Este es un repaso breve sobre librerías, clases y métodos para controlar el sistema de archivos. Las librerías tienen muchas más funciones que las mostradas en este documento.

Algunas acciones pueden ser dependientes del sistema operativo (por ejemplo, si manejamos permisos en Linux), pero las funciones aquí comentadas pretenden ser multiplataformas.

La documentación es sobre Python 3.13.8.

platform: Información del sistema (SO, CPU, versión de Python, etc.). Es multiplataforma y, salvo algunas funciones específicas, funciona igual en Windows, Linux y macOS.

Funciones: Retornan un str con la información, a menos que se indique lo contrario.

- **Información del SO**
  - system(). Tipo de Sistema operativo. 'Windows' 'Linux'
  - release(). Versión del sistema operativo/kernel.
  - version(). Versión del sistema operativo/kernel.
  - node(). Hostname.
  - platform.freedesktop_os_release(). Información más completa en Linux. Retorna un diccionario (los veremos en tema 2).
  - platform.mac_ver(). Información más completa en Mac.
  - platform.win32_ver(). Información más completa en Windows.

- **Información de la máquina**
  - machine(). Arquitectura, 'x86-64', 'arm64'
  - processor(). Retorna el nombre del procesador. Puede indicar lo mismo o venir vacío.

- **Información de Python**
  - python_version(). Versión de Python que ejecuta el programa.

pathlib: Proporciona clases para manejar rutas (entradas de directorio).

Una clase es una estructura que agrupa datos y funciones sobre los datos (estas funciones se denominan métodos en este contexto). Crear una clase es como tener un nuevo tipo de dato junto a las funciones para manejarlo.

No vamos a crear clases, pero sí a utilizarlas. Crear un dato a partir de una clase se llama crear o instanciar un objeto a partir de la clase.

Ya habéis creado objetos con los constructores de tipos int(), float(), etc. El constructor de una clase tiene el nombre de una clase (comienzan en mayúsculas) y se le pasan o no argumentos para inicializar sus datos internos.

Por ejemplo, dando por hecho que existe la clase Coche, puedo crear una variable invocando a su constructor y pasándole un argumento, en este caso la matrícula.

mi_coche = Coche(“1234ABC”)

Una vez tengo creado el objeto a partir de la clase, accedo a los datos internos (se llaman propiedades o atributos) o a las funciones (se llaman métodos) usando el nombre de la variable: variable.método() o variable.propiedad. Nótese que las propiedades no llevan paréntesis, mientras que los métodos sí ya que son funciones invocadas y se le pueden pasar parámetros.

Por ejemplo, la clase Coche tiene una propiedad que es la matrícula y un método que es acelerar, pasándole por parámetro la velocidad deseada.

```
print(mi_coche.matricula)

mi_coche.acelerar(120)
```

Pathlib proporciona un conjunto de clases para manejar rutas y operar con ellas. Una de estas clases, que además es genérica (servirá para Windows o Linux igualmente) es Path.

No es necesario importar todo el módulo, solamente la clase -> from pathlib import Path.

Una vez importado, creamos un objeto pasándole como argumento un str con la ruta.
Admite rutas absolutas o relativas (además del uso del . y ..)

```python
from pathlib import Path

p = Path("documentos/archivo.txt")  # ruta relativa
abs_p = Path("/home/usuario")       # ruta absoluta (Linux/macOS)
home = Path.home()                  # directorio del usuario
cwd = Path.cwd()                    # directorio actual

# Construcción de rutas con operador /
ruta = home / "proyectos" / "ejemplo.py"
```

La construcción de rutas con '/' es multiplataforma. La librería se encarga de crear la ruta correcta según el sistema donde se ejecuta (Linux o Windows).

Print: Si hacemos print(ruta), mostraremos la cadena de texto que representa la ruta almacenada en el objeto.

Una vez creado el objeto, podemos invocar todos los siguientes métodos y consultar propiedades a través de la variable. Algunas de ellas son:

Propiedades Considerando ejemplo ruta = Path("/home/alumnom/Documentos/archivo.txt")

- name: Nombre del archivo con extensión (archivo.txt)
- stem: Nombre sin extensión (archivo)
- suffix: Extensión con punto (.txt)
- suffixes: Lista de extensiones.
- parent: Directorio contenedor (Documentos)
- parents: Secuencia de objetos Path (o subclases de Path) con cada ancestro de la ruta, es decir, para el ejemplo: un objeto Path con "/", un objeto Path con "/home", un objeto Path "/home/alumnom", objeto Path "/home/alumnom/Documentos". Podemos recorrer esta secuencia en un for o convertirla a lista con list().
- drive: Unidad (en Windows)
- .root: Raíz ('/')
- anchor: Unidad y raíz ('/' o por ejemplo en Windows 'C:\\')

Métodos

- **Consulta tipo y estado de la ruta**
  - exists(*follow_symlinks=True*): Retorna True si existe la ruta. Por defecto sigue los enlaces simbólicos.
  - is_file(*follow_symlinks=True*): Retorna True si es un archivo regular. Retorna False si no lo es, no existe, es un enlace simbólico roto y otros errores.

