Study

Unit 2

String Manipulation and Data Structures in Python

String Manipulation and Data Structures in Python Outline

- Python Strings
- Basic string method
- List and its methods
- Tuple and its methods
- Set and its methods
- Dictionary and its methods

Study Unit Duration

This Study Session requires a minimum of 3 hours' formal study time.

You may spend an additional 2-3 hours on revision.

Preamble

Apart from different data type such as integer, float, string and Boolean, Python also has data structure which include list, tuple, set, and dictionary. Data structures are a collection of data elements that are structured in some manner and are the core of Python programming language.

In this study unit, you will learn string manipulation and data structure in Python.

Learning Outcomes of Study Unit 2

Upon completion of this study unit, you should be able to:

2.1 Manipulate string in Python

2.2 Create and use data structures (list and tuple) and their methods to perform various tasks

2.3 Create and use data structures (set, and dictionary) and their methods to perform various tasks

Terminologies, Acronyms and their Meaning

.ipynb	Jupyter notebook or Jupyter lab extension
.py	Python extension
IDE	Integrated Development environment (IDE)
R	R programming language
EDA	Exploratory Data Analysis
NaN	Not a Number

Np	Numpy
Pd	Pandas
Os	Operating system
AI	Artificial Intelligence
Int	Integer data type
Str	String data type
Bool	Boolean data type

2.1 String Manipulation in Python

2.1.1 Python Strings

A string is an ordered sequence of characters. Two key words here, **ordered** and **characters.** Ordered means that we will be able to use *indexing* and *slicing* to grab elements from the string.

Creating Strings

Single or double quotes are okay.

```
"Hi, welcome to string manipulation in Python"
```

‘Hi, welcome to string manipulation in Python’

We can use another set of quotes to capture that inside a single quote. For example:

```
"I'm a beginner in python programming!"
```

“I’m a beginner in python programming!”

```
"I'm feeling curious"
```

“I’m feeling curious”

The len() function

We can get the length of a string by using **len()** function. Every position is counted including spaces.

Examples

```
len("Python")
```

6

```
len("Python is simple")
```

16

```
len("I'm feeling curious")
```

19

2.1.2 String Operations

We can perform some operations such as string concatenation and replication in Python.

String Concatenation

Addition operator (+) enable us to join two strings together. This is known as string concatenation. For example, **“Python” + “string”** will become **Pythonstring**. Also, if **S1** and **S2** are strings, then **S1 + S2** is also a string (string concatenation).

We can use single quote with a space ‘ ’ or double quote with a space “ ” to create an empty string.

Example 1

```
a = "Python is"

b = "a programming language"

a + b
```

‘Python isa programming language’

Example 2

```
a + " " + b
```

‘Python is a programming language’

Example 3

```
"My name is Jamal" + " and I am from Somalia"
```

‘My name is Jamal and I am from Somalia’

Example 4

```
A = "Hello"

B = " "

C = "world"

print(A + B + C)
```

Hello world

Example 5

```
A = "Hello " # Note the extra space after Hello

B = "world"

print(A + B)
```

```
Hello world
```

String Replication

With multiplication operator (*), we can repeat the number of times a particular string should be repeated.

Example 1

Hello will be repeated three times

```
"Hello" * 3
```

```
‘HelloHelloHello’
```

Example 2

This will print Sudan in six times

```
country = "Sudan"


print(country*6)
```

```
SudanSudanSudanSudanSudanSudan
```

Example 3

```
"Python is simple " * 3
```

```
‘Python is simple Python is simple Python is simple’
```

Example 4

String replication and string concatenation

```
"Python is simple, " * 3 + "and Python is simple"
```

```
‘Python is simple, Python is simple, Python is simple, and Python is simple’
```

To concatenate two strings, we use the + operator

The * operator can be used to repeat the string for a given number of times.

2.1.3 Indexing and Slicing

Since strings are *ordered sequences* of characters, it means we can **select** single characters (indexing) or grab sub-sections of the string (slicing).

Indexing

Indexing starts at **0**, consider the string Somalia:

```
character:   S   o   m   a   l   i   a


index:       0   1   2   3   4   5   6
```

You can access a particular character by putting its position index in a square bracket.

```
country = "Somalia"


print(country)
```

Somalia

```
country[0]
```

‘S’

```
country[3]
```

‘a’

```
country[5]
```

‘i’

Python also supports reverse indexing. Consider the string

```
character:      S   o   m   a   l   i   a
index:          0   1   2   3   4   5   6
reverse index: -7  -6  -5  -4  -3  -2  -1
```

Reverse indexing is used commonly to grab the last “chunk” of a sequence.

```
country[-1]
```

‘a’

```
country[-2]
```

‘i’

```
country[-7]
```

‘S’

Slicing

We can grab entire subsections of a string with *slice* notation.

This is the notation:

```
[start : stop]
```

Key things to note:

- The starting index directly corresponds to where your slice will start
- The stop index corresponds to where your slice will go up to. **It does not include this index character!**
- The step size is how many characters you skip as you go grab the next one.

For example, the index [0: 4] will be 0, 1, 2, 3. The last index will not be included. The index [0 : 2] pulls the first two values out of the string. The index [1 : 3] pulls the second and third values out of the string.

Let’s see some examples

```
country = "Ethiopia"

country[0 : 3]
```

‘Eth’

```
country[0 : 4]
```

‘Ethi’

```
country[2 : 4]
```

‘hi’

On either side of the colon, a blank stand for default.

- **[: 2]** corresponds to **[start=default : stop = 2]**. Default value here is 0
- **[1:]** corresponds to **[start=1 : stop = default]**. Default value here is the last index of the string

Therefore, the slicing operation **[:2]** pulls out the first and second values in an array. The slicing operation **[1:]** pull out the second through the last values in an array. The examples below illustrate the default stop value is the last value in the array.

```
print(country)
```

Ethiopia

```
country[:2]
```

‘Et’

```
country[1:]
```

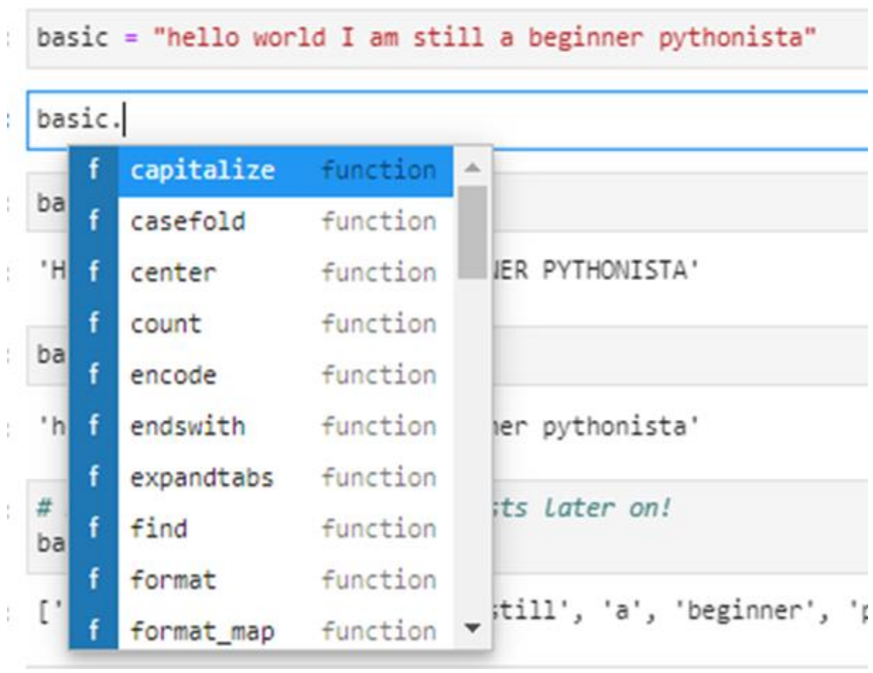
‘thiopia’

```
country[: 3]
```

‘Eth’

