Generalitat de Catalunya
Departament d'Educació
Institut Thos i Codina

El mòdul 'os' de Python

Domini Públic (CC0)

Índex

1. Informació bàsica del sistema **1**

2. Directori de treball **1**

3. Llistar el contingut d'un directori **2**
`os.listdir()` . . . 2
`os.scandir()` (recomanat) . . . 2
Recórrer un arbre: `os.walk()` . . . 2

4. Crear i eliminar directoris **3**

5. Operacions amb fitxers **3**
Metadades: `os.stat()` . . . 4
Permisos (Unix) . . . 4

6. `os.path`: treballar amb camins **4**

7. Variables d'entorn **5**

8. Executar ordres i processos **6**
`os.system()` (només per a coses ràpides) . . . 6
Alternativa recomanada: `subprocess` . . . 6
Identificadors de procés (Unix) . . . 6
`os.fork()` i `os.exec*` (Unix) . . . 6

9. Fitxers temporals i aleatorietat **7**

10. Gestió d'errors **7**

11. `os` i `pathlib`: quina fer servir? **7**

12. Exemples complets **8**
12.1. Mida total d'un directori . . . 8
12.2. Cercar fitxers per extensió . . . 8
12.3. Reanomenar fitxers en lot . . . 8
12.4. Escriptura atòmica d'un fitxer . . . 9

13. Exercicis proposats **9**

14. Resum **9**

15. Referències **10**

El mòdul `os` és la interfície de la biblioteca estàndard de Python amb el sistema operatiu. Permet treballar amb fitxers i directoris, variables d'entorn, processos i informació del sistema d'una manera **portable** entre Linux, macOS i Windows.

NOTA

Per què existeix? Cada sistema operatiu té les seves pròpies crides al sistema i convencions (separador de camins `/` o `\`, permisos, variables d'entorn…). `os` amaga aquestes diferències darrere una API comuna, i quan no es pot, exposa funcions específiques de plataforma (per exemple `os.getuid()` només existeix en sistemes Unix).

```
import os
```

1. Informació bàsica del sistema

```
import os

print(os.name)        # 'posix' (Linux/macOS) o 'nt' (Windows)
print(os.sep)         # '/' o '\\'
print(os.linesep)     # '\n' o '\r\n'
print(os.cpu_count()) # nombre de CPU lògiques (pot ser None)
print(os.getlogin())  # usuari que ha iniciat la sessió
print(os.uname())     # només Unix: sysname, nodename, release,
 ↪  version, machine
```

Constant	Significat
`os.name`	Família del SO: `posix`, `nt` o `java`
`os.sep`	Separador de components d'un camí
`os.pathsep`	Separador de camins dins `PATH` (`:` o `;`)
`os.curdir` / `os.pardir`	`'.'` i `'..'`
`os.linesep`	Final de línia natiu

Per saber exactament la plataforma, és més precís `sys.platform` o el mòdul `platform`.

2. Directori de treball

Cada procés té un **directori de treball actual** (*current working directory*, CWD). Els camins relatius es resolen respecte d'aquest directori, no respecte de la ubicació del script. Aquest és l'origen d'un error molt comú: executar el mateix script des de dos directoris diferents i que no trobi els fitxers.

```
print(os.getcwd())        # directori actual

os.chdir("/tmp")          # canvia el directori de treball
```

```
print(os.getcwd())        # /tmp
```

Per treballar amb fitxers situats al costat del script, no depenguis del CWD:

```
base = os.path.dirname(os.path.abspath(__file__))
fitxer = os.path.join(base, "dades.csv")
```

3. Llistar el contingut d'un directori

`os.listdir()`

```
for nom in os.listdir("."):
    print(nom)
```

Retorna **només els noms** (no els camins complets), sense ordre garantit i sense `.` ni `..`. Si vols ordre: `sorted(os.listdir(...))`.

