HackerFrogs Afterschool
Beginner Cybersecurity Skills

Intro to Python Programming with Learnpython.org: Part 2

Class:	**Programming (Python)**
Workshop Number:	**AS-PRO-PY-02**
Document Version:	**1.0**
Special Requirements:	**Completion of AS-PRO-PY-01**

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 second 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 Python string formatting
- Learn about basic string operations
- Learn about conditional statements

Part 1 – String Formatting

When working with Python, it is important to know how to print out strings which incorporate program variables. Let's say we have a program which reports the weather and temperature for the city of London, England. We could have variables in our code that look like the following:

```
weather = "rainy"
temperature = 12
```

If we wanted the code to print out a sentence for the user that reports the weather and temperature contained within the variables, we could do so in one of two ways (note that the code in the examples below are meant to fit on a single line, but are presented here on two lines for the sake of readability):

Method 1

```
print('The weather in London is ' + weather + ' with a temperature of
' + str(temperature) + ' degrees celcius.')
```

In this method, we have to use the addition operator to indicate additional content to be printed to the line. We also must use the **str** method to force the printing of a non-string object as a string (the **temperature** variable is an integer type). This method is a bit clunky, as it involves a lot of sets of quotes and concatenation, as well as using the **str** method for our integer variable.

Method 2

```
print('The weather in London is %s with a temperature of %d degrees
celcius.' % (weather, temperature))
```

This method uses C-style string formatting. In this format, the **%s** denotes a string and the **%d** denotes an integer.

Now that we've discussed string formatting a bit, 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/String_Formatting

Let's review the **String Formatting** section.

In the first interactive python window (IPW) we see that we can print out a message that includes a single variable by including **%s** inside the quotes (in this case it denotes a string variable). Then, after the quotes, we include a **%** symbol, then the name of the variable to be inserted into the quotes.

In the second IPW, we see that we can include the values of more than one variable in our string if we list out all of the variable names in a tuple (a simple list). In this case, we need to name the variables we want to appear in our string, in order, within the tuple, and ensure that the data types all match up.

In the third IPW we learn that in fact any variable can be rendered as a string by using the **%s** syntax.

Step 2

In this **Exercise** section, we're tasked with editing the script in such a way that the following output is displayed:

Hello John Doe. Your current balance is $53.44.

We also need to output the above text while using the string formatting method we learned in this section. The only variable we need to modify in the script is the **format_string** variable on line 2. The following format should work:

```
format_string = "Hello %s %s. Your current balance is $%s."
```

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

If we take the whole print statement with the variable values, it would look like the following:

```
print("Hello %s %s. Your current balance is $%s." % ("John", "Doe",
53.44))
```

Part 2 – Basic String Operations

In this portion of the workshop we'll be looking at how Python strings be can manipulated to retrieve specific information, mostly through different functions and methods available to the string class of objects in Python.

The Len Function

The **len** function returns the length of a particular string, in characters. For example:

```
>>> len('tomato')
6
```

Here the output is **6**, because that is the number of characters contained within the string **tomato**. To use the **len** function, put the string argument between the parentheses.

The Index Method

The **index** method is used to locate the first instance of the argument within the object. The output is the number of characters into the string where the argument appears (counting from zero).

```
>>> food = "pizza"
>>> food.index("a")
4
```

Here the output is **4**, because we're starting our count from zero, and the **a** in pizza is the fifth letter.

The Count Method

The count method returns the number of times the argument appears in the object. Note that we don't count from zero in this case. See the following example:

```
>>> greeting = "Hello"
>>> greeting.count("l")
2
```

Here the output is 2, because that's the number of times the letter **l** appears in the string **Hello**.

String Index Slicing

Just as we can index lists to retrieve the item contained at a specific position, we can index strings, with each character in the string counting as a separate item.

```
>>> pet = "dog"
>>> print(pet[0])
d
```

Remember, when indexing in Python, counting starts from zero instead of one.

What's more interesting is the ability to select specific slices of the string.

```
>>> pet = "python"
>>> print(pet[2:4])
th
```

Here we've isolated the third and fourth letters from the string **python**. Inside the brackets, we indicate the character to start from (counting from zero), then a colon, then the character that marks the stopping point (which is not returned).

So **2:4** indicates to start from the third character, and stop at the fifth character, not including it in the output.

If a second colon is included in index slicing, the number to the right of it indicates the number of characters to count by, when slicing. For example:

```
>>> bug = "centipede"
>>> bug[0:9:2]
cniee
```

In this output, we see that the slice starts from the first character, ends before the tenth character, and skips every second character.

