HackerFrogs Afterschool
Python Programming Basics: Part 3

Class:
Programming (Python)

Workshop Number:
AS-PRO-PY-03

Document Version:
1.2

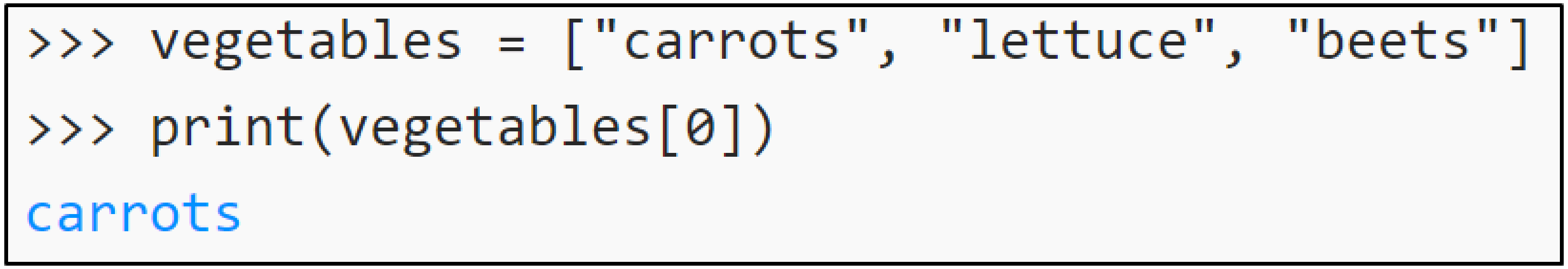
Special Requirements:
Completion of AS-PRO-PY-02

What We Learned Before

This workshop is the third 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

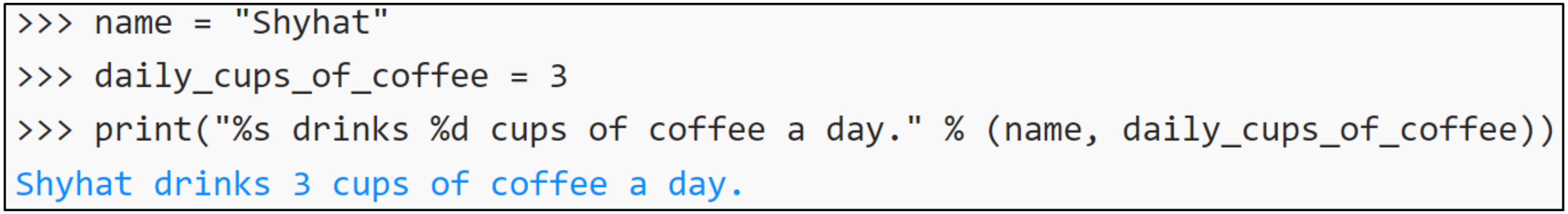
And the other thing we learned in this workshop is Python basic operators, which are used to perform arithmetic operations.

This Workshop's Topics

- Part 1: String Formatting

- Part 2: Basic String Operations

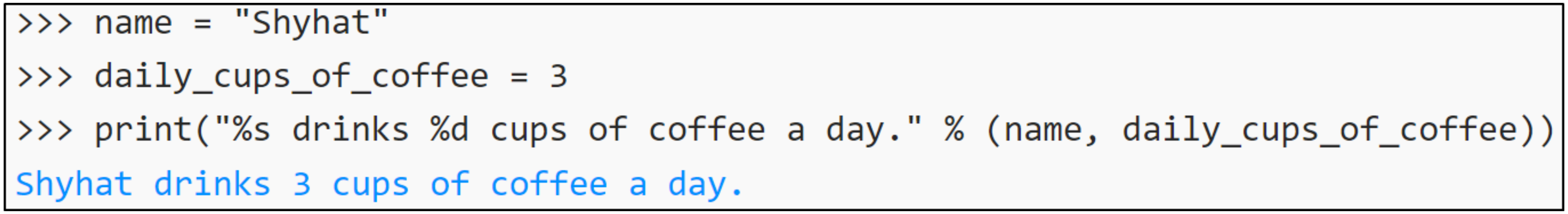
Part 1: String Formatting



```
>>> 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 lets us use variables in strings, allowing for dynamic print output.

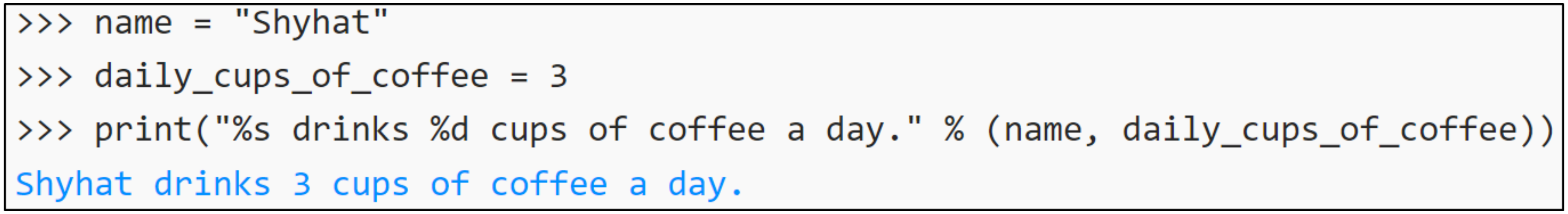
String Formatting



```
>>> 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.
```

When formatting in this way, we use the **%** symbol followed by a letter as a placeholder for the variable in the string.

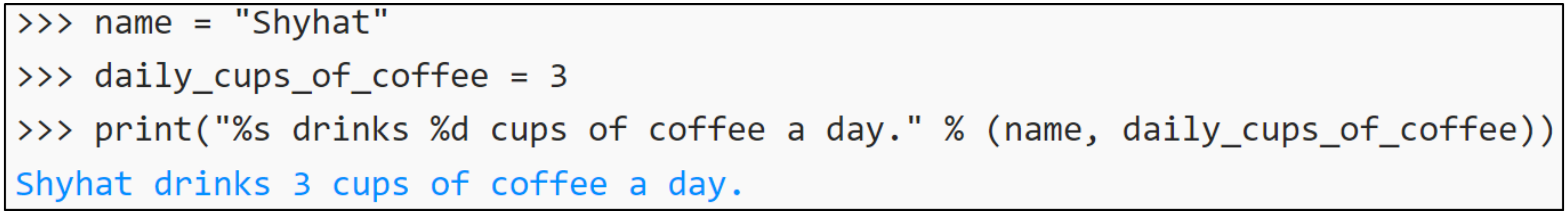
String Formatting



