2.1 What is python

Python is a very popular *general-purpose interpreted, interactive, object-oriented, and high-level programming language*. It is dynamically-typed and garbage-collected programming language. It supports multiple programming paradigms*, including Procedural, Object Oriented and Functional programming language*. Python design philosophy emphasizes code readability with the use of significant indentation.

Python was created by Guido van Rossum during 1985-1990 at the National Research Institute for Mathematics and Computer Science in the Netherlands. It was later released in 1991. It's now maintained by a core development team at the institute though Guido van Rossum still holds a vital role in directing its progress.

Python is derived from many other languages, including *ABC, Modula-3, C, C++, Algol-68, Smalltalk, and Unix shell* and other scripting languages.

2.2 Why python?

It is consistently rated as one of the world's most popular programming languages. It has lower learning curve. Many schools, Colleges and Universities are teaching Python as their primary programming language.

Other good reasons which make Python as the top choice of any programmer include

i). Python is Open Source which means its available at no cost at all.
ii). It works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
iii). Has less learning curve
iv). Its versatility and can be used to create many different kinds of applications
v). Python has powerful development inbuilt libraries include AI, ML etc.
vi). Python is Interpreted meaning its processed at runtime by the interpreter. You therefore do not need to compile your program before executing it which makes it faster to execute your programs

vii). Python is Interactive meaning you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

viii). Python is Object-Oriented meaning the programmers can benefit from OOP concepts such as inheritance, abstraction, polymorphism and encapsulation

2.3 What python can do

Being a general-purpose, software engineers can make applications in almost every domain of software development. Python makes its presence in every emerging field. It is the fastest-growing programming language and can develop any application.

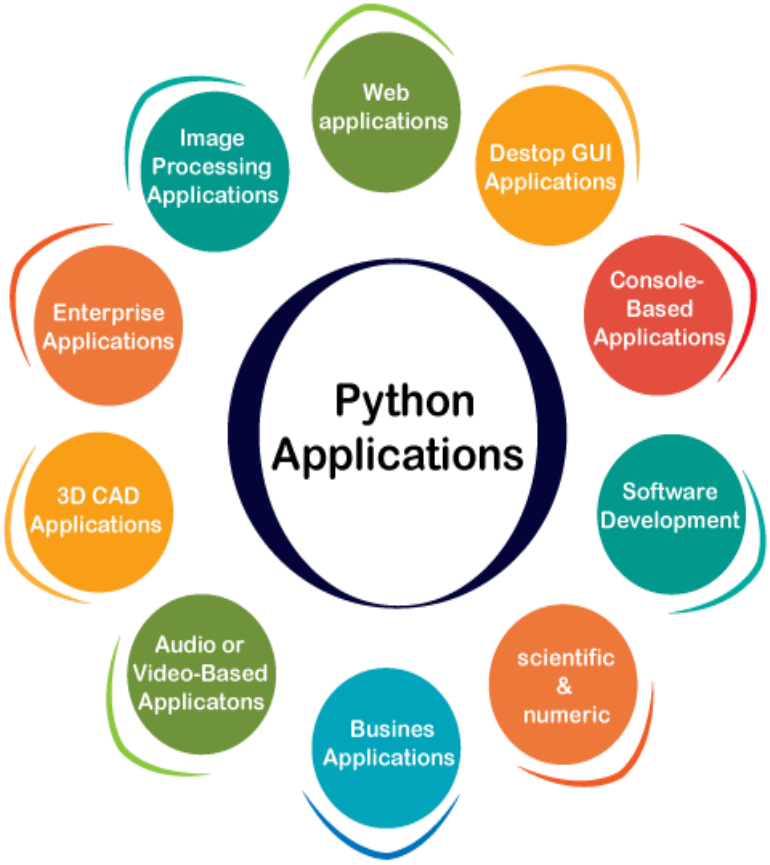

Fig 1.0-Phython applications

2.3.1 Web Applications

Python is used extensively to develop web applications. It provides libraries to handle internet protocols such as

HTML and XML, JSON, Email processing, request. Django is used on Instagram. The language provides useful frameworks, such as

- Django & Pyramid framework (Use for heavy applications)
- Flask and Bottle (Micro-framework)
- Plone and Django CMS

2.3.2 Desktop GUI Applications

Python provides a **Tk GUI library** to develop a user interface. Some popular GUI libraries are given below.

