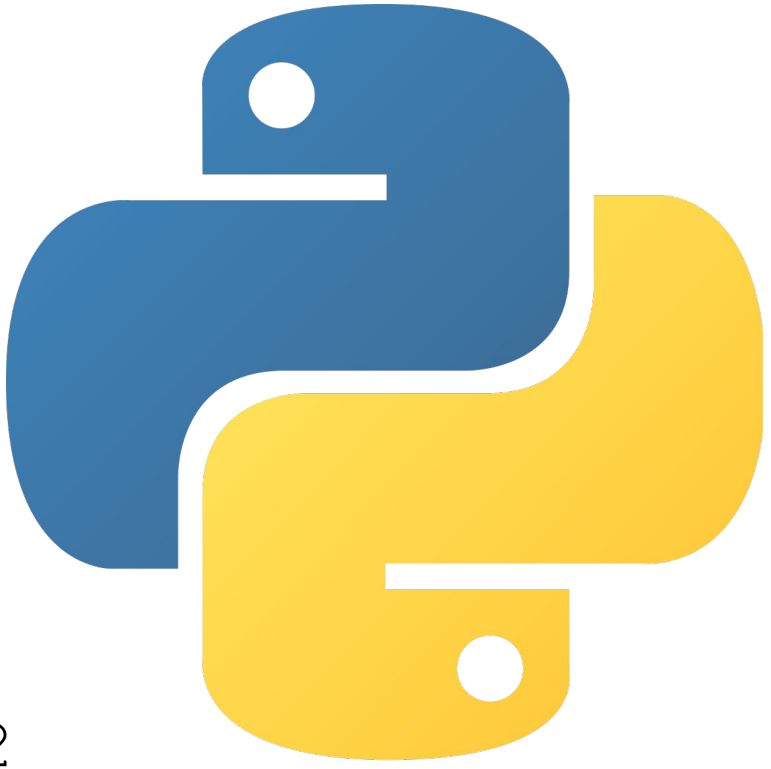
HackerFrogs Afterschool
Python Programming Basics: Part 3

Class:
Programming (Python)

Workshop Number:
AS-PRO-PY-03

Document Version:
1.0

Special Requirements:
Completion of AS-PRO-PY-02



What We Learned Before

This workshop is the third class for intro to Python programming.

During our last workshop, we learned about a few programming concepts through Python, including the following:

String Formatting

```
>>> name = "Shyhat"
>>> daily_cups_of_coffee = 3
>>> print("%s drinks %d cups of coffee a day." % (name, daily_cups_of_coffee))
Shyhat drinks 3 cups of coffee a day.
```

String formatting is a way for us to include variable values in strings by using C-style syntax.

Basic String Operations

```
>>> city = "Seattle"
>>> city.count("e")
2
```

Strings in Python have a number of different functions and methods that can be useful for obtaining specific details about the strings.

Basic String Operations

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

We can also output slices of strings through string indexing.

Conditions

```
password = 'mysecretpassword'

if password == 'mysecretpassword':
    print('password correct')
else:
    print('incorrect password')
```

Conditions are used in programs to determine a program's course of action. They rely on the evaluation of Boolean statements to determine if certain instructions are performed or not.

This Workshop's Topics

New topics for this session:

- Part 1: Programming Loops

- Part 2: Functions

- Part 3: Classes and Objects

This Workshop's Topics

New topics for this session:

- Part 1: Programming Loops

- Part 2: Functions

- Part 3: Classes and Objects

This Workshop's Topics

New topics for this session:

- Part 1: Programming Loops

- Part 2: Functions

- Part 3: Classes and Objects

This Workshop's Topics

New topics for this session:

- Part 1: Programming Loops

- Part 2: Functions

- Part 3: Classes and Objects

Part 1: Programming Loops

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

```
0
1
2
3
```

Programming loops are tools that programmers use to run the same instructions multiple times.

Programming Loops

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

```
0
1
2
3
```

Here, the code repeats the instructions four times, and each time the value of **i** is printed. The first time, **i** is 0, then 1, 2, and 3. It stops before the number 4, because it started counting from zero.

