Introdución a Python, Git y Github

Fran Rúa
Breixo Camiña

Grupo de Programadores e Usuarios Linux

8 de febrero de 2016

Licencia

Índice

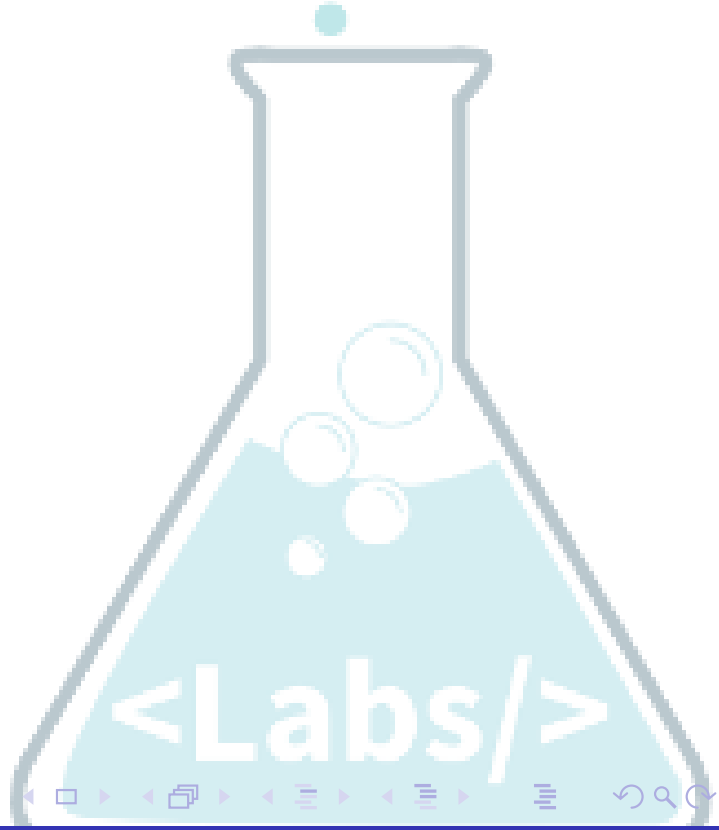
Introducción a Python

Fontes e referencias

- Documentación oficial de Python
  www.python.org/doc
- Guías de estilo PEP
  www.python.org/dev/peps/pep-0008/
- Guía para principiantes
  wiki.python.org/moin/BeginnersGuide
- Learn Python (español)
  http://www.learnpython.org/es/
- Tracducción do manual de Guido van Rossum
  http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf
- Información adiccional
  s.wikipedia.org/wiki/Python

Introducción

- Historia
- Para qué é usado.
- Características
- Ventaxas
- Inconvintes



Historia

- Creado a finais dos 80's por Guido van Rossum
- Desenvolvido para o SO Amoeba
- Toma partes de outros linguaxes de programación, como Haskell ou ABC
- A súa finalidade é programar facilmente e obrigar ó programador a realizar código entendible.
- Python 1.0 liberado en xaneiro de 1994
- Actualmente dous proxectos paralelos: Python 2.7 e Python 3.4
- Instalado por defecto na maioría das distribucións GNU/Linux

Características

- Linguaxe de programación interpretado
- Multiplataforma
- Multiparadigma (Programación imperativa, orientada a obxectos e funcional)
- Tipado dinámico
- Código aberto (Python Software Foundation License, compatible con GNU)
- Herencia
- Modo interactivo

Para qué é usado

- Inicialmente, tareas livianas ou de administración.
- Posúe unha ampla cantidade de librerías que posibilita un amplo rango de escenarios de uso
- Computación numérica
- Xeración de gráficos
- Programación web
- Interacción con bases de datos
- Programas de usuario con interface gráfico.

Funcionamento

Ó contrario que C ou Java, Python non precisa ser compilado.
En realidade Python é un intérprete de comandos que acepta unha serie de instruccións que respetan unha serie de normas lóxicas e semánticas.
Por tanto, se queremos executar o noso código en múltiples plataformas, só é necesario instalar o intérprete, xa que despois será éste o encargado de executar as órdes indicadas polas instruccións.

