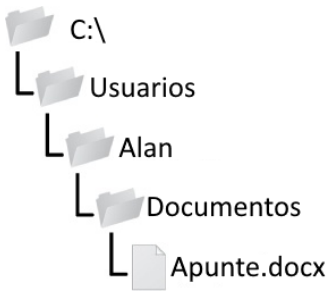
Archivos

Leyendo y escribiendo archivos

Las variables nos sirven mientras estamos corriendo un programa, pero cuando necesitamos guardar datos de manera permanente hay que utilizar archivos. Los contenidos de un archivo son algo parecido a una cadena o string que se extiende hasta llegar a veces hasta varios gigabytes en tamaño.

Archivos y Direcciones o Paths

Un archivo tiene 2 propiedades: un nombre de archivo o filename y una dirección o path. La dirección especifica la ubicación del archivo en la computadora. Por ejemplo, hay un archivo en mi computadora con el nombre Apunte.docx en la dirección C:\Usuarios\Alan\Documentos. La parte del nombre de un archivo luego del punto es la extensión y nos dice que tipo de archivo es. Podemos decir entonces que Apunte.docx en este ejemplo es un archivo de texto de Microsoft Office Word que se encuentra dentro de la carpeta Documentos, dentro de la carpeta Alan, dentro de la carpeta Usuarios.



C:\ es lo que se llama el directorio raíz o root, y que contiene a todas las otras carpetas. En Windows, la carpeta raíz o root folder se llama C:\ y también se llama el disco C: . En OSX y Linux, la carpeta raíz es / .

Discos o dispositivos adicionales, como Pendrives o DVD's también aparecen de manera distinta dependiendo del sistema operativo. En Windows se les asignarán letras como D:\ o E:\. En OSX, aparecen como carpetas nuevas en la carpeta /Volumes. En Linux, aparecen como carpetas nuevas en la carpeta /mnt o "mount".

En Windows se utiliza la barra invertida para separar carpetas y archivos en una dirección, mientras que en OSX y Linux se usa la barra. Esto es importante si quieren que sus programas de Python funcionen en otros sistemas operativos.

Para facilitar esta diferencia existe una función llamada os.path.join(). Si se le pasamos cadenas con los nombres de las carpetas y archivos en la dirección, os.path.join() deolverá una cadena con el separador correspondiente al sistema operativo donde se está corriendo el programa.

```
>>> import os
>>> os.path.join('Usuarios', 'Documentos', 'Spam.docx')
>>> misArchivos = ['cuentas.txt', 'detalles.csv', 'invitación.docx']
>>> for nombreArch in misArchivos:
    print(os.path.join('C:\\Usuarios\\Alan', nombreArch))
```

Directorio de Trabajo Actual o Current Working DIrectory

Todos los programas tienen un Directorio de Trabajo Actual o Current Working DIrectory, o CWD. Todos los archivos, o direcciones que no comienzan en el directorio raíz, se asume que se encuentran en el cwd. Podemos averiguar cuál es el cwd utilizando la función os.getcwd() y cambiarlo utilizando os.chdir().

```
>>> import os
>>> os.getcwd()
>>> os.chdir('C:\\Windows\\System32')
>>> os.getcwd()
>>> os.chdir('C:\\EstaCarpetaNoExiste')
```

Dirección Absoluta o Dirección Relativa

Hay 2 maneras de escribir la dirección de un archivo o carpeta.

- Como dirección absoluta, que siempre comienza con el directorio raíz.
- como dirección relativa, es decir se lee en relación al directorio de trabajo actual.

También existen las carpetas punto (.) y dos-puntos (..). Estas no son carpetas de verdad sino nombres especiales que se pueden utilizar en una dirección. Un solo punto significa "este directorio". Y dos-puntos significa carpeta de nivel superior.

Para directorio de trabajo actual: **C:\cerdo**	
Dirección Relativa	Dirección Absoluta
..\	C:\
.\	C:\cerdo
.\frito	C:\cerdo\frito\
.\frito\spam.txt	C:\cerdo\frito\spam.txt
.\spam.txt	C:\cerdo\spam.txt
..\huevos	C:\huevos
..\huevos\spam.txt	C:\huevos\spam.txt
..spam.txt	C:\spam.txt

Creando carpetas con os.makedirs()

Nuestros programas pueden crear carpetas utilizando la función os.makedirs().

```
>>> import os
>>> os.makedirs('C:\\deliciosos\\waffles\\belgas')
```

La función os.makedirs() creará todas las carpetas necesariás entre el directorio raíz C: y la carpeta belgas.

