HackerFrogs Afterschool
Beginner Cybersecurity Skills

Intro to Python Programming with Learnpython.org: Part 4

Class:	Programming (Python)
Workshop Number:	AS-PRO-PY-04
Document Version:	1.0
Special Requirements:	Completion of AS-PRO-PY-03

Table of Contents

Introduction

Welcome to HackerFrogs Afterschool! HackerFrogs Afterschool is a a series of online workshops meant to teach cybersecurity skills and concepts to beginners who may not be familiar with the field of cybersecurity. This lesson is the fourth and final lesson on python programming, which is important for understanding other cybersecurity skills. This lesson will cover the following learning objectives:

Learning Objectives:

- Learn about Python dictionaries
- Learn about Python modules and packages
- Learn about importing modules and packages into current code

Part 1 – Python Dictionaries

In Python, **dictionaries** are a type of collection object, like **lists** or **tuples**. The biggest difference between dictionaries and other collection objects is that dictionary entries come in pairs that consist of a **key** and a **value**. For example:

```
country_capital = {'England': 'London', 'Canada': 'Ottawa'}
```

Dictionaries can be identified by curly braces, within which the dictionary entries are contained. In the example above, we see that the first piece of data in each entry is key ('England' and 'Canada'), and the second piece of data in each entry is the value associated with the key ('London' and 'Ottawa'). The key and value in each entry is separated by a colon, and each entry is separated by a comma, similar to other collections, such as lists and tuples.

The format of dictionaries make them very useful for storing data where each piece of data (the value) must be labelled with a name (the key). A couple of use cases might be literal dictionaries, where a word (the key) is matched with its definition, or a phone book, where each phone number (the value) is matched with a name (the key).

Adding Entries to Dictionaries

Adding entries to dictionaries can be done at any time after the dictionary has been instantiated. For example, if we wanted to add an entry to the **country_capital** dictionary we saw earlier, we would use the following code:

```
country_capital['Japan'] = 'Tokyo'
```

So we see from the example, to add the entry, we use the name of the dictionary, then inside a pair of square brackets we specify the name of the **key** of the entry pair, then use the equals symbol, followed by the **value** of the entry pair.

Iterating Over Dictionaries

Iterating over the entries in dictionaries cannot be done in the same way it would be done when iterating over a typical list in Python. Instead, we must use the **items** dictionary method to render the dictionary entries as a series of tuples inside of a list. See the following example:

```
print(country_capital.items())
```

Results in the following output:

```
dict_items([('England', 'London'), ('Canada', 'Ottawa')])
```

As we can see, the **items** method has rendered the dictionary into a series of tuples inside of a list. So in order to iterate through entries in a dictionary we must use the following method:

```
for i, j in country_capital.items():
    print(i, j)
```

Which produces the following result:

```
England London
Canada Ottawa
```

Removing Values In Dictionaries

Suppose we're still using the same dictionary from the previous section. If we wanted to remove the **England: London** entry from the dictionary, we could use either of the following two ways:

```
del country_capitals['England']
```

or

```
country_capitals.pop('England')
```

The first way of deleting dictionary entries uses the **del** keyword, which is used to delete objects, and the second way uses the dictionary **pop** method, which deletes the specified entry.

Now that we've discussed a few aspects about dictionaries, let's learn more from the learnpython.org website.

Step 1

Using a web browser, navigate to the following webpage:

https://www.learnpython.org/en/Dictionaries

Let's review the different sections on the page, and play around with the concepts covered in those sections, but stop once we reach the **Exercise** portion of the page.

Step 2

In this **Exercise** section, we're tasked with making the following modifications to the dictionary in the interactive Python window's code:

- add the entry **'Jake': 938273443** to the dictionary
- delete the **Jill** entry from the dictionary

The following code should fulfill the above requirements.

```
phonebook['Jake'] = 938273443
phonebook.pop('Jill')
```

Then click the **Run** button. This should solve the task.

Part 2 – Modules and Packages

Namespaces

Before we start exploring what modules and packages are, first we need to talk about what a Python namespace is. Namcepsaces are collections of defined symbolic names and their corresponding values. Every Python program exists within a namespace, and modules are collections of objects that can be added to a particular namespace.

Importing modules into a Python namespace allows increased functionality and flexibility. First let's explore how modules are written.