- Tkinter or Tk
- wxWidgetM
- Kivy (used for writing multitouch applications)
- PyQt or Pyside

2.3.3 Console-based Application

They are application that run from the command-line/ shell. It provides many free library or module which helps to build the command-line apps. There are also advance libraries that can develop independent console apps.

2.3.4 Software Development

It works as a support language and can be used to build control and management, testing, etc.

- **SCons** is used to build control.
- **Buildbot** and **Apache Gumps** are used for automated continuous compilation and testing.
- **Round or Trac** for bug tracking and project management.

2.3.5 Scientific and Numeric

Python is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations.

Python has many libraries for scientific and numeric such as **Numpy, Pandas, Scipy, Scikit-learn**, etc. Few popular

frameworks of machine libraries include ***SciPy,Scikit-learn, NumPy, Panda, Matplotlib***

2.3.6 Business Applications

E-commerce and ERP are an example of a business application. This kind of application requires extensively, scalability and readability, and Python provides all these features.

Oddo is an example of the all-in-one Python-based application which offers a range of business applications. Python provides a Tryton platform which is used to develop the business application.

2.3.7 Audio or Video-based Applications

Python can used to perform multiple tasks and can be used to create multimedia applications. Example of multimedia applications made by using Python a include TimPlayer, ***cplay,*** etc. The few multimedia libraries exposed by python include ***Gstreamer, Pyglet, QT Phonon***

2.3.8 3D CAD Applications

Python can create a 3D CAD application by using the following functionalities ***Fandango (Popular), CAMVOX, HeeksCNC, AnyCAD, RCAM***

2.3.9 Enterprise Applications

Python can be used to create applications that can be used within an Enterprise or an Organization. Some real-time applications are OpenERP, Tryton, Picalo, etc.

2.3.10 Image Processing Application

Python contains many libraries that are used to work with the image. Some libraries of image processing include *OpenCV, Pillow and SimpleITK*

2.4 Python versions and IDE

The most recent major version of Python is Python 3, which we shall be using in this module. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

It is possible to write Python in an IDE such as ***Thonny, Pycharm, Netbeans*** *or* ***Eclipse*** which are particularly useful when managing larger collections of Python files.

2.5 Python Syntax

Python Syntax compared to other programming languages

Python was designed for readability, and has some similarities to the English language with influence from mathematics.

Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.

Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

2.6 Python Identifiers

An identifier refers to the name we give to various programming elements i.e. *variable, function, class, module or another object*.

Rules for naming identifies in python

i). An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

ii). Python does not allow punctuation characters such as @, $, and % within identifiers.

iii). Python is a case sensitive programming language. Thus, Camel and camel are two different identifiers in Python.

iv). Class names start with an uppercase letter. All other identifiers start with a lowercase letter.

2.7 Reserved Words

These are words that have a special/inbuilt meaning to that programming language and cannot be used as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only. Some include

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

2.8 Lines and Indentation

Python provides no braces like other programing languages to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is strictly enforced.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

For example

```
if a>b:
  print "A is greater than B"
else:
  print "B is greater than A"
```

Python will give you a syntax error if you skip the indentation:

```
if a>b:
print "A is greater than B"
else:
 print "B is greater than A"
```

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

```
if a>b:
print "A is greater than B"
else:
```

```
        print "B is greater than A"
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you a syntax error

```
if a>b:
print "welcome to python programming "
        print "B is greater than A"
```

2.9 Multi-Line Statements

Statements in Python characteristically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue as shown below

```
fullNames = firs_name + \
            middle_name+ \
            last_name
```

2.10 Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

Triple quotes are used to span the string across multiple lines. For example, all the following are legal

```
word = 'paragraph'
sentence = "This is a paragraph."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

2.11 Python and comments

Python interpreter ignores them. Comments in programming are used to

i). Document/explain code.
ii). Make the code more readable.
iii). Prevent execution when testing code.

Comments starts with a #, and Python will ignore them e.g.

```
#This is a comment
```

```
print("Hello, World!")
```

Python does not really have a syntax for multi-line comments as other programming languages

To add a multiline comment, you could insert a # for each line:

You can as well use a multiline string as shown in the example below. Python will ignore any string literal that hasn't been assigned to a variable

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

2.12 Waiting for the User

The following line of the program displays the prompt, the statement saying "Press the enter key to exit", and waits for the user to take action –

```
raw_input("\n\nPress the enter key to exit.")
```

Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

2.13 Python Variables

Variables are programmers' memory locations/containers for storing data values.

