



Curso profesional de Python

Módulo I - Estructura y elementos del Lenguaje

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 código escapes
 'Texto entre \n\tcomillas simples'
- Cadena multilinea
 """ Texto linea1
 Linea 2
 .
 .
 """

Tipos de datos

Booleanos:

- 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
//	División 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 indentación

- 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

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
<code>ljust(longitud, "caracter de relleno")</code>	Copia de la cadena alineada a la izquierda.
<code>rjust(longitud, "caracter de relleno")</code>	Copia de la cadena alineada a la derecha.
<code>zfill(longitud)</code>	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
<code>format(*args, **kwargs)</code>	Dar formato a una cadena, sustituyendo texto dinámicamente.
<code>replace("subcadena a buscar", "subcadena por la cual reemplazar")</code>	Busca una cadena específica y la reemplaza por otra.
<code>strip("caracter")</code>	Elimina caracteres a la derecha y a la izquierda de una cadena.
<code>lstrip("caracter")</code>	Elimina caracteres a la izquierda de una cadena.
<code>rstrip("caracter")</code>	Elimina caracteres a la derecha de una cadena.

Métodos de unión y división

Metodo	Retorna
<code>join(iterable)</code>	Unir una cadena de forma iterativa.
<code>partition("separador")</code>	Parte una cadena en tres, utilizando un separador.
<code>split("separador")</code>	Parte una cadena en varias partes, utilizando un separador.
<code>splitlines()</code>	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

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
<code>remove("valor")</code>	Elimina un elemento por su valor.
<code>reverse()</code>	Ordena una lista en reverso.
<code>sort()</code>	Ordena una lista en ascendente.
<code>count(elemento)</code>	Cuenta la cantidad de apariciones de un elemento.
<code>index(elemento[, indice_inicio, indice_fin])</code>	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
<code>diccionario.clear()</code>	Vacía el diccionario.
<code>dict.copy()</code>	Copia un diccionario.
<code>update(diccionario)</code>	Concatenar diccionarios.
<code>get(clave, "valor x defecto si la clave no existe")</code>	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

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 código
```

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, **kwargs):  
    print(parametro_fijo)  
    for argumento in arbitrarios:  
        print(argumento)
```

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

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

Definiendo funciones

* args -----> Tuplas

**kwargs -----> 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

Módulos y Paquetes

```
.  
└─ paquete  
    ├── __init__.py  
    ├── modulo1.py  
    └─ subpaquete  
        ├── __init__.py  
        ├── modulo1.py  
        └─ modulo2.py
```

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`

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

Colores en nuestra terminal

```
\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

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 = ""  
    tamaño = ""
```

```
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 = ""  
    tamaño = ""  
    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.

Programación orientada a objetos - Herencia

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

```
class NuevoObjeto(Objeto):  
    sexo = ""  
  
    def get_sexo(self):  
        return self.sexo
```

Módulo VII - Conversión de tipos

Conversión y tipos

- Concatenación simple de colecciones
- Valor máximo y mínimo
- Contar elementos

Módulo VIII - Ficheros

Modos de apertura

Indicador	Modo de apertura	Ubicación del puntero
r	Solo lectura	Al inicio del archivo
rb	Solo lectura en modo binario	Al inicio del archivo
r+	Lectura y escritura	Al inicio del archivo
rb+	Lectura y escritura en modo binario	Al inicio del archivo
w	Solo escritura. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo

Modos de apertura

Indicador	Modo de apertura	Ubicación del puntero
wb	Solo escritura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
w+	Escritura y lectura. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
wb+	Escritura y lectura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
a	Añadido (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo

Modos de apertura

Indicador	Modo de apertura	Ubicación del puntero
ab	Añadido en modo binario (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
a+	Añadido (agregar contenido) y lectura. Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
ab+	Añadido (agregar contenido) y lectura en modo binario. Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo

Métodos del objeto File

El objeto File, tiene muchos métodos pero los más frecuentes son:

- **seek(byte)** Mueve el puntero hacia el byte indicado
 Archivo = open('ejemplo.txt', 'r')
 contenido = Archivo.read()
 Archivo.seek(0) el puntero queda al final del documento
- **read()** Lee todo el contenido del fichero. Si se le pasa la longitud de bytes, leerá solo el contenido hasta la longitud indicada.
 Archivo = open('ejemplo.txt', 'r')
 contenido = Archivo.read()
 print(contenido)

Métodos del objeto File

- **readline()** Lee una línea del archivo.
 Archivo = open('ejemplo.txt', 'r')
 linea = Archivo.readline()
 print(linea)
- **readlines()** Lee todas líneas de un archivo.
 Archivo = open('ejemplo.txt', 'r')
 for linea in Archivo.readlines():
 print(linea)

Métodos del objeto File

- **tell()** Retorna la posición actual del puntero..
 Archivo = open('ejemplo.txt', "
 linea = Archivo.readline()
 leer_mas = Archivo.read(Archivo.tell() *2)
 print(leer_mas)
- **write()** Escribe cadena dentro del archivo.
 archivo = open('ejemplo.txt', 'r+')
 archivo.write('mi nueva linea')

Métodos del objeto File

- **writelines(secuencia)** Escribe una secuencia dentro del archivo. Puede ser cualquier iterable.

