

1. Archivos.

Como sabemos los archivos permiten guardar información de forma persistente en nuestras computadoras. Los datos que forman un archivo tienen un formato concreto. Existen archivos de texto, de imágenes, de sonido y también los hay que guardan un programa ejecutable.

En esta unidad trataremos con archivos de texto plano, archivos CSV y archivos JSON. Leeremos datos de los mismos y crearemos archivos en estos formatos.

2. Rutas.

En el tema que nos ocupa, una ruta es el camino hasta llegar a un archivo o directorio. Estas rutas pueden ser de dos tipos: **absolutas y relativas**.

- Ruta absoluta. Una ruta absoluta indica el camino para llegar a un archivo o directorio desde la raíz del sistema de archivos. Ejemplos:
 - c:\usuarios\pepe\archivo.txt.
 - /home/pepe/archivo.txt.
- Ruta relativa. Una ruta relativa indica el camino para llegar a un archivo o directorio tomando como punto de partida el directorio actual de trabajo. Es relativa con respecto al directorio actual de trabajo. **En el caso de Python será el directorio desde el que ejecutamos la aplicación.**

La función **open()** lleva como primer argumento una ruta (path). Al intentar abrir un archivo mediante **open("archivo.txt", "r", encoding="utf")**, Python realiza una búsqueda relativa en el directorio actual (desde el que ejecutamos la aplicación). Si el archivo existe en esta carpeta creará un objeto que apunte a él, en caso contrario lanzará una excepción del tipo **FileNotFoundError**. Lo mismo sucede si se accede al archivo mediante **open("archivos/archivo.txt", "r", encoding="utf")**, Python buscará la carpeta **archivos** dentro del directorio actual.

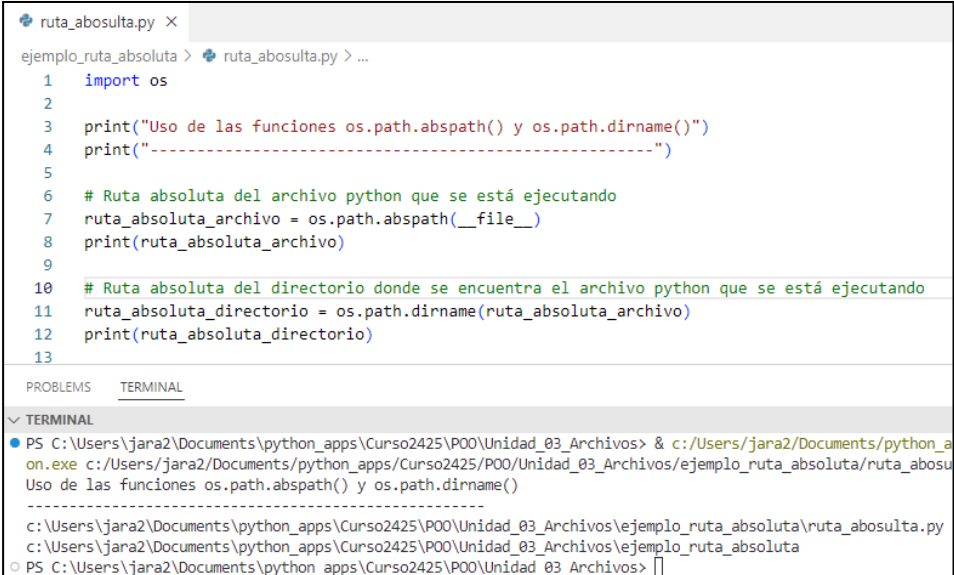
En Python se puede trabajar tanto con rutas absolutas como relativas, siendo siempre conscientes de la operación que se está llevando a cabo.

En las aplicaciones que desarrollemos, utilizaremos el carácter / (forward slash) para indicar directorios, independientemente del sistema operativo utilizado.

Python dispone de varios módulos o librerías que pueden ayudarnos a conocer el directorio actual. Por ejemplo, **el módulo os (import os)** proporciona un conjunto de funciones para interactuar con el sistema operativo en la gestión de archivos y directorios. Entre estas funciones podemos encontrar las siguientes:

- **os.path.abspath(path)**. Retorna la ruta absoluta del path indicado.
- **os.path.dirname(path)**. Retorna el directorio asociado a la ruta indicada.
- **os.path.join(path, *paths)**. Retorna una ruta absoluta a partir de varias.
- **os.getcwd()**. Retorna la ruta o directorio actual de trabajo. Desde el que ejecuto mi aplicación.

Para conocer la ruta absoluta del archivo Python que se está ejecutando podemos hacer uso de las funciones `os.path.abspath()` y `os.path.dirname()` tal y como se muestra en la imagen siguiente.



```
ruta_abosulta.py x
ejemplo_ruta_absoluta > ruta_abosulta.py > ...
1  import os
2
3  print("Uso de las funciones os.path.abspath() y os.path.dirname()")
4  print("-----")
5
6  # Ruta absoluta del archivo python que se está ejecutando
7  ruta_absoluta_archivo = os.path.abspath(__file__)
8  print(ruta_absoluta_archivo)
9
10 # Ruta absoluta del directorio donde se encuentra el archivo python que se está ejecutando
11 ruta_absoluta_directorio = os.path.dirname(ruta_absoluta_archivo)
12 print(ruta_absoluta_directorio)
13

PROBLEMS  TERMINAL
✓ TERMINAL
PS C:\Users\jara2\Documents\python_apps\Curso2425\P00\Unidad_03_Archivos> & c:/Users/jara2/Documents/python_apps/curso2425/p00/unidad_03_archivos/ejemplo_ruta_absoluta/ruta_abosulta.py
Uso de las funciones os.path.abspath() y os.path.dirname()
-----
c:\Users\jara2\Documents\python_apps\Curso2425\P00\Unidad_03_Archivos\ejemplo_ruta_absoluta\ruta_abosulta.py
c:\Users\jara2\Documents\python_apps\Curso2425\P00\Unidad_03_Archivos\ejemplo_ruta_absoluta
PS C:\Users\jara2\Documents\python_apps\Curso2425\P00\Unidad_03_Archivos> 
```

- Línea 7. La variable `__file__` guarda la ruta absoluta al archivo (o módulo) en el que se hace referencia a dicha variable. Mediante `os.path.abspath(__file__)` se obtiene la ruta actual del archivo Python.
- Línea 11. Mediante `os.path.dirname(ruta)` se obtiene la ruta en la que se encuentra el archivo, sin incluir el nombre de este.