Here is no command for declaring a variable in python. A variable is created the moment you first assign a value to it. You do not need to specify any particular type when declaring a variable in python, and can even change type after they have been set. Examples of variable declaration are shown below

```
a = 10 #integer variable
name= "joe" #string variable
present=”yes” #boolean variable
salary=10000.50 #floa variable
```

2.13.1 Assigning variables values

Python allows you to assign values to multiple variables in one line as shown below

```
x, y, z = "crown", "Benz", "Passat"
print (x ,y ,z)
```

Always make sure the number of variables matches the number of values, or else you will get an error.

We can also assign the same value to multiple variables in one line as shown below

```
x, y, z = "benz”
print (x ,y ,z)
```

2.13.2 printing out variable values

we use the Python print () function to output variables as shown in example below

```
a = 5
print(5)
```

To print multiple values, we separate variables with commas as shown below

```
Car1 = "Benz"
Car2 = "Passat"
Car3= "Prado"
print(Car1,Car2,Car3)
```

2.13.3 Global Variables

These are variables created outside of a function hence they can be accessed by everyone i.e., both inside of functions and outside. An example is shown below

```
x=5
def myfunc():
 y=6
 print ("The sum is " + (x+y))
myfunc()
```

If a variable with the same name is created inside a function, this variable will be local, and can only be used inside that function. The global variable with the same name will remain

as it was, global and with the original value. An example is shown below

```
x = "5"
def myfunc():
  x= "7"
  print("Value of x is " + x)

myfunc()
print("Value of x is " + x)
```

2.13.4 using global Keyword

A variable inside a function, that variable is local, and can only be used inside that function. To create a global variable inside a function, we use the global keyword. An example is presented below

```
def myfunc():
  global x="5"
myfunc()

print("Python is " + x)
```

We can also use the global **keyword** if you want to change a global variable inside a function. As shown in the example below

```
x = "5"
def myfunc():
  global x
  x = "5"
print("X= " + x)
myfunc()
print("x= " + x)
```

2.14 Python data Types

In every programming, data type is such an essential concept. Variable are used to store data of different values. python supports the following

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

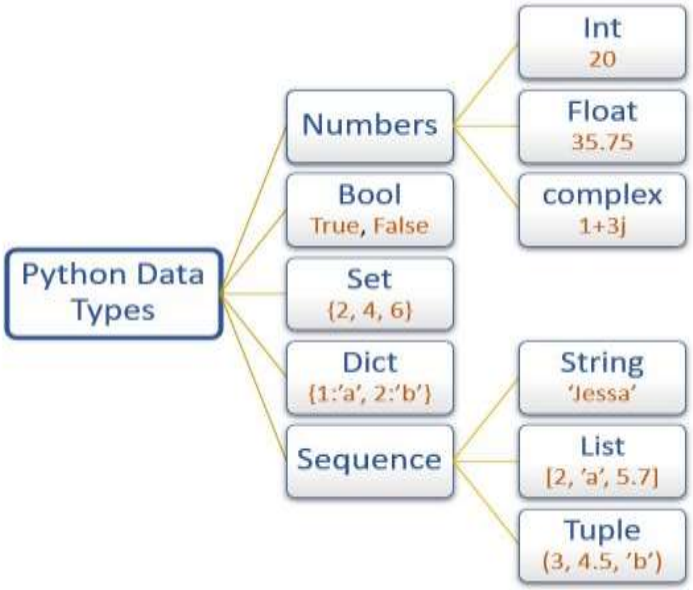

Fig 2.0-Python data types

Python has the following data types built-in by default, in these categories:

Table 1.0 Python primitive/inbuilt data types

In python, data type is set when you assign a value to a variable. If you want to specify the data type, you can use the following constructor functions

Example	Data Type
a = str("hello")	str
a = int(5)	int
a = float(100.7)	float
a = complex(1j)	complex
cars = list(("benz", "passat",))	list

cars = tuple(("benz", "passat"))	tuple
a = range(8)	range
person = dict(name="joe", age=18)	dict
cars= set(("benz", "passat"))	set
cars = frozenset(("benz", "prado"))	frozenset
a = bool(5)	bool
a= bytes(5)	bytes
a = bytearray(5)	bytearray
a = memoryview(bytes(5))	memoryview

Table 2.1Data type constructor methods

2.15 Python Numbers

There are three numeric types in Python; int, float and complex

Example

```
a= 5                    # int
salary  = 100000.98     # float
c = 1j                  # complex
```

2.15.1 Integers

They are used to store whole number, positive or negative, without decimals, of unlimited length.

