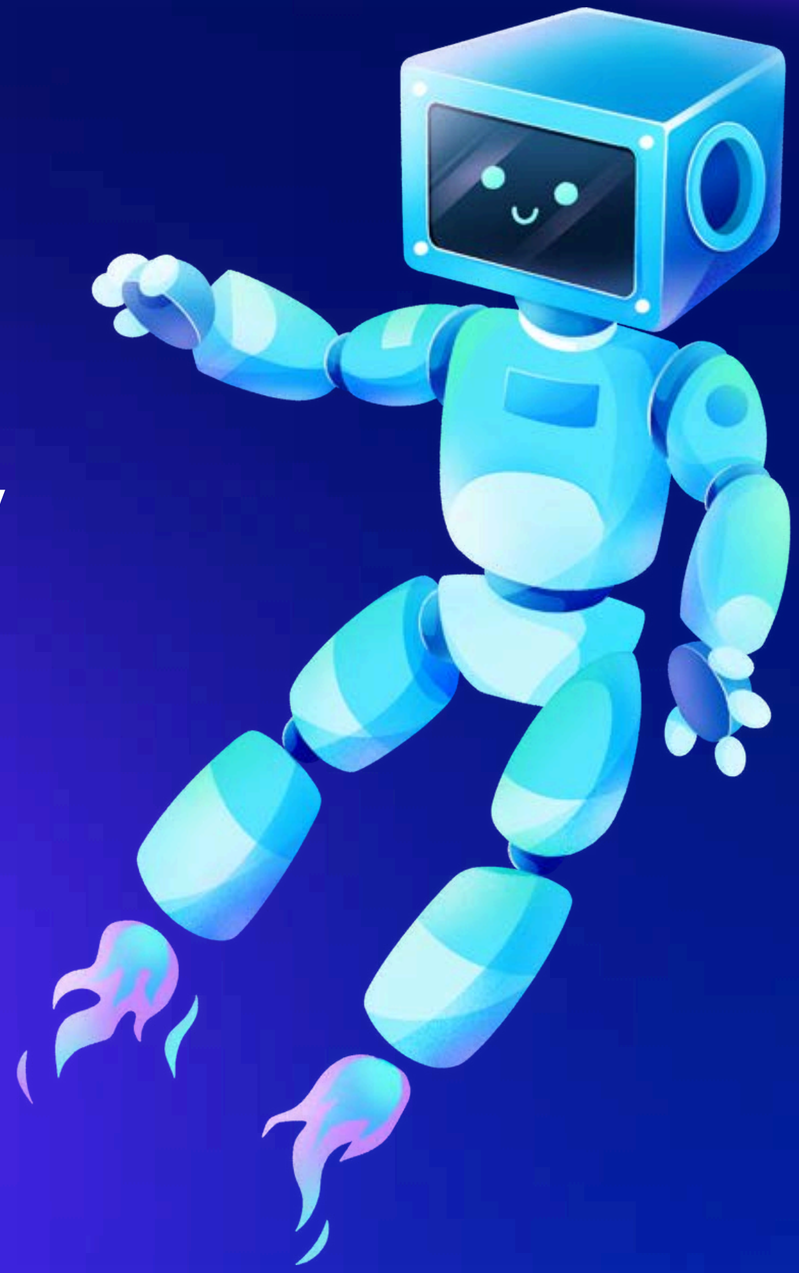
Python Básico

Manejo de Archivos y Estructuras de Datos Avanzadas



Presentación realizada por Grupo 5 de Factoria F5

CONTENIDO

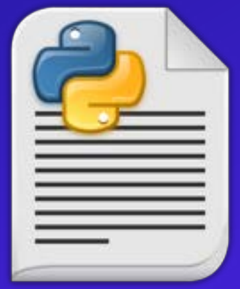
Manejo de Archivos y Estructuras de Datos Avanzadas

- Manejo de archivos: Apertura, lectura, escritura y cierre de archivos.
- Diccionarios: Creación, acceso y manipulación de pares clave-valor.
- Conjuntos: Operaciones básicas con sets.
- Manejo de excepciones: Try, except, else, finally

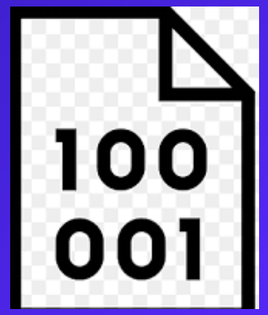
Manejo de archivos en Python

Un archivo es un contenedor de información.

