HackerFrogs Afterschool
Beginner Cybersecurity Skills

Intro to Python Programming with Learnpython.org: Part 1

Class:	Programming (Python)
Workshop Number:	AS-PRO-PY-01
Document Version:	1.0
Special Requirements:	None

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 introductory lesson of 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 an overview of the Python language
- Learn about programming variables and types
- Learn about programming lists
- Learn about basic mathematical operators

What is Programming?

According to Wikipedia, computer programming is the process of performing a particular computation (or more generally, accomplishing a specific computing result), usually by designing and building an executable computer program. Put in broader terms, programming is the process of writing instructions for a computer to execute. The instructions can result in output as simple as printing the answer to a simple math equation or a word to the screen, or as complex as the latest video editing software, games, websites, or countless other pieces of software used in our daily lives.

Introduction to the Python Language

Python is a high-level, interpreted, general-purpose programming language. What does this mean?

High-Level

Programming languages are classified as either low-level or high-level, depending on how abstract the syntax of the language is from the pure machine language that computer processors use. Programming in a high-level language has two major effects: First, abstract syntax allows easier understanding for people who need to read code written in high-level languages and write code using the same. Second, due to its high-level of abstraction from machine language, processes using high-level languages are relatively much slower than the same processes performed by low-level languages. In other words, high-level programming languages are easier to code in, but are much slower than low-level languages.

Interpreted

Interpreted languages are programming languages that do not require compilation before instructions are run, as opposed to compiled languages where programming code must be converted into machine code by a compilation program before they can be run. This makes interpreted languages much quicker as far as resolving whether or not any given segment of code is working properly, since the program does not have to be compiled each time a change is made, which is the case with compiled programs.

However, interpreted languages are inherently slower than compiled languages in their speed of program execution because interpreted languages convert the program's instructions into machine code when the program is run, whereas compiled programs skip this step because the conversion of program code into machine code is performed prior to program execution.

General-Purpose

General-purpose programming languages are programming languages designed to be able to create a wide range of application types, as opposed to domain-specific programming languages, which typically are used to operate in specialized environments, such as database programming, or programming for industrial control systems.

Part 1 – Hello World

In this introduction to Python language learning, we'll be looking at our first built-in function, the print function, as well as the classic programming tradition of writing a **Hello, World!** program as our first exercise.

Step 1

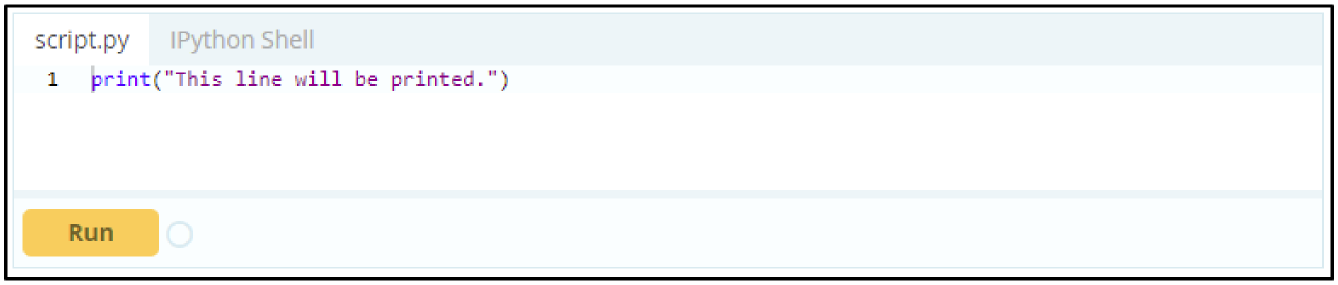
Using a web browser, navigate to the following webpage:

https://www.learnpython.org/en/Hello%2C_World%21

Let's review the **Hello, World!** section of the webpage.