2.15.2 Float

They are used to store a number, positive or negative, containing one or more decimals. Floats can also be scientific numbers with an "e" to indicate the power of 10.

2.15.3 Complex

They are numbers written with a "j" as the imaginary part. Some examples are shown below

```
a = 7+5j
b = 4j
c = -2j
```

Specify a Variable Type

There may be times when you want to specify a type on to a variable. Python being object-orientated language uses classes to define data types, including its primitive types.

```
a = int(1)   # x will be 1
b= int(5.9) # y will be 5
c= int("8") # z will be 3
d = float(7)     # x will be 7.0
x = str("str1") # x will be 'str1'
y = str(10)    # y will be '10'
z = str(5.8)  # z will be '5.8'
```

2.16 Strings

In python, strings are surrounded by either single/double quotation marks as shown below

```
print("Hello")
print('Hello')
```

To assign a string multiline string to a variable, we use three quotes as shown below

```
a = """This is a example of a string,
which spans more than one line."""
print(a)
```

we can also use three single quotes as shown below

```
a = '''This is a example of a string,
which spans more than one line.'''
print(a)
```

Manipulating strings in Python

Just like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.

There are a number of operations we can perform on strings as discussed below

2.16.1 Accessing string element

We use square brackets can be used to access elements of the string i.e., name[index]. The following example

accesses and displays the third element of array known a name.

```
name = "Hello, World!"
print(name[2])
```

2.16.2 String Concatenation

To combine, two strings in python, we you can use the + operator as show in the following basic example

```
first = "Joe"
otherNames = "Chris"
fullNames = first + " " + otherNames
print(fullNames)
```

2.16.3 String Length

To get the length of a string in python, we the len() function. The following return the length of a string known a name

```
name = "Hello, World!"
print(len(name))
```

2.16.4 Check for a String

To check if a certain phrase or character is present in a string, we can use the keyword in.

```
str1 = 'The best things in life is when you love what you are doing'
print("best" in str1)
```

we can also use python if statement:

```
str1= " The best things in life is when you love what you are doing'"
if "best" in str1:
  print("Yes, 'best' is present.")
```

2.16.5 Check if NOT

In python, we use keyword NOT IN To check if a certain phrase or character is present in a string or not

For example, to check if "programming " is NOT present in the following text, we can use the following python code

```
str1 = 'The best things in life is when you love what you are doing'
print("programming" not in str1)
```

we can also use an if statement as shown below

```
str1= " The best things in life is when you love what you are doing'"
if "programming" not in str1:
  print("Yes, 'programming ' is absent.")
```

2.16.6 String slicing

It's possible to return a range of characters by using the slice syntax. We specify the start and end index separated by a colon, to return a part of the string.

For example, to return characters from 3 to position 8 of a string known as str1, we can use the following code

```
str1 = 'The best things in life is when you love what you are doing'
print(str1[2:8])
```

2.16.7 slicing from start

By leaving out the start index, the range will start at the first character. The following example slices the string from start up to character seven

```
str1 = 'The best things in life is when you love what you are doing'
print(str1[:7])
```

2.16.8 Slice To the End

By leaving out the *end* index, the range will go to the end. The following example slices the string from position 11 up to end

```
str1 = 'The best things in life is when you love what you are doing'
print(str1[10:])
```

2.16.9 Python string built-in methods

1. To convert a string into upper case, we use upper () method e.g.

```
Str1 = "Hello"
print(Str1.upper())
```

2. To convert a string into upper case, we use lower () method e.g.

```
Str1 = "Hello"
print(Str1.lower())
```

3. To remove whitespace before and/or after the actual text, we use strip() in python e.g.

```
Str1 = " Hello, welcome "
print(Str1.strip())
```

4. To replace a string with another string we use replace() in python. An example is presented below

```
str1 = "Hello, Welcome to python"
print(str1.replace("Hello", "Hi")
```

5. The split () method splits the string into substrings if it finds instances of the separator:

```
str1 = "Hello/ Welcome to python"
print(str1.split("/"))
```

2.17 Escape Character

An escape character in python are preceded by a backslash \ followed by the character you want to insert.

The following example will generate an error if you use double quotes inside a string that is surrounded by double quotes:

String1 = "Python is “general” purpose programming language."

We can fix the problem using the escape character \" as shown below

String1 = "Python is \“general”\ purpose programming language."

