abrir y leer archivos

> La manipulación de archivos desde Python se engloba bajo las funciones de E/S (entrada/salida) o I/O (en inglés: input/output). Comenzaremos explorando las funciones utilizadas para abrir, leer y cerrar archivos:

open(*archivo, modo*): abre un archivo, y devuelve un objeto de tipo archivo sobre el que pueden aplicarse métodos

read(*bytes*): devuelve un número especificado de bytes del archivo. De manera predeterminada (sin indicar un valor en el argumento *bytes*), devolverá el archivo completo (equivalente a escribir **-1**).

readline(*bytes*): devuelve una línea del archivo, limitada por el número indicado en el parámetro tamaño (en bytes).

readlines(*bytes*): devuelve una lista que contiene cada una de las líneas del archivo como item de dicha lista. Si el tamaño excede lo indicado en el parámetro bytes, no se devolverán líneas adicionales.

close( ): cierra el archivo abierto, tal que no puede ser leído o escrito luego de cerrado. Es una buena práctica utilizar este método si ya no será necesario realizar acciones sobre un archivo.

Por lo general, un byte equivale a un caracter.

Python TOTAL

crear y escribir archivos

Python TOTAL

> Para escribir en un archivo desde Python, deberemos elegir con cuidado el parámetro "modo de apertura".

```
open(arhivo, modo)
```

Parámetros de modo de apertura:

- "r" - **Read (Lectura)** - Predeterminado. Permite leer pero no escribir, y arroja un error si el archivo no existe.
- "a" - **Append (Añadir)** - Abre el archivo para añadir líneas a continuación de la última que ya exista en el mismo. Crea un archivo en caso de que el mismo no exista.
- "w" - **Write (Escritura)** - Abre o crea un archivo (si no existe previamente) en modo de escritura, lo que significa que cualquier contenido previo se sobreescribirá.
- "x" - **Create (Creación)** - Crea un archivo, y arroja un error si el mismo ya existe en el directorio.

El método **write( )** escribe un texto especificado en el argumento sobre el archivo.
writelines(*lista*) recibe el texto a ser escrito en forma de lista.

Python TOTAL

directorios

> Trabajar sobre archivos que se encuentran en directorios diferentes al de nuestro código requiere del soporte del módulo OS, que contiene una serie de funciones para interactuar con el sistema operativo.

```
import os
```

- os.getcwd(): obtiene y devuelve el directorio de trabajo actual. Será el mismo en el que corre el programa si no se ha modificado.
- os.chdir(*ruta*): cambia el directorio de trabajo a la ruta especificada
- os.makedirs(*ruta*): crea una carpeta, así como todas las carpetas intermedias necesarias de acuerdo a la ruta especificada.
- os.path.basename(*ruta*): dada una ruta, obtiene el nombre del archivo (nombre de base)
- os.path.dirname(*ruta*): dada una ruta, obtiene el directorio (carpeta) que almacena el archivo
- os.path.split(*ruta*): devuelve una tupla que contiene dos elementos: el directorio, y el nombre de base del archivo.
- rmdir(*ruta*): elimina el directorio indicado en la ruta.

En Windows, es necesario indicar las rutas con dobles barras invertidas (\\) para que sean correctamente interpretadas por Python.

Path

Python TOTAL

La clase Path permite representar rutas de archivos en el sistema de archivos de nuestro sistema operativo. Se destaca por su legibilidad frente a alternativas semejantes.

```
base = Path.home()
```

