UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, JALOZAI CAMPUS

Lab 10: Modules and Packages

Lab Title: EE-271 "OOP & Data Structures Lab"

i. Time: 10 min/ Task

Lab report task:

- Open a script and save.

  # __init__.py
- Make add.py and save in the same folder.
- Make sub.py and save in the same folder.
- Make mul.py and save in the same folder.
- Make divide.py and save in the same folder.

a. Import and run different module (at least 3).

Lab work tasks:

1. Creating Modules

```
def add(x, y):
    return x + y
```

a. Save the file as adder.py in a new directory called myproject/ somewhere on your computer.

  a. Use this module as given below.

```
import adder
value = adder.add(2, 2)
print(value)
```

  b. Add another function to double a given value.

```
def double(x):
  return x + x
```

  - Run the following code.

```
import adder
value = adder.add(2, 2)
double_value = adder.double(value)
print(double_value)
```

c. `import <module> as <other_name>`

You can change the name of an import using the as keyword:

- When you import a module this way, the module's namespace is accessed through <other_name> instead of <module>.

  i. Run the following code and note what you observe and why this happened.

```
import adder as a # <-- Change this line
# Leave the code below unchanged
value = adder.add(2, 2)
double_value = adder.double(value)
print(double_value)
```

ii. Run the following code and note what you observe and why this happened.

```
import adder as a
value = a.add(2, 2) # <-- Change this line
double_value = a.double(value) # <-- Change this line, too
print(double_value)
```

d. `from <module> import <name>`

Instead of importing the entire namespace, you can import only a specific name from a module.

- Run the following code.

```
from adder import add # <-- Change this line
value = adder.add(2, 2)
print(value)
```

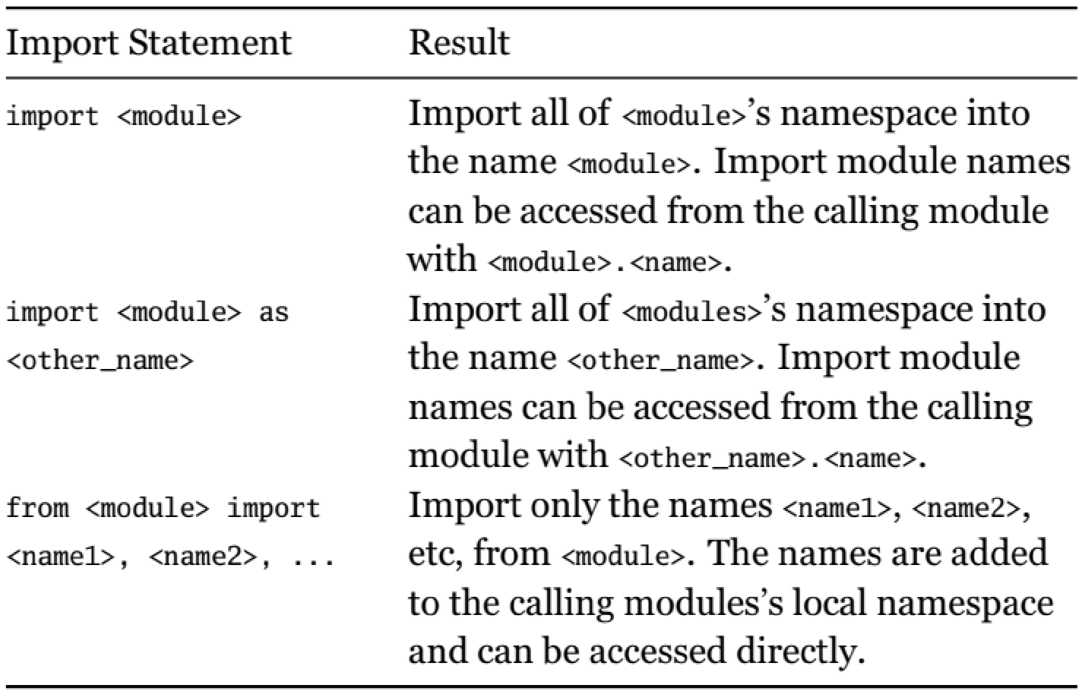
- Run the following code and note what you observe and why this happened.

```
from adder import add # <-- Change this line
value = adder.add(2, 2)
double_value = adder.double(2, 2)
print(double_value)
```

- Run the following code and note what you observe and why this happened.

```
from adder import add, double # <-- Change this line
# Leave the code below unchanged
value = add(2, 2)
double_value = double(value)
print(double_value)
```

