Generalitat de Catalunya
Departament d'Educació
Institut Thos i Codina

El mòdul 'pathlib' de Python

Domini Públic (CC0)

Índex

1. Les classes de `pathlib` **1**

2. Crear camins **1**
L'operador `/` . . . 2
El directori del propi script . . . 2

3. Parts d'un camí **2**
Canviar parts: `with_name`, `with_suffix`, `with_stem` . . . 3

4. Camins absoluts, relatius i resolts **3**
Camins relatius a un altre: `relative_to` . . . 4

5. Consultar el sistema de fitxers **4**
Metadades: `stat()` . . . 4

6. Llegir i escriure fitxers **5**

7. Directoris **6**
Crear i eliminar . . . 6
Llistar el contingut: `iterdir()` . . . 6
Cerca amb patrons: `glob()` i `rglob()` . . . 6
Recórrer un arbre: `walk()` (Python 3.12+) . . . 7

8. Operacions amb fitxers **7**
Diferència entre `rename` i `replace` . . . 7
Permisos . . . 8
Enllaços . . . 8
Copiar i moure . . . 8

9. Comparar i buscar patrons en camins **8**

10. Camins "purs": manipular sense tocar el disc **9**

11. Interoperabilitat amb os i la resta del món **9**
Taula de correspondències `os.path` ↔ `pathlib` . . . 10

12. Seguretat: camins que venen de fora (*path traversal*) **10**

13. Exemples complets **11**
13.1. Llistar els fitxers més grans d'un directori . . . 11
13.2. Mida total d'un directori . . . 11
13.3. Organitzar fitxers per extensió . . . 11
13.4. Reanomenar fitxers en lot . . . 12
13.5. Escriptura segura d'un fitxer (atòmica) . . . 12
13.6. Mostrar un directori en forma d'arbre . . . 12
13.7. Còpia de seguretat amb data al nom . . . 12
13.8. Trobar fitxers duplicats pel seu contingut . . . 13

14. Errors habituals **13**

15. Exercicis proposats **14**

16. Resum **14**

17. Referències **15**

`pathlib` ofereix una **interfície orientada a objectes per treballar amb camins del sistema de fitxers**. En lloc de manipular cadenes amb `os.path.join()`, `os.path.dirname()`, etc., treballes amb objectes `Path` que tenen mètodes i atributs, i que s'uneixen amb l'operador `/`. El resultat és codi més curt, més llegible i portable entre Linux, macOS i Windows.

```
from pathlib import Path
```

NOTA

Un `Path` **no és un fitxer**, és la representació d'un camí. Pots crear un `Path` a un fitxer que no existeix. Només quan crides a un mètode que consulta o modifica el disc (`exists()`, `read_text()`, `mkdir()`...) s'accedeix realment al sistema de fitxers.

1. Les classes de `pathlib`

Classe	Accedeix al disc	Descripció
`PurePath`	No	Classe base: només manipula camins com a text
`PurePosixPath`	No	Camins estil Unix (`/`), sigui quin sigui el SO
`PureWindowsPath`	No	Camins estil Windows (`C:\`), sigui quin sigui el SO
`Path`	**Sí**	Camí concret; en crear-lo es converteix en la subclasse adequada
`PosixPath`	Sí	`Path` en sistemes Unix (és el que obtens a Linux/macOS)
`WindowsPath`	Sí	`Path` a Windows

En el dia a dia només cal `Path`: Python tria `PosixPath` o `WindowsPath` segons el sistema on s'executa.

```
p = Path("/etc/hostname")
print(type(p))     # <class 'pathlib.PosixPath'> (a Linux)
```

Els objectes `Path` són **immutables** i es poden fer servir com a claus de diccionari: cada operació retorna un `Path` nou.