Python puede trabajar con varios tipos de archivos y ficheros.



ARCHIVOS DE TEXTO



ARCHIVOS BINARIOS

ARCHIVOS CSV

Archivos de texto

Estos archivos contienen texto sin formato y se pueden leer o escribir directamente con Python.

La función *open()*

Es una función incorporada que abre un archivo y permite que tu programa tenga acceso a él.

```
OPEN(<ARCHIVO>,<MODO>)
```

Parámetros función *open()*

buffering: Es un entero opcional que configura la política de buffering.

encoding: Es el nombre de la codificación empleada con el fichero.

errors: Es una cadena opcional que especifica como deben manejarse los errores de codificación y descodificación

newline: Este parámetro define cómo analizar los saltos de líneas, para los .txt.

Métodos de acceso

(‘r’):	Abre archivos de texto solo para lectura (por defecto).
(‘w’):	Abre para escritura, truncando primero el fichero.
(‘x’):	Abierto para creación en exclusiva, falla si el fichero ya existe.
(‘a’):	Abierto para escritura, añadiendo al final del fichero si este existe.
(‘b’):	modo binario
(‘t’):	Modo texto.
(‘+’):	Abierto para actualizar (lectura y escritura).

Método *readable()*

Este método devolverá un tipo booleano, siendo True si el flujo de datos se puede leer; de lo contrario, devuelve False.

Método *read()*

Este método leerá todo el contenido de un archivo y lo devolverá como una cadena de texto, este es una buena forma de leer un archivo solo si tu archivo de texto no es muy grande. Este método acepta un parámetro adicional donde podemos especificar el número de caracteres a leer.

Método *close()*

Una vez terminaste de leer un archivo, es importante cerrarlo, de otra forma el archivo se queda abierto, lo cual puede generar problemas

palabra clave *with*

Su uso es considerado una buena práctica, porque el archivo se cierra automáticamente en lugar de tener que cerrarlo manualmente.

CREAR Y LEER ARCHIVO CSV

```
import csv

introducir datos
datos = [
  ['Nombre', 'Edad', 'Ciudad'],
  [...] ,
]

with  open('datos.csv','r') as archivo_csv:
                leer_csv = csv.reader(archivo_csv)
                for i in leer_csv:
                      print(i)
```

CREAR Y LEER ARCHIVO CSV

```
import csv
introducir datos
datos = [
  ['Nombre', 'Edad', 'Ciudad'],
  [...] ,
]
#sin usar with
archivo_csv=open('datos.csv','r')
leer_csv = csv.reader(archivo_csv)
        for i in leer_csv:
                print(i)
archivo_csv.close()
```

ESCRIBIR EN ARCHIVO CSV

```
import csv

introducir nuevos datos

with  open('datos.csv','w',newline='') as archivo_csv:
                escribir_csv = csv.writer(archivo_csv)
                escribir_csv.writerows(datos)


datos
```

Evitar que se agreguen líneas en blanco al final del archivo.