Compilación vs. Interpretación

Figura: Compilacion de un programa

Compilación vs. Interpretación

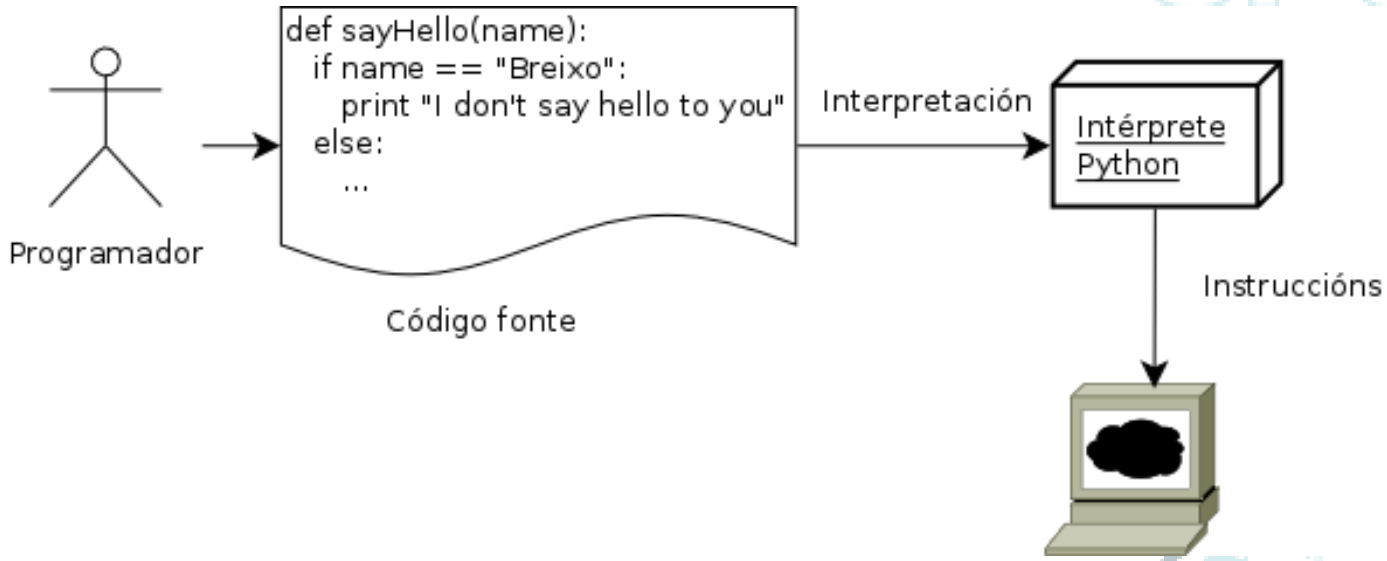


Figura: Interpretacion de un programa

Que é o modo interactivo?

O modo interactivo permítenos iniciar unha instancia do intérprete de python, o que nos permite programar en tempo real e comprobar a sintaxis do noso programa.
Neste modo podemos cargar librerías e módulos, ee executar funcións de forma rápida sen necesidade de realizar programas previamente.

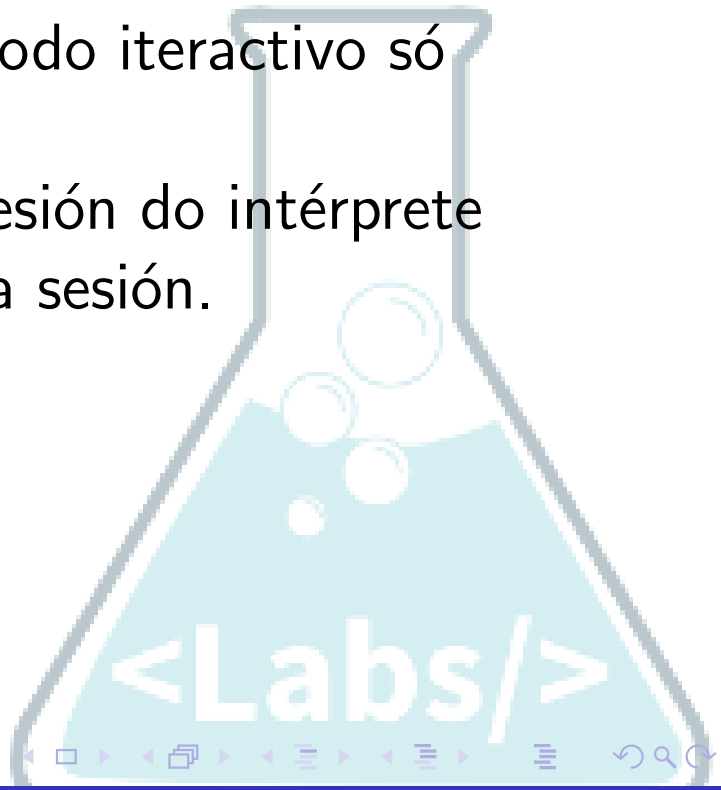
Iniciar modo interactivo

Nos sistemas GNU/Linux soamente é necesario abrir un emulador do terminal e escribir *python*.

```
[fran@izanami ~]$ python
Python 2.7.10 (default, Sep 24 2015, 17:50:09)
[GCC 5.1.1 20150618 (Red Hat 5.1.1-4)] on linux2
Type "help", "copyright", "credits" or "license" for more i
>>>
```

