Introducción a Python

Araguás, Gastón
Redolfi, Javier

Características de Python

Características de Python

- Simple
- Fácil de aprender
- Libre y de código abierto
- Lenguaje de alto nivel
- Portable
- Interpretado
- Orientado a objetos
- Extensible
- Embebible
- Con muchas librerías

Instalación en Linux

Debian

```
apt-get update && apt-get install python3
```

Ubuntu

```
sudo apt-get update && sudo apt-get install python3
```

Usando el intérprete

- Abrir una terminal, por ejemplo gnome-terminal
- En la terminal escribir python3 y luego presionar enter

```
python3
Python 3.6.4 (default, Jan 5 2018, 02:13:53)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola Mundo")
Hola Mundo
>>>
```

- Para salir del intérprete presionamos
  - Ctrl + d
  - exit() y enter

Intérprete interactivo

IPython

- Existe un intérprete interactivo llamado IPython
- Características principales:
  - Resaltado de líneas y errores con colores
  - Autocompletado de variables, módulos y atributos usando el tabulador (Tab)

Para instalarlo ejecutamos los siguientes comandos

```
apt-get update && apt-get install ipython3
```

Lo ejecutamos desde una consola con:

```
ipython3
```

Eligiendo un editor

- gedit
- kate
- vim
- emacs
- PyCharm

Usando un archivo fuente

- Crear un archivo **`hola.py`**
- Agregarle la línea siguiente y guardarlo

```
print("Hola Mundo")
```

- Abrir una terminal
- Nos movemos al directorio con el comando (donde user es nuestro usuario):

```
mkdir /home/user/borrar
cd /home/user/borrar/
```

- Ejecutamos el programa con

```
python hola.py
```

Práctica

- Instalar las librerías
- Instalar IPython
- Elegir un editor de texto
- Crear el programa **`hola.py`** que imprima “Hola Mundo”
- Ejecutarlo desde una consola

Comentarios

```
# Esto es un comentario
print('Hola Mundo') # Esto es otro comentario
```

Conceptos Básicos

Números

- Tenemos dos tipos de números
- enteros, por ejemplo 2
- flotantes, por ejemplo 3.23, 2.45 o 23.5e-4
- ambos son inmutables

```
entero0 = 2
entero1 = -2
flotante0 = 3.23
flotante1 = 23.5e-4
```

Conceptos Básicos

Strings

- Comillas simples
- Comillas dobles
- Comillas triples
- También son inmutables

```
'String con comillas simples'
"String con comillas dobles"
'''String con múltiples líneas
   y 3 comillas simples.'''
"""String con múltiples líneas
   y 3 comillas dobles."""
```

Strings - format

- Usado para construir strings desde otra información.

```
edad = 20
nombre = 'JuanP'
print('{0} tiene {1} años'.format(nombre, edad))
print(nombre + ' tiene ' + str(edad) + ' años')
print('{0:.3f}'.format(1.0/3)) # imprime 0.333
```

```
JuanP tiene 20 años
JuanP tiene 20 años
0.333
```

Conceptos Básicos

Variables

- Son identificadores usados para guardar cosas y luego poder usarlas.

Convención de nombres

- El primer carácter debe ser una letra, minúscula o mayúscula
- El resto puede ser letras, números o el carácter _
- Son sensitivos a mayúsculas o minúsculas
- Ejemplos válidos: `i`, `E`, `hola_1_3`
- Ejemplos no válidos: `2i`, `hola-1`

Indentación

- Los espacios en blanco al inicio de una línea son importantes
- Esto se llama indentación
- Estos espacios definen un nivel de indentación y todo lo que esté en ese nivel forma un bloque que se ejecuta junto

```
i = 5
# Error en la siguiente línea!
# Notar un espacio al inicio de la línea
 print('El valor es', i)
print('Repito, el valor es', i)
```

Si corremos el programa de arriba obtenemos un error parecido al siguiente:

```
IndentationError: unexpected indent
```

Conceptos Básicos

Indentación

- La indentación es similar al uso de las llaves de C/C++
- La recomendación oficial del lenguaje es usar 4 espacios para indentar

```
a = True
if(a is True):
    print('Es verdadero')
else:
    print('No es verdadero')
```

