HackerFrogs Afterschool
Beginner Cybersecurity Skills

Intro to Python Programming with Learnpython.org: Part 3

Class:	**Programming (Python)**
Workshop Number:	**AS-PRO-PY-03**
Document Version:	**1.0**
Special Requirements:	**Completion of AS-PRO-PY-02**

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 third lesson for programming, which is important for understanding other cybersecurity skills, such as binary exploitation and reverse engineering. This lesson will cover the following learning objectives:

Learning Objectives:

- Learn about programming functions
- Learn about programming loops
- Learn about Python classes and objects

Part 1 – Programming Loops

Programming loops are tools that programmers use to run the same instructions multiple times against a set of data. Take the following example:

```
for i in range(4):
    print(i)
```

The output from this code would be:

```
0
1
2
3
```

The loop executes the print instruction four times, and each time the loop executed, it printed a different item from a list created by the **range** function. But why was the number 4 not printed? That's because the numbers in the list created by the range function start from the number zero, and we specified 4 numbers. The numbers 0, 1, 2, and 3 are a total of 4 numbers.

Loops in Python fall into two major types, **for loops** and **while loops**.

For Loops

For loops are instructions that repeat for each item in a list. The loop stops after the instructions have been performed on all items in the list. Any object that can be referenced by index can be used in a **for loop**, including strings. See the following example:

```
for i in 'frog':
    print(i)
```

Which results in the following output:

```
f
r
o
g
```

To write a for loop, begin with **for**, the a variable name that is only used inside the loop, typically the letter **i**, then **in**, then the name of the object o be iterated over, then a colon. On the next line, indent the line (4 spaces), then input the instructions to be run. Since a loop is a code block, all lines in the loop must be indented. All **for loops** iterate over the contents of an object, typically a list.

While Loops

The other type of loop is the **while loop**, which, unlike the **for loop**, does not have an assumed end condition. See the following example:

```
count = 0

while count <3:
    print(count)
    count += 1
```

The result of this code is the following:

```
0
1
2
```

In the above example, the loop keeps running until the **count** variable is no longer less than 3. Each iteration of the loop prints the value of **count**, then increases the value of **count** by one.

We can see that **while loops** and **if** conditionals are closely related, since each time a **while loop** iterates, it checks if its running condition is still True, and terminates the loop if the statement is False. In the example, the loop runs over and over again while the Boolean statement supplied to it remains True. Once the **count** variable is incremented to 3, on the next iteration of the loop, **count <3** returns False, and the loop ends.

To create a **while loop**, start with 'while', then the statement that must evaluate True for the loop to execute, then a colon. On the next line, indent (4 spaces), then input the loop instructions. Typically, the last loop instruction progresses a variable towards a state where the loop condition will return False.

Infinite Loops

What happens if a **while loop** is coded in such a way where the Boolean statement will never return False? The loop will never end naturally, and creates what is called an infinite loop. See the following example:

```
count = 1

while count != 0:
    print(count)
    count += 1
```

Since it is not possible for the **count** variable to hold the value that would return the **count != 0** (not equals zero) as false, this is an infinite loop.

Although there are cases where programmers intentionally create infinite loops in their programs, unintentional infinite loops effectively halt program execution, leading to system memory issues, crashes, or other problems.

The Break Statement

The **break** statement is an instruction we can insert into programming loops. When executed, the break instruction terminates execution of the current loop. The **break** statement can be used in any loop, but is most commonly associated with **while true loops**.

While True Loops

The **while true loop** is a type of **while loop** which is special in that it will never stop operating until it encounters the **break** statement in its instructions. A **while true loop** is by default, an infinite loop, but the **break** statement allows it to work within functional code. See the following example code:

```
count = 0

while True:
    print(count)
    count += 1
    if count == 4:
        break
```

This code will print out the value of the **count** variable in each iteration of the loop, but after each print instruction, there is a check to see if the value of **count** is **4**, and if it is, the **break** instruction is executed and the loop terminates.