Writing Modules

To write a Python module, we create a **.py** file that contains all of the variables and functions relevant to the module, with a file name that is identical to the desired module name. For example, if we wanted to create a module called **greetings**, we would create a file named **greetings.py**, then within that file, define variables and / or functions, like the following:

```
def default_greeting():
    print('Hello, friend! How are you?')
```

Importing Modules to the Current Namespace

To continue our example, after creating the **greetings.py** file, we could import the **greetings** module into another Python namespace with the following code:

```
import greetings
```

Then we can access the **default_greeting** function in the greetings module with in the following way:

```
greetings.default_greeting()
```

Which results in the following output:

```
Hello, friend! How are you?
```

Importing all Objects from a Module

The method of accessing the **default_greetings** function in the previous section required us to access the function while referencing the **greetings** module (greetings.default_greeting). If we want to import all objects from a module without having to reference the module name each time we use a variable or function from it, we can use the **from** keyword and the ***** operator. For example, if we wanted to import all objects from the **greetings** module, we would use the following code:

```
from greetings import *
```

This is an easy way of moving a large number of functions and variables from a module into the current namespace, however, it must be done with caution, because if too many module objects are moved into a namespace that already has a lot of objects, the potential for one or more of the objects having identical names is much higher. Therefore, it is recommended to use the next procedure to import specific objects into the namespace.

Importing Individual Objects from a Module

As opposed to importing all objects from a module, we can instead be selective about which objects we import. For example, if we wanted to only import the **default_greeting** function from the **greetings** module, we would use the following code:

```
from greetings import default_greetings
```

The object is imported into the namespace without being attached to the module, so we can use the following code to access the function:

```
default_greetings()
```

Custom Import Names

When importing modules into a namespace, we can assign them a custom name by using the **as** keyword. See the example below:

```
import greetings as gree
```

There are many reasons why we would want to assign different names to imported modules, including:

- to avoid identical module / function names in the namespace
- to shorten the names of modules for easier typing
- to better remember the purpose of the module

After importing a module using a custom name, from that point in the code, the custom name will be used in the place of the original module name. Custom names can also be used for specific objects imported from modules. For example:

```
from greetings import default_greeting as dg
```

We could then use the following code:

```
dg()
```

To receive the following output:

```
Hello, friend! How are you?
```

Module Initialization

When a module is imported into code, all objects imported from that module are initialized into the code. If objects from the same module are imported into the same code at a later point in code execution, the objects will not be initialized a second time. This means that, if for some reason, the same module objects are imported multiple times, Python will not re-initialize objects with identical names and values.

The Dir Function

The **dir** function is used to list all of the objects contained in any particular module. Simply run the **dir** function with the name of the module as the argument. For example, the following code uses the **dir** function to list the objects contained in the **random** module:

```
dir(random)
```

The output will be every object contained in the **random** module.

The Help Function

The help function will return information on any object supplied to it, and is useful for determining the use of functions or other objects inside of modules. For example, if we wanted to get the help entry for the **shuffle** function in the **random** module, we would use the following code:

```
import random
help(random.shuffle)
```

Note that we cannot access the **help** function to a module without importing it first. If we instead imported only the **shuffle** function into our namespace, then we could use the following code to receive the help text for **shuffle**:

```
from random import shuffle
help(shuffle)
```

Writing Packages

Python packages are collections of one or more modules which are tied together by a special file named **__init__.py**. The directory on the filesystem where the package files are must contain the **__init__.py** file, and the name of the directory where the package files are must be the name of the package.

Importing Packages

Importing packages is done in the exact same way modules are imported. When a package is imported, all modules in the package are imported, unless otherwise specified.

In order to import individual modules from a package, we must use the from keyword. For example:

```
from requests import cookies
```

This imports the **cookies** module from the **requests** package. For context, the **requests** package handles HTTP requests, and the **cookies** module includes many functions for managing HTTP cookies.

Now that we've discussed a few different aspects of Python modules, let's work through a few Python module-related exercises on a web-based Linux machine.

Step 1

Navigate to the following URL:

https://bellard.org/jslinux/

Once there, click on the Startup Link associated with the Fedora 33 OS with the Console User Interface. If, for some reason, that link is not available, any other Linux OS with a Console User Interface should suffice for the purposes of this workshop.

Step 2

User the Nano text editor to edit a file named **greetings.py**:

```
nano greetings.py
```

Step 3

Input the following code into the text editor:

```
def greet():
    print('Greetings, friend!')
```

NOTE: When copying or pasting to / from the web-based Linux machine, keyboard shortcuts will not work. Please use the mouse to highlight the **right-click** selection for copying or pasting.

Step 4

Save and exit out of the Nano text editor by using the **ctrl-x** keyboard shortcut, then press the **Y** key, then press the **Enter** key.