Operadores

```
# Suma
3 + 5 = 8
'a' + 'b' = 'ab'
# Resta
5 - 4 = 1
4 - 5 = -1
-4.3 = -4.3
# Multiplicación
3 * 5 = 15
3 * 'b' = 'bbb'
# Potencia
2 ** 3 = 8
# División
13 / 3 = 4.33333333333
# División y redondeo al entero menor
13 // 3 = 4
-13 // 3 = -5
9 // 1.8 = 4.0
# Módulo
13 % 2 = 1
```

Más Operadores

```
# Shift a la izquierda
3 << 2 = 12
# Shift a la derecha
16 >> 2 = 4
# AND
3 & 5 = 1
# OR
3 | 5 = 7
# XOR
3 ^ 5 = 6
```

Muchos más Operadores

```
# Menor que
3 < 2 = False
2 < 3 = True
# Mayor que
3 > 2 = True
2 > 3 = False
# Menor o igual que
3 <= 2 = False
2 <= 2 = True
# Mayor o igual que
3 >= 2 = True
3 >= 3 = True
# Igual que
3 == 3 = True
# Distinto que
3 != 3 = False
```

Los últimos operadores

```
# not booleano
not True = False
# and booleano
True and True = True
True and False = False
# or booleano
False or False = False
True or False = True
```

Operación y asignación en la misma sentencia

```
a = 2
a = a * 3
#a = 6
```

Esto puede ser escrito como:

```
a = 2
a *= 3
#a = 6
```

Operadores y Expresiones

Asociatividad

- Para asociar operaciones usamos paréntesis
- `2 + 3 + 4`
- `2 + (3 + 4)`

Control de Flujo

La sentencia if

- Se utiliza para chequear condiciones
- Las palabras reservadas que usa son: **`if`**, **`elif`** y **`else`**.

```
number = 23
guess = int(input('Ingrese un entero: '))

if guess == number:
    # El bloque comienza aquí
    print('Felicitaciones, adivinaste.')
    print('(pero no ganaste ningún premio!)')
    # El bloque termina aquí
elif guess < number:
    # Otro bloque
    print('No, es un poco mayor')
    # En un bloque se puede hacer lo que una quiera ...
else:
    print('No, es un poco menor')
    # Este bloque se ejecuta si: guessed > number

print('Hecho')
# Esta última sentencia siempre se ejecuta,
# después de que el if se ejecute.
```

Control de Flujo

La sentencia while

- Permite ejecutar un bloque de código varias veces
- Puede tener una claúsula opcional **`else`**

```
number = 23
running = True

while running:
    guess = int(input('Ingrese un entero: '))

    if guess == number:
        print('Felicitaciones, adivinaste.')
        # Esto causa que el buclec while termine
        running = False
    elif guess < number:
        print('No, es un poco mayor')
    else:
        print('No, es un poco menor')
else:
    print('El bucle while terminó.')

print('Hecho')
```

Control de Flujo

El bucle for

- La sentencia **`for..in`** itera sobre una secuencia de objetos, o sea sobre cada elemento de la secuencia
- Una secuencia es una colección ordenada de elementos

```
for i in range(1, 5):
    print(i)
else:
    print('El bucle for terminó')
```

```
1
2
3
4
El bucle for terminó
```

Control de Flujo

La sentencia break

- Termina un bucle aunque la condición sea verdadera (**`while`**) o aunque queden elementos para iterar
- Si hay una claúsula **`else`**, no se ejecuta

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    print('Length of the string is', len(s))
print('Done')
```

```
Enter something : Programming is fun
Length of the string is 18
Enter something : When the work is done
Length of the string is 21
Enter something : if you wanna make your work also fun:
Length of the string is 37
Enter something : use Python!
Length of the string is 11
Enter something : quit
Done
```

Control de Flujo

La sentencia continue

- Corta el flujo de la iteración actual y continua con la siguiente iteración

```
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
    if len(s) < 3:
        print('Too small')
        continue
    print('Input is of sufficient length')
    # Do other kinds of processing here...
```

salis cuando escribas el quit osea condicion verdadera (True)

si pones False el prgrma termina aunque que lo compiles

```
Enter something : a
Too small
Enter something : 12
Too small
Enter something : abc
Input is of sufficient length
Enter something : quit
```

Funciones

Funciones

- Son piezas de código reutilizable
- Se definen con la palabra clave **`def`**

