Module 2 Part 3: Python Tuples, Dictionaries, Reading Data from a File, Formatting Print Output

This module is designed as an introduction to the Python programming language. It covers the basic syntax of Python, main data types and most used data collections with examples.

This module consists of 3 parts:

- **Part 1** - Introduction to Python.
- **Part 2** - Python Strings and Lists.
- **Part 3** - Python Tuples, Dictionaries, Reading data from a file, Formatting print output.

Each part is in a separate notebook. It is recommended to follow the order of the notebooks from Part 1 to Part 3.

Table of Contents

Tuples

Tuples are similar to lists, but are immutable. A tuple is declared as a comma-separated sequence of values or using the `tuple()` function. In the example below, two tuples are created:

- `new_tuple` is created from a comma-separated sequence of strings
- `vowels` is created by splitting a string

In [1]:
```
new_tuple = "apple", "banana", "orange"
```

In [2]:
```
new_tuple
```

Out[2]:
```
('apple', 'banana', 'orange')
```

In [3]:
```
type(new_tuple)
```

Out[3]:
```
tuple
```

In [4]:
```
'''The tuple of English vowels:'''

vowels = tuple('yeiouAEIOU')
vowels
```

Out[4]:
```
('y', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U')
```

In [5]:
```
'''Since a tuple is immutable, it cannot be changed.
Instead, a new tuple will be created.'''

vowels_corrected = ('a',) + vowels[1:]
t_single_elem = ("one",)
empty_tuple = ()

print(t_single_elem)
print(empty_tuple)
vowels_corrected
```

```
('one',)
()
```

Out[5]:
```
('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U')
```

Tuple assignments

Tuples can be used to easily swap the values of two variables. This can be achieved in Python without the need for a temporary variable, in one line of code:

In [6]:
```
a = 1
b = 5

a,b = b,a

print(a)
print(b)
```

```
5
1
```

The example below demonstrates how to extract values from or "unpack" a tuple.

NOTE: The number of variables on the left should be the same as the number of elements in the tuple.

```
In [7]: '''Validating the tuple `new_tuple`'''

        new_tuple
```

```
Out[7]: ('apple', 'banana', 'orange')
```

```
In [8]: '''Need to define three variables to unpack a tuple'''

        x, y, z = new_tuple
        print(x)
        print(y)
        print(z)
```

```
apple
banana
orange
```

```
In [9]: '''If the number of variables does not equal number of values in the tuple,
        we will get an error:'''

        d, e, f, g = new_tuple
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-003583785d27> in <module>()
      2 we will get an error:'''
      3
----> 4 d, e, f, g = new_tuple

ValueError: not enough values to unpack (expected 4, got 3)
```

Quite often, a function returns values as tuples. Technically, a Python function can return only one value, so if we need a function to return several values they can be returned as a tuple.

The return expression `return result_1, result_2, result_3` in a function will produce a tuple `(result_1, result_2, result_3)`.

Create tuples with the `zip()` function

The `zip()` function is an example of a function that returns tuples. This function pairs up the elements from multiple sequences, starting with the first values, then the second, etc.

In [6]:
```
list_num = [1,2,3,4,5]
list_alpha =['a','b','c','d','e']
zipped = zip(list_num, list_alpha)
```

Note that in Python 3 the `zip()` function returns a **zip** object which is an iterator:

In [7]:
```
type(zipped)
```

Out[7]:
```
zip
```

An **iterator** in Python is a special object type that works as a sequence, and can be looped over using (for example) a `for` statement. An iterator is created when we loop over Python lists, tuples, or dictionaries (will be reviewed in the next section).

Let's loop over `zipped`:

In [8]:
```
for i in zipped:
    print(i)
```

```
(1, 'a')
(2, 'b')
(3, 'c')
(4, 'd')
(5, 'e')
```

Let's try to convert the `zipped` object to a list:

In [9]:
```
zipped_list = list(zipped)
zipped_list
```

Out[9]:
```
[]
```

NOTE: Conversion to a list was unsuccessful in the cell above; `zipped_list` is empty. This is because an iterator can be traversed only once and for this reason it is convenient to convert it to a list first and then operate with the list.

