1. Manipulación de archivos

Hasta el momento, todos los programas que se han expuesto y desarrollado almacenan sus datos en RAM. Es decir, al finalizar la ejecución del software todo es *desalocado* de la memoria. Es deseable, en la mayoría de los sistemas, poder contar con un mecanismo de **persistencia** de datos. Esto puede ser logrado de muchas formas y en muchos formatos distintos, popularmente –aunque no necesariamente– en una *Base de Datos*.

En esta oportunidad se expondrá cómo almacenar y leer datos en **archivos de texto plano**. Es decir, que podremos guardar información en documentos que podrían ser modificados externamente a través de un simple editor. Y a su vez, poder recuperar estos datos desde el mismo programa.

1.1. Apertura y cierre

En `Python` para poder manipular (leer y/o escribir) en un archivo de texto debemos primeramente abrirlo con la función `open()`. Ésta posee dos parámetros, la primera es el nombre del archivo a leer o escribir y el segundo es el modo en que queremos abrirlo: escritura, lectura o lectura-escritura con algunas variantes:

```
open("nombre_archivo.txt", <MODO>)
```

Los **modos** de apertura son los siguientes:

`'r'`: Modo de solo lectura. El *cursor* se posiciona al principio del archivo. Si no existe esto **causa un error**. Si no se especifica el modo, este es el *default*.

`'r+'`: Modo lectura-escritura. El *cursor* se posiciona al principio del archivo. Si no existe **causar un error**.

`'w'`: Modo de solo escritura. Si no existe el archivo, este es creado, en caso de que sí exista, se trunca (es sobreescrito). El *cursor* se posiciona al principio del archivo.

`'w+'`: Modo de escritura-lectura. Si no existe el archivo, este es creado, en caso de que sí exista, se trunca (es sobreescrito). El *cursor* se posiciona al principio del archivo.

`'a'`: Modo anexado de solo escritura. Si no existe el archivo, éste es creado. En caso de que sí exista, se mantiene la información presente en el archivo. El *cursor* se posiciona al final del archivo para agregar datos (manteniendo los anteriores).

`'a+'`: Modo anexado de escritura-lectura. Si no existe el archivo, éste es creado. En caso de que sí exista, se mantiene la información presente en el archivo. El *cursor* se posiciona al final del archivo para agregar datos (manteniendo los anteriores).

Es importante que luego de abrir un archivo de texto lo cerremos utilizando la función interna `close()`. En caso de que no exista el archivo, se puede optar por que `Python` arroje el error por *default* terminando el programa o "atajarlo" y mostrar un mensaje personalizado y luego terminar el programa por nuestra cuenta con la función `exit()`.

Programa 1: Abrir y cerrar un archivo de texto.

```
def main():
  try:
    archivo = open("Hola.txt")
    print("El archivo fue abierto con éxito")
    archivo.close()
  except FileNotFoundError:
    print("Error, no existe el arcvhio 'Hola.txt'")
    exit(1) # valor distinto de 0 indica un error...

main()
```

1.2. Escritura

Para únicamente escribir en un archivo se utilizará el modo `'w'` y luego podemos utilizar la función `write()`:

Programa 2: Escribir en un archivo de texto.

```
def main():
  ingreso_usuario = input("Escriba un texto:")
  archivo = open("Hola.txt","w") # crea el archivo si no existe
  archivo.write(ingreso_usuario)
  archivo.close()

main()
```

1.3. Lectura

En el caso de que utilicemos un archivo de solo lectura, se utilizará el modo `'r'` y luego la función `read()` que obtendrá todo el contenido dentro del archivo en un *string*.

Programa 3: Lectura de un archivo completo de texto.

```
def main():
  archEntrada = open("Hola.txt","r")
  print(archEntrada.read())
  archEntrada.close()

main()
```

Además podemos utilizar la función `readline()` para obtener de a uno las líneas que se encuentran dentro del archivo, las cueles están delimitadas por el carácter *nueva línea* (`'\n'`). La línea es leída con el carácter `'\n'` incluido, para deshacernos de él utilizaremos la función `strip()`.

