7. Lectura y Escritura

7.1. Lectura

Para abrir un archivo de texto plano en Python se puede usar las siguientes estructuras, donde se indica abrir el Archivo.txt que está en la ubicación señalada, etiquetarlo con la variable file, realizar las instrucciones que estén el bloque de lectura y cerrar el archivo.

```
# Leer Archivo

with open("Ubicación/Archivo.txt") as file:
    Instrucción_1
    Instrucción_2
    ...
    Instrucción_n
    file.close()
```

- Una estructura alternativa es la siguiente:

```
# Leer archivo
```

```
file = open("Ubicación/Archivo.txt")
Instrucción_1
Instrucción_2
...
Instrucción_n
file.close()
```

Los archivos tienen algunas de las propiedades de un iterable donde cada línea es un elemento, por lo cual, se pueden recorrer con un ciclo for, sin embargo, no se puede acceder a estos pro medio de los índices.

Se debe tener en cuenta que todo elemento en archivo es un str, por lo cual, si el archivo contiene datos numéricos que se deben procesar estos deben convertirse a int o float.

Por último, como cada línea del documento es un archivo, los salto de línea también quedan registrados como se muestra en la impresión de la lista del siguiente Mini_Ejemplo.

```
# Mini_Ejemplo: Lectura Archivo_0.txt

file = open("Archivo_0.txt")
lista=[]
for i in file:
    print(i)
    lista.append(i)

file.close()

print("\nLista=",lista)
```

```
# Pase aquí el mini ejemplo anterior
```

7.1.1. Ubicación de Archivos

Cuando los archivos están en la misma ubicación que el programa solamente se debe indicar el nombre del Archivo.txt, como se vio en el Mini_Ejemplo anterior.

Sin embargo, cuando el archivo está en una ubicación diferente hay que conocer la ruta relativa o absoluta para poder abrirlo, en esta guía solo se van a manejar rutas relativas.

Ojo: En el caso en que el archivo sea ingresado por el usuario, se le debe pedir la ruta relativa o indicarle que copie y pegue el archivo donde tenga guardado el programa.

7.1.1.1. Ruta Relativa

La ruta relativa de un archivo como dice su nombre es relativa a la ubicación de partida, en este caso es la carpeta del programa, es decir, que la ruta es construida partiendo desde la carpeta 7_Lectura_y_Escritura hasta llegar a la ubicación del archivo que se requiera.

Imagine que tiene está disposición de carpetas:

```
C_Superior_2
|..Archivo_s2.txt
|..C_Superior_1
|..|..Archivo_s1.txt
|..|..C_Programa (7_Lectura_y_Escritura)
|..|..|..Archivo_0.txt
|..|..|..C_Inferior_1 (Archivos)
|..|..|..|..Archivo_i1.txt
|..|..|..|..C_Inferior_2
|..|..|..|..|..Archivo_i2.txt
```

- **Archivos en carpetas superiores:** Para acceder a estos archivos hay que recorrer hacía atrás las carpetas, para esto se utiliza **../**, por ejemplo, la ruta del Archivo_s2.txt es: "**. . / . . /Archivo_s2.txt**"

```
# Mini_Ejemplo: Lectura Archivo_s1.txt

file = open("../Archivo_s1.txt")

for i in file:
    print(i)

file.close()
```

```
# Pase aquí el mini ejemplo anterior
```

- **Archivos en carpetas inferiores:** Para acceder a estos archivos hay que recorrer hacía adelante escribiendo el nombre de las carpetas a las cuales se necesite entrar, por ejemplo, la ruta del Archivo_i2.txt es: "**C_Inferior_1/C_Inferior_2/Archivo_i2.txt**"

```
# Mini_Ejemplo: Lectura Archivo_i1.txt

file = open("Archivos/Archivo_i1.txt")
```

```
for i in file:
    print(i)

file.close()
```

```
# Pase aquí el mini ejemplo anterior
```