2.1.4 Basic String Methods

Methods are actions you can call off an object usually in the form **.method_name()** notice the closed parenthesis at the end. Strings have many methods which you can use. In fact, you can get a list of them by putting a dot(.) at the end of already assigned string and press a Tab key on your keyboard. Let's see some of them with examples.



```
basic = "hello world I am still a beginner pythonista"

basic.
```

f	capitalize	function
f	casefold	function
f	center	function
f	count	function
f	encode	function
f	endswith	function
f	expandtabs	function
f	find	function
f	format	function
f	format_map	function

```
ba
'H                              ER PYTHONISTA'
ba
'h                              er pythonista'
#                               ts later on!
ba
['                              till', 'a', 'beginner', '
```

```
basic = "hello world, I am still a beginner pythonista"
```

.upper()

.upper() will convert the string to upper case.

```
basic.upper()
```

'HELLO WORLD, I AM STILL A BEGINNER PYTHONISTA'

.lower()

.lower() will convert the string to lower case.

```
basic.lower()
```

'hello world, i am still a beginner pythonista'

.capitalize()

.capitalize() make the first character have upper case and the rest lower case.

```
basic.capitalize()
```

‘Hello world, i am still a beginner pythonista’

.title()

.title() will capitalize each word in a string.

```
basic.title()
```

‘Hello World, I Am Still A Beginner Pythonista’

.split()

.split() will split each character in the string.

```
basic.split()
```

[‘hello’, ‘world,’, ‘I’, ‘am’, ‘still’, ‘a’, ‘beginner’, ‘pythonista’]

2.1.5 String formatting

Sometimes, we will like to print our string with a specific format. We can use the function **print()** to force the string to **print** to a new line by using **\n**:

```
print('this is a new line \nnotice how this is on another new line')
```

this is a new line notice how this is on another new line

We can also use the function **print()** to force the string to have some extra spaces (something like when pressing **tab** on the keyboard) by using **\t**:

```
print('this is a tab\t notice how this prints with space between')
```

this is a tab notice how this prints with space between

2.1.6 String interpolation

String interpolation is the act of substituting values of variables into placeholders in a string. For example, as your teacher, I can greet every student taking this course like this "Hello **{Name of person}**, thank you for taking this course!". I would like to replace the placeholder **{Name of person}** with an actual name. This process is called **string interpolation**.

You can use the **.format()** method in a string, to perform string interpolation, this essentially insert **variables** when printing a string.

Example 1

```
Name_of_student = "Ruth"

print("Welcome {}, thank you for taking this course!".format(Name_of_student))
```

Welcome Ruth, thank you for taking this course!

With the introduction of **f**-strings, we can actually do it in this way:

```
print(f"Hello {Name_of_student}, thank you for taking this course!")
```

Hello Ruth, thank you for taking this course!

In above example, the prefix **f** tells Python to substitute the value of the variable **Name_of_student** inside curly brackets { }. So, that when we print, we get the above output. This new way of formatting strings is powerful, easy to use, and we shall be using **f** string henceforth.

Example 2

```
username = "Jamal_cs21"
password = 8022021

print(f"Welcome {username} and your password is {password}")
```

Welcome Jamal_cs21 and your password is 8022021

Example 3

```
name = "Jamal"
action = 'learn'
```

```
print(f"The {name} needs to {action} python")
```

The Jamal needs to learn python

If we have just one variable to do string extrapolation for, we can use print() function with the f-string.

Example 4

```
name = "Addisu"
print("My name is", name)
```

My name is Addisu

Example 5

```
name = "Addisu"
print(f"My name is {name}")
```

My name is Addisu

2.1.7 Formatting Numbers

You can also control the number of floating point or decimal places

```
num = 245.908343


print(f"The number is {num}")
```

The number is 245.908343

```
# For one decimal place


print(f"The code is {num:.1f}")
```

The code is 245.9

```
# For two decimal places

print(f"The code is {num:.2f}")
```

```
The code is 245.91
```

```
# For three decimal places

print(f"The code is {num:.3f}")
```

```
The code is 245.908
```

```
# For four decimal places

print(f"The code is {num:.4f}")
```

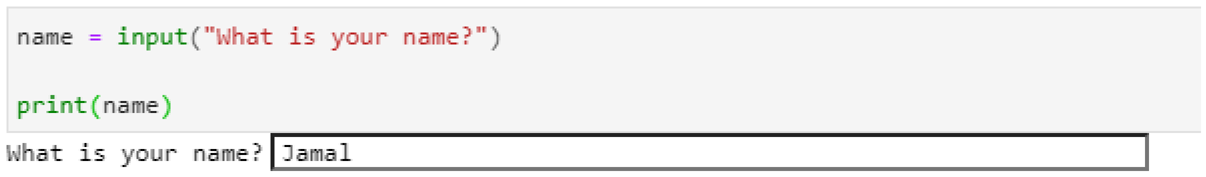
```
The code is 245.9083
```

2.1.8 Input and Output function

input() and **print()** functions are widely used for standard input and output operations respectively in Python. There are many cases where we might want to take the input from the user. In Python, we have the input() function that allows for this. You can get an input from a user and then save that as a variable that can be used later in your program with the help of **input()** function.

Example 1

This program will ask you of your name. Please respond by typing your name, then, press enter button on your keyboard.



```
name = input("What is your name?")

print(name)
```

```
What is your name? Jamal
```

Jamal

Any variable that comes as a result of **input()** will always be a string data type.

Example 2

```
name = input("What is your name?")

print(name)

type(name)
```

```
What is your name? Jamal
```

```
What is your name? Jamal
Jamal

str
```

Example 3

The code below when run will ask of your name, please supply your name to it:

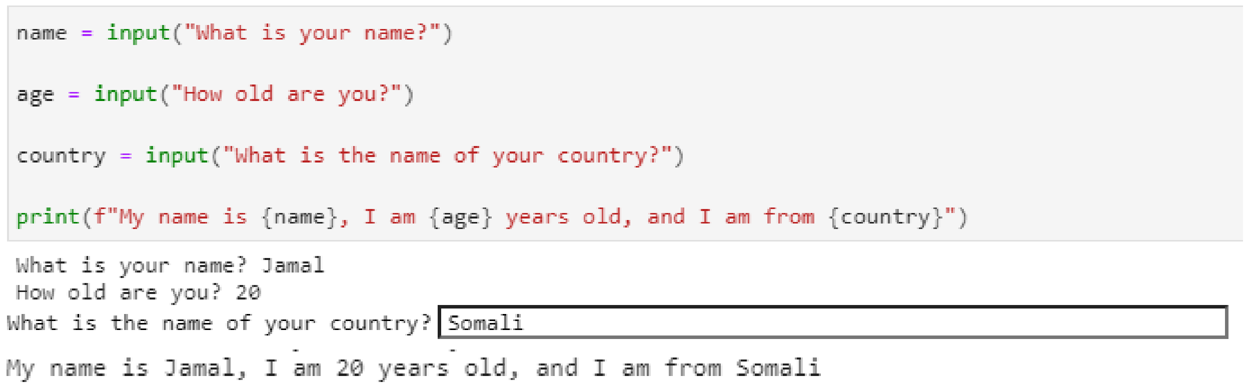
```
name = input("What is your name?")

print(f"My name is {name}")

print(f"{name} is of type {type(name)}")
```

```
What is your name? Jamal
```

```
What is your name? Jamal
My name is Jamal
Jamal is of type <class 'str'>
```

Example 4

This program will ask of your name, your age, and your country. It will then print for example, **My name is Jamal, I am 20 years old, and I am from Somali**.



```
name = input("What is your name?")

age = input("How old are you?")

country = input("What is the name of your country?")

print(f"My name is {name}, I am {age} years old, and I am from {country}")
```

```
What is your name? Jamal
How old are you? 20
What is the name of your country? Somali
My name is Jamal, I am 20 years old, and I am from Somali
```

2.2 Data Structures in Python – Lists and Tuples

2.2.1 Lists

We've learned that strings are sequences of characters. Similarly, lists are sequences of objects, they can hold a variety of data types in order, and they follow the same sequence and indexing bracket rules that strings do. We can create a list by putting all items or elements in a square bracket [ ] where each element is being separated by a comma. The items or elements in a list can be of any data types i.e. floats, integers, strings, boolean or their combination.

