Overview

In this lesson, you will learn how to use **multiple Python files together** in a single program. You will learn how to organize code using helper files, different ways to import code, how naming works when importing, and why importing files can sometimes cause unexpected behavior.

Important Information

As programs grow larger, keeping all code in one file becomes difficult to manage. A common solution is to split code across multiple Python files.

Each Python file is called a **module**.

By separating code into different files (or **modules**), it becomes easier to group related code together. You may also find yourself reusing the same kinds of functions over and over again. By storing these "helper" functions in a module, you can import them into any other program you need. Separating code into different modules also makes debugging easier. If you know what function or module has a bug, you only need to focus on fixing that area.

Helper Files

A helper file is a Python file that contains functions meant to be used by another file.

For example:

- One file might handle calculations
- Another file might handle text output
- The main file connects everything together

Helper files usually contain **function definitions only**, not code that runs automatically.

Importing an Entire File

You can import an entire Python file using the `import` keyword.

Example file structure:

- `helpers.py`
- `main.py`

`helpers.py`

```
def say_hello():
    print("Hello from helpers!")
```

`main.py`

```
import helpers

helpers.say_hello()
```

When you import the whole file you must use the file name as a prefix (in the above example `helpers` is used a prefix for the `say_hello()` function). This prefix is called a namespace, and it helps avoid name conflicts. If you had a `say_hello()` function defined in both your main program and your helper module, prefixing the one from the module with `helpers` clarifies which function should be used to the interpreter.

Importing a File With an Alias

Sometimes file names are long or awkward to type. Python allows you to rename a module when importing it using `as`.

```
import helpers as h

h.say_hello()
```

This shortens the namespace so you can just write `h` instead of `helpers` everytime, while maintaining the protection of using different namespaces. Aliases are especiall useful when module names are long or multiple modules have similar names.

Code Runs on Import

A very important rule in Python:

When a file is imported, all top-level code in that file runs immediately.

Example:

`helpers.py`

```
print("Helpers file is running")

def add(a, b):
    return a + b
```

`main.py`

```
import helpers

print("Main file is running")
```

Output:

```
Helpers file is running
Main file is running
```

Even though `add()` was never called, the `print()` statement ran because it was not inside a function.

For now, the safest rule is helper modules should only (or mostly) contain function definitions. This avoids code that runs automatically at the top level when a module is imported.

Importing Specific Functions

Instead of importing an entire file, you can import only the functions you need.

```
from helpers import add
```

Now you can call the function directly:

```
result = add(3, 4)
print(result)
```

Notice that no file name prefix is needed. This is because all of the functions you import like this are placed directly into your program's namespace rather than being protected in their own namespace.

Importing Multiple Specific Functions

You can import more than one function at a time.

```
from helpers import add, subtract
```

Calling the functions:

```
total = add(5, 2)
difference = subtract(5, 2)
print(total)
print(difference)
```

This is useful when you only need a handful of functions or want to avoid retyping namespaces over and over again.

Importing Everything With `*`

Python also allows importing **everything** from a file.

```
from helpers import *
```

When you import `*` all functions and variables from the module you import are available. In the above example, everything from `helpers.py` is added to the program's namespace. This allows all functions from `helpers.py` to be used directly without a prefix.

Example:

`helpers.py`

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

`main.py`

```
from helpers import *

print(add(4, 2))
print(subtract(4, 2))
```

Use With Caution

While `import *` can be convenient, it has drawbacks. Importing everything makes it unclear where functions come from which can make debugging more difficult. It also increases the likelyhood of naming conflicts (like accidentally redefining a function by using the same name in your main program). Code can also be more difficult to read in terms of identifying where functions are defined because imported functions aren't labeled with a prefix anymore.

As a general rule of thumb, `import *` should only be used in **small, controlled programs**. Whenever possible, it's preferred that you import a handful of specific functions instead.

Choosing How to Import

Import Syntax	Guidelines
`import helpers`	Clear structure and namespaces. Requires the full module name as a prefix.
`import helpers as h`	Same clarity and namespace rules as importing the module, but allows shorter or custom prefixes.
`from helpers import add`	Allows you to pick and choose specific functions to import. Best when using a few functions. Does not require a prefix on the functions when called.
`from helpers import *`	Imports all functions from the module and uses the main program's namespace. Convenient for smaller projects, but has the risk of muddling the namespace by not using prefixes on functions. Use sparingly and carefully.

Set Up

Create two Python files:

- `helpers.py`
- `main.py`

Copy, Change, Challenge - Importing Helpers

Copy

`helpers.py`

```
def multiply(a, b):
    return a * b
```

`main.py`

```
import helpers as h

result = h.multiply(4, 5)
print(result)
```

Change

Rename the alias `h` to something else and update the function call.

Challenge

Add two more functions to `helpers.py` and call all of them using the alias.

Copy, Change, Challenge - Importing Functions Directly

Copy

`helpers.py`

```
def greet():
    print("Hello!")

print("Helpers loaded")
```

`main.py`

```
from helpers import greet

greet()
```

Run `main.py` and observe what prints.

Change

Move the `print("Helpers loaded")` line into a function so it no longer runs automatically.

Challenge

Rewrite the program using `from helpers import *` and call at least two functions directly.