3. Operaciones con archivos de texto plano.

Los pasos que deben llevarse a cabo para trabajar con archivos sea cual sea su tipo son los siguientes:

- Abrir el archivo.
- Leer o escribir en el mismo.
- Cerrar el archivo.

La forma más sencilla y ágil de abrir archivos en Python se basa en la utilización de la **sentencia with**. Esta sentencia **cierra de forma automática** el archivo una vez que finaliza el bloque with, por tanto libera al programador de dicha tarea. En otro caso, la forma de cerrar un archivo es usando el método ***archivo.close()***.

Formato de la sentencia:

```
with open('archivo', 'modo apertura', encoding) as f:  
    # operaciones
```

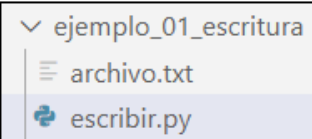
- Los modos de apertura más comunes son los siguientes:

Opción de apertura	Utilidad
r (read)	Abrir en modo lectura. Opción por defecto.
w (write)	Abrir en modo escritura. Si el archivo existe sobrescribe en él.
a (append)	Abrir para añadir contenido al final del mismo

- La codificación es importante para interpretar correctamente los caracteres del mismo. Generalmente la codificación que usaremos será “utf-8”.
- Dentro del bloque “with” se detallan las operaciones que se realizarán con el archivo.
- Al abrir un archivo obtenemos un objeto de tipo “_io.TextIOWrapper” que proporciona acceso al archivo que tiene asociado.
- Los métodos que usaremos al trabajar con archivos son los siguientes. Son métodos asociados al objeto “_io.TextIOWrapper”.
 - write(). Para escribir en el archivo.
 - read(). Lee el contenido del archivo y lo retorna como str.
 - readlines(). Lee todas las líneas del archivo y las retorna como una lista (list).

Ejemplo 1. Cómo escribir en un archivo de texto.

```
escribir.py x
ejemplo_01_escritura > escribir.py > ...
1  import os
2
3  # Generar un archivo de texto.
4
5  # Utilizamos una ruta absoluta para el archivo de texto.
6  directorio = os.path.dirname(os.path.abspath(__file__))
7  ruta_archivo_texto = os.path.join(directorio, "archivo.txt")
8
9  # Abrir el archivo en modo escritura
10 with open(ruta_archivo_texto, "w", encoding="utf-8") as archivo:
11     # Escribir en el archivo
12     archivo.write("Todo hombre sabio teme tres cosas:\n")
13     archivo.write("la tormenta en el mar, \n")
14     archivo.write("la noche sin luna y \n")
15     archivo.write("la ira de un hombre amable.\n")
```



- Líneas 6 y 7. En este ejemplo, se utiliza una ruta absoluta para ubicar el archivo que se crea en la carpeta en la que está la propia aplicación python.
- Línea 10. Observa el modo de apertura del archivo, "w". De esta forma, si el archivo no existe se crea, y si existe se sobrescribe.
- Línea 12 y siguientes. El método **write()** permite escribir en un archivo de texto.
- En la imagen de la derecha se observa que el archivo "archivo.txt" se ha creado en la carpeta de la aplicación Python.

Ejemplo 2. Cómo leer un archivo de texto.

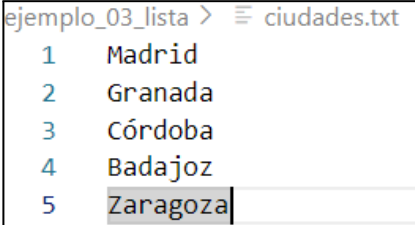
```
lectura.py ●
ejemplo_02_lectura > lectura.py > ...
1  import os
2  # Leer de un archivo de texto.
3
4  # Utilizamos una ruta absoluta para el archivo de texto.
5  directorio = os.path.dirname(os.path.abspath(__file__))
6  ruta_archivo_texto = os.path.join(directorio, "archivo.txt")
7
8  # Abrir el archivo en modo lectura
9  with open(ruta_archivo_texto, "r", encoding="utf") as archivo:
10     # Leer el contenido del archivo
11     contenido = archivo.read()
12     print(contenido)
```

- Líneas 5 y 6. Para este ejemplo, partimos del archivo generado en el ejemplo anterior y la aplicación leerá el contenido del mismo. Se utiliza una ruta absoluta para acceder a dicho archivo.
- Línea 9. El modo de apertura del archivo es "r" de lectura.