Other escape characters used in Python:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed

2.18 Python Boolean

Booleans represent one of two values: True or False. When writing your programs, you may need to if an expression is True or False. Anytime you compare two values, the expression is evaluated and Python returns the Boolean answer

Simple example are presented below

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

consequently, if run a condition in an if statement, Python returns True or False:

```
x = 101
y = 100

if x > y:
    print("x is greater than y")
else:
    print("y is not greater than x")
```

2.19 Python Collections (Arrays)

There are four collection data types supported by Python programming language:

- **List** is a collection which is *ordered* and *changeable* and allows *duplicate members.*
- **Tuple** is a collection which is ordered and *unchangeable* but *allow duplicate members.*
- **Set** is a collection which is *unordered*, unchangeable/immutable, and unindexed. It doesn't allow duplicate members.
- **Dictionary** is a collection which is *ordered* and *changeable*. No duplicate members.

*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

**As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

2.19.1 Python List

To create a list in python, we use square brackets as shown in the example below

```
cars = ["volvo", "benz", "passat"]
print(cars)
```

we can also use the list () Constructor when creating a as shown below

```
cars = list(["volvo", "benz", "passat"])
print(cars)
```

Items in a list are ordered, changeable(mutable), and allow duplicate values.

To reference to items in a list, we use item e.g. the first item has index [0], the second item has index [1] etc.

Items in a list item can be of data any type; string, int, Boolean, float e.t.c. A list can contain values of different data types as shown below

```
student = ["Chris Joe", "Male", 20, 6.2, True]
print(student)
```

Operations on list

1. To determine how many items a list has, use the len() function as shown in example below

```
cars = ["volvo", "benz", "passat"]
print(len(cars))
```

2. Accessing List Items

- To access an individual item in a list, we use the index number e.g. to display the third element in our list cars, we can use the following python code

```
cars = list(["volvo", "benz", "passat"])
print(cars[2])
```

Note: The first item has index 0.

- We can also use negative Indexing whereby indexing start from the end with -1 refers to the last item, -2 refers to the second last item etc. To print the second last item in our list, we can use the following code

```
cars = list(["volvo", "benz", "passat"])
print(cars[-1])
```

- We can also specify range of index from start to end. The result shall always be a new list with the specified items e.g. to return the second third items in our list we can use the following code

```
cars = list(["volvo", "benz", "passat"])
print(cars[1:3])
```

Note: The search will start at index 1 (included) and end at index 3 (not included).

1. Remember that the first item has index 0.

- By leaving out the start value, the range will start at the first item. Example below returns the first two items in our list

```
cars = list(["volvo", "benz",
"passat"])
print(cars[:2])
```

- By leaving out the end value, the range will go on to the end of the list. The following will return the second item until the last item in our list

```
cars = list(["volvo", "benz",
"passat"])
print(cars[1:])
```

3. Check if Item Exists

- To determine if a specified item is present in a list, we can use in keyword as shown in example below

```
cars = list(["volvo", "benz",
"passat"])
print("benz" in cars)
```

- We can also use the if keyword in python as shown below

Example

```
cars = list(["volvo", "benz", "passat"])
search = input("Enter item to search for\n")
if search in cars:
    print("item found in the list")
else:
    print("item not found in the list")
```

4. Change Item Value

- To change the value of a specific item, we use the index number as shown below

```
cars = list(["volvo", "benz", "passat"])
cars[0] = 'subaru'  # update first item
with subaru
print(cars)
```

- To change the value of items within a specific range, define a list with the new values, and refer to the

range of index numbers where you want to insert the new values as shown in example below

```
cars = list(["volvo", "benz",
"passat"])
cars[1:3] = ['prado', 'audi']  # update
second and third item
```

- If you insert more items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly as shown by example below

```
cars = list(["volvo", "benz", "passat"])
cars[1:2] = ['prado', 'audi']  # update
second and third item
print(cars)
```

Note: The length of the list will change when the number of items inserted does not match the number of items replaced.

5. Insert Items

- To insert a new list item, without replacing any of the existing values, we can use the insert() method. The item shall be inserted at the specified index as shown below

```
cars = list(["volvo", "benz",
"passat"])
cars.insert(1, "BmW")
print(cars)
```

Note: As a result of the example above, the list will now contain 4 items.

- To add an item to the end of the list, we use the append () method as shown in example below

```
cars = list(["volvo", "benz",
"passat"])
cars.append("bmw")
print(cars)
```

6. Extend List

To append elements from *another list* to the current list, use the extend() method as shown in the following python code

```
cars = list(["volvo", "benz", "passat"])
cars2 = ["probox", "Nissan", "Toyotta"]
cars.extend(cars)
print(cars)
```

The elements will be added to the *end* of the list.