Sair do modo interactivo

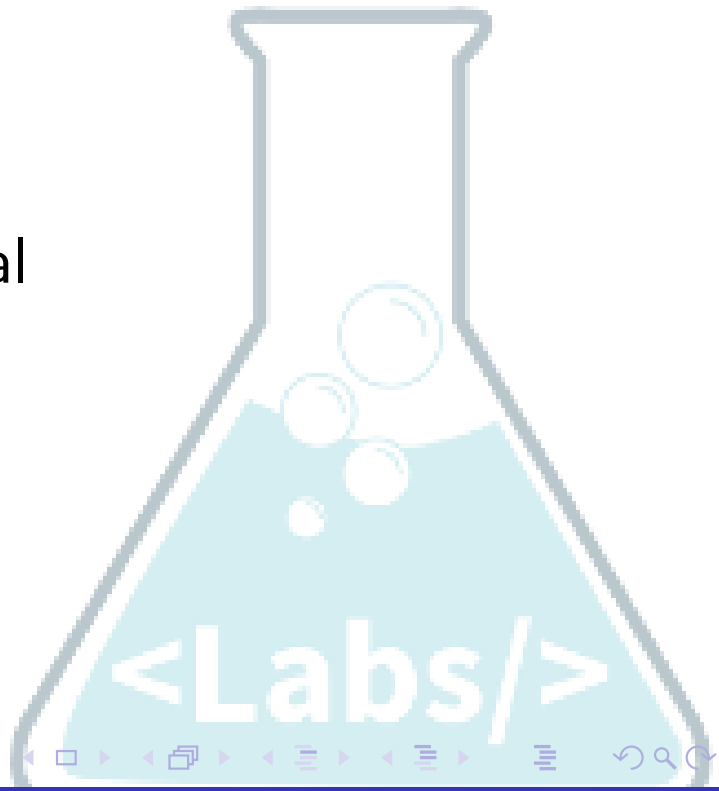
Unha vez que rematemos, para sair do modo iteractivo só temos que chamar á función *exit()*
As funcións e variables definidas nunha sesión do intérprete son borrados unha vez que se finaliza dita sesión.



Tipos básicos

Os tipos de datos básicos definidos por Python son os seguintes:

- Enteiros (int)
- Numeros en punto flotante (float)
- Números longos (long)
- Números en base octal e hexadecimal
- Números complexos (complex)
- Caracteres (char)
- Cadeas de caracteres (string)
- tuplas (tuple)

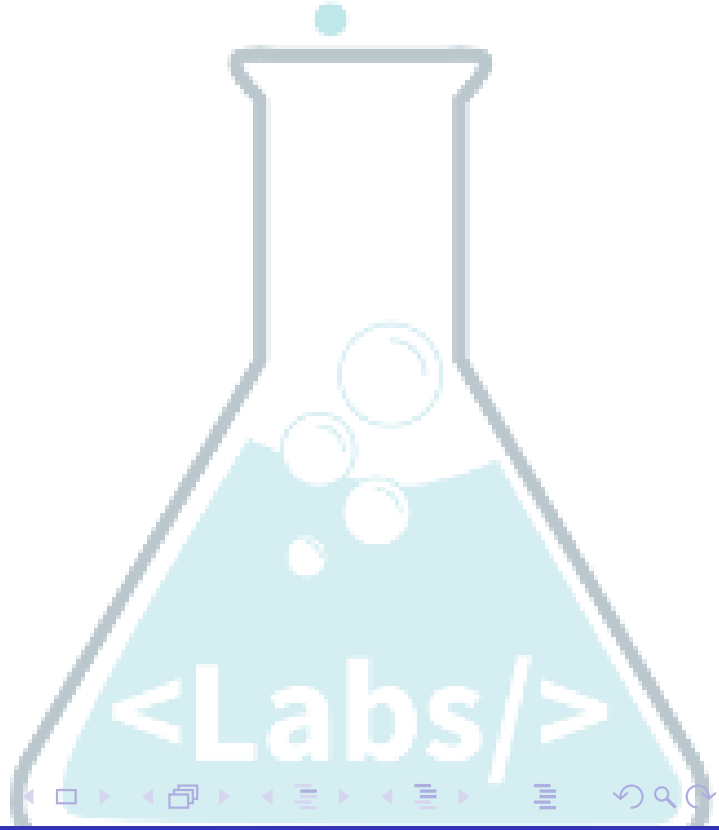


Operadores lóxicos

- and, &
- or, |
- not
- is, is not
- in, not in

Operadores matemáticos

- + (suma)
- - (resta)
- / (division)
- * (multiplicación)
- % (módulo)



Estructuras de datos

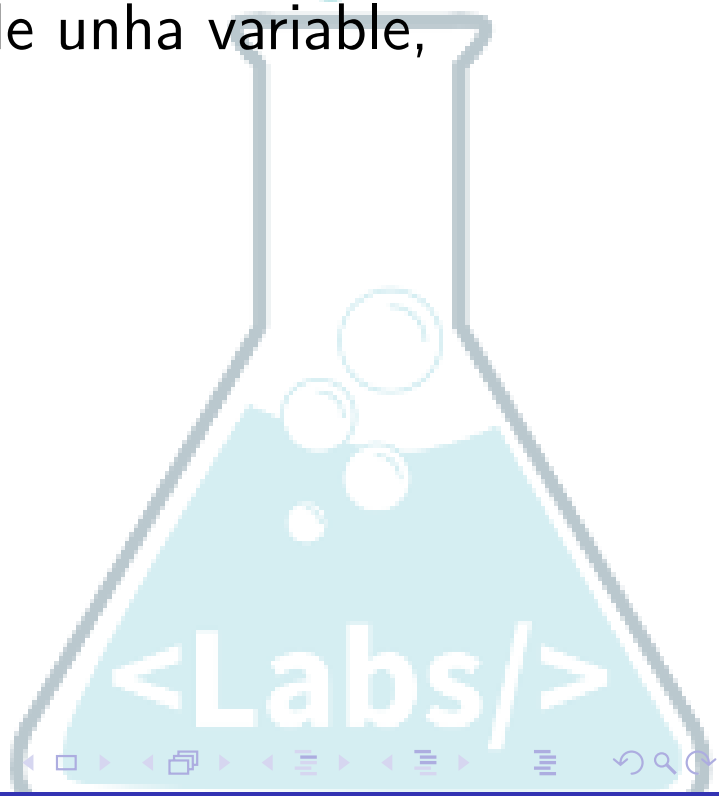
Ademais dos tipos de datos básicos, Python soporta de forma nativa varias estructuras de datos, das cales veremos as dúas máis empregadas.

