

Ficheros

January 20, 2026



**INSTITUTO SUPERIOR
TECNOLÓGICO QUITO**
Formamos tu **PROPÓSITO DE VIDA**

1 Ficheros



1.1 Nombre: *Adriel Bedoya*

1.2 Acceso a Ficheros

1.2.1 Ficheros

Primero vamos clarando que *File* es un tipo de objeto predefinido de Python, se pueden acceder a ficheros directamente desde Python y son un tipo especial:

- Son predefinidos, pero no son ni números, ni secuencias ni mappings. Tampoco responden a operadores con extensiones.

Tenemos una función para poder abrir los ficheros en modo lectura que es *open()*, permite crear objetos de tipo fichero

Formato para abrir el fichero: `nombre_carpeta = open(archivo, modo)`

- El modo es opcional. Por defecto, los ficheros se abren en modo lectura
- Los datos leídos de un fichero siempre se obtienen en formato string. Lo mismo ocurre con escritura.
- Los ficheros se deben cerrar invocando `close` (liberación de recursos).

```
[1]: # Lectura desde fichero usando método 'read'. Devuelve todo el contenido del fichero.
fichero_read = open('res/multiple_lines.txt')
print(fichero_read.read())
fichero_read.close() #Siempre debemos cerrar en caso que hagamos de esta forma,
                    ↪ más adelante hay otra forma de resolver
```

```
10
20
30
```

```
[2]: # Lectura línea a línea a través del bucle 'for'.
fichero_for = open('res/multiple_lines.txt')
for linea in fichero_for:
    print(f"{linea}")
fichero_for.close()
```

```
10
20
30
```

```
[3]: # leer de un archivo a una lista.
fichero_lista = open('res/multiple_lines.txt')
#Simplemente lo transforma a una lista, hace lo mismo que la primera solo que
    ↪ en vez de espacios, es \n
leer_entre_lineas = fichero_lista.readlines()
fichero_lista.close()
print(leer_entre_lineas)
```

```
['10\n', '20\n', '30\n']
```

1.2.2 Resolución de Paths independiente a la plataforma

Esto nos ayudará bastante a que sea portable, es decir, que nosotros vamos a poder correr los ficheros desde cualquier sistema operativo, para eso vamos a usar la librería *os*, en caso de querer ver más documentación, se la puede ver [aquí](#)

```
[4]: import os
      #La sintaxis es os.path.join(carpeta, archivo.ext)
      ruta = os.path.join("res", "multiple_lines.txt")
      print(ruta)
```

res\multiple_lines.txt

```
[5]: #En caso que tengamos almacenado en una variable, solo ponemos el * para
      ↪referenciarla
      dataset_file_path = ["res", "multiple_lines.txt"]
      ruta = os.path.join(*dataset_file_path)
      print(ruta)
```

res\multiple_lines.txt

```
[6]: #Esta es la otra forma de ver y la que más se usa para utilizar los ficheros.
      #Generalmente los programadores nos olvidamos cerrar las cosas y con este
      ↪ciclo, podremos hacerlo directamente sin necesidad de mencionarlo
      #Este ciclo tiene el siguiente ciclo de vida
      #Recibe con una ruta y nombramos una variable
      #Luego usamos un for para que lea todo el archivo y al final ponemos un print
      ↪del archivo
      with open(ruta) as mi_fichero:
          for linea in mi_fichero:
              print(linea, end= '')
```

10

20

30

```
[7]: #Se puede abrir múltiples ficheros con el with
      ruta1 = os.path.join("res", "one_line.txt")
      ruta2 = os.path.join("res", "multiple_lines.txt")
      #Recuerda, es un ciclo así que al momento de que termine las condiciones, se
      ↪cerrará
      with open(ruta1) as fichero1, open(ruta2) as fichero2:
          print(fichero1.readlines())
          print(fichero2.readlines())
```

['Hola mundo desde un fichero.']

