

PYTHON – INSTALLATION

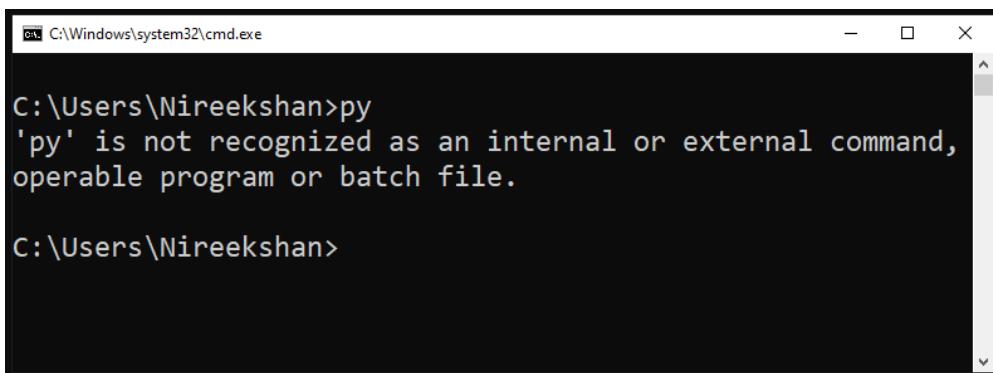
Contents

1. Checking python software installed or not in laptop/computer	2
2. Download python software	2
3. Important step while installing software	2
4. Check the python version.....	5

PYTHON – INSTALLATION

1. Checking python software installed or not in laptop/computer

- ✓ We can check python software is installed or not in laptop/computer.
- ✓ Open a command prompt in your laptop/computer.
- ✓ Type **py** command in command prompt and enter then check it.



```
C:\Windows\system32\cmd.exe
C:\Users\Nireekshan>py
'py' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Nireekshan>
```

- ✓ If python is not installed then we will get response as above screenshot.

2. Download python software

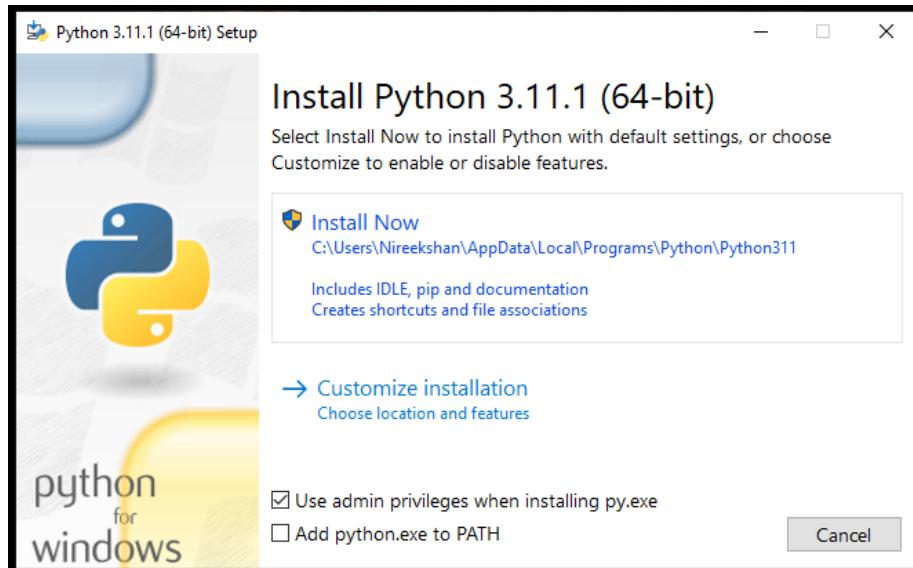
- ✓ We can download python software from official website, we can find url in below,

