HackerFrogs Afterschool
Python Programming Basics: Part 5

Class:
Programming (Python)

Workshop Number:
AS-PRO-PY-05

Document Version:
1.75

Special Requirements:
Completion of AS-PRO-PY-04

What We Learned Before

This workshop is the fifth intro class to Python programming.

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

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

- Loops (in general)

- For Loops

- While Loops

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.

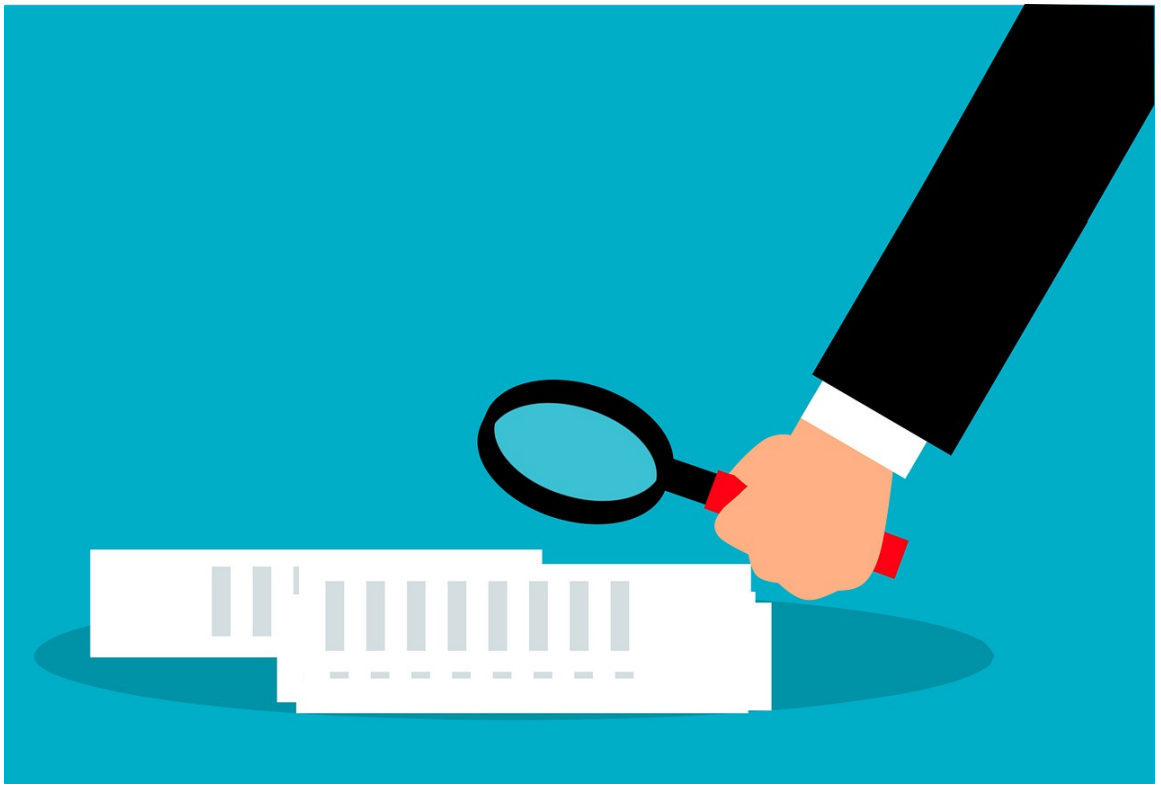
Programming Loops Quiz (3 of 3)

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.

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**.

What's Next?

In the next HackerFrogs Afterschool programming workshop, we'll continue learning Python with the learnpython.org website.

What's Next?

Next workshop topic:

- Functions

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!