

Programación con python

Manejo de archivos en Python:
write, append y read

Manejo de archivos en Python: write, append y read

Un archivo de texto permite guardar información de forma permanente en el disco.

Sirve para guardar resultados, logs, configuraciones, etc.

En Python usamos la función `open()` para crear, leer o modificar archivos.

Manejo de archivos en Python: write, append y read

```
archivo = open(r'ruta\al\archivo.txt', 'modo')
```

- Primer parámetro: ruta del archivo (puede ser absoluta o relativa).
- Segundo parámetro (modo):

'w' > write (escritura, crea o sobrescribe).

'a' > append (agregar al final sin borrar lo que ya había).

'r' > read (lectura).

- open() devuelve un objeto archivo con el que trabajamos.

Escribir en un archivo (write)

```
write_file = open(r'FakeFile.txt', 'w')  
write_file.write('This is our first sentence in our file.')  
write_file.close()
```

- Modo 'w' crea el archivo si no existe y borra el contenido anterior si ya existía.
- El método write() escribe texto en el archivo.
- write() regresa el número de caracteres escritos.
- Es importante cerrar el archivo con close() para que se guarden los cambios.

Agregar texto (append)

```
append_file = open(r'FakeFile.txt', 'a')  
append_file.write(' This is our Second sentence in our file.')  
append_file.close()
```

- Modo 'a' no borra lo que ya está, solo agrega al final.
- Ideal para ir registrando información nueva (por ejemplo, logs).

Usar with open(...) (manejo automático)

```
with open(r'FakeFile.txt', 'a') as append_file:  
    append_file.write('\nThis is our Third sentence in our file on a new  
line.')
```

- La palabra clave with crea un contexto.
- Al salir del bloque with, el archivo se cierra automáticamente, sin llamar close().
- Es la forma recomendada de trabajar con archivos.
- En el ejemplo también usamos \n para brincar a una nueva línea.

El carácter de nueva línea \n

\n es el “new line character”.

No se ve como texto, pero le indica a la computadora que salte a la siguiente línea.

```
print("hello\nworld\n!")
```

Salida:

```
hello
world
!
```

Escribir varias líneas

```
multi_line = """  
This is the Fourth sentence.  
This is the Fifth sentence.  
This is the Sixth sentence.  
"""
```

```
with open(r'FakeFile.txt', 'a') as append_file:  
    append_file.write(multi_line)
```

- Se usa una cadena multilínea con `""" """`.
- Cada renglón se guarda en el archivo tal como se ve.
- El modo `'a'` sigue agregando al final del archivo.

Leer un archivo completo (read)

```
with open(r'FakeFile.txt', 'r') as read_file:  
    contenido = read_file.read()  
    print(contenido)
```

- Modo 'r' abre el archivo en solo lectura.
- read() lee todo el contenido del archivo como una sola cadena.
- print() nos permite ver en pantalla lo que hay en el archivo.

Buenas prácticas: manejo de errores con `try / except`

Cuando trabajamos con archivos pueden ocurrir errores como:

- El archivo no existe
- La ruta es incorrecta
- No hay permisos de lectura/escritura
- El archivo está siendo usado por otro programa
- Por eso es recomendable usar `try/except`

Buenas prácticas: manejo de errores con try / except

try:

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

except FileNotFoundError:

```
    print("Error: El archivo no existe. Verifica la ruta.")
```

except PermissionError:

```
    print("Error: No tienes permisos para abrir este archivo.")
```

except Exception as e:

```
    print(f"Ocurrió un error inesperado: {e}")
```

Explicación

- `try`: intenta ejecutar el código normalmente.
- `FileNotFoundError`: captura el error si el archivo no existe.
- `PermissionError`: captura problemas de permisos.
- `Exception`: captura cualquier otro error no previsto.
- El archivo se cierra automáticamente gracias al `with open(...)`.

Explicación

- `try`: intenta ejecutar el código normalmente.
- `FileNotFoundError`: captura el error si el archivo no existe.
- `PermissionError`: captura problemas de permisos.
- `Exception`: captura cualquier otro error no previsto.
- El archivo se cierra automáticamente gracias al `with open(...)`.

Porque es importante

- Evita que el programa se “rompa”.
- Muestra mensajes amigables al usuario.
- Facilita depuración y manejo de errores.
- Hace el código más seguro y profesional.

Manejo de errores al escribir archivos (write y append)

- Cuando escribimos o agregamos contenido a un archivo, también pueden aparecer errores como:
- No existe la carpeta donde queremos guardar
- No tenemos permisos en esa ubicación
- El archivo está bloqueado por otro programa
- La ruta está mal escrita

Escribir en un archivo con manejo de errores

- Cuando escribimos o agregamos contenido a un archivo, también pueden aparecer errores como:
- No existe la carpeta donde queremos guardar
- No tenemos permisos en esa ubicación
- El archivo está bloqueado por otro programa
- La ruta está mal escrita

Escribir en un archivo con manejo de errores

```
try:
    with open('notas.txt', 'w') as file:
        file.write("Estas son mis notas guardadas correctamente.\n")
        print("Archivo escrito exitosamente.")

except FileNotFoundError:
    print("Error: La ruta indicada no existe.")

except PermissionError:
    print("Error: No tienes permisos para escribir en esta ubicación.")

except Exception as e:
    print(f"Ocurrió un error inesperado: {e}")
```

Agregar contenido (append) con manejo de errores

try:

```
with open('notas.txt', 'a') as file:  
    file.write("Nueva línea agregada.\n")  
    print("Línea agregada con éxito.")
```

except Exception as e:

```
    print(f"No se pudo agregar información al archivo: {e}")
```

Actividad

El usuario proporciona una ruta a un archivo .txt.

Tu programa debe:

1. Abrir el archivo (manejar errores).
2. Contar:
 - * Número de líneas
 - * Número de palabras
 - * Número de caracteres
3. Guardar el reporte en un archivo llamado "reporte.txt"
4. Mostrar el reporte en pantalla.

Actividad - Ejemplo

Líneas: 8

Palabras: 42

Caracteres: 210