`os.scandir()` (recomanat)

Retorna objectes `DirEntry` que ja porten informació del tipus d'entrada, de manera que evita crides addicionals al sistema (`stat`). És més eficient en directoris grans.

```
with os.scandir(".") as entrades:
    for e in entrades:
        tipus = "DIR " if e.is_dir() else "FILE"
        print(tipus, e.name, e.path)
```

Recórrer un arbre: `os.walk()`

```
for arrel, directoris, fitxers in os.walk("/etc"):
    print(f"Directori: {arrel}")
    for f in fitxers:
        print("   ", os.path.join(arrel, f))
```

- Per defecte recorre de dalt cap a baix (`topdown=True`). Aleshores pots **modificar la llista `directoris` in situ** per podar el recorregut:

```
for arrel, directoris, fitxers in os.walk("."):
    directoris[:] = [d for d in directoris if d != ".git"]
```

- No segueix enllaços simbòlics a directoris per defecte (`followlinks=False`), cosa que evita bucles infinits.

4. Crear i eliminar directoris

```
os.mkdir("nou")                        # crea un sol nivell
os.makedirs("a/b/c", exist_ok=True)    # crea tota la jerarquia

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

Funció	Comportament
`os.mkdir(p)`	Falla amb `FileExistsError` si ja existeix o amb `FileNotFoundError` si falta el pare
`os.makedirs(p, exist_ok=True)`	Crea els pares necessaris; no falla si ja existeix
`os.rmdir(p)`	Falla amb `OSError` si el directori no és buit
`os.removedirs(p)`	Elimina el directori i els pares que quedin buits

Per eliminar un directori **amb contingut** no hi ha cap funció a `os`; cal `shutil.rmtree()`:

```
import shutil
shutil.rmtree("a")
```

5. Operacions amb fitxers

```
os.rename("vell.txt", "nou.txt")     # reanomena o mou
os.replace("vell.txt", "nou.txt")    # igual, però sobreescriu el
 ↪  destí en tots els SO
os.remove("nou.txt")                # elimina un fitxer (os.unlink és
 ↪  sinònim)
```

NOTA

Diferència entre `rename` i `replace`: si el destí ja existeix, `os.rename` es comporta diferent segons el SO (a Unix el sobreescriu; a Windows llança `FileExistsError`). `os.replace` té el mateix comportament arreu, i a més és atòmic en el mateix sistema de fitxers, cosa útil per escriure fitxers de configuració de manera segura (escriure a un temporal i després fer `replace`).

Metadades: `os.stat()`

```
info = os.stat("document.txt")
print(info.st_size)    # mida en bytes
print(info.st_mtime)   # última modificació (segons des de l'època)
print(oct(info.st_mode))  # tipus i permisos

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

Permisos (Unix)

```
os.chmod("script.sh", 0o755)   # rwxr-xr-x  (nota: prefix 0o per a
 ↪  octals)
os.chown("fitxer", 1000, 1000) # uid, gid (cal ser root)
```

Comprovar si hi ha accés sense obrir el fitxer:

```
if os.access("fitxer.txt", os.R_OK | os.W_OK):
    print("Lectura i escriptura permeses")
```

AVÍS

Comprovar amb `os.access()` i després obrir crea una *condició de carrera* (TOCTOU: el permís pot canviar entre les dues operacions). El patró més robust és intentar l'operació i capturar l'excepció (`PermissionError`).

6. `os.path`: treballar amb camins

`os.path` conté funcions per manipular **cadenes** que representen camins. No toquen el disc (excepte les de comprovació).

```
p = "/home/ramon/docs/informe.final.pdf"

os.path.basename(p)        # 'informe.final.pdf'
os.path.dirname(p)         # '/home/ramon/docs'
os.path.split(p)           # ('/home/ramon/docs', 'informe.final.pdf')
os.path.splitext(p)        # ('/home/ramon/docs/informe.final',
 ↪  '.pdf')

os.path.join("a", "b", "c.txt")   # 'a/b/c.txt' (o 'a\\b\\c.txt' a
 ↪  Windows)
os.path.abspath("fitxer.txt")     # camí absolut respecte al CWD
os.path.realpath("enllac")        # resol enllaços simbòlics
os.path.expanduser("~/docs")      # substitueix ~ pel directori
 ↪  personal
```

```
os.path.normpath("a//b/../c")     # 'a/c'
```

Comprovacions que consulten el sistema de fitxers:

```
os.path.exists(p)    # existeix?
os.path.isfile(p)    # és un fitxer regular?
os.path.isdir(p)     # és un directori?
os.path.islink(p)    # és un enllaç simbòlic?
os.path.getsize(p)   # mida en bytes
os.path.getmtime(p)  # data de modificació
```

Errors típics:

- Concatenar camins amb + o `f"{a}/{b}"`: no és portable. Usa `os.path.join`.
- `os.path.join("/a", "/b")` retorna `'/b'`: **un component absolut descarta tots els anteriors**. Això pot ser un problema de seguretat si un dels components ve de l'usuari.