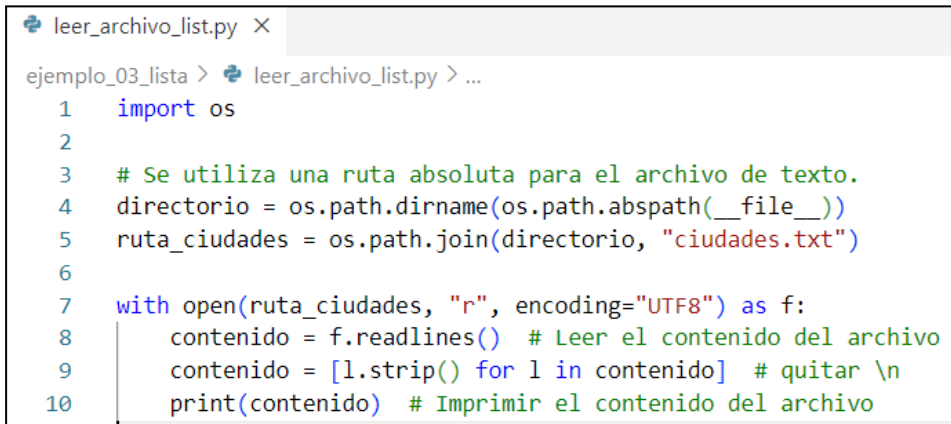
- Línea 10. El método **read()** lee todo el contenido del archivo en un objeto str.

Ejemplo 3. Leer el contenido de un archivo de texto y guardar las líneas en una lista (list).

- En este caso, leeremos datos del archivo “ciudades.txt” y dejaremos el resultado en una lista.
- La imagen de la derecha muestra el contenido de dicho archivo.

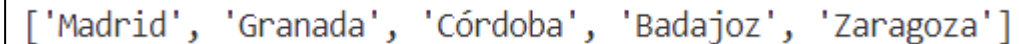


```
ejemplo_03_lista > ciudades.txt
1 Madrid
2 Granada
3 Córdoba
4 Badajoz
5 Zaragoza
```



```
leer_archivo_list.py X
ejemplo_03_lista > leer_archivo_list.py > ...
1 import os
2
3 # Se utiliza una ruta absoluta para el archivo de texto.
4 directorio = os.path.dirname(os.path.abspath(__file__))
5 ruta_ciudades = os.path.join(directorio, "ciudades.txt")
6
7 with open(ruta_ciudades, "r", encoding="UTF8") as f:
8     contenido = f.readlines() # Leer el contenido del archivo
9     contenido = [l.strip() for l in contenido] # quitar \n
10    print(contenido) # Imprimir el contenido del archivo
```

- Líneas 4 y 5. Se busca el archivo en la carpeta que contiene la aplicación Python.
- Línea 7. El archivo se abre en modo lectura, “r”.
- Línea 8. El método `readlines()` sin argumentos lee todos los bytes del archivo en un objeto “list”. Ojo, cada elemento de la lista contiene al final el salto de línea (\n).
- Línea 9. El método “strip()” elimina los saltos de línea de cada cadena. En este caso se ha usado “list comprehensions” para crear la lista sin saltos de línea.
- La salida de la aplicación es la que se muestra a continuación.



```
['Madrid', 'Granada', 'Córdoba', 'Badajoz', 'Zaragoza']
```

Ejemplo 4. Añadir líneas a un archivo de texto existente.

En el ejemplo que sigue se añaden líneas al archivo “ciudades.txt”.

```
anadir_lineas.py X
ejemplo_04_append > anadir_lineas.py > ...
1  # Añadir líneas a un archivo
2  import os
3
4  # Se utiliza una ruta absoluta para el archivo de texto.
5  directorio = os.path.dirname(os.path.abspath(__file__))
6  ruta_ciudades = os.path.join(directorio, "ciudades.txt")
7
8  with open(ruta_ciudades, "a", encoding="UTF8") as f:
9      f.write("Vigo\n") # Añadir una línea al archivo
10     f.write("La Coruña\n") # Añadir otra línea al archivo
```

- Línea 8. La apertura del archivo se realiza con la opción “a” de “append”.
- Líneas 9 y 10. Por cada línea que se añade se inserta el carácter salto de línea, para evitar que las ciudades se escriban en la misma línea dentro del archivo.
- La imagen de la derecha muestra cómo quedaría el archivo tras añadir las líneas.

1	Madrid
2	Granada
3	Córdoba
4	Badajoz
5	Zaragoza
6	Vigo
7	La Coruña

Ejemplo 5. Leer línea a línea un archivo de texto.

Puede darse el caso de querer llevar a cabo una lectura línea a línea de un archivo y aplicar algún tratamiento a cada una de estas líneas de forma individual. Para recorrer un archivo por líneas basta con utilizar un bucle “for” con el objeto que apunta al archivo que proporciona “with”.

```
with open("archivo.txt", "r", encoding="utf-8") as archivo:
    for linea in archivo:
        # procesamiento de la línea
```

En el ejemplo que se estudia a continuación se leerá un archivo de texto línea a línea, y por cada una de ellas se contará el número de vocales que contiene.

```
linea_linea.py X
ejemplo_05_linea_linea > linea_linea.py > ...
1  import os
2
3  # Leer línea a línea un archivo de texto
4
5  # Ruta absoluta del archivo
6  directorio = os.path.dirname(os.path.abspath(__file__))
7  ruta_archivo = os.path.join(directorio, "archivo.txt")
8
9  # Abrir el archivo en modo lectura
10 with open(ruta_archivo, "r", encoding="utf-8") as archivo:
11     # Leer línea a línea
12     for linea in archivo:
13         numero_vocales = len([x for x in linea.lower()
14                               | if x in "aeiouáéíóú"])
15         print(f'\{linea.strip()}\ ' tiene {numero_vocales} vocales")
```

- Línea 10. Apertura del archivo.
- Línea 12. Bucle "for" para recorrer el objeto **archivo** línea a línea.
- Línea 13. Contar las vocales de cada línea. Se crea una lista con las vocales y se obtiene la longitud de esa lista.
- Línea 15. Mostrar los resultados.
- La salida de la aplicación será la que se observa en la imagen adjunta.

```
'Todo hombre sabio teme tres cosas:' tiene 12 vocales
'la tormenta en el mar,' tiene 7 vocales
'la noche sin luna y' tiene 6 vocales
'la ira de un hombre amable.' tiene 10 vocales
```

4. Operaciones con archivos CSV.

Un archivo con extensión **CSV** es un archivo de texto en el que cada línea es un registro formado por uno o más campos separados por coma, de ahí su nombre, CSV o comma separated values. Son muy útiles para exportar datos entre aplicaciones como bases de datos, hojas de cálculo, etc. y además pueden ser tratados por las herramientas de análisis de datos de Python como Pandas.