```
lista = ['hola', 'chao']  
archivo = open('ejemplo.txt', 'r+')  
archivo.writelines(lista)
```

- **close()** Cierra un archivo.
archivo.close()

Métodos del objeto File

Se pueden acceder a las siguientes propiedades del objeto file:

- **closed**: retorna True si el archivo se ha cerrado. De lo contrario, False.
- **mode**: retorna el modo de apertura.
- **name**: retorna el nombre del archivo.
- **encoding**: retorna la codificación de caracteres de un archivo de texto.

Métodos del objeto File

Para cerrar de forma automática podemos usar with

```
with open("remeras.txt", "r") as archivo:  
    contenido = archivo.read()
```

```
print archivo.closed  
# True
```

Ficheros JSON

JSON, acrónimo de **JavaScript Object Notation**, es un formato de texto ligero para el intercambio de datos.

Para hacer uso de ficheros Json importamos la librería necesaria.

```
import json
```

Ficheros JSON

```
import json
```

```
d = {"nombre": "Pepe", "edad": 18, "activo": True}
```

```
# Para convertir d en una cadena Json usamos dumps
```

```
json.dumps(d)
```

```
# Si queremos decodificar un Json podemos usar loads
```

```
data = json.loads('[null, true, false, "Hola, mundo!"]')
```

Codificar y decodificar ficheros JSON

```
# Para escribir Json nos provee de dump
data = [True, False, None, 'Hola, mundo!']
with open("data.json", "w") as f:
    json.dump(data, f)
```

```
# Para leer o recuperar Json nos permite usar load
with open("data.json") as f:
    data = json.load(f)
```

Ficheros JSON

```
data = [True, False, {"nombre": "Juan"}]  
print(json.dumps(data, indent=4))  
[  
    true,  
    false,  
    {  
        "nombre": "Juan"  
    }  
]
```

Módulo IX - Excepciones

Manejo de Excepciones

```
dividendo = 10  
divisor = 0  
dividendo / divisor
```

```
try:  
    cociente = dividendo / divisor  
except:  
    print('esto es un error controlado')
```

Manejo de Excepciones

```
dividendo = 10  
divisor = 0  
dividendo / divisor
```

try:

```
    cociente = dividendo / divisor
```

except:

```
    print('esto es un error controlado')
```

Manejo de Excepciones

try:

aquí ponemos el código que puede lanzar excepciones

except ZeroDivisionError:

entrará aquí en caso que se haya producido

una excepción ZeroDivisionError

except:

entrará aquí en caso que se haya producido

una excepción que no corresponda a ninguno

de los tipos especificados en los except previos

Manejo de Excepciones

try:

aquí ponemos el código que puede lanzar excepciones

except ZeroDivisionError as e:

e nos dara informacion de la excepción

Por otra parte podemos seguir devolviendo la excepción o incluso otra con más información.

except ZeroDivisionError:

raise ZeroDivisionError("El divisor no puede ser cero")

Módulo X - Bases de datos

Bases de datos

Una base de datos es un “**almacén**” que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente.

Bases de datos

Entre las principales características de los sistemas de base de datos podemos mencionar:

- Independencia lógica y física de los datos.
- Redundancia mínima.
- Acceso concurrente por parte de múltiples usuarios.
- Integridad de los datos.
- Consultas complejas optimizadas.
- Seguridad de acceso y auditoría.
- Respaldo y recuperación.
- Acceso a través de lenguajes de programación estándar.

Bases de datos

- Existen bases de datos relacionales y no relacionales o No-SQL
- Existen muchos motores de bases de datos relacionales los más famosos son:



ORACLE



Bases de datos - Tipos de datos

Tipo de dato	Denominación	Parametros	Ejemplo
Entero	INT(N)	N = cantidad de dígitos.	INT(10)
Decimal	DECIMAL(N, D)	N = cantidad de dígitos totales D = cantidad de decimales	DECIMAL(10, 2)
Booleano	BOOL		BOOL

Bases de datos - Tipos de datos

Tipo de dato	Denominación	Parametros	Ejemplo
Fecha	DATE		DATE
Fecha y hora	DATETIME		DATETIME
Fecha y hora automática	TIMESTAMP		TIMESTAMP
Hora	TIME		TIME

Bases de datos - Tipos de datos

Tipo de dato	Denominación	Parametros	Ejemplo
Año	YEAR(D)	D = cantidad de dígitos (2 o 4)	YEAR(4)
Cadena de longitud fija	CHAR(N)	N = longitud de la cadena (entre 0 y 255)	CHAR(2)
Cadena de longitud variable	VARCHAR(N)	N = longitud máximo de la cadena (entre 0 y 65532)	VARCHAR(100)
Bloque de texto	BLOB		BLOB

Bases de datos - Sintaxis

```
CREATE DATABASE ejemplo;
```

```
CREATE TABLE productos(  
    producto VARCHAR(125),  
    descripcion BLOB,  
    precio DECIMAL(6, 2),  
    en_stock BOOL  
    estado BOOL  
);
```

```
INSERT INTO  
    productos(producto, precio, en_stock)  
VALUES('Bolsa de dormir para alta  
montaña', 234.65, TRUE);
```

```
SELECT producto, precio FROM productos;
```

Módulo XI - Django

Comandos basicos - Django

1. `python manage.py runserver` # Este comando es para arrancar el servidor
1. `python manage.py makemigrations` # Para generar fichero de migraciones
1. `python manage.py migrate` # Para hacer migraciones a base de datos