7. Variables d'entorn

`os.environ` és un objecte semblant a un diccionari amb les variables d'entorn del procés.

```
print(os.environ["HOME"])           # KeyError si no existeix
print(os.environ.get("EDITOR"))     # None si no existeix
print(os.getenv("EDITOR", "nano"))  # valor per defecte

os.environ["MEVA_VAR"] = "hola"     # s'hereta pels processos fills
del os.environ["MEVA_VAR"]
```

Dos punts importants:

1. Els valors són **sempre cadenes**. Per a números o booleans cal convertir: `int(os.getenv("PORT", "8080"))`.
2. Els canvis afecten **només el procés actual i els seus fills**. No modifiquen l'entorn de la shell que ha llançat el script (no és un defecte de Python, és com funcionen els processos Unix: un fill no pot canviar l'entorn del pare).

Exemple pràctic, llegir configuració sense deixar secrets al codi:

```
db_host = os.getenv("DB_HOST", "localhost")
db_pass = os.environ["DB_PASSWORD"]   # obligatòria: falla si falta
```

Afegir un directori al `PATH` del procés:

```
os.environ["PATH"] = "/opt/eines/bin" + os.pathsep +
 ↪  os.environ["PATH"]
```

8. Executar ordres i processos

`os.system()` (només per a coses ràpides)

```
codi = os.system("ls -l")
print(codi)  # codi de sortida (a Unix, codificat: codi << 8)
```

Passa la cadena a la shell, de manera que és vulnerable a **injecció d'ordres** si hi barreges dades de l'usuari, i no permet capturar la sortida.

Alternativa recomanada: `subprocess`

```
import subprocess

r = subprocess.run(["ls", "-l", "/tmp"], capture_output=True,
 ↪  text=True, check=True)
print(r.stdout)
```

En passar una llista, no hi ha shell pel mig i no hi ha risc d'injecció.

Identificadors de procés (Unix)

```
print(os.getpid())    # PID del procés actual
print(os.getppid())   # PID del procés pare
print(os.getuid(), os.getgid())
```

`os.fork()` i `os.exec*` (Unix)

```
pid = os.fork()
if pid == 0:
    # procés fill
    os.execvp("echo", ["echo", "hola des del fill"])
else:
    os.waitpid(pid, 0)   # el pare espera el fill
    print("El fill ha acabat")
```

`fork` duplica el procés; `exec*` el substitueix per un altre programa. És el mecanisme base de com una shell llança ordres. En el codi del dia a dia, `subprocess` ho encapsula millor.

9. Fitxers temporals i aleatorietat

Per a fitxers temporals, usa `tempfile`, no `os` directament. Sí que és útil `os.urandom` per obtenir bytes criptogràficament segurs:

```
token = os.urandom(16).hex()
print(token)
```

10. Gestió d'errors

Les funcions d'`os` llancen `OSError` o alguna de les seves subclasses:

Excepció	Quan
`FileNotFoundError`	El camí no existeix
`FileExistsError`	Ja existeix i s'intenta crear
`PermissionError`	Permisos insuficients
`IsADirectoryError` / `NotADirectoryError`	Tipus d'entrada incorrecte
`OSError` (genèrica)	Directori no buit, disc ple, etc.

```
try:
    os.remove("fitxer.txt")
except FileNotFoundError:
    print("No existia")
except PermissionError:
    print("No tens permís")
except OSError as e:
    print(f"Error: {e.strerror} (errno {e.errno})")
```

11. os i `pathlib`: quina fer servir?