La librería **CSV** de Python proporciona utilidades para trabajar con este tipo de archivos. Por ello al principio de cada proyecto será necesario realizar la importación mediante **import CSV**.

La forma de trabajar será la misma que se vio en el punto anterior, es decir, abrir el archivo, procesar la información y cerrar el archivo.

Para crear/escribir en un archivo **CSV** se necesitará un objeto **_CSV.writer**, este objeto dispone del método **writerow()** que escribe información en el archivo.

Para leer será necesario un objeto **_CSV.reader** que contiene el método **reader()** que será el encargado de la lectura.

Ejemplo 1. Generar un archivo CSV.

En este caso partimos de una lista con nombre, apellido y edad de una serie de personas que serán guardadas en un archivo CSV.

```
crear_csv.py X
ejemplo_07_crear_csv_list > crear_csv.py > [?] directorio
1  # Crear un archivo CSV
2  import csv
3  import os
4
5  # Datos a escribir
6  datos = [
7      ['Nombre', 'Apellido', 'Edad'],
8      ['Juan', 'Pérez', 25],
9      ['Ana', 'Gómez', 30],
10     ['Pedro', 'García', 45],
11     ['María', 'López', 35]
12 ]
13
14 # Ruta absoluta del archivo
15 directorio = os.path.dirname(os.path.abspath(__file__))
16 ruta_archivo = os.path.join(directorio, 'datos.csv')
17
18 # Apertura del archivo en modo escritura
19 with open(ruta_archivo, mode='w', encoding="utf8") as archivo:
20     # Crear un objeto writer.
21     csv_writer = csv.writer(archivo, delimiter=',', lineterminator='\n')
22     for elemento in datos:
23         csv_writer.writerow(elemento)
```

- Línea 19. Apertura del archivo para escritura.
- Línea 20. Obtener el objeto writer a partir del archivo. Indicamos el separador de columnas (delimiter) y el de líneas (lineterminator).
- Línea 22 y siguientes. Mediante un bucle for se recorre la lista y cada elemento se escribe en el archivo mediante el método "writerow(elemento)". Ten en cuenta que cada objeto **elemento** es una lista (o sublista).

Ejemplo 2. Leer un archivo CSV.

Ahora se llevará a cabo el proceso contrario, es decir, se parte de un archivo CSV y su información se traslada a una lista (list).

```
leer_csv.py X
ejemplo_06_leer_csv > leer_csv.py > ...
1  import csv
2  import os
3
4  # Ruta absoluta del archivo
5  directorio = os.path.dirname(os.path.abspath(__file__))
6  ruta_archivo = os.path.join(directorio, 'Summer_olympic_Medals.csv')
7
8  with open(ruta_archivo, 'r', encoding="utf8") as archivo:
9      # Leer el archivo
10     csv_reader = csv.reader(archivo)
11
12     # headers = next(csv_reader) # lee la primera fila del archivo. Cabeceras.
13
14     l = list(csv_reader) # transforma el objeto csv_reader en una lista
15     for fila in l: # recorre la lista
16         print(fila)
```

PROBLEMS TERMINAL

✓ **TERMINAL**

```
['2020', 'Japan', 'Tokyo', 'Armenia', 'ARM', '0', '2', '2']
['2020', 'Japan', 'Tokyo', 'Morocco', 'MAR', '1', '0', '0']
['2020', 'Japan', 'Tokyo', 'Fiji', 'FIJ', '1', '0', '1']
```

- Línea 1. Importación del módulo CSV.
- Línea 8. Apertura del archivo.
- Línea 10. Leer el archivo mediante el objeto **reader**.
- Línea 14. Convertir el objeto leído en una lista.
- Línea 15 y 16. Recorrer la lista.

Ejemplo 3. Tratar las líneas de un CSV como diccionarios.

La clase **csv.DictReader** permite manipular cada línea de un archivo CSV como un diccionario. Este diccionario tiene como claves las cabeceras del archivo CSV y como valor los datos propiamente dichos.

```
1  import csv, os
2
3  try:
4      directorio = os.path.dirname(os.path.abspath(__file__))
5      ruta_archivo = os.path.join(directorio, 'departaments.csv')
6
7      with open(ruta_archivo, "r") as archivo:
8          lector = csv.DictReader(archivo)
9
10         cabeceras = next(lector) # Cabeceras.
11
12         for linea in lector:
13             for k,v in linea.items():
14                 print(f"{k}: {v}", end=" ")
15             print()
16
17     except FileNotFoundError:
18         print("El archivo no existe")
19
```

PROBLEMS TERMINAL

TERMINAL

```
department_id: 180 department_name: Construction manager_id: location_id: 1700
department_id: 190 department_name: Contracting manager_id: location_id: 1700
```

5. Tratamiento de archivos JSON.

Aunque estos archivos nacieron unidos a Javascript, JSON viene de “Javascript Object Notation”, se ha convertido en el formato de archivo más utilizado en la distribución e intercambio de datos.

Un archivo JSON es un archivo de texto que se ajusta a una determinada estructura, y esta estructura es la misma que se utiliza en los diccionarios de Python. Es decir, un archivo JSON está formado por parejas clave-valor.

Para trabajar con archivos JSON en Python debemos importar el módulo JSON mediante **import JSON**.