2. Crear camins

```
Path("document.txt")                    # camí relatiu
Path("/home/ramon/docs")                # camí absolut
Path("carpeta", "subcarpeta", "fitxer.txt")  # unió de components

Path.cwd()                              # directori de treball actual
Path.home()                            # directori personal de l'usuari
```

L'operador /

La manera idiomàtica d'unir camins:

```
base = Path("/home/ramon")
informe = base / "docs" / "informe.pdf"
print(informe)        # /home/ramon/docs/informe.pdf

# També amb cadenes pel mig
informe = base / "docs/informe.pdf"
```

`joinpath()` fa el mateix: `base.joinpath("docs", "informe.pdf")`.

AVÍS

Igual que `os.path.join`, **un component absolut descarta tot l'anterior**:

```
Path("/home/ramon") / "/etc/passwd"   # → PosixPath('/etc/passwd')
```

Si una part del camí ve de l'usuari, valida-la abans (vegeu l'apartat de seguretat).

El directori del propi script

El directori de treball (CWD) depèn de **des d'on** s'executa el programa. Per localitzar fitxers que van al costat del script, no confiïs en el CWD:

```
BASE = Path(__file__).resolve().parent
config = BASE / "config.ini"
dades = BASE / "dades" / "entrada.csv"
```

3. Parts d'un camí

```
p = Path("/home/ramon/docs/informe.final.tar.gz")

p.name          # 'informe.final.tar.gz'     (últim component)
p.stem          # 'informe.final.tar'        (nom sense l'últim sufix)
p.suffix        # '.gz'                      (últim sufix)
p.suffixes      # ['.final', '.tar', '.gz']  (tots els sufixos)
p.parent        # PosixPath('/home/ramon/docs')
p.parents[0]    # PosixPath('/home/ramon/docs')
p.parents[1]    # PosixPath('/home/ramon')
p.parts         # ('/', 'home', 'ramon', 'docs',
 ↪  'informe.final.tar.gz')
p.anchor        # '/'                        (arrel; a Windows 'C:\\')
p.drive         # ''                         (a Windows 'C:')
```

Puntualitzacions:

- `suffix` només retorna **l'últim** sufix. Per a `.tar.gz` caldria mirar `suffixes`.
- Un fitxer ocult com `.bashrc` té `suffix == ""` i `stem == ".bashrc"`: el punt inicial no es considera una extensió.
- `parents` és una seqüència: `list(p.parents)` dona tots els directoris ancestres fins a l'arrel.

Canviar parts: `with_name`, `with_suffix`, `with_stem`

Retornen un camí nou amb la part substituïda (no toquen el disc):

```
p = Path("/home/ramon/docs/informe.txt")

p.with_name("resum.txt")       # /home/ramon/docs/resum.txt
p.with_suffix(".md")           # /home/ramon/docs/informe.md
p.with_suffix("")              # /home/ramon/docs/informe  (treu
 ↪  l'extensió)
p.with_stem("informe_v2")      # /home/ramon/docs/informe_v2.txt
 ↪  (Python 3.9+)
```

NOTA

Amb `.tar.gz`, `with_suffix(".zip")` només canvia l'últim sufix: `a.tar.gz` → `a.tar.zip`. Si ho vols treure tot, fes-ho a mà: `p.with_name(p.name.removesuffix(".tar.gz") + ".zip")`.

4. Camins absoluts, relatius i resolts

```
p = Path("docs/informe.txt")

p.is_absolute()        # False
p.absolute()           # /directori/actual/docs/informe.txt (només hi
 ↪  afegeix el CWD al davant)
p.resolve()            # absolut + elimina '..' i '.' + resol enllaços
 ↪  simbòlics
Path("~/docs").expanduser()   # /home/ramon/docs
```

Mètode	Fa
`absolute()`	Afegeix el CWD davant si és relatiu. No normalitza `..` ni segueix enllaços
`resolve()`	Absolut, normalitzat i amb els **enllaços simbòlics resolts**
`expanduser()`	Substitueix ~ pel directori personal. **Els `Path` no expandeixen ~ per si sols**

Camins relatius a un altre: `relative_to`

```
p = Path("/home/ramon/docs/informe.txt")

p.relative_to("/home/ramon")        # PosixPath('docs/informe.txt')
p.is_relative_to("/home/ramon")     # True  (Python 3.9+)
p.is_relative_to("/etc")            # False

p.relative_to("/etc")               # ValueError: no és subcamí
```

`relative_to` **no afegeix `..`**: només funciona si el camí és realment dins de l'altre. És la base de la comprovació de seguretat que veurem més endavant.