Escribir múltiples filas de datos

IMPORTAR ARCHIVO CSV

```
#Desde un archivo local con Pandas

import pandas as pd

df = pd.read_csv('/content/datos.csv')
df
```

dataframe: estructura de datos construida con filas y columnas

```
#Desde url con Pandas

import pandas as pd

df = pd.read_csv('http://.....')
df
```

CREAR Y ESCRIBIR ARCHIVO DE TEXTO

```
with open('archivo_texto.txt', 'w') as archivo_texto:
    archivo_texto.write("Hola, mundo!\n")
    archivo_texto.write("Este es un archivo de texto.\n")
    archivo_texto.write("Python facilita el manejo de archivos.")
```

\n indica un salto de línea

LEER ARCHIVO DE TEXTO

```
with open('archivo_texto.txt', 'r') as archivo_texto:
    contenido = archivo_texto.read()
    print("Contenido del archivo de texto:")
    print(contenido)
```

Contenido del archivo de texto:
Hola, mundo!
Este es un archivo de texto.
Python facilita el manejo de archivos.

AÑADIR CONTENIDO A UN ARCHIVO DE TEXTO

```
with open('archivo_texto.txt', 'a') as archivo_texto:
    archivo_texto.write("\nAñadiendo una nueva línea al archivo.")

with open('archivo_texto.txt', 'r') as archivo_texto:
    contenido = archivo_texto.read()
    print("Contenido del archivo de texto:")
    print(contenido)
```

Contenido del archivo de texto:
Hola, mundo!
Este es un archivo de texto.
Python facilita el manejo de archivos.
Añadiendo una nueva línea al archivo

DICCIONARIOS

Los diccionarios en Python son una estructura de datos que permite almacenar pares de **clave-valor**.
Cada **clave** es un identificador **único** e **inmutable.**
Los **valores** son **mutables**.

Útiles cuando se desea asociar un valor con una clave única, como en una base de datos simple o una tabla de búsqueda

diccionario= {clave1 : valor1, clave2 : valor2}

cliente= {'nombre': 'Pedro', 'documento': '023568'}

DICCIONARIOS

Creación de diccionarios

Para crear un diccionario se usan **llaves {}** y las claves y los valores se separan con **dos puntos :**

diccionario= **{**clave1 **:** valor1**,** clave2 **:** valor2**}**

```
# Crear un diccionario vacío
diccionario_vacio = {}
print(diccionario_vacio)
```

```
{}
```

```
# Crear un diccinario con elementos
diccionario_con_elementos = {'nombre': 'Juan','edad': 30,'altura': '1.80'}
print(diccionario_con_elementos)
```

```
{'nombre': 'Juan', 'edad': 30, 'altura': '1.80'}
```

DICCIONARIOS

Acceso a valores

Para acceder al valor asociado con una clave en un diccionario se utilizan corchetes **[]**

Si se intenta acceder así a una clave que no existe en el diccionario, se genera un error.

Para evitar esto , se puede utilizar el método get() que devuelve None si la clave no se encuentra en el diccionario

```
[ ]  # Acceder a valores en el diccionario
     diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
     nombre = diccionario['nombre']
     edad = diccionario['edad']
     altura = diccionario['altura']
     print(f'Nombre: {nombre}, Edad: {edad}, Altura: {altura}')
```

```
Nombre: Juan, Edad: 30, Altura: 1.8
```

```
[ ]  # Acceder al valor de una clave que no existe con get()
     diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
     peso = diccionario.get('peso')
     print(peso)
```

```
None
```

DICCIONARIOS

Añadir o modificar elementos

Tanto para agregar un nuevo elemento como para modificar un elemento existente, se accede a la clave y se le da (o modifica) el valor.

```
[ ]  # Modificar valores en el diccionario
     diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
     diccionario['edad'] = 31
     print(diccionario)
```

```
{'nombre': 'Juan', 'edad': 31, 'altura': 1.8}
```

```
[ ]  # Añadir un nuevo par clave-valor
     diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
     diccionario['peso'] = 80
     print(diccionario)
```

```
{'nombre': 'Juan', 'edad': 30, 'altura': 1.8, 'peso': 80}
```

DICCIONARIOS

Eliminar elementos: del , pop(), popitem()