- is_dir(*follow_symlinks=True*): Retorna True si es un directorio. `False` también se retorna si la ruta no existe o es un enlace simbólico roto; se propagan otros errores (como errores de permiso).
- is_symlink(): Retorna `True` si la ruta apunta a un enlace simbólico, `False` de lo contrario. `False` también se retorna si la ruta no existe; se extiende a otros errores (como errores de permiso).
- .samefile(ruta2): Retorna True si la ruta apunta al mismo archivo que *ruta2*, que puede ser un objeto *Path* o una cadena.
- stat(*follow_symlinks=True*): Devuelve un nuevo objeto de tipo os.stat_result con información (se puede acceder a cada campo añadiendo .nombre_atributo). Este método normalmente sigue enlaces simbólicos; para evitar enlaces simbólicos, agregue el argumento `follow_symlinks = False`, o use `lstat()`. Las estadísticas cambiarán en función del sistema de archivos o sistema operativo.

- **Iteración de directorios**
  - iterdir(). Crea un generador (como un range()) de objetos Path, para recorrer el contenido de un directorio dado (se puede recorrer con un for o generar una lista con ellos) Las entradas '.' y '..' no están incluidas. Las rutas hijas generadas están creadas en función de la ruta original: si la ruta original es relativa, los objetos Path del generador son rutas relativas. Si la ruta original es absoluta, los objetos Path son rutas absolutas.
  - glob(*pattern*, *case_sensitive=None*, *recurse_symlinks=False*). Como el anterior, pero aplicando un filtro mediante un patrón. Distingue entre mayúsculas y minúsculas en POSIX y no distingue entre mayúsculas y minúsculas en Windows. Se configura *case_sensitive* en `True` o `False` para anular este comportamiento. Los patrones siguen estas reglas.
  - rglob(). Igual que el anterior pero recursivo (buscando en subdirectorios).

- **Expansión y resolución de rutas**
  - Path.home(): Retorna objeto Path con la ruta absoluta del directorio del usuario. Esta función se invoca directamente a través de la librería Path, no a través de un objeto.
  - Path.cwd(): Retorna objeto que almacena la ruta al directorio de trabajo actual.

- expanduser(): Si la ruta almacenada en el objeto Path contiene '~', este método retorna un nuevo objeto Path con ~ expandida para tener una ruta absoluta.

```
>>> p = PosixPath('~/films/Monty Python')
>>> p.expanduser()
PosixPath('/home/eric/films/Monty Python')
```

- absolute(). Retorna nuevo objeto ruta con la ruta absoluta, sin normalización ni resolución de enlaces simbólicos. Básicamente lo que hace es añadir a la ruta, la ruta absoluta del directorio de trabajo.
- resolve(). Similar al anterior pero resuelve enlaces simbólicos y limpia '..' y '.'.

```
>>> ruta = Path('.')
>>> print(ruta.resolve())
/home/pedro/Documentos
```

- readlink(): Retorna la ruta a la que apunta un enlace simbólico.

- **Creación, cambios y eliminación**
  - **mkdir(*mode=0o777*, *parents=False*, *exist_ok=False*).** Crea un nuevo directorio en la ruta que almacena el objeto Path. Si parents es True se crean los padres que faltan, si es false y falta algún padre se genera un error. Si exist_ok es False, se genera un error si ya existe, si es True, se genera error si existe y no es un directorio.
  - **touch(*mode=0o666*, *exist_ok=True*).** Crea un archivo.
  - **rmdir().** Borra directorio vacío.
  - **symlink_to(*target*).** Hace que la ruta almacenada en el objeto Path sea un enlace simbólico de "target".
  - **hardlink_to(*target*).** Igual para enlaces duros.
  - **rename(*target*).** Renombra o mueve un archivo. El método retorna el objeto Path con la nueva ruta. Target puede ser un str o Path. En Windows si target existe se genera error.
  - **replace(*target*).** Igual que el anterior forzando si existe.
  - **unlink(*missing_ok=False*).** Elimina el archivo. Si missing_ok es True no lanza error si el archivo no existe.

- **Usuario, grupo y permisos:** Existen métodos similares a los comandos Linux pero no funcionan en Windows.

- **Lectura y escritura de archivos:** La clase Path también nos permite leer y escribir ficheros. Los veremos más adelante, ya que esto forma parte del tema 3.

shutil: Manejar archivos y colecciones de archivos a alto nivel (copiar , mover, eliminar, comprimir, info del disco). Las siguientes son funciones a invocar directamente de la librería (no son métodos de un objeto). Donde pone src o dst puede ser un string o un objeto Path.

- **Copiar archivos**
  - `shutil.copy(source, destination)`
  - `shutil.copy2(src, dst)`
  - `shutil.copyfile(src, dst)`
  - `shutil.copymode(src, dst)`
- **Copiar directorios**
  - `shutil.copytree(src, dst, dirs_exist_ok=False)`
- **Mover o renombrar archivos o directorios** (equivalente a Path.rename() funcionando mejor entre diferentes particiones o discos.
  - `shutil.move(src, dst)`
- **Borrar archivos y directorios**
  - `shutil.rmtree(path)`
- **Comprimir y descomprimir**
  - `shutil.make_archive(nombre, format, root_dir)`
  - `shutil.unpack_archive(filename, extract_dir)`
- **Información del disco**
  - `shutil.disk_usage(path)`

os. Esta librería nos permite trabajar a bajo nivel para hacer muchas funcionalidades de gestión del SO y obtener información (incluso las que hemos visto anteriormente mediante pathlib y shutil, pero estas lo hacen de forma más eficiente o cómoda).