HackerFrogs Afterschool
Python Programming Basics: Part 4

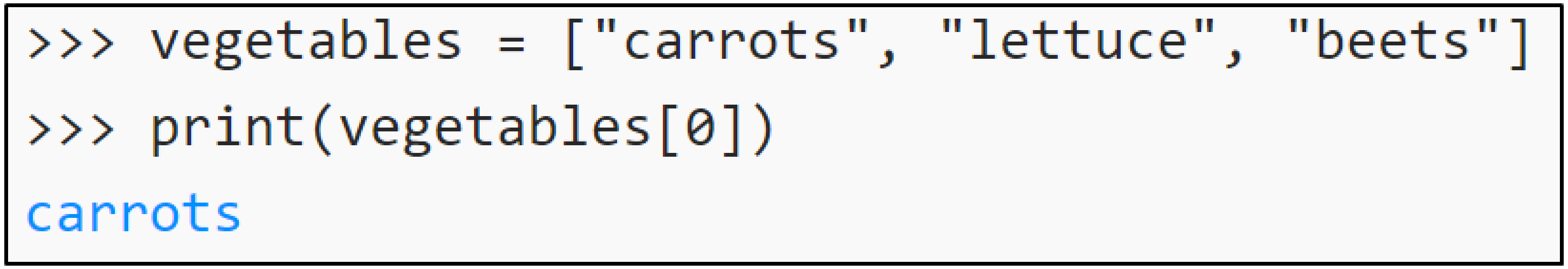
```
Class:
Programming (Python)

Workshop Number:
AS-PRO-PY-04

Document Version:
1.0

Special Requirements:
Completion of AS-PRO-PY-03
```

What We Learned Before

This workshop is the fourth intro class to Python programming.

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

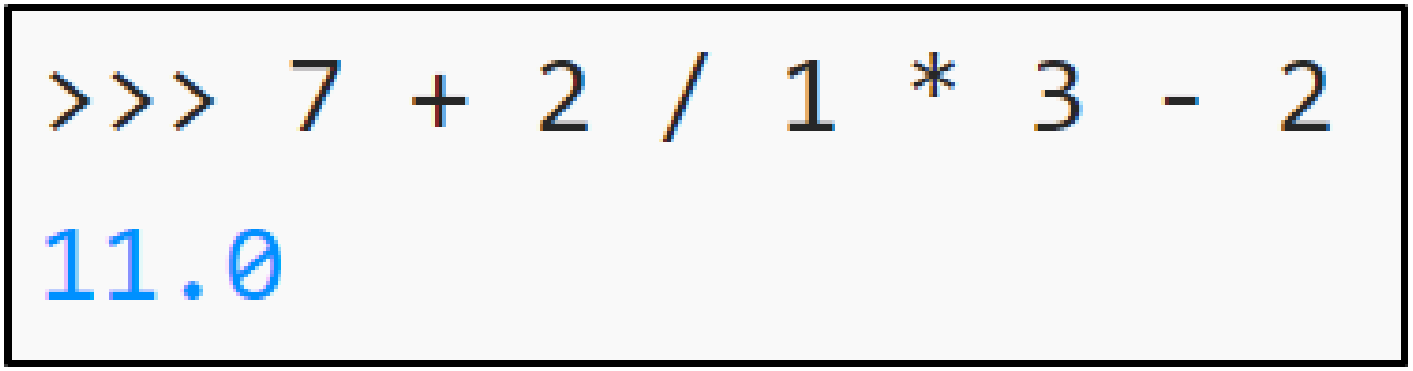
Lists



```
>>> vegetables = ["carrots", "lettuce", "beets"]
>>> print(vegetables[0])
carrots
```

We learned about lists, which are variables with one or more items associated with them. Items in the list can be referenced by a process called **list indexing**.

Basic Operators



```
>>> 7 + 2 / 1 * 3 - 2
11.0
```

The last thing we learned in the previous workshop is Python basic operators, which are used to perform arithmetic operations.

This Workshop's Topics

- Part 1: Conditions

- Part 2: Programming Loops

Conditions

Conditions are how programs make decisions as to which instructions to perform at a specific point in program execution.

Conditions

Generally, conditions trigger when a statement evaluates to **True** or **False**, so let's discuss what this means, exactly.

Conditions

The basic conditional statement in Python is the **if** statement, and **if** statements are written in code blocks like the following:

Conditions

```
password = 'mysecretpassword'

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

Here, the program will print one message if the password variable is the string 'mysecretpassword' (Boolean True), and print another message if it is not (Boolean False).

Conditions

```
password = 'mysecretpassword'

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

If statements start with if, then a Boolean equation, then a semicolon.

Conditions

```
password = 'mysecretpassword'

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

Then an indentation on the next line, followed by instructions if the condition is met, then on the next line else:, then an indent on the next line, followed by instructions if the condition is not met.

Conditions

```
password = 'mysecretpassword'

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

print('Was your password correct?')
```

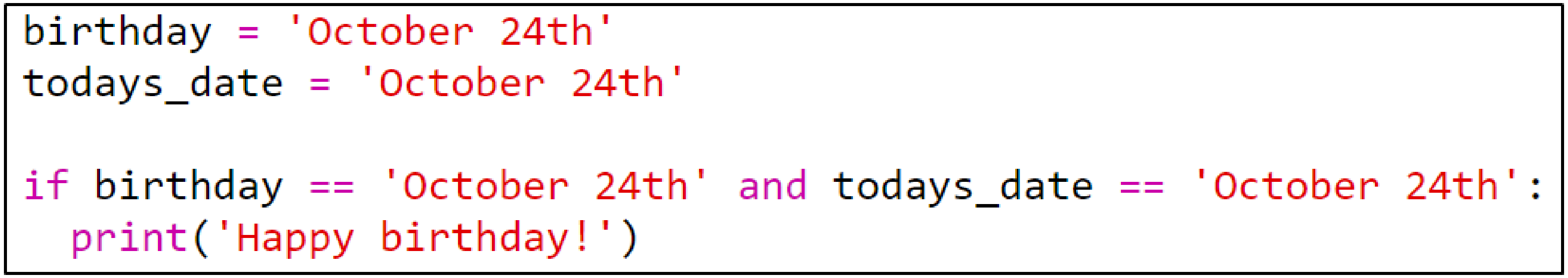
The **else** portion of the code block is optional. After the **if** statement code block finishes, program execution continues on the next line.

Boolean Operators

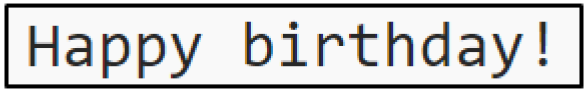
Operator	Meaning
==	Equal to
<	Less than
>	Greater than
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to

We can use any of the above Boolean operators in If Statements.

The AND Operator



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

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