In [10]:
```
zip_again = zip(list_num, list_alpha)
zipped_list_2 = list(zip_again)
zipped_list_2
```

Out[10]:
```
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
```

The function `zip()` used with the `*` operator can be used to unzip a list:

In [11]:
```
list(zip(*zipped_list_2))
```

Out[11]:
```
[(1, 2, 3, 4, 5), ('a', 'b', 'c', 'd', 'e')]
```

In [12]:
```
first_list = list(list(zip(*zipped_list_2))[0])
second_list = list(list(zip(*zipped_list_2))[1])
```

```
In [13]: first_list
```

```
Out[13]: [1, 2, 3, 4, 5]
```

```
In [14]: second_list
```

```
Out[14]: ['a', 'b', 'c', 'd', 'e']
```

Dictionaries

A dictionary is similar to a list except the indices are not limited to only integers. The dictionary is a set of key-value pairs where the key is the index to its associated value. The general form of a dictionary is:

```
{key_1: value_1, key_2: value_2, ...}
```

A dictionary can be created by enclosing a sequence of `key: value` pairs in curly brackets.

Dictionaries are **mutable**; a dictionary can be built by adding items (key-value pairs) to an empty list. The order of items in a dictionary does not matter, they are not indexed with integers. Instead, keys are used to look up values.

NOTE: The operator `in` also works on dictionaries, but it only scans keys, not values.

```
In [19]: my_dictionary = {}
         my_dictionary["one"] = 1
         my_dictionary['two'] = 2
         print(my_dictionary)
         list(my_dictionary.values())
```

```
{'one': 1, 'two': 2}
Out[19]: [1, 2]
```

```
In [20]: '''Traversing over dictionary is very similar to that of a list,
         just this time keys are used not indices.
         Compare these two loops: the first one prints keys, not values.'''

         for value in my_dictionary:
             print(value)

         for key in my_dictionary:
             print(my_dictionary[key])
```

```
one
two
1
2
```

Dictionary comprehension

In the previous section, we learned about list comprehensions. A dictionary comprehension works very similar to a list comprehension, but the end result is a dictionary. The structure of a dictionary comprehension can be described as follows:

```
{key: value for (key, value) in iterable}
```

Let's see how it works. One of the applications is to take two lists and create a dictionary using dictionary comprehension.

NOTE: You will find in the documentation that the term *dictionary comprehension* is often shortened to *dict comprehension*.

In [21]:
```
'''Create a dictionary where the key is an integer from 0 to 9
and the value is the same integer to the power of three.'''

{x: x**3 for x in range(10)}
```

Out[21]:
```
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343, 8: 512, 9: 729}
```

In [22]:
```
'''Creating a dictionary where the key is a letter of the alphabet:'''

import string
{x: y**3 for (x, y) in zip(string.ascii_lowercase, range(10))}
```

Out[22]:
```
{'a': 0,
 'b': 1,
 'c': 8,
 'd': 27,
 'e': 64,
 'f': 125,
 'g': 216,
 'h': 343,
 'i': 512,
 'j': 729}
```

The expression `zip(string.ascii_lowercase, range(10))` creates a list of tuples where the first element in each pair is a lowercase letter of an English alphabet, and the second element is an integer from 0 to 9:

In [23]:
```
list(zip(string.ascii_lowercase, range(10)))
```

```
Out[23]: [('a', 0),
          ('b', 1),
          ('c', 2),
          ('d', 3),
          ('e', 4),
          ('f', 5),
          ('g', 6),
          ('h', 7),
          ('i', 8),
          ('j', 9)]
```

NOTE: For details on the `string` package, please refer to the Python documentation: Common string operations (Python Software Foundation, 2018).

EXERCISE 4: Word count

Imagine that you are given a long sentence and need to count how many times each word appears in the sentence.