The last part of string index slicing we'll cover is using negative index numbers. If a negative index number is used, the index starts counting from the end of the string instead of the beginning. Take the following example:

```
>>> fruit = "grapes"
>>> print(fruit[-1])
s
```

Here we're printing out the string index for **grapes** at the **-1** position, which is the last letter.

When indexing, if we include a colon (:) without a number to its left, the first colon will have a default value of 0 (the first character of the string), and the second colon will have a default value of the number of characters in the string, plus one (the end of the string). Take the following example:

```
>>> fruit = "grapes"
>>> print(fruit[::-1])
separg
```

Here the whole string is in range, due to the inclusion of two colons with no numbers to the left of them, and we're stepping through the characters in reverse order, one by one (-1).

The Upper and Lower Methods

The **upper** and **lower** methods output all of the characters in uppercase or lowercase, respectively. For example:

```
>>> birthday_message = "Happy Birthday!"
>>> print(birthday_message.upper())
HAPPY BIRTHDAY!
>>> print(birthday_message.lower())
happy birthday!
```

The Startswith and Endswith Methods

The **startswith** and **endswith** methods are used to test whether strings contain the specified argument at the start or end of the string, respectively. For example:

```
>>> group_name = "the HackerFrogs"
>>> group_name.startswith("the")
True
>>> group_name.endswith("Frog")
False
```

The second method call returns False because the endswith method is checking against **Frog**, when the end of the group_name variable is **Frogs** (note the letter **s** at the end).

The Split Method

The split method converts a string into a list, with the separation between items indicated by the argument given (usually a space or a comma). For example:

```
>>> groceries = "eggs,milk,bread,lettuce"
>>> groceries_list = groceries.split(",")
>>> print(groceries_list)
['eggs', 'milk', 'bread', 'lettuce']
```

Now that we've covered the different methods and function we'll encounter in the next exercise, let's test them out over at the learnpython.org website.

Step 1

Navigate to the following URL:

https://learnpython.org/en/Basic_String_Operations

Step 2

Play around with the different interactive python windows. Each one covers a different string function or method that was introduced in this section. We'll stop when we reach the **Exercise** portion of the page.

Step 3

In the **Exercise** section, we're given the following criteria to set to the value of the **s** variable to complete the exercise:

- Modify the length of the string to exactly 20 characters
- Modify the content of the string so that the letter **a** first appears at the ninth character
- Modify the string so that the letter **a** appears exactly two times.
- Modify the string so that it beings with the characters **Str**
- Modify the string so that it ends with the characters **ome!**
- Modify the string so that it contains exactly three words, separated by spaces

Changing the **s** variable in the following way will solve the exercise:

```
s = "Strings are awesome!"
```

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

A Quick Note on Boolean Data

The next part is heavily reliant on a new data type, the Boolean. The Boolean data type is either True or False, and these two values correspond to numbers 1 and 0, respectively. Boolean values are used in evaluation statements to determine whether a statement is True or False.

Part 3 – Conditions

In programming, conditions are the evaluations of statements, and takes on a boolean value of True and False. If we have the following code:

```
x = 3
print(x == 3)
```

In Python (and several other programming languages) the double equal signs (==) represents a comparison between two values, and is comparing whether or not the values are equal. So in the above

code, the boolean value of whether x is equal to 3 will be printed out, and because it is, the console prints out **True**.

If instead we wanted to evaluate whether or not **x does not** equal three, we could use the following print method:

```
print(x != 3)
```

In this case, the Python console would return **False**

When comparing values, the following operators can be used

Operator Symbol	Meaning	Example (True Statement)
==	Equal to	1 == 1
!=	Not equal to	1 != 2
<	Less than	0 < 1
>	Greater than	10 > 1
<=	Less than or equal to	7 <= 7
>=	Greater than or equal to	17 >= 8

The If Conditional

The **if** conditional is very important to Python programming because it enables the program to make decisions depending on whether certain conditions have been met or not.

Let's look at the following example:

```
password = "bad_password"

if password == "super_secure_password":
    print("Access granted!")
else:
    print("Access denied!")
```

Here, we see that the **password** variable is set to the string **bad_password**, then there is an **if** statement which compares the **password** value to the **super_secure_password** string. Should the **if** conditional evaluates to True, then the instructions on the next line are executed, but if not, then the **else** clause takes effect and executes instructions below the **else** clause.

Note that in any **if** statement, the **else** clause that gives instructions when the if statement is False is entirely optional.