Programming Loops

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

```
0
1
2
3
```

There are two major categories of loops in Python: **for loops** and **while loops**. We'll cover **for loops** first.

“For” Loops

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

```
0
1
2
3
```

For loops are instructions that repeat for each item in a set. The loop stops after the instructions have been performed on all items in the set.

“For” Loops

```
animal = "frog"

for i in animal:
  print(i)
```

```
f
r
o
g
```

Any object that can be referenced by index can be used in a for loop, including strings.

“For” Loops

```
for i in range(10):
    print(i)
```

To write a **for loop**, begin with for, then a variable name that is only used inside the loop, typically the letter i, then in, then the name of the object to be iterated over, then a colon.

“For” Loops

```
for i in range(10):
    print(i)
```

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.

“For” Loops

```
>>> print(type(range(5)))
<class 'list'>
```

All **for** loops iterate over the contents of an object, typically a list. In fact, the **range** function creates a list with a series of numbers depending on the arguments given to it.

“While” Loops

```
count = 0

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

```
0
1
2
```

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

“While” Loops

```
count = 0

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

```
0
1
2
```

Here, the loop keeps running until the **count** variable is no longer less than 3, and each iteration of the loop prints the value of count, then increases the value of count by one.

“While” Loops

```
count = 0

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

```
0
1
2
```

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 ends if the statement is False.

“While” Loops

```
count = 0

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

```
0
1
2
```

In the example, the loop runs over and over again while the statement supplied to it remains True. Once the **count** variable is incremented to 3, then **count < 3** returns False, and the loop ends.

“While” Loops

```
count = 0

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

We begin a while loop with while, then the statement that must evaluate True for the loop to execute, then a colon.

“While” Loops

```
count = 0

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

Then, 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

```
count = 1

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

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**.

Infinite Loops

```
count = 1

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

Although there are cases where programmers want to 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

```
count = 0

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

The **break** statement is an instruction we can insert into programming loops. When executed, the break instruction terminates execution of the current loop.

“While True” Loops

```
count = 0

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

This example also uses the **while True** style of loop, which is an infinite loop by default, and requires another means inserted within the loop to terminate the loop properly (e.g., the break statement).

The Continue Statement

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

```
0 is an even number
2 is an even number
4 is an even number
```

The **continue** statement is an instruction that stops execution in the current loop and 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.

The Else Clause

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

```
0 is an even number
1 is an odd number
2 is an even number
3 is an odd number
```

Just as we can use **if** conditionals within loops, we can use the **else** clause to execute instructions in the case that the **if** condition returns False.

Programming Loops Exercise

Let's practice using loops with Python at the following URL:

https://learnpython.org/en/Loops

Programming Loops Quiz (1 of 3)

When will the programming loop end?

```
my_list = [1, 2, 3]
for num in my_list:
  print(num)
  my_list.append(num + 1)
```

A) After the last number in the list is printed.
B) After the number 3 is printed.
C) This loop will never end.

Programming Loops Quiz (1 of 3)

When will the programming loop end?

```
my_list = [1, 2, 3]
for num in my_list:
  print(num)
  my_list.append(num + 1)
```

A) After the last number in the list is printed.
B) After the number 3 is printed.
C) This loop will never end.

Programming Loops Quiz (2 of 3)

When will the programming loop end?

```
while True:
  count = 0
  print(count)
  count = count + 1
  if count = 3:
    break
```

A) After the number 2 is printed
B) After the number 3 is printed.
C) This loop will never end.

Programming Loops Quiz (2 of 3)

When will the programming loop end?

```
while True:
   count = 0
   print(count)
   count = count + 1
   if count = 3:
     break
```

A) After the number 2 is printed
B) After the number 3 is printed.
C) This loop will never end.

Programming Loops Quiz (3 of 3)

When will the programming loop end?

```
count = 0
while count < 5:
  print(count)
  count = count + 1
```

A) After the number 4 is printed.
B) After the number 5 is printed.
C) This loop will never end.