The Continue Statement

The continue statement is an instruction that stops execution in the current loop an begins a new iteration of the loop. It differs from the break statement because it doesn't terminate the whole loop, just that one iteration of it. See the following example:

```
for i in range(5):
    if i % 2 == 1:
        continue
    print('%d is an even number' % i)
```

The above code iterates through the numbers 0-4, and if the modulo of that number and 2 is 1 (the number is odd), then the **continue** statement is executed, that iteration of the loop ends and the next iteration begins. But if the number is even, then the print instruction executes.

Conditionals Within Loops

It is quite common to find if conditionals and else clauses within loops, but we must remember to keep the proper indentation when writing the code. Take the following example:

```
for i in range(4):
    if i % 2 == 0:
        print('%d is an even number' % i)
    else:
        print('%d is an odd number' % i)
```

As we can see from the above code, everything within the **for loop** code is indented one level, and within that block, everything within the **if** code block is indented to a second level, and everything in within the **else** code block is also indented to the second level. If there were further code blocks within the **if** or **else** code blocks, those would be indented to a third level.

Now that we've discussed programming loops a bit, let's learn more from the learnpython.org website.

Step 1

Using a web browser, navigate to the following webpage:

https://learnpython.org/en/Loops

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 editing the for loop on line 12 so that the following is true:

- all of the even numbers in the **numbers** list are printed
- no numbers are printed after the number 237 appears in the list

The following code should fulfill the above requirements.

```
for number in numbers:
    if number == 237:
        break
    if number % 2 == 1:
        continue
    print(number)
```

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

Part 2 – Functions

Within programming, functions are defined blocks of code which can be used multiple times to perform the same set of instructions. If a specific set of instructions needs to be run multiple times in the same code, it is more efficient to create a function out of those instructions rather than input the same instructions again. In fact, we've been making use of the most common built-in Python function this whole time: the **print** function.

Executing a function in code is known as "calling" the function, and any arguments that the function requires to execute properly (if any) are provided inside the parentheses when calling the function.

To create a function, we start with **def**, then the name of the function with a pair of parentheses attached to the end of the name, with the names of any variables required for the function to run inside the parentheses. Following the parentheses is a colon. The next lines in the function are all indented, and contain the instructions the function performs when called. See the following example:

```
def my_greeting(name):
    print('Greetings, %s!' % name)
```

If we called this function with the argument **shyhat**, the function call would look like this:

```
my_greetings('shyhat')
```

And the output would look like the following:

```
Greetings, shyhat!
```

The Return Statement

Return is an instruction that can be used in functions. When used, **return** stores a specified value, but doesn't print it out to the console. The **return** statement is useful for passing values to other parts of the program, but it won't output to the console unless we use the **print** function with it. See the following example:

```
def sales_tax_calc(amount):
    sales_Tax = 1.15
    return amount * sales_tax
```

We could call the function by using the following example:

```
sales_tax_calc(55.00)
```

The value **63.25** would be returned, but nothing would be printed to the console. To actually display this amount, we would have to use the following **print** function:

```
print(sales_tax_calc(55.00))
```

Now that we've covered some of the concepts behind functions, let's learn more with the learnpython.org website.

Step 1

Navigate to the following URL:

https://learnpython.org/en/Functions

Step 2

Play around with the different interactive python windows. Each one covers a different function function concept we were introduced to at the start of this section. Let's stop when we reach the **Exercise** portion of the page.

Step 3

In the **Exercise** section, we're given the following criteria to complete the exercise:

- modify the **list_benefits** function on lines 2 and 3 so that it returns a number of different strings
- modify the **build_sentence** function on lines 6 and 7 so that it accepts a single string argument and returns a sentence starting with the supplied string argument, and ending with the string **" is a benefit of functions!"**

First we'll modify the **list_benefits** function in the following way:

```
def list_benefits():
    return ["More organized code", "More readable code", "Easier code
reuse", "Allowing programmers to share and connect code together"]
```

Note that the second and third lines of this code are meant to fit on a single line, but is presented here on two lines for the sake of legibility.

Lastly, we'll modify the **build_sentence** function in the following way:

```
def build_sentence(benefit):
    return benefit + " is a benefit of functions!"
```

Click on the **Run** button, and the exercise should be solved.

Part 3 – Classes and Objects

In this part of the workshop, we'll be looking at classes and objects as terms that are relevant to Python.