En esta parte de la unidad veremos cómo convertir un diccionario de Python en una cadena con formato JSON y también el proceso contrario. Además crearemos y leeremos archivos JSON.

La tabla siguiente muestra la correspondencia entre objetos de Python y su equivalente en JSON.

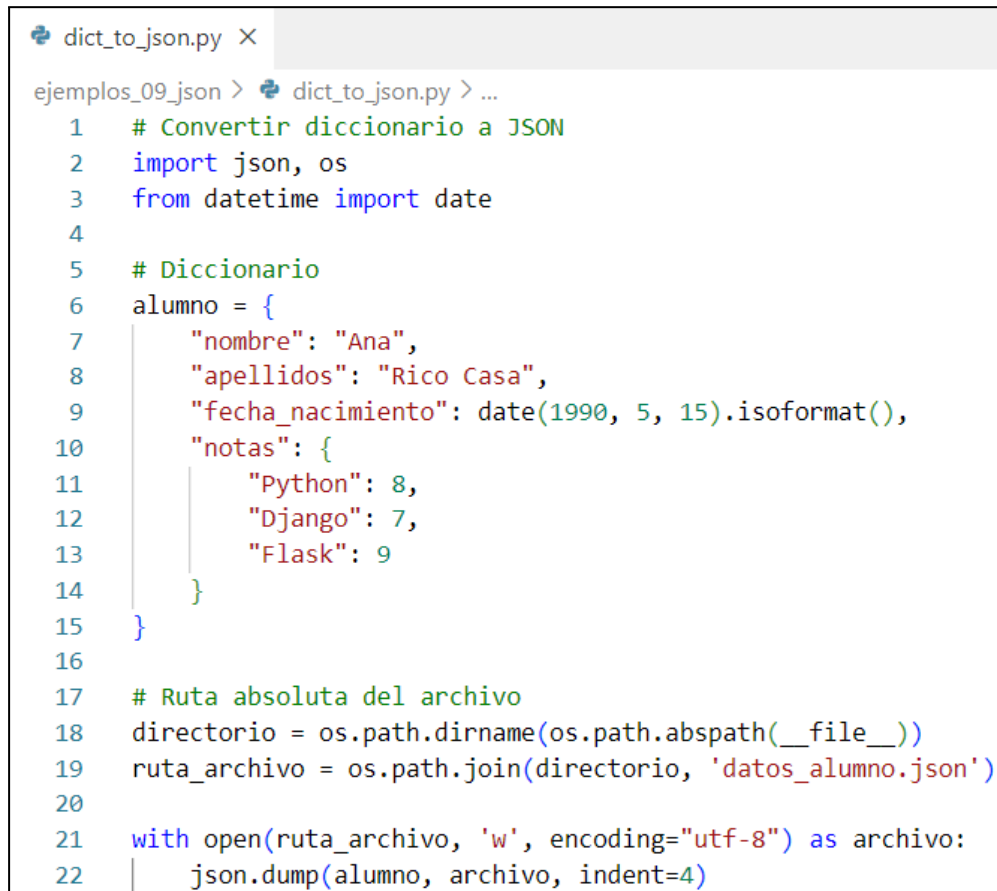
Python	JSON
dict	Object
list, tuple	Array
str	String
int, float	Number
True, False	true, false
None	null

Las funciones del módulo JSON que estudiaremos son las siguientes:

- `JSON.load()`. Retorna un diccionario a partir de una cadena JSON. La cadena puede ser tomada de un archivo mediante **`JSON.load(archivo)`**.
- `JSON.dump()`. Escribe un diccionario Python en un archivo JSON.
- `JSON.dumps()`. Convierte un diccionario Python en cadena JSON.


Ejemplo 1. Convertir un diccionario Python en archivo JSON.

En el ejemplo que sigue, el objeto datos (dict) se escribe en el archivo **datos.JSON**.



```
dict_to_json.py X
ejemplos_09_json > dict_to_json.py > ...
1  # Convertir diccionario a JSON
2  import json, os
3  from datetime import date
4
5  # Diccionario
6  alumno = {
7      "nombre": "Ana",
8      "apellidos": "Rico Casa",
9      "fecha_nacimiento": date(1990, 5, 15).isoformat(),
10     "notas": {
11         "Python": 8,
12         "Django": 7,
13         "Flask": 9
14     }
15 }
16
17 # Ruta absoluta del archivo
18 directorio = os.path.dirname(os.path.abspath(__file__))
19 ruta_archivo = os.path.join(directorio, 'datos_alumno.json')
20
21 with open(ruta_archivo, 'w', encoding="utf-8") as archivo:
22     json.dump(alumno, archivo, indent=4)
```

- Línea 6 y siguientes. Creación del diccionario.
- Línea 21. Apertura del archivo.
- Línea 22. Escribir el objeto diccionario en el archivo mediante **JSON.dump()**. Mediante el parámetro **indent=4**, el archivo se genera con indentación.
- En la siguiente imagen se puede ver el contenido del archivo creado.



```
dict_to_json.py  {} datos_alumno.json X
ejemplos_09_json > {} datos_alumno.json > nombre
1 {
2     "nombre": "Ana",
3     "apellidos": "Rico Casa",
4     "fecha_nacimiento": "1990-05-15",
5     "notas": {
6         "Python": 8,
7         "Django": 7,
8         "Flask": 9
9     }
10 }
```

Ejemplo 2. Cargar un archivo JSON en un diccionario Python.