7. Remove List Items

- To remove Specified Item, we use the remove() method as shown in example below.

```
cars = list(["volvo", "benz",
"passat"])
cars.remove("benz")
print(cars)
```

- To remove an item based on specified Index, we use pop() method as shown below

```
cars = list(["volvo", "benz",
"passat"])
cars.pop(0)
print(cars)
```

- we can also use the del keyword as shown below

```
cars = list(["volvo", "benz",
"passat"])
del cars[2]
print(cars)
```

- to remove the last item, we use the pop() method without specifying the index as shown below

```
cars = list(["volvo", "benz", "passat"])
cars.pop()
print(cars)
```

8. To delete a list, we use del keyword as shown below

```
cars = list(["volvo", "benz", "passat"])
del cars[]
print(cars)
```

9. Clear the List

We use the clear () method to empty the list. However, the list still remains but with no content. An example is presented below

```
cars = list(["volvo", "benz", "passat"])
cars.clear()
print(cars)
```

2.19.2 Python and Tuple

Tuples are used to store multiple items in a single variable and are defined using round brackets. An example is shown below

```
countries = ("Kenya", "United Kingdom", "United States", "Congo")
print(countries)
```

Items in a tuple are *ordered, unchangeable/immutable, and allow duplicate values.* Tuple items are indexed, the first item has index [0], the second item has index [1] etc. Since tuples are , we can have items with the same value

```
countries = ("Kenya", "United Kingdom", "United States", "Congo", "Kenya")
print(countries)
```

Create Tuple with One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
countries = ("Kenya",)
print(countries)
```

It is also possible to use the tuple () constructor to make a tuple. as demonstrated below

```
mypc = tuple(("Hp", "16 GB RAM", "500TB", "Icore 7", 98000.00, True))
print(mypc)
```

Tuple Items - Data Types

Tuple items can be of any data type and a tuple can contain different data type just like a list as shown below

```
mypc = ("Hp", "16 GB RAM", "500TB", "Icore 7", 98000.00, True)
print(mypc)
```

Tuple operations

1. **Tuple Length**

To determine tuple Length, we use the len() function as shown below

```
countries = ("Kenya", "United Kingdom", "United States", "Congo", "Kenya")
print(len(countries))
```

2. **Access Tuple Items**
   - We can access tuple items by referring to the index number, inside square brackets. An example is presented below which prints the second item in our tuple

```
mypc = tuple(("Hp", "16 GB RAM",
"500TB", "Icore 7", 98000.00, True))
print(mypc[1])
```

Note: The first item has index 0.

- We can also use negative indexing whereby the items are accessed from the end. To print the last item on the tuple, we can use the following python code

```
mypc = tuple(("Hp", "16 GB RAM",
"500TB", "Icore 7", 98000.00, True))
print(mypc[-1])
```

- We can also specify a range of indexes by specifying start and end range. The result shall be a new tuple with the specified items.

```
mypc = tuple(("Hp", "16 GB RAM", "500TB",
"Icore 7", 98000.00, True))
print(mypc[1:4])
```

The above code returns the second, third and fourth item in the tuple

Note: The search will start at index 1 (included) and end at index 4 (not included).

Remember that the first item has index 0.

- By leaving out the start value, the range will start at the first item:

```
mypc = tuple(("Hp", "16 GB RAM",
"500TB", "Icore 7", 98000.00, True))
print(mypc[:4])
```

The above example returns the items from the beginning with index 4 not included

- By leaving out the end value, the range will go on to the end of the list as demonstrated in example below

```
mypc = tuple(("Hp", "16 GB RAM",
"500TB", "Icore 7", 98000.00, True))
print(mypc[1:])
```

The above example returns the items from index one up to the end the end of the list

3. Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword as shown in example below

```
mypc = tuple(("Hp", "16 GB RAM", "500TB", "Icore
7", 98000.00, True))
searchvar=input("Enter HDD specs in TB\n")
if searchvar in mypc:
    print("A pc with such specs has been found")
else:
    print("No pc with this spec matches your
search criteria")
```

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created. But there are some workarounds.

4. Change Tuple Values

Once we create a tuple, we cannot change its values since they ate **unchangeable/ immutable**

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
mypc = tuple(("Hp", "16 GB RAM", "500TB", "Icore
7", 98000.00, True))
mylist = list(mypc)  # convert tuple into tuple
mylist[1] = "32 GB RAM"  # modify item 3
mypc = tuple(mylist)  # convert the list to tuple
print(mypc)
```

5. Add Items

Tuples being immutable, they do not have a build-in append () method. However, there other ways to add items to a tuple.