Manejo de direcciones Absolutas y Relativas.

A continuación, una lista de funciones para el manejo de direcciones:

- **os.path.abspath(***direccion***)** devolverá una cadena con la dirección absoluta de la dirección.
- **os.path.isabs(***dirección***)** devolverá un booleano True o False si el argumento es una dirección absoluta o no.

- **os.path.relpath(***dirección1***,** *dirección2***)** devolverá una cadena con la dirección relativa entre una dirección y otra. Si el segundo argumento no está definido se utilizará el cwd en su lugar.

```
>>> os.path.abspath('.')
>>> os.path.abspath('.\\Scripts')
>>> os.path.isabs('.')
>>> os.path.isabs(os.path.abspath('.'))
>>> os.path.relpath('C:\\Windows', 'C:\\')
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\huevos')
>>> os.getcwd()
```

- **os.path.dirname**(dirección) devuelve una cadena con todo lo que esté antes de la última barra de la dirección.
- **os.path.basename**(dirección) devuelve una cadena con todo lo que esté después de la ultima barra de la dirección, es decir por lo general el nombre del archivo.

```
>>> path = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(path)
>>> os.path.dirname(path)
```

Si necesitan el dirname y el basename al mismo tiempo, se puede utilizar os.path.split()

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(calcFilePath)
>>> (os.path.dirname(calcFilePath), os.path.basename(calcFilePath))
>>> calcFilePath.split(os.path.sep)
```

Averiguando tamaños de archivo y contenidos de carpeta

- os.path.getsize(dirección) devuelve el tamaño en bytes del archivo en esa dirección.
- os.listdir(dirección) devuelve una lista de cadenas con los nombres de cada archivos en esa dirección.

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
>>> os.listdir('C:\\deliciosos\\waffles')
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
    totalSize = totalSize + os.path.getsize(os.path.join('C:\\Windows\\System32', filename))
>>> print(totalSize)
```

Validación de direcciones

Es muy fácil que una función de Python detenga el programa si recibe una dirección errónea. Para evitar esos errores existen las siguientes funciones.

- **os.path.exists**(dirección) devuelve un booleano True o False dependiendo de si la dirección o archivo existen.
- **os.path.isfile**(dirección) devuelve un booleano True o False dependiendo de si la dirección apunta a un archivo o no.
- **os.path.isdir**(dirección) devuelve un booleano True o False dependiendo de si la dirección apunta a una carpeta.

```
>>> os.path.exists('C:\\Windows')
>>> os.path.exists('C:\\una_carpeta_cualquiera')
>>> os.path.isdir('C:\\Windows\\System32')
>>> os.path.isfile('C:\\Windows\\System32')
>>> os.path.isdir('C:\\Windows\\System32\\calc.exe')
>>> os.path.isfile('C:\\Windows\\System32\\calc.exe')
>>> os.path.exists('D:\\')
```

El Proceso de Escribir y Leer Archivos

Hay 3 pasos a seguir para leer o escribir archivos en Python:

1. Llamar a la función open() para conseguir un objeto File.
2. Llamar a los métodos read() o write() en ese objeto.
3. Cerrar el archivo utilizando el método close() en el objeto File.

Abriendo Archivos con la Función open()

Para abrir un archivo con la función open() hay que pasarle una cadena indicando la dirección del archivo a abrir. Esa dirección puede ser absoluta o relativa.

```
>>> holaFile = open('C:\\Usuarios\\Alan\\hola.txt')
```

Este comando abre el archivo en modo de lectura. Cuando abrimos un archivo en este modo, Python permite leer datos del mismo, pero no puede ser escrito o modificado. Este es el modo que se utiliza por defecto o al pasarle como segundo argumento ‘r’.

