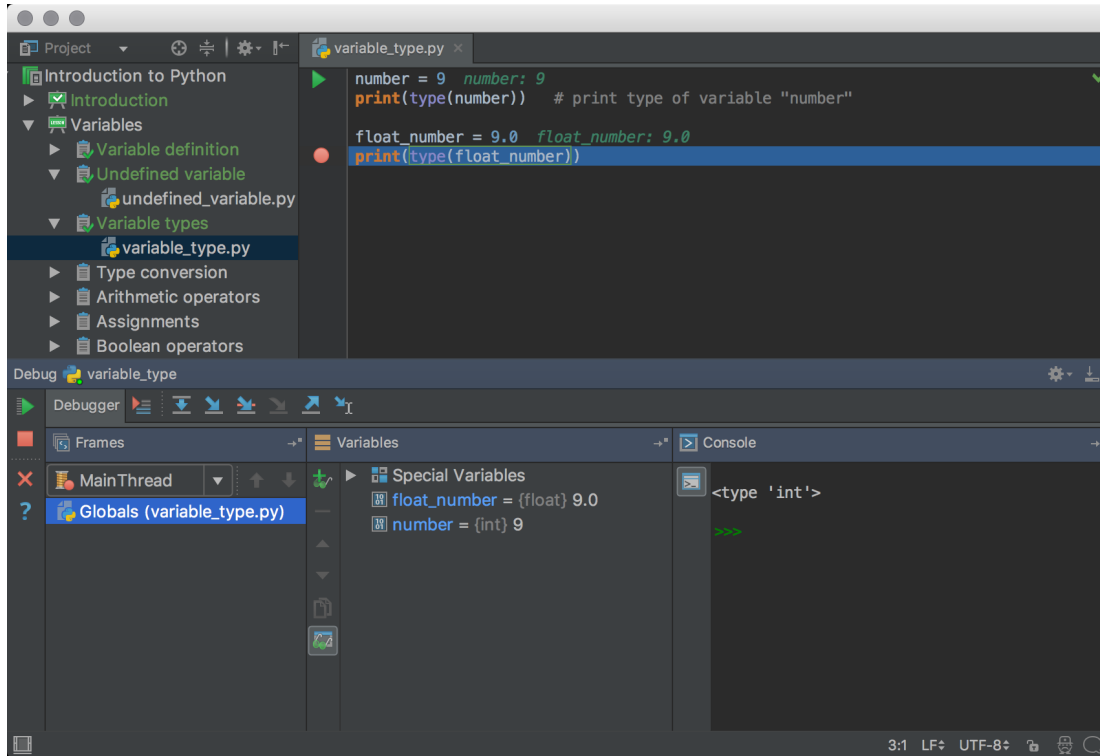
From Windows:

- <https://www.python.org/>

Características de Python

- Python es orientado a objetos, cualquier cosa en python es un objeto (incluyendo listas, diccionarios)
- Lenguaje de alto nivel
- Lenguaje interpretado (aunque puede ser compilado con ayuda de módulos externos)
- Disponibilidad de librerías de forma extensa
- Open Source