En este ejemplo leeremos el contenido de un archivo JSON y lo convertiremos en un diccionario de Python.



```
json_to_dict.py X
ejemplos_09_json > json_to_dict.py > ...
1 # De JSON a dictionary
2 import json, os
3
4 # Ruta absoluta del archivo
5 directorio = os.path.dirname(os.path.abspath(__file__))
6 ruta_archivo = os.path.join(directorio, 'datos_alumno.json')
7
8 with open(ruta_archivo, 'r', encoding="utf-8") as archivo:
9     datos_alumno = json.load(archivo)
10
11 print(datos_alumno)
```

PROBLEMS TERMINAL

✓ TERMINAL

```
PS C:\Users\jara2\Documents\python_apps\Curso2425\P00\Unidad_03_Archivos> & c:/Users/jara2/Documents/python_apps/Curso2425/P00/2/Documents/python_apps/Curso2425/P00/Unidad_03_Archivos/ejemplos_09_json/json_to_dict.py
{'nombre': 'Ana', 'apellidos': 'Rico Casa', 'fecha_nacimiento': '1990-05-15', 'notas': {'Python': 8, 'Django': 7, 'Flask': 9}}
```

- Línea 8. Lectura del contenido del archivo y transformación en diccionario mediante **JSON.load()**.
- En la imagen puede observarse el diccionario obtenido.

6. Serializar objetos.

Serializar consiste en transformar objetos en flujos de bytes para transmitirlos o almacenarlos. El proceso contrario es la deserialización, es decir, transformar los bytes de nuevo en objetos.

En este apartado estudiaremos cómo serializar objetos para guardarlos en archivos binarios y cómo recuperar los objetos guardados. Al ser los archivos de tipo binario, el modo de apertura difiere de los vistos antes en esta unidad. **Para escribir utilizaremos el modo “wb” y para leer “rb”**. La “b” añadida viene de binario.

Para llevar a cabo la serialización utilizaremos el módulo `pickle` (**`import pickle`**). Los métodos usados de este módulo son **`dump()`** para guardar objetos y **`load()`** para recuperarlos.

En los ejemplos que siguen trabajaremos con la clase “Universidad” descrita en la imagen. En el primer ejemplo (`serializar.py`) se crean varias instancias de esta clase y se guardan en un archivo binario. En el segundo ejemplo (`deserializar.py`) se realiza la operación inversa.

```
1  # clases del proyecto
2  class Universidad:
3      def __init__(self, nombre, ciudad, pais):
4          self.nombre = nombre
5          self.ciudad = ciudad
6          self.pais = pais
7
8      def __str__(self):
9          return f"Universidad: {self.nombre}, " \
10             + f"Ciudad: {self.ciudad}, Pais: {self.pais}"
11
```

6.1. Escribir objetos en un archivo binario.

Estudiaremos ahora cómo escribir objetos en un archivo binario. Para ello será necesario utilizar `pickle` (`import pickle`) y su método **`pickle.dump(objeto, archivo)`**.

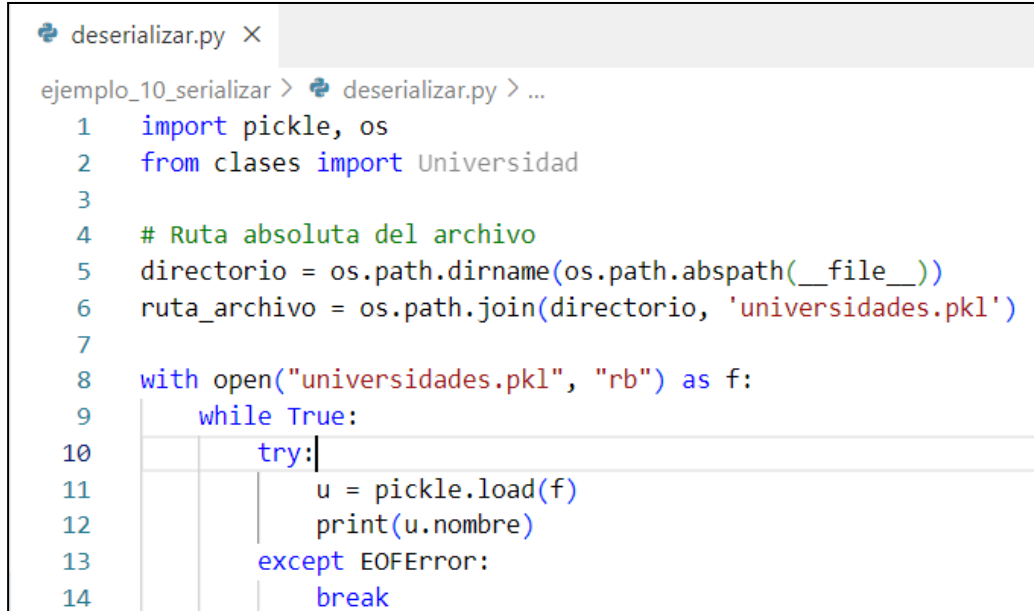
En el ejemplo que se muestra a continuación se escriben tres objetos en el archivo mediante **`pickle.dump(objeto, archivo)`**. Cabe decir que estos podrían estar almacenados en una lista, con lo que en la operación de volcado solo sería necesaria una llamada al método **`dump()`**, quedaría: **`pickle.dump(lista, archivo)`**.

```
serializar.py x
ejemplo_10_serializar > serializar.py > ...
1  import pickle, os
2  from clases import Universidad
3
4  u1 = Universidad("Universidad Complutense", "Madrid", "España")
5  u2 = Universidad("Universidad de los Andes", "Bogota", "Colombia")
6  u3 = Universidad("Universidad de Buenos Aires", "Buenos Aires", "Argentina")
7
8  # Ruta absoluta del archivo
9  directorio = os.path.dirname(os.path.abspath(__file__))
10 ruta_archivo = os.path.join(directorio, 'universidades.pkl')
11
12 with open(ruta_archivo, "wb") as f:
13     pickle.dump(u1, f)
14     pickle.dump(u2, f)
15     pickle.dump(u3, f)
```

- Líneas 1 y 2. Importaciones necesarias.
- Líneas 3, 4, y 5. Creación de objetos.
- Líneas 12 y siguientes. Apertura del archivo (wb) y volcado de los objetos. Para el volcado se utiliza el método **`dump(objeto, archivo)`**.