Contenido en `'Hola.txt'`

```
10↲
¡Hola, mundo!↲
```

Programa 4: Lectura de un archivo por línea.

```
def main():
  # Se abre el archivo: Error si no existe
  archEntrada = open("Hola.txt","r")

  # Se lee la 1ra línea del archivo (el 10)
  n = int(archEntrada.readline().strip())

  # Se lee la 2da línea del archivo
  texto = archEntrada.readline().strip()

  for i in range(n):
    print(texto)

  # No olvidar cerrar el archivo
  archEntrada.close()

main()
```

1.3.1. Fin del archivo (*End-of-File*)

Los archivos de texto contienen un carácter especial que indica cuando se ha alcanzado el final del mismo. En `Python` no hace falta nada especial para detectarlo. En caso de desconocer la longitud del archivo podemos comprobar con un simple condicional lo que devuelve la lectura del archivo. Por otro lado, podemos leer una cantidad específica de caracteres si pasamos un argumento entero a la función `read()`.

Programa 5: Copiar texto carácter a carácter de un archivo a otro en mayúsculas.

```
def main():
  # Se abre el archivo: Error si no existe
  archEntrada = open("entrada.txt","r")
  # Se crea el archivo de salida:
  archSalida  = open("salida.txt","w")

  print("Copiando archivo en mayúsculas...")

  c = archEntrada.read(1) # Se lee el primer carácter

  # Mientras haya un carácter para leer:
  while c:
    archSalida.write(c.upper()) # Se escribe la mayúscula
    c = archEntrada.read(1)     # se lee el siguinte carácter

  print("Copia completa.")

  archEntrada.close()
  archSalida.close()

main()
```

1.4. Borrado, copiado y renombrado de archivos

En el módulo estándar `os` (*Sistema Operativo*) podemos acceder a diversas funciones que manejan aspectos del sistema. Especialmente se hará foco en la manipulación de archivos: borrado y renombrado. Esto se hará a través de las funciones `rename()` y `remove()`.

Para utilizarlas importaremos el módulo `os`. Pero para copiar archivos se utilizará la función `copy()` del módulo `shutil` (*shell utilities*).

```
import os
import shutil
# ...

# Borra el archivo 'copia.txt'
os.remove("copia.txt")

# Si existe es sobrescrito:
shutil.copy("original.txt","copia.txt")

# Cambia el nombre de 'nombre_original.txt' a 'nuevo_nombre.txt'
os.rename("origina.txt", "copia.txt")
```

Al disponer de estas herramientas es posible hacer software más sofisticado y robusto, que esté preparado para diversos escenarios:

Programa 6: Copia a mayúsculas con validaciones y copias.

```
1  import shutil
2
3  def obtener_si_o_no(msj:str) -> bool:
4    si_no = input(msj)
5    while si_no[0].upper() != 'S' and si_no[0].upper() != 'N':
6      print("Error: Complete Sí o No")
7      si_no = input(msj)
8    return si_no[0].upper() == 'S'
9
10 def comprobar_y_copiar(nom: str):
11   try:
12     arch = open(nom,"r")
13     arch.close()
14     # El archivo existe...
15     pisar=obtener_si_o_no(f"{nom} ya existe,¿crear copia?(S/N)")
16     if pisar:
17       nom_copia = input("Ingrese nombre para la copia: ")
18       shutil.copy(nom, nom_copia)
19       print(f"\nSe creo la copia de {nom} en {nom_copia}\n")
20   except FileNotFoundError:
21     return
22
23 def main():
24   nom_entrada = input("Archivo de entrada: ")
25
```

```
26    # Se abre el archivo de entrada:
27    archEntrada = open(nom_entrada, "r")
28
29    nom_salida = input("Archivo de salida: ");
30    # Si un archivo con el mismo nombre existe, se ofrece copiarlo
31    comprobar_y_copiar(nom_salida);
32
33    # Se crea/pisa el archivo de salida con el nombre elegido
34    archSalida = open(nom_salida, "w");
35
36    print(f"Copiando del archivo '{nom_entrada}'...")
37
38    c = archEntrada.read(1) # Primer carácter del archivo.
39
40    # Mientras haya caracteres para leer:
41    while c:
42      archSalida.write(c.upper()) # Se escribe la mayúscula.
43      c = archEntrada.read(1)     # Próximo carácter.
44
45    print(f"Copia completa en '{nom_salida}'.")
46
47    archEntrada.close()
48    archSalida.close()
49
50  main()
```