- **Listas**
  Conxunto de tipos básicos(enteiros, numeros en punto flotante) ou compostos(tuplas), poden estar ordenados ou non.
  a = [1,2,3,4]
- **Diccionarios**
  Asocian un valor a unha clave para mellorar o acceso a un determinado elemento.
  alumnos = {'00001A':'Breixo','00002B':'Fran'}

Definir variables

O intérprete python ten incorporado un motor de inferencia, o que significa que cando declaramos unha variable non temos que declarar o seu tipo, xa se encarga o propio intérprete de inferilo. Se temos dudas acerca do typo de unha variable, podemos sabelo coa función *type()*

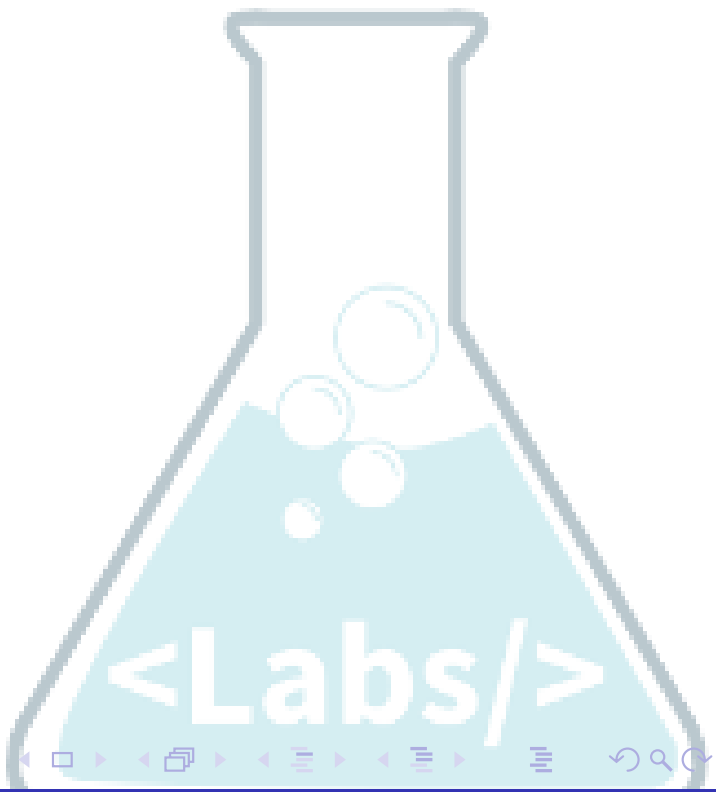
```
>>> sete = 7
>>> type(sete)
<type 'int'>
>>> a = "primeiro"
>>> type(a)
<type 'str'>
```



Funcións incorporadas (Built-in functions)

Python fai uso ten librerías por defecto, que non é necesario cargar, que proveen funcionalidades básicas, coma *print()*, que aplicado sobre un tipo devolve a súa representación en caracteres.

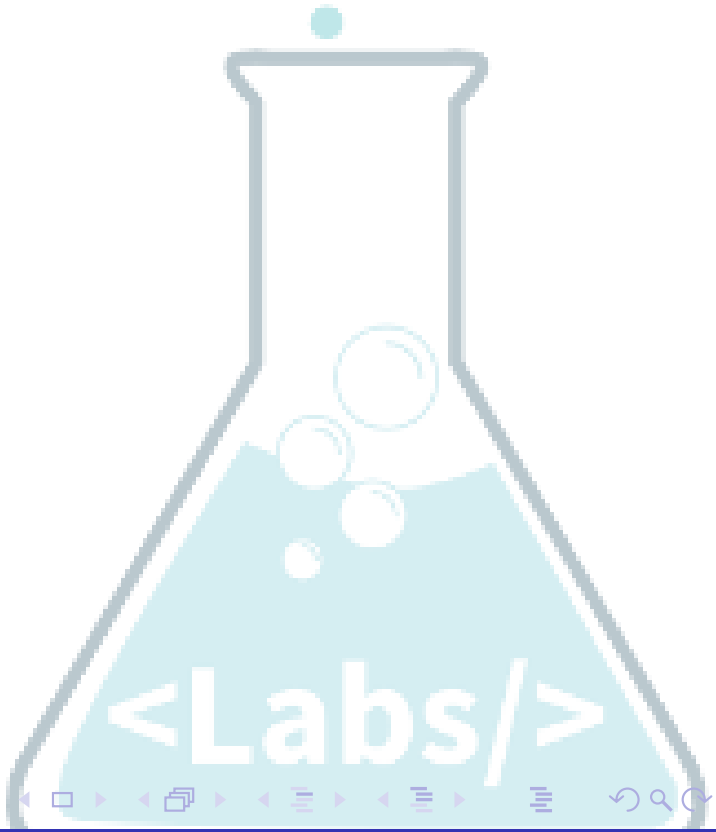
```
>>> n = "cadea"
>>> print(n)
cadea
>>> numero = 745
>>> print(numero)
745
>>> flotante = 3.14
>>> print(flotante)
3.14
```