```
Happy birthday!
```

When using the AND operator, both statements to the right and left of the operator must be True in order for the overall statement to evaluate to True.

The AND Operator

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

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

```
Happy birthday!
```

In this case, since the variables **birthday** and **todays_date** are the same day (October 24th), the **if** statement returns True, and the program will print **Happy Birthday!**

The OR Operator

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

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

```
Let's take a vacation to Italy!
```

When using the OR operator either statement to the left or right of the operator must evaluate True for the overall statement to evaluate to True.

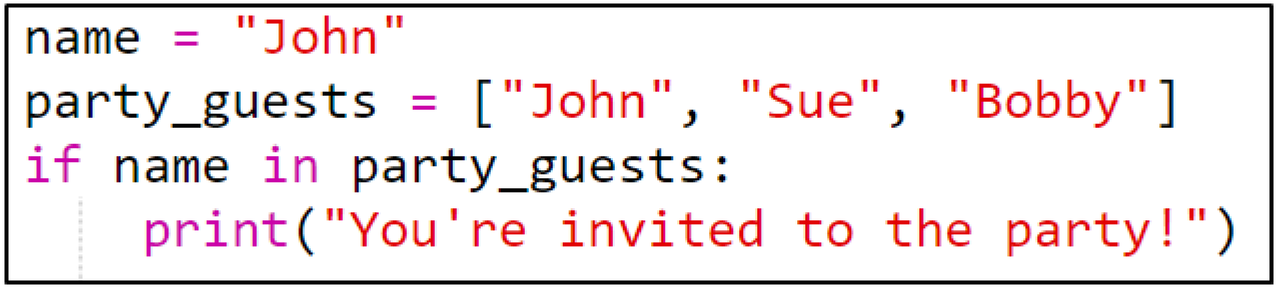
The OR Operator

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

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

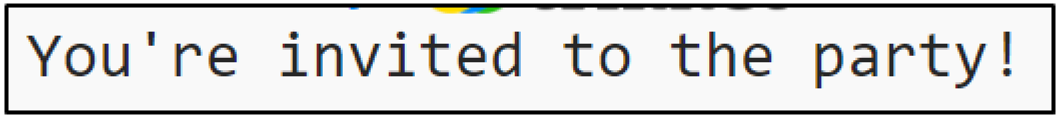
```
Let's take a vacation to Italy!
```

In this example, the if condition would not trigger if **both** statements to the left and right of the OR operator were False.

The IN Operator



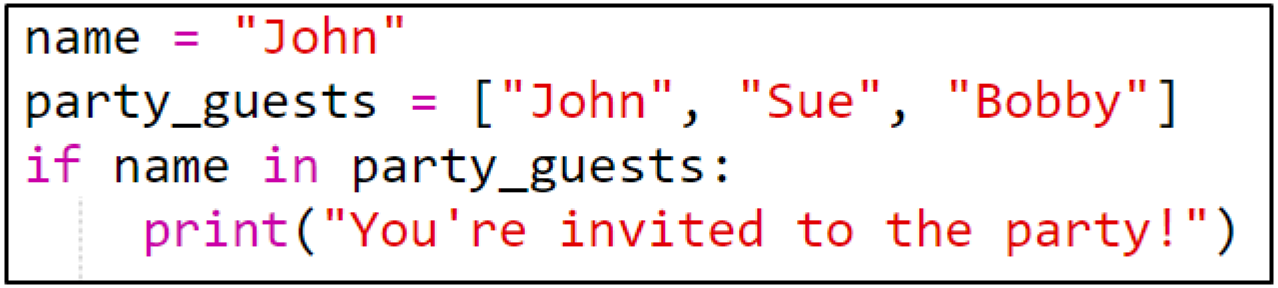
```
name = "John"
party_guests = ["John", "Sue", "Bobby"]
if name in party_guests:
    print("You're invited to the party!")
```



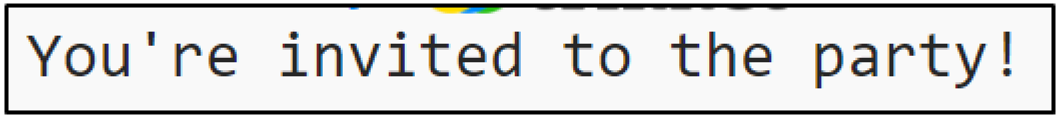
```
You're invited to the party!
```

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

The IN Operator



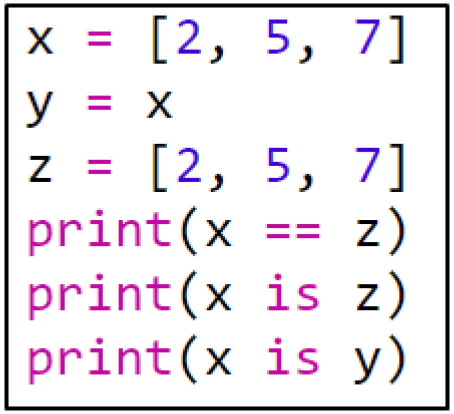
```python
name = "John"
party_guests = ["John", "Sue", "Bobby"]
if name in party_guests:
    print("You're invited to the party!")
```



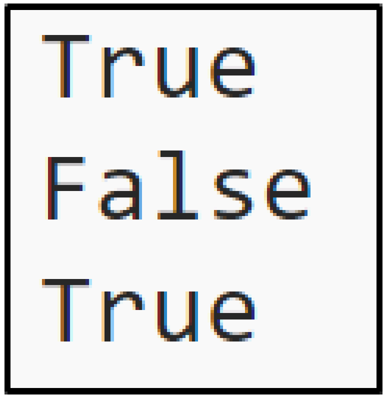
```
You're invited to the party!
```

Here the print instruction is executed because the string listed in the **name** variable is included as an item in the **party_guests** list.

The IS Operator



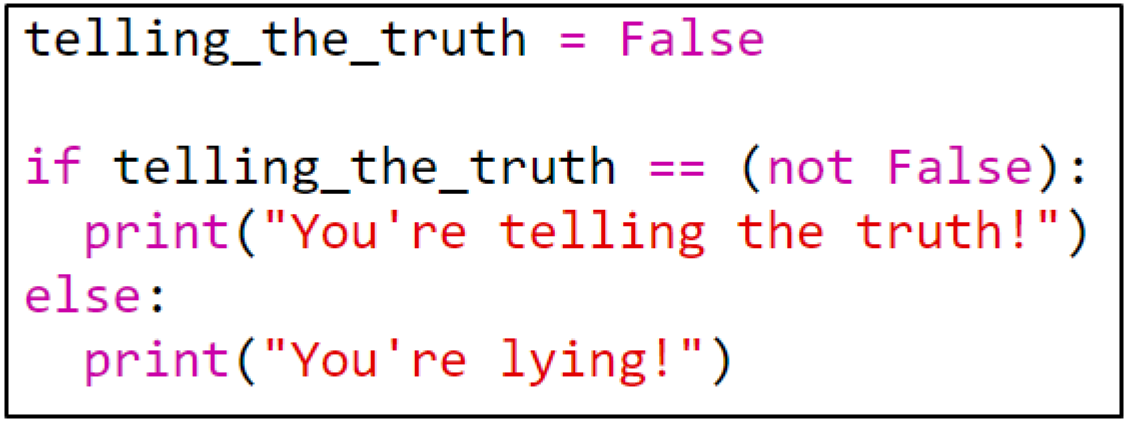
```python
x = [2, 5, 7]
y = x
z = [2, 5, 7]
print(x == z)
print(x is z)
print(x is y)
```



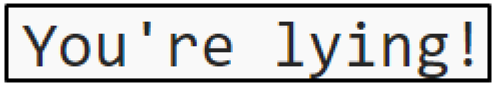
```
True
False
True
```

The IS operator is used to compare whether an object is exactly referencing another object, as opposed to matching the values contained within.

The NOT Operator



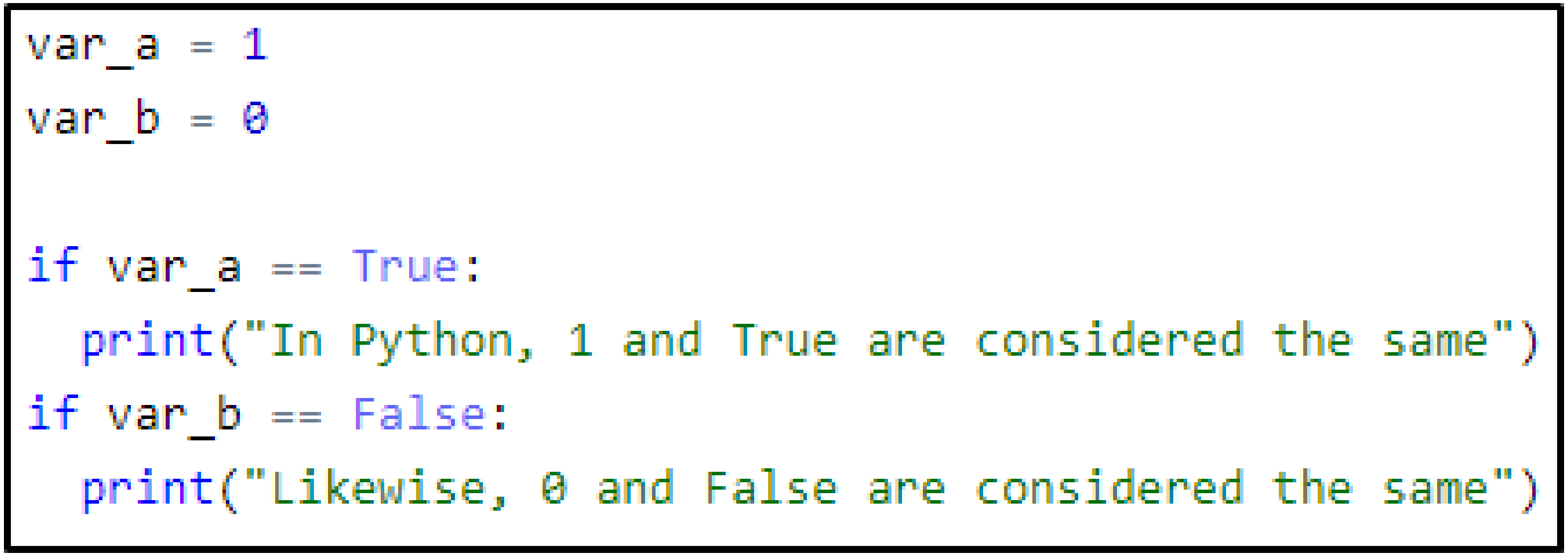
```python
telling_the_truth = False