7.1.1.2. Validación de Ruta

Como es bastante probable que el usuario cometa un error al ingresar la ruta lo más recomendable es tener una función para validarla y poder manejar la excepción **FileNotFoundError**.

```
# Mini_Ejemplo: Validar Ubicación

def v_ubicacion():
    mensaje="No se ha encontrado el archivo, por favor verifique la ruta ingresada."
    linea="\n"+str(len(mensaje)*"-")+"\n"
    try:
        ruta=input("Por favor ingrese la ruta del archivo que quiere abrir\npor ejemplo: 'Archi
        return open(ruta)
    except(FileNotFoundError):
        print(linea+mensaje+linea)
        return v_ubicacion()

    return None

# Uso
file=v_ubicacion()
for i in file:
    print(i)
```

```
# Pase aquí el mini ejemplo anterior
```

7.1.2. Almacenar Datos

Para almacenar los datos de los archivos se debe tener en cuenta su contenido, como está organizado y cuál es la estructura de datos más adecuada para el programa.

Lo ideal es que los archivos de texto cumplan las siguientes reglas.

- No tener rótulos, es decir, que el primer dato sea la primera línea del programa.
- Si contiene números decimales que todos contengan la misma cantidad y que estén señalados con punto (.) y **No** con coma (,).
- No tener saltos de línea en blanco ni siquiera al final, es decir, que el último dato sea la última línea del programa.
- Los conjuntos de datos deben estar organizados por columnas, es decir, si el archivo tiene un conjunto de 100 datos, entonces debe tener una columna de 100 filas.
- Si los conjuntos son dependientes como coordenadas o bases de datos lo mejor es que estén en mismo archivo, de lo contrario es preferible que estén en archivos separados y hacer varias lecturas.
- Tener un separador diferente al espacio cuando contiene base datos, por ejemplo, usar punto y coma (;) o tabulador.
- Si un archivo tiene más de un conjunto de datos lo más recomendable es que los conjuntos tengan la misma cantidad de datos.

A continuación, hay algunos ejemplos de cómo almacenar los datos de archivos con distintos formatos.

- Archivo_i1: Conjunto de datos en una columna

```
# Mini_Ejemplo: Almacenar Archivo_i1.txt

datos_1=[]
file = open("Archivos/Archivo_i1.txt")

for i in file:
    datos_1.append(int(i))

file.close()

print(datos_1)
```

```
# Pase aquí el mini ejemplo anterior
```

- Archivo_i2: Conjunto de datos en una fila

```
# Mini_Ejemplo: Almacenar Archivo_i2.txt

datos_2=[]

file = open("Archivos/Archivo_i2.txt")
for i in file:
    datos_2=i.split()
    datos_2 = [float(x) for x in datos_2] # Conveirta cada elemento x de datos en float.
file.close()

print(datos_2)
```

```
# Pase aquí el mini ejemplo anterior
```

- Archivo_i3: Dos conjuntos de datos no relacionados en columnas, con rótulos, saltos de línea intermedios, y decimal señalado con coma (,).

```
# Mini_Ejemplo: Almacenar Archivo_i3.txt

datos_3=[]
d_Pieza_A=[]
d_Pieza_B=[]
lista=[]
c=0

file = open("Archivos/Archivo_i3.txt")
for i in file:

    lista=i.split("\t")
    if c>0 and lista!=["\n"]:
        d_Pieza_A.append(float(lista[0].replace(",",".")))
        d_Pieza_B.append(float(lista[1].replace(",",".")))
    c+=1

file.close()

datos_3=[d_Pieza_A,d_Pieza_B]
print(datos_3)
```

```
# Pase aquí el mini ejemplo anterior
```

- **Coordenadas_x_y.txt**: Dos conjuntos de datos no relacionados en columnas, con rótulos, saltos de línea intermedios, y decimal señalado con coma (,).

```
# Mini_Ejemplo: Almacenar Coordenadas_x_y.txt

datos_4=[]
lista=[]

file = open("Archivos/Coordenadas_x_y.txt")
for i in file:

    lista=i.split("\t")
    lista=[float(x) for x in lista]
    datos_4.append(lista)
file.close()

print(datos_4)
```

```
# Pase aquí el mini ejemplo anterior
```