5. Consultar el sistema de fitxers

```
p = Path("/etc/passwd")

p.exists()       # existeix?
p.is_file()      # és un fitxer regular?
p.is_dir()       # és un directori?
p.is_symlink()   # és un enllaç simbòlic?
p.is_mount()     # és un punt de muntatge? (Unix)
p.samefile("/etc/passwd")   # apunten al mateix fitxer real?
```

Metadades: `stat()`

```
info = p.stat()
print(info.st_size)    # mida en bytes
print(info.st_mtime)   # última modificació (segons des de 1970)

import datetime as dt
print(dt.datetime.fromtimestamp(info.st_mtime))
```

`p.lstat()` fa el mateix, però **sense seguir** l'enllaç simbòlic (informa sobre l'enllaç mateix). En sistemes Unix, `p.owner()` i `p.group()` retornen els noms de propietari i grup.

NOTA

Condicions de carrera (TOCTOU): fer `if p.exists(): p.read_text()` no garanteix que el fitxer continuï existint un instant després. En codi crític, és més robust provar l'operació i capturar l'excepció (`FileNotFoundError`).

6. Llegir i escriure fitxers

Per a fitxers petits o mitjans, `pathlib` ho fa en una línia, sense `open()` ni `with`:

```
p = Path("notes.txt")

p.write_text("Hola, món!\nSegona línia\n", encoding="utf-8")
text = p.read_text(encoding="utf-8")

p.write_bytes(b"\x00\x01\x02")
dades = p.read_bytes()
```

- **Indica sempre `encoding="utf-8"`** en text. Si no, s'usa la codificació per defecte del sistema, que pot variar entre màquines (típicament problemes d'accents a Windows).
- `write_text` i `write_bytes` **sobreescriuen** el fitxer si ja existia, sense avisar.
- `read_text` carrega tot el fitxer a memòria. Per a fitxers grans, obre'l i itera:

```
with p.open(encoding="utf-8") as f:
    for linia in f:
        print(linia.rstrip())
```

`Path.open()` accepta els mateixos paràmetres que la funció `open()` (`"r"`, `"w"`, `"a"`, `"rb"`, `encoding`...).

Afegir al final d'un fitxer:

```
with p.open("a", encoding="utf-8") as f:
    f.write("Una línia més\n")
```

7. Directoris

Crear i eliminar

```
Path("nou").mkdir()                                    # un sol nivell
Path("a/b/c").mkdir(parents=True, exist_ok=True)     # tota la
 ↪  jerarquia, sense error si ja existeix

Path("nou").rmdir()                                  # només si és buit
```

Paràmetre de `mkdir`	Efecte
`parents=True`	Crea també els directoris pares que faltin
`exist_ok=True`	No llança `FileExistsError` si ja existeix
`mode=0o755`	Permisos (modificats per l'*umask*)

Per eliminar un directori **amb contingut**, `pathlib` no té mètode: cal `shutil.rmtree(p)`.

Llistar el contingut: `iterdir()`

```
for entrada in Path(".").iterdir():
    tipus = "DIR " if entrada.is_dir() else "FILE"
    print(tipus, entrada.name)
```

- Retorna un **generador** de `Path` (camins complets, no només noms).
- L'ordre és **arbitrari**; si vols ordre: `sorted(p.iterdir())`.
- No inclou `.` ni `...`

Cerca amb patrons: `glob()` i `rglob()`

```
d = Path("/home/ramon/projecte")

list(d.glob("*.py"))           # fitxers .py directament dins de d
list(d.glob("*/*.py"))         # un nivell més avall
list(d.glob("**/*.py"))        # recursiu (inclou d)
list(d.rglob("*.py"))          # equivalent curt de glob("**/*.py")
list(d.rglob("test_*.py"))     # test_xxx.py a qualsevol profunditat
```