['10\n', '20\n', '30\n']

1.2.3 Modos de acceso

Al crear un objeto de tipo **File** se puede especificar que modo de acceso queremos dar (**Read or Write**)

Modo de acceso	Descripción
r	Solo lectura
w	Solo escritura (borra el archivo si ya existe)
x	Solo escritura (falla si el archivo ya existe)
a	Crea el fichero; si existe, se abre y se añade al final
r+	Lectura y escritura
b	Se puede añadir a otros modos para acceso binario
t	Acceso para archivos de texto (valor por defecto)

1.2.4 Acceso para Escritura

```
[8]: #Creamos una función que recibira un parámetro
def crear_lista(tamanyo):
    #Creamos una lista vacía para almacenar valores
    lista = []
    #Creamos un for para recorrer la información del parámetro
    for i in range(tamanyo):
        #Convierte el índice en texto, da salto de línea y añade ese texto a la
        ↪ lista
        lista.append(str(i) + '\n')
    #retornamos la lista
    return lista

#Creamos una variable ruta la cual va hacer referencia a mi archivo
ruta = os.path.join("res", "a_dummy.txt")
#Con with nosotros abrimos el archivo y el "wt" es write and text
with open(ruta, 'wt') as fichero:
    #Con esto, en caso que exista algo dentro, lo borra y lo reemplaza
    fichero.write('Cabecera del ciclo del for en el ejemplo\n')
    #Llama a la función crear_lista y genera una lista con 10 líneas (0 a 9).
    lista = crear_lista(10)
    #Escribe todos los strings de la lista en el archivo
    fichero.writelines(lista)
    #Print notificación que se completo el ciclo
    print("archivo creado")
```

archivo creado

1.3 Buffering

- Por defecto, el texto que transferimos desde Python a un ficho, no se guarda directamente en el disco. Se almacena en un buffer.

- Acciones como cerrar un fichero o invocar el método flush fuerzan que se transfiera el contenido del buffer a disco.

```
[9]: #Hacemos referencia al archivo en nuestra PC
ruta = os.path.join("res", "FicheroParaEscritura.txt")
#Abre el archivo en modo escritura, si el archivo existe entonces se borra su
    ↳ contenido, sino se crea y el archivo queda abierto para escribir.
fichero_write = open(ruta, 'w')
#Escribe el texto "foter" en el archivo. El texto queda en el buffer, no
    ↳ necesariamente en el disco aún.
fichero_write.write('foter')

#Abre el archivo en modo lectura
fichero_read = open(ruta, "r")
#Intenta leer todas las líneas del archivo
#sale [] porque el contenido aún está en el buffer de escritura.
print(fichero_read.readlines())

#Fuerza a que el buffer de escritura se guarde en el archivo.
#Ahora el texto "foter" sí existe físicamente en el archivo.
fichero_write.flush()

#Ya se puede ver
print(fichero_read.readlines())

#Cerramos la escritura
fichero_write.close()
#Cerramos la lectura
fichero_read.close()

[]
['foter']
```

1.4 Archivos CSV

Python permite leer datos de ficheros **CSV** y también escribir ficheros en este formato, es popular para la *Ciencia de Datos*

```
[10]: # tabla_operaciones.csv contiene valores separados por comas
import csv #módulo estándar del csv
#Hacemos referencia a nuestro archivo csv
ruta = os.path.join("res", "tabla_operaciones.csv")
#Creamos el ciclo with donde abriremos el archivo en la variable fichero
with open(ruta) as fichero:
    #Creamos una variable el cual me permita leer mi archivo csv
    #delimiter=', ' indica que los campos están separados por comas.
    #Cada fila leída será una lista de cadenas strings.
    data_reader = csv.reader(fichero, delimiter=',')
```

```

#Creamos un bucle for para que recorra fila por fila
for linea in data_reader:
    #Accede al primer elemento de la fila linea[0]
    #Accede al segundo elemento de la fila linea[1]
    #Concatena ambos usando -----
    #print
    print(linea[0] + ' ---- ' + linea[1])

```

Operacion ---- Descripcion

```

a + b ---- suma a y b
a - b ---- resta a menos b
a / b ---- a dividido
a // b ---- a dividido entre b (quitando decimales)
a % b ---- devuelve el resto de la divisiÃ³n a/b (modulus)
a * b ---- a multiplicado por b
a ** b ---- a elevado a b