```
def say_hello():
    # block belonging to the function
    print('hello world')
    # End of function

say_hello() # call the function
say_hello() # call the function again
```

```
hello world
hello world
```

Funciones

Parámetros de las funciones

- Los parámetros son variables que le pasamos a las funciones para que esta haga algo con ellos
- Se establecen entre los paréntesis de la declaración de la función
- Se separan con comas
- Los valores que pasamos como parámetros a la función se llaman argumentos

Funciones

```
def print_max(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')

# directly pass literal values
print_max(3, 4)

x = 5
y = 7
# pass variables as arguments
print_max(x, y)
```

```
4 is maximum
7 is maximum
```

Funciones

Valores por defecto de los argumentos

- Los valores por defecto se definen con el igual
- Cuando usamos la función estos parámetros pueden no estar definidos, en este caso se toman los argumentos por defecto
- Todos los parámetros con argumentos por defecto deben estar definidos al final de la lista de parámetros. Nunca pueden estar antes que parámetros sin argumentos por defecto.

```
def say(message, times=1):
    print(message * times)

say('Hello')
say('World', 5)
```

```
Hello
WorldWorldWorldWorldWorld
```

Funciones

Keyword arguments

- Si tenemos funciones con muchos parámetros, podemos pasar los argumentos de los que usamos
- Para pasarlos usamos el nombre del parámetro y le damos el argumento que queremos
- También podemos cambiar el orden de los parámetros, siempre que usemos el nombre del parámetro

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

```
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

Funciones

La sentencia return

- Se usa para salir de una función
- Y también para devolver un valor cuando volvemos de la función

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'The numbers are equal'
    else:
        return y

print(maximum(2, 3))
```

```
3
```

Práctico 1

- Crear una función **`adivinar`** que permita adivinar un número generado en forma aleatoria
  - El número debe estar entre 0 y 100
  - Este número se genera adentro de la función
  - Además debe recibir un parámetro que sea la cantidad de intentos y en caso de que esta cantidad de intentos sea superada el programa debe terminar con un mensaje
  - Si el usuario adivina antes de superar el número de intentos máximo, se debe imprimir un mensaje con el número de intentos en los que adivinó
- Después de crear la función, llamarla en el mismo archivo
- Ejecutar el script desde la consola

Ayuda: código para generar un número aleatorio

```
import random
numero = random.randint(0, 100)
```

Módulos

Módulos

- Los módulos sirven para reusar código ya escrito
- El método más simple para hacer un módulo es crear un archivo con la extensión **`.py`** con variables, funciones u objetos
- Para usar un módulo debemos importarlo

```
import sys

print('The command line arguments are:')
for i in sys.argv:
    print(i)
```

```
python module_using_sys.py we are arguments
The command line arguments are:
module_using_sys.py
we
are
arguments
```

Módulos

La sentencia from..import

- Sirve para importar en forma directa alguna variable, función u objeto que pertenece al módulo
- Nos evitamos escribir el nombre del módulo cada vez que usamos lo que importamos

```
import math
print("Square root of 16 is", math.sqrt(16))   # long

from math import sqrt
print("Square root of 16 is", sqrt(16))   # short
```

Módulos

Haciendo nuestros propios módulos

Creamos el archivo `mymodule.py`

```
def say_hi():
    print('Hi, this is mymodule speaking.')

__version__ = '0.1'
```

Creamos el archivo `mymodule_demo.py`

```
import mymodule

mymodule.say_hi()
print('Version', mymodule.__version__)
```

```
Hi, this is mymodule speaking.
Version 0.1
```

Módulos

Haciendo nuestros propios módulos

```
def say_hi():
    print('Hi, this is mymodule speaking.')

__version__ = '0.1'
```

Creamos el archivo `mymodule_demo2.py`

```
from mymodule import say_hi, __version__

say_hi()
print('Version', __version__)
```

```
Hi, this is mymodule speaking.
Version 0.1
```

Módulos

Paquetes

- Los paquetes se utilizan para organizar los módulos
- Un paquete es solamente una carpeta con un archivo llamado `__init__.py`

```
– <some folder present in the sys.path>/
    – world/
        – __init__.py
        – asia/
            – __init__.py
            – india/
                – __init__.py
                – foo.py
        – africa/
            – __init__.py
            – madagascar/
                – __init__.py
                – bar.py
```