1). First, split the sentence into a list of words. We will use a sentence consisting of 251 words from "Barnaby Rudge", by Charles Dickens.

```
In [16]: long_sentence = """To none of these interrogatories, whereof every one was more pat
the last, did Mrs Varden answer one word: but Miggs,
not at all abashed by this circumstance,
turned to the small boy in attendance—her eldest nephew—son of her own married sist
number twenty-sivin,
and bred in the very shadow of the second bell-handle on the right- hand door-post—
of her pocket- handkerchief, addressed herself to him: requesting that on his retur
his parents for the loss of her, his aunt, by delivering to them a faithful stateme
her in the bosom of that family, with which, as his aforesaid parents well knew, he
incorporated; that he would remind them that nothing less than her imperious sense
and devoted attachment to her old master and missis, likewise Miss Dolly and young
should ever have induced her to decline that pressing invitation which they, his pa
as he could testify, given her, to lodge and board with them, free of all cost and
lastly, that he would help her with her box upstairs, and then repair straight home
bearing her blessing and her strong injunctions to mingle in his prayers a supplica
that he might in course of time grow up a locksmith, or a Mr Joe, and have Mrs Vard
and Miss Dollys for his relations and friends."""
```

```
In [26]: '''Type your code here'''
words  = long_sentence.split(sep = " ")
words
```

```
Out[26]: ['To',
          'none',
          'of',
          'these',
          'interrogatories,',
          'whereof',
          'every',
          'one',
          'was',
          'more',
          'pathetically',
          'delivered',
          'than\nthe',
          'last,',
          'did',
          'Mrs',
          'Varden',
          'answer',
          'one',
          'word:',
          'but',
          'Miggs,\nnot',
          'at',
          'all',
          'abashed',
          'by',
          'this',
          'circumstance,\nturned',
          'to',
          'the',
          'small',
          'boy',
          'in',
          'attendance—her',
          'eldest',
          'nephew—son',
          'of',
          'her',
          'own',
          'married',
          'sister—born',
          'in',
          'Golden',
          'Lion',
          'Court,\nnumber',
          'twenty-sivin,\nand',
          'bred',
          'in',
          'the',
          'very',
          'shadow',
          'of',
          'the',
          'second',
          'bell-handle',
          'on',
                    'the',
                    'right-',
                    'hand',
                    'door-post—and',
                    'with',
                    'a',
                    'plentiful',
                    'use\nof',
                    'her',
                    'pocket-',
                    'handkerchief,',
                    'addressed',
                    'herself',
                    'to',
                    'him:',
                    'requesting',
                    'that',
                    'on',
                    'his',
                    'return',
                    'home',
                    'he',
                    'would',
                    'console\nhis',
                    'parents',
                    'for',
                    'the',
                    'loss',
                    'of',
                    'her,',
                    'his',
                    'aunt,',
                    'by',
                    'delivering',
                    'to',
                    'them',
                    'a',
                    'faithful',
                    'statement',
                    'of',
                    'his',
                    'having',
                    'left\nher',
                    'in',
                    'the',
                    'bosom',
                    'of',
                    'that',
                    'family,',
                    'with',
                    'which,',
                    'as',
                    'his',
                    'aforesaid',
                    'parents',
                    'well',
                    'knew,',
                    'her',
                    'best',
                    'affections',
                    'were\nincorporated;',
                    'that',
                    'he',
                    'would',
                    'remind',
                    'them',
                    'that',
                    'nothing',
                    'less',
                    'than',
                    'her',
                    'imperious',
                    'sense',
                    'of',
                    'duty,\nand',
                    'devoted',
                    'attachment',
                    'to',
                    'her',
                    'old',
                    'master',
                    'and',
                    'missis,',
                    'likewise',
                    'Miss',
                    'Dolly',
                    'and',
                    'young',
                    'Mr',
                    'Joe,\nshould',
                    'ever',
                    'have',
                    'induced',
                    'her',
                    'to',
                    'decline',
                    'that',
                    'pressing',
                    'invitation',
                    'which',
                    'they,',
                    'his',
                    'parents,',
                    'had,\nas',
                    'he',
                    'could',
                    'testify,',
                    'given',
                    'her,',
                    'to',
                    'lodge',
                    'and',
    'board',
    'with',
    'them,',
    'free',
    'of',
    'all',
    'cost',
    'and',
    'charge,',
    'for',
    'evermore;\nlastly,',
    'that',
    'he',
    'would',
    'help',
    'her',
    'with',
    'her',
    'box',
    'upstairs,',
    'and',
    'then',
    'repair',
    'straight',
    'home,\nbearing',
    'her',
    'blessing',
    'and',
    'her',
    'strong',
    'injunctions',
    'to',
    'mingle',
    'in',
    'his',
    'prayers',
    'a',
    'supplication\nthat',
    'he',
    'might',
    'in',
    'course',
    'of',
    'time',
    'grow',
    'up',
    'a',
    'locksmith,',
    'or',
    'a',
    'Mr',
    'Joe,',
    'and',
    'have',
    'Mrs',
    'Vardens\nand',
    'Miss',
    'Dollys',
    'for',
    'his',
    'relations',
    'and',
    'friends.']
```