IDE para python



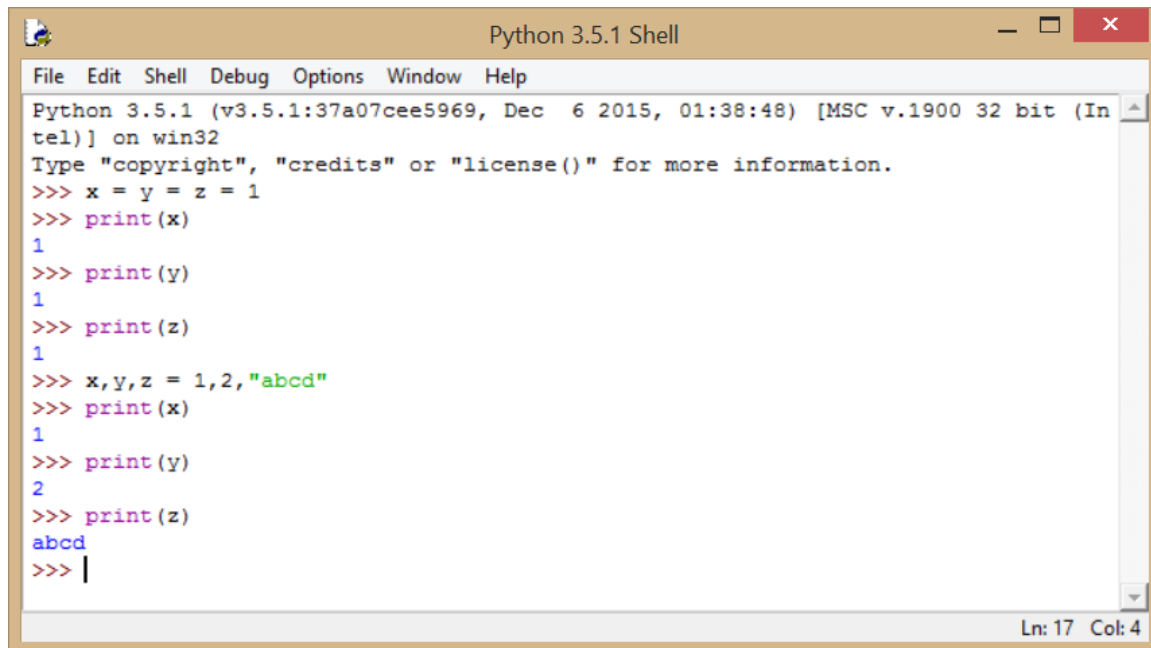
<https://www.jetbrains.com/pycharm/>

Tipos de variables

- ✓ Int
- ✓ Long
- ✓ Float
- ✓ Bool
- ✓ string
- ✓ Unicode

- ✓ Listas
- ✓ Tuplas
- ✓ Diccionarios

- ✓ `type(variable)`



```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = y = z = 1
>>> print(x)
1
>>> print(y)
1
>>> print(z)
1
>>> x,y,z = 1,2,"abcd"
>>> print(x)
1
>>> print(y)
2
>>> print(z)
abcd
>>> |
```

Ln: 17 Col: 4

Listas / Tuplas / Diccionarios

★ Listas:

- Almacenan datos de mismo (o diferente) tipo
- Los datos pueden ser modificados sobre la marcha
- lista = ["asd",2,3,True]

★ Tuplas:

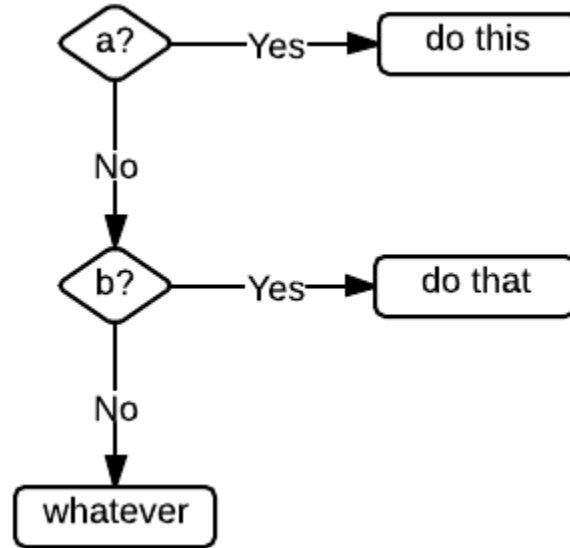
- Almacenan datos de mismo (o diferente tipo)
- Los datos NO pueden modificarse sobre la marcha
- tupla = ("AD",2,3,False)

★ Diccionario:

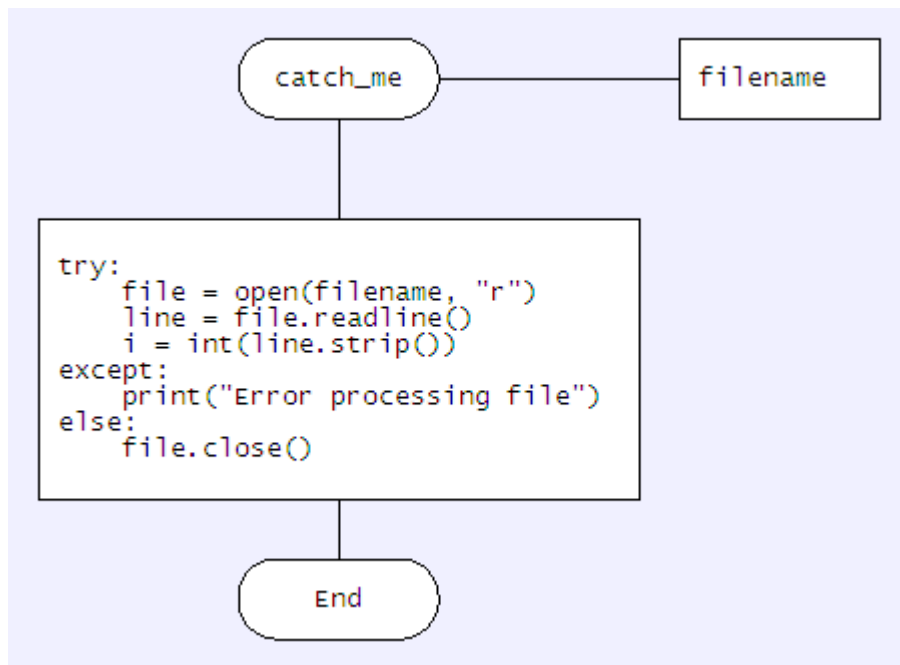
- Almacenan valores clave:valor
- diccionario = {"hola":"pepe", "Adiós":"tito"}

If / else / elif

```
if a:  
    do this  
elif b:  
    do that  
else:  
    whatever
```



Try / Except



For Loop

```
string = "Hello World"  
for x in string:  
    print x
```

```
collection = ['hey', 5, 'd']  
for x in collection:  
    print x
```

```
list_of_lists = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]  
for list in list_of_lists:  
    for x in list:  
        print x
```



While Loop

```
n = raw_input("Please enter 'hello':")  
while n.strip() != 'hello':  
    n = raw_input("Please enter 'hello':")
```

```
while True:  
    n = raw_input("Please enter 'hello':")  
    if n.strip() == 'hello':  
        break
```


¿Pointers on python?



I want `form.data['field']` and `form.field.value` to always have the same value

34



This is feasible, because it involves decorated names and indexing -- i.e., **completely** different constructs from the **barenames** `a` and `b` that you're asking about, and for with your request is utterly impossible. Why ask for something impossible **and** totally different from the (possible) thing you actually *want*?!



Maybe you don't realize how drastically different barenames and decorated names are. When you refer to a barename `a`, you're getting exactly the object `a` was last bound to in this scope (or an exception if it wasn't bound in this scope) -- this is such a deep and fundamental aspect of Python that it can't possibly be subverted. When you refer to a *decorated* name `x.y`, you're asking an object (the object `x` refers to) to please supply "the `y` attribute" -- and in response to that request, the object can perform totally arbitrary computations (and indexing is quite similar: it also allows arbitrary computations to be performed in response).

