Curso profesional de Python

Marcos Antonio Serrano M.

Módulo I - Estructura y elementos del Lenguaje

Marcos Antonio Serrano M.

Introducción a Python

- Apareció en 1991
- Python es un lenguaje de programación interpretado
- Su filosofía es hacer hincapié en una sintaxis que favorezca un código legible
- Es un lenguaje de programación multiparadigma (Orientado a objetos, imperativa y funcional)
- Es multiplataforma
- Es de código abierto
- Tiene una comunidad muy grande

Comentarios en Python

```
“””
        Hola mundo!
        Este comentario es multilineas
 “””
# Hola mundo
```

Variables en Python

Nombre_de_variable = valor o operación

A = 2

B = “Hola”

Tipos de datos

Numeros:

Tipo	Ejemplo
Int	23
Long	23L
Octal	027
hexadecimal	0x17

Tipos de datos

Cadenas:

- Comillas simples

  ‘Texto entre comillas simples’

- Comillas dobles

  “Texto con comillas dobles”

- Cadena con codigo escapes

  ‘Texto entre \n\tcomillas simples’

- Cadena multilinea

  “”” Testo linea1

  Linea 2

  .

  .

  “””

Tipos de datos

Boleanos:

- True
- False

Listas:

- A = [“coche”, “moto”, 21, 300]

Tuplas:

- (1, 2, 3, 4)

Diccionario:

- {“nombre”: “Pedro”,
  “Apellidos”: “Galindo”,
  “Edad”: 24}

Tipos de datos

Operador	Descripción	Ejemplo
+	Suma	a = 3 + 3 # resultado 6
-	Resta	a = 3 - 2 # resultado 1
	Negación	a = -3 # resultado -3
*	Multiplicación	a = 2 * 2 # resultado 5
**	Exponente	a = 2 ** 6 # resultado 12
/	División	a = 3.5 / 2 # resultado 1.75
//	Divisíon entera	a = 3.5 / 2 # resultado 1.0
%	Módulo	a = 7 % 2 # resultado 1

Operadores Aritméticos

Operador	Descripción	Ejemplo
==	¿Es igual a y b?	10 == 15 # False
!=	¿Es distinto a y b?	3 != 19 # True
<	¿Es a menor que b?	8 < 9 # True
>	¿Es a mayor que b?	10 > 40 # False
<=	¿Es a menor o igual que b?	10 <= 10 # True
>=	¿Es a mayor o igual que b?	15 >= 10 # True

Estructura de control de flujo e identación

- Identación
- PEP 8
- Encoding

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

- Asignación múltiple

```
a, b, c = 19, 'hola', False
```

Estructuras de control de flujo condicionales

Los condicionales nos permiten comprobar condiciones y hacer que nuestro programa se comporte de una forma u otra, que ejecute un fragmento de código u otro, dependiendo de esta condición

- If
- Else
- Elif

Estructuras de control iterativas

Nos permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición.

En Python se dispone de dos estructuras cíclicas:

- El bucle while
- El bucle for

Módulo II - Métodos principales del objeto string

Marcos Antonio Serrano M.

Metodos de formato

Metodo	Retorna
capitalize()	Copia de la cadena con la primera letra en mayúsculas.
lower()	Copia de la cadena en minúsculas
upper()	Copia de la cadena en mayúsculas.
swapcase()	Copia de la cadena convertidas las mayúsculas en minúsculas y viceversa.
title()	Copia de la cadena con la primera de cada palabra en mayúsculas.
center(longitud, "caracter de relleno")	Copia de la cadena centrada.

Metodos de formato

Metodo	Retorna
ljust(longitud, "caracter de relleno")	Copia de la cadena alineada a la izquierda.
rjust(longitud, "caracter de relleno")	Copia de la cadena alineada a la derecha.
zfill(longitud)	Copia de la cadena rellena con ceros a la izquierda hasta alcanzar la longitud final indicada.

Métodos de conversión

Los métodos de conversión nos permiten transformar un tipo de valor a otro tipo:

- int()
- str()
- float()
- tuple()
- list()

Métodos de Búsqueda

Metodo	Retorna
count(elemento)	Cantidad de apariciones de un elemento.
index(elemento, indice_inicio, indice_fin)	Numero de indice en el cual se encuentra la busqueda.

Métodos de Validación

Metodo	Retorna
startswith("subcadena" , posicion_inicio, posicion_fin)	Válida si una cadena comienza por una subcadena específica.
endswith("subcadena" , posicion_inicio, posicion_fin)	Válida si una cadena termina por una subcadena específica.
isalnum()	Válida si una cadena es alfanumérico.
isalpha()	Válida si una cadena tiene solo letras.
isdigit()	Válida si solo son números.
islower()	Válida si la cadena solo contiene minúsculas.

Métodos de Validación

Metodo	Retorna
isupper()	Válida si la cadena solo contiene mayúsculas.
isspace()	Válida si una cadena contiene solo espacios en blanco.
istitle()	Válida si una cadena tiene formato de título.