```
>>> 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.
```

After the string, we include the **%** symbol again, then in parentheses, we specify the names of variables in the order they appear in the string.

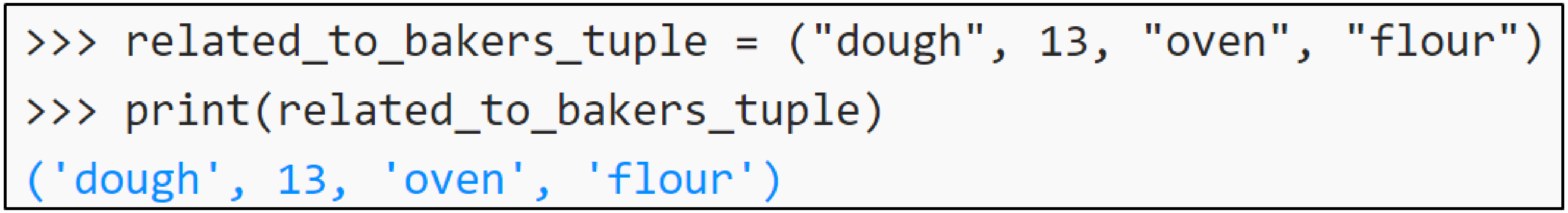
String Formatting



```
>>> 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.
```

Note that inside of the string, we must pair the **%** symbol with a letter, depending on the data type of the variable we're inserting. **%s** for strings, **%d** for integers, and **%f** for floats.

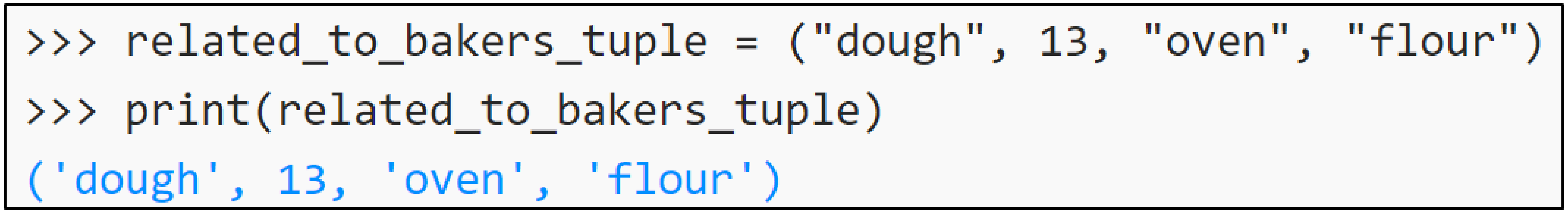
Tuples



```
>>> related_to_bakers_tuple = ("dough", 13, "oven", "flour")
>>> print(related_to_bakers_tuple)
('dough', 13, 'oven', 'flour')
```

Before doing the exercise for this section, let's quickly go over tuples, since they appear in the exercise. In Python, tuples are very similar to lists, with one important difference:

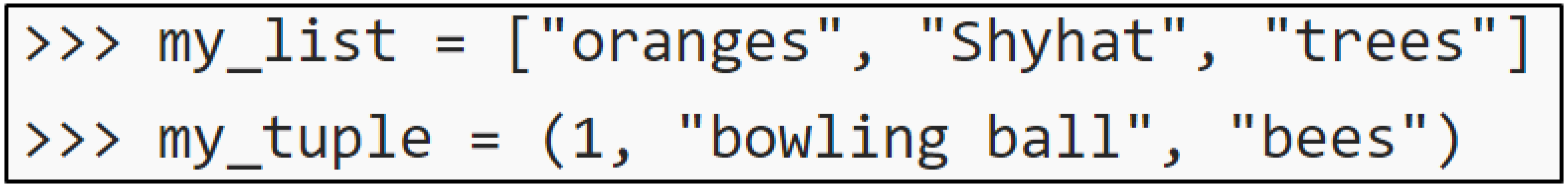
Tuples



```
>>> related_to_bakers_tuple = ("dough", 13, "oven", "flour")
>>> print(related_to_bakers_tuple)
('dough', 13, 'oven', 'flour')
```

Whereas lists in Python can be modified during program execution (mutable), tuples are fixed, and cannot be modified during program execution (immutable).

Tuples



```
>>> my_list = ["oranges", "Shyhat", "trees"]
>>> my_tuple = (1, "bowling ball", "bees")
```

Visually, we can distinguish between lists and tuples by the use of square brackets or parentheses, respectively.

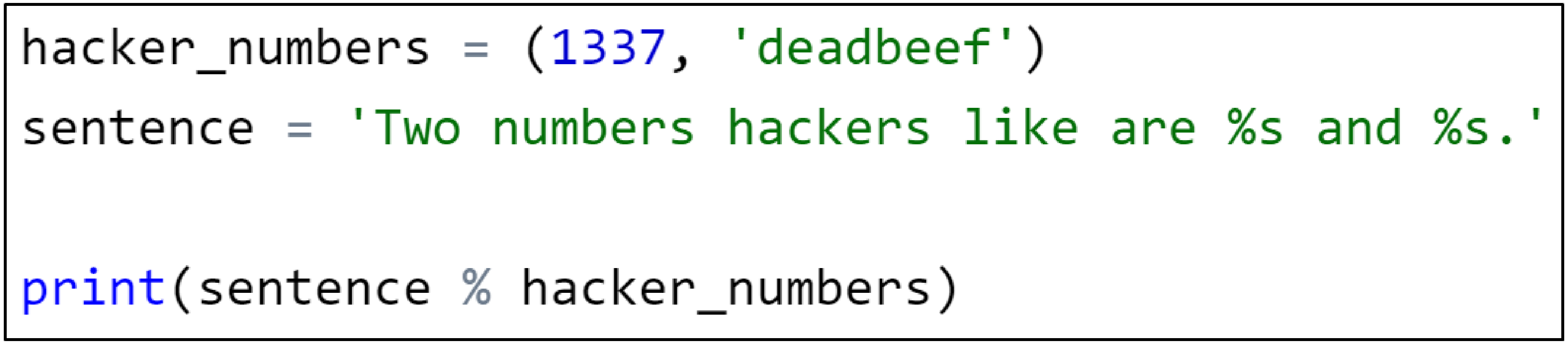
Tuples + String Formatting

```
hacker_numbers = (1337, 'deadbeef')

print('Two numbers hackers like are %s and %s.' % (hacker_numbers))
```

We can also use a tuple variable to store all of the data we want to include in our string.

Tuples + String Formatting



```python
hacker_numbers = (1337, 'deadbeef')
sentence = 'Two numbers hackers like are %s and %s.'

print(sentence % hacker_numbers)
```

We can also use a tuple variable to store all of the data we want to include in our string, then the formatted sentence as another variable.

String Formatting Exercise

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

https://learnpython.org/en/String_Formatting

Part 1 – String Formatting Quiz - Q1

Which of the following Python code will print `HackerFrogs Python workshop` ?

A) print("%s %s workshop" % ['HackerFrogs', 'Python'])
B) print("%s %s workshop" % ('HackerFrogs', 'Python'))
C) print("%s %d workshop" % ('HackerFrogs', 'Python'))
D) print 'HackerFrogs Python workshop'

Part 1 – String Formatting Quiz - Q1

Which of the following Python code will print `HackerFrogs Python workshop` ?

A) print("%s %s workshop" % ['HackerFrogs', 'Python'])
B) print("%s %s workshop" % ('HackerFrogs', 'Python'))
C) print("%s %d workshop" % ('HackerFrogs', 'Python'))
D) print 'HackerFrogs Python workshop'

Part 1 – String Formatting Quiz - Q2

Which of the following Python code is a valid method of printing variables in strings?