In [19]:
```
len(long_sentence)
```

Out[19]:
```
1432
```

2). Create a dictionary from that list where keys are unique words (e.g. if "the" appears in the sentence 3 times, there will only be one key for "the") from the list, and the values are its occurrence number.

In [28]:
```
'''Type your code here'''
my_dict = {}

for i in words:
    if i not in my_dict:
        my_dict[i] = 1
    else:
        my_dict[i] += 1

print(my_dict.values())
print(my_dict.keys())

print(my_dict)
```

```
dict_values([1, 1, 9, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1,
2, 1, 1, 7, 6, 1, 1, 6, 1, 1, 1, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
4, 5, 1, 1, 1, 1, 1, 1, 1, 1, 6, 7, 1, 1, 5, 3, 1, 2, 3, 1, 2, 1, 1, 2, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 8, 1, 1, 2, 1, 1, 2,
1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
dict_keys(['To', 'none', 'of', 'these', 'interrogatories,', 'whereof', 'every', 'on
e', 'was', 'more', 'pathetically', 'delivered', 'than\nthe', 'last,', 'did', 'Mrs',
'Varden', 'answer', 'word:', 'but', 'Miggs,\nnot', 'at', 'all', 'abashed', 'by', 'th
is', 'circumstance,\nturned', 'to', 'the', 'small', 'boy', 'in', 'attendance—her',
'eldest', 'nephew—son', 'her', 'own', 'married', 'sister—born', 'Golden', 'Lion', 'C
ourt,\nnumber', 'twenty-sivin,\nand', 'bred', 'very', 'shadow', 'second', 'bell-hand
le', 'on', 'right-', 'hand', 'door-post—and', 'with', 'a', 'plentiful', 'use\nof',
'pocket-', 'handkerchief,', 'addressed', 'herself', 'him:', 'requesting', 'that', 'h
is', 'return', 'home', 'he', 'would', 'console\nhis', 'parents', 'for', 'loss', 'he
r,', 'aunt,', 'delivering', 'them', 'faithful', 'statement', 'having', 'left\nher',
'bosom', 'family,', 'which,', 'as', 'aforesaid', 'well', 'knew,', 'best', 'affection
s', 'were\nincorporated;', 'remind', 'nothing', 'less', 'than', 'imperious', 'sens
e', 'duty,\nand', 'devoted', 'attachment', 'old', 'master', 'and', 'missis,', 'likew
ise', 'Miss', 'Dolly', 'young', 'Mr', 'Joe,\nshould', 'ever', 'have', 'induced', 'de
cline', 'pressing', 'invitation', 'which', 'they,', 'parents,', 'had,\nas', 'could',
'testify,', 'given', 'lodge', 'board', 'them,', 'free', 'cost', 'charge,', 'evermor
e;\nlastly,', 'help', 'box', 'upstairs,', 'then', 'repair', 'straight', 'home,\nbear