Patró	Significat
`*`	Qualsevol seqüència de caràcters (sense `/`)
`?`	Un sol caràcter
`[abc]` / `[0-9]`	Un caràcter del conjunt / rang
`**`	Zero o més directoris (recursiu)

Notes:

- `glob` i `rglob` retornen **generadors**: convé consumir-los amb un bucle o `list()`.
- Per defecte, **inclouen fitxers ocults** (a diferència de la shell).
- A Windows la cerca no distingeix majúscules; a Linux sí. Des de Python 3.12 existeix el paràmetre `case_sensitive=` per controlar-ho.

Recórrer un arbre: `walk()` (Python 3.12+)

```
for arrel, directoris, fitxers in Path("/etc").walk():
    print(arrel, len(fitxers), "fitxers")
```

Funciona igual que `os.walk` però amb objectes `Path`. En versions anteriors, usa `os.walk` o `rglob("*")`.

8. Operacions amb fitxers

```
p = Path("vell.txt")

p.touch()                          # crea el fitxer buit, o actualitza
 ↪  la data si ja existeix
p.touch(exist_ok=False)           # llança FileExistsError si ja
 ↪  existeix

p.rename("nou.txt")               # reanomena o mou (retorna el Path
 ↪  nou, Python 3.8+)
p.replace("nou.txt")              # igual, però sobreescriu el destí
 ↪  en tots els SO

Path("nou.txt").unlink()                    # elimina el fitxer
Path("nou.txt").unlink(missing_ok=True)     # sense error si no
 ↪  existeix (Python 3.8+)
```

Diferència entre `rename` i `replace`

Si el destí ja existeix, `rename` es comporta diferent segons el SO (a Windows falla; a Unix sobreescriu). `replace` sempre sobreescriu i, dins d'un mateix sistema de fitxers, és **atòmic**: ideal per al patró d'escriptura segura.

Permisos

```
p.chmod(0o640)        # rw-r-----   (només Unix)
```

Enllaços

```
Path("enllac").symlink_to("/etc/hostname")       # crea un enllaç
 ↪  simbòlic
Path("enllac").readlink()                        # destí de l'enllaç
 ↪  (Python 3.9+)
Path("dur").hardlink_to("/etc/hostname")         # enllaç dur (Python
 ↪  3.10+)
```

Copiar i moure

Fins a Python 3.13, `Path` **no té mètode per copiar**: es combina amb `shutil`, que accepta objectes `Path`:

```
import shutil

shutil.copy2(Path("a.txt"), Path("backup") / "a.txt")
shutil.copytree(Path("projecte"), Path("projecte_backup"))
shutil.move(Path("a.txt"), Path("arxiu"))
```

A partir de **Python 3.14** hi ha mètodes natius (`Path.copy()`, `Path.copy_into()`, `Path.move()`, `Path.move_into()`), però `shutil` continua sent la solució més portable entre versions.

9. Comparar i buscar patrons en camins

```
Path("informe.pdf") == Path("informe.pdf")     # True

Path("/home/ramon/docs/a.txt").match("*.txt")        # True (compara
 ↪  des de la dreta)
Path("/home/ramon/docs/a.txt").match("docs/*.txt")   # True
```

- La comparació **distingeix majúscules** a Unix i **no** a Windows (`PureWindowsPath`).
- Es poden **ordenar** camins del mateix tipus (`sorted(llista_de_paths)`).
- `match()` aplica el patró començant per la dreta del camí. Des de Python 3.13, `full_match()` el compara amb tot el camí.

