

Manejo de Archivos

Leyendo y escribiendo datos en archivos

Jonatan Gómez Perdomo, Ph. D.

jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.

aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)

eccubidesg@unal.edu.co

Carlos Andrés Sierra, M.Sc.

casierrav@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)

Computer and System Department

Engineering School

Universidad Nacional de Colombia

Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo `pickle`
- 10 Copiar un archivo binario



Definición

- La **memoria** del computador permite almacenar datos (valores y programas) con los que el computador puede ejecutar una tarea.
- Existen dispositivos externos a la memoria donde se pueden guardar datos que pueden ser usados por el computador. Estos pueden ser discos duros, cintas, memorias SD, USB, CD's, entre otras.
- Un **archivo** es una unidad lógica de almacenamiento de datos en dispositivos externos (no memoria principal). Se dice *unidad lógica* porque todos los bytes almacenados en un archivo se ven como una sola unidad que tiene un inicio y un fin, sin importar lo que esos bytes estén representando.
- El **tamaño** (coloquialmente **peso**) de un archivo es el número de bytes que almacena.



Tipos

Aunque existen muchos tipos de archivos, los programadores los clasifican en dos tipos básicos:

Texto : Cuando cada byte representa un ASCII (en algunos casos UTF-8 o UTF-16) como los archivos `.txt`; los de páginas web `.html`; los de código fuente en muchos lenguajes de programación `.py`, `.c`, `.java`, `.cpp`, `.tex`; los de formato abierto para representar datos en forma de tabla como `.csv`.

Binario : Cuando cada byte representa algo diferente o requiere un conjunto de bytes para representar algo, como los archivos de imagen `.jpg`, `.png`, de documento `.doc`, `.pdf`, `.xls`, `.ods`, comprimidos `.rar`, `.zip`, entre otros.



Operaciones

Existen cuatro (4) operaciones que se pueden realizar con un archivo desde un programa:

Crear : Crear un archivo en un dispositivo externo desde el programa.

Abrir : Darle permiso al programa de acceder al contenido almacenado por el archivo en diferentes **modos**, ya sea leer (traer datos del archivo al programa), escribir (guardar datos del programa en el archivo), adicionar (agregar datos del programa al final del archivo), y/o moverse.

Cerrar : Remover al programa el permiso de leer, escribir, adicionar, y/o moverse en el contenido del archivo.

Eliminar : Remover desde el programa un archivo de un dispositivo externo.



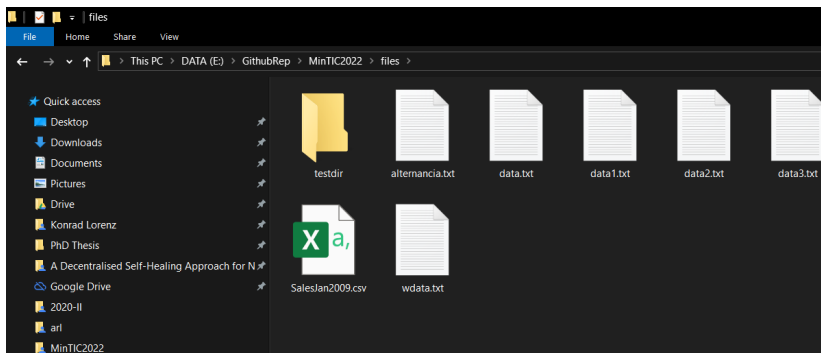
Agenda

- 1 Introducción
- 2 Configuración de esta sesión**
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo pickle
- 10 Copiar un archivo binario