Let's explore some useful examples:

```
# An empty list

alist = []

type(alist)
```

```
<class 'list'>
```

```
my_list = [1, 2, 3]

my_list
```

```
[1, 2, 3]
```

```
a = 100

b = 200

c = 300

my_list3 = [a, b, c]
```

List of integers

```
ages = [21, 23, 16, 6, 76, 7]

ages
```

```
[21, 23, 16, 6, 76, 7]
```

```
type(ages)
```

```
<class 'list'>
```

List of float

```
time = [2.23, 1.59, 4.18, 3.51]

time
```

```
[2.23, 1.59, 4.18, 3.51]
```

```
type(time)
```

```
<class 'list'>
```

List of string

```
countries = ["Sudan", "South Sudan", "Kenya", "Ethiopia", "Somalia", "Uganda"]

countries
```

```
['Sudan', 'South Sudan', 'Kenya', 'Ethiopia', 'Somalia', 'Uganda']
```

```
type(countries)
```

```
<class 'list'>
```

Mixed lists

```
Mixed = [90, 2.5, "Jamal", 123, 0.75, True, False]

Mixed
```

```
[90, 2.5, ‘Jamal’, 123, 0.75, True, False]
```

```
type(Mixed)
```

```
<class ‘list’>
```

Nested Lists

Lists can hold other lists! This is called a nested list.

Examples

Let’s see some examples:

```
new_list = [1, 2, 3, "Ruth",  ["a", "b", "c"]]

new_list
```

```
[1, 2, 3, ‘Ruth’, [‘a’, ‘b’, ‘c’]]
```

```
type(new_list)
```

```
<class ‘list’>
```

```
eastafrica_northafrica = [["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda”, “Burundi"],
                          ["Algeria", "Egypt", "Libya", "Morocco", "Sudan", "Tunisia"]]
eastafrica_northafrica
```

```
[[‘Ethiopia’, ‘Kenya’, ‘Rwanda’, ‘Somalia’, ‘South Sudan’, ‘Uganda’, ‘Burundi’], [‘Algeria’,
‘Egypt’, ‘Libya’, ‘Morocco’, ‘Sudan’, ‘Tunisia’]]
```

```
type(eastafrica_northafrica)
```

```
<class ‘list’>
```

The range() function

We can also generate a list of sequence of numbers by using **range()** function. For example, **range(12)** will generate numbers from 0 to 11 (12 numbers).

The range function also has the **start**, **stop** and **step size** i.e. **range(start, stop,step_size)**. The **step_size** is 1 if not provided. The range object is “lazy” and does not store all the values in the memory. So, it remembers the **start**, **stop**, **step_size**.

Examples

```
range(10)
range(0, 10)
```

```
range(2, 8)
```

```
range(2, 8)
```

```
range(2, 20, 3)
```

```
range(2, 20, 3)
```

To force this function to output all the items, we can use the **list()** function.

Examples

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

As you know, the data type is a list

```
type(list(range(10)))
```

```
<class ‘list’>
```

```
list(range(2, 8))
```

```
[2, 3, 4, 5, 6, 7]
```

```
list(range(2, 20, 3))
```

```
[2, 5, 8, 11, 14, 17]
```

The len() function

We can get the number of elements in a list by using **len()** function.

Examples

```
Mixed = [90, 2.5, 'Ezekiel', 123, 0.75, True, False]

len(Mixed)
```

```
7
```

```
east_africa = ["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda", "Burundi"]

len(east_africa)
```

```
7
```

```
eastafrica_northafrica = [["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda", "Burundi"],
                          ["Algeria", "Egypt", "Libya", "Morocco", "Sudan", "Tunisia"]]


len(eastafrica_northafrica)
```

```
2
```

```
new_list = [1, 2, 3, ['a', 'b', 'c']]

len(new_list)
```

```
4
```

Indexing and Slicing

This works the same as the indexing and slicing of a string.

Example 1

```
mylist = [90, 2.5, "Joan", 123, 0.75, True, False, "Ruth", "Jamal"]

mylist[2]
```

```
‘Joan’
```

```
mylist[0:3]
```

```
[90, 2.5, ‘Joan’]
```

Example 2

```
new_list = [1, 2, 3, ['a', 'b', 'c']]

new_list[0]
```

```
1
```

```
new_list[3]
```

```
[‘a’, ‘b’, ‘c’]
```

```
new_list[3][0]
```

```
‘a’
```

Example 3

```
password_list = [2, 3, "four", [20, 30, 40, ["one", "two", "three"]]]

password_list[3]
```

[20, 30, 40, [‘one’, ‘two’, ‘three’]]

```
password_list[3][3]
```

[‘one’, ‘two’, ‘three’]

```
password_list[3][3][1:]
```

[‘two’, ‘three’]

Mutability of a list

List is mutable, that is you can change the element of a list to another.

Example 1

```
mylist = [1, 2, 3, 4, 5]

mylist[0]
```

1

```
mylist[0] = 9

mylist
```

[9, 2, 3, 4, 5]

As you can see, we have changed the first element in a list from 1 to 9.

Example 2

```
another_list = [90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True, "No"]

another_list[2]
```

‘Aaden’

```
another_list[2] = "Jamal"

another_list
```

[90, 2.5, ‘Jamal’, 123, 0.75, True, False, ‘Adhra’, True, ‘No’]

2.2.2 List Methods

Methods are actions you can call from an object. Their typical format is:

```
mylist = [elements in a list]

mylist.method()
```

You must call the parenthesis to execute the method! Let’s go through a few methods that pertain to lists.

.append( )

This appends or add an object to the end of the list. It can only add one element at a time

Examples

```
mylist = [90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True, "No"]

mylist
```

```
[90, 2.5, ‘Aaden’, 123, 0.75, True, False, ‘Adhra’, True, ‘No’]
```

```
mylist.append(6)

mylist
```

```
[90, 2.5, ‘Aaden’, 123, 0.75, True, False, ‘Adhra’, True, ‘No’, 6]
```

```
mylist.append("Jamal")

mylist
```

```
[90, 2.5, ‘Aaden’, 123, 0.75, True, False, ‘Adhra’, True, ‘No’, 6, ‘Jamal’]
```

.extend()

Alternative to **.append()** is **.extend()** which extends list by appending more than one element.

```
mylist.extend(["Addisu", "Aklilu", "Amondi", "Bakari"])

mylist
```

```
[90, 2.5, ‘Aaden’, 123, 0.75, True, False, ‘Adhra’, True, ‘No’, 6, ‘Jamal’, ‘Addisu’, ‘Aklilu’,
‘Amondi’, ‘Bakari’]
```

.insert( )

.insert() method insert object before a given index position.

