Chapter 11.2 – Finding Modules

Teacher Edition with Full Solutions and Explanations

---

1 Learning Objectives

- Explain how Python searches for a module when executing an `import`.
- Identify the directories listed in `sys.path`.
- Use the environment variable `PYTHONPATH`.
- Avoid naming conflicts with built-in modules.

2 Introduction

When you write `import mymodule`, Python searches through a list of directories to find that file. If the module is built-in (like `math` or `sys`), it's loaded immediately. Otherwise, Python looks in the directories listed in the `sys.path` variable.

The Search Order

`sys.path` is a list of directories that Python checks in this order:

1. The directory of the executing script.
2. Directories set by the environment variable `PYTHONPATH`.
3. The standard library and installed site-packages directory.

Instructor Notes – Teaching Emphasis

Students often think "Python magically knows where everything is." Show them that `sys.path` is just a list they can print, inspect, and even modify temporarily for testing.

3 Inspecting sys.path

Activity – Show Me the Search Path

Create a short file `show_path.py`:

```
import sys
print("Python searches these directories for modules:\n")
for path in sys.path:
    print("  ", path)
```

Run it using `python3 show_path.py`. Students should see something like:

```
/home/jevert/git/SwosuCsPythonExamples/CS2/Ch11
/usr/lib/python3.11
/usr/lib/python3.11/site-packages
```

Instructor Notes – Discussion

Ask: "Where do your own modules usually live in this list?" Guide them to notice that the current working directory is always first. This explains why you can import your own files easily when running from the same folder.

4 Built-in vs. Custom Modules

A **built-in module** ships with Python. Examples: `sys`, `math`, `time`, `os`.

If you create a file named `math.py`, Python might load that one instead of the built-in version—causing chaos!

Try It – The Math Mix-up

Step 1: Create a file named `math.py`:

```
def area():
    print("I am NOT the real math module!")
```

Step 2: Create another file, `test_math.py`:

```
import math
print(dir(math))
```

Expected Output (partial):

```
['__builtins__', '__cached__', '__doc__', 'area']
```

What went wrong? Your fake file replaced Python's `math` module! Delete or rename it to fix the problem.

Instructor Notes – Solution Discussion

Emphasize that name collisions are one of the top beginner bugs. Encourage naming conventions like `mymath.py` or `utils_math.py`. You can demonstrate that after deleting `math.py`, re-importing shows real math functions again.

5 Using PYTHONPATH

`PYTHONPATH` is an environment variable that adds extra directories to the module search list. For example:

```
export PYTHONPATH="/home/student/shared_modules:/opt/utils"
```

or on Windows:

```
set PYTHONPATH="C:\shared_modules;D:\more_libs"
```

Try It – Custom Module Path

Step 1: Make a folder called `mytools` with a file `helper.py`:

```
def announce():
    print("Imported from mytools/helper.py")
```

Step 2: Add that folder to your path temporarily inside Python:

```
import sys
sys.path.append("/home/student/mytools")

import helper
helper.announce()
```

Expected Output:

```
Imported from mytools/helper.py
```

Instructor Notes – Instructor Commentary

Students see immediate cause-and-effect by modifying `sys.path`. Clarify that this change is temporary—only for the current session. Setting `PYTHONPATH` in the shell is a more permanent solution.

6 Quiz – Check for Understanding

Participation Check

1. When an import statement executes, Python first checks the current directory for the module. **Answer: True.**
2. The environment variable `PYTHONPATH` can specify extra directories for

Python to search. **Answer: True.**

3. Naming your own module `math.py` is a good idea. **Answer: False.**

Instructor Notes – Pedagogical Note

Encourage quick pair-discussion here: "Why does #3 break things?" This short chat cements how import order and file names interact.

7 Challenge Task – Module Locator Tool

Mini-Project

Write a script `module_finder.py` that:

1. Takes a module name as input.
2. Attempts to import it using `importlib`.
3. Prints the location of the module file.

Starter Code:

```
import importlib.util

name = input("Enter module name: ")
spec = importlib.util.find_spec(name)

if spec and spec.origin:
    print(f"Found at: {spec.origin}")
else:
    print("Module not found!")
```

Example Output:

```
Enter module name: math
Found at: built-in
```

Instructor Notes – Solution Notes

This is an excellent tie-in to system introspection and debugging. Let advanced students extend it to show whether the module is built-in, from a package, or local. This can lead naturally into Section 11.3 on importing specific names.

8 Reflection – Why This Matters

Takeaways

- Python's import system is predictable and inspectable.
- Knowing `sys.path` and `PYTHONPATH` prevents "module not found" headaches.
- Avoid naming collisions with built-ins.

Instructor Notes – Wrap-Up Discussion

Ask students to describe one "real-life debugging moment" where an import failed. Then, have them use the new knowledge to explain how they'd diagnose it.