75.40 Algoritmos y Programación II Curso 4
Python

Dr. Mariano Méndez[1]

[1]Facultad De Ingeniería. Universidad de Buenos Aires

7 de agosto de 2020

1. Introducción

Java C y Python son lo tres lenguajes de alto nivel más utilizados según el índice TIOBE (consultado en noviembre 2019). El lenguaje de programación Pyton fue desarrollado por Guido van Rossum en la década de los 90. En 1991, van Rossum publicó el código de la versión 0.9.0.

Sus Características principales son :

- Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos:
  - programación orientada a objetos
  - programación imperativa
  - programación funcional
- Python usa tipado dinámico y conteo de referencias para la administración de memoria.
- Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).
- Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

1.0.1. Zen de Python

El Zen de Python es una colección de 20 principios de software que influyen en el diseño del Lenguaje de Programación Python, de los cuales 19 fueron escritos por Tim Peters1? en junio de 1999:

Bello es mejor que feo.

Explícito es mejor que implícito.

Simple es mejor que complejo.

Complejo es mejor que complicado.

Plano es mejor que anidado.

Espaciado es mejor que denso.

La legibilidad es importante.

Los casos especiales no son lo suficientemente especiales como para romper las reglas.

Sin embargo la practicidad le gana a la pureza.

Los errores nunca deberían pasar silenciosamente.

A menos que se silencien explícitamente.

Frente a la ambigüedad, evitar la tentación de adivinar.

Debería haber una, y preferiblemente solo una, manera obvia de hacerlo.

A pesar de que esa manera no sea obvia a menos que seas Holandés.

Ahora es mejor que nunca.

A pesar de que nunca es muchas veces mejor que *ahora* mismo.

Si la implementación es difícil de explicar, es una mala idea.

Si la implementación es fácil de explicar, puede que sea una buena idea.

Los espacios de nombres son una gran idea, ¡tengamos más de esos!

1.0.2. Instalación

En este apunte se estará utilizando python 3. Para instalara en un sistema unix-like se debe utilizar el gestor de paquetes de la distribución, dado que Ubuntu es la más utilizada, se muestran los pasos para instalar el interprete en esta distribución:

```
sudo apt-get update

sudo apt-get install python3.6
```

Para determinar que versión de python se está utilizando se debe ejecutar en una terminal:

```
darthmendez:new_system\:(master)$
$ python3 -V
Python 3.7.3
```

Dado que Python es un lenguaje interpretado se debe utilizar el interprete para poder utilizar el lenguaje de programación. Para ello se ejecuta desde una terminal:

```
$ python3
Python 3.7.3 (default, Oct  7 2019, 12:56:13)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. El Lenguaje

2.1. Hola Mundo

Posiblemente el clásico programa hola mundo, en python sea una de las versiones más simples:

```
print('hola munndo')
```

```
#include<stdio.h>

int main(){

    printf("hello world \n");
    return 0;
}
```

2.1.1. Input - Output

La acción que se utiliza para mostrar mensajes a la consola python es **print()** en el ejemplo anterior se puede ver claramente como se realiza una salida de datos por la consola python

```
print('imprime un literal')
```

por otro lado cuando se desea tomar de teclado un valor de entrada, se utiliza **input()**:

```
valor=input("ingrese un número")
```

2.2. Estructuras de Control

Como en todos los lenguajes de programación básicamente Python provee estructuras de control condicionales e iterativas.

2.2.1. Estructuras Condicional

Python solo provee la estructura if como estructura de control condicional que se basa en la validez de una expresion boolean:

```
if first condition:
    first body
elif second condition:
    second body
elif third condition:
    third body
else:
    fourth body
```

2.2.2. Estructuras Iterativs

Python posee dos tipos de estructuras de control iterativas. La primera el **while** basada en repeticiones sobre una expresión booleana que en algún momento debe dejar de ser verdadera. Y el segundo , basado en el conocimiento de la cantidad de iteraciones que hay que realizar **for**

```
while condition:
    body
```

por ejemplo:

```
c = 0
while c <= 5:
    c+=1
    print("c vale", c)
```

otro ejemplo:

```
indice = 0
while indice < 10:
    print(indice)
    indice+=1
```

El for es más conveniente utilizarlo cuando se debe recorrer una serie de elementos conocidos a priori:

```
for element in iterable:
    body
```

existe una función llamada range() -range (start, stop[, step])- que devuelve todos los valores de o al valor que se le pasa a range; por ejemplo el siguiente fragmento de código fuente imprime los valores del 0 al 6:

```
for i in range(6):
    print(i, ',')
```

```
for i in range(5, 10):
    print(i, ',')
```

```
for i in range(1, 10, 2):
    print(i,',')
```

2.3. Tipos de Datos

El tipado dinámico es la la desición de diseño de lenguajes de programación en el cual las variables no tienen tipo de dato hasta que no se les asigna un valor. Una vez asignado dicho valor la variable toma en correspondiente al valor almacenado.

Esto significa que no es necesario definir a priori que tipo de dato contendrá una variable, eso lo determinará en caso de python en la acción de asignación.

En python todos los tipos de datos son objetos. Los tipos de datos más comunes son :

- Numerico
- Booleano
- String
- Listas
- Diccionarios

2.3.1. Numéricos

- Enteros: Números naturales positivos o negativos.
- Floats: Cualquier número real que se represente con una parte entera y una parte decimal
- Complejos: Números con una parte real , otra imaginaria y j=-1 (x+yj)

2.3.2. Booleanos

El conjunto de los valores posibles de una variable boolean es = True,False

2.3.3. Strings

Un string en python es una secuencia de caracteres delimitada por dos comillas dobles o dos comillas simples:

```
'Esto es un string'
```

o

```
"Esto tambien es un string"
```

Cabe destacar que todos estos tipos de datos son objetos. Y,¿Que es un objeto?

2.3.4. Listas

En python una lista es una colección de objetos. definir una lista en python es tan sencillo coomo:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
```

Operaciones:

L.append(X)

L.extend(iterable)

L.sort(key=none,reverse=False)

L.reverse()

L.count()

L.insert(i,X)

L.remove(X)

L.pop()

L.clear()

2.3.5. Diccionarios

Un diccionario es un tipo de dato en el cual se almacenan pares (clave,valor). En python se crea un diccionario de la siguiente forma:

```
favorite_languages = { 1: 'python', 2: 'c', 3: 'ruby',  4: 'python'}
```

operaciones:

D.keys()

D.values()

D.items()

D.clear()

D.copy()

3. Lectura de Archivos

```
file=open("nombre","r")
lista= file.readlines()
print(lista)
file.close()
```

4. Funciones

Para definir una función en python es necesario un nombre y usar la palabra reservada def, por ejemplo:

```
def saludar():
    print("Hola! Este print se llama desde la función saludar()")

saludar()
```

la función anterior no posee argumentos, para agregarle argumentos a la función se lo hace dentro de los paréntesis:

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

resta(30, 10)
```

Existe una construcción interesante que se puede combinar con **return** que se puede ver en el siguiente ejemplo:

```
def es_par(numero):
    return True if (numero%2 ==0) else False

resta(30)
```

5. Clases y Objetos

Python es un lenguaje que muliparadigma. Uno de los paradigmas en el cual se suele utilizar con más fuerza la programación en Python es el Paradigma de la Programación Orientada a Objetos (OOP). En este paradigma se va un paso más que en el del estructurado en el cual los datos y las funciones que se pueden realizar sobre los mismos están separados, por ejemplo un clásico struct en C de algún TDA:

```
#ifndef SUBE_H
#define SUBE_H

#define MAX_VIAJES 3

struct Sube {
   char    dni[8];
   double  saldo;
} Sube_t;


Sube * sube_crear( char dni[8],  double saldo_inicial );

void sube_cargar(Sube* tarjeta, double monto);

double sube_saldo(Sube* tarjeta);

const char *  sube_documento( Sube * tarjeta);

void sube_realizar_viaje(Sube* tarjeta, double costo);

void sube_destruir( Sube * tarjeta);

#endif

```

En lenguajes orientados a objetos los datos y las funciones (métodos) que pueden ser ejecutados sobre estos conforman una única estructura una clase:

```
class Dog():

    """Un simple intento de modelar un perro."""

    def __init__(self, nombre, edad):
    """Initializa los atributos nombres y edad"""
        self._nombre = nombre
        self._edad = edad

    def sit(self):
    """Simula que el perro responde al comando sit"""
        print(self._name.title() + " is now sitting.")

    def dar_vueltas(self):
    """Simula que el perro responde al comando dar_vueltas."""
        print(self._name.title() + " rolled over!")
```

Referencias