```

```

[11]: #Hacemos referencia a nuestro archivo csv en la variable ruta
ruta = os.path.join("res", "tabla_operaciones.csv")
#Hacemos que se abra en modo lectura cuando se referencie en la variable
f = open(ruta)
#Creamos un archivo modo lectura para el csv, indicamos que los campos estÃ¡n
↳separados por comas ","
data_reader = csv.reader(f, delimiter = ',')

#Hacemos referencia al nuevo archivo
ruta_o = os.path.join("res", "4_tabla_operaciones.csv")

#Abrimos el ciclo with con el segundo archivo y lo ponemos en modo escritura
with open(ruta_o, 'w') as f2:
    #Decimos que el archivo se abra en modo escritura y su separaciÃ³n serÃ¡ el
    ↳sÃ­mbolo "/"
    csv_writer = csv.writer(f2, delimiter = '|')
    #Hacemos un for para que lea lÃ­nea por lÃ­nea
    for line in data_reader:
        #Imprimimos la lÃ­nea
        print(line)
        #Convierte la lista en una lÃ­nea de texto y Usa "/" como separador
        csv_writer.writerow(line)
f.close()

```

```

['Operacion', 'Descripcion']
['a + b', 'suma a y b']
['a - b', 'resta a menos b']
['a / b', 'a dividido']
['a // b', 'a dividido entre b (quitando decimales)']
['a % b', 'devuelve el resto de la divisiÃ³n a/b (modulus)']
['a * b', 'a multiplicado por b']

```

```
['a ** b', 'a elevado a b']
```

1.5 Docstrings

- Python permite adjuntar documentación a los objetos e inspeccionarla a través de la línea de comandos o durante la ejecución del programa.
- Los docstrings se almacenan en el atributo “__ doc __” de cada objeto.
- El valor de dicho atributo se puede mostrar por medio de la función help

help(open)

- No es necesario editar este atributo directamente.
- Para asociar un docstring a un objeto basta con escribir el texto (entre triples comillas) al principio de los módulos, funciones o clases, antes del código ejecutable

```
[12]: def funcion_de_prueba():  
      """  
      Aquí puedo poner la documentación de la función  
      por el momento no existe, pero así se hace  
      """  
      pass  
      help(funcion_de_prueba)
```

Help on function funcion_de_prueba in module __main__:

```
funcion_de_prueba()  
  Aquí puedo poner la documentación de la función  
  por el momento no existe, pero así se hace
```

- Accediendo al atributo __ doc __ sólo se obtiene el docstring

```
[13]: #Solo muestra la documentación  
      print(funcion_de_prueba.__doc__)
```

```
  Aquí puedo poner la documentación de la función  
  por el momento no existe, pero así se hace
```

Algunos Comentarios Adicionales

- Comentarios con # suelen asociarse a expresiones sencillas, instrucciones individuales o pequeños bloques de código.
- Los docstrings son más apropiados para construcciones de más alto nivel: funciones, clases y módulos.

1.5.1 Sphinx

- Herramienta de documentación.
- Especialmente útil para sistemas grandes.

- Da soporte a una gran variedad de formatos de salida: HTML, LaTeX, ePub,

Aquí es la Web de [Sphinx](#)

También están los proyectos que la usan [aquí](#)

1.5.2 Estilo

- Es importante que los docstrings sean consistentes.
- El equipo de desarrollo debe acordar un formato y seguirlo rigurosamente.

Google Python Style Guide: Google define la siguiente [guía de estilo](#)

Esta contiene reglas para los docstrings de: 1. **Módulos.** 2. **Métodos y Funciones.** 3. **Clases.**