Operacións sobre diccionarios

As operacións sobre direccionarios son proporcionados polas librerías básicas, xa que son un tipo moi empregado.

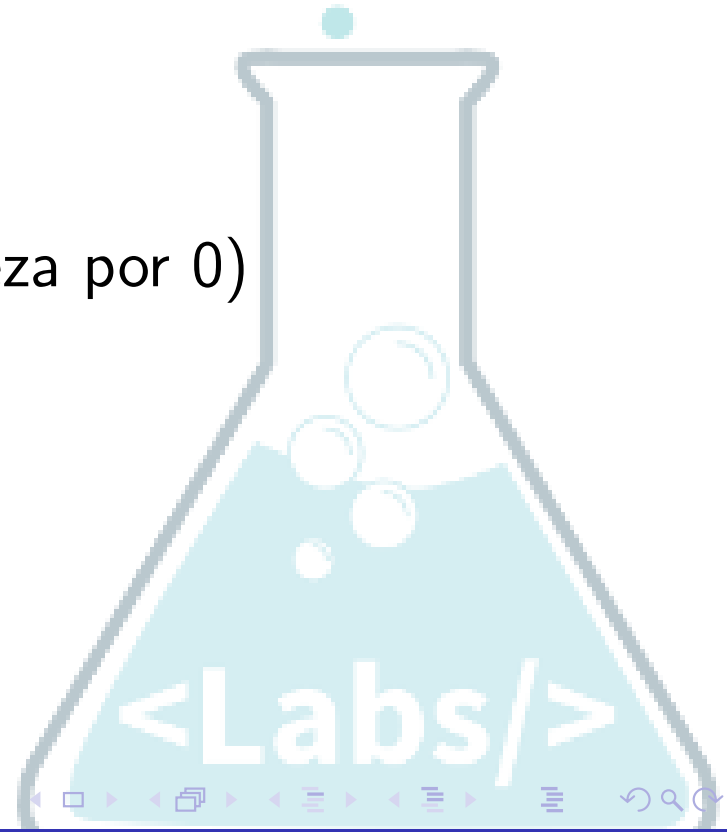
- Declaración
  d1 =
- Obter un elemento
  d1['clave']
- Engadir un elemento
  d1.update('clave':'valor')
- Borrar un elemento
  del d1['clave]



Operacións sobre listas

Ó igual que os diccionarios, as listas tamén son moi empregadas e as operacións sobre estas son amplamente soportadas.

- Declaración
  l = []
- Obter un elemento por índice (comeza por 0)
  l[7]
- Engadir un elemento o final da lista
  l.append(n)
- Borrar un elemento i
  del l[i]



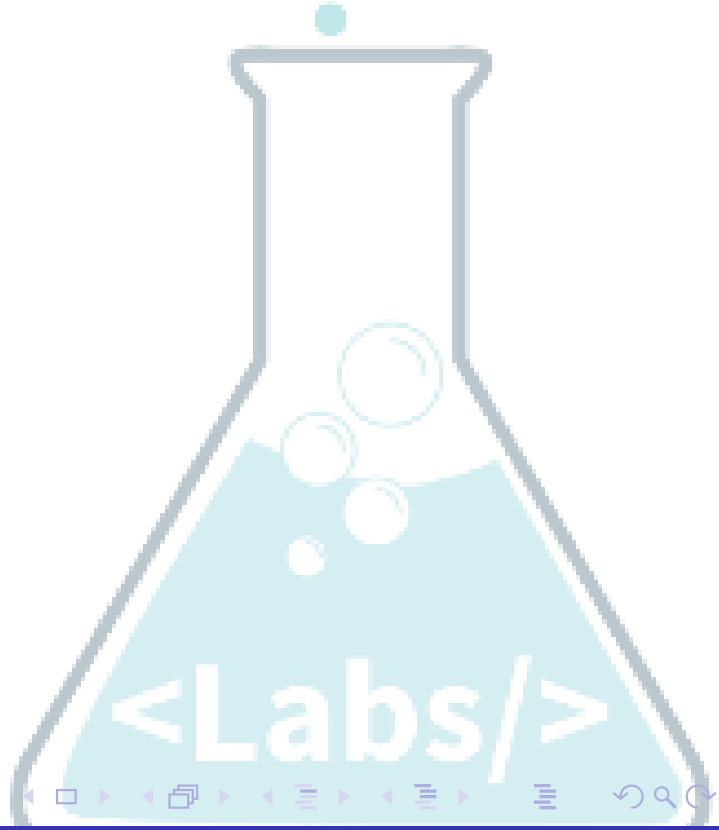
Definir funcións

Cando un fragmento de código é executado en numerosas ocasións, ou o código é extenso e complexo faise necesario estructuralo en funcións. En python as funcións declaranse da seguinte forma:

```
def <nome_funcion>(parametro1,parametro2,...,parametroN):
    sentencias dentro da función
```