if telling_the_truth == (not False):
  print("You're telling the truth!")
else:
  print("You're lying!")
```



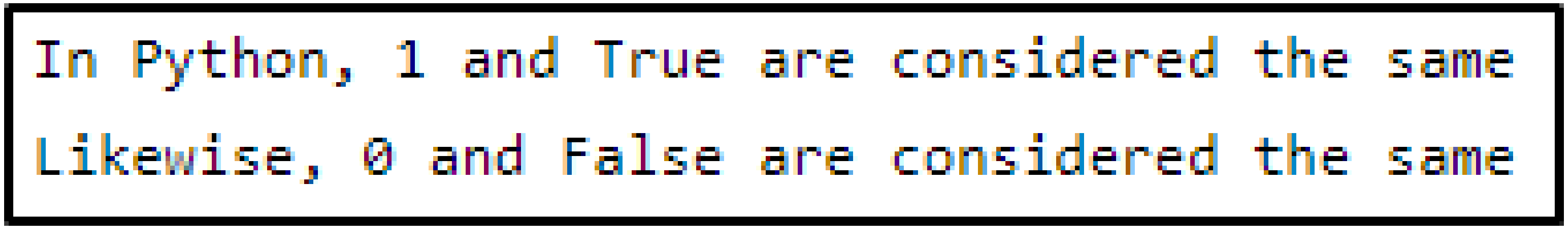
```
You're lying!
```

The NOT operator is essentially an inversion of the whatever Boolean value is paired with it.

True is 1 and False is 0



```python
var_a = 1
var_b = 0

if var_a == True:
  print("In Python, 1 and True are considered the same")
if var_b == False:
  print("Likewise, 0 and False are considered the same")
```



```
In Python, 1 and True are considered the same
Likewise, 0 and False are considered the same
```

Conditionals Exercise

Let's practice our if conditionals in Python at the following URL:

https://learnpython.org/en/Conditions

Part 1 – Conditionals Quiz

What will the output be?