Programming Loops Quiz (3 of 3)

When will the programming loop end?

```
count = 0
while count < 5:
  print(count)
  count = count + 1
```

A) After the number 4 is printed.
B) After the number 5 is printed.
C) This loop will never end.

Part 2: Functions

```
>>> print("Print() is a built-in function in Python.")
Print() is a built-in function in Python.
```

Programming functions are defined instructions that can be used multiple times in the same code.

Functions

```
>>> print("Print() is a built-in function in Python.")
Print() is a built-in function in Python.
```

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.

Functions

```
>>> print("Print() is a built-in function in Python.")
Print() is a built-in function in Python.
```

In fact, we've been making use of the most common Python function this whole time: the **print** function.

Functions

```
>>> print("Print() is a built-in function in Python.")
Print() is a built-in function in Python.
```

Any arguments that the function requires to execute properly (if any) are provided inside the parentheses when calling (running) the function.

Functions

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

To create a function, start with def, then the name of the function, then a pair of parentheses, with any arguments required inside the parentheses, then a colon.

Functions

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

Since a function is a code block, on the next line, indent (4 spaces), then input the function's instructions. If the instructions span multiple lines, each line must be indented. Any argument variables should be included in the instructions.

Functions

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

my_greeting("Shyhat")
```

```
Greetings, Shyhat!
```

Here, our example function is defined, then the function is called, with the corresponding output shown thereafter.

The Return Statement

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

sales_tax_calc(55.00)
```

Return is an instruction that can be used in functions. When used, **return** stores the value specified, but doesn't print it out to the console.

The Return Statement

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

print(sales_tax_calc(55.00))
```

```
63.25
```

So the **return** portion of a function is useful for passing values to other parts of the program, but won't output to the console unless we use the **print** function with it.

Programming Functions Exercise

Let's practice using functions with Python at the following URL:

https://learnpython.org/en/Functions

Functions Quiz (1 of 2)

What will print to the console?

```
def add_num(a, b)
  return a + b

add_num(2, 3)
```

A) The number 5
B) There is a syntax error in the code
C) Nothing will be printed to the console

Functions Quiz (1 of 2)

What will print to the console?

```
def add_num(a, b)
   return a + b

add_num(2, 3)
```

A) The number 5
B) There is a syntax error in the code
C) Nothing will be printed to the console

Functions Quiz (2 of 2)

What will print to the console?

```
my_number = 7
def number_set(num):
   my_number = num
number_set(14)
print(my_number)
```

A) The number 7
B) The number 14

Functions Quiz (2 of 2)

What will print to the console?

```
my_number = 7
def number_set(num):
   my_number = num
number_set(14)
print(my_number)
```

A) The number 7
B) The number 14

Part 3: Objects and Classes

```
my_integer = 1
my_float = 0.25
my_string = "hackerFrogs"

print(type(my_integer),type(my_float),type(my_string))
```

```
(<class 'int'>, <class 'float'>, <class 'str'>)
```

Python is an object-oriented programming (OOP) language, and as such, all pieces of data in Python code are considered objects.

Classes

```
my_integer = 1
my_float = 0.25
my_string = "hackerFrogs"

print(type(my_integer),type(my_float),type(my_string))
```

```
(<class 'int'>, <class 'float'>, <class 'str'>)
```

In addition, all objects in Python are separated into different classes. We see that the three variables here are in the **integer**, **float**, and **string** classes, respectively.

Classes

```
my_string = "The hackerFrogs"

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

```
15
THE HACKERFROGS
```

An object's class indicates which built-in functions, methods and variables it has.