Métodos de Sustitución

Metodo	Retorna
format(*args, **kwargs)	Dar formato a una cadena, sustituyendo texto dinámicamente.
replace("subcadena a buscar", "subcadena por la cual reemplazar")	Busca una cadena específica y la reemplaza por otra.
strip("caracter")	Elimina caracteres a la derecha y a la izquierda de una cadena.
lstrip("caracter")	Elimina caracteres a la izquierda de una cadena.
rstrip("caracter")	Elimina caracteres a la derecha de una cadena.

Métodos de unión y división

Metodo	Retorna
join(iterable)	Unir una cadena de forma iterativa.
partition("separador")	Parte una cadena en tres, utilizando un separador.
split("separador")	Parte una cadena en varias partes, utilizando un separador.
splitlines()	Parte una cadena que contenga líneas.

Encoding

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

u = 'idzie wąż wąską dróżką'
uu = u.decode('utf8')
s = uu.encode('cp1250')
print(s)
```

Módulo III - Tipos de datos complejos

Marcos Antonio Serrano M.

Tuplas

- Las tuplas son secuencias o listas de elementos.
- No son mutables, es decir que no se pueden modificar.
- Su acceso y eso consume menos recurso por ende son mucho más rápido que las listas o diccionarios.

A = (‘elemento1’, 2, 3,)

Listas

- Las listas son compuestas por datos cuya cantidad o valor varían.
- Son mutables, es decir que se pueden modificar en cualquier momento.
- Están dotadas de una variedad de operaciones muy útiles.

A = [‘elemento1’, 2, 3,]

Listas

Metodo	Retorna
append("nuevo elemento")	Permite agregar un nuevo elemento.
extend(otra_lista)	Agregar varios elemento al final de la lista.
insert(posición, "nuevo elemento")	Agregar un elemento en la posición deseada.
pop()	Eliminar el último elemento de la lista.
pop(índice)	Eliminar el elemento por su índice.

Listas

Metodo	Retorna
remove("valor")	Elimina un elemento por su valor.
reverse()	Ordena una lista en reverso.
sort()	Ordena una lista en ascendente.
count(elemento)	Cuenta la cantidad de apariciones de un elemento.
index(elemento[, indice_inicio, indice_fin])	Nos da el índice donde se encuentra un elemento.

Listas

Metodo	Retorna
len(element)	Cuenta la cantidad de elementos de una lista.
max(lista)	El valor maximo.
min(lista)	El valor minimo.

Diccionarios

- Los diccionarios son mutables, es decir que se pueden modificar.
- Se construye por medio de {}
- Tienen una llave y un valor

Coche = {'puertas': 2, 'tipo': 2, 'matricula': ‘554CD’}

Diccionarios

Metodo	Retorna
diccionario.clear()	Vacía el diccionario.
dict.copy()	Copia un diccionario.
update(diccionario)	Concatenar diccionarios.
get(clave, "valor x defecto si la clave no existe")	Retorna el valor de un elemento buscado por su clave.

Diccionario

Metodo	Retorna
has_key(clave)	Permite saber si una clave existe.
iteritems() - items()	Obtener claves y valores de un diccionario.
keys()	Claves de un diccionario.
values()	Valores de un diccionario.
len(diccionario)	Cantidad de elementos de un diccionario.

Módulo IV - Funciones definidas

Marcos Antonio Serrano M.

Definiendo funciones

- Una función es una forma de agrupar expresiones o sentencias que realizan determinadas acciones.
- Las funciones se ejecutan solo cuando son llamadas.
- Las funciones pueden recibir y retornar valores.

Definiendo funciones

```
def mi_funcion():
    # aquí el algoritmo
```

Para ejecutar la función solo tenemos que llamarla por su nombre:

```
mi_funcion()
```

Definiendo funciones

Las funciones pueden recibir parámetros y estos pueden tener un valor por defecto.

```
def mi_funcion(a, b):
        # aquí el algoritmo

def funcion_a(a=1, b=2):
        # mi codigo
```

Definiendo funciones

- Parámetros arbitrarios.

```
def mi_funcion(parametro_fijo, *arbitrarios):
    print parametro_fijo

    # Los parámetros arbitrarios se corren como tuplas o listas
    for argumento in arbitrarios:
        print argumento

mi_funcion(0, 'arbitrario 1', 'arbitrario 2', 'arbitrario 3')
```

Definiendo funciones

```
def mi_funcion(parametro_fijo, *arbitrarios, **kwords):
    print(parametro_fijo)
    for argumento in arbitrarios:
        print(argumento)

    # Los argumentos arbitrarios tipo clave, se recorren como los diccionarios
    for clave in kwords:
        print("El valor de", clave, "es", kwords[clave])

mi_funcion(0, "arbitrario 1", "arbitrario 2", "arbitrario 3", clave1="valor uno",
clave2="valor dos")
```

Definiendo funciones

* args ----------> Tuplas

**kworks ----------> Diccionario

Anteponiendo los asteriscos desempaquetamos las estructuras para que puedan ser recibidas por una función

Definiendo funciones

Las funciones de locals() y globals() retornan un diccionario.