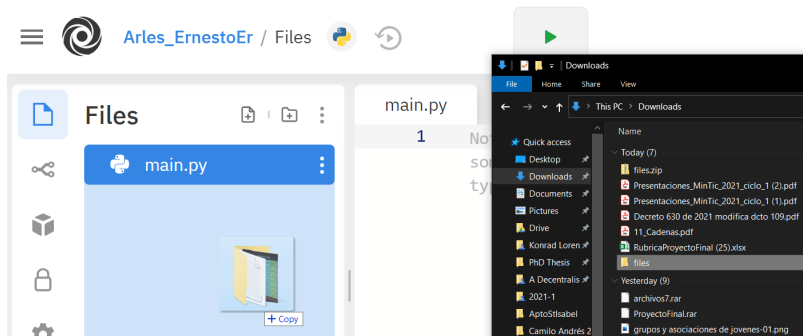
Descargar archivos de ejemplo

Los archivos de ejemplo de esta sesión se descargarán del siguiente enlace:
<https://github.com/arleserp/MinTIC2022/raw/master/files.zip>
Después de descomprimir, así luce la carpeta files en el computador:



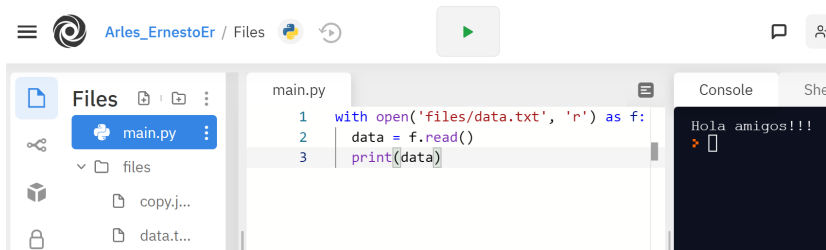
Cargar carpeta en replit

Ahora se arrastra la carpeta a la sección donde se encuentran los scripts de replit:



Ejecutar códigos de esta presentación!

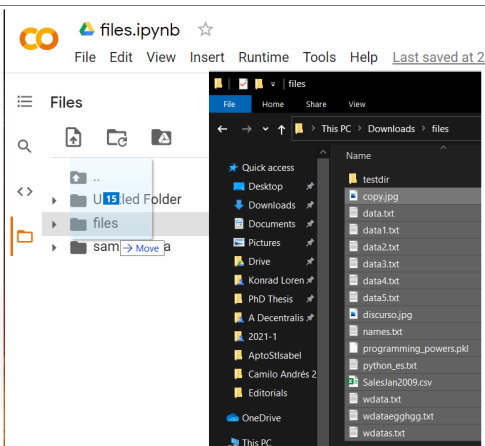
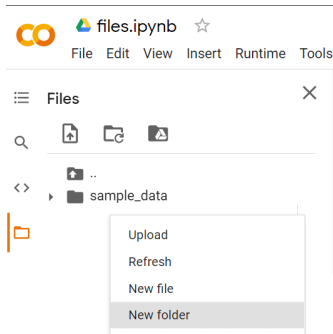
¡Cualquier código de esta presentación está listo para ser probado en replit!



Cargar carpeta en colab

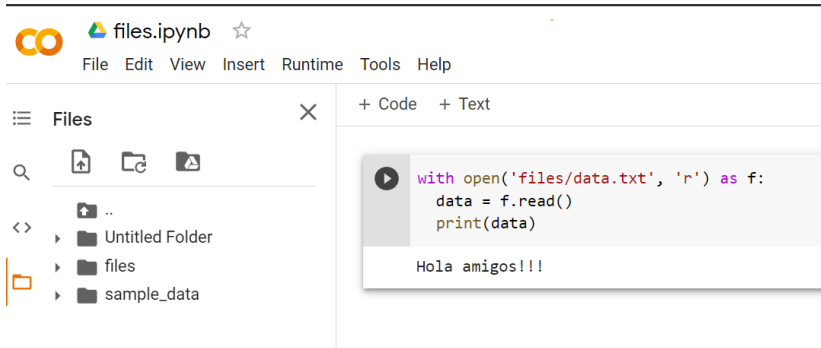
Si se desea usar colab una forma de utilizar archivos es:

- Crear la carpeta files.
- Arrastrar los archivos dentro de la carpeta files a colab.



Ejecutar códigos de esta presentación!

¡Cualquier código de esta presentación está listo para ser probado en colab!



Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos**
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo pickle
- 10 Copiar un archivo binario



Direccionamiento

Es posible trabajar con direccionamiento absoluto o relativo en Python. En los siguientes ejemplos se realizará direccionamiento relativo a partir de la carpeta donde se ejecuta el código. En esta carpeta hay una carpeta llamada `files` y por ejemplo, allí se encuentra el archivo `'data.txt'`.

Para dar soporte a los diferentes sistemas operativos como Windows y Linux, y trabajar con direccionamiento absoluto se toma como nombre del archivo la ruta completa del archivo:

```
"C:\\ThisFolder\\workData.txt"  
"C:/ThisFolder/workData.txt"
```



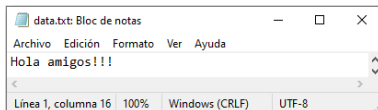
Abrir y leer un archivo (`open` y `with`) I

El comando `open(ruta del archivo, 'r')` toma un nombre de archivo y modo como argumentos. `r` abre el archivo en modo de lectura (*read*). Adicionalmente y como buena práctica de programación en Python se utilizará el comando `with` que permite cerrar el archivo automáticamente y asegura un uso propio del archivo, mientras éste está bajo el alcance del entorno del comando `with`.



Abrir y leer un archivo (open y with) II

Para el archivo `data.txt` ubicado en la carpeta `files`



El siguiente código abre el archivo y muestra su contenido:

```
with open("files/data.txt", "r") as f:  
    data = f.read()  
    print(data)
```

La salida obtenida es

```
Hola amigos!!!
```



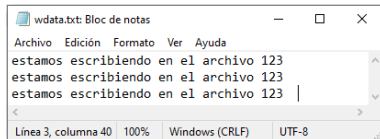
Escribir y crear un archivo (open y with)

El comando `open(ruta del archivo, 'w')` abre el archivo en modo de escritura creando el archivo si no existe (*write*) ó sobrescribiendo el archivo si existe. El siguiente código crea el archivo `wdata.txt` en la carpeta `files`:

```
with open("files/wdata.txt", "w") as f:  
    data = "estamos escribiendo en el archivo 123\n"  
    f.write(data)  
    f.write(data)  
    f.write(data)
```



El archivo obtenido es



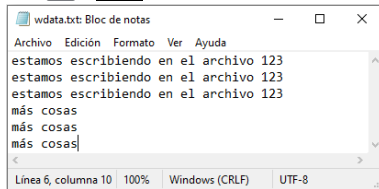
Agregar contenido a un archivo existente (open y with)

El comando `open(ruta del archivo, 'a')` abre el archivo en modo de adjuntar creando el archivo si no existe (*append*) ó escribiendo al final del archivo si existe. El siguiente código escribe al final del archivo `wdata.txt`:

```
with open("files/wdata.txt", "a") as f:  
    data = "más cosas\n"  
    f.write(data)  
    f.write(data)  
    f.write(data)
```



El archivo obtenido es



Otros modos de apertura de archivos

Otros modos de abrir archivos en Python son:

- 'r+' – (*read/write*): Este modo es usado cuando se desean hacer cambios al archivo y a la vez leer información. El puntero del archivo se ubica al principio del archivo.
- 'a+' – (*append/read*): Se abre el archivo para escribir y leer del archivo. El puntero del archivo se ubica al final del archivo.
- x (*exclusive creation*): usado exclusivamente para crear un archivo. Si existe un archivo con el mismo nombre la función fallará.
- b : Para leer archivos binarios se agrega la letra b al final del modificador. Por ejemplo, para leer “rb”, y los otros serían “wb”, “ab”, “r+b” y “a+b”.



Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos**
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo pickle
- 10 Copiar un archivo binario



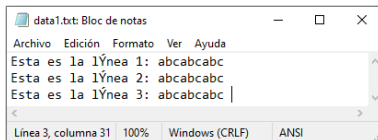
Lectura de un archivo de texto (read) I

Para leer el contenido de un archivo se puede usar `read()`. Sin parámetros, leerá el archivo entero y almacenará el contenido la salida como un `string`, si es un archivo de texto o como objetos en bytes si se trata de un archivo binario. Si se trata de leer un archivo más grande que la memoria disponible no se podrá leer todo el archivo de una vez.