Now, your "actual desiderata" example is mysterious because in each case two levels of indexing or attribute-getting are involved, so the subtlety you crave could be introduced in many ways. What other attributes is `form.field` suppose to have, for example, besides `value`? Without that further `.value` computations, possibilities would include:

<https://stackoverflow.com/questions/3106689/pointers-in-python>

Comentarios

```
5  #asad
6
7  '''
8  asd
9  asd
10 '''
```

`#!/usr/bin/env python`

`#-*- coding: utf-8 -*-`

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

print "Hello World"
```

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

class Hello:
    def __init__(self,nombre):
        self.nombre = nombre

    def say_hello(self):
        print "Hello %s"%(self.nombre)

Hello("Hector").say_hello()
```

```
#include <iostream>

int main(int argc, char const *argv[]) {
    /* code */
}
```

```
#include <iostream>

class hello{

public:
    void say_hello(){
        std::cout<<"Hello World \n";
    }
};

int main(int argc, char const *argv[]) {
    /* code */

    hello h = hello();
    h.say_hello();

    return 0;
}
```

Strings Stuff

Accediendo al contenido

- Accediendo por índices:
 - ◆ `string[0]`
- Accediendo rangos:
 - ◆ `string[0:6]`
- A la inversa:
 - ◆ `string[-2]`

Strip y Split

`split(",")` → Separa las palabras según un carácter

`strip("0")` → Elimina el carácter especificado de la cadena

Contenido dinámico

`nombre = raw_input()`

`saludos = "Buenas tardes %s" % nombre`

Conversion	Meaning
'd'	Signed integer decimal.
'i'	Signed integer decimal.
'o'	Signed octal value.
'u'	Obsolete type – it is identical to 'd'.
'x'	Signed hexadecimal (lowercase).
'X'	Signed hexadecimal (uppercase).
'e'	Floating point exponential format (lowercase).
'E'	Floating point exponential format (uppercase).
'f'	Floating point decimal format.
'F'	Floating point decimal format.
'g'	Floating point format. Uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
'G'	Floating point format. Uses uppercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
'c'	Single character (accepts integer or single character string).
'r'	String (converts any Python object using <code>repr()</code>).
's'	String (converts any Python object using <code>str()</code>).
'%'	No argument is converted, results in a '%' character in the result.

<https://docs.python.org/2.7/library/stdtypes.html#string-formatting-operations>

Listas Stuff

- ❖ Midiendo tamaños de listas:
 - `len(lista)`
- ❖ Accediendo valores de la lista:
 - `Lista[0]`
- ❖ Agregando valores de la lista:
 - `lista[1] = "Python"`
- ❖ Agregando elementos al final de la lista:
 - `lista.append(variable)`
- ❖ Eliminando elementos de la lista:
 - `del lista[9]`

Diccionarios Stuff

- Listando las claves de los diccionarios:
 - ◆ `diccionario.keys()`
- Accediendo a los valores de diccionarios:
 - ◆ `diccionario[key]`
- [RE]Asignando valores:
 - ◆ `diccionario[key] = "Adios"`
- Obteniendo la longitud de los diccionarios:
 - ◆ `len(diccionario)`

Funciones

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def ooo_some_code():
    print ("Hello World \n")

ooo_some_code()
```

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

name = raw_input("¿Qual es tu nombre?")

def say_hello(nombre):
    print ("Hello %s"%(nombre))

say_hello(name)
```

Let's do the terminal stuff



I don't know what to do

- ❖ Pedir entrada por teclado hasta que el usuario introduzca la palabra exit.
- ❖ Pedir una serie de datos por teclado y almacenarlos en una lista. Imprimiendo esta última antes de salir del programa.
- ❖ Crear un programa que lleve el registro del nombre, apellidos y peso de los alumnos de un instituto.

Mr Chicken as a class

```
class Chicken(object):  
    "Describes a chicken."  
    def __init__(self, name):  
        self.name = name  
    def make_sounds(self):  
        "Print chicken sound."  
        print "squeaaaak!"
```

P00

CLASE

Propiedad y comportamiento
de un objeto concreto

ATRIBUTO

Propiedad del objeto

MÉTODO

Lo que un objeto
puede hacer
(algoritmo)

OBJETO

Instancia de una clase

MENSAJE

Comunicación dirigida a un
objeto ordenándole que
ejecute uno de sus métodos

Resumen de conceptos

Clase → Propiedad o comportamiento del objeto. “Como se comporta una botella”

Método → Lo que un objeto puede hacer. “¿Que podemos hacer con la botella?”