Classes

```
my_string = "The hackerFrogs"

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

```
15
THE HACKERFROGS
```

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

Classes

```
my_string = 1337

print(my_string.upper())
```

```
AttributeError: 'int' object has no attribute 'upper' on line 3 in main.py
```

If we changed the **my_string** value to an integer, executing either the **len** function or **upper** method would result in an error message.

Creating Classes

```
class Student:
    def __init__(self, name, fave_subject):
        self.name = name
        self.fave_subject = fave_subject
```

To create a class, we begin with class, then the name of the class, then a colon. The first letter of the class name is capitalized.

Creating Classes

```
class Student:
    def __init__(self, name, fave_subject):
        self.name = name
        self.fave_subject = fave_subject
```

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.

The __init__ Function

```
class Student:
    def __init__(self, name, fave_subject):
        self.name = name
        self.fave_subject = fave_subject

CryptoGuy = Student("CryptoGuy","Cryptography")
```

One important feature of class creation is defining the **__init__** function, which specifies the attributes of the class.

The Class Self Attribute

```
class Student:
    def __init__(self, name, fave_subject):
        self.name = name
        self.fave_subject = fave_subject

CryptoGuy = Student("CryptoGuy","Cryptography")
```

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.

Classes and Objects Exercise

Let's practice using classes and objects with Python at the following URL:

https://learnpython.org/en/Classes_and_Objects

Classes and Objects Quiz (1 of 2)

Which of the following is statements regarding the relationship between a class and an instance is false?

A) A class is a blueprint for creating objects, an instance is an object belonging to a class

B) Instances of a class have all of the attributes and methods of their class

C) The class variables of instances and their associated class can be different

D) All of the above

Classes and Objects Quiz (1 of 2)

Which of the following is statements regarding the relationship between a class and an instance is false?

A) A class is a blueprint for creating objects, an instance is an object belonging to a class

B) Instances of a class have all of the attributes and methods of their class

C) The class variables of instances and their associated class can be different

D) All of the above

Classes and Objects Quiz (2 of 2)

Which of the following is not considered objects in the Python programming language?

A) Operators, like +, -, /, *, etc, etc

B) Variables, such as integers, lists, strings, etc, etc

C) Keywords, like **if**, **else**, **for**, **while**, etc, etc

D) A and C

Classes and Objects Quiz (2 of 2)

Which of the following is not considered objects in the Python programming language?

A) Operators, like +, -, /, *, etc, etc

B) Variables, such as integers, lists, strings, etc, etc

C) Keywords, like **if**, **else**, **for**, **while**, etc, etc

D) A and C

Workshop Review Exercise

Before we finish, let's a write a program which features many of the concepts we learned in this workshop.

We can use a window in the learnpython.org website to write the program

Workshop Review Exercise

The program should have two lists.

One with four colors: blue, red, green, and yellow.

Another with five fruits: apple, orange, grape, banana, and watermelon.

Workshop Review Exercise

The program should have a function which loops through a list, printing out each item.

The function should be called twice, once for each list.

Workshop Review Exercise

- One list with colors: blue, red, green, and yellow

- Another list with five fruits: apple, orange, grape, banana, and watermelon

- A function which iterates and prints out each item in a list

- Call the function once for each list

Workshop Review Exercise

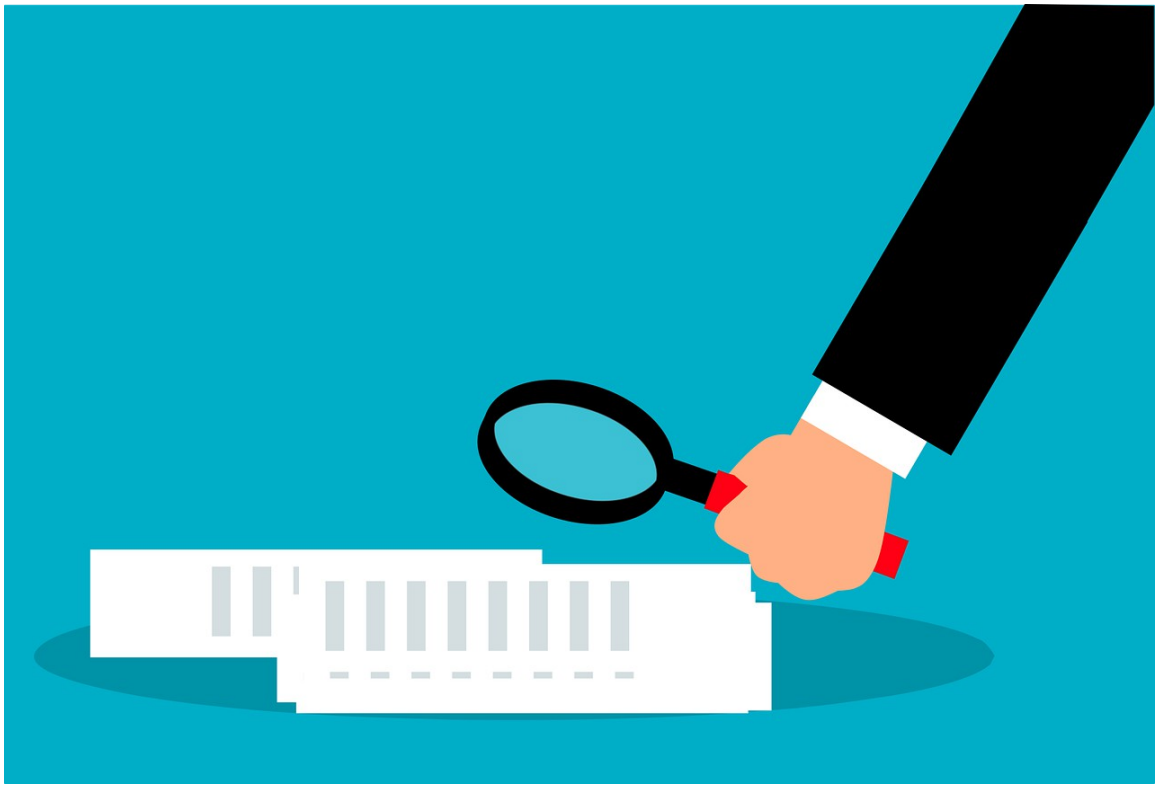
The following code will fulfill the exercise requirements:

```
colors = ['blue', 'red', 'green', 'yellow']
fruits = ['apple', 'orange', 'grape', 'banana', 'watermelon']

def list_out(target):
  for i in target:
    print(i)

list_out(colors)
list_out(fruits)
```

Summary



Let's review the programming concepts we learned in this workshop:

Programming Loops

```
for i in range(3):
    print(i)
```

```
0
1
2
```

Loops are tools that programmers use to run the same instructions multiple times.

Programming Loops

```
for i in range(3):
    print(i)
```

```
0
1
2
```

Loops can either run a preset number of times before terminating, or iterate through all items in a list or string, in the case of **for loops**...

Programming Loops

```
count = 0

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

```
0
1
2
```

Or run continuously until a predetermined trigger state is achieved, in the case of **while loops**.

Functions

```
def greetings():
    print('Hello and good day!')

greetings()
```

```
Hello and good day!
```

Functions are defined blocks of code that can be executed multiple times in the same program.

Functions

```
def greetings():
    print('Hello and good day!')

greetings()
```

```
Hello and good day!
```

Executing a function is usually referred to as “calling a function”, or “a function call”.

Classes & Objects

```
print(type(1), type('word'))
```

```
<class 'int'> <class 'str'>
```

Python is an object-oriented programming language, and as such all data in Python code are considered objects.

Classes & Objects

```
print(type(1), type('word'))
```

```
<class 'int'> <class 'str'>
```

All objects belong to a class, and an object's class indicates which built-in functions, methods, and variables it has.

What's Next?

In the next HackerFrogs Afterschool programming workshop, we'll conclude our intro to Python with the learnpython.org website.

What's Next?

Next workshop topics:

- Dictionaries

- Modules and Packages

What's Next?

Next workshop topics:

- Dictionaries

- Modules and Packages

What's Next?

Next workshop topics:

- Dictionaries

- Modules and Packages

Extra Credit

Looking for more study material on this workshop's topics?

See this video's description for links to supplemental documents and exercises!

Until Next Time, HackerFrogs!