Also note that the standard if conditional is written as a code block, which means that any instructions that are written below the if statement require indentation (4 spaces) to be considered valid code.

The AND Operator

When writing if conditionals, the AND operator can be used to add an additional condition. Both conditions to the left and right of the AND operator must be True in order for the overall statement to evaluate to True. For example:

```
birthday = 'October 24th'
todays_date = 'October 24th'

if birthday == 'October 24th' and todays_date == 'October 24th':
  print('Happy birthday!')
```

Here, the print instruction will not execute unless both the **birthday** and the **todays_date** values match the **October 24th** string.

The OR Operator

When using the OR operator in if statements, if either of the conditions to the right or left of the OR operator evaluate to True, then the overall statement is True. See the following example:

```
favorite_food = 'spaghetti'
hobby = 'watching movies'

if favorite_food == 'spaghetti' or hobby == 'classical painting':
    print("Let's take a vacation to Italy!")
```

Here, if either the **favorite_food** value is **spaghetti** or the **hobby** value is **classical painting**, the print instruction will execute.

The IN Operator

The IN operator is used to check if there is a specific item within a container object, such as a list. See the following example:

```
name = "John"
party_guests = ["John", "Sue", "Bobby"]
if name in party_guests:
    print("You're invited to the party!")
```

Here, the print instruction will be executed if one (or more) of the items in the **party_guests** list matches the value of the **name** variable, which is the **John** string.

The IS Operator

The IS operator is used to compare whether an instance is exactly referencing another object, as opposed to matching the value of the object. See the following example:

```
x = [2, 5, 7]
y = x
z = [2, 5, 7]
print(x == z)
print(x is z)
print(x is y)
```

We can see that the values of **x**, **y**, and **z** are all the same. However, the is operator only returns True is the two objects on either side of it reference the same object. Thus the first and third print functions return True, while the second function returns False.

Note that both the IS and NOT operators have special interactions with the numbers 0 and 1, in that they are considered False and True, respectively, for comparison operations. See the following example:

```
>>> x = 0
>>> y = 1
>>> x is False
True
>>> y is not True
False
```

As mentioned earlier, Python considers the numbers 0 and 1 to be equivalent to the values False and True.

Now that we've covered all the comparison operators, let's solve the exercises over at learnpython.org!

Step 1

Navigate to the following URL:

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

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:

- Change all of the variables in the code so that all of the if statements evaluate to True

The first if statement on line 7 requires the **number** variable to be greater than 15, so we can change it to any number above 15.

The second if statement on line 10 requires the **first_array** list has at least one object contained in it. Edit the list so that it has 3 objects in it.

The third if statement on line 13 requires that the **len** (number of objects) in the list **second_array** be equal to **2**, so let's edit is so that there are two objects in there.

The fourth if statement on line 16 requires that the sum of the **first_array** length and the **second_array** length be equal to 5. We need to make sure there are exactly 3 objects in **first_array** and 2 objects in **second_array**.

The fifth if statement on line 19 requires that the **first_array** list has at least one object in it, and the first object has to be the number **1**. We should modify the first object in the list so that it's the number **1**.

The sixth if statement on line 23 requires the **second_number** variable be not True. We can edit the value of **second_number** to be False, or we could also give it a value of 0, which in Boolean terms is False.

There are many possibilities, but the code below will solve the exercise if substituted in for the default variables:

```
number = 16
second_number = 0
first_array = [1,2,3]
second_array = [1,2]
```

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:

String Formatting

When using the **print** function we can incorporate the values of variables by using C-style string formatting, allowing for the printing of dynamic strings.

Basic String Operations

Python string objects can be manipulated in many different ways through the use of indexing, functions, and methods that are specific to the string class. These methods and functions can output the number of times a particular letter appears in a string, or render the entirety of the string in uppercase or lowercase letters, for example.

Conditionals

Conditionals are instructions that are only executed if a certain condition exists. E.g., if **x** is true, then do **y**. Conditionals are tied to Boolean equations, whether a given statement or equation is True or False, and performing different instructions depending on which is the case.

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:

Udacity: Introduction to Python Programming – Control Flow: Parts 1-9

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

NOTES: We will need to have a registered user account to complete these exercises. Work through the sections in the course, but stop before the Control Flow part 10 section.

Extra Research

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

Different ways to format strings in Python:
https://www.geeksforgeeks.org/string-formatting-in-python/

More practice with "if conditionals":
https://realpython.com/python-conditional-statements/

Until next time, HackerFrogs!