Atributo → Característica especial de la botella. “Material con la que esta construida”

Objeto → La maldita botella

Clases, Objetos etc...

```
class dog():  
    '''code '''
```

A. Definición de una clase en Python

```
class dog():  
    def new_dog(self):  
        self.born()  
        self.breathe()
```

1. Definición de funciones dentro de clases (método)

```
c = dog()  
c.new_dog()
```

I. Llamada a la clase(objeto) y a una función dentro de la misma

Our friend *self*

[show 10 more comments](#)

18 Answers

active

oldest

votes



535



The reason you need to use `self.` is because Python does not use the `@` syntax to refer to instance attributes. Python decided to do methods in a way that makes the instance to which the method belongs be *passed* automatically, but not *received* automatically: the first parameter of methods is the instance the method is called on. That makes methods entirely the same as functions, and leaves the actual name to use up to you (although `self` is the convention, and people will generally frown at you when you use something else.) `self` is not special to the code, it's just another object.

Python could have done something else to distinguish normal names from attributes -- special syntax like Ruby has, or requiring declarations like C++ and Java do, or perhaps something yet more different -- but it didn't. Python's all for making things explicit, making it obvious what's what, and although it doesn't do it entirely everywhere, it does do it for instance attributes. That's why assigning to an instance attribute needs to know what instance to assign to, and that's why it needs `self.`

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

class Simple_class:

    homer = "ouch"

    def __init__(self, nombre):
        self.nombre = nombre
        print "Obviously it's a simple class"

    def say_hello(self):
        print "Hello %s"%(self.nombre)

    def say_goodbye(self):
        print "Goodbye %s"%(self.nombre)

    def flirt(self):
        self.say_hello()
        self.say_goodbye()
        print self.homer

objet = Simple_class("tito")
objet.flirt()
```


» def __init__(self):

Python __init__ and self what do they do

In this code:

```
class A(object):
    def __init__(self):
        self.x = 'Hello'

    def method_a(self, foo):
        print self.x + ' ' + foo
```

... the **self** variable represents the instance of the object itself. Most object-oriented languages pass this as a hidden parameter to the methods defined on an object; **Python** does not. You have to declare it explicitly. When you create an instance of the **A** class and call its methods, it will be passed automatically, as in ... 

```
a = A() # We do not pass any argument to the __init__ method
a.method_a('Sailor!') # We only pass a single argument
```

The **__init__** method is roughly what represents a constructor in **Python**. When you call **A()** **Python** creates an object for you, and passes it as the first parameter to the **__init__** method. Any additional parameters (e.g., **A(24, 'Hello')**) will also get passed as arguments—in this case causing an exception to be raised, since the constructor isn't expecting them.

—Chris B.



Show Less



More at Stack Overflow

P A R E N T A L

ADVISORY

EXPLICIT CONTENT

Herencia

La herencia nos permite que una clase herede las variables y métodos a varias subclases.

Estas subclases, aparte de los métodos propios, tiene incorporados los atributos y métodos heredados de la anterior clase (superclase)



```
1  #!/usr/bin/env python
2  -*- coding:utf-8 -*-
3
4  class Animal:
5      def __init__(self):
6
7          print "It's an animal"
8
9      def born(self):
10         print "It's borning a new animal"
11
12     def breathe(self):
13         print "It's breathing"
14
15     class dog(Animal):
16         def new_dog(self):
17             self.born()
18             self.breathe()
19
20     c = dog()
21
22     c.new_dog()
23
```

Poliformismo

En el ámbito de la orientación a objetos el polimorfismo hace referencia a la habilidad que tienen los objetos de diferentes clases a responder a métodos con el mismo nombre, pero con implementaciones diferentes.