Examples

```
mylist = [90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True, "No"]

mylist.insert(3, "Bakaffa")

mylist
```

```
[90, 2.5, 'Aaden', 'Bakaffa', 123, 0.75, True, False, 'Adhra', True, 'No']
```

```
visited_countries = ["America", "Rwanda", "Singapore", "Italy", "Canada", "Mauritius"]

visited_countries.insert(0, "South Africa")

visited_countries
```

```
['South Africa', 'America', 'Rwanda', 'Singapore', 'Italy', 'Canada', 'Mauritius']
```

.pop()

.pop() method remove and return item at index (default last).

```
mylist = [90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True, "No"]

mylist.pop()
```

```
'No'
```

```
mylist
```

```
[90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True]
```

```
mylist.pop(0)
```

```
90
```

```
mylist
```

```
[2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True]
```

.reverse()

.reverse() method reverse the order of the list

```
your_list = ["Sylvera", 90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True, "No"]

your_list.reverse()

your_list
```

```
['No', True, 'Adhra', False, True, 0.75, 123, 'Aaden', 2.5, 90, 'Sylvera']
```

```
visited_countries = ["America", "Rwanda", "Singapore", "Italy", "Canada", "Mauritius"]

visited_countries.reverse()
```

```
visited_countries
```

[‘Mauritius’, ‘Canada’, ‘Italy’, ‘Singapore’, ‘Rwanda’, ‘America’]

.sort()

.sort() method sort the list in ascending order and return None

Example 1

```
# Example 1

egg_weight = [59, 56, 61, 68, 52, 53, 69, 54, 57, 51]


egg_weight.sort()

egg_weight
```

[51, 52, 53, 54, 56, 57, 59, 61, 68, 69]

Example 2

Data relating to the marks of 13 students in the Introduction to Python quiz are given below:

10, 15, 10, 9, 18, 16, 14, 12, 16, 13, 15, 20, 17.

Sort the marks in descending order.

```
marks = [10, 15, 10, 9, 18, 16, 14, 12, 16, 13, 15, 20, 17]

marks.sort(reverse = True)

marks
```

[20, 18, 17, 16, 16, 15, 15, 14, 13, 12, 10, 10, 9]

Class activity 6 (Peer to peer review activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

- Create a list that contains the names of your best friends.
- Use Python to access the third element in the name list.
- How many friends are in your list?

2..2.3 Tuples

Tuples are ordered sequences just like a list, but have one major difference, they are **immutable**. That is, you cannot *change* them. So, in practice what does this actually mean? It means that you cannot reassign an item once it's in the tuple, unlike a list, where you can do a reassignment.

Just like the elements in a list are put in a square bracket **[ ]** separated by a comma, elements in a tuple are enclosed in a parentheses or brackets **( )** separated by comma **,**.

You use parenthesis and commas (,) for a tuple

List is mutable while tuple is immutable

Examples

```
# An empty tuple

atuple = ()

type(atuple)
```

```
<class 'tuple'>
```

```
t = (3, 2)

t
```

```
(3, 2)
```

Tuple of integers

```
ages = (23, 2, 45, 6, 76, 7)

type(ages)
```

```
<class 'tuple'>
```

Tuple of float

```
marks = (23.1, 20.8, 25.1, 17.9)

type(marks)
```

```
<class 'tuple'>
```

Tuple of string

```
teachers_name = ("Diric", "Bilen", "Baruk", "Jamal", "Gelila")
```

```
type(teachers_name)
```

<class ‘tuple’>

Mixed Tuple

```
mixed_tuple = (21, 12.3, 33.6, 9, "Gelila", True, False)

type(marks)
```

<class ‘tuple’>

Nexted tuple

This is also known as a tuple of tuple

```
nexted = ("Ethiopia", (1, 2.5, 6), ('Kenya'), (1, 22, 14, 15))

type(nexted)
```

<class ‘tuple’>

The len() function

We can get the number of elements in a tuple by using **len()** function.

Example 1

```
mixed_tuple = (21, 12.3, 33.6, 23.8, 9, "Gelila", True, False)

len(mixed_tuple)
```

8

Example 2

```
nexted = ("Ethopia", (1, 2.5, 6), ('Kenya'), (1, 22, 14, 15))

len(nexted)
```

4

Indexing and Slicing

This works the same as the indexing and slicing of a list.

Example 1

```
mytuple = (90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True, "No")

mytuple[3]
```

123

```
mytuple[0:3]
```

(90, 2.5, ‘Aaden’)

Example 2

```
new_tuple = (1, 2, 3, ('a', 'b', 'c'))

new_tuple[0]
```

1

```
new_tuple[3]
```

(‘a’, ‘b’, ‘c’)

```
new_tuple[3][0]
```

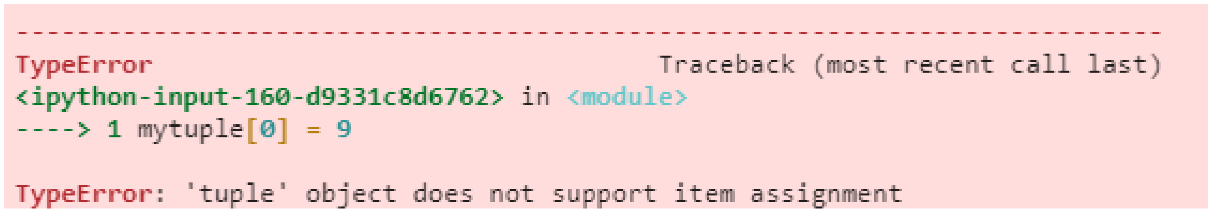
‘a’

Immutability of tuple

While list is mutable, tuple is not. That is, you can’t replace element of a tuple.

Examples

```
mytuple = (1, 2, 3)

mytuple[0] = 9
```

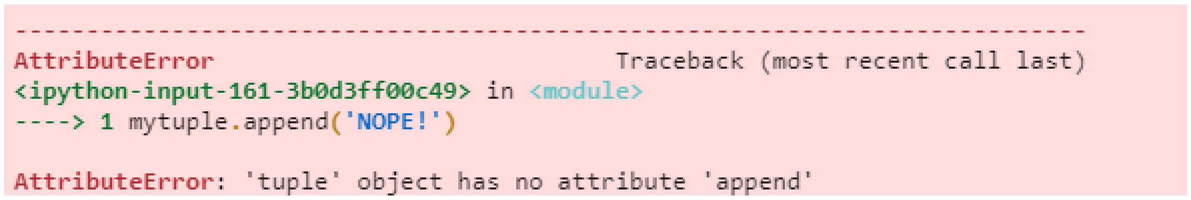


```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-160-d9331c8d6762> in <module>
----> 1 mytuple[0] = 9

TypeError: 'tuple' object does not support item assignment
```

As you can see, you can’t add or change element of a tuple.

List’s methods are different from tuple’s method. Therefore, none of the methods in a list can work in tuple. For example:

```
mytuple.append('NOPE!')
```



```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-161-3b0d3ff00c49> in <module>
----> 1 mytuple.append('NOPE!')

AttributeError: 'tuple' object has no attribute 'append'
```

Tuple methods

Tuples only have two methods available .index() and .count()

- .index() returns the index of value.
- .count() returns number of occurrences of value

```
t = ("a", "b", "a",  "c", "a")


t.index("b")
```

```
1
```

```
t.count("a")
```

```
3
```

Why use tuples?

Lists and tuples are very similar, so you may find yourself exchanging use cases for either one. However, you should use a tuple for collections or sequences that shouldn't be changed, such as the dates of the year, or user information such as an address, street, city, etc.

