Module 3:

Modules and Packages

By Ninad Gaikwad

Reference - “python.org”

Modules

- Same as a code library
- File containing a set of functions you want to include in your application
- To create a module just save the code with the file extension .py
- Creating Module: Eg. # mymodule.py

```
def greeting(name):
    print("Hello, " + name)
```

- Using module: Eg. # Mainfile.py

```
import mymodule
mymodule.greeting("Jonathan")
```

Modules

- Importing variables:

  Eg. # mymodule.py

  person1 = { "name": "John", "age": 36, "country": "Norway" }

- # Mainfile.py

  import mymodule

  a = mymodule.person1["age"]

  print(a)

- Re-naming a Module:

  import mymodule as mx

Built-in Modules

- Import and use the platform module:

  Eg. import platform

  x = platform.system()

  print(x)

- Using the dir() Function: built-in function to list all the function names (or variable names) in a module

  Eg. import platform

  x = dir(platform)

  print(x)

Import From Module

- You can choose to import only parts from a module, by using the from keyword
- Eg. # mymodule.py

```
def greeting(name):

  print("Hello, " + name)

person1 = {  "name": "John",  "age": 36,  "country": "Norway" }
```

- Import only the person1 dictionary from the module:

  Eg. # Mainfile.py

```
from mymodule import person1

print (person1["age"])
```

- When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], not mymodule.person1["age"]

Packages

- Packages are a way of structuring Python's module namespace by using "dotted module names"
- For example, the module name A.B designates a submodule named B in a package named A.
- The __init__.py files are required to make Python treat directories containing the file as packages
- __init__.py can just be an empty file, but it can also execute initialization code for the package

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Importing modules from packages

- Users of the package can import individual modules from the package, for example:

  ```
  import sound.effects.echo
  sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
  ```

- An alternative way of importing the submodule is

  ```
  from sound.effects import echo
  echo.echofilter(input, output, delay=0.7, atten=4)
  ```

- Another variation is to import the desired function or variable directly

  ```
  from sound.effects.echo import echofilter
  echofilter(input, output, delay=0.7, atten=4)
  ```

Importing from package

- Importing * From a Package
  - The import statement uses the following convention: if a package's __init__.py code defines a list named __all__, it is taken to be the list of module names that should be imported when from package import * is encountered.
  - Eg. The file sound/effects/__init__.py could contain the following code: __all__ = ["echo", "surround", "reverse"]
- Intra-package References
  - From the surround module for example, you might use:
    - from . import echo
    - from .. import formats
    - from ..filters import equalizer
- Packages in Multiple Directories
  - Packages support one more special attribute, __path__ , initialized to be a list containing the name of the directory holding the package's __init__.py before the code in that file is executed.
  - This variable can be modified; doing so affects future searches for modules and subpackages contained in the package.

Experiment 9:

Create a package and module for demonstrating scientific calculator with method overloading