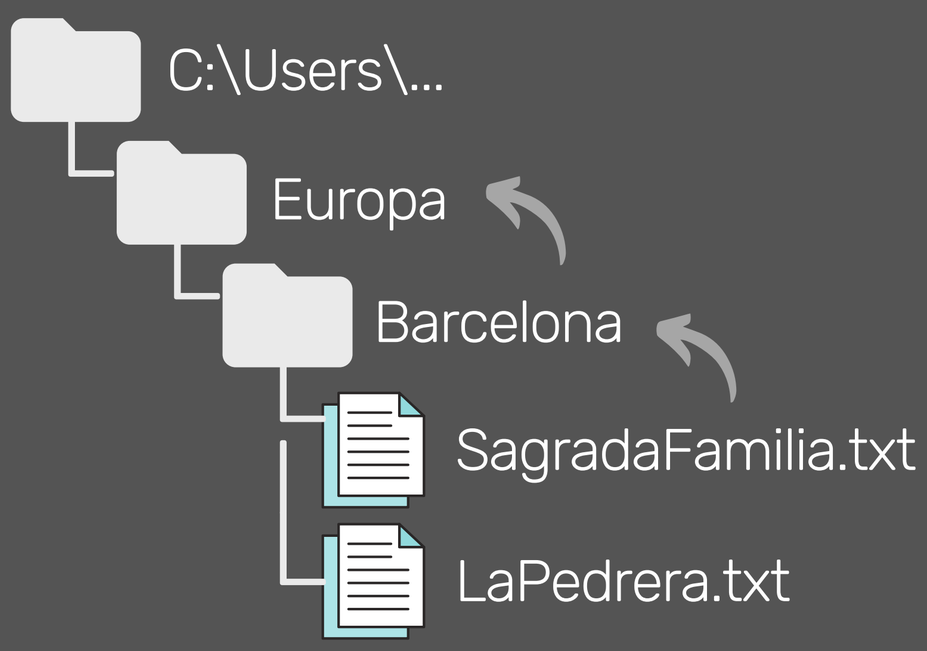
Devuelve un objeto Path representando el directorio base del usuario

```
ruta = Path(base, "Europa", "Barcelona", "SagradaFamilia.txt")
```

Se aceptan strings y otros objetos Path

```
ruta2 = ruta.with_name("LaPedrera.txt")
```

Devuelve un nuevo objeto Path cambiando únicamente el nombre de archivo



Cada invocación de la propiedad parent devuelve la ruta de jerarquía inmediata superior

```
continente = ruta.parent.parent
print(continente)
>> C:\Users\...\Europa
```

Devuelve el conjunto de archivos que coinciden con el "patrón"

```
Path(ruta).glob("*.txt")
Path(ruta).glob("**/*.txt")
```

Búsqueda recursiva en subdirectorios

pathlib

> El módulo **pathlib**, disponible desde Python 3.4, permite crear objetos Path, generando rutas que pueden ser interpretadas por diferentes sistemas operativos y cuentan con una serie de propiedades útiles.

```
from pathlib import Path
```

```
ruta = Path("C:/Users/Usuario/Desktop")
```

A partir de una semántica sencilla, devuelve una ruta que el sistema puede comprender.
Por ejemplo, en Windows, devolverá: **C:\Users\Usuario\Desktop**
y en Mac: **C:/Users/Usuario/Desktop**

Navegación

```
ruta = Path("C:/Users/Usuario/Desktop") / "archivo.txt"
```

Es posible concatenar objetos Path y strings con el delimitador "/" para construir rutas completas.

Algunos métodos y propiedades sobre objetos Path

read_text( ): lee el contenido del archivo sin necesidad de abrirlo y cerrarlo
name: devuelve el nombre y extensión del archivo
suffix: devuelve la extensión del archivo (sufijo)
stem: devuelve el nombre del archivo sin su extensión (sufijo)
exists( ): verifica si el directorio o archivo al que referencia el objeto Path existe y devuelve un booleano de acuerdo al resultado (True/False)

limpiar la consola

Para controlar la información mostrada al usuario en consola podemos limpiarla, eliminando los diferentes mensajes que han aparecido conforme se va ejecutando el programa.

```
from os import system
```

En Unix/Linux/MacOS:

```
system("clear")
```

En DOS/Windows:

```
system("cls")
```

Python TOTAL

archivos + funciones

Python TOTAL

> *Recordatorio*: puedes crear funciones para que ejecuten código cada vez que sean invocadas, evitando repeticiones y facilitando su lectura. Esto aplica para todo Python, y desde luego también cuando manipulamos archivos.