La función open() devuelve un objeto File. Este objeto representa el archivo en la computadora, y se comporta como cualquier otro tipo de valor en Python. Es decir, podemos pasarlo como argumento a otras funciones o usar métodos con el mismo.

Leyendo los contenidos de un archivo

Para leer un archivo se utiliza el método read() que devuelve en un valor cadena todo el contenido del mismo.

```
>>> holaContenido = holaFile.read()
>>> holaContenido
```

Si no se puede utilizar el método readlines() para conseguís una lista con cadenas del archivo.

```
>>> sonetoFile = open('soneto29.txt')
>>> sonetoFile.readlines()
```

Escribiendo Archivos

Escribir archivos es parecido a utilizar la función print() para imprimir cadenas por pantalla, solo que esta vez las “imprimiremos” en un archivo. Pero no podemos escribir en un archivo abierto en modo lectura, hay que abrirlo en modo escritura o modo agregado (write plaintext mode o append plaintext mode).

El modo escritura sobrescribirá por completo los contenidos de un archivo existente, y se accede a él pasándole el argumento ‘w’ a open(). El modo agregado por otro lado, agrega texto al final del archivo que abrimos, y se accede utilizando el argumento ‘a’.

Si el archivo no existe, ambos modos, escritura y agregado, crearan un uno nuevo con ese nombre. Luego de leer o escribir un archivo hay que llamar al método close() antes de poder abrir el archivo nuevamente.

```
>>> cerdoFile = open('cerdo.txt', 'w')
>>> cerdoFile.write('Hola Mundo!\n')
>>> cerdoFile.close()
>>> cerdoFile = open('cerdo.txt', 'a')
>>> cerdoFile.write('El cerdo no es una verdura.')
>>> cerdoFile.close()
>>> cerdoFile = open('cerdo.txt')
>>> contenido = cerdoFile.read()
>>> cerdoFile.close()
>>> print(contenido)
```

Guardando Variables con el Módulo Shelve

Para guardar variables en un archivo con Python se puede utilizar el módulo Shelve. Este módulo nos permite guardar y/o abrir la configuración de nuestro programa.

```
>>> import shelve
>>> shelfFile = shelve.open('misdatos')
>>> gatos = ['Sofia', 'Pooka', 'Simon']
>>> shelfFile['gatos'] = gatos
>>> shelfFile.close()
```

Luego de ejecutar este código habrá 3 archivos nuevos en el cwd: misdatos.bak, misdatos.dat, y misdatos.dir. Estos son archivos binarios que contienen los datos del shelf.

Nuestros programas pueden abrir y recuperar estos datos desde estos archivos. A diferencia de archivos regulares, los archivos shelf automáticamente se abren en modo lectura y escritura al mismo tiempo.

```
>>> shelfFile = shelve.open('misdatos')
>>> type(shelfFile)
>>> shelfFile['gatos']
>>> shelfFile.close()
```

Estos objetos shelf funcionan parecido a los diccionarios, con claves, valores y pares clave-valor.

```
>>> shelfFile = shelve.open('misdatos')
>>> list(shelfFile.keys())
>>> list(shelfFile.values())
>>> shelfFile.close()
```

Guardando variables con la función pprint.pformat()

La función pprint.pformat() que ya vimos anteriormente también sirve para generar archivos .py con facilidad, ya que la cadena que genera respeta las reglas de sintaxis de Python, y que luego podremos utilizar para crear nuestros propios módulos de Python.

```
>>> import pprint
```

```
>>> gatos = [{'nombre': 'Sofia', 'desc': 'gordito'}, {'name': 'Pooka', 'desc': 'peludo'}]
>>> pprint.pformat(gatos)
>>> fileObj = open('misGatos.py', 'w')
>>> fileObj.write('gatos = ' + pprint.pformat(gatos) + '\n')
>>> fileObj.close()
```

```
>>> import misGatos
>>> misGatos.gatos
>>> misGatos.gatos[0]
>>> misGatos.gatos[0]['nombre']
```

La ventaja de utilizar un archivo .py para nuestras variables es que puede leerse como un archivo de texto a diferencia de los archivos binarios que genera el módulo shelve. Aunque esto solo funciona para datos simples como cadenas, enteros, listas y diccionarios y no para objetos.