1.6 Ejercicios

1. Escribe una función que reciba una ruta de un fichero de texto y una cadena de caracteres a buscar y determine si la cadena aparece en el fichero.
2. Escribe una función que reciba una lista, una ruta destino y un número n. La función debe crear un fichero en la ruta especificada. El contenido del fichero serán los primeros n elementos de la lista. La función debe controlar de manera apropiada los posibles valores de n que estén fuera de rango.
3. Escribe una función que reciba una ruta de un fichero de texto devuelva un diccionario con la frecuencia de aparición de cada palabra. Ejemplo: un fichero que contenga la frase ‘es mejor que venga que que no venga’ devolverá el siguiente diccionario: {‘es’ : 1, ‘mejor’ : 1, ‘que’ : 3, ‘venga’ : 2, ‘no’ : 1}. Para dividir un string en palabras puedes hacer uso del método split.

```
[14]: #Escribe una función que reciba una ruta de un fichero de texto y una cadena de
      ↪ caracteres a buscar
      #y determine si la cadena aparece en el fichero.

      #LÓGICA
      #1) Importamos el módulo os
      #2) Creamos una función que reciba dos parámetros : ruta del fichero y la
      ↪ palabra que queremos buscar
      #3) Creamos un with para que el archivo se abra y esté en modo lectura, esto se
      ↪ guarda en una variable
      #4) Creamos una variable donde haremos que se lea el fichero
      #5) hacemos un if que si la palabra a buscar está en la variable donde leímos
      ↪ el fichero, dará un mensaje, sino dará un mensaje negativo
      #6) Fuera de la función, nosotros creamos la referencia a un archivo
      #7) Hacemos referencia a la función y ponemos la palabra a buscar, luego
      ↪ hacemos print

      #Paso 1:
      import os
      #Paso 2:
      def buscar_letras (ruta_fichero, palabra_buscar):
```

```

#Paso 3:
with open(ruta_fichero, 'r') as fichero:
    #Paso 4:
    contenido = fichero.read()
    #Paso 5:
    if palabra_buscar in contenido:
        print("La palabra está dentro del fichero")
    else:
        print("La palabra no está dentro del fichero")

#Paso 6:
archivo_importante = os.path.join("res", "multiple_lines.txt")
#Paso 7:
buscar_letras(archivo_importante, "contiene tres") #Está en la segunda fila
buscar_letras(archivo_importante, "Me llamo Adriel") #No está

```

La palabra no está dentro del fichero
La palabra no está dentro del fichero

```

[15]: """
Escribe una función que reciba una lista, una ruta destino y un número n.
La función debe crear un fichero en la ruta especificada.
El contenido del fichero serán los primeros n elementos de la lista.
La función debe controlar de manera apropiada los posibles valores de n que
    ↪estén fuera de rango.
"""

#1) Importamos el módulo de os
import os
#2) Creamos la función con los 3 parámetros mencionados
def guardar_primeros_n(lista, ruta_destino, n):
    #3) Hacemos un control de valores fuera del rango
    #Si n es negativo, entonces se muestra un mensaje de error y se sale de la
    ↪función con return para evitar fallos.
    if n < 0:
        print("Error: n no puede ser negativo")
        return
    #Si n es mayor que el tamaño de la lista, entonces:
    if n > len(lista):
        #Se ajusta n al tamaño real de la lista.
        n = len(lista) #Así se escriben todos los elementos disponibles sin
    ↪error.

    #4) Creación del fichero
    #Abrimos el fichero en modo escritura y lo almacenamos en una variable
    with open(ruta_destino, 'w') as fichero: #encoding='utf-8' = si pongo esto,
    ↪el programa no tendrá problema en leer los caracteres especiales

```

```

        #Usamos un slice para tomar solo los primeros n elementos de la lista.
        ↪Es seguro incluso si n es igual al tamaño de la lista.
        for elemento in lista[:n]:
            #Convierte cada elemento a texto con parsear el str, lo escribe en
            ↪el archivo y separa cada dato en una nueva línea .
            fichero.write(str(elemento) + '\n')

datos = [10, 20, 30, 40, 50]
ruta_archivo = os.path.join("res", "multiple_lines.txt")

guardar_primeros_n(datos, ruta_archivo, 3)

#5) Verifiquemos
ruta_datos = os.path.join("res", "multiple_lines.txt")
with open(ruta_datos) as fichero_ver:
    print(fichero_ver.read())