- **B_Datos.txt**: Dos conjuntos de datos no relacionados en columnas, con rótulos, saltos de línea intermedios, y decimal señalado con coma (,).

```
# Mini_Ejemplo: Almacenar B_Datos.txt

datos_5=[]
lista=[]

file = open("Archivos/B_Datos.txt")
for i in file:

    lista=i.split(";")
    datos_5.append(lista)
file.close()

datos_5=[[x[0],int(x[1]),x[2],int(x[3])] for x in datos_5] # Convertir los elementos con índice

print(datos_5)
```

```
# Pase aquí el mini ejemplo anterior
```

7.2. Escritura

Para escribir un archivo se utiliza una estructura parecida para la lectura, sin embargo, se le indica a la función que el modo de apertura es para modificar con "w"; en caso de que el archivo no exista se creará uno nuevo en la ruta y con el nombre indicado, si el archivo ya existe este será sobrescrito y se borrará todo o parte del contenido original, por lo tanto, si se desea empezar a escribir al final del documento, el modo de apertura que se debe utilizar es "a".

El método write contiene lo que se debe escribir en el documento, tenga en cuenta que siempre debe ser tipo str.

```
# Mini_Ejemplo: Escribir Arch_Ejemplo.txt

with open("Arch_Ejemplo.txt","w") as file:
    file.write("Este es un archivo.txt de ejemplo generado en Python")
    file.close()
```

También válido escribirlo así:

```
# Mini_Ejemplo: Escribir Arch_Ejemplo.txt
file = open("Arch_Ejemplo.txt","w")
file.write("Este es un archivo de ejemplo generado en Python")
file.close()
```

```
# Pase aquí el mini ejemplo anterior, una vez ejecutado busque el archivo en la misma carpeta d
```

7.2.1. Crear Carpetas

Para crear una carpeta en Python se debe usar la función `mkdir()` del módulo **os**, indicando la ruta donde va a ser creada, si es en la misma ubicación del programa solo basta escribir el nombre.

El siguiente ejemplo muestra cómo crear un directorio llamado Carpeta_Ejemplo dentro de Archivos y guardar un documento en la carpeta creada.

```
# Mini_Ejemplo: Crear Carpeta_Ejemplo

import random
import os
```

```
os.mkdir("Archivos/Carpeta_Ejemplo")

file = open("Archivos/Carpeta_Ejemplo/Numeros_Normales.txt","w")

for i in range(1000):
    # Escribir un número normal con media 100 y desviación 0.2, redondeado a 3 decimales y con
    file.write(str(round(random.normalvariate(100,0.2),3))+"\n")
file.close()
```

```
# Pase aquí el mini ejemplo anterior
```

Para saber si una carpeta ya existe se puede usar el método `path.isdir()` del módulo **os**

```
import os
os.path.isdir("Archivos")
```

```
# Pase el código anterior aquí
```

En el caso de que ya exista una carpeta con el nombre indicado se la lanzará la excepción **FileExistsError**, el siguiente Mini_Ejemplo muestra una función para validar la existencia de una carpeta en la ubicación indicada.

```
# Mini_Ejemplo: Validar Carpeta

import os

def v_carpeta():
    mensaje="La carpeta ya existe, intente con otro nombre o cambie la carpeta existente de nom
    linea="\n"+str(len(mensaje)*"-")+"\n"
    try:
        ruta=input("Por favor ingrese la ruta de la carpeta que quiere crear,\npor ejemplo, 'Ar
        os.mkdir(ruta)
        print("La carpeta ha sido creada")
    except(FileExistsError):
        print(linea+mensaje+linea)
        return v_carpeta()

    return None
```

```
# Uso
v_carpeta()
```

```
# Pase aquí el mini ejemplo anterior
```

7.3. Crear Archivos .xlsx

Existen varios módulos para crear un archivo de Excel .xlsx, en esta guía se va a explicar cómo hacer con **pandas** usando **xlsxwriter** como motor de escritura.

Para más información sobre el manejo de archivos de excel con pandas ingrese aquí y para ir la documentación oficial del módulo **xlswriter** de click aquí.

A continuación, se muestran los pasos para crear un Archivo.xlsx.