```
name = 'HackerFrog'
```

A) `print(f'Hello {name}!')`
B) `print('Hello %s!' % name)`
C) `print('Hello ' + name + '!')`
D) all of the above

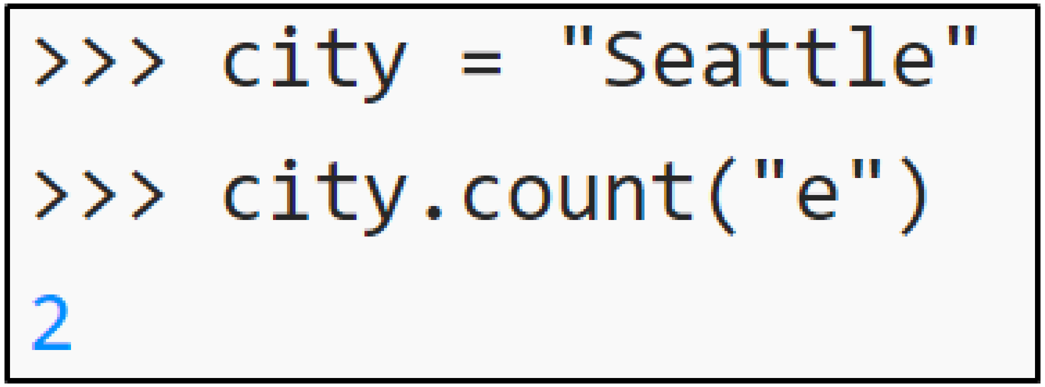
Part 1 – String Formatting Quiz - Q2

Which of the following Python code is a valid method of printing variables in strings?

```
name = 'HackerFrog'
```

A) `print(f'Hello {name}!')`
B) `print('Hello %s!' % name)`
C) `print('Hello ' + name + '!')`
D) all of the above

Part 2: Basic String Operations



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

Strings in Python are considered an object **class**, and classes in Python have special operations, called **methods**, that can be applied to those class objects.

Basic String Operations

```
city.count('t')
```

Use of methods in Python code can be identified by the name of the object, followed by a . (dot), then the name of the method, then parentheses, and any arguments the method requires inside of the parentheses.

Basic String Operations

```
len('HackerFrogs')
```

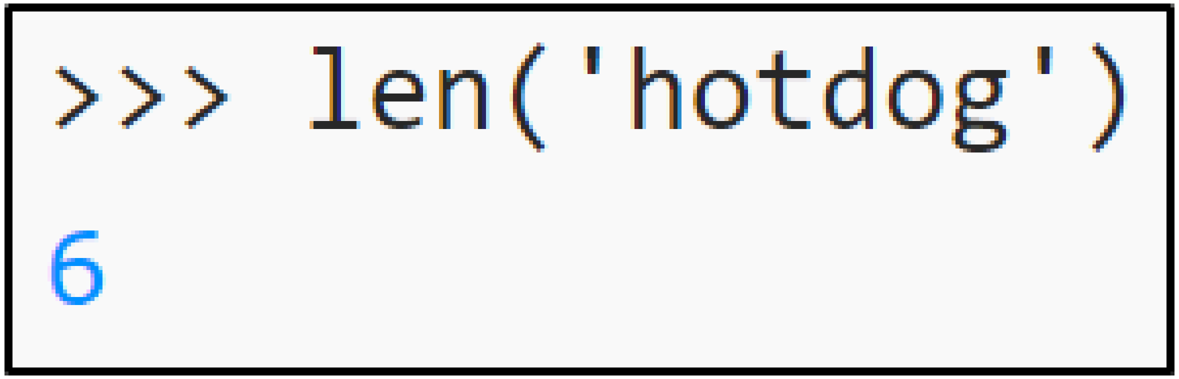
Use of functions in Python code can be identified by the name of the function, followed by parentheses, and any arguments the function requires inside of the parentheses.

Basic String Operations

```
len('HackerFrogs')
```

Note that, in both functions and methods, inclusion of parentheses are required, even if there are no arguments required for the function or method to execute.

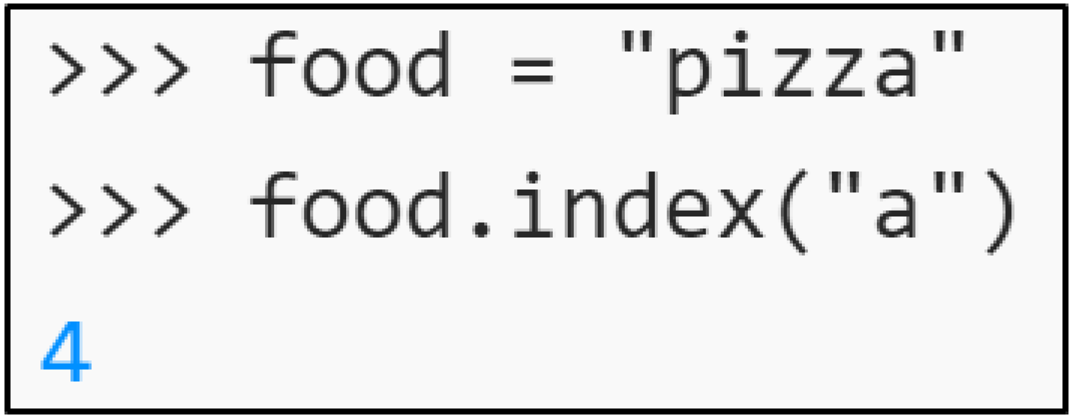
The Len Function