```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

class Animal:
    def __init__(self):

        print "It's an animal"

    def born(self):
        print "It's borning a new animal"

    def breathe(self):
        print "It's breathing"

class dog(Animal):
    def __init__(self):
        self.born()
        self.breathe()
        print "It's a dog :-)"

    def look_a_cat(self):
        print "The dog try to catch the cat"

class cat(Animal):
    def __init__(self):
        self.born()
        self.breathe()
        print "It's a cat"

    def look_a_cat(self):
        print "Tha cat fell in 'love'"

rayo = dog()
rayo.look_a_cat()
bigotes = cat()
bigotes.look_a_cat()
```

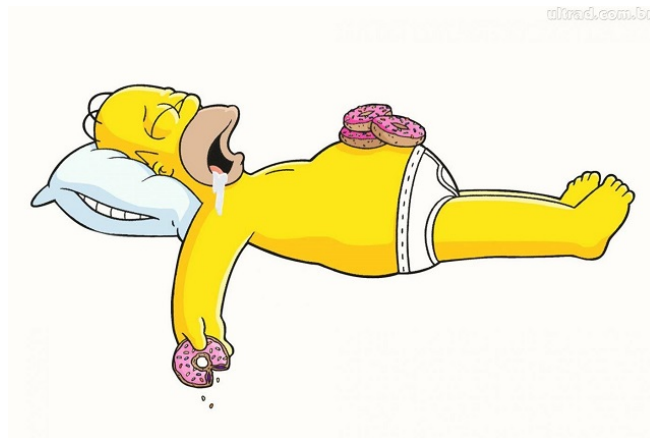
Encapsulación

La encapsulación nos permite crear funciones privadas dentro de nuestras clases, de forma que no pueden ser accesibles desde el exterior de dicha clase.

```
class Fecha
{
    public:
        /* Comentado por ineficiente
        int anho;
        int mes;
        int dia;    */

        long numeroSegundos;

        void metodoMaravilloso1();
        void metodoMaravilloso2();
};
```



```
2
3  #!/usr/bin/env python
4  #-*- coding: utf-8 -*-
5
6
7  class super_calculator:
8      def __init__(self,num1,num2):
9          print "It's too simple"
10         self.num1 = num1
11         self.num2 = num2
12
13     def __mult(self):
14
15         f_number = self.num1 * self.num2
16         return f_number
17
18     def operate(self):
19         print self.__mult()
20
21
22 casio = super_calculator(2,2)
23 casio.operate()
```

Working with files

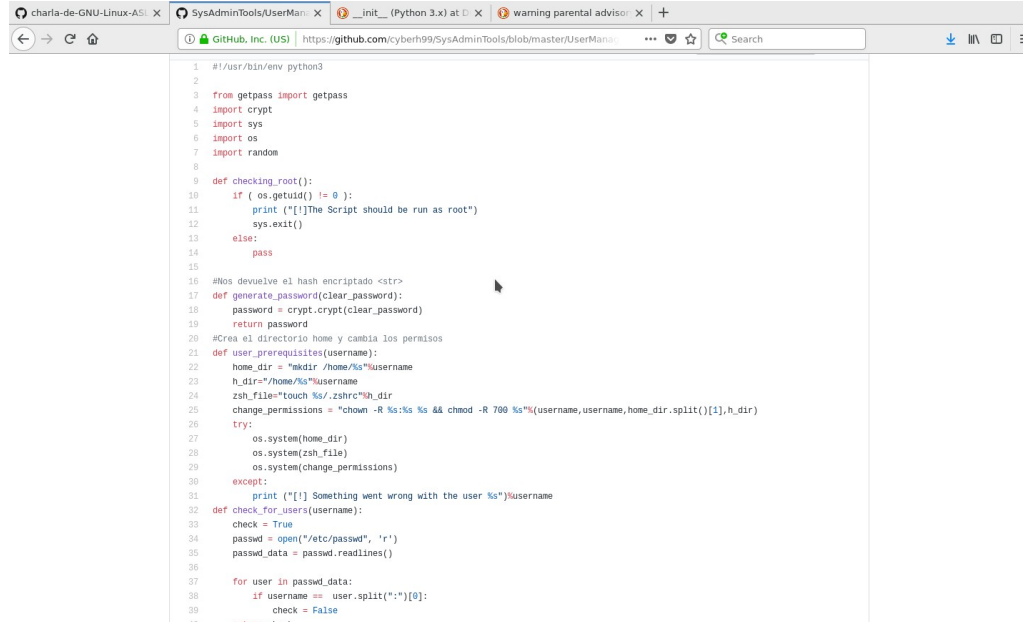
```
cyberh99@pipo 141x42
>>> file = open('exercise.md', 'r')
>>> data = file.readlines()
>>> print data
['# Enunciado del ejercicio\n', '\n', '\n', 'Llevar el control de un zoologico en Python, de forma que cada vez que se agregue un animal se r
egistre que el numero de ese animal y de su especie se ha incrementado.\n', '\n', 'Para ello hay que tener en cuenta las siguientes relacion
es:\n', '\n', '\n', 'Animal --> Reptil\n', '\n', 'Animal --> mamifero --> Bovino\n', '\n', 'Animal --> mamifero --> Canino\n', '\n', 'Animal
--> mamifero --> Felino --> Gato\n', '\n', '\n', 'Podemos utilizar las siguientes funciones (para que concuerde con la solucion)\n', '\n', '
numAnimals _numMamamls _numFelines _numCats\n', '\n', 'talk()\n', '\n', 'show()\n', '\n', '\n', '\n', 'Ejemplo: Si a\xc3\xbladimos un gato, s
e aumenta el numero de gatos, felinos, mamiferos y animales\n', '\n']
>>> file.close()
>>> 
```

Compilando archivos .py



Pyinstaller <script.py>

Some example



The screenshot shows a web browser with multiple tabs. The active tab is titled 'SysAdminTools/UserMan...' and displays a Python script from GitHub. The script is for user management and includes functions for checking root access, generating passwords, creating user directories, and checking the password file.