6.2. Leer objetos desde un archivo binario.

Para leer objetos de un archivo binario, abrimos este con el modo “rb” y se realizan tantas operaciones de lectura ***pickle.load()*** como objetos se guardaran en el archivo.



```
deserializar.py X
ejemplo_10_serializar > deserializar.py > ...
1  import pickle, os
2  from clases import Universidad
3
4  # Ruta absoluta del archivo
5  directorio = os.path.dirname(os.path.abspath(__file__))
6  ruta_archivo = os.path.join(directorio, 'universidades.pkl')
7
8  with open("universidades.pkl", "rb") as f:
9      while True:
10         try:
11             u = pickle.load(f)
12             print(u.nombre)
13         except EOFError:
14             break
```

- Líneas 1 y 2. Importaciones necesarias.
- Línea 8. Apertura del archivo en modo lectura binaria (rb).
- Línea 9 y siguientes. Bucle while sin fin. Se lee del archivo hasta llevar al final del mismo (EOFError). El método usado para leer es ***load(archivo)***. Cuando se genera esta excepción el bucle finaliza con “break”.

7. Excepciones en el tratamiento de archivos.

El intérprete de Python lanza una excepción cuando en la ejecución de un programa se produce algún error. Por ejemplo, abrir un archivo que no existe, escribir en un archivo para el que no se tienen permisos, error al conectar a un servidor de bases de datos, etc.

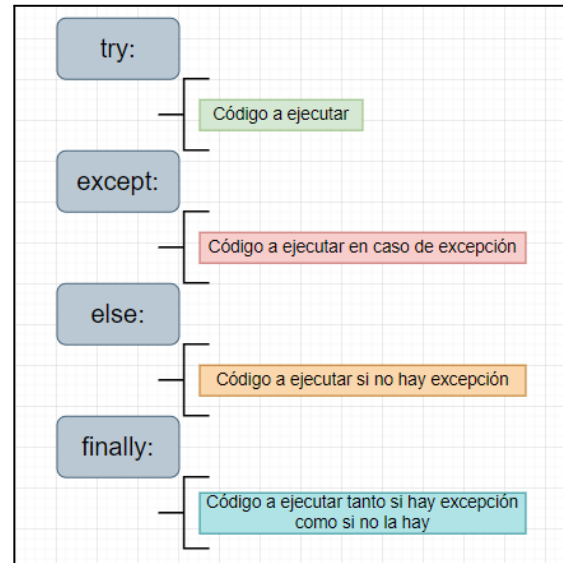
Las excepciones más comunes que pueden darse al trabajar con archivos son:

- ***FileNotFoundError***. Esta excepción se lanza cuando se intenta acceder a un archivo que no existe.
- ***PermissionError***. Cuando un programa intenta llevar a cabo una operación para la que no se tiene permiso.
- ***IOError***. Esta excepción engloba un conjunto más amplio de operaciones, tales como intentar leer de un archivo que está abierto para escritura, acceder a un archivo que no existe o para el que no se tienen permisos, etc.

Como sabemos las excepciones en Python se tratan mediante el bloque `try-except` (ver imagen).

Si al ejecutar el código del bloque ***try*** (en verde) se produce una excepción, se ejecutan las instrucciones del bloque ***except*** (rosa). Si no se lanza la excepción/es controladas en este bloque ***except*** se ejecuta el código asociado al bloque ***else*** (naranja). El código asociado a ***finally*** (azul) se ejecuta siempre.

Los bloques ***else*** y ***finally*** son opcionales.



En el ejemplo que sigue, se muestra el funcionamiento de ***try-except***. Se intenta escribir en un archivo para el que no se tienen permisos de escritura. La aplicación termina con normalidad mostrando el mensaje asociado a la excepción.

```
1 # Control de excepciones
2
3 try:
4     with open('c:/windows/unsu.log', 'w') as archivo:
5         archivo.write('Hola mundo')
6 except PermissionError as e:
7     print(f'Error: {e}')
8
```

PROBLEMS OUTPUT TERMINAL COMMENTS

▼ TERMINAL

PS C:\Users\jara2\Documents\python_apps\Curso2425\P00\Unidad_03_Archivos\exceptions\leer_archivo.py
Error: [Errno 13] Permission denied: 'c:/windows/unsu.log'

En este otro ejemplo, se intenta leer un archivo que no existe en el directorio actual, por tanto se lanza la excepción ***FileNotFoundError***.

```
1  # Control de excepciones
2
3  try:
4      with open('datos.txt', 'r') as archivo:
5          contenido = archivo.read()
6  except FileNotFoundError as e:
7      print(f'El archivo no existe')
```

PROBLEMS OUTPUT TERMINAL COMMENTS

✓ TERMINAL

PS C:\Users\jara2\Documents\python_apps\Curso2425\P00\Unidad_03_Archivos\exceptions\error_notfound.py
El archivo no existe

Por último, cabe señalar que como programadores nos puede interesar lanzar una excepción cuando se de una determinada situación en nuestras funciones o aplicaciones. En estos casos se debe utilizar la sentencia ***raise nombre_excepción***.

8. Módulos y paquetes.

8.1. Módulos.

Un módulo en Python es un archivo con extensión ***.py*** que contiene código Python en forma de funciones, clases, variables, etc. y que puede ser utilizado (importado) en otros módulos. Existen módulos estándar (built-in modules) y módulos creados por el programador. Los módulos estándar son un conjunto de librerías que vienen preinstaladas con el intérprete.

El uso de módulos facilita la organización de los proyectos y además permite la reutilización de código.