Class Activity 7 (Peer to peer review activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

- Create a tuple that includes the names of your best friends
- Use Python to access the third element in the name list.
- What is the length of the tuple?

2.3 Data Structures in Python – Sets and Dictionary

2.3.1 Sets

Another fundamental data structure is Set! Set is an unordered and unindexed collection of unique elements. We can construct them by using a curly bracket **{}** while elements in a set is being separated by a comma (,). Let’s go ahead and make a set to see how it works:

Example 1

```
call_received = {0, 1, 4, 2, 3, 5}


call_received
```

```
{0, 1, 2, 3, 4, 5}
```

```
type(call_received)
```

```
<class ‘set’>
```

You will notice that elements in a set has been arranged in an organised ascending order

Example 2

```
# Here are the set of fruits in my Fridge

myfruit = {"Apple", "Banana", "Cherry", "Orange", "Pineapples", "Grape", "Pawpaw"}

myfruit
```

```
{‘Cherry’, ‘Banana’, ‘Pineapples’, ‘Grape’, ‘Pawpaw’, ‘Orange’, ‘Apple’}
```

```
type(myfruit)
```

```
<class ‘set’>
```

One unique feature about set is that, it doesn’t support duplicate of an element.

Example 1

```
student_age = {19, 20, 15, 19, 16, 21, 17}

student_age
```

{15, 16, 17, 19, 20, 21}

```
type(student_age)
```

<class 'set'>

You will see that our initial elements in student age are 19, 20, 15, 19, 16, 21, and 17 but set has removed all the duplicates and we now left with the elements 15, 16, 17, 19, 20, and 21

The len() function

We can get the number of elements in a set by using **len()** function.

Examples

```
len(student_age)
```

6

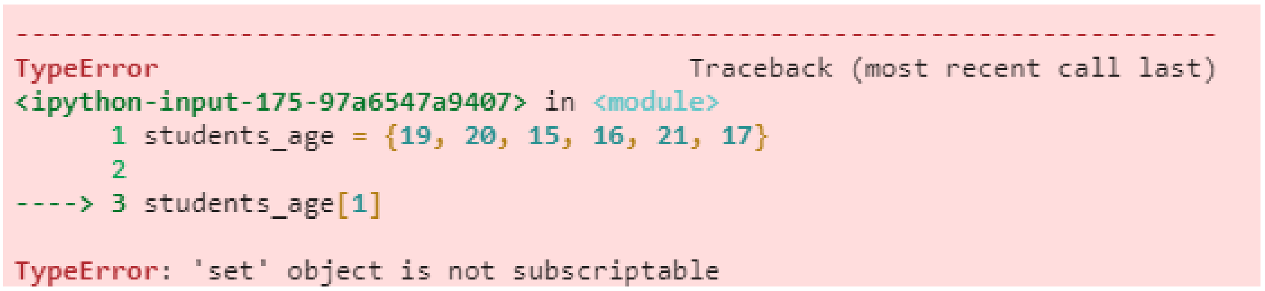
```
myset= {90, 2.5, 'Aaden', 123, 0.75, True, False, 'Adhra', True, "No"}

len(myset)
```

9

Indexing and Slicing

Since, set are unordered collection of unique elements, indexing has no meaning. Hence, the slicing operator [ ] will not work.

```
students_age = {19, 20, 15, 16, 21, 17}

students_age[1]
```



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-175-97a6547a9407> in <module>
      1 students_age = {19, 20, 15, 16, 21, 17}
      2 
----> 3 students_age[1]

TypeError: 'set' object is not subscriptable
```

As you can see, that throws an error. Since indexing is not working, then element in a set is not replaceable.

Adding element to a set

Since set is mutable, hence it is possible to add element to an existing set by using **.add()** attribute. If the element is already present in the set, then the function will ignore adding that element.

Example 1

```
student_age = {19, 20, 15, 16, 21, 17}
student_age.add(18)
student_age
```

```
{15, 16, 17, 18, 19, 20, 21}
```

Example 2

```
myset= {"Gurey", "Guuleed", "Jamal", "Ruth", "Habsade", "Habtom"}

myset
```

```
{‘Jamal’, ‘Habtom’, ‘Habsade’, ‘Gurey’, ‘Guuleed’, ‘Ruth’}
```

```
len(myset)
```

```
6
```

```
myset.add("Juma")
myset
```

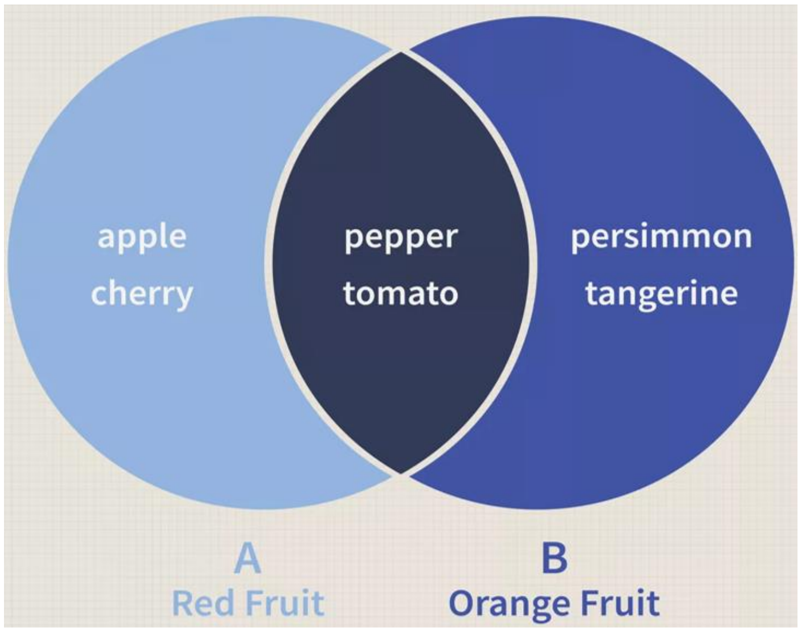
```
{‘Jamal’, ‘Habtom’, ‘Habsade’, ‘Gurey’, ‘Juma’, ‘Guuleed’, ‘Ruth’}
```

```
len(myset)
```

```
7
```

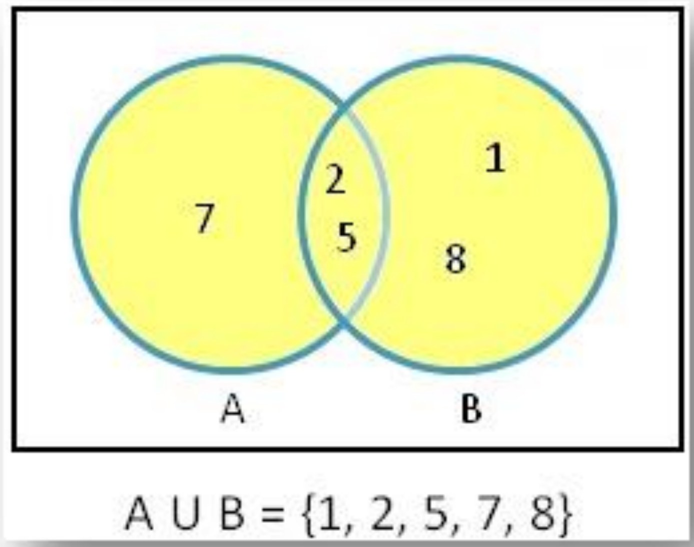
Basic Set Operations

We can perform set operations like union, intersection, compliment of two sets. As you know, sets have unique values. They eliminate duplicates. We can represent the relationship between sets in a diagram known as a Venn diagram.



Union of sets

The union of set **A** and **B**, denoted by $A \cup B$, is the collection of all elements in both sets without any duplication of elements.



To get the union of two sets, we put | at the middle of the two sets.

Example 1

```
A = {7, 2, 5}

B = {2, 5, 1, 8}
```

What is the union of **A** and **B**?