Des de Python 3.4, `pathlib` ofereix una interfície orientada a objectes per als camins. En codi nou sol ser més llegible:

Tasca	`os` / `os.path`	`pathlib`
Unir camins	`os.path.join(a, b)`	`Path(a) / b`
Existeix?	`os.path.exists(p)`	`Path(p).exists()`
Nom de fitxer	`os.path.basename(p)`	`Path(p).name`
Extensió	`os.path.splitext(p)[1]`	`Path(p).suffix`
Llistar	`os.listdir(p)`	`Path(p).iterdir()`
Crear jerarquia	`os.makedirs(p, exist_ok=True)`	`Path(p).mkdir(parents=True, exist_ok=True)`
Llegir text	`open(p).read()`	`Path(p).read_text()`

`os` continua sent imprescindible per a variables d'entorn, processos, permisos de baix nivell, `os.walk` i compatibilitat amb codi existent. Moltes funcions d'`os` accepten objectes `Path` directament (protocol `os.PathLike`).

12. Exemples complets

12.1. Mida total d'un directori

```
import os

def mida_directori(ruta):
    total = 0
    for arrel, _, fitxers in os.walk(ruta):
        for f in fitxers:
            cami = os.path.join(arrel, f)
            if not os.path.islink(cami):
                total += os.path.getsize(cami)
    return total

mida = mida_directori(".")
print(f"{mida / 1024 / 1024:.2f} MiB")
```

12.2. Cercar fitxers per extensió

```
import os

def cerca(ruta, extensio):
    for arrel, _, fitxers in os.walk(ruta):
        for f in fitxers:
            if f.lower().endswith(extensio):
                yield os.path.join(arrel, f)

for fitxer in cerca(".", ".md"):
    print(fitxer)
```

12.3. Reanomenar fitxers en lot

```
import os

carpeta = "fotos"
for i, nom in enumerate(sorted(os.listdir(carpeta)), start=1):
    base, ext = os.path.splitext(nom)
    nou = f"vacances_{i:03d}{ext}"
    os.rename(os.path.join(carpeta, nom), os.path.join(carpeta, nou))
```

12.4. Escriptura atòmica d'un fitxer

```
import os

def escriu_atomic(cami, contingut):
    tmp = cami + ".tmp"
    with open(tmp, "w") as f:
        f.write(contingut)
        f.flush()
        os.fsync(f.fileno())   # força l'escriptura a disc
    os.replace(tmp, cami)      # canvi atòmic
```

13. Exercicis proposats

1. Escriu un script que mostri el directori actual, l'usuari i el nombre de CPU.
2. Fes una funció que, donat un directori, retorni les 5 entrades més grans (nom i mida).
3. Crea l'estructura `projecte/{src,docs,tests}` amb `os.makedirs`, de manera que es pugui executar diverses vegades sense errors.
4. Escriu un programa que llegeixi `NOM_USUARI` de l'entorn i, si no existeix, usi `os.getlogin()`.
5. Recorre `/var/log` amb `os.walk` i mostra els fitxers modificats en les últimes 24 hores.
6. Reescriu l'exercici 3 amb `pathlib` i compara'n la llegibilitat.

14. Resum

Vull…	Funció
Saber el directori actual	`os.getcwd()`
Canviar de directori	`os.chdir(p)`
Llistar un directori	`os.scandir(p)` / `os.listdir(p)`
Recórrer un arbre	`os.walk(p)`
Crear directoris	`os.mkdir(p)` / `os.makedirs(p, exist_ok=True)`
Eliminar fitxer / directori buit	`os.remove(p)` / `os.rmdir(p)`
Reanomenar / moure	`os.rename(a, b)` / `os.replace(a, b)`
Metadades i permisos	`os.stat(p)` / `os.chmod(p, mode)`
Unir camins	`os.path.join(...)`
Variables d'entorn	`os.environ`, `os.getenv()`
PID del procés	`os.getpid()`

15. Referències

- Documentació oficial: https://docs.python.org/es/3.15/library/os.html
- `os.path`: https://docs.python.org/es/3.15/library/os.path.html
- `pathlib`: https://docs.python.org/es/3.15/library/pathlib.html
- `subprocess`: https://docs.python.org/es/3.15/library/subprocess.html

Versions d'aquest document

- HTML
- PDF
- ODT
- MD

Domini Públic (CC0)