Eliminar elemento con declaración ***del***

Si se intenta eliminar una clave que no existe con del, se genera un error.

```
# Eliminar un par clave-valor
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80, 'peso': 80}
del diccionario['peso']
print(diccionario)
```

```
{'nombre': 'Juan', 'edad': 30, 'altura': 1.8}
```

Método **pop()** que devuelve el valor asociado a la clave y elimina el elemento del diccionario. En este caso. Si la clave no existe y no tiene valor asociado en el método, da error. Si la clave no existe y tiene un valor asociado, lo elimina y devuelve.

```
# Eliminar par usando método pop() con elemento que no existe pero con valor asociado
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
ciudad = diccionario.pop('ciudad', None)
print(diccionario)
print(ciudad)
```

```
{'nombre': 'Juan', 'edad': 30, 'altura': 1.8}
None
```

El método **popitem()**, se elimina el último par del diccionario. También devuelve valor eliminado

```
[25] # Eliminar el último par del diccionario usando el método popitem()
     diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
     ultimo_valor = diccionario.popitem()
     print(diccionario)
     print(ultimo_valor)
```

```
{'nombre': 'Juan', 'edad': 30}
('altura', 1.8)
```

DICCIONARIOS

Eliminar diccionario

Para borrar un diccionario completamente utilizamos la palabra reservada **del**.

```
[13] # Borrar completamente un diccionario (no se puede imprimir porque el diccionario ya no existe)
     diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
     del diccionario
     print(diccionario)
```

Vaciar diccionario

Para eliminar todos los elementos de un diccionario sin borrar el diccionario lo que hacemos es vaciarlo, para ello usamos el método **clear()**. El diccionario está vacío pero sigue existiendo.

```
# Borrar todos los elementos del diccionario sin eliminar el diccionario
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
diccionario.clear()
print(diccionario)
```

```
{}
```

DICCIONARIOS

Tamaño del diccionario

Para obtener el nú´mero de elementos en un diccionario se utiliza la función l**en()**

```
# Tamaño del diccionario (número de pares)
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
num_elementos = len(diccionario)
print(num_elementos)
```

```
3
```

Comprobar si una clave está (o no) en el diccionario

Para comprobar si una clave está en el diccionario, se utiliza la palabra clave **in,** para comprobar si no está: **not in**

```
# Comprobar si una clave está en el diccionario
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
resultado1 = 'nombre' in diccionario
resultado2 = 'ciudad' in diccionario
print(f'Resutado1: {resultado1}, Resultado2: {resultado2}')
```

```
Resutado1: True, Resultado2: False
```

DICCIONARIOS

Listar claves, valores, elementos

Lista de claves, valores, elementos por orden de inserción:

```
[25] # Listar en orden de creación (devuelve listas)
     diccionario = {'nombre': 'Juan', 'edad':20, 'altura':1.70}
     lista_claves = list(diccionario.keys())
     lista_valores= list(diccionario.values())
     lista_elementos= list(diccionario.items())
     print(lista_claves)
     print(lista_valores)
     print(lista_elementos)
     type(lista_elementos)
```

```
['nombre', 'edad', 'altura']
['Juan', 20, 1.7]
[('nombre', 'Juan'), ('edad', 20), ('altura', 1.7)]
list
```

Lista ordenada de claves, valores, elementos:

```
# Listar de forma ordenada (devuelve listas)
# OJO se mezclan tipos de datos, da error
diccionario = {'nombre': 'Juan', 'edad':20, 'altura':1.70}
lista_claves = sorted(diccionario.keys())
lista_valores= sorted(diccionario.values())
lista_elementos= sorted(diccionario.items())
print(lista_claves)
print(lista_valores)
print(lista_elementos)
```

DICCIONARIOS

Iterar sobre un diccionario

Iterar sobre las claves: método **keys()**

```
# Iterar sobre un diccionario: Iterar sobre las claves
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1
for clave in diccionario.keys():
  print(clave)
```

```
nombre
edad
altura
```

Iterar sobre los valores: método **values()**

```
# Iterar sobre un diccionario: Iterar sobre los valore
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1
for valor in diccionario.values():
  print(valor)
```

```
Juan
30
1.8
```