```
>>> len('hotdog')
6
```

The **len** function is used to count the number of indexes contained in an object. In the case of strings, it returns the number of characters, and in the case of a list, it returns the number of items.

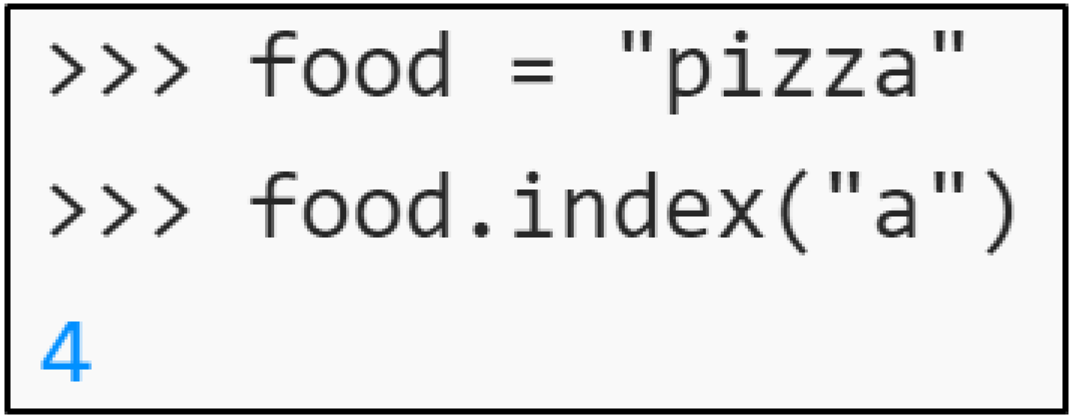
The Index Method



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

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).

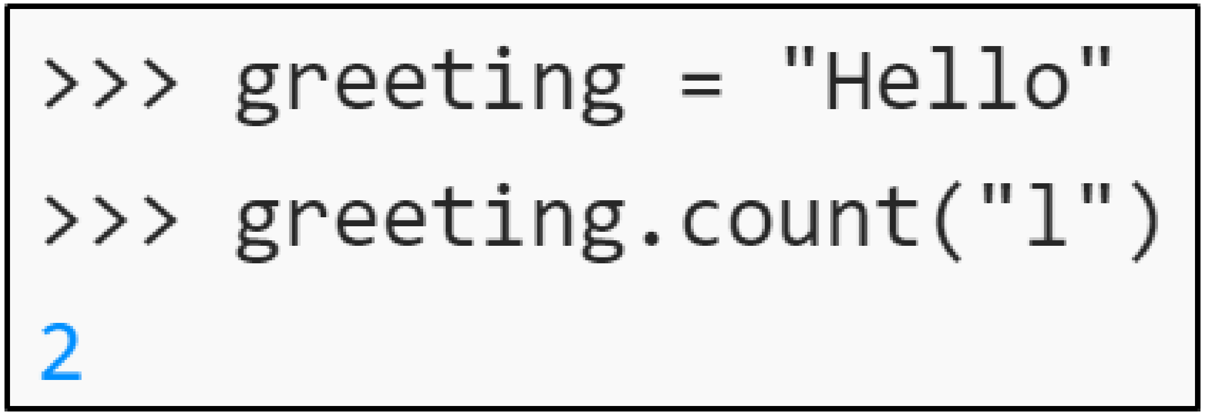
The Index Method



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

Here, the output of the method is 4, because we're starting our count from zero.

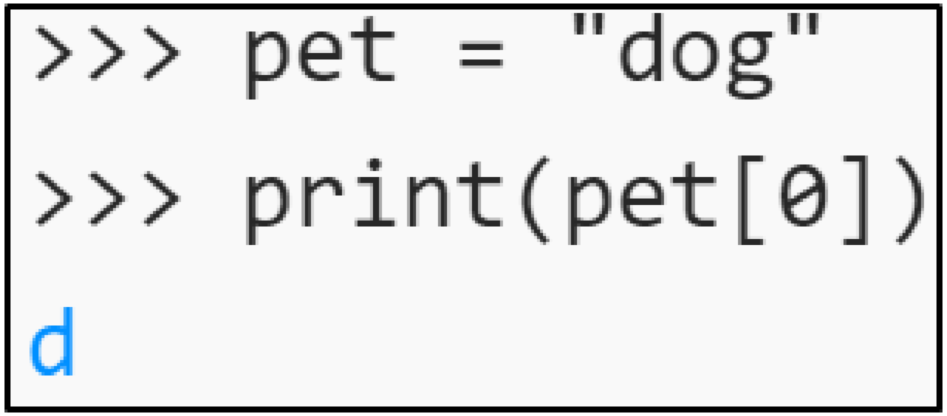
The Count Method



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

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.

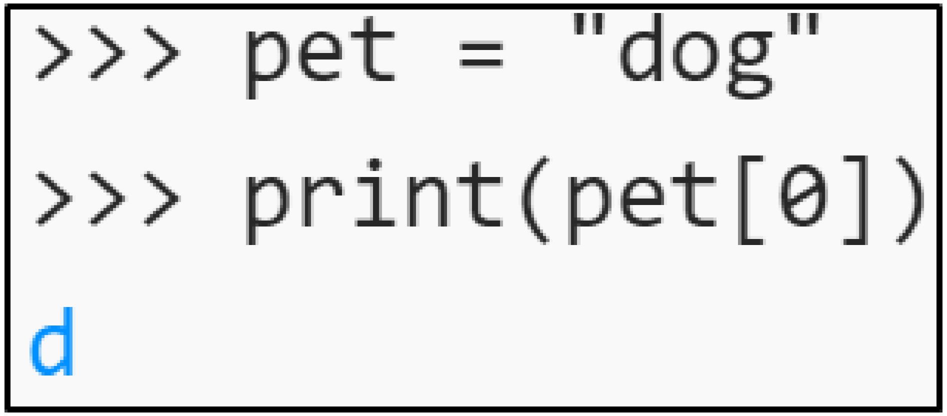
String Index Slicing



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

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.

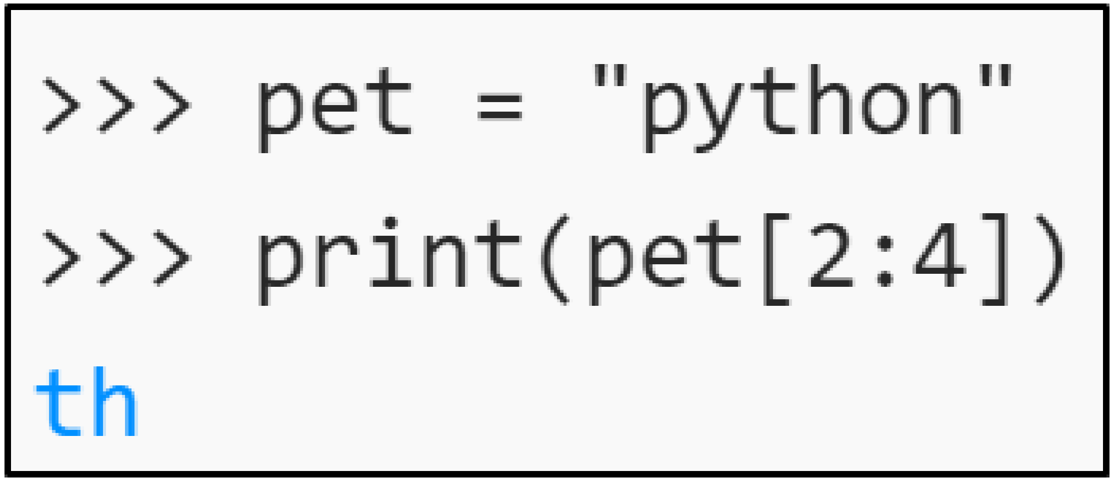