```
def funcion():
    return "Hola Mundo"


def llamada_de_retorno(func=""):
    """Llamada de retorno a nivel global"""
    return globals()[func]()

print(llamada_de_retorno('funcion'))

# Llamada de retorno a nivel local
nombre_de_la_funcion = 'funcion'
print(locals()[nombre_de_la_funcion]())
```

Definiendo funciones

```
def mi_funcion():
    print('hola mundo!')


callable(mi_funcion)
True

'mi_funcion' in globals()
True

'mi_funcion' in locals()
True
```

Definiendo funciones

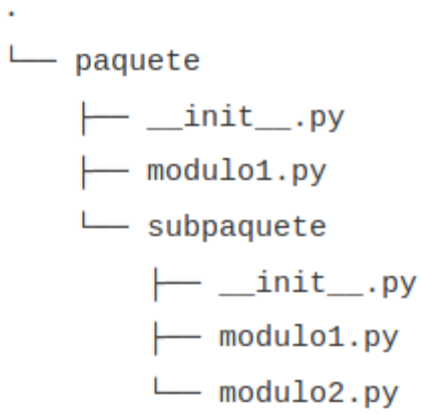
```
In [37]: def mi_fun(a,b):
    ...:     if b > 0:
    ...:         b-=1
    ...:         mi_fun(a,b)
    ...:     print(a,b)
    ...:

In [38]: mi_fun(5,5)
5 0
5 0
5 1
5 2
5 3
5 4
```

Módulo V - Módulos, Paquetes y Namespaces

Marcos Antonio Serrano M.

Módulos y Paquetes



Importar paquetes y Alias

```
import modulo                 # importar un módulo que no pertenece a un paquete

import paquete.modulo1        # importar un módulo que está dentro de un paquete

import paquete.subpaquete.modulo1
```

Para agregar un alias solo tendríamos que usar ‘ as ‘ y el nombre del alias

```
import modulo as mi_modulo
```

Colores en nuestra terminal

Estilos*	Cod. ANSI
Sin efecto	0
Negrita	1
Débil	2
Cursiva	3
Subrayado	4
Inverso	5
Oculto	6
Tachado	7

\033[cod_formato;cod_color_texto;cod_color_fondor

```
print("\033[4;35m"+"Texto de ejemplo")
```

Colores	Texto	Fondo
Negro	30	40
Rojo	31	41
Verde	32	42
Amarillo	33	43
Azul	34	44
Morado	35	45
Cian	36	46
Blanco	37	47

Módulo VI - Programación orientada a objetos

Marcos Antonio Serrano M.

Programación orientada a objetos

La Programación Orientada a Objetos (POO), **es un paradigma de programación.**

Teoría que nos suministra la base y el modelo para resolver problemas.

Programación orientada a objetos

¿Que es un un objeto?

- Un objeto es una cosa, es todo lo que nos rodea.
  - Un carro
  - Una casa
  - Tu ordenador
  - Tu

Todos los objetos tienen cualidades o atributos:

- Es **color** verde
- **Pesa** 3 kilos

Programación orientada a objetos

Objeto	Atributo	Cualidad del atributo
(el) Objeto	Color	verde
(el) Objeto	Tamaño	180
(el) Objeto	Edad	35

- Un objeto puede estar compuesto por otros objetos
- Un objeto puede compartir características de otro objeto
- Un objeto puede hacer cosas, realizar acciones

Programación orientada a objetos

Las clases son los modelos sobre los cuáles se construirán nuestros objetos.

```
class Objeto:
    pass

class Antena:
    pass

class Pelo:
    pass
```

El nombre de las clases se define en singular, utilizando CamelCase.

Programación orientada a objetos - Atributos

```
class Pelo():
    color = ""
    textura = ""

class Ojo():
    forma = ""
    color = ""
    tamanio = ""

class Objeto():
    color = ""
    ojos = Ojo()       # propiedad compuesta por el objeto objeto Ojo
    pelos = Pelo()     # propiedad compuesta por el objeto objeto Pelo
```

Programación orientada a objetos - Métodos

```
class Objeto(object):
    color = ""
    tamanio = ""
    edad = ""

# NuevoObjeto sí hereda de otra clase: Objeto
class NuevoObjeto(Objeto):
    sexo = ""

    def get_sexo(self):
        return self.sexo

objeto = NuevoObjeto()
print(objeto.get_sexo())
```

Programación orientada a objetos - Métodos especiales

__init__

- Este es el método constructor. Este método se va a ejecutar cada vez que se cree una nueva instancia de la clase.

__str__

- El método __str__ nos devuelve un string que se mostrará al llamar nuestro objeto.