Lectura de un archivo de texto (read) II

Para el archivo `data1.txt` ubicado en la carpeta `files`



El siguiente código lo abre, lee su contenido completo y lo imprime:

```
with open("files/data1.txt", "r") as f:  
    print(f.read())
```

La salida obtenida podría ser

```
Esta es la lÍnea 1: abcabcbc  
Esta es la lÍnea 2: abcabcbc  
Esta es la lÍnea 3: abcabcbc
```



Lectura de un archivo con tildes (read) I

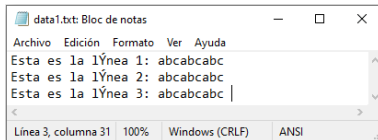
En caso de tener problemas de codificación se puede agregar el parámetro `encoding="utf-8"` al comando `with open`. El siguiente código abre el archivo para lectura y especifica que tiene como juego de caracteres la codificación `utf-8`:

```
with open("files/data1.txt", "r", encoding="utf-8") as f:
```



Lectura de un archivo con tildes (read) II

Para el archivo `data1.txt` ubicado en la carpeta `files`



El siguiente código lo abre, lee su contenido completo y lo imprime:

```
with open("files/data1.txt", "r", encoding="utf-8") as f:  
    print(f.read())
```

La salida obtenida es

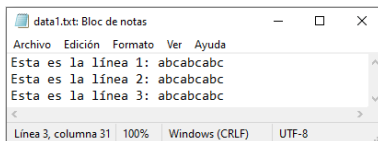
```
Esta es la línea 1: abcabcabc  
Esta es la línea 2: abcabcabc  
Esta es la línea 3: abcabcabc
```



Leer byte a byte (read) I

En algunos casos no se desea leer la totalidad del contenido de un archivo, sino solamente una fracción, en ese caso el comando `read()` admite un parámetro entero que especifica cuantos bytes se leen y deja el apuntador justo después del último byte leído.

Ahora, para el archivo `data1.txt` ubicado en la carpeta `files`



Leer byte a byte (read) II

Si se desea leer en primera instancia los primeros 6 bytes y a continuación los siguientes 10 bytes, se puede utilizar la instrucción:

```
with open("files/data1.txt", "r", encoding="utf-8") as f:  
    linea1 = f.read(6)  
    linea2 = f.read(10)  
  
print(linea1)  
print(linea2)
```

La salida obtenida es

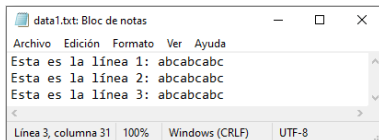
```
Esta e  
s la línea
```



Leer archivo línea por línea I

Es posible leer un archivo línea por línea utilizando el comando `readline()`. Esta es una forma eficiente en memoria de leer un archivo:

Para el archivo `data1.txt` ubicado en la carpeta `files`



Leer archivo línea por línea II

El siguiente código abre el archivo, lee el contenido de las dos primeras líneas y las imprime con un sólo salto de línea entre ellas:

```
with open("files/data1.txt", "r", encoding="utf-8") as f:  
    linea1 = f.readline()  
    linea2 = f.readline()  
  
print(linea1, end="")  
print(linea2, end="")
```

La salida obtenida es

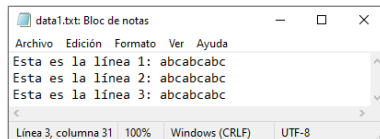
```
Esta es la línea 1: abcabcabc  
Esta es la línea 2: abcabcabc
```



Cargar las líneas de un archivo como una lista I

Es posible cargar todas las líneas del archivo como una lista si se utiliza el comando `readlines()`.

