The datetime, os & sys Modules

AiCE Software Engineering Foundations — Python Programming (Intermediate)

Real applications need to know what time it is, work with files on disk, and sometimes talk to the Python interpreter itself. This lesson covers three more standard library modules that make that possible: datetime, os, and sys.

1. The datetime Module

The datetime module handles dates and times — one of the most common requirements in real applications. Registration timestamps, payment records, report generation dates, session expiry times all need datetime.

```
import datetime

# Get today's date
today = datetime.date.today()
print(f"Today's date: {today}")

# Get the current date and time
now = datetime.datetime.now()
print(f"Current time: {now}")

# Access individual components
print(f"Year: {now.year}, Month: {now.month}, Day: {now.day}")

# Format dates as readable strings
formatted = now.strftime("%d %B %Y, %H:%M")
print(f"Formatted: {formatted}")

# Create a specific date
aice_founded = datetime.date(2025, 1, 1)

# Calculate the difference between two dates
days_since = (today - aice_founded).days
print(f"Days since AiCE was founded: {days_since}")
```

```
# Add days to a date
deadline = today + datetime.timedelta(days=30)
print(f"Deadline (30 days): {deadline}")
```

Key concepts:

datetime.date.today() — returns today's date (year, month, day only).

datetime.datetime.now() — returns the current date and time.

strftime() — formats a datetime object as a readable string using standard format codes.

datetime.timedelta(days=n) — a duration of n days; adding it to a date gives a future date.

Subtracting two dates gives a timedelta object — use .days to get the number of days as an integer.

Common strftime Format Codes

Code	Meaning	Example Output
%d	Day of month (zero-padded)	07
%m	Month as number (zero-padded)	06
%Y	Year (4 digits)	2026
%B	Full month name	June
%A	Full weekday name	Tuesday
%H	Hour (24-hour clock)	14
%M	Minutes	35
%S	Seconds	09
%p	AM or PM	PM

2. The os Module

The os module lets your Python program interact with the operating system — navigating directories, listing files, creating folders, and checking paths. Everything you learned to

do in the terminal in the Developer Environment section, the os module lets you do from inside a Python program.

```
import os

# Get the current working directory
current_dir = os.getcwd()
print(f"Current directory: {current_dir}")

# List files and folders in the current directory
contents = os.listdir(current_dir)
print(f"Contents ({len(contents)} items): {contents}")

# Check if a path exists
path_to_check = "/home"
print(f"Exists: {os.path.exists(path_to_check)}")
print(f"Is directory: {os.path.isdir(path_to_check)}")

# Join paths safely (works on any operating system)
home = os.path.expanduser("~")
docs_path = os.path.join(home, "Documents", "aice")
print(f"Path joined: {docs_path}")

# Get just the filename or directory from a full path
full_path = "/home/adesina/projects/aice/students.py"
print(f"Filename : {os.path.basename(full_path)}")
print(f"Directory: {os.path.dirname(full_path)}")
```

The os module is especially important in the file handling unit ahead, where you'll use it to check whether files exist before reading them.

Creating and Removing Directories

```
import os

# Create a new directory
os.makedirs("aice_output", exist_ok=True)
print("Directory created.")
```

```
# Check it exists
print(f"Exists: {os.path.exists("aice_output")}")
```

> **exist_ok=True Prevents Crashes on Rerun**
> *exist_ok=True stops os.makedirs() from raising an error if the directory already exists. Without it, running the code twice would crash the second time — a small detail that matters a great deal once your programs run repeatedly.*

3. The sys Module

The sys module gives your program access to the Python interpreter itself — command-line arguments, the Python version, and the ability to exit the program programmatically.

```
import sys

# Python version information
print(f"Python version: {sys.version}")
print(f"Version info  :
{sys.version_info.major}.{sys.version_info.minor}")

# The platform
print(f"Platform: {sys.platform}")

# Command-line arguments
# When run as: python3 sys_module.py arg1 arg2
# sys.argv = ['sys_module.py', 'arg1', 'arg2']
print(f"Script name: {sys.argv[0]}")
print(f"Arguments  : {sys.argv[1:]}")
```

sys.argv is a list where sys.argv[0] is always the script name and sys.argv[1:] contains any additional arguments passed when running the script. This is how command-line tools are built — the user provides arguments and the program processes them.

Key Takeaways

datetime.date.today() and datetime.datetime.now() give you the current date and time, strftime() formats them into readable strings using standard format codes, and timedelta lets you add or subtract durations. The os module bridges Python and the file system — checking paths, listing directories, and joining paths safely across operating systems. sys gives access to the interpreter itself, including command-line arguments through sys.argv. The next lesson covers selective imports, building your own reusable modules, and the guard that keeps test code from running when a file is imported rather than executed directly.