# 🐍 Python OS Module - Colab Notebook

The `os` module in Python provides a way of using operating system–dependent functionality.
It lets us interact with files, directories, environment variables, and processes.

---

## 🔷 Importing the Module

```
import os
```

## 📂 1. Current Working Directory

- `os.getcwd()` → Returns the current working directory.
- `os.chdir(path)` → Changes the current working directory.

```
print("Current Directory:", os.getcwd())

# Change directory example (use your own path in Colab/Local)
os.chdir("/content/sample_data")
print("After Change:", os.getcwd())
```

## 📂 2. Listing Files & Directories

- `os.listdir(path)` → Lists all files and directories in the given path.

```
print("Files in current directory:", os.listdir())
```

## 📂 3. Creating and Removing Directories

- `os.mkdir("folder")` → Creates a single directory.
- `os.makedirs("a/b/c")` → Creates nested directories.
- `os.rmdir("folder")` → Removes empty directory.
- `os.removedirs("a/b/c")` → Removes nested empty directories.

```
# Create directory
# os.mkdir("demo_folder")
# print("After mkdir:", os.listdir())

# Remove directory
os.rmdir("demo_folder")
print("After rmdir:", os.listdir())
```

## 📂 4. Working with Files

- `os.remove("file.txt")` → Deletes a file.
- `os.rename("old", "new")` → Renames file/directory.

```
# Create file
with open("sample.txt", "w") as f:
    f.write("Hello OS Module!")

print("Before rename:", os.listdir())

# Rename file
os.rename("sample.txt", "renamed.txt")
print("After rename:", os.listdir())

# Remove file
os.remove("renamed.txt")
print("After remove:", os.listdir())
```

## 🌍 5. Environment Variables & System Info

- `os.name` → OS type (`posix` for Linux/Mac, `nt` for Windows).
- `os.environ` → Access environment variables.
- `os.getlogin()` → Current logged user.
- `os.getpid()` → Process ID.
- `os.cpu_count()` → Number of CPUs.

```
print("OS Name:", os.name)
print("PATH variable:", os.environ.get("PATH"))
print("Process ID:", os.getpid())
print("CPU Count:", os.cpu_count())
```

## ⚙️ 6. Path Handling (`os.path`)

- `os.path.join()` → Joins paths safely.
- `os.path.exists()` → Check if path exists.
- `os.path.isfile()` / `os.path.isdir()` → Check file/dir.
- `os.path.abspath()` → Get absolute path.
- `os.path.basename()` / `os.path.dirname()` → File/Directory name.

```
file_path = "demo.txt"

# Create file for testing
with open(file_path, "w") as f:
    f.write("Testing os.path")

print("Join path:", os.path.join("/content",(file_path)))
print("Exists:", os.path.exists(file_path))
print("Is File:", os.path.isfile(file_path))
print("Absolute Path:", os.path.abspath(file_path))
print("Base Name:", os.path.basename("/content/demo.txt"))
print("Dir Name:", os.path.dirname("/content/demo.txt"))

# Clean up
os.remove(file_path)
```

## 📘 Python `sys` Module

The `sys` module provides access to some variables and functions that interact with the **Python interpreter**.
It lets you:

- Get system info
- Handle command-line arguments
- Manage input/output streams
- Work with recursion limits
- Exit programs gracefully

## 🔷 1. sys.version & sys.version_info

Gives the Python version and build details.

```
import sys

print("Python Version:", sys.version)
print("Python Version Info:", sys.version_info)
```

```
Python Version: 3.12.11 (main, Jun  4 2025, 08:56:18) [GCC 11.4.0]
Python Version Info: sys.version_info(major=3, minor=12, micro=11, releaselevel='final', serial=0)
```

## 🔷 2. sys.path

- A list of directories Python searches for modules.

- You can also add your own paths.

```
print("System Path Directories:")
for p in sys.path:
    print(p)

# Add custom path
sys.path.append("/my/custom/path")
print("Updated Last Path:", sys.path[-1])
```

## 🔷 3. sys.argv

- Holds command-line arguments passed to the script.
- `sys.argv[0]` = script name
- `sys.argv[1:]` = actual arguments

```
print("Number of arguments:", len(sys.argv))
print("Argument List:", sys.argv[0])
print("Argument List:", sys.argv[1:])
```

## 🔷 4. sys.exit([arg])

- Terminates the program.
- `0` → Successful exit, Non-zero → Error.

```
print("Before exit")
# sys.exit(0)   # Uncomment this line to stop execution
print("This will not run if sys.exit() is called")
```

## 🔷 5. sys.stdin, sys.stdout, sys.stderr

- Standard input, output, and error streams.

```
# Writing to stdout
sys.stdout.write("This is written using sys.stdout\n")

# Reading from stdin (works in terminal, not always in Colab)
name = sys.stdin.readline()
print("Hello", name)
```

## 🔷 6. sys.getsizeof(object)

Returns the size of an object in **bytes**.

```
num = 1000
print("Size of integer:", sys.getsizeof(num))

text = "Python"
print("Size of string:", sys.getsizeof(text))
```

## 🔷 7. sys.modules

Dictionary of all **loaded modules**.

```
print("Total modules loaded:", (sys.modules))
print("Is 'os' module loaded?", 'os' in sys.modules)
```

## 🔷 8. sys.platform

Gives the platform/OS name.
Useful for writing OS-specific code.

```
print("Platform:", sys.platform)
```

## 🔷 9. sys.maxsize

Maximum size of an integer variable.

```
print("Max size of int:", sys.maxsize)
```

## 🔷 10. sys.getrecursionlimit() & sys.setrecursionlimit(n)

- Gets/sets the maximum recursion depth.

```
print("Current recursion limit:", sys.getrecursionlimit())
sys.setrecursionlimit(2000)
print("Updated recursion limit:", sys.getrecursionlimit())
```

```
Current recursion limit: 1000
Updated recursion limit: 2000
```