Note:

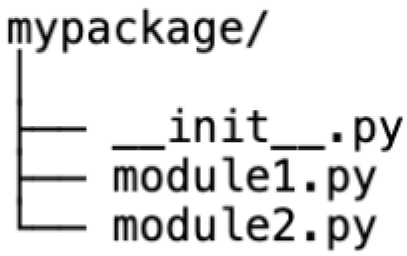


Import Statement	Result
`import <module>`	Import all of `<module>`’s namespace into the name `<module>`. Import module names can be accessed from the calling module with `<module>.<name>`.
`import <module> as <other_name>`	Import all of `<modules>`’s namespace into the name `<other_name>`. Import module names can be accessed from the calling module with `<other_name>.<name>`.
`from <module> import <name1>, <name2>, ...`	Import only the names `<name1>`, `<name2>`, etc, from `<module>`. The names are added to the calling modules’s local namespace and can be accessed directly.

2. Master the above different ways of importing.
   Sometimes, modules contain a single function or class that has the same name as the module. For example, there is a module in the Python standard library called datetime that contains a class called datetime.
   a. Run the following code and understand how different ways of importing are important.
   ```
   import datetime
   datetime.datetime(2020, 2, 2)
   ```
   b. Run the following code and understand how different ways of importing are important.
   ```
   import datetime as dt
   dt.datetime(2020, 2, 2)
   ```
   c. Run the following code and understand how different ways of importing are important.
   ```
   from datetime import datetime
   datetime(2020, 2, 2)
   ```

3. Creating Packages

A **package** is a folder that contains one or more Python modules. It must also contain a special module called _init_.py. Here is an example of a package so that you can see this structure:



Note: The _init_.py module doesn't need to contain any code!

- Open a script and save.

  #__init__.py

- Make module1.py and save in the same folder.

```
# module1.py
def greet(name):
  print(f"Hello, {name}!")
```

- Make module2.py and save in the same folder.

```
# module2.py
def depart(name):
  print(f"Goodbye, {name}!")
```

b. Run the following code and observe what happens and why.

```
import mypackage
mypackage.module1.greet("Pythonista")
mypackage.module2.depart("Pythonista")
```

c. Run the following code and observe what happens and why.

```
import mypackage.module1 # <-- Change this line
# Leave the below code unchanged
mypackage.module1.greet("Pythonista")
mypackage.module2.depart("Pythonista")
```

d. Run the following code and observe what happens and why.

```
import mypackage.module1
import mypackage.module2 # <-- Add this line
# Leave the below code unchanged
mypackage.module1.greet("Pythonista")
mypackage.module2.depart("Pythonista")
```

Note:

Import Statement Variations For Packages

There are three variations of the `import` statement that you learned for importing names from modules. These three variations translate to the following four variations for importing modules from packages:

1. `import <package>`
2. `import <package> as <other_name>`
3. `from <package> import <module>`
4. `from <package> import <module> as <other_name>`

e. Run the following code and observe what happens and why.

```
# main.py
from mypackage import module1, module2
module1.greet("Pythonista")
module2.depart("Pythonista")
```

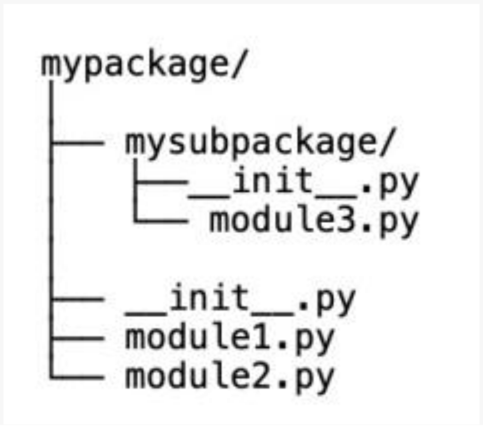
f. Run the following code and observe what happens and why.

```
from mypackage import module1 as m1, module2 as m2
m1.greet("Pythonista")
m2.depart("Pythonista")
```

g. Run the following code and observe what happens and why.

```
from mypackage.module1 import greet
from mypackage.module2 import depart
greet("Pythonista")
depart("Pythonista")
```

Note: A package nested inside of another package is called a **subpackage**.



```
mypackage/
│
├── mysubpackage/
│   ├──__init__.py
│   └── module3.py
│
├── __init__.py
├── module1.py
└── module2.py
```

Recommended reading:

1. **Chapter 11**: Python Basics: A Practical Introduction to Python 3 By Real Python