```
1 #!/usr/bin/env python3
2
3 from getpass import getpass
4 import crypt
5 import sys
6 import os
7 import random
8
9 def checking_root():
10     if (os.getuid() != 0):
11         print ("[!]The Script should be run as root")
12         sys.exit()
13     else:
14         pass
15
16 #Nos devuelve el hash encriptado <str>
17 def generate_password(clear_password):
18     password = crypt.crypt(clear_password)
19     return password
20
21 #Crea el directorio home y cambia los permisos
22 def user_prerequisites(username):
23     home_dir = os.path.join('/home', username)
24     h_dir = os.path.join(home_dir, username)
25     zsh_file = os.path.join(h_dir, '.zshrc')
26     change_permissions = "chown -R %s:%s %s && chmod -R 700 %s"%(username, username, home_dir, h_dir)
27     try:
28         os.system(home_dir)
29         os.system(zsh_file)
30         os.system(change_permissions)
31     except:
32         print ("[!] Something went wrong with the user %s"%username)
33
34 def check_for_users(username):
35     check = True
36     passwd = open("/etc/passwd", 'r')
37     passwd_data = passwd.readlines()
38
39     for user in passwd_data:
40         if username == user.split(':')[0]:
41             check = False
```

- <https://github.com/cyberh99/SysAdminTools/blob/master/UserManagement.py>

Bibliografía recomendada

- Python Machine Learning → <http://books.tarsoit.com/Python%20Machine%20Learning.pdf>
- Python for Linux and Unix Administration → <http://docs.linuxtone.org/ebooks/Python/OReilly.Python.for.Unix.and.Linux.System.Administration.Sep.2008.pdf>
- Web Development with Django cookbook → <https://doc.lagout.org/programming/Django/Web%20Development%20with%20Django%20Cookbook%20%5BBendoraitis%202014-11-17%5D.pdf>
- Documentación oficial → <https://docs.python.org/3/>



Relax.. Last diapo