```
AunionB = A | B

print(AunionB)
```

```
{1, 2, 5, 7, 8}
```

We can also use **.union()** attribute to get the union of a set

```
AunionB = A.union(B)

print(AunionB)
```

```
{1, 2, 5, 7, 8}
```

Example 2

```
odd_number = {1, 3, 5, 7, 9}

even_number = {2, 4, 6, 8, 10}
```

What is the union of set **odd_number** and **even_number**?

```
all_numbers = odd_number | even_number

print(all_numbers)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Example 3

```
myfruits = {"Apple", "Banana", "Cherry", "Orange", "Water melon"}

friend_fruit = {"Pineapples", "Grape", "Pawpaw", "Banana", "Mango"}

all_fruits = myfruits | friend_fruit

all_fruits
```

```
{‘Mango’, ‘Water melon’, ‘Cherry’, ‘Banana’, ‘Pineapples’, ‘Grape’, ‘Pawpaw’, ‘Orange’,
‘Apple’}
```

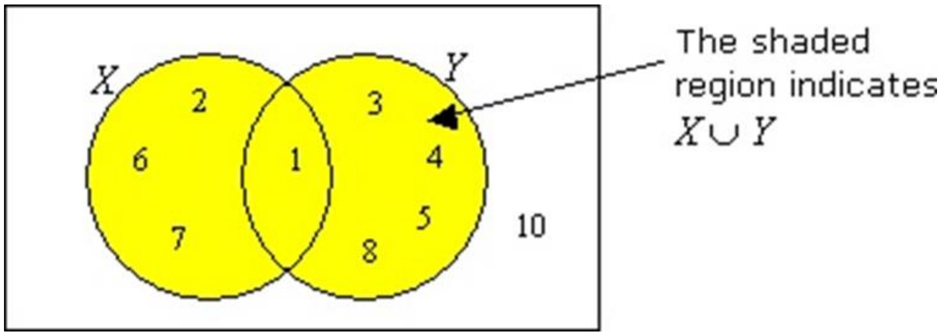
```
len(all_fruits)
```

```
9
```

My friend and I have 9 fruits altogether.

Class Activity 8 (Peer to peer review activity)

Peer to Peer Interaction



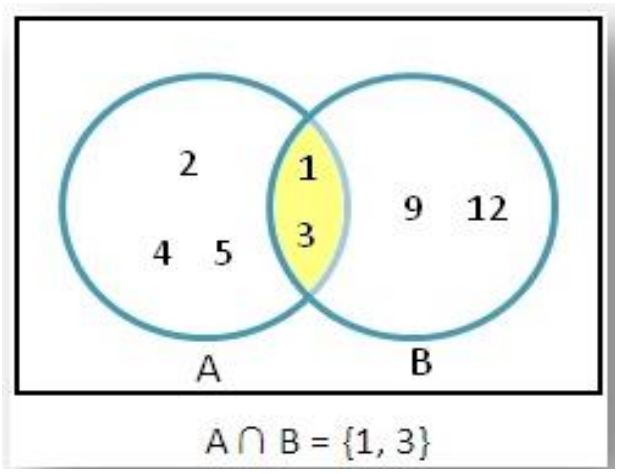
Visit the LMS, locate forum activity and participate in the discussion

Consider the Venn diagram above:

- represent both sets X and Y in Python
- what is the union of set X and Y?

Intersection

Intersection of two sets **A** and **B**, denoted by $A \cap B$, is the set containing all elements of A that also belong to B (or equivalently, all elements of B that also belong to A).



To compute intersection of two in Python, we put **&** at the middle of the two sets.

Example 1

```
A = {2, 4, 5, 1, 3}

B = {1, 3, 9, 12}

intersection = A & B

print(intersection)
```

```
{1, 3}
```

We can also use the function **.intersection()**.

```
intersection = A.intersection(B)

print(intersection)
```

```
{1, 3}
```

Example 2

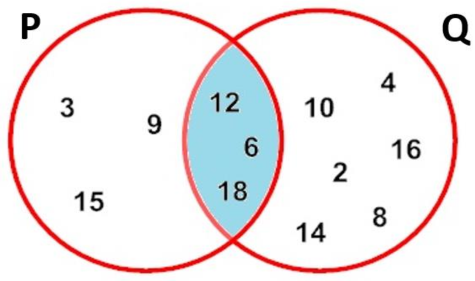
```
myfruits = {"Apple", "Banana", "Cherry", "Orange", "Water melon"}

friend_fruits = {"Pineapples", "Grape", "Pawpaw", "Banana", "Mango"}

common_fruit = myfruits & friend_fruit

common_fruit
```

```
{'Banana'}
```

As you can see, we have only Banana in common.

Class Activity 9 (Peer to peer review activity)

Peer to Peer Interaction



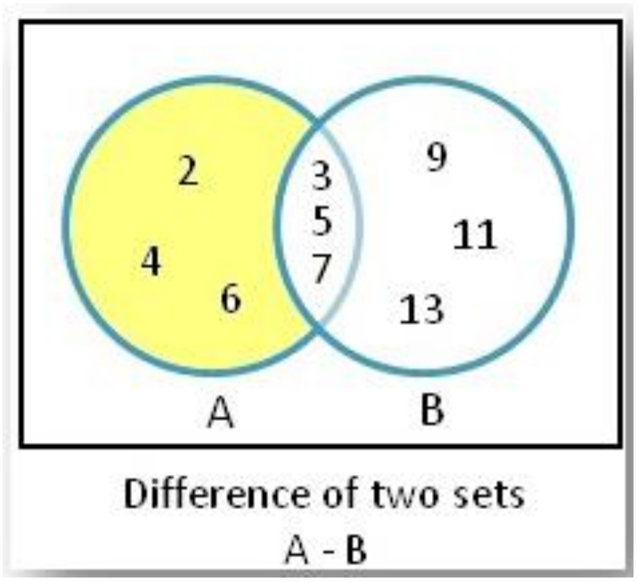
Visit the LMS, locate forum activity and participate in the discussion

Consider the Venn diagram above:

- represent both sets P and Q in Python
- what is the intersection of set X and Y?

Complement of a set (or set difference)

The complement or set difference of sets A and B, denoted by **A – B**, is the set of all elements in A that are not in B.



Difference of two sets
A - B

To compute complement of two sets in Python, we put **-** at the middle of the two sets.

Example 1

```
A = {2, 4, 6, 3, 5, 7}

B = {3, 5, 7, 9, 11, 13}

difference = A-B

print(difference)
```

```
{2, 4, 6}
```

We can also use the function **.difference()**.

```
difference = A.difference(B)

print (difference)
```

```
{2, 4, 6}
```

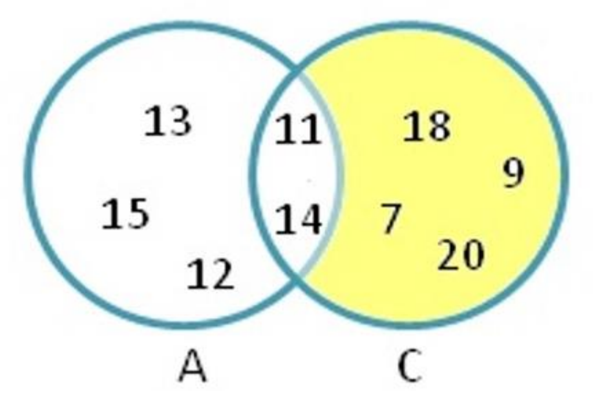
Example 2

```
myfruits = {"Apple", "Banana", "Cherry", "Orange", "Water melon"}

friend_fruits = {"Pineapples", "Grape", "Pawpaw", "Banana", "Mango"}

# Fruits that I have that my friend did not have

difference_fruit = myfruits - friend_fruit

difference_fruit
```

```
{‘Apple’, ‘Water melon’, ‘Orange’, ‘Cherry’}
```

The fruits that I have that my friend did not have are Apple, Cherry, Orange, and Water melon.