10. Camins "purs": manipular sense tocar el disc

Les classes `PurePath` serveixen per treballar amb camins **d'un altre sistema** o simplement quan no vols accedir al disc:

```
from pathlib import PureWindowsPath, PurePosixPath

w = PureWindowsPath(r"C:\Users\ramon\Documents\informe.docx")
print(w.name)        # informe.docx
print(w.drive)       # C:
print(w.parts)       # ('C:\\', 'Users', 'ramon', 'Documents',
 ↪  'informe.docx')

# Convertir un camí Windows a estil Unix (per exemple per a un URL o
 ↪  una configuració)
print(PurePosixPath(*w.parts[1:]))   #
 ↪  Users/ramon/Documents/informe.docx
print(w.as_posix())                  #
 ↪  C:/Users/ramon/Documents/informe.docx
```

És útil, per exemple, per analitzar camins que venen d'un servidor Windows des d'un script que corre a Linux (rutes SMB/UNC, camins en logs…).

11. Interoperabilitat amb os i la resta del món

`Path` implementa el protocol `os.PathLike`, de manera que **la major part de funcions de la biblioteca estàndard l'accepten directament**:

```
import os, shutil, subprocess

p = Path("/tmp/dades.txt")

open(p)                          # OK
os.stat(p)                       # OK
shutil.copy(p, "/tmp/copia.txt")  # OK
subprocess.run(["cat", p])      # OK (Python 3.6+ accepta PathLike als
 ↪  arguments)
```

Quan una biblioteca antiga espera un `str`:

```
str(p)          # '/tmp/dades.txt'
os.fspath(p)    # igual, però és la manera "oficial"
```

I a l'inrevés, qualsevol `str` es converteix amb `Path(cadena)`.

Taula de correspondències `os.path` ↔ `pathlib`

Tasca	`os` / `os.path`	`pathlib`
Unir camins	`os.path.join(a, b)`	`Path(a) / b`
Nom del fitxer	`os.path.basename(p)`	`p.name`
Directori pare	`os.path.dirname(p)`	`p.parent`
Extensió	`os.path.splitext(p)[1]`	`p.suffix`
Nom sense extensió	`os.path.splitext(basename)[0]`	`p.stem`
Absolut	`os.path.abspath(p)`	`p.absolute()` / `p.resolve()`
Expandir ~	`os.path.expanduser(p)`	`p.expanduser()`
Existeix?	`os.path.exists(p)`	`p.exists()`
És fitxer / directori	`os.path.isfile(p)` / `isdir(p)`	`p.is_file()` / `p.is_dir()`
Mida	`os.path.getsize(p)`	`p.stat().st_size`
Llistar	`os.listdir(p)`	`p.iterdir()`
Cerca per patró	`glob.glob(patró)`	`p.glob(patró)`
Crear jerarquia	`os.makedirs(p, exist_ok=True)`	`p.mkdir(parents=True, exist_ok=True)`
Eliminar fitxer	`os.remove(p)`	`p.unlink()`
Reanomenar	`os.rename(a, b)`	`a.rename(b)`
Directori actual	`os.getcwd()`	`Path.cwd()`
Llegir fitxer	`open(p).read()`	`p.read_text()`

`os` continua sent necessari per a variables d'entorn, processos, permisos de baix nivell i `os.walk` en versions anteriors a 3.12.

12. Seguretat: camins que venen de fora (*path traversal*)

Si un camí es construeix amb dades externes (nom de fitxer d'un formulari, d'un arxiu `zip`, d'un paràmetre d'una API), un atacant pot enviar `../../etc/passwd` o un camí absolut per sortir del directori previst.

```
from pathlib import Path

BASE = Path("/srv/uploads").resolve()

def camí_segur(nom_extern: str) -> Path:
    candidat = (BASE / nom_extern).resolve()
    if not candidat.is_relative_to(BASE):      # Python 3.9+
        raise ValueError(f"Camí fora del directori permès:
        ↪  {nom_extern}")
    return candidat

camí_segur("informe.pdf")            # OK
camí_segur("../../etc/passwd")       # ValueError
camí_segur("/etc/passwd")            # ValueError (un component
 ↪  absolut descarta BASE)
```

L'important és fer `resolve()` **abans** de comprovar `is_relative_to`, perquè així es normalitzen els `..` i se segueixen els enllaços simbòlics.