String Index Slicing



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

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

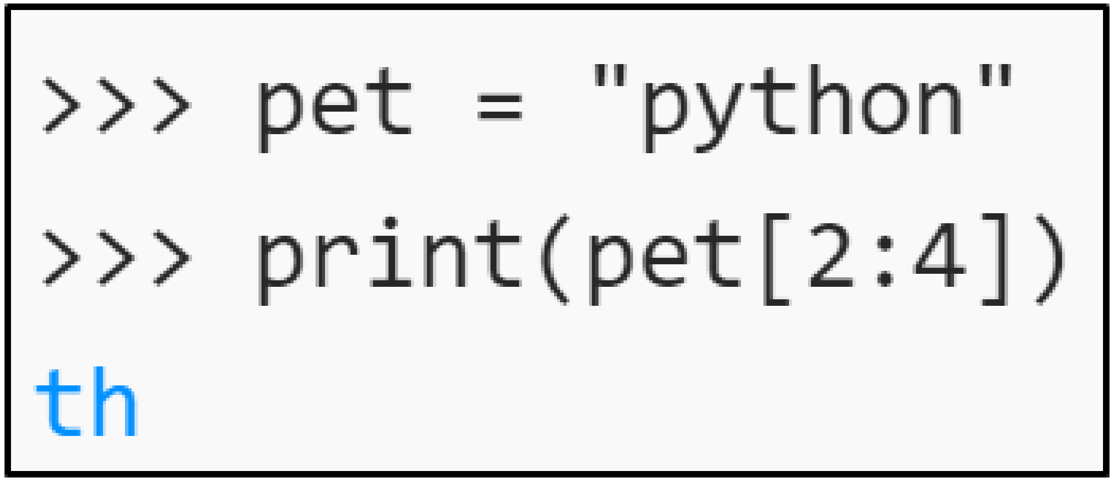
String Index Slicing



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

What's more interesting is the ability to select specific slices of the string. Here, we've isolated the third and fourth letters from the string 'python'.

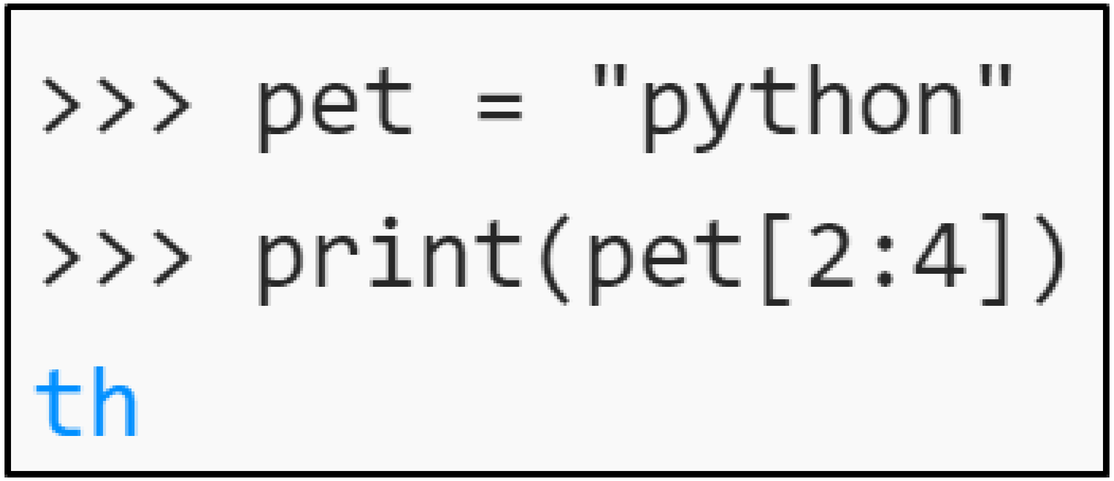
String Index Slicing



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

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).

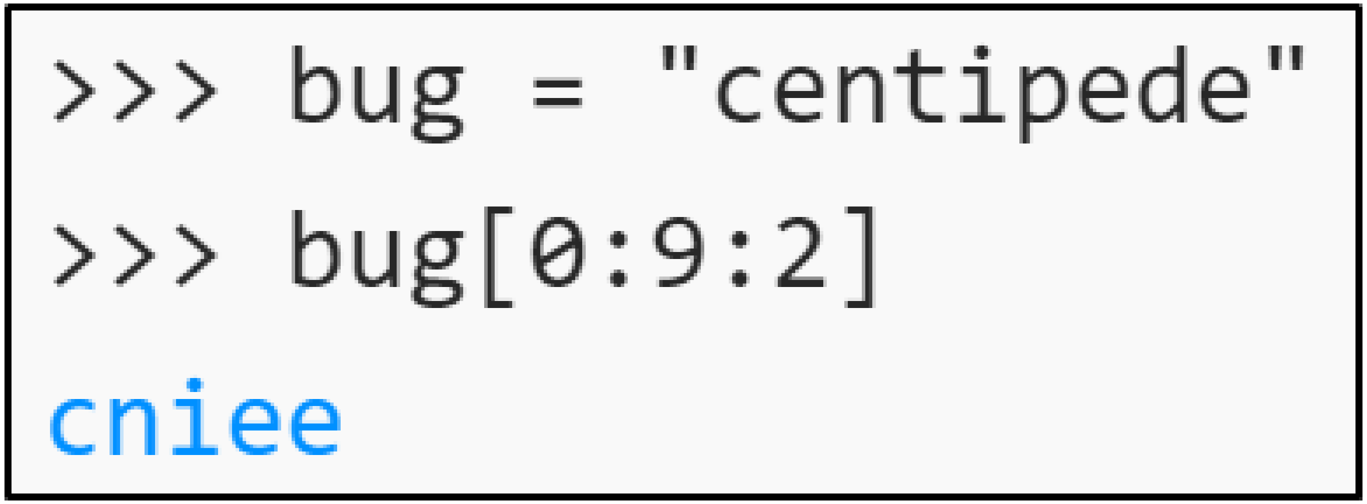
String Index Slicing



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

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

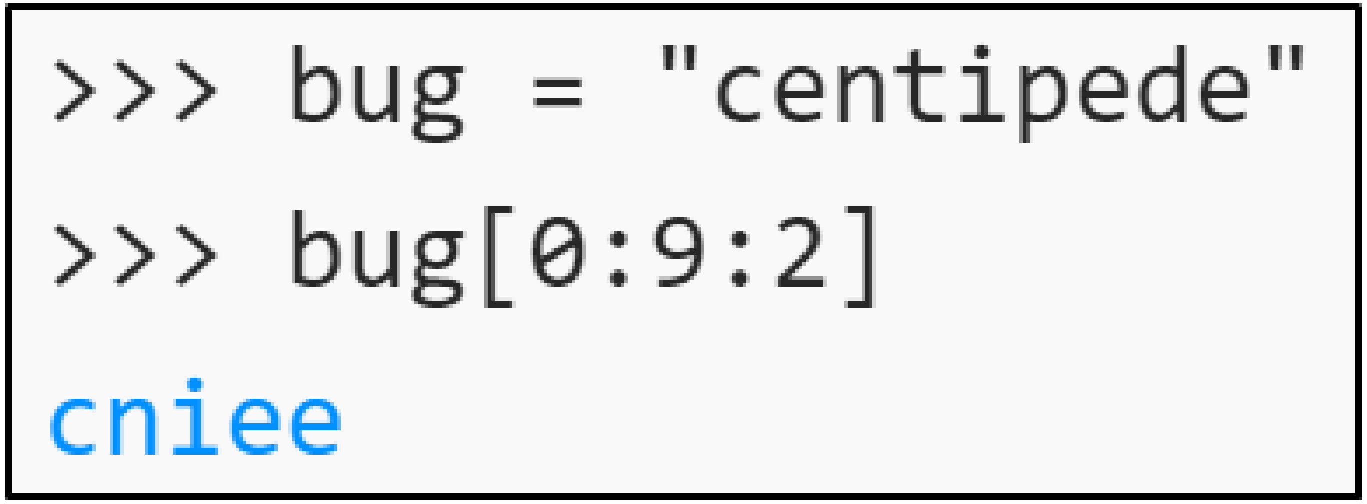
String Index Slicing



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

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.

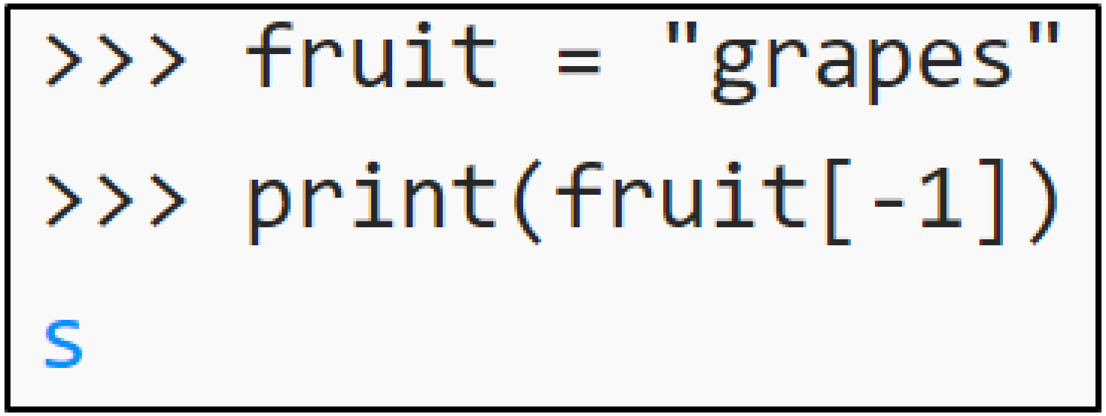
String Index Slicing