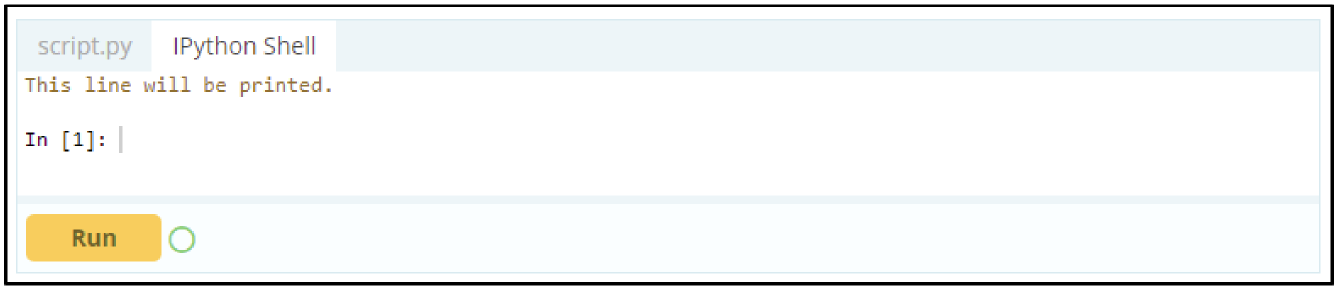
One major takeaway from this section is that there are two commonly encountered version of the Python language, versions 2 and 3. Python version 3 (Python 3) is the most widely used version of Python currently, so we'll be learning Python 3.

Let's try printing some different statements using the print function before moving on to the next section. To do so, click on the white space in the interactive Python window (IPW). The IPW looks like the screenshot below:



```
print("This line will be printed.")
```

Click on the orange **Run** button. The IPW will switch from the **script.py** tab to the **iPython Shell** tab. See the screenshot below:



```
This line will be printed.

In [1]:
```

To continue experimenting with the IPW, click on the **script.py** tab and modify the text between the double quotes (“”). After we've tried printing some different things, we should proceed to the next step.

Step 2

In the **Indentation** section, we are introduced to the Python system of using indents in the code to indicate code blocks. This concept will become more important when we cover later topics, so we can skip this section.

Step 3

In this **Exercise** section, we'll be solving our first programming task, outputting **Hello World!** to the console. If we didn't have any problems printing other text strings to the IPW so far, this shouldn't be difficult. Edit the text on line 1 of the IPW so it resembles the following:

```
print("Hello, World!")
```

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

Part 2 – Variables and Types

In this portion of the workshop we'll be looking at how Python handles variables, and the different types of data that Python uses.

Variables

Virtually all programming languages use variables, which are storage locations associated with a specific name, as well as a value that is associated with it. Take the following as an example:

```
my_name = "shyhat"
```

In this case **my_name** is the name of the variable, and **shyhat** is the value of the variable. The value **shyhat** is enclosed in quotes, because the value of the variable is a string (non-number characters)

The real benefit of using variables in programming is that variables can be referenced multiple times in the same code, saving a lot of time and accounting while writing code. In addition, variables can change value after they've been defined. For example: say we have a game where points are recorded. The initial variable that tracks points might look like this:

```
point_total = 0
```

If at some point in the game, 20 points are scored, we can update the variable in the following manner:

```
point_total = point_total + 20
```

At this point in code execution, the **point_total** variable would hold the value 20. With this method, we could adjust the **point_total** variable any number of times.

Data Types

In programming, data can be one of a few different types, and here we will learn 3 common data types:

`Strings:`	Non-numerical characters. E.g. letters and words are considered strings.
`Integers:`	Whole numbers. E.g., 1, 255, 1337.
`Floating-Point Numbers:`	Numbers that are accurate to several decimal places. E.g, 0.5 or 12.758

To save time and space, floating-point numbers are often referred to as **floats**.

Step 1

Navigate to the following URL:

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

In the **Numbers** section, we are first introduced to integers and floats. Note that when we define integers or floats, we do not enclose them in quotes.

Step 2

In the **Strings** section, we're informed that strings must be enclosed in quotes to work properly in Python. Note that if we want to use quotes or apostrophes inside of our strings, we will have to alternate between single quotes and double quotes in order to have the strings come out properly. For example, we have to construct the following string variables in this way:

```
dialogue = “That car is Amy's!”
```

or

```
narration = 'John shouted, “Watch out!” as he jumped back'
```

One thing that is mentioned at the end of the **Strings** section is that strings cannot be mixed with integers or floats in mathematical operations.

Step 3

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

- define a string value of **hello** for the **mystring** variable
- define a float of 10.0 for the **myfloat** variable
- define an integer of 20 for the **myint** variable

So we change the values of the variables on lines 2-4 of the code to look exactly like the following:

```
mystring = "hello"
myfloat = 10.0
myint = 20
```

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

Part 3 – Lists

Lists are part of what Python categorizes as **collections**, objects that hold one or more values, which can be strings, numbers, or a mixture of different data types. An example list in Python could look like this:

```
vegetables = ["carrots", "lettuce", "beets", "celery", "potatoes"]
```

or

```
related_to_pie = ["round", 3.14, "delicious"]
```

As we can see by the example, a list is defined as a number of objects (separated by commas " , ") between a pair of square brackets "[ ]".

