

Modules, Packages and Programs

Vivek K. S., Deepak G.

Information Systems Decision Sciences (ISDS)
MUMA College of Business
University of South Florida
Tampa, Florida

2017

Objectives

- To learn about packages, modules and programs.
- To understand the reusability of Python Packages and how they could be imported into another program.
- To learn how to create packages and modules.
- To learn how to create standalone programs and run them from the interactive prompt.
- To understand how to move away from interactive programming to creating standalone programs with function calls, namespaces and deriving the the desired output.

First Standalone Program

- Create a file called ex1.py on your computer. Open it in edit mode.
- Write the following piece of code.

```
print(5+5)
```

- The line should print the sum 10 on the screen when it is executed.
- To run the file, we'll use the Command prompt (terminal for Mac users).
- At the prompt, we'll say python ex1.py and hit return.
- The number "10" should be printed on the screen.

Command Line Arguments

- Arguments can be passed to our program directly from the command line as well.
- Lets create a second file called ex2.py and write the following line of code in it.

```
import sys
print('Program arguments : ', sys.argv)
print(type(sys.argv))
```

- Execute the code from the command prompt as "python ex2.py".
- You will notice that the name of the python file gets printed on the screen. It is because, the .py file is an argument in itself.
- Passing a couple of more arguments, we can understand that the arguments are internally converted into a list and the first argument to the program is always the file name itself.

Importing Modules

- A module is Python code file that can be reused.
- Modules are reused by simply importing them into the current program. Example:

dice.py

```
def throw_dice():  
    """Roll your dice like a player!"""  
    from random import choice  
    possibilities = [1,2,3,4,5,6]  
    return choice(possibilities)
```

ex3.py

```
import dice  
side = dice.throw_dice()  
print("You have rolled a ",side)
```

Explanation

- In the example, we see two types of import statements.
- In `dice.py`, we import only the `choice` function from the `random` module. This is usually a best practice as it saves on memory and avoids unnecessary code imports.
- The `random` module is a fully standalone python code with multiple functions and *choice* is one amongst them.
- In the `ex3.py` file, we import the `dice.py` module and using `dice` as a handle, we call the `throw_dice()` function of it.
- Remember that `throw_dice()` method is a property of the `dice` object (In python, everything is treated as an object).
- The dot notation is used to call the property as seen in `dice.throw_dice()`.
- We are basically qualifying the content of the `dice` module with the module's name itself.

Using Aliases for Module Names

- Modules can be imported with an alias name as well.
- The new alias name can then be used to access any property of the imported module and this results in ease of typing.

Example,

```
import numpy as np
arr = np.array([1,2,3,4,5])
print(arr.shape)
```

- In the code, we assigned an alias name to the numpy module as 'np'.
- It is later used to call the array function of the module as "np.array()".

Packages

- Modules can be further organized into file hierarchies called Packages.
- Maybe in the previous exercise, we may want to roll one die and sometimes roll two dice.
- One way to structure this is to have two separate modules, one for a single die and another for a pair of dice and each of them will have a function called roll.
- It will look something like this.

```
# single.py
def roll_dice():
    """Roll your dice like a player!"""
    from random import choice
    possibilities = [1,2,3,4,5,6]
    return choice(possibilities)
```


Packages Continued

```
#double.py
def roll_dice():
    """Roll a pair of dice"""
    from random import choice
    possibilites = [1,2,3,4,5,6]
    return(choice([(one, two)
                    for one in possibilites
                    for two in possibilites])))
```

We'll add another python file called `__init__.py` that contains the following code.

```
#__init__.py
from roll import single
from roll import double
```

Packages

```
#ex5.py
```

```
from roll import single,double
```

```
side = single.roll_dice()
```

```
print("You have rolled a {}".format(side))
```

```
sides = double.roll_dice()
```

```
print("You have rolled {} and {}".  
format(sides[0],sides[1]))
```

When the above code is executed, the modules "single" and "double" will be imported from the package "roll".

Explanation

- The way the package is created is by placing all our modules in a single directory that has been named with the desired Package's name.
- The next step is to place a python file called `__init__.py` for Python to identify the directory as a Package.
- Once that is done, we are free to import the entire package or just the select modules from the package into our program using *from* as follows *from roll import single*.
- The methods of the module can be called by the module name or the name(alias) assigned to the module as part of the import.

Complete Coding Exercise

Complete coding exercise is available at <https://github.com/vivek14632/Python-Workshop/tree/master/Introducing%20Python/Chapter%205/Code%20Examples>

Summary

- We understood the use of modules in importing additional functionality into our code that gives us more firepower to work with.
- Using command line arguments help us with more control over providing inputs to the code, while debugging especially.
- Use of Packages to package all methods and classes together gives us more options to save the code for later reuse.