```
hacking = 1
if hacking == True:
  print('We're hacking!')
```

A) We're hacking!
B) True
C) Nothing will print, hacking does not equal True
D) Nothing will print, the syntax is incorrect

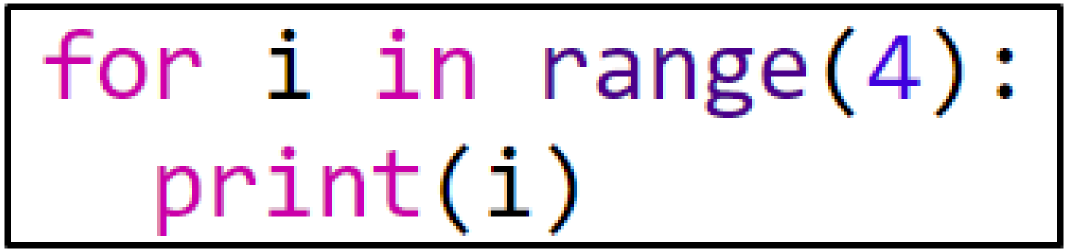
Part 1 – Conditionals Quiz

What will the output be?

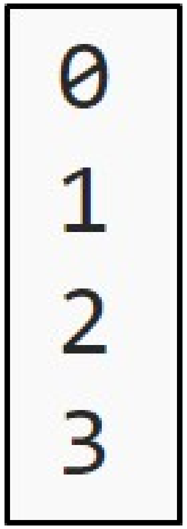
```
hacking = 1
if hacking == True:
  print('We're hacking!')
```

A) We're hacking!
B) True
C) Nothing will print, hacking does not equal True
D) Nothing will print, the syntax is incorrect

Part 1: Programming Loops



```python
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.

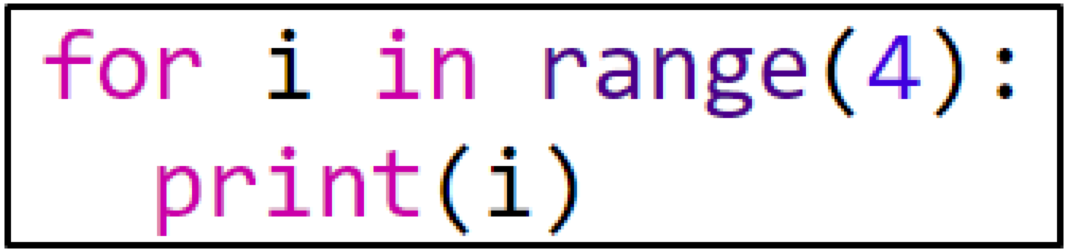
Programming Loops

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

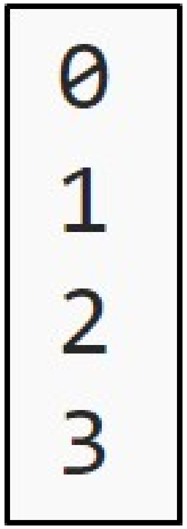
```
0
1
2
3
```

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



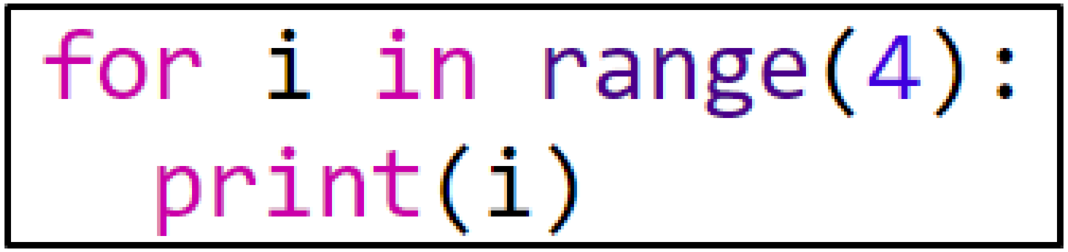
```python
for i in range(4):
  print(i)
```



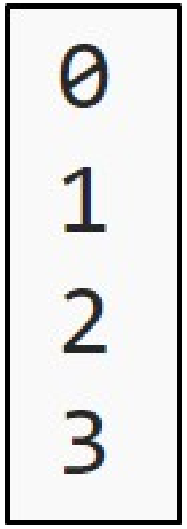
```
0
1
2
3
```

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

“For” Loops



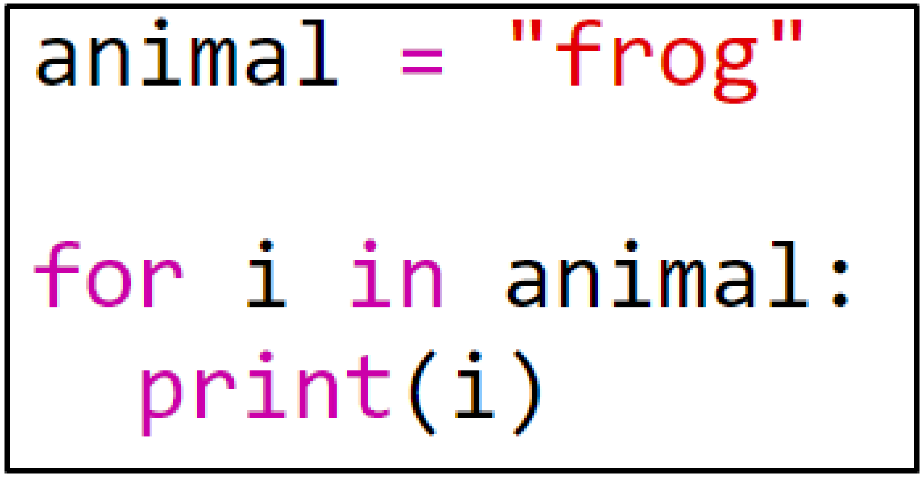
```python
for i in range(4):
    print(i)
```