Función de suma

```
>>> def suma(a,b):
...     return a+b
...
>>> suma(3,4)
7
```



Indentación

Como vimos ó principio, un dos obxectivos de Python é conseguir un código claro e lexible, para o cal fai obligatorio o uso da indentación.
Unha mala indentación cambia o significado do programa.

```
def access_control(name, pass):
  if authentication(name,pass) == 0:
    print(''Access denied'')
    exit()
  else:
    print(''Access granted'')
    grant_access()
```

```
def access_control(name, pass):
  if authentication(name,pass) == 0:
    print(''Access denied'')
    exit()
  else:
    print(''Access granted'')
  grant_access()
```

Estructuras de control

If

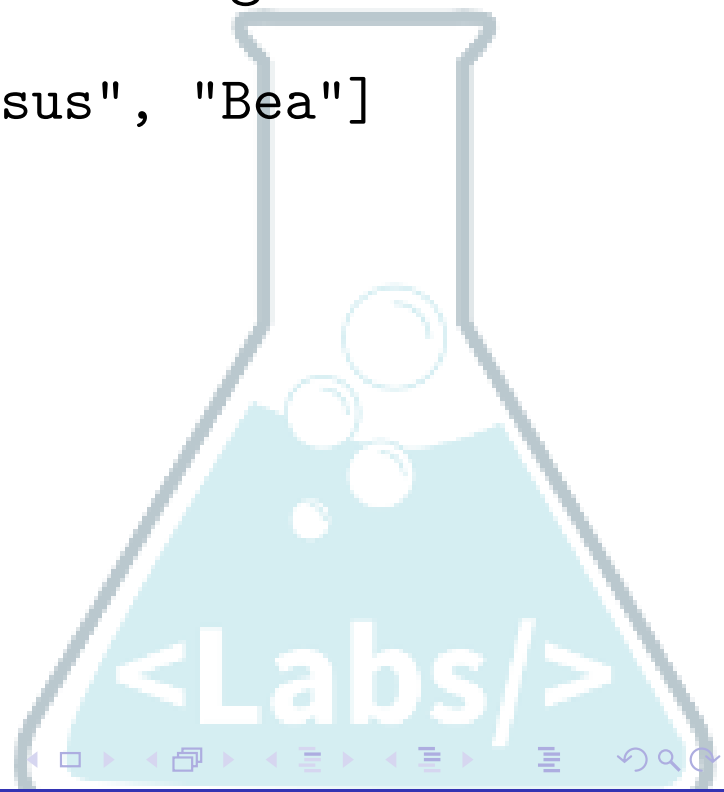
```
if (test1):
   print a
elif (test2 and test3):
   print b
   exit()
elif (test4):
   print(‘‘nada que facer’’)
else:
   print(‘‘Saindo’’)
```

Estructuras de control

Bucle for

Os bucles for son aplicados sobre listas. Esto quere decir que se queremos percorrer unha lista e aplicar unha función a cada elemento, ésto é posible mediante o seguinte código

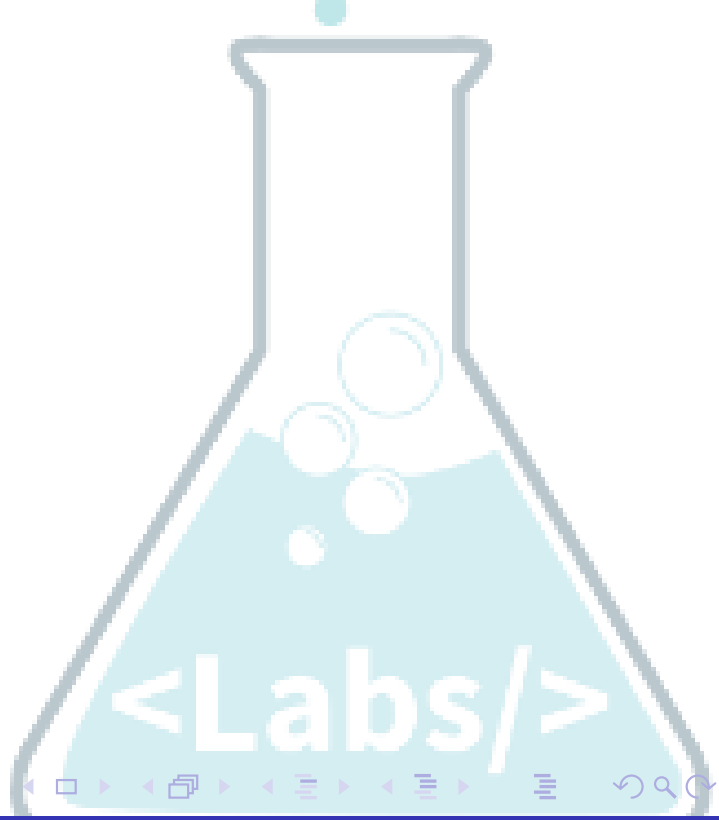
```
>>> alumnos = ["Xoan", "Marcos","Jesus", "Bea"]
>>> for alumno in alumnos:
...     print alumno
...
Xoan
Marcos
Jesus
Bea
>>>
```