```
import world
world.asia.india.population

from world.asia import india
india.population
```

Estructuras de Datos

Estructuras de datos

- Son estructuras para guardar datos agrupados
- Son usados para guardar una colección de datos relacionados
- Hay 4 estructuras de datos que vienen por defecto en python (built-in)
  - lista (list)
  - tupla (tuple)
  - diccionario (dict)
  - conjunto (set)

Estructuras de Datos

Listas

- Es una lista ordenada de items
- Los elementos de la lista se separan entre comas
- La lista de items se encierran con corchetes

Estructuras de Datos

```
shoplist = ['apple', 'mango', 'carrot', 'banana']

print('I have', len(shoplist), 'items to purchase.')

print('These items are:', end=' ')
for item in shoplist:
    print(item, end=' ')

print('\nI also have to buy rice.')
shoplist.append('rice')
print('My shopping list is now', shoplist)

print('I will sort my list now')
shoplist.sort()
print('Sorted shopping list is', shoplist)

print('The first item I will buy is', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('I bought the', olditem)
print('My shopping list is now', shoplist)
```

```
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my list now
Sorted shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is apple
I bought the apple
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

Estructuras de Datos

Tuplas

- Son similares a las listas, pero con menos funcionalidad, por lo tanto son más livianas
- Son inmutables, o sea que no se pueden modificar
- Los elementos de la tupla se separan entre comas
- y se encierran con paréntesis, aunque son opcionales

Estructuras de Datos

```
zoo = ('python', 'elephant', 'penguin')
print('Number of animals in the zoo is', len(zoo))

# parentheses not required but are a good idea
new_zoo = 'monkey', 'camel', zoo
print('Number of cages in the new zoo is', len(new_zoo))
print('All animals in new zoo are', new_zoo)
print('Animals brought from old zoo are', new_zoo[2])
print('Last animal brought from old zoo is', new_zoo[2][2])
print('Number of animals in the new zoo is',
      len(new_zoo)-1+len(new_zoo[2]))
```

```
Number of animals in the zoo is 3
Number of cages in the new zoo is 3
All animals in new zoo are ('monkey', 'camel', ('python',
    'elephant', 'penguin'))
Animals brought from old zoo are ('python', 'elephant', '
    penguin')
Last animal brought from old zoo is penguin
Number of animals in the new zoo is 5
```

Diferencias entre tuplas y listas

- Las listas se pueden modificar, las tuplas no
- Las listas ocupan más lugar que las tuplas
- Las listas tienen métodos para agregar, insertar, modificar, eliminar y ordenar valores, las tuplas no

Diccionarios

- Asocian claves (keys) con valores (values)
- Las claves deben ser únicas en un diccionario, no deben repetirse, para poder recuperar el valor
- Las claves deben ser objetos inmutables (incluso tuplas)
- Los valores pueden ser cualquier tipo de objeto
- Los distintos pares clave:valor de un diccionario se separan con comas
- El par clave:valor se separa con el carácter :
- Los diccionarios se definen (encierran) entre llaves

Diccionarios

- Asocian claves (keys) con valores (values)
- Las claves deben ser únicas en un diccionario, no deben repetirse, para poder recuperar el valor
- Las claves deben ser objetos inmutables (incluso tuplas)
- Los valores pueden ser cualquier tipo de objeto
- Los distintos pares clave:valor de un diccionario se separan con comas
- El par clave:valor se separa con el carácter :
- Los diccionarios se definen (encierran) entre llaves

Para pensar/investigar: ¿Para qué serviría una tupla como clave de un diccionario?

Estructuras de Datos

```
# 'ab' is short for 'a'ddress'b'ook
ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotmail.com'
}
print("Swaroop's address is", ab['Swaroop'])

# Deleting a key-value pair
del ab['Spammer']
print('\nThere are {} contacts\n'.format(len(ab)))

for name, address in ab.items():
    print('Contact {} at {}'.format(name, address))

# Adding a key-value pair
ab['Guido'] = 'guido@python.org'
if 'Guido' in ab:
    print("\nGuido's address is", ab['Guido'])
```

```
Swaroop's address is swaroop@swaroopch.com

There are 3 contacts in the address-book

Contact Swaroop at swaroop@swaroopch.com
Contact Matsumoto at matz@ruby-lang.org
Contact Larry at larry@wall.org

Guido's address is guido@python.org
```