Class Activity 10 (Peer to Peer Review Activity)



Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Consider the Venn diagram above:

- represent both sets A and C in Python
- what is the difference of set C and A?

2.3.2 Dictionary

Dictionary is an ordered collection of **key-value** pairs. That is, it makes use of two elements, namely, a key and a value. Dictionary is usually used when we have a huge amount of data. We must know the **key** before we can retrieve the **value**.

We can create a dictionary by defining keys and value elements inside a curly bracket {}.

Example 1

```
my_dictionary = {"Key 1": "Value 1", "Key 2": "Value 2", "Key 3": "Value 3"}

my_dictionary
```

```
{‘Key 1’: ‘Value 1’, ‘Key 2’: ‘Value 2’, ‘Key 3’: ‘Value 3’}
```

```
type(country)
```

```
<class ‘str’>
```

Example 2

Life expectancy is the average number of years a person is expected to live based on the year of its birth.

```
# life expectancy in the year 2020

life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67, "Rwanda": 65, "Tanzania": 64, "Somalia": 54}

life_expectancy
```

```
{‘Nigeria’: 60, ‘Kenya’: 69, ‘Uganda’: 68, ‘Ethiopia’: 68, ‘Sudan’: 67, ‘Rwanda’: 65,
‘Tanzania’: 64, ‘Somalia’: 54}
```

```
type(life_expectancy)
```

```
<class ‘dict’>
```

Example 3

```
# Some countries in the given Africa regions

africa_regions = {"East Africa": ["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda","Burundi"],
                 "North Africa": ["Algeria", "Egypt", "Libya", "Morocco", "Sudan", "Tunisia"],
                 "West Africa": ["Nigeria", "Ghana" "Senegal", "Benin", "Liberia", "Mali", "Niger"],
                  "Central Africa": ["Angola", "Cameroon", "Chad", "Congo", "DRC", "Gabon"],
                  "Southern Africa": ["Botswana", "Eswatini", "Lesotho", "Namibia", "RÈunion", "South Africa"]}

print(africa_regions)
```

```
{‘East Africa’: [‘Ethiopia’, ‘Kenya’, ‘Rwanda’, ‘Somalia’, ‘South Sudan’, ‘Uganda’, ‘Burundi’],
‘North Africa’: [‘Algeria’, ‘Egypt’, ‘Libya’, ‘Morocco’, ‘Sudan’, ‘Tunisia’], ‘West Africa’:
[‘Nigeria’, ‘GhanaSenegal’, ‘Benin’, ‘Liberia’, ‘Mali’, ‘Niger’], ‘Central Africa’: [‘Angola’,
‘Cameroon’, ‘Chad’, ‘Congo’, ‘DRC’, ‘Gabon’], ‘Southern Africa’: [‘Botswana’, ‘Eswatini’,
‘Lesotho’, ‘Namibia’, ‘RÈunion’, ‘South Africa’]}
```

```
type(africa_regions)
```

```
<class ‘dict’>
```

Dictionary Length

To determine how many items a dictionary has, use the **len()** function.

Example 1

```
life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67, "Rwanda": 65
,
                   "Tanzania": 64, "Somalia": 54}

len(life_expectancy)
```

8

Example 2

```
africa_regions = {"East Africa": ["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda",
"Burundi"],
                   "North Africa": ["Algeria", "Egypt", "Libya", "Morocco", "Sudan", "Tunisia"],
                   "West Africa": ["Nigeria", "Ghana" "Senegal", "Benin", "Liberia", "Mali", "Niger"],
                     "Central Africa": ["Angola", "Cameroon", "Chad", "Congo", "DRC", "Gabon"],
                     "Southern Africa": ["Botswana", "Eswatini", "Lesotho", "Namibia", "RÈunion", "South Afri
ca"]}

len(africa_regions)
```

5

Accessing dictionary items

Dictionary items are presented in key: value pairs, and can be referred to by using the key name. To access a specific value in the dictionary data set, you need to index the right **key**. Dictionaries in Python are mutable and the elements in a dictionary can be added, removed, modified, and changed accordingly.

You can access the items of a dictionary by referring to its key name, inside square brackets. For example, consider life expectancy dictionary:

```
life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67, "Rwanda": 65
, "Tanzania": 64, "Somalia": 54}
```

To access the value in the key Nigeria, we use:

```
life_expectancy["Nigeria"]
```

60

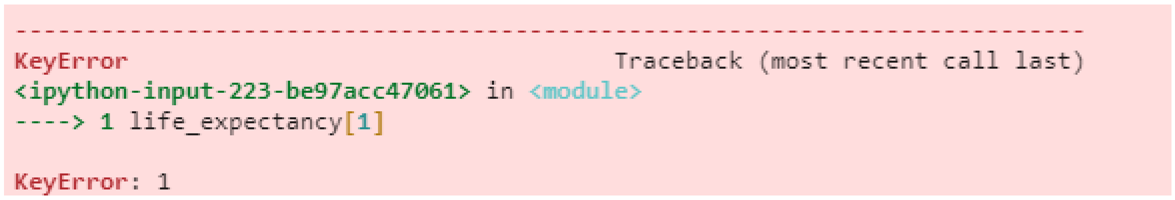
Also, for Ethiopia, we use:

```
life_expectancy["Ethiopia"]
```

68

You cannot access items in a dictionary by index

```
life_expectancy[1]
```



```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-223-be97acc47061> in <module>
----> 1 life_expectancy[1]

KeyError: 1
```

Dictionary Methods

Methods in a dictionary are as follows:

- **.keys()**
- **.values()**
- **.items()**
- **.updates()**

.keys()

The **.keys()** method will return a list of all the keys in the dictionary.

Example 1

```
life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67, "Rwanda": 65
,
                   "Tanzania": 64, "Somalia": 54}

life_expectancy.keys()
```

dict_keys([‘Nigeria’, ‘Kenya’, ‘Uganda’, ‘Ethiopia’, ‘Sudan’, ‘Rwanda’, ‘Tanzania’, ‘Somalia’])

Example 2

```
africa_regions = {"East Africa": ["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda",
"Burundi"],
                  "North Africa": ["Algeria", "Egypt", "Libya", "Morocco", "Sudan", "Tunisia"],
                  "West Africa": ["Nigeria", "Ghana" "Senegal", "Benin", "Liberia", "Mali", "Niger"],
                   "Central Africa": ["Angola", "Cameroon", "Chad", "Congo", "DRC", "Gabon"],
               "Southern Africa": ["Botswana", "Eswatini", "Lesotho", "Namibia", "RÈunion", "South Africa"]
}

africa_regions.keys()
```

dict_keys([‘East Africa’, ‘North Africa’, ‘West Africa’, ‘Central Africa’, ‘Southern Africa’])

.values()

The **.values()** method will return a list of all the values in the dictionary.