Para el archivo `data1.txt` ubicado en la carpeta `files`



Cargar las líneas de un archivo como una lista II

El siguiente código abre el archivo, lee el contenido completo del archivo y lo almacena en una lista línea por línea:

```
with open("files/data1.txt", "r", encoding="utf-8") as f:  
    print("Nombre del archivo: ", f.name)  
    lista = f.readlines()  
  
print(lista)
```

La salida obtenida es

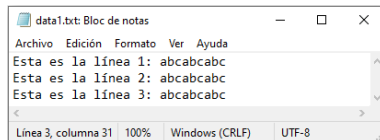
```
Nombre del archivo:  files/data1.txt  
['Esta es la línea 1: abcabcabc\n', +  
'Esta es la línea 2: abcabcabc\n', +  
'Esta es la línea 3: abcabcabc ']
```



Procesar un archivo de forma eficiente I

La forma más sencilla y eficiente, de abrir y procesar un archivo de texto es iterar línea por línea el archivo, así como se muestra a continuación:

Para el archivo `data1.txt` ubicado en la carpeta `files`



Procesar un archivo de forma eficiente II

El siguiente código permite procesar eficientemente el contenido del archivo

```
with open("files/data1.txt", "r", encoding="utf-8") as f:  
    for line in f:  
        print(line, end="")
```

La salida obtenida es

```
Esta es la línea 1: abcabcabc  
Esta es la línea 2: abcabcabc  
Esta es la línea 3: abcabcabc
```



Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos**
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo pickle
- 10 Copiar un archivo binario

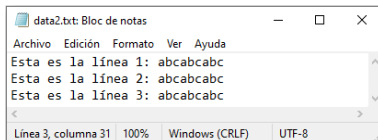


Escritura de archivos I

A menudo se prefiere utilizar `a+` como el modo de apertura para escribir un archivo porque permite añadir datos al final del mismo. Utilizar `w+` borrará cualquier dato que existe en el archivo y pues producirá un archivo en limpio para trabajar.

El método por defecto para añadir datos en un archivo es `write`:

Para el archivo `data2.txt` ubicado en la carpeta `files`

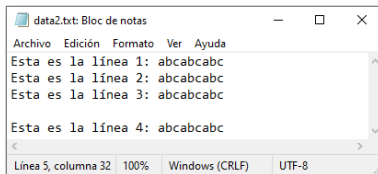


Escritura de archivos II

El siguiente código permite añadir una nueva línea al final del archivo

```
with open("files/data2.txt", "a+", encoding="utf-8") as f:  
    f.write("\nEsta es la línea 4: abcabcabc")
```

El resultado obtenido es el archivo data2.txt modificado



Escribir datos que no son de tipo cadena en un archivo I

Si se desea escribir datos que no son cadenas de texto en un archivo es necesario convertir cada dato a `string` con el comando `str()` así como se muestra a continuación.

El siguiente código permite escribir en el archivo `data3.txt` una serie de datos almacenados en una lista luego de convertirse cada uno a cadena.

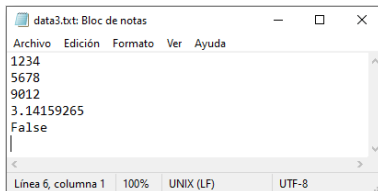
```
values = [1234, 5678, 9012, 3.14159265, False]

with open("files/data3.txt", "w+") as f:
    for value in values:
        str_value = str(value)
        f.write(str_value)
        f.write("\n")
```



Escribir datos que no son de tipo cadena en un archivo II

El resultado obtenido es el archivo data3.txt



```
data3.txt: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
1234
5678
9012
3.14159265
False
|
Línea 6, columna 1  100%  UNIX (LF)  UTF-8
```



Seek: ubicarse en un archivo I

A veces es necesario moverse al principio del archivo o a alguna posición específica en un archivo dado. La forma más fácil de hacerlo es utilizar el comando o método

```
seek(offset, from_what)
```

`offset` es el número de caracteres desde la posición `from_what`.

Hay algunos valores por defecto para la posición `from_what` así:

0 : Principio del archivo

1 : Posición actual

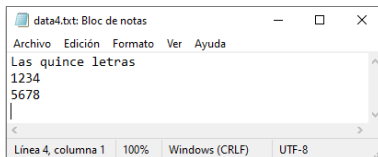
2 : Al final del archivo

Con archivos de texto 0 ó `seek(0,2)`, lo llevarán al final del archivo.