Estructuras de Datos

Secuencias

- Las listas, tuplas y strings son secuencias
- Pueden ser iteradas con un **`for..in`**
- Tienen test de membresía: **`in`** o **`not in`**
- Soportan operaciones de indexación
- Además soportan la operación de rebanar (slice)

Estructuras de Datos

```
shoplist = ['apple', 'mango', 'carrot', 'banana']
name = 'swaroop'

# Indexing or 'Subscription' operation
print('Item 0 is', shoplist[0])
print('Item 1 is', shoplist[1])
print('Item 2 is', shoplist[2])
print('Item 3 is', shoplist[3])
print('Item -1 is', shoplist[-1])
print('Item -2 is', shoplist[-2])
print('Character 0 is', name[0])

# Slicing on a list
print('Item 1 to 3 is', shoplist[1:3])
print('Item 2 to end is', shoplist[2:])
print('Item 1 to -1 is', shoplist[1:-1])
print('Item start to end is', shoplist[:])

# Slicing on a string
print('characters 1 to 3 is', name[1:3])
print('characters 2 to end is', name[2:])
print('characters 1 to -1 is', name[1:-1])
print('characters start to end is', name[:])
```

Estructuras de Datos

```
Item 0 is apple
Item 1 is mango
Item 2 is carrot
Item 3 is banana
Item -1 is banana
Item -2 is carrot
Character 0 is s
Item 1 to 3 is ['mango', 'carrot']
Item 2 to end is ['carrot', 'banana']
Item 1 to -1 is ['mango', 'carrot']
Item start to end is ['apple', 'mango', 'carrot', 'banana']
characters 1 to 3 is wa
characters 2 to end is aroop
characters 1 to -1 is waroo
characters start to end is swaroop
```

Conjuntos

- Es una colección no ordenada de objetos
- Son usados cuando importa la existencia de un objeto más que el orden o cuantas veces aparece
- Podemos hacer test de membresía, chequear si es un subconjunto o calcular intersección

```
>>> bri = set(['brazil', 'russia', 'india'])
>>> 'india' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

Referencias

- Cuando creamos una variable y le asignamos un objeto, el objeto es solo una referencia, no es el objeto
- La variable apunta a la dirección de memoria en donde se encuentra el objeto (binding)

Estructuras de Datos

```
print('Simple Assignment')
shoplist = ['apple', 'mango', 'carrot', 'banana']
# mylist is just another name pointing to the same object!
mylist = shoplist

# I purchased the first item, so I remove it from the list
del shoplist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that both shoplist and mylist both print
# the same list without the 'apple' confirming that
# they point to the same object

print('Copy by making a full slice')
# Make a copy by doing a full slice
mylist = shoplist[:]
# Remove first item
del mylist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that now the two lists are different
```

Estructuras de Datos

```
Simple Assignment
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
Copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

Práctica Listas

- Crear la siguiente lista:

$$\begin{bmatrix} 2 & 2 & 5 & 6 \\ 0 & 3 & 7 & 4 \\ 8 & 8 & 5 & 2 \\ 1 & 5 & 6 & 1 \end{bmatrix}$$

- Seleccionar el subarray [8 8 5 2].
- Poner la diagonal de la matriz en cero.
- Sumar todos los elementos del array.
- Setear los valores pares en 0 y los impares en 1.

Práctico 2

Segmentando una imagen

Práctico 2

Segmentando una imagen

- Crear un programa que lea un imagen en blanco y negro
- Aplique un umbral sobre los valores de los pixeles de la misma
- Guarde el resultado en otra imagen
- No usar ninguna función de las OpenCV, excepto para leer y guardar la imagen

Práctico 2

Segmentando una imagen

- Crear un programa que lea un imagen en blanco y negro
- Aplique un umbral sobre los valores de los pixeles de la misma
- Guarde el resultado en otra imagen
- No usar ninguna función de las OpenCV, excepto para leer y guardar la imagen

Ayuda:

- Usar el template disponible en la filmina siguiente como base
- Antes de ejecutar el programa debemos instalar las OpenCV de la siguiente manera:

```
apt update && apt install libopencv-dev python3-opencv
```

Template

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

import cv2

img = cv2.imread('hoja.png', 0)

# Para resolverlo podemos usar dos for anidados
for row in img:
    for col in row:
        # Agregar código aquí

cv2.imwrite('resultado.png', img)
```