```

10
20
30

```

[16]: #Escribe una función que reciba una ruta de un fichero de texto devuelva un
        ↪diccionario con la frecuencia de aparición de cada palabra.
#Ejemplo: un fichero que contenga la frase 'es mejor que venga que que no
        ↪venga' devolverá el siguiente diccionario: {'es' : 1, 'mejor' : 1, 'que' :
        ↪3, 'venga' : 2, 'no' : 1}.
#Para dividir un string en palabras puedes hacer uso del método split.

#1) Creamos una función que me traiga la ruta del fichero
def frecuencia_palabras(ruta_fichero):
    #2) Creamos un diccionario vacío
    frecuencias = {}

    #3) Creamos un ciclo with para poder leer el fichero y guardarlo en una
    ↪variable
    with open(ruta_fichero, 'r') as fichero:
        #4) Leemos el fichero
        contenido = fichero.read()
        #5) Separamos el contenido en palabras
        palabras = contenido.split()

        #6) Recorremos para que el programa lea todas las palabras y se
        ↪almacene en una variable
        for palabra in palabras:
            #7) Si la palabra está en frecuencias, entonces
            if palabra in frecuencias:

```

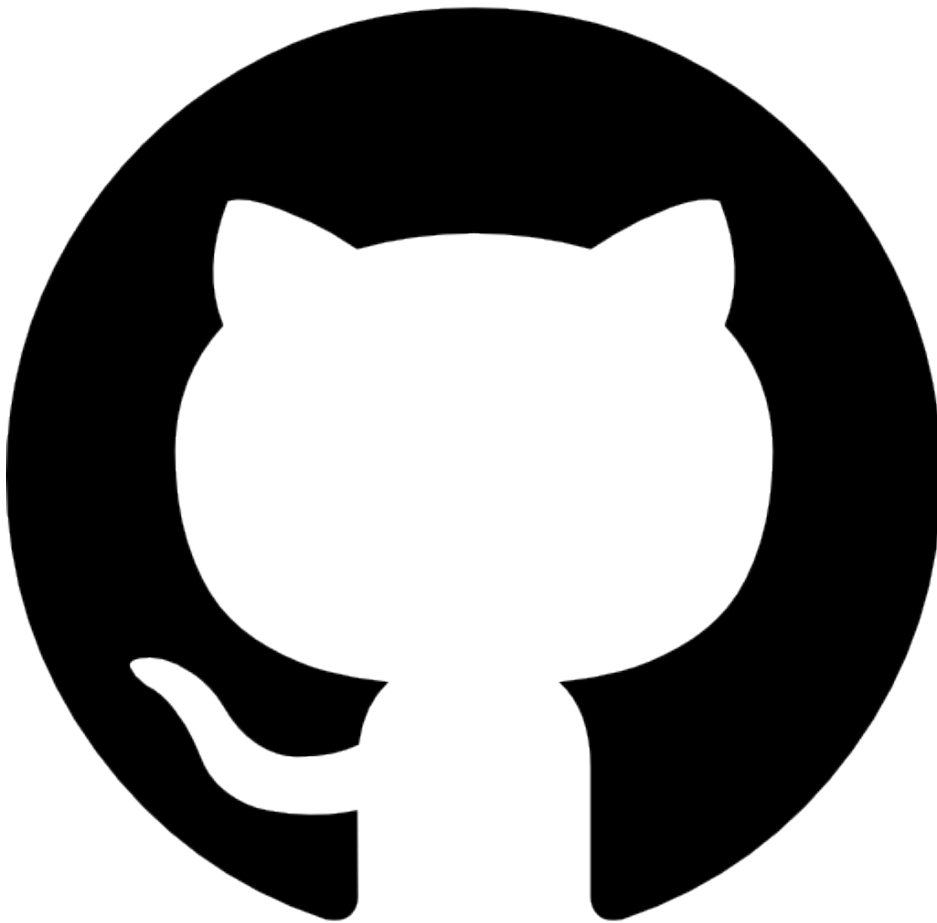
```

        #Se irá incrementando el número hasta que pare
        frecuencias[palabra] += 1
    else:
        #Sino, se mantendrá en uno porque solo se menciona una vez
        frecuencias[palabra] = 1
    #8) Regresamos el diccionario
    return frecuencias
#9) Validación
ruta_secreta_supersecreta = os.path.join("pruebas", "textito.txt") #Creamos un
    ↪ejemplo en nuestra carpeta "pruebas"
resultado = frecuencia_palabras(ruta_secreta_supersecreta) #instanciamos en una
    ↪variable la referencia a la función
print(resultado)#Print

```

```
{'es': 1, 'mejor': 1, 'que': 3, 'venga': 2, 'no': 1}
```

2 Github



2.0.1 Click aquí para [ver el repositorio](#)