13. Exemples complets

13.1. Llistar els fitxers més grans d'un directori

```
from pathlib import Path

def mes_grans(directori, n=5):
    fitxers = [p for p in Path(directori).rglob("*") if p.is_file()]
    return sorted(fitxers, key=lambda p: p.stat().st_size,
     ↪  reverse=True)[:n]

for p in mes_grans("/var/log"):
    print(f"{p.stat().st_size / 1024:>10.1f} KiB  {p}")
```

13.2. Mida total d'un directori

```
from pathlib import Path

def mida_directori(directori):
    return sum(p.stat().st_size for p in Path(directori).rglob("*") if
     ↪  p.is_file())

print(f"{mida_directori('.') / 1024**2:.2f} MiB")
```

13.3. Organitzar fitxers per extensió

```
import shutil
from pathlib import Path

origen = Path.home() / "Downloads"

for fitxer in origen.iterdir():
    if fitxer.is_file() and fitxer.suffix:
        desti = origen / fitxer.suffix.lstrip(".").lower()
        desti.mkdir(exist_ok=True)
        shutil.move(fitxer, desti / fitxer.name)
```

13.4. Reanomenar fitxers en lot

```
from pathlib import Path

carpeta = Path("fotos")
for i, f in enumerate(sorted(carpeta.glob("*.jpg")), start=1):
    f.rename(f.with_name(f"vacances_{i:03d}{f.suffix}"))
```

13.5. Escriptura segura d'un fitxer (atòmica)

```
from pathlib import Path

def escriu_atomic(cami, contingut):
    cami = Path(cami)
    tmp = cami.with_name(cami.name + ".tmp")
    tmp.write_text(contingut, encoding="utf-8")
    tmp.replace(cami)                    # substitució atòmica

escriu_atomic("config.ini", "[general]\nidioma = ca\n")
```

13.6. Mostrar un directori en forma d'arbre

```
from pathlib import Path

def arbre(directori, prefix=""):
    entrades = sorted(Path(directori).iterdir(), key=lambda p:
    ↪  (p.is_file(), p.name.lower()))
    for i, entrada in enumerate(entrades):
        ultim = i == len(entrades) - 1
        print(prefix + ("└── " if ultim else "├── ") + entrada.name)
        if entrada.is_dir() and not entrada.is_symlink():
            arbre(entrada, prefix + ("    " if ultim else "│   "))

arbre(".")
```

13.7. Còpia de seguretat amb data al nom

```
import datetime as dt
import shutil
from pathlib import Path

origen = Path("/home/ramon/projecte")
desti = Path("/mnt/backups") /
 ↪  f"{origen.name}_{dt.datetime.now():%Y%m%d_%H%M%S}"
```

```
shutil.copytree(origen, desti,
 ↪  ignore=shutil.ignore_patterns("__pycache__", "*.pyc", ".git"))
print("Backup creat a", desti)
```

13.8. Trobar fitxers duplicats pel seu contingut

```
import hashlib
from collections import defaultdict
from pathlib import Path

def hash_fitxer(p, bloc=1024 * 1024):
    h = hashlib.sha256()
    with p.open("rb") as f:
        while dades := f.read(bloc):
            h.update(dades)
    return h.hexdigest()

per_hash = defaultdict(list)
for p in Path("/home/ramon/fotos").rglob("*"):
    if p.is_file():
        per_hash[hash_fitxer(p)].append(p)

for h, camins in per_hash.items():
    if len(camins) > 1:
        print("Duplicats:")
        for c in camins:
            print("  ", c)
```