What are Objects?

Python is an object-oriented programming (OOP) language, and as such, all pieces of data in Python code are considered objects. Objects in Python share attributes and characteristics depending on which class they fall under.

What are Classes?

In Python, objects are grouped into different classes according to what type of data they contain. There are classes associated with all of the data types we've been working with previously such as:

- strings
- integers
- floats
- Booleans

In addition, container objects such as **lists** and **tuples** are also considered their own classes in Python.

An object's class indicates which built-in functions, methods and variables it has. Let's look at some of the functions and methods associated with the string object class as an example:

```
my_string = 'The HackerFrogs'

print(len(my_string))
print(my_string.upper())
```

This code would resulting in the following:

```
15
THE HACKERFROGS
```

Here we've executed a function an method on the **my_string variable**. Both the **len** function and **upper** method are unique to the string class.

If we were to change the value of my_string to an integer, executing either the len function or upper method would result in an error message like the one below:

```
AttributeError: 'int' object has no attribute 'upper'
```

How do we Create Classes?

To create a class, we begin with **class**, then the name of the class (the first letter of the name is capitalized), then a colon. On the next line, indent (4 spaces), then **def**, then **__init__**, a pair of parentheses, then inside the parentheses, self, then any other attributes the class requires, then a colon. One important feature of class creation is defining the **__init__** function, which specifies the attributes of the class. One attribute which all classes share is the self attribute, which is used to access other attributes and methods of the class in the code.

Now that we've covered some concepts about classes and objects, let's solve the exercises over at learnpython.org!

Step 1

Navigate to the following URL:

https://learnpython.org/en/Classes_and_Objects

We can play around with each of the concepts covered in the various sections on the webpage, but we'll stop when we reach the **Exercise** section.

Step 2

In the **Exercise** section, we're given the following task:

- create two objects in the Vehicle class with several different attributes:
- one Vehicle object is to be set to the variable “car1”, with a name of “Fer”, a color of “red”, a kind set to “convertible”, and a value equal to 60 000.00
- the other Vehicle object is to be set to the variable “car2”, with a name of “Jump”, a color of “blue”, a kind set to “blue”, and a value equal to 10 000.00

Starting on line 12, we can insert our own code to complete the exercise.

There are many possibilities, but the code below will solve the exercise

```
car1 = Vehicle()
car1.name = "Fer"
car1.color = "red"
car1.kind = "convertible"
car1.value = 60000.00
car2 = Vehicle()
car2.name = "Jump"
car2.color = "blue"
car2.kind = "van"
car2.value = 10000.00
```

Click the **Run** button and the exercise should be solved!

Summary

In this introductory lesson to programming, we learned about a few basics of the Python language and the following concepts:

Functions

Functions are sets of defined programming instructions which can be used multiple times in a program. Executing a function in a program is called a “function call”, and functions allow programming code to be compartmentalized and allows for the repetition of instructions with less space used in the source code file.

Loops

Programming loops are a very common tool which allows programs to repeat instructions on a set of data until either A: the loop instructions have been executed on all of the data in the set, or B: a pre-set stop condition is achieved.

Classes and Objects

Python is an object oriented programming language, which means that all data pieces are considered objects, and objects are divided into classes according to what data type they are. All objects in a class share certain attributes, functions, and methods.

In the next workshop we'll continue to learn about Python programming with learnpython.org, covering two essential tools of programming, loops and functions, as well as discussing how object classes work in Python.

Extra Credit

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

Pynative: Python If Else And For Loop Exercises With Solutions

https://pynative.com/python-if-else-and-for-loop-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 – Control Flow and Functions

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 Control Flow (part 10 to end) and Functions sections.

Automate the Boring Stuff with Python: Chapters 2 and 3

https://automatetheboringstuff.com/2e/chapter2/
https://automatetheboringstuff.com/2e/chapter3/

Extra Research

The following articles have more information about the topics discussed in this workshop:

More about using classes in Python:
https://www.dataquest.io/blog/using-classes-in-python/

More about Python functions:
https://www.w3schools.com/python/python_functions.asp

Until next time, HackerFrogs!