Una posible salida de este programa podría ser como se muestra a continuación, teniendo en cuenta el siguiente archivo de entrada:

Contenido en 'entrada.txt'

```
Esto es un archivo de prueba↲
con multiples lineas↲
el cual sera copiado↲
en un nuevo archivo↲
con todas las letras en↲
mayuscula.↲
```

```
Archivo de entrada: entrada.txt
Archivo de salida: salida.txt
salida.txt ya existe,¿crear copia?(S/N) d
Error: Complete Sí o No

salida.txt ya existe,¿crear copia?(S/N) s
Ingrese nombre para la copia: salida_anterior.txt

Se creo la copia de salida.txt en salida_anterior.txt

Copiando del archivo 'entrada.txt'...
Copia completa en 'salida.txt'.
```

Se obtendría el resultado:

Contenido en 'salida.txt'

```
ESTO ES UN ARCHIVO DE PRUEBA↲
CON MULTIPLES LINEAS↲
EL CUAL SERA COPIADO↲
EN UN NUEVO ARCHIVO↲
CON TODAS LAS LETRAS EN↲
MAYUSCULA.↲
```

2. Ámbito `with`

`Python` ofrece una palabra reservada (*ketword*) `with`, la cual abre un ámbito con el cual podemos trabajar con un archivo:

Programa 7: Utilizar `with` para trabajar con archivos.

```
1 with open("personas.txt") as archivo:
2   nombre = archivo.readline()
3   while nombre:
4     telefono = archivo.readline()
5     print(f"{nombre} -> {telefono}")
6     nombre = archivo.readline()
7 # fin del ámbito with, no hace falta llamar a close()
```

En este caso el uso del archivo está restringido al ámbito de `with`. Esto restringe la utilización del archivo y además lo cierra automáticamente, sin la necesidad de utilizar `close()`.

3. Ejercitación

Codifique en `Python` los siguiente programas utilizando arreglos, registros y archivos.

1. Desarrolle un software que pida un nombre para un archivo de texto y permita ingresar y almacenar en dicho archivo todo lo que se escriba a continuación hasta que el usuario ingrese la cadena `'FinalizarIngreso'` (con `'F'` e `'I'` en mayúscula).

2. Desarrolle un software que sea capaz de leer datos de hasta 20 equipos de futbol para poder generar una tabla de posiciones con la siguiente información:

   - Nombre del equipo
   - Total de partidos jugados
   - Cantidad de partidos ganados
   - Cantidad de partidos empatados
   - Cantidad de partidos perdidos

- Cantidad de goles a favor
- Cantidad de goles a en contra
- Diferencia de goles entre los a favor y en contra
- Puntos totales

El software debe mostrar los equipos en orden descendente según la cantidad de puntos que tengan, sabiendo que los puntos son: 3 por partido ganado, 1 por empate y 0 por perdidos. Los datos necesarios para generar la tabla están almacenados en un archivo de texto plano **'liga.dat'** con el siguiente formato:

```
1  Cangallo FC
2  6
3  2
4  2
5  8
6  4
7
8  Cebollitas
9  4
10 1
11 5
12 4
13 7
```

Donde **línea 1:** nombre del equipo; **línea 2:** partidos ganados; **línea 3:** partidos empatados; **línea 4:** partidos perdidos; **línea 5:** goles a favor; **línea 6:** goles en contra. Una línea vacía indica una separación entre equipos.

3. Programe una agenda donde pueda ingresar nombres, apellidos, fecha de cumpleaños, e-mail y teléfono de hasta 100 personas. Guarde sus datos en un archivo de texto **'agenda.dat'** del que pueda recuperar los datos en cada ejecución y agregar más personas.

4. Al punto anterior agregue la posibilidad de modificar los *registros* ya existentes.