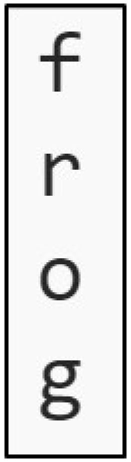
```
0
1
2
3
```

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)
```

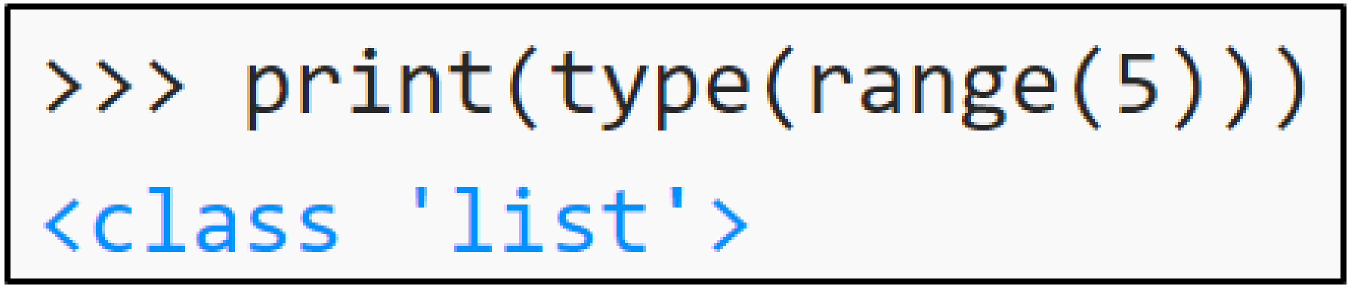
To write a **for loop**, begin with for, then a variable name that is only used inside the loop, typically the letter i, then in, then the name of the object to be iterated over, then a colon.

“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.

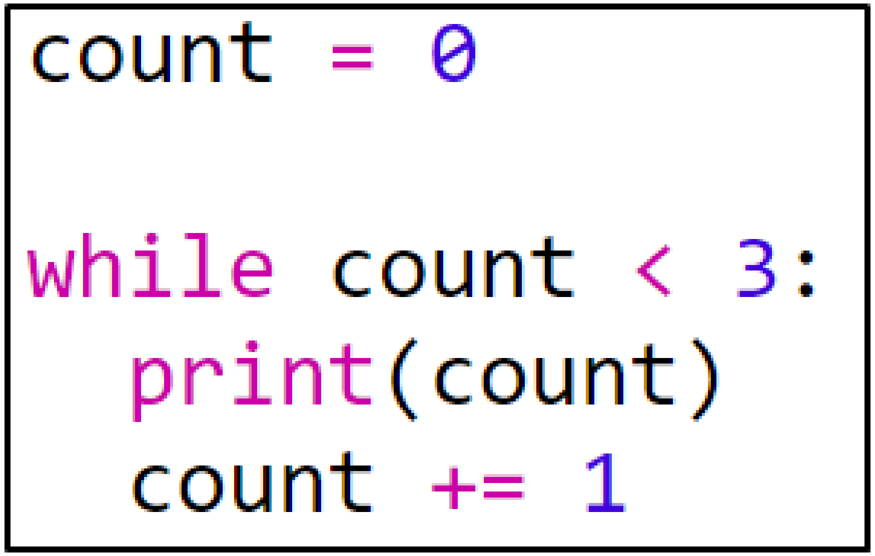
“For” Loops



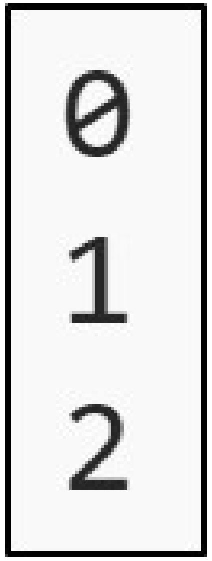
```
>>> print(type(range(5)))
<class 'list'>
```

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



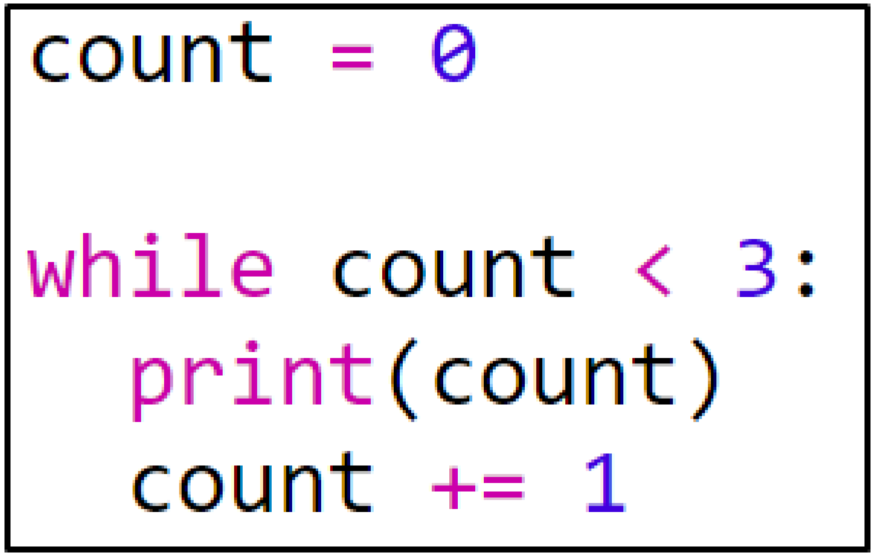
```python
count = 0

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



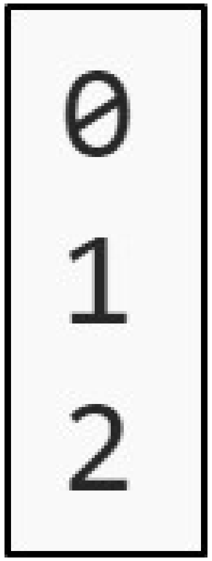
```
0
1
2
```

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.