Seek: ubicarse en un archivo II

Para el archivo data4.txt ubicado en la carpeta files



Seek: ubicarse en un archivo III

El siguiente código permite ubicar el puntero en la posición 11 dentro del archivo

```
with open("files/data4.txt", "r", encoding="utf-8") as f:  
    f.seek(11,0)  
    for line in f:  
        print(line, end="")
```

La salida obtenida es

```
letras  
1234  
5678
```



Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés**
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo pickle
- 10 Copiar un archivo binario



Determinar tamaño de un archivo en bytes

Para obtener la longitud de un archivo o la posición en donde se encuentra dentro del archivo se puede usar el método o comando

```
tell()
```

Para determinar el tamaño del archivo `data4.txt` en bytes se puede utilizar la instrucción

```
with open("files/data4.txt", "a+") as f:  
    print(f.tell())
```

La salida de este programa es

```
29
```



Agenda

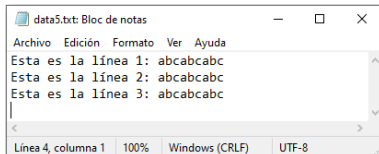
- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto**
- 8 Listar archivos y directorios
- 9 El módulo pickle
- 10 Copiar un archivo binario



Edición de archivos I

La forma más fácil de editar un archivo dado es guardar el archivo entero en una lista y utilizar el comando `list.insert(i,x)`, que permite insertar nuevos datos a la lista. Una vez creada la lista se puede unir (`join`) de nuevo y escribir ésto en el archivo.

Para el archivo `data5.txt` ubicado en la carpeta `files`



Edición de archivos II

El siguiente código abre el archivo `data5.txt` en modo lectura, lee su lista de líneas, inserta una nueva línea en la lista, lo abre de nuevo `data5.txt` en modo escritura y escribe en éste la totalidad del contenido de la lista.

```
# Abre el archivo en modo de sólo lectura
with open("files/data5.txt", "r", encoding="utf-8") as f:
    list_content = f.readlines()

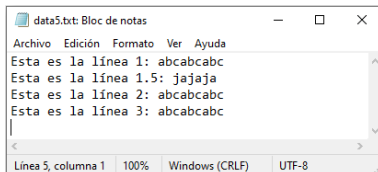
list_content.insert(1, "Esta es la línea 1.5: jajaja\n")

# Re-abre el archivo en modo de sólo escritura
# para sobrescribir la versión anterior de éste
with open("files/data5.txt", "w", encoding="utf-8") as f:
    contenido = "".join(list_content)
    f.write(contenido)
```



Edición de archivos III

El resultado obtenido es el archivo data5.txt sobrescrito



Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios**
- 9 El módulo `pickle`
- 10 Copiar un archivo binario



Listar archivos (os) I

En las versiones modernas de Python una alternativa a `os.listdir()` es utilizar `os.scandir()` (desde Python 3.5) y `pathlib.Path()`. El módulo `os` trae funciones de utilidad para remover archivos (`remove`), crear directorios (`mkdir(path)`), etc.

Para el directorio `files`, el siguiente código permite caracterizar el contenido de archivos y subdirectorios contenidos en éste.

```
import os
entries = os.scandir("files/")

for entry in entries:
    print(entry.name + ", es directorio: "
          + str(entry.is_dir()) + ", size: " +
          str(entry.stat().st_size) + " bytes.")
```



Listar archivos (os) II

La salida de este programa es

```
data.txt, es directorio: False, size: 17 bytes.  
data1.txt, es directorio: False, size: 95 bytes.  
data2.txt, es directorio: False, size: 131 bytes.  
data3.txt, es directorio: False, size: 32 bytes.  
data4.txt, es directorio: False, size: 31 bytes.  
data5.txt, es directorio: False, size: 128 bytes.  
sample_data, es directorio: True, size: 4096 bytes.
```



Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo pickle
- 10 Copiar un archivo binario



Guardar estructuras de datos en archivos: pickle I

A veces se desea almacenar información contenida en objetos Python, (diccionarios ó listas por ejemplo). En este caso se puede utilizar el módulo pickle con su método dump, para serializar objetos.