```
>>> 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.

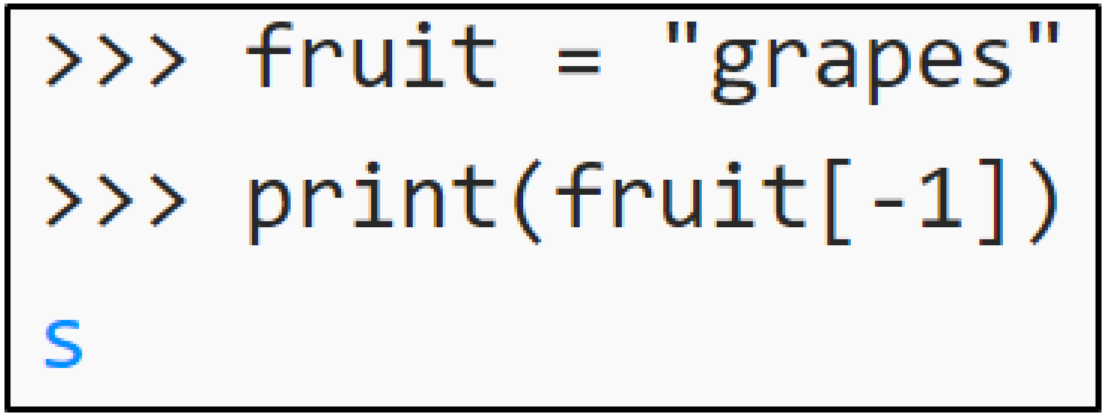
String Index Slicing



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

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.

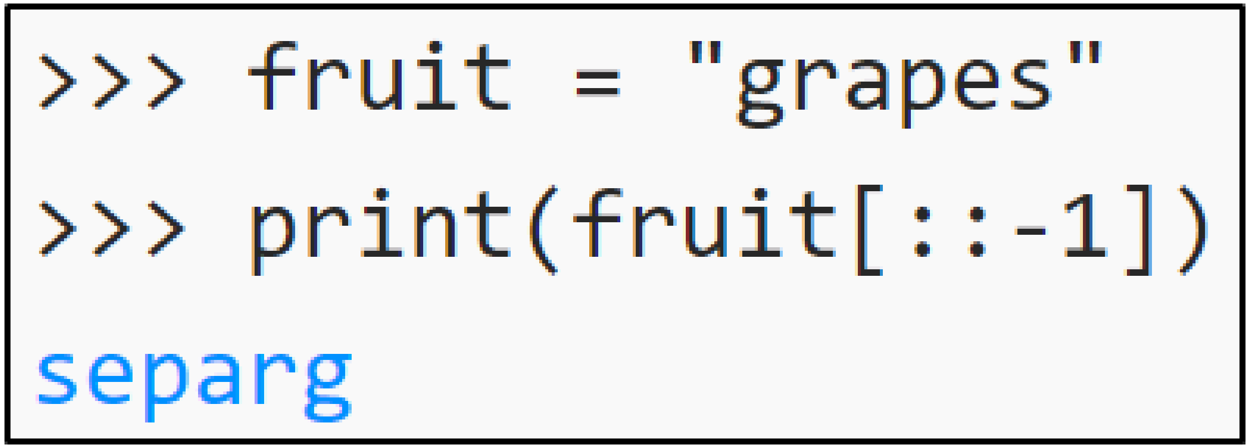
String Index Slicing



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

So here we're printing out the string index for “grapes” at the -1 position, which is the last letter.