14. Errors habituals

1. **Oblidar `encoding="utf-8"`** a `read_text()` / `write_text()` i tenir accents trencats entre sistemes.
2. **Esperar que `Path("~/docs")` expandeixi `~`**: cal `expanduser()`.
3. **Unir amb un component absolut** (`base / "/etc"`) i perdre la base.
4. **Confondre `suffix` amb `suffixes`** en fitxers `.tar.gz`.
5. **`with_suffix` sobre noms amb diversos punts**: només canvia l'últim.
6. **Consumir dues vegades un generador** (`glob`, `iterdir`): la segona vegada és buit. Converteix a llista si cal.
7. **Suposar que `iterdir`/`glob` retornen ordenat**: no ho fan; usa `sorted()`.
8. **`rmdir()` en un directori no buit** → `OSError`. Per a arbres complets, `shutil.rmtree`.
9. **`mkdir()` sense `parents=True`** quan el pare no existeix → `FileNotFoundError`.
10. **Barrejar `str` i `Path`** amb + (`path + "/fitxer"` dona `TypeError`). Usa `/`.
11. **`relative_to` amb camins no relacionats** → `ValueError`. Comprova abans amb `is_relative_to` (3.9+).
12. **Fer servir `write_text` com a "desa" en dades importants**: no és atòmic ni conserva còpia; usa el patró d'escriptura segura.

15. Exercicis proposats

1. Escriu un script que, donada una ruta, mostri nom, extensió, directori pare, mida i data de modificació.
2. Fes una funció `fitxers_per_extensio(directori)` que retorni un diccionari `{extensió: nombre de fitxers}` recorrent el directori recursivament.
3. Escriu un programa que trobi els fitxers de més de 100 MiB dins de `/var` i els mostri ordenats per mida.
4. Crea l'estructura `projecte/{src,docs,tests}` i un fitxer `README.md` a l'arrel amb `pathlib`, de manera que es pugui executar diverses vegades sense errors.
5. Reanomena tots els fitxers `.txt` d'un directori a minúscules i amb guions en lloc d'espais (`Meu Fitxer.TXT` → `meu-fitxer.txt`).
6. Implementa la funció `camí_segur()` de la secció 12 i prova-la amb camins amb `..`, camins absoluts i enllaços simbòlics.
7. Converteix l'script Mida total d'un directori a `pathlib` i compara'n la llegibilitat.
8. Fes un script que mostri l'arbre d'un directori fins a una profunditat màxima indicada per l'usuari (`--depth`) amb `argparse`.

16. Resum

Vull…	Eina
Crear un camí	`Path("a/b.txt")`
Unir camins	`base / "dir" / "fitxer.txt"`
Directori actual / personal	`Path.cwd()` / `Path.home()`
Directori del script	`Path(__file__).resolve().parent`
Nom / extensió / pare	`p.name` / `p.suffix` / `p.parent`
Canviar extensió o nom	`p.with_suffix(".md")` / `p.with_name("x.txt")`
Camí absolut i normalitzat	`p.resolve()`
Camí relatiu a un altre	`p.relative_to(base)`
Existeix / fitxer / directori	`p.exists()` / `p.is_file()` / `p.is_dir()`
Llegir / escriure text	`p.read_text(encoding="utf-8")` / `p.write_text(...)`
Llistar un directori	`p.iterdir()`
Cercar per patró	`p.glob("*.py")` / `p.rglob("*.py")`
Crear jerarquia de carpetes	`p.mkdir(parents=True, exist_ok=True)`
Eliminar fitxer	`p.unlink(missing_ok=True)`
Reanomenar / substituir	`p.rename(nou)` / `p.replace(nou)`
Mida i data	`p.stat().st_size` / `p.stat().st_mtime`
Copiar / moure / esborrar arbres	`shutil`
Convertir a text	`str(p)` / `os.fspath(p)`

17. Referències

- Documentació oficial de `pathlib`: https://docs.python.org/es/3.15/library/pathlib.html
- PEP 428 (disseny de `pathlib`): https://peps.python.org/pep-0428/
- `os.path`: https://docs.python.org/es/3.15/library/os.path.html
- `shutil`: https://docs.python.org/es/3.15/library/shutil.html
- Protocol `os.PathLike`: https://docs.python.org/es/3.15/library/os.html#os.PathLike

Versions d'aquest document

- HTML
- PDF
- ODT
- MD

Domini Públic (CC0)