Bucle for

Se queremos un bucle secuencial, podemos empregar a función *range*, que crea unha lista de elementos.

```
>>> for n in range(2,8):
...     print n
...
2
3
4
5
6
7
>>>
```



Estructuras de control

Bucle while

Igual que no resto de linguaxes de programación, o bucle while permítenos realizar unha acción repetitiva mentrs se cumpla unha condición, usando a a sentencia **break** para saír. Se non queremos saír do bucle, senón saltar as sentencias pendentes e executar o seguinte salto de bucle usamos **continue**.

```
while(true):
    if(test1):
        continue
    elif(test2):
        print("Hola")
    else:
        break
```

Ficheiros de código

Python é un programa de scripting, é decir, as instruccións son escritas nun ficheiro de texto plano, con extensión *.py* tal e como serían introducidas no intérprete.
A execución de un script consiste básicamente na lectura secuencial do ficheiro de texto e a execución das insctruccións correspondentes por parte do intérprete.
Existen dous tipos de programar en ficheiros: módulos e programas.

Módulos

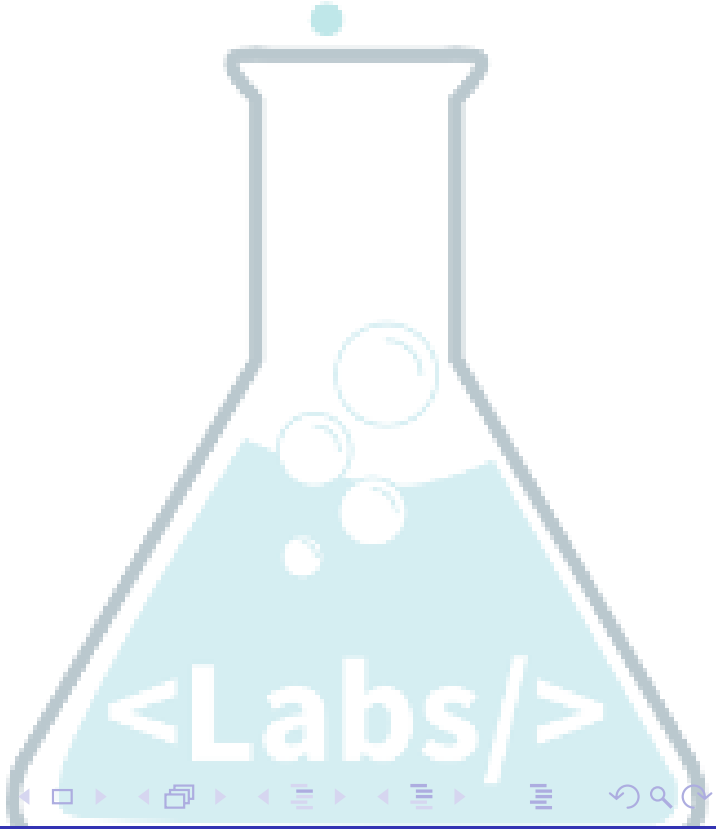
Os módulos son definicións de funcións. Se existen funcións que usamos reiteradamente, podemos definilas nun ficheiro con extensión *.py* e importalo despois mediante a orde **import**.
Os módulos proporcionan funcionalidades, pero non realizan accións de por sí, senon que son usados polos programas para axilizar o desenvolvemento de aplicacións.
Existen dúas formas de referenciar módulos, **import** e **from**

Modulo operacions.py

```
def suma (a,b):
    return a+b

def resta (a,b):
    return a-b

def devolve_maior (a,b):
    if (a>b):
        return a
    else:
        return b
```



import

A sentencia **import** carga todas as funcións do módulo ó que referenciamos, o cal pode ser ineficiente se non as empregamos.
Además, o programador debe especificar o espazo de nomes, é decir, referenciar o módulo ó que pertence a función que queremos empregar.

```
>>> import operacions
>>> suma(2,8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'suma' is not defined
>>> operacions.suma(2,8)
10
```

from

Mediante a orde **from**, podemos importar funcións concretas de un módulo e, además, importar o seu espazo de nomes, polo que non é necesario facer referencia ó módulo ó que pertence.

```
>>> from operacions import devolve_maior
>>> devolve_maior(15,8)
15
>>> operacions.devolve_maior(15,8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'operacions' is not defined
```

Scripts Python

Obviamente é interesante ter programas que se executen de forma que non sexa necesario o uso de unha sesión interactiva co intérprete. A estes ficheiros denomínase scripts, e, ó igual que os módulos, son leidos polo intérprete de forma secuencial. Neles podemos definir un conxunto de funcións e variables coas que construir unha secuencia de pasos a executar.
Para que se execute é necesario que exista unha función **main**, que é a última en ser declarada.
Cando o intérprete lee un arquivo de texto, se existe unha función main executa as ordes dentro dela como si se tratara dunha sesión interactiva, de forma que pode ser usada como disparador de un conxunto de accións.