El siguiente código permite crear dos lista, un diccionario a partir de éstas y por último se serializan como objetos

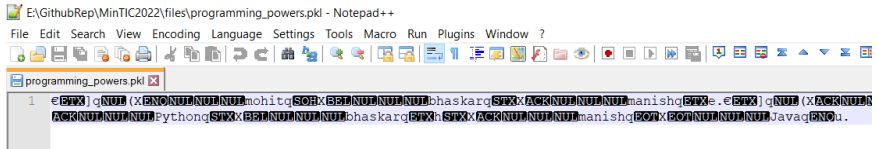
```
import pickle

name = ["mohit", "bhaskar", "manish"]
skill = ["Python", "Python", "Java"]
dict1 = dict([(k,v) for k,v in zip(name, skill)])
with open("files/programming_powers.pkl", "wb") as p_file:
    pickle.dump(name, p_file)
    pickle.dump(skill, p_file)
    pickle.dump(dict1, p_file)
```



Guardar estructuras de datos en archivos: pickle II

El resultado obtenido es el archivo `programming_powers.pkl`, como se puede observar a continuación



Cargar estructuras de datos desde archivos: pickle I

Para realizar el proceso inverso de serialización (leer de archivo los objetos serializados con pickle), se puede seguir el método `load`:

```
import pickle

with open("files/programming_powers.pkl", "rb") as p_file:
    list1 = pickle.load(p_file)
    list2 = pickle.load(p_file)
    dict1 = pickle.load(p_file)

print(list1)
print(list2)
print(dict1)
```



Cargar estructuras de datos desde archivos: pickle II

La salida de este programa es

```
['mohit', 'bhaskar', 'manish']  
['Python', 'Python', 'Java']  
{'mohit': 'Python', 'bhaskar': 'Python', 'manish': 'Java'}
```



Agenda

- 1 Introducción
- 2 Configuración de esta sesión
- 3 Modos de apertura de archivos
- 4 Lectura de archivos
- 5 Escritura de archivos
- 6 Otras operaciones de interés
- 7 Editando un archivo de texto
- 8 Listar archivos y directorios
- 9 El módulo `pickle`
- 10 Copiar un archivo binario



Copiar un archivo binario

Una imagen de tipo *.jpg es un archivo de tipo binario. Con cierto procesamiento es posible crear una copia de la imagen de la siguiente manera

```
with open("files/discurso.jpg", "rb") as imagen:  
    data = imagen.read()  
  
with open("files/copy.jpg", "wb") as f:  
    f.write(data)
```



Sugerencia

Se sugiere consultar un manual de Python o de sus librerías para determinar si ya existe un método para lo que se quiera realizar con respecto a archivos.



Ejercicios I

Problemas

- 1 Dado el archivo de texto files/SalesJan2009.csv, procese el archivo para obtener las compras realizadas en un país dado. Ejemplo:

Input	Output
United Kingdom	100



Ejercicios II

Problemas

- ② Dado el archivo de texto files/SalesJan2009.csv, procese el archivo para obtener las compras realizadas con un medio de pago dado. Ejemplo:

Input	Output
Visa	521



Referencias

- Das, B. N. (2017). Learn Python in 7 Days. Packt Publishing Ltd.
- <https://www.dummies.com/programming/python/how-to-find-path-information-in-python/>
- Rodríguez, A (2020). Curso de Programación en Python.
<https://github.com/arleserp/cursopython>
- <https://support.spatialkey.com/spatialkey-sample-csv-data/>
- <https://www.geeksforgeeks.org/with-statement-in-python/>
- <https://stackoverflow.com/questions/40918503/how-to-read-a-utf-8-encoded-text-file-using-python/40918636>
- <https://dbader.org/blog/python-file-io>

