

🐍 Python Cheatsheet – Managing Data and Processes

Curso: Google IT Automation with Python

Módulo: Managing Data and Processes with Python

Tema: Manejo de datos, subprocessos y automatización de procesos del sistema

[Introducción]

Python permite interactuar con el sistema operativo, ejecutar comandos, procesar datos y automatizar tareas de manera sencilla mediante módulos como:

- `os` → operaciones sobre archivos, directorios y entorno del sistema
- `sys` → argumentos de línea de comando y manejo de la ejecución del script
- `subprocess` → ejecutar comandos externos y capturar su salida
- `shutil` → copiar, mover y eliminar archivos/directorios

[Working with the OS – Módulo os]

Obtener información del sistema

```
import os

print(os.name)      # nombre del sistema: 'posix', 'nt', etc.
print(os.getcwd())  # directorio actual
print(os.listdir()) # listar archivos del directorio actual
```

Crear, renombrar y eliminar directorios

```
os.mkdir("nueva_carpeta")
os.makedirs("carpeta1/carpeta2") # crea jerarquía
os.rename("vieja_carpeta", "nueva_carpeta")
os.rmdir("nueva_carpeta")       # solo si está vacía
```

Variables de entorno

```
print(os.environ.get("HOME"))      # obtiene la variable de entorno HOME
os.environ["MY_VAR"] = "123"       # asignar una variable de entorno
```

[System Arguments – Módulo sys]

- `sys.argv` → lista con los argumentos de línea de comando:

```
import sys

# python script.py arg1 arg2
print(sys.argv) # ['script.py', 'arg1', 'arg2']

# Acceder al primer argumento
arg1 = sys.argv[1]
```

- `sys.exit()` → termina la ejecución del script:

```
import sys

if len(sys.argv) < 2:
    print("Faltan argumentos")
    sys.exit(1) # código de error 1
```

⌚ Running Commands – Módulo `subprocess`

- Permite ejecutar comandos externos desde Python y capturar su salida.

```
import subprocess

# Ejecutar un comando y obtener salida
result = subprocess.run(["ls", "-l"], capture_output=True, text=True)
print(result.stdout)
```

Ejecutar y manejar errores

```
result = subprocess.run(
    ["cat", "archivo.txt"],
    capture_output=True,
    text=True
)

if result.returncode == 0:
    print("Éxito:", result.stdout)
else:
    print("Error:", result.stderr)
```

Comandos en shell

```
subprocess.run("echo Hola Mundo", shell=True)
```

 Usar `shell=True` con cuidado por riesgos de seguridad.

File and Directory Operations – Módulo `shutil`

Función	Descripción	Ejemplo
<code>shutil.copy(src, dst)</code>	Copiar archivo	<code>shutil.copy("a.txt", "b.txt")</code>
<code>shutil.move(src, dst)</code>	Mover archivo/directorio	<code>shutil.move("a.txt", "backup/")</code>
<code>shutil.rmtree(path)</code>	Borrar directorio completo	<code>shutil.rmtree("folder")</code>

Reading and Writing Files (Recap)

- Abrir, leer y escribir archivos con `open()` y `with open()`.
- Procesar datos línea por línea.
- Usar `csv` y `DictReader/DictWriter` para datos tabulares.

```
import csv

with open("data.csv") as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row["name"], row["age"])
```

◆ Good Practices

- Usar `with open()` para archivos → cierra automáticamente.
- Usar `subprocess.run()` con `capture_output=True, text=True` para manejar resultados.
- Validar argumentos con `sys.argv` y usar `sys.exit()` para códigos de error claros.
- Para operaciones complejas de archivos, usar `os + shutil` en conjunto.
- Manejar excepciones siempre que se interactúe con archivos o procesos externos.

Referencias oficiales

-  [Python os module](#)
-  [Python sys module](#)
-  [Python subprocess module](#)
-  [Python shutil module](#)
-  [Real Python – Working with Files and Directories](#)