A couple of aspects that make Python lists different from other collection methods in Python (i.e., tuples, sets and dictionaries). Lists are:

- mutable (objects in the list can be changed)
- ordered (objects in the list take a specific order, depending on the point at which they are added to the list)
- allows duplicates (objects in the list can have the same values)

Let's learn a little more about lists on Learnpython.org.

Step 1

Navigate to the following URL:

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

Let's read the Lists section. As we can see in the first IPW in the section, on line 1, the variable **mylist** is defined as an empty list:

```
mylist = []
```

So we see that lists don't need to have any objects in them when they're created.

On line 2 of script, we're introduced to a method available to Python list objects: the **append** method. A method is an operation that can be performed with a specific class of objects. In this case, the class is list objects, and the **append** method allows values to be added to the end of a list.

We can recognize when methods are called (used or run) by the dot (.) that appears after the object's name, for example:

```
mylist.append()
```

On line 5 of the script, we're introduced to **list indexing**, which is how specific entries in a list are referenced. For example, if we have the following list:

```
fruits = ["apples", "oranges", "bananas"]
```

If we wanted to reference **apples** from the **fruits** list and print it to the console, we could input the following code:

```
print(fruits[0])
```

Note that counting in list indexing starts from 0 instead of 1. Using the same list with the index of 2, i.e, **fruits[2]**, would reference **bananas** from the list, since its the third object in the list.

Step 2

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

- use the append list method to add the numbers **1**, **2**, and **3** to the **numbers** list.
- use the same method to add the words **hello** and **world** to the **strings** list.
- use list indexing to define the **second_name** variable as the second item in the **names** list.

As the comment on line 5 of the script indicates, we should write out code under that line. Since the **second_name** variable is right on line 6, we can start there. To fulfill the third requirement for the exercise, we can replace the current value of **None** for **second_name** and replace it with the second name in the **names** list with the code **names[1]**. Then on line 7, we can begin adding items to the **numbers** list, using the append list method (i.e., **numbers.append(1)**, **numbers.append(2)**, and **numbers.append.(3)**). Likewise, to add the necessary values to the **strings** list, we also use the **append** method (**strings.append("hello")**, and **strings.append("world")**). After we've finished, it should look the following:

```
second_name = names[1]
numbers.append(1)
numbers.append(2)
numbers.append(3)
strings.append("hello")
strings.append("world")
```

After the above code is added to the IPW starting on line 7, click the **Run** button and the exercise should be finished.

Additionally, we can add multiple values to a list at the same time (as opposed to one at a time with the **append** method) using the **extend** method. E.g., **numbers.extend((1,2,3))**.

```
second_name = names[1]
numbers.extend((1,2,3))
strings.extend(("hello","world"))
```

We can use the above code to solve the exercise as well.

Part 4 – Basic Operators

Step 1

Navigate to the following URL:

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

Review the **Arithmetic Operators** section.

In Python, common arithmetic operations can be performed using the following symbols:

```
addition:           +
subtraction:        -
multiplication:     /
division:           *
```

The arithmetic operations that occur in Python follow the PEMDAS order of operations. For those unfamiliar with PEMDAS, that means that the arithmetic operations occur in the following order:

```
(P)arenthesis
(E)xponents
(M)ultiplication
(D)ivision
(A)ddition
(S)ubtraction
```

In summary, all operations enclosed in parentheses "( )" are performed first, then all exponent operations are resolved, followed by multiplication, division, addition, then subtraction operations at the end.

Let's play around with the equation in the `number` variable in the first interactive Python window. The PEMDAS order of operations can be seen if we supply the following for the **number** variable:

```
(4 / 2) * 2 ** 2 / 4 + 2 - 4
```

First the division operation inside of the parentheses (4 divided by 2) is performed. Second, the exponent operation (2 ** 2, which is Python's syntax for "two to the second power"), is resolved. Third, the multiplication (2 times 4) is resolved. Forth, the division (8 divided by 4) is resolved. Fifth, the addition (2 + 2) is resolved. Finally, the subtraction (4 - 4) is resolved, which sets our final result to 0 (zero).

Step 2

Under the first interactive Python window, the modulo (%) operator is introduced. When the modulo operation is performed, the first number is divided by the second, and returns the remainder of the division. For example, 3 % 3 = 0, since 3 divides into 3 evenly. But if the equation is 3 % 7, then the remainder after division would be 1, so 3 % 7 = 1.