Iterar sobre los pares: método **items()**

```
# Iterar sobre un diccionario: Iterar sobre pares
diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
for clave_valor in diccionario.items():
  print(clave_valor)
```

```
('nombre', 'Juan')
('edad', 30)
('altura', 1.8)
```

Iterar sin indicar método (se itera por clave)

```
[11] # Iterar sobre un diccionario sin método: (por defecto itera por clave)
     diccionario = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
     for por_defecto in diccionario:
       print(por_defecto)
```

```
nombre
edad
altura
```

DICCIONARIOS

Creación de conjutos a partir de diccionarios **set()**

```python
# Creación de conjuntos a partir de diccionarios
diccionario = {'nombre': 'Juan', 'edad':20, 'altura':1.70}
conjunto_claves = set(diccionario.keys())
conjunto_valores= set(diccionario.values())
conjunto_elementos= set(diccionario.items())
print(conjunto_claves)
print(conjunto_valores)
print(conjunto_elementos)
type(conjunto_elementos)
```

```
{'nombre', 'edad', 'altura'}
{1.7, 'Juan', 20}
{('edad', 20), ('nombre', 'Juan'), ('altura', 1.7)}
set
```

DICCIONARIOS

Copiar un diccionario

Para crear una copia de un diccionario, se utiliza el método **copy()**.

Una vez copiado, la modificación de un diccionario no afecta al otro.

También podemos utilizar el constructor **dict()**.

```
[6] # Copiar un diccionario
    diccionario_original = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
    diccionario_copia = diccionario_original.copy()
    print(diccionario_copia)
```

```
{'nombre': 'Juan', 'edad': 30, 'altura': 1.8}
```

```
# Copiar un diccionario con el constructor dict()
diccionario_original = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
diccionario_copia2 = dict(diccionario_original)
print(diccionario_copia2)
```

```
{'nombre': 'Juan', 'edad': 30, 'altura': 1.8}
```

DICCIONARIOS

Combinar diccionarios

Para combinar dos diccionarios se usa el mètodo update()
Este método actualiza el diccionario original con los pares clave-valor del diccionario que se pasa como argumento.
Si las claves ya existieran, se actualizan con el nuevo argumento.

```python
# Combinar diccionarios
diccionario1 = {'nombre': 'Juan', 'edad':30, 'altura':1.80}
diccionario2 = {'edad': 31, 'ciudad': 'Madrid'}
diccionario1.update(diccionario2)
print(diccionario1)
```

```
{'nombre': 'Juan', 'edad': 31, 'altura': 1.8, 'ciudad': 'Madrid'}
```

Diccionarios anidados

Un diccionario utiliza como valor otro diccionario.

```python
# Diccionarios anidados: se agrupan varidos diccionarios creados previamente
casa1 = {'provincia':'madrid', 'metros': 100}
casa2 = {'provincia':'segovia', 'metros': 150}
casa3 = {'provincia':'zamora', 'metros': 200}
propiedades = {'casa1':casa1, 'casa2': casa2, 'casa3':casa3}
print(propiedades)
```

```
{'casa1': {'provincia': 'madrid', 'metros': 100}, 'casa2': {'provincia': 'segovia', 'metros': 150}, 'casa3': {'provincia': 'zamora', 'metros': 200}}
```

CONJUNTOS: OPERACIONES BÁSICAS CON SETS

Los conjuntos (sets) en Python son una colección ***desordenada*** de elementos ***únicos***.