ing', 'blessing', 'strong', 'injunctions', 'mingle', 'prayers', 'supplication\ntha
t', 'might', 'course', 'time', 'grow', 'up', 'locksmith,', 'or', 'Joe,', 'Vardens\na
nd', 'Dollys', 'relations', 'friends.'])
{'To': 1, 'none': 1, 'of': 9, 'these': 1, 'interrogatories,': 1, 'whereof': 1, 'ever
y': 1, 'one': 2, 'was': 1, 'more': 1, 'pathetically': 1, 'delivered': 1, 'than\nth
e': 1, 'last,': 1, 'did': 1, 'Mrs': 2, 'Varden': 1, 'answer': 1, 'word:': 1, 'but':
1, 'Miggs,\nnot': 1, 'at': 1, 'all': 2, 'abashed': 1, 'by': 2, 'this': 1, 'circumsta
nce,\nturned': 1, 'to': 7, 'the': 6, 'small': 1, 'boy': 1, 'in': 6, 'attendance—he
r': 1, 'eldest': 1, 'nephew—son': 1, 'her': 10, 'own': 1, 'married': 1, 'sister—bor
n': 1, 'Golden': 1, 'Lion': 1, 'Court,\nnumber': 1, 'twenty-sivin,\nand': 1, 'bred':
1, 'very': 1, 'shadow': 1, 'second': 1, 'bell-handle': 1, 'on': 2, 'right-': 1, 'han
d': 1, 'door-post—and': 1, 'with': 4, 'a': 5, 'plentiful': 1, 'use\nof': 1, 'pocket-
': 1, 'handkerchief,': 1, 'addressed': 1, 'herself': 1, 'him:': 1, 'requesting': 1,
'that': 6, 'his': 7, 'return': 1, 'home': 1, 'he': 5, 'would': 3, 'console\nhis': 1,
'parents': 2, 'for': 3, 'loss': 1, 'her,': 2, 'aunt,': 1, 'delivering': 1, 'them':
2, 'faithful': 1, 'statement': 1, 'having': 1, 'left\nher': 1, 'bosom': 1, 'famil
y,': 1, 'which,': 1, 'as': 1, 'aforesaid': 1, 'well': 1, 'knew,': 1, 'best': 1, 'aff
ections': 1, 'were\nincorporated;': 1, 'remind': 1, 'nothing': 1, 'less': 1, 'than':
1, 'imperious': 1, 'sense': 1, 'duty,\nand': 1, 'devoted': 1, 'attachment': 1, 'ol
d': 1, 'master': 1, 'and': 8, 'missis,': 1, 'likewise': 1, 'Miss': 2, 'Dolly': 1, 'y
oung': 1, 'Mr': 2, 'Joe,\nshould': 1, 'ever': 1, 'have': 2, 'induced': 1, 'decline':
1, 'pressing': 1, 'invitation': 1, 'which': 1, 'they,': 1, 'parents,': 1, 'had,\na
s': 1, 'could': 1, 'testify,': 1, 'given': 1, 'lodge': 1, 'board': 1, 'them,': 1, 'f
ree': 1, 'cost': 1, 'charge,': 1, 'evermore;\nlastly,': 1, 'help': 1, 'box': 1, 'ups
tairs,': 1, 'then': 1, 'repair': 1, 'straight': 1, 'home,\nbearing': 1, 'blessing':
1, 'strong': 1, 'injunctions': 1, 'mingle': 1, 'prayers': 1, 'supplication\nthat':
1, 'might': 1, 'course': 1, 'time': 1, 'grow': 1, 'up': 1, 'locksmith,': 1, 'or': 1,
'Joe,': 1, 'Vardens\nand': 1, 'Dollys': 1, 'relations': 1, 'friends.': 1}
```