- We can convert tuple into a list: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

```
mypc = tuple(("Hp", "16 GB RAM", "500TB",
"Icore 7", 98000.00, True))
mylist = list(mypc)  # convert tuple into
tuple
mylist.append("Silver")  # add a new item
at the end
mypc = tuple(mylist)  # convert the list to
tuple
print(mypc)
```

- Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

```
mypc = tuple(("Hp", "16 GB RAM",
"500TB", "Icore 7", 98000.00, True))
mypc2 = tuple(("silver",))
```

```
mypc += mypc2
print(mypc)
```

2.19.3 Python and Set

A set is a collection which is unordered, unchangeable*, and unindexed. Set doesn't allow duplicate values hence duplicate values will always being ignored

Sets are defined with curly brackets.

*** Note:** Set *items* are unchangeable, but you can remove items and add new items.

To create a set-in python, we use the following code in python:

```
wishlist = {"fridge", "car", "fees"}
print(wishlist)
```

It is also possible to use the set() constructor to make a set as shown below

```
myWishList = set({"apple", "banana", "cherry",
"apple"})

print(myWishList)
```

Set items can be of any data type. A set can also contain different data types

```
shoppingList = set({"apple", 5, 25.00, True})

print(len(shoppingList))
```

1. Get the Length of a Set

To determine how many items a set has, use the len() function. An example is presented below

```
myWishList = set({"apple", "banana",
"cherry", "apple"})

print(len(myWishList))
```

2. Access Set Items

It's not possible to access items in a set by referring using an index/ key. You can however use

- for loop to loop through the items, or ask if a specified value is present in a set,

```
shoppingList = set({"apple", 5, 25.00,
True})
for i in shoppingList:
 print(i)
```

- use in keyword.

```
shoppingList = set({"apple", 5, 25.00,
True})
print("apple" in shoppingList)
```

3. Add items

Once a set is created, you cannot change its items, but you can add new items using the add() method. An example is shown below

```
shoppingList = set({"apple", 5, 25.00, True})
shoppingList.add("22/09/2022")
print(shoppingList)
```

1. Remove Item

- To remove an item in a set, use the remove()/ the discard() method. Remove apple using remove ()

```
shoppingList = set({"apple", 5, 25.00, True})
shoppingList.remove("apple")
print(shoppingList)
```

Note: If the item to remove does not exist, remove () will raise an error.

- Remove "25.00" by using the discard() method:

```
shoppingList = set({"apple", 5, 25.00, True})
shoppingList.remove(25.00)
print(shoppingList)
```

Note: If the item to remove does not exist, discard() will **NOT** raise an error.

- You can also use the pop() method to remove an item, but this method will remove the *last* item.

```
shoppingList = set({"apple", 5, 25.00, True})
x = shoppingList.pop()
print(x)
print(shoppingList)
```

2. The clear() method empties the set:

```
shoppingList = set({"apple", 5, 25.00, True})
shoppingList.clear()
print(shoppingList)
```

3. The del keyword will delete the set completely

```
 shoppingList = set({"apple", 5, 25.00, True})
del shoppingList
print(shoppingList)
```

2.19.4 Dictionary

Dictionaries are used to store data values in key: value pairs. It is *ordered*, changeable* and *do not allow duplicates.*

Dictionaries cannot have two items with the same key.

Creating dictionary in python

Dictionaries are written with curly brackets, and have keys and values. An example is defined below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
```

```
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
print(mypc)
```

Dictionary Items - Data Types

The values in dictionary items can be of any data type i.e., string, int, Boolean, and list data types:

Dictionary Length

We use the len () function to determine how many items a dictionary has. An example is presented below.

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
print(len(mypc))
```

dictionary operations

1. **Access Dictionary Items**
   - You can access the items of a dictionary by referring to its key name, inside square brackets. To get your PC speed, we can use the following python code

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
print(mypc["CPU speed"])
```

   - We can also use the get() as shown below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
x = mypc.get("CPU speed")
print(x)
```

2. **Get Keys**

The keys() method returns a list of all the keys in the dictionary.