Cabeceiras

Para executar un programa en python, tan só é necesario asegurarnos de que o arquivo ten permisos de execución e a continuación chamar ó interprete sobre o ficheiro.

```
python arquivo.py
```

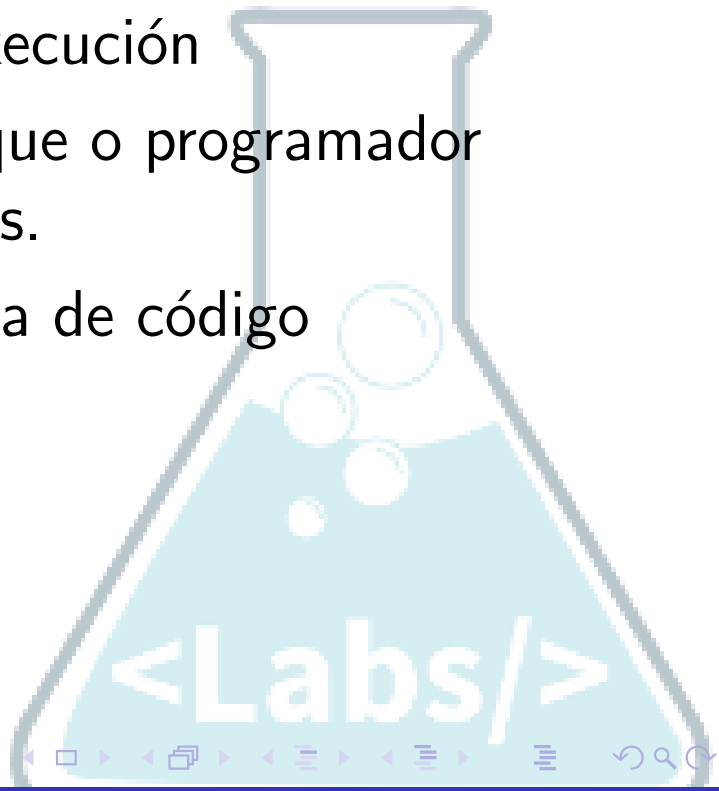
É posible engadir a seguinte liña para que o script sexa executado polo propio entorno de usuario.

```
#!/usr/bin/env python
```

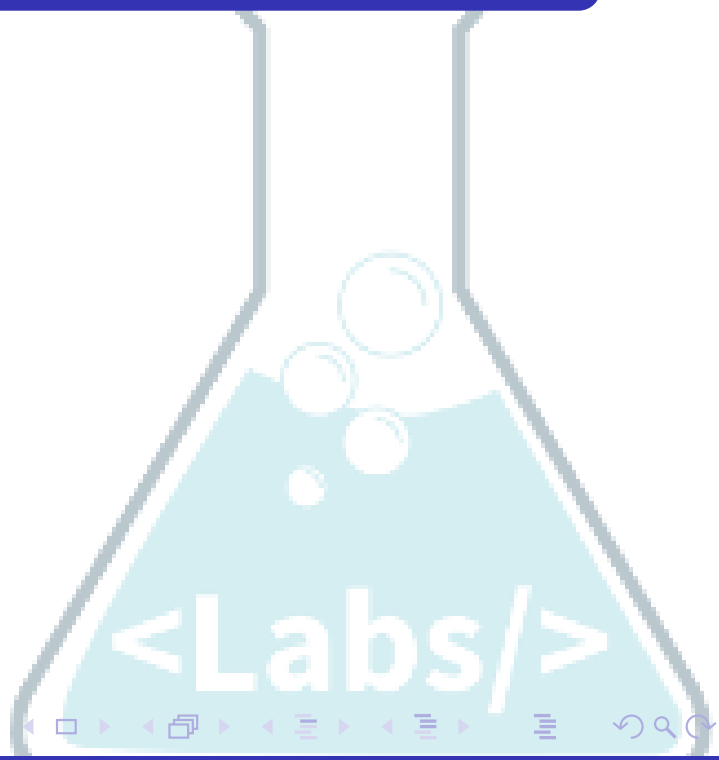
Tamén é recomendable, se o texto vai incluir carácteres que poden ser problemáticos coma 'ñ', ou tildes, especificar a codificación na que se gardou o ficheiro. O máis apropiado é facelo en UTF-8

Inconvintes

- Os erros teñen lugar en tempo de execución
- Como o tipado é dinamico, require que o programador preste maior atención a estos detalles.
- Rendemento, principalmente na carga de código



Control de versións



Fontes e referencias

Problemas de programador

- Trazabilidade do código
- Múltiples versións con múltiples funcionalidades
- Desenvolvemento continuo
- Control de versións
- Programación colectiva