Aquí te explico las operaciones básicas que puedes realizar con ellos

CREACION DE UN CONJUNTO

```
# Crear un conjunto
conjunto = {1, 2, 3, 4, 5}
print(conjunto)
```

```
{1, 2, 3, 4, 5}
```

AÑADIR UN ELEMENTO

```
# Añadir elementos a un conjunto .add(x)
conjunto.add(6)
print(conjunto)
```

```
{1, 2, 3, 4, 5, 6}
```

ELIMINAR UN ELEMENTO

```
# Eliminar elementos de un conjunto con .remove(x)
conjunto.remove(2)
print(conjunto)
# Eliminar elementos de un conjunto con .discard(x)
conjunto.discard(3)
print(conjunto)
```

```
{1, 3, 4, 5}
{1, 4, 5}
```

ELIMINAR ELEMENTOS DE UN CONJUNTO

```
# Eliminar todos los elementos: set1.clear() vacía el conjunto.
conjunto.clear()
print(conjunto)
```

```
set()
```

UNIÓN, INTERSECIÓN Y DIFERENCIA DE CONJUNTOS

```python
# Unión de conjuntos
conjunto1 = {1, 2, 3}
conjunto2 = {3, 4, 5}
union = conjunto1 | conjunto2
print(f'Unión: {union}')

# Intersección de conjuntos
# Nos regresa un set con los elementos en común entre dos o mas conjuntos
interseccion = conjunto1 & conjunto2
print(f'Intersección: {interseccion}')

# Diferencia de conjuntos
# Nos regresa un set con los elementos que estan en el conjunto de la izquierda pero no en el de la derecha
diferencia = conjunto1 - conjunto2
print(f'Diferencia: {diferencia}')
```

```
Unión: {1, 2, 3, 4, 5}
Intersección: {3}
Diferencia: {1, 2}
```

EJEMPLO DE PERMANENCIA EN CONJUNTOS

```
#Ejemplo de Pertenencia en Conjuntos
# Definimos un conjunto
conjunto1 = set([1,2,3,4,5,6])
# Verificamos si ciertos elementos están en el conjunto
print(5 in conjunto1)
print(6 in conjunto1)
print(7 in conjunto1)
```

```
True
True
False
```

LONGITUD DE UN CONJUNTO

```
#Longitud de un conjunto: len(set1)
#Devuelve el número de elementos en el conjunto.

len(conjunto1)
```

```
6
```

DIFERENCIA SIMÉTRICA

```python
# Es una operación de conjuntos que devuelve un nuevo conjunto
# con los elementos que están en uno de los conjuntos o en el otro, pero no en ambos.
# Es decir, la diferencia simétrica entre dos conjuntos A y B contiene los elementos que están en A o en B,
# pero no en la intersección de A y B.
set1 = {1,2,3}
set2 = {3,4,5}

diferencia_simetrica_set = set1 ^ set2
print(diferencia_simetrica_set)

diferenci_symetric = set1.symmetric_difference(set2)
print(diferenci_symetric)
```

```
{1, 2, 4, 5}
{1, 2, 4, 5}
```

MANEJO DE EXCEPCIONES

Try, except, else y finally

El manejo de excepciones te permite manejar errores de manera controlada sin interrumppir el flujo del programa.

- Try: El bloque try se puede usar para envolver el código que puede generar una excepcion
- Except: Se usa para excepciones específica. Puede haber varias bloques para diferentes tipo de excepciones.
- Else: Es para indicar que se puede ejecutar si no se produjo ninguna excepción.
- Finally: Tanto else y Finally es opcional, Se utiliza para ejecutar codigo de limpieza o para indicar el final de bloque try.

MANEJO DE EXCEPCIONES

Try, except, else y finally. Ejemplos

```
try:
 file = open('data.txt', 'r')
 data = file.read()
except FileNotFoundError:
 print("El archivo no existe.")
except IOError:
 print("Error al leer el archivo.")
else:
 print("Lectura exitosa:", data)
finally:
 if 'file' in locals():
  file.close()
```

```
try:
    file = open('data.txt', 'r')
    data = file.read()
except FileNotFoundError:
    print("El archivo no existe.")
except IOError:
    print("Error al leer el archivo.")
else:
    print("Lectura exitosa:", data)
finally:
    if 'file' in locals():
        file.close()
```


¡Agradezco la gentil atención
prestada!