Example 1

```
life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67, "Rwanda": 65
,
              "Tanzania": 64, "Somalia": 54}

life_expectancy.values()
```

dict_values([60, 69, 68, 68, 67, 65, 64, 54])

Example 2

```
africa_regions = {"East Africa": ["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda",
"Burundi"],
               "North Africa": ["Algeria", "Egypt", "Libya", "Morocco", "Sudan", "Tunisia"],
               "West Africa": ["Nigeria", "Ghana" "Senegal", "Benin", "Liberia", "Mali", "Niger"],
                "Central Africa": ["Angola", "Cameroon", "Chad", "Congo", "DRC", "Gabon"],
                "Southern Africa": ["Botswana", "Eswatini", "Lesotho", "Namibia", "RÈunion", "South Afri
ca"]}

africa_regions.values()
```

dict_values([[‘Ethiopia’, ‘Kenya’, ‘Rwanda’, ‘Somalia’, ‘South Sudan’, ‘Uganda’, ‘Burundi’], [‘Algeria’, ‘Egypt’, ‘Libya’, ‘Morocco’, ‘Sudan’, ‘Tunisia’], [‘Nigeria’, ‘GhanaSenegal’, ‘Benin’, ‘Liberia’, ‘Mali’, ‘Niger’], [‘Angola’, ‘Cameroon’, ‘Chad’, ‘Congo’, ‘DRC’, ‘Gabon’], [‘Botswana’, ‘Eswatini’, ‘Lesotho’, ‘Namibia’, ‘RÈunion’, ‘South Africa’]])

.items()

The **items()** method will return each item in a dictionary, as tuples in a list.

Example 1

```
life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67, "Rwanda": 65
,
              "Tanzania": 64, "Somalia": 54}

life_expectancy.items()
```

dict_items([(‘Nigeria’, 60), (‘Kenya’, 69), (‘Uganda’, 68), (‘Ethiopia’, 68), (‘Sudan’, 67), (‘Rwanda’, 65), (‘Tanzania’, 64), (‘Somalia’, 54)])

Example 2

```
africa_regions = {"East Africa": ["Ethiopia", "Kenya", "Rwanda", "Somalia", "South Sudan", "Uganda",
"Burundi"],
               "North Africa": ["Algeria", "Egypt", "Libya", "Morocco", "Sudan", "Tunisia"],
               "West Africa": ["Nigeria", "Ghana", "Senegal", "Benin", "Liberia", "Mali", "Niger"],
                  "Central Africa": ["Angola", "Cameroon", "Chad", "Congo", "DRC", "Gabon"],
                  "Southern Africa": ["Botswana", "Eswatini", "Lesotho", "Namibia", "RÈunion", "South Afri
ca"]}

africa_regions.items()
```

dict_items([(‘East Africa’, [‘Ethiopia’, ‘Kenya’, ‘Rwanda’, ‘Somalia’, ‘South Sudan’, ‘Uganda’, ‘Burundi’]), (‘North Africa’, [‘Algeria’, ‘Egypt’, ‘Libya’, ‘Morocco’, ‘Sudan’, ‘Tunisia’]), (‘West Africa’, [‘Nigeria’, ‘Ghana’, Senegal’, ‘Benin’, ‘Liberia’, ‘Mali’, ‘Niger’]), (‘Central Africa’, [‘Angola’, ‘Cameroon’, ‘Chad’, ‘Congo’, ‘DRC’, ‘Gabon’]), (‘Southern Africa’, [‘Botswana’, ‘Eswatini’, ‘Lesotho’, ‘Namibia’, ‘RÈunion’, ‘South Africa’])])

.update

The **.update()** method will update the dictionary with the items from the given argument. The argument must be an object with key: value pairs.

```
life_expectancy = {"Nigeria": 60, "Kenya": 69, "Uganda": 68, "Ethiopia": 68, "Sudan": 67, "Rwanda": 65
,
                "Tanzania": 64, "Somalia": 54}
```

You were given the life expectancy of Burundi as 70. You can add that to the existing dictionary by using:

```
life_expectancy.update({"Burundi": 67})

life_expectancy
```

{‘Nigeria’: 60, ‘Kenya’: 69, ‘Uganda’: 68, ‘Ethiopia’: 68, ‘Sudan’: 67, ‘Rwanda’: 65, ‘Tanzania’: 64, ‘Somalia’: 54, ‘Burundi’: 67}

You can also change the value of a specific item by referring to its key name. Let’s say the government of Nigeria said that the life expectancy of Nigeria is 70 and not 60. We can change this by using:

```
life_expectancy["Nigeria"] = 70

life_expectancy
```

{‘Nigeria’: 70, ‘Kenya’: 69, ‘Uganda’: 68, ‘Ethiopia’: 68, ‘Sudan’: 67, ‘Rwanda’: 65, ‘Tanzania’: 64, ‘Somalia’: 54}

Another example:

```
life_expectancy["Eritrea"] = 66

life_expectancy
```

{‘Nigeria’: 60, ‘Kenya’: 69, ‘Uganda’: 68, ‘Ethiopia’: 68, ‘Sudan’: 67, ‘Rwanda’: 65, ‘Tanzania’: 64, ‘Somalia’: 54, ‘Burundi’: 67, ‘Eritrea’: 66}

Removing Items in a dictionary

The **del** keyword removes the item with the specified key name.

Example 1

For example, to delete information about Eritrea in the life expectancy dictionary we use:

```
del life_expectancy["Eritrea"]

life_expectancy
```

{‘Nigeria’: 60, ‘Kenya’: 69, ‘Uganda’: 68, ‘Ethiopia’: 68, ‘Sudan’: 67, ‘Rwanda’: 65, ‘Tanzania’: 64, ‘Somalia’: 54, ‘Burundi’: 67}

Example 2

To delete information about Burundi, we use:

```
del life_expectancy["Burundi"]

life_expectancy
```

{‘Nigeria’: 60, ‘Kenya’: 69, ‘Uganda’: 68, ‘Ethiopia’: 68, ‘Sudan’: 67, ‘Rwanda’: 65, ‘Tanzania’: 64, ‘Somalia’: 54}

Class Activity 11 (Peer to Peer Review Activity)

Peer to Peer Interaction

Visit the LMS, locate forum activity and participate in the discussion

Just like the life expectancy data, the number of confirmed COVID-19 cases were also given and shown below:

Burundi	694
Ethiopia	113295
Ghana	52198
Kenya	88380
Nigeria	68937
Rwanda	6129
Somalia	4525
South Sudan	3166

Sudan	19196
Uganda	22499

- Create a dictionary for the given data and name COVID_19
- How many keys are in the data?
- Remove Nigeria from the data
- Add Tanzania with the value 509

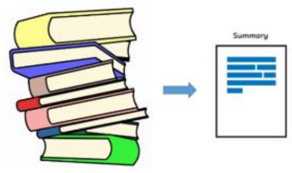
Class Activity 12

Consider the dictionary:

```
d = {"fruits": ["apples", "oranges", "pears", "mangoes"],
"vegetables": ["tomatoes", "lettuce", "spinach", "green peppers"],
"meat": ["chicken", "fish", "beef", "ostrich"],
"dairy": ["yogurt", "milk", "cheese", "ice-cream"] }
```

1. How many keys does d have?
2. List the values of d
3. How do you access spinach using the dictionary d?
4. How do you add a new fruit?

Summary of Study Unit 2



In this study unit, you have learnt that:

1. Data structure in Python includes list, tuple, set and dictionary
2. input() function can be used to take information from the user and that the value of the variable taken is always a string data type.
3. + operator is used for string concatenation while * operator is used for string repetition
4. The len() function is used to get the number of characters in a string.
5. This len() function can also be used to get the number of elements in a list, tuple, and set.