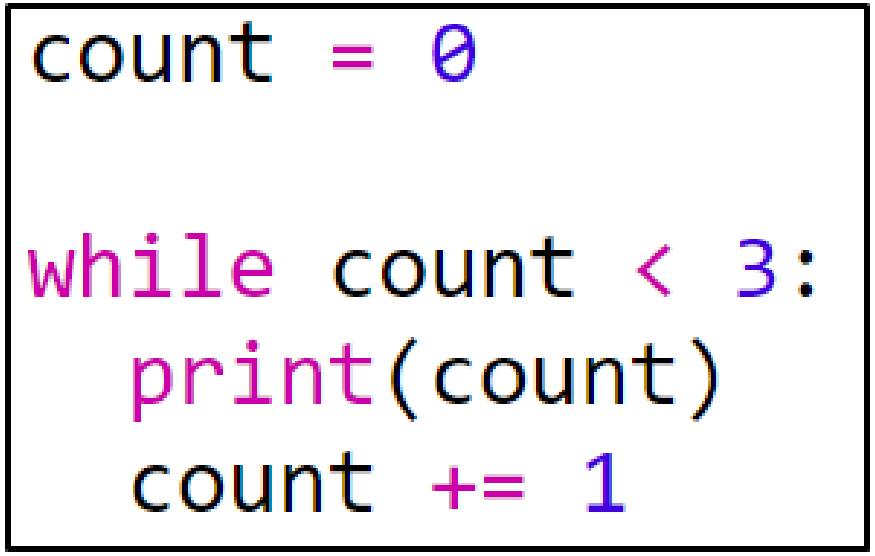
“While” Loops

```
count = 0

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

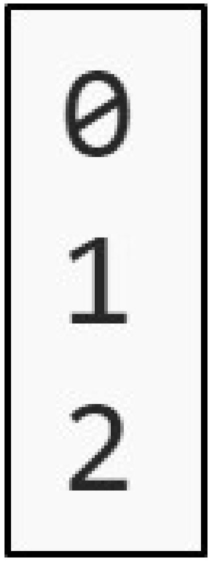
```
0
1
2
```

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
```

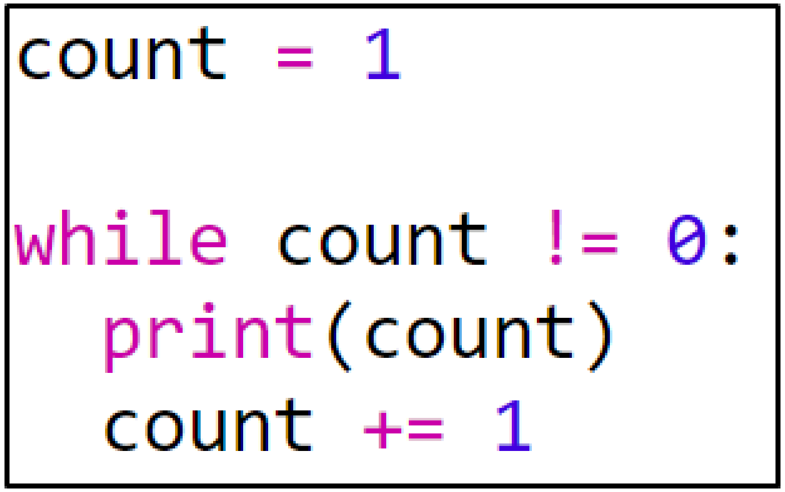
We begin a while loop with while, then the statement that must evaluate True for the loop to execute, then a colon.

“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.

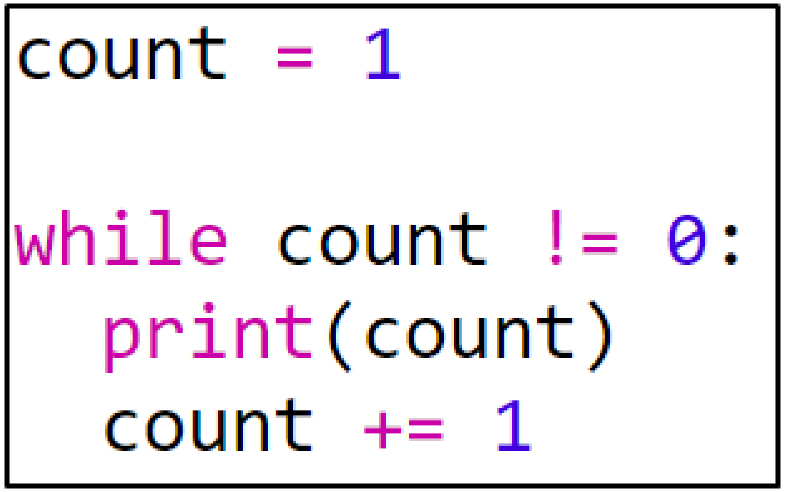
Infinite Loops



```python
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**.

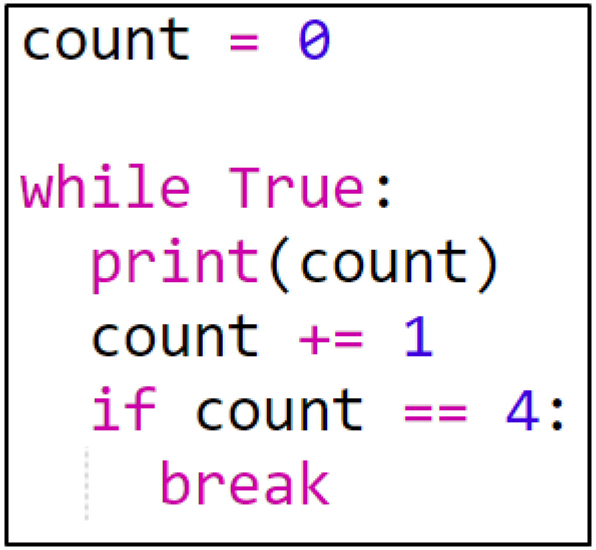
Infinite Loops



```
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.