7.3.1. Crear los DataFrames

Antes de crear el archivo se debe organizar la información en una estructura de datos especial llamada DataFrame, la cual es ampliamente utilizada en el lenguaje estadístico R, ya que simula una matriz que puede contener diferentes tipos de datos, por lo cual es perfectamente adecuada para recrear una tabla, como una hoja de excel.

Hay diferentes maneras de crear un DataFrame, en esta guía se mostrará cómo hacerlo desde un diccionario, con la función DataFrame() de pandas.

Tenga en cuenta que cada pareja clave:valor del diccionario es una columna, donde la clave es el encabezado y el valor puede ser un solo dato o una lista de datos, por ejemplo, el siguiente diccionario, al convertirse en DataFrame simularía la tabla de abajo.

```
vetas_2018={"Trimestre":"I II III IV".split(),"Ventas (miles de millones de pesos)":[2.345,1.50
```

Trimestre	Ventas (miles de millones de pesos)
I	2.345
II	1.50

Trimestre	Ventas (miles de millones de pesos)
III	1.62
IV	6.018

```
# Importar Pandas

import pandas as pd

# Crear DataFrames

df_v_2018=pd.DataFrame({"Trimestre":"I II III IV".split(),"Ventas (miles de millones de pesos)"

df_v_2019=pd.DataFrame({"Trimestre":"I II III IV".split(),"Ventas (millones de pesos)":[1.982,1
print(type(df_v_2018))
```

```
# Pase el código anterior aquí.
```

7.3.2. Crear y Editar el Documento

Ya creados los DataFrames se puede proceder a crear el Achivo.xlsx, para esto se hace uso de la función `ExelWriter()` de Pandas.

```
ExcelWriter(path, engine=None, date_format=None, datetime_format=None, mode='w', **engine_kwarg
```

Dónde los argumentos más importantes para el desarrollo de la guía son:

- **path**: Ruta del archivo. (Este argumento es obligatorio)
- **engine**: Motor de escritura, en este caso se va a utilizar "xlsxwriter"

Nota: Para más información sobre esta función ingrese **aquí**

```
# Crear el documento
doc=pd.ExcelWriter('Archivos/Ventas_ARB3354.xlsx', engine='xlsxwriter')
```

Para escribir los DataFrames en el archivo de excel se usa el método `.to_excel` de Pandas.

```
DataFrame.to_excel(self, excel_writer, sheet_name='Sheet1', na_rep='', float_format=None, colum
```

Dónde los atributos más importantes para el desarrollo de la guía son:

- **self**: Nombre de la variable con la que se creó el documento. (Este argumento es obligatorio)
- **sheet_name**: Nombre de la hoja donde se va escribir el DataFrame, se creará en caso de que no exísta.
- **header**: Si es True muestra los encabezados (Las claves del diccionario), si es False no.
- **index**: Si es True muestra el índice de las filas, si es False no.
- **startrow**: La fila donde se empieza a escribir, por defecto es 0.
- **startcol**: La columna donde se empieza a escribir, por defecto es 0.

```
# Escribir DataFrames

df_v_2018.to_excel(doc, sheet_name='2018',index=False, startrow=2)
df_v_2019.to_excel(doc, sheet_name='2019',index=False, startrow=2)
```

7.3.3. Aplicar Títulos y Formatos

Por último por cuestiones de presentación algunas veces se necesite poner títulos o aplicar un formato algún formato al documento, para lo cual, primero se debe abrir el archivo y las hojas como objetos en Python con el método `.book` y `.sheet`.

```
# Abrir objetos del documento

libro=doc.book # Abrir libro
hoja_1=doc.sheets['2018'] # Abrir la hoja 1
hoja_2=doc.sheets['2019'] # Abrir la hoja 2
```

Luego se deben crear los formatos que se van a aplicar con el método `.add_format()`, el cual tiene los siguientes atributos entre muchos otros:

- '**num_format**': A través de un patrón ('#,##0.000') indica si se debe utilizar punto o coma para los miles o decimales, y la cantidad de decimales.
- '**blod**': Negrita, se activa y desactiva con True o False.
- '**italic**': Itálica, se activa y desactiva con True o False.
- '**underline**': Subrayado, se activa y desactiva con True o False.