Step 5

Now's let's great a Python script which imports the **greetings** module. Use the following command:

```
nano example.py
```

Step 6

In the text editor, enter the following code:

```
import greetings

greetings.greet()
```

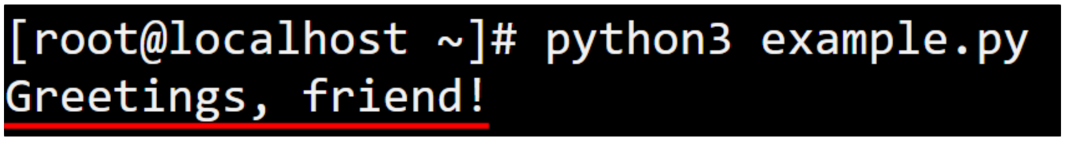
Then save and exit out of the text editor like in Step 4.

Step 7

Run the **example.py** script.

```
python3 example.py
```

The output should look like the screenshot below (expected output underlined in red):



```
[root@localhost ~]# python3 example.py
Greetings, friend!
```

We successfully imported the module and accessed the module's function with our **example.py** script.

Step 8

Next we'll import the **greet** function without importing the entire **greetings** module. Edit the **example.py** with **Nano** again and replace the code in the file with the following:

```
from greetings import greet

greet()
```

Save the file and exit.

Step 9

Run the **example.py** script in the same way we did in Step 7.

The output should be the same as the previous time we ran the script. If so, we successfully imported a single function from a module and executed that function.

Step 10

Next, we'll import the **greetings** module using a custom import name. Use Nano to edit the **example.py** file again, and replace the code in the file with the following:

```
import greetings as g

g.greet()
```

Then save the file and exit the text editor.

Step 11

Run the example.py script. The output should be the same as the previous times. If so, we have successfully imported a module using a custom import name.

Step 12

Next, we'll address the issue of module initialization. When a module is imported, all code in it is run once to initialize its contents into the namespace. Let's modify the **greetings.py** module in the following way:

Using Nano, edit the **greetings.py** file so that the last line of the file (after defining the **greet** function) contains the following:

```
greet()
```

The whole file should look like the following:

```
def greet():
    print('Greetings, friend!')
greet()
```

Save the file and exit out of the Nano text editor.

Step 13

Edit the **example.py** file with Nano, and replace the contents of the file with the following:

```
import greetings
```

Save file and exit the text editor.

Step 14

Run the **example.py** file. The output should be the same as the previous times. If so, we've just seen the results of module initialization, in that despite there being no function call to the **greet** function in the code, the **greet** function was called by merely importing the module.

Summary

In this workshop, we learned about the following Python language features:

Dictionaries

Dictionaries are a type of collection object in Python, similar to lists and tuples. They are unique in the fact that they store items as key / value pairs. They are useful for storing the same info type covering many different labels (e.g., name / phone number pairs), or different info types that all pertain to the same subject (e.g., statistics pairs all about a specific person).

Modules and Packages

Modules are a way to add outside functionality to Python code through the importing of module files into the current code. After import, any functions and methods within the imported module can be used in the current code. Python packages are collections of modules, which can also be imported into current code, either as a whole, or as individual modules.

This is the last workshop on beginner's Python in the HackerFrogs Afterschool programming course, but if you want to learn more about Python,there are a plethora of free online resources for learning about Python, including the following:

Codecademy: Learn Python 3

https://www.codecademy.com/learn/learn-python-3

NOTES: Although not a free course, Codecademy offers a free 7-day trial of their pro subscription, which includes the Learn Python 3 course.

Udacity: Introduction to Python Programming

https://www.udacity.com/course/introduction-to-python--ud1110

TryHackMe – Python Basics Room

https://tryhackme.com/room/pythonbasics

Extra Credit

The following exercises cover topics that were included in this workshop and are provided for an extra challenge:

Pynative: Python Dictionary Exercise With Solutions

https://pynative.com/python-dictionary-exercise-with-solutions/

NOTES: It may be useful to use the web-based Python code editor at the following link to solve these exercises

https://pynative.com/online-python-code-editor-to-execute-python-code/

Udacity: Introduction to Python Programming – Data Structures

https://www.udacity.com/course/introduction-to-python--ud1110

NOTES: We will need a registered user account to access this course. The target materials for this workshop's extra credit is the Data Structures section.

Automate the Boring Stuff with Python: Chapter 5 – Dictionaries and Structuring Data

https://automatetheboringstuff.com/2e/chapter5/

Until next time, HackerFrogs!