Let's play around with the remainder variable in the second interactive Python window before moving on to the next step.

Step 3

Under the second interactive Python window, the Python syntax for handling exponents (squared, cubed numbers, etc) is introduced. The symbol for exponents in Python is two asterisks (**), with the exponent power indicated after a space. For example, 2 to the power of 2 is represented in Python as 2 ** 2 and 3 to the power of 16 is written as 3 ** 16.

Step 4

Under the heading **Operators with Strings**, we are introduced to the idea of using arithmetic operators with strings. There are only two commonly used operators used with strings in Python, and the first one is the addition operator. However, addition in the case of Python strings can more accurately be described as concatenation (the process of joining strings from end-to-end). For instance, the following Python code:

```
print("Hello" + "World")
```

Produces the following when run:

```
HelloWorld
```

NOTE: spaces do not appear automatically when using the addition operator in this way. If we wanted to write the "Hello World!" string with the space using the addition operator in Python, we use any of the following code:

```
print("Hello" + " " + "World!")
```

or

```
print("Hello " + "World!")
```

or even

```
print("H" + "e" + "l" + "l" + "o" + " " + "W" + "o" + "r" + "l" + "d"
+ "!")
```

We can play around with the first interactive Python window to experiment a bit before moving on to the next step.

Step 5

Under the first interactive Python window, we are introduced to using the multiplication operator with strings. For example, if we wanted to print the string **Go** five times, we could use the following code:

```
print("Go" * 5)
```

Which will result in the following output:

```
GoGoGoGoGo
```

Step 6

We'll now look at the **Using Operators with Lists** section. Lists in Python can be joined by using the addition operator. Much like using the addition operator with strings, the contents of the lists are concatenated (joined from end-to-end). If we run the default script in the first interactive Python window in this section we see that the list contents from the **odd_numbers** list appear first in the output, followed by the contents of the **even_numbers** list.

Under the first interactive Python window we see that, just like strings, we can use the multiplication operator to repeat the contents of lists

Step 7

Now let's tackle the **Exercise** section. The exercise goal is to create two lists, **x_list** and **y_list**, which contain ten of the **x** and **y** variables, respectively, then define another variable, **big_list**, which contains the contents of both **x_list** and **y_list**.

It's easy to get lost in all of the code in the script, but we're given a handy comment on line 4 of the script to change the code on lines 5-7. The code on the other lines should not be modified.

To solve the exercise, we'll need to make the following adjustments to the code on lines 5-7:

- In both **x_list** and **y_list**, we should use the multiplication operator to define the relative list objects (x and y) ten times each.
- In the definition for **big_list**, we should concatenate **x_list** and **y_list** using the addition operator.

When all the changes have been written, the code on lines 5-7 should look similar to the following:

```
x_list = [x] * 10
y_list = [y] * 10
big_list = x_list + y_list
```

Click the **Run** button at the bottom of the interactive Python window 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:

Hello World!

A beginner's program that displays the message **Hello World!** to the console. It is often the first program written by any student learning a new programming language.

Variables

Variables are objects in a program that reference other values, e.g. **pi = 3.14**, or **hours_in_a_day = 24**. Whenever the name of the variable is used in a program, its value is substituted in its place.

Data Types

Each piece of data in a program is categorized, and the program must know what type of data each piece is in order to process it correctly. The three data types introduced in this lesson were:

> integer – whole numbers (e.g., 1337, -7)
> float – numbers accurate to several decimal places (e.g., 3.14, -1.297)
> string – non-numeric characters (e.g., snake, Hello)

Lists

Collections of objects, which can be a mixture of different data types. List objects referenced by their index number, which counts up, starting from zero.

Basic Operators

All of the basic mathematical operators for addition (+), subtraction (-), multiplication (*), and division (/).

In the next workshop we'll continue to learn about Python programming with Learnpython.org.

Extra Credit

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

> **Datacamp: Intro To Python For Data Science – Chapter 1**

https://campus.datacamp.com/courses/intro-to-python-for-data-science

NOTES: We will need to have a registered user account to complete these exercises

Extra Research

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

This article explains the directory structure of the Linux filesystem:
https://www.linuxfoundation.org/blog/blog/classic-sysadmin-the-linux-filesystem-explained

This article explains file permissions in Linux:
https://www.redhat.com/sysadmin/linux-file-permissions-explained

Until next time, HackerFrogs!