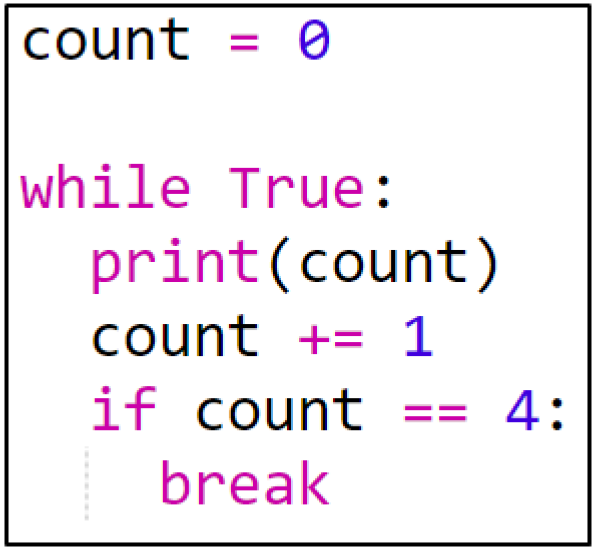
The Break Statement



```python
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.

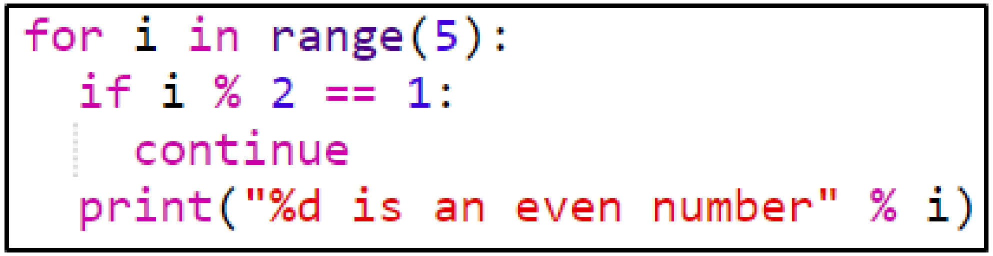
“While True” Loops



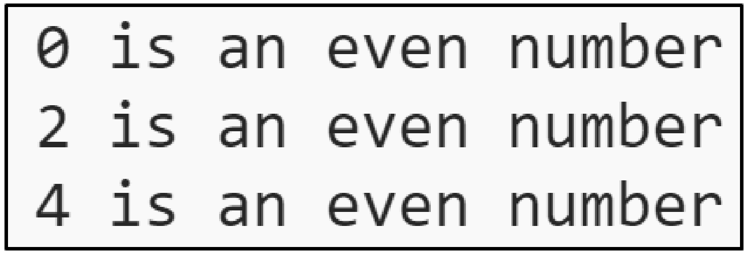
```
count = 0

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



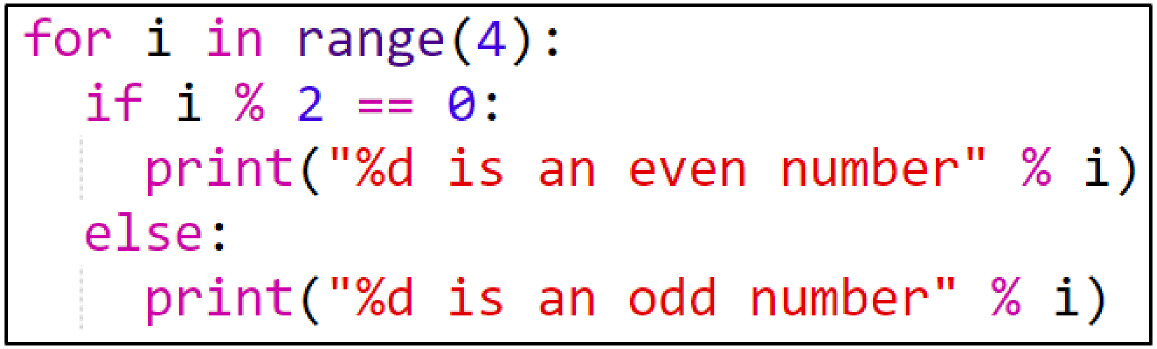
```python
for i in range(5):
  if i % 2 == 1:
    continue
  print("%d is an even number" % i)
```



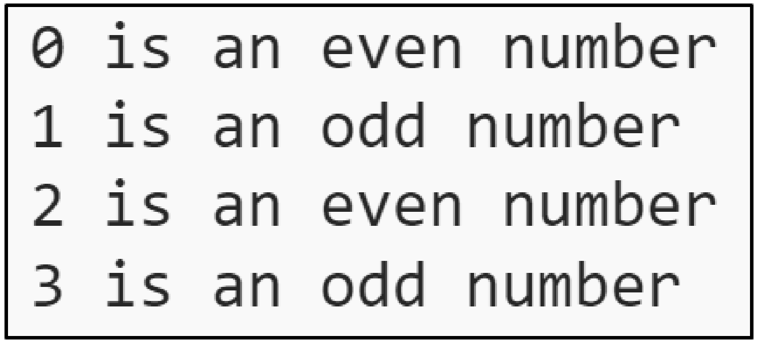
```
0 is an even number
2 is an even number
4 is an even number
```

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



```python
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.

Workshop Review Exercise

Let's write a program that uses some of the concepts we learned in this workshop.

We could write the program on the online-python.com webpage.

The program should take one user-provided input in the form of an integer

Workshop Review Exercise

The program should print out the string

X is an even number!
if the variable is an even number, or

X is an odd number!
if the variable is an odd number.

The value of the variable should substitute in for X in the printed statement. E.g., '7 is an odd number!'

Workshop Review Exercise

The program should be run three times, with the value of the variable set as 0, 13, and 256 for each execution, respectively.

The program should report 0 and 256 as even numbers, and 13 as an odd number.

Workshop Review Exercise

We'll present a possible solution after this, but you should attempt to write your own program now. Remember, the modulo % operation divides a number and returns its remainder!