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
```

```
x = mypc.keys()
print(x)
```

3. Add new item to Dictionary

Add a new item to the original dictionary, and see that the keys list gets updated as well:

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
x = mypc.keys()
print(x)
mypc["color"] = "silver"
print(x)
```

4. Get Values

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

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
x = mypc.values()
print(x)
```

If any changes is done on the dictionary, changes will reflect on the values list. An example is presented below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
x = mypc.values()
print(x)
mypc["price"] = 120000.00
print(x)
```

5. Get Items

Items() method will return each item in a dictionary, as tuples in a list inform of key: value pairs. An example is presented below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
x = mypc.items()
print(x)
mypc["price"] = 120000.00
print(x)
```

Every time the dictionary is updated, the items list gets updated as well

6. Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword. An example has been presented below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
searchvar=input("Enter key to search for\n")
if searchvar in mypc:
    print(searchvar+"\t" + 'has been found')
else:
    print(searchvar + "\t" + 'has not  been
found')
```

7. Change Values

You can change the value of a specific item by referring to its key name as shown below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
mypc["model"] = "lenovo"
print(mypc)
```

8. Update Dictionary

update() method will update the dictionary with the items from the given argument. An example is presented below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
mypc.update({"HDD": "1TB"})
print(mypc)
```

9. Python - Add Dictionary Items

This is done by Adding by using a new index key and assigning a value to it. An example is presented below

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
```

```
mypc["color"] = "black"
print(mypc)
```

10. Removing Items

There are several methods to remove items from a dictionary:

- The pop () method which removes the item with the specified key name:

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
mypc.pop("model")
print(mypc)
```

- The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
mypc.popitem()
print(mypc)
```

- The del keyword removes the item with the specified key name e.g.

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
del mypc['HDD']
print(mypc)
```

11. The del **keyword** deletes the entire the dictionary completely:

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
del mypc
print(mypc)
```

12. The clear() method empties the dictionary:

```
mypc = {
    "model": "hp",
    "RAM": "16GB",
    "HDD": "500TB",
    "price": 90000.00,
    "CPU speed": "Icore 7"
}
mypc.clear()
print(mypc)
```

2.20 Python Operators

Operators are used to perform operations on variables and values e.g. + operator is used to add together two values

Python divides the operators in the following 8 categories

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

2.20.1 Python Arithmetic Operators

These are category of operators used to in conjunction with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	a+b
-	Subtraction	a-b
*	Multiplication	A*b
/	Division	a/b
%	Modulus	A%b
**	Exponentiation	A**b
//	Floor division	a//b

Table 2:2 Arithmetic operators in Python

2.20.2 Python Assignment Operators

They are category of operators used to assign values to variables:

Operator	Example	Same As
=	a = 10	a= 10
+=	a+= 10	a = a+ 10
-=	a-= 7	a = a-7
*=	a *= 6	a = a * 6
/=	a /= 8	a = 8 / 3
%=	a %= 2	a = 2% 3
//=	a //= 8	a = a // 3
**=	a **= 9	a = a** 3

Table 2:3 Assignment operators in Python

2.20.3 Python Comparison Operators

They are set of operators used to compare two values

Operator	Name	Example
==	Equal	a == b
!=	Not equal	a!= b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

Table 2:4 Assignment operators in Python

2.20.4 Python Logical Operators

They are category of operators used to combine conditional statements

Operator	Description	Example
and	Returns True if both statements are true	Score>70 and score < 100
or	Returns True if one of the statements is true	Score==0 or score<=39
not	Negates a Boolean result, returns False if the result is true	not(a < 5 and b< 10)

Table 2:5 logical operators in Python

2.20.5 Python Identity Operators

They are used to compare the objects to confirm if they are actually the same object, with the same memory location

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Table 2:6 Identity operators in Python

2.20.6 Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example

in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Table 2:7 Identity operators in Python

2.21 Python & Casting

Type casting refers to the process a of converting variable from one data type to another. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- int () – converts a value into an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float () – converts a value to float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str () – converts a value to a string from a wide variety of data types, including strings, integer literals and float literals

```
# type casting using integers
x = int(1)   # x will be 1
y = int(2.8)   # y will be 2
z = int("3")   # z will be
```

```
# float type casting
x = float(1)   # x will be 1.0
y = float(5.4)   # y will be 2.8
z = float("6")   # z will be 6.0
w = float("4.2")   # w will be 4.2
```

```
x = str("s1")  # x will be 's1'
y = str(2)  # y will be '2'
z = str(3.0)  # z will be '3.0'
```

2.22 chapter summary

2.23 chapter exercise