- '**font_color**': Color de la fuente
- '**font_name**': Tipo de fuente
- '**font_size**': Tamaño de la fuente
- '**bg_color**': Color de la celda

Nota: Para ver todos los atributos diríjase **aquí**

```
# Crear formatos

formato_numeros=libro.add_format({'num_format': '#,##0.000'})
formato_titulo=libro.add_format({'bold': True, 'font_size': 16})
```

Una vez creados los formatos solo queda usarlos, guardar y cerrar el documento, para esto se emplean los métodos `.set_colum()`, `.set_row()`, y `.write()`, cuya documentación encuentra aquí.

```
set_column(first_col, last_col, width, cell_format, options)
set_row(row, height, cell_format, options)
write(row, col, *args)
```

- **first_col:last_col**: Segmento de columnas, puede ser escrito con números o con letras, por ejemplo: "1:4" o "A:D"
- **width**: Ancho de la columna.
- **cell_format**: Formato de la celda.
- **row**: Fila.
- **height**: Altura de la fila.
- **col**: Columna.
- ***args**: Contenido que se quiere escribir y formato.

```
# Aplicar formatos

formato_numeros=libro.add_format({'num_format': '#,##0.00'})
ancho_1=len("Trimestre")
ancho_2=len("Ventas (miles de millones de pesos)")

hoja_1.write(0, 0, 'Ventas Trimestrales del Código ARB3354 en 2018', formato_titulo) # Escribir
hoja_1.set_column('A:A', ancho_1)
hoja_1.set_column('B:B', ancho_2, formato_numeros)

hoja_2.write(0, 0, 'Ventas Trimestrales del Código ARB3354 en 2019', formato_titulo) # Escribir
hoja_2.set_column('A:A', ancho_1)
```

```
hoja_2.set_column('B:B', ancho_2, formato_numeros)

# Guardar documento.

doc.save()

# Cerrar el documento.

doc.close()
```

Mini_Ejemplo: Archivo.xlsx

A continuación, está la recopilación de los pasos explicados anteriormente en un Mini_Ejemplo.

```
# Mini_Ejemplo: Archivo.xlsx

# Importar Pandas

import pandas as pd

# Crear DataFrames

df_v_2018=pd.DataFrame({"Trimestre":"I II III IV".split(),"Ventas (miles de millones de pesos)"

df_v_2019=pd.DataFrame({"Trimestre":"I II III IV".split(),"Ventas (millones de pesos)":[1.982,1

# Crear el documento

doc=pd.ExcelWriter('Archivos/Ventas_ARB3354.xlsx', engine='xlsxwriter')

# Escribir DataFrames

df_v_2018.to_excel(doc, sheet_name='2018',index=False, startrow=2)
df_v_2019.to_excel(doc, sheet_name='2019',index=False, startrow=2)

# Abrir objetos del documento

libro=doc.book # Abrir libro
hoja_1=doc.sheets['2018'] # Abrir la hoja 1
hoja_2=doc.sheets['2019'] # Abrir la hoja 2

# Crear formatos

formato_numeros=libro.add_format({'num_format': '#,##0.000'})
formato_titulo=libro.add_format({'bold': True, 'font_size': 16})
```

```
# Aplicar formatos

formato_numeros=libro.add_format({'num_format': '#,##0.00'})
ancho_1=len("Trimestre")
ancho_2=len("Ventas (miles de millones de pesos)")

hoja_1.write(0, 0, 'Ventas Trimestrales del Código ARB3354 en 2018', formato_titulo) # Escribir
hoja_1.set_column('A:A', ancho_1)
hoja_1.set_column('B:B', ancho_2, formato_numeros)

hoja_2.write(0, 0, 'Ventas Trimestrales del Código ARB3354 en 2019', formato_titulo) # Escribir
hoja_2.set_column('A:A', ancho_1)
hoja_2.set_column('B:B', ancho_2, formato_numeros)

# Guardar el documento.

doc.save()

# Cerrar el documento.

doc.close()
```

```
# Pase aquí el mini ejemplo anterior, una vez ejecutado busque en la carpeta Archivos el docume
```

Anterior - *Siguiente*

Home