```
number = ?
if (number % ?) == ?
  print(f'{number} is an even number!')
```

Workshop Review Exercise

- a variable named **number**

- an if statement that prints out '**X is an even number!**' if the **number** variable is an even number

- an if statement that prints out '**X is an odd number!**' if the **number** variable is an odd number

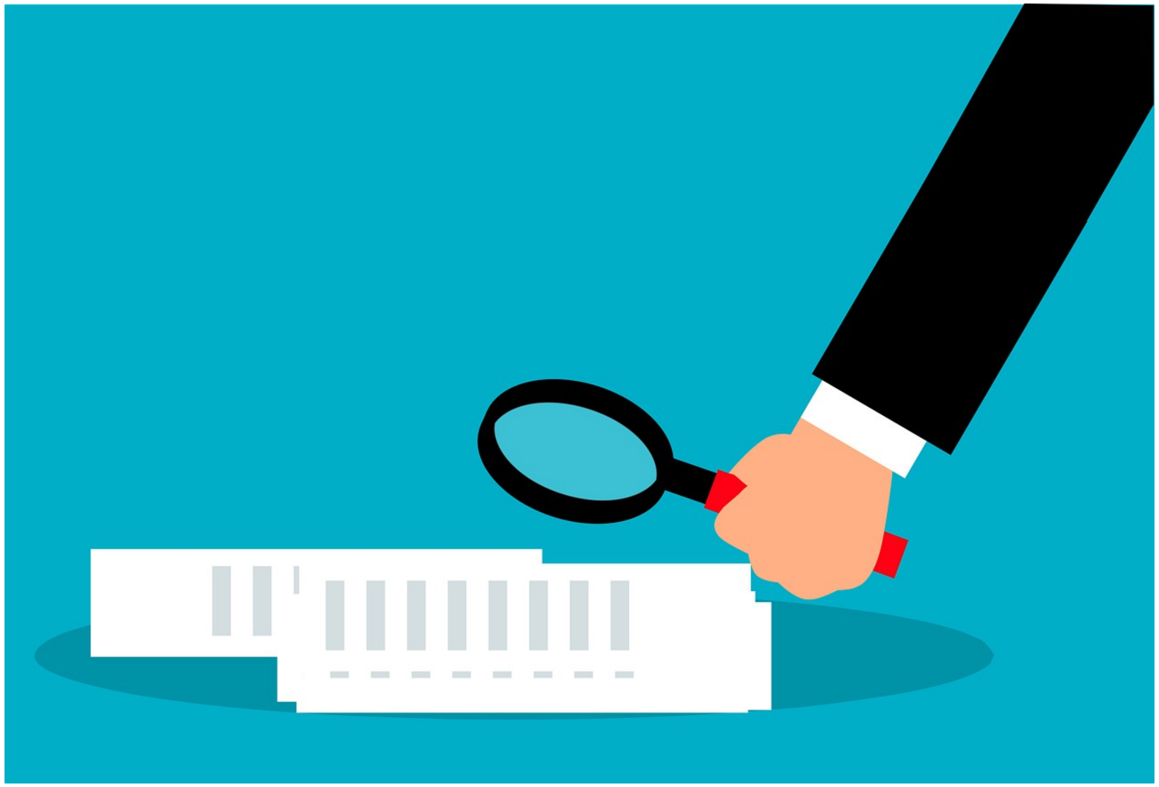
- run the program three times, with the **number** variable set to 0, 13, and 256, respectively

Workshop Review Exercise

A possible solution to this exercise is the following:

```
number = input('Enter a number:')

if (number % 2) == 0:
  print(f'{number} is an even number!')
if (number % 2) == 1:
  print(f'{number} is an odd number!')
```

Summary



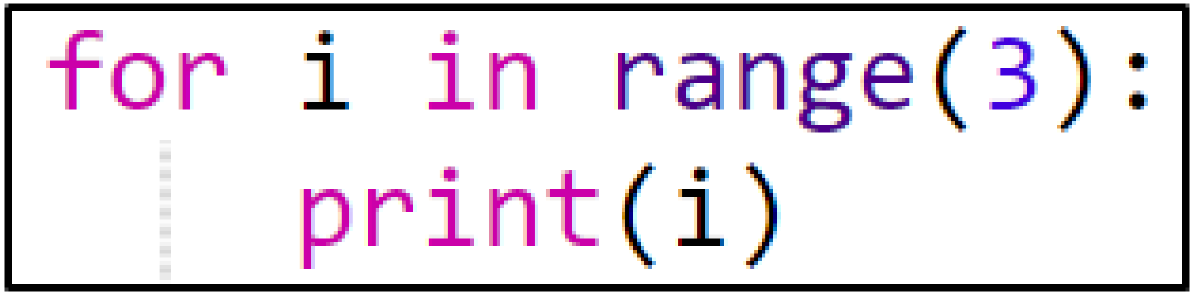
Let's review the programming concepts we learned in this workshop:

Conditions

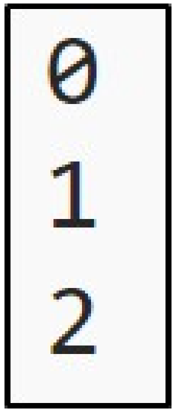
```
password = 'mysecretpassword'

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.

Programming Loops



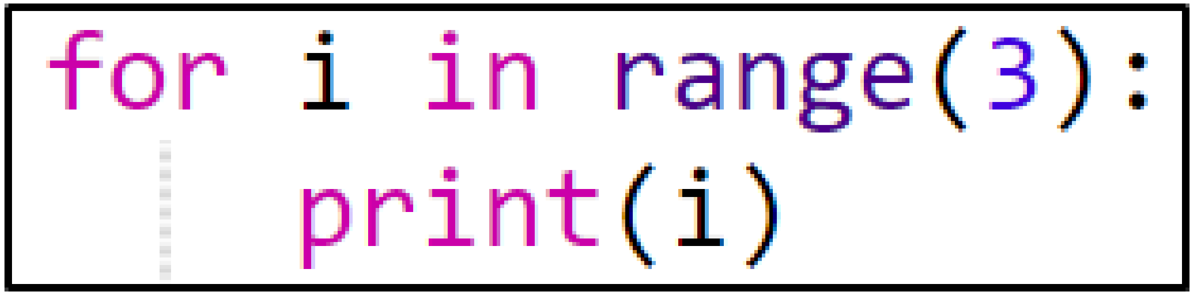
```python
for i in range(3):
    print(i)
```



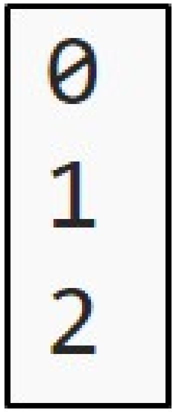
```
0
1
2
```

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 (big topic)

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!