In [ ]:
```
'''Exercise 4 solution:'''

'''1). The method 'split()' can be used here.
```

```
Any delimiter can be passed as the first argument to split
the string 'long_sentence':'''

'''Note: if sep is not specified, any whitespace string is a separator by default.'

list_of_words = long_sentence.split(sep=' ')
list_of_words
```

In [ ]:
```
'''2). Create the empty dictionary:'''

word_count_dictionary = {}

'''This dictionary will have all unique words from the list as keys
and the word occurrence number as values.
If the word is not in the dictionary we add new key and value is 1,
for existing keys the value is incremented by 1.'''

for w in list_of_words:
    if w not in word_count_dictionary:
        word_count_dictionary[w] = 1
    else:
        word_count_dictionary[w] += 1
```

Reading Data from a File

Often, we need to read data from a file. The file may contain simple text, an Excel spreadsheet, an XML document, or be in any other format. Python offers multiple tools for file reading.

The simplest way is to use the built-in function **`open()`**. This function opens a file and returns a file object; it gives a *file handle*. The file handle is not the actual data contained in the file, but instead it can be used to read the data. Typing `?open` in a command line will return a docstring about this function.

Open and read a text file

The general syntax of the `open()` function is:

```
open(name[, mode[, buffering]]) -> file object
```

This shows that the first argument is the path to a file (if the file is in the same directory only the filename needs to be provided; if not, the path to the file must be provided). The second argument states the mode. Here are the most used modes:

- 'r' - to open for reading,
- 'w' - to open for writing (old data is erased),
- 'a' - to open for appending to what is already in the file,

- 'r+' - read and write mode

A text file is a sequence of lines. A special character called the **newline** character represents the end of each line. In Python, `\n` represents a new line. **NOTE:** `\n` is actually one character even though it looks like two.

There are four different methods to read from a file: `read(), readlines(), readline()` and `for` loop over the file_object:

- `read()` can be used to read the whole file at once and use it as a single string (not recommended for big files)
- `readlines()` returns the content of a file as a list of strings; each line can be accessed by index
- `readline()` can be used to read only a part of file; the first call returns the first line, the second call returns the second line and so on. This function can be used inside a `while` loop to read the file line-by-line until a certain place reached.
- `for line in file_object: <...>` allows for processing of every line in a file, one at a time:

```
file_object = open('example_file_for_reading.txt', 'r')
for line in file_object:
    print(line)
```

Also, the `file_object` can be converted to a list using the `list()` constructor, i.e. `list(file_object)`. This will return a result similar to the `readlines()` method.

NOTE: When you are done working with the file, it's important to close it. If you don't, the actual file may end up empty, incomplete or corrupted. Once the method `close()` is called, subsequent method calls on that file object will not work:

```
file_object.close()
file_obj.read()
% ValueError: I/O operation on closed file
```

Writing to a file

The `write()` method in Python works like the `print()` function but it does not add a newline character `'\n'`. Again, open the file first:

```
file_object = open('file_to_write_to.txt', 'w')
file_object.write(('text goes here\n')
file_object.close()
```

In [30]:
```
two_rows_string = "This is the first line,\nand this is the second line."
```

In [31]:
```
two_rows_string
```

Out[31]:
```
'This is the first line,\nand this is the second line.'
```

In [32]:
```
print(two_rows_string)
```

```
This is the first line,
and this is the second line.
```

In [33]:
```
file_object = open('file_to_write_to.txt', 'w')
file_object.write(two_rows_string + '\n')
file_object.close()
```

In [34]:
```
file_object = open('file_to_write_to.txt', 'a')
file_object.write('Oh, one more thing\n')
file_object.close()
```

Formatting Print Output

In this short section we will quickly review how to format the output of the `print()` function. Here are a few examples:

In [30]:
```
'''Formatting strings with format specifiers'''

'''Pad with spaces if < 10 chars long'''
print("{0:20} is the best of them all".format("Stella Artois"))

'''Take the width from the second parameter'''
print("{0:{1}} is the best of them all".format("Stella Artois", 10))

'''Take named arguments'''
print("{beer:{width}} is the best of them all".format(beer="Stella Artois", width=1

'''Force right justification'''
print("{0:>20} is the best of them all".format("Stella Artois"))

'''Force right justification and pad with &'''
print("{0:&>20} is the best of them all".format("Stella Artois"))
```

```
Stella Artois        is the best of them all
Stella Artois is the best of them all
Stella Artois is the best of them all
       Stella Artois is the best of them all
&&&&&&&Stella Artois is the best of them all
```

In [31]:
```
'''String interpolation'''
personA = "Mary"
personB = "Jane"
print("%s and %s went up the hill" % (personA, personB))
```

```
Mary and Jane went up the hill
```

End of Module

You have reached the end of this module.

If you have any questions, please reach out to your peers using the discussion boards. If you and your peers are unable to come to a suitable conclusion, do not hesitate to reach out to your instructor on the designated discussion board.

When you are comfortable with the content, and have practiced to your satisfaction, you may proceed to any related assignments, and to the next module.

References

Python Software Foundation (2018). https://docs.python.org/3/library/string.html