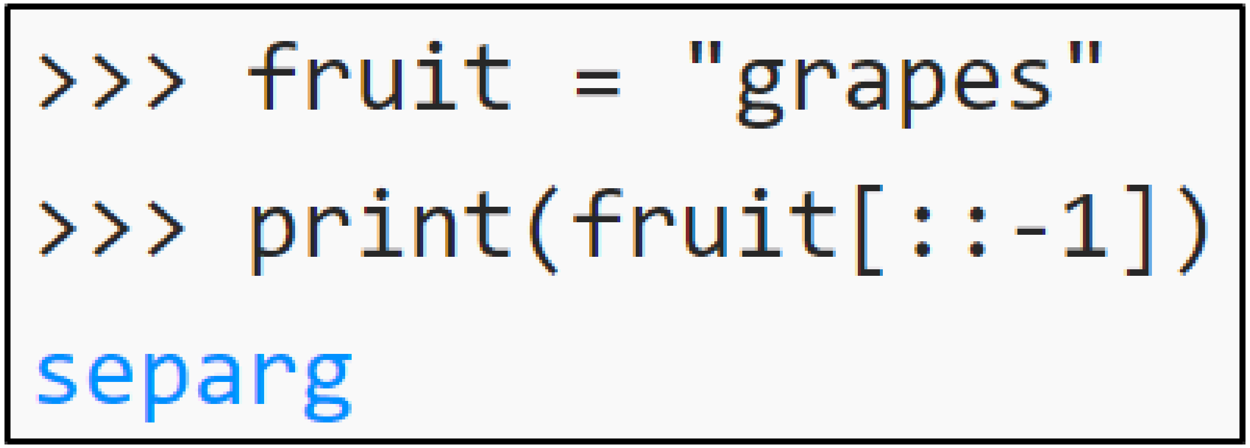
String Index Slicing



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

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

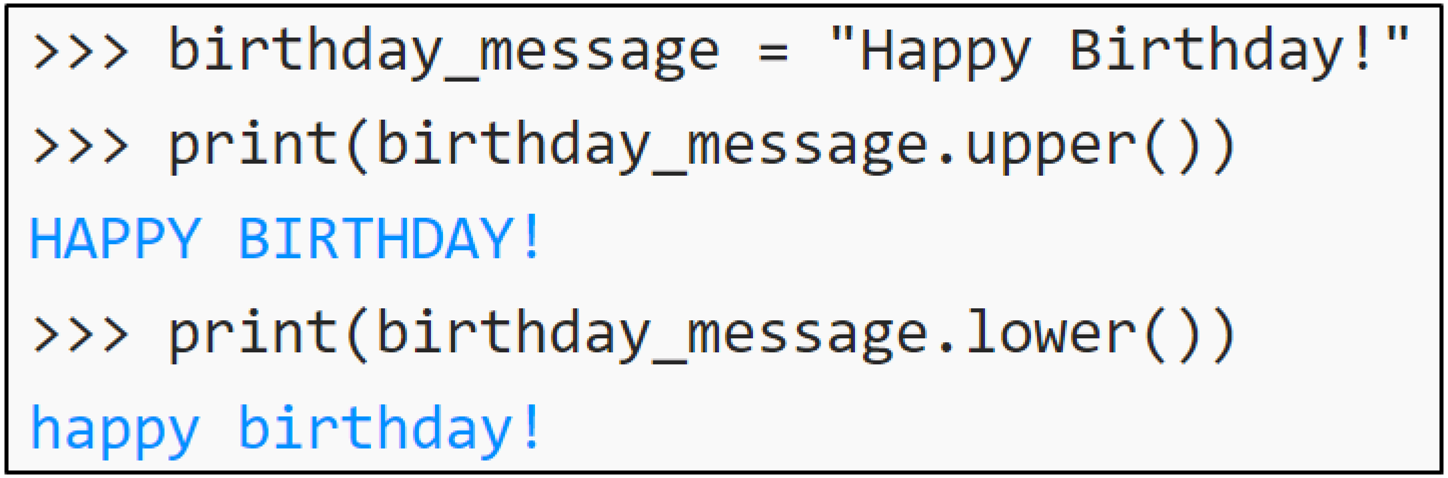
String Index Slicing



```
>>> 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).

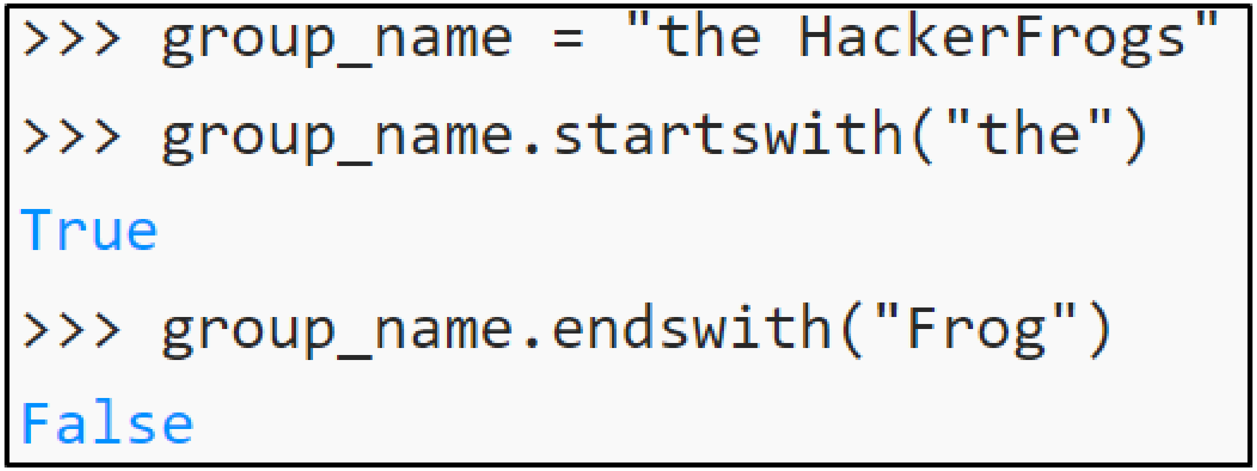
The Upper and Lower Methods



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

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

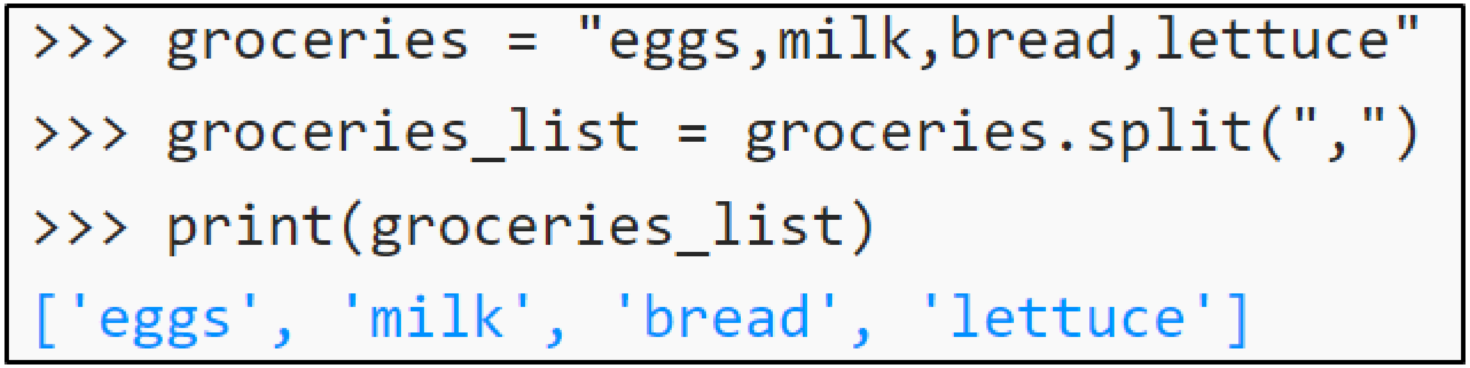
The Startswith and Endswith Methods



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

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

The Split Method



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

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).

Basic String Operations Exercise

Let's practice our basic string operations in Python at the following URL:

https://learnpython.org/en/Basic_String_Operations

Part 2 – String Operations Quiz

What is the output if we use the following code:

```
print('HackerFrogs'[6:-1])
```

A) Frogs

B) rFro

C) Frog

D) Nothing will print, the syntax is incorrect

Part 2 – String Operations Quiz

What is the output if we use the following code:

```
print('HackerFrogs'[6:-1])
```

A) Frogs

B) rFro

C) Frog

D) Nothing will print, the syntax is incorrect

Workshop Review Exercise

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

We're going to write a program which records your full name, then tells you what your last name is, and how many letters are contained in it.

Workshop Review Exercise

Here are some key lines of code for this program:

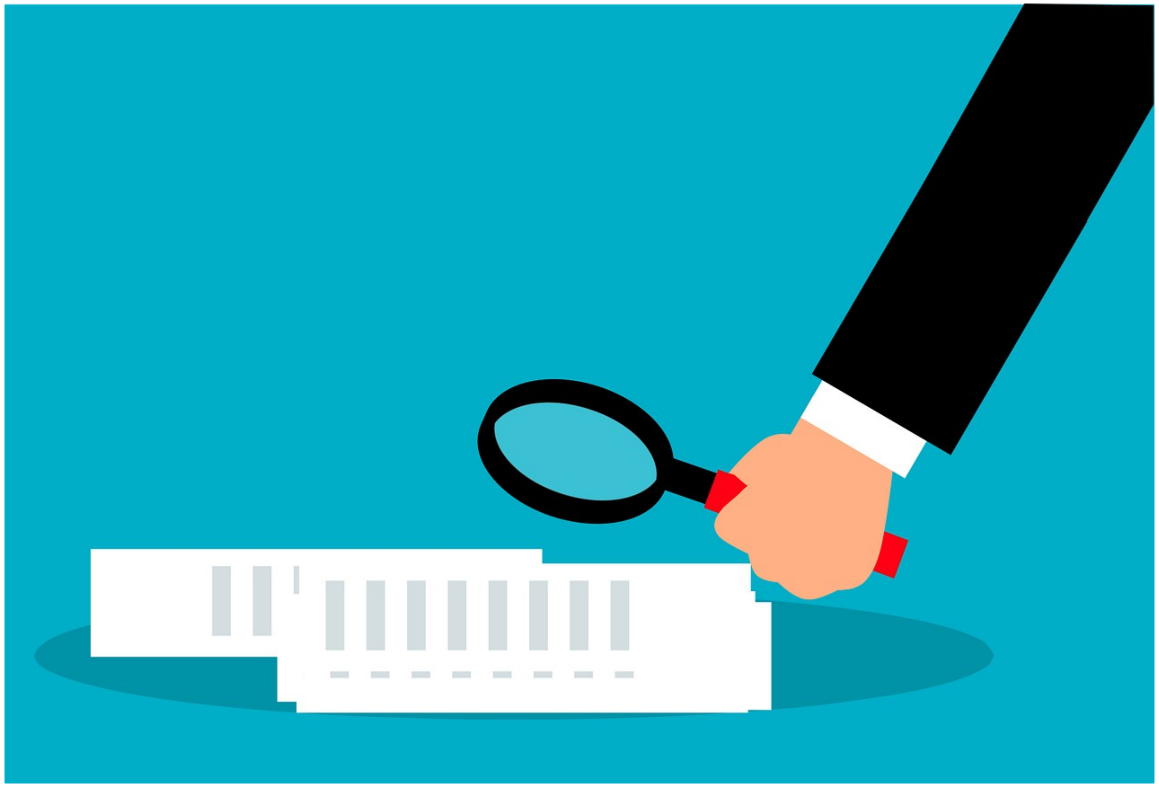
```
full_name = input('What...')
split_name = full_name.split()
last_name = split_name[1]
print(f'Your last name is {last_name}')
```

Workshop Review Exercise

Here are some key lines of code for this program:

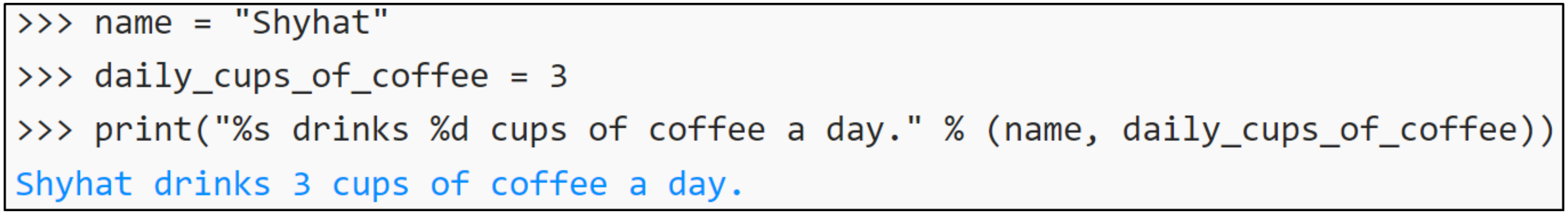
```
full_name = input('What...')
split_name = full_name.split()
last_name = split_name[1]
print(f'Your last name is {last_name}')
```

Summary



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

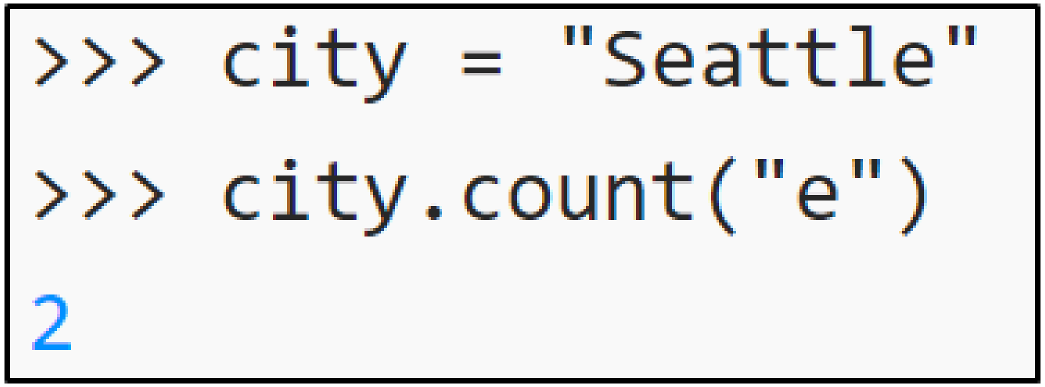
String Formatting



```
>>> 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.

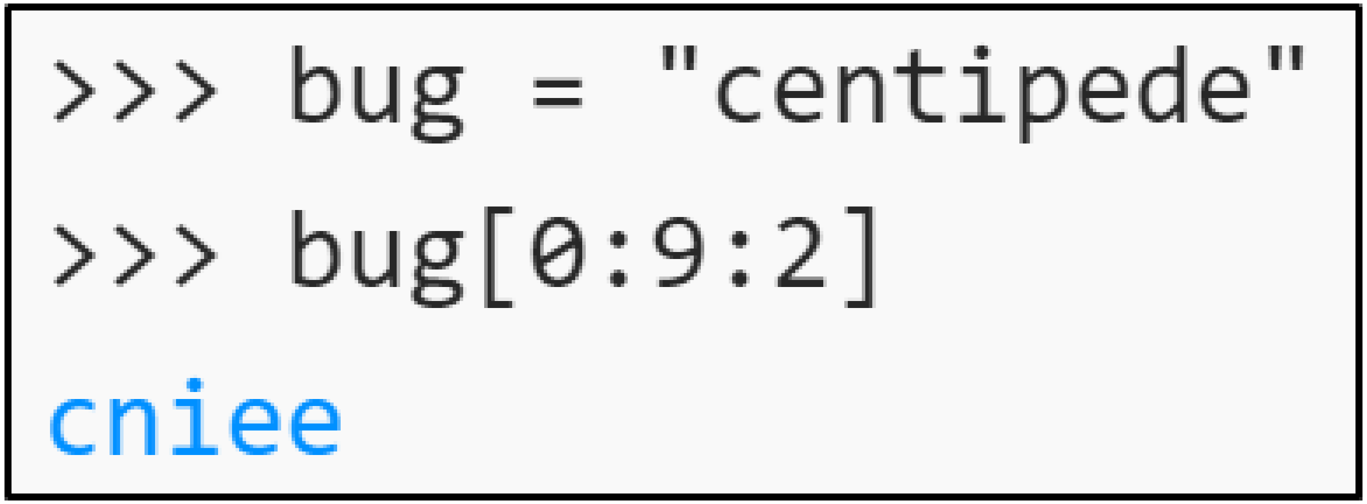
Basic String Operations



```
>>> 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.

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 topics:

- Loops

- 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!