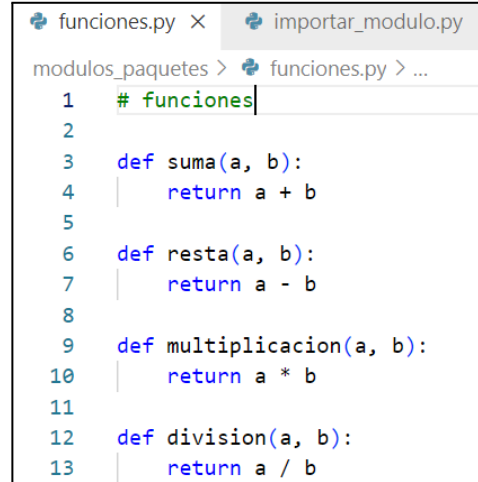
Para utilizar dentro de un módulo, código desarrollado en otro, es necesario **importar el módulo que contiene el código**.

8.2. Formas de importar un módulo.

PEP8 recomienda:

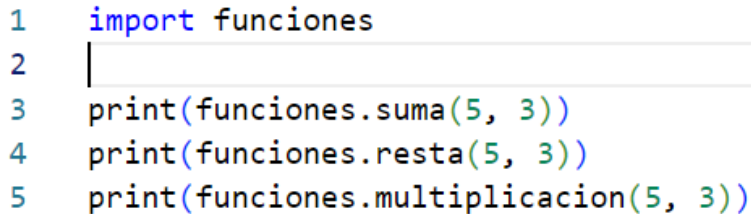
- Realizar las importaciones al principio del archivo, tras los comentarios iniciales.
- Escribir primero los ***import*** de los built-in modules, luego los módulos de terceros y por último los módulos del proyecto.
- Separar cada grupo de ***import*** del resto por una línea en blanco.
- Utilizar ***Absolute Imports***, esto es, hacer referencia al recurso desde la carpeta raíz del proyecto.

Veamos mediante ejemplos las distintas posibilidades que existen para importar módulos. Partimos de un módulo llamado **funciones.py**, que como se observa en la imagen de la derecha contiene la definición de varias funciones.



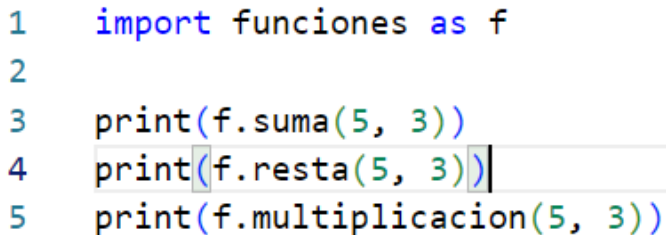
```
funciones.py X importar_modulo.py
modulos_paquetes > funciones.py > ...
1  # funciones
2
3  def suma(a, b):
4      return a + b
5
6  def resta(a, b):
7      return a - b
8
9  def multiplicacion(a, b):
10     return a * b
11
12 def division(a, b):
13     return a / b
```

- **Opción 1. import nombre_modulo.** Para usar las funciones y variables definidas en este módulo debemos utilizar la forma: **nombre_modulo.nombre_función**.



```
1  import funciones
2
3  print(funciones.suma(5, 3))
4  print(funciones.resta(5, 3))
5  print(funciones.multiplicacion(5, 3))
```

- **Opción 2. import nombre_modulo as alias.** En este caso utilizamos un alias para el módulo, acortando su nombre. Si se importan varios módulos con el mismo alias, prevalece el último.



```
1  import funciones as f
2
3  print(f.suma(5, 3))
4  print(f.resta(5, 3))
5  print(f.multiplicacion(5, 3))
```

- **Opción 3. from módulo import función/funciones/*.** Importar una función de un módulo, varias separadas por comas, o bien usar asterisco "*" para importarlas todas. Debe tenerse en cuenta que si un módulo contiene gran cantidad de funciones y se importan todas, incluso las que no se van a usar, estaremos consumiendo memoria de forma innecesaria.

```
1  from funciones import suma, resta
2
3  print(suma(5, 3))
4  print(resta(5, 3))
```

8.3. Ejecutar un módulo como un script.

Un módulo puede ser ejecutado como un script o como punto de entrada de un programa cuando se pasa directamente como parámetro al intérprete de Python. En este caso el módulo debe llevar **al final del mismo** la sentencia `if __name__ == '__main__':`, seguida de alguna llamada a una función, pass, etc. De esta forma podemos probar las funciones del módulo desde el mismo módulo.

```
def suma(a, b):
    return a + b

if __name__ == '__main__':
    print(suma(2, 3))
```

8.4. Paquetes.

Un paquete es un directorio que contiene otros paquetes y/o módulos. El código de estos paquetes suele estar relacionado entre sí.

La distribución del código de una aplicación en paquetes mejora la organización y reutilización. Para que un directorio sea considerado un paquete debe tener un archivo vacío con el nombre `__init__.py`.

Los paquetes pueden contener subpaquetes, o paquetes dentro de paquetes. Esto permite dar un paso más en la organización de archivos del proyecto. Los subpaquetes también deben contener el archivo `__init__.py`.

Para importar un módulo contenido en un paquete debe usarse la forma:

- `from paquete import modulo`
- `from paquete.modulo import funciones/*`
- `from paquete.subpaquete.modulo import funciones/*`

9. Enlaces de interés.

- <https://www.geeksforgeeks.org/file-handling-python/>
- <https://plainenglish.io/blog/exception-and-file-handling-in-python>
- <https://ellibrodepython.com/excepciones-try-except-finally>
- <https://www.squash.io/how-to-save-and-load-objects-using-pickleload-in-python/#:~:text=In%20Python%2C%20the%20pickle..back%20into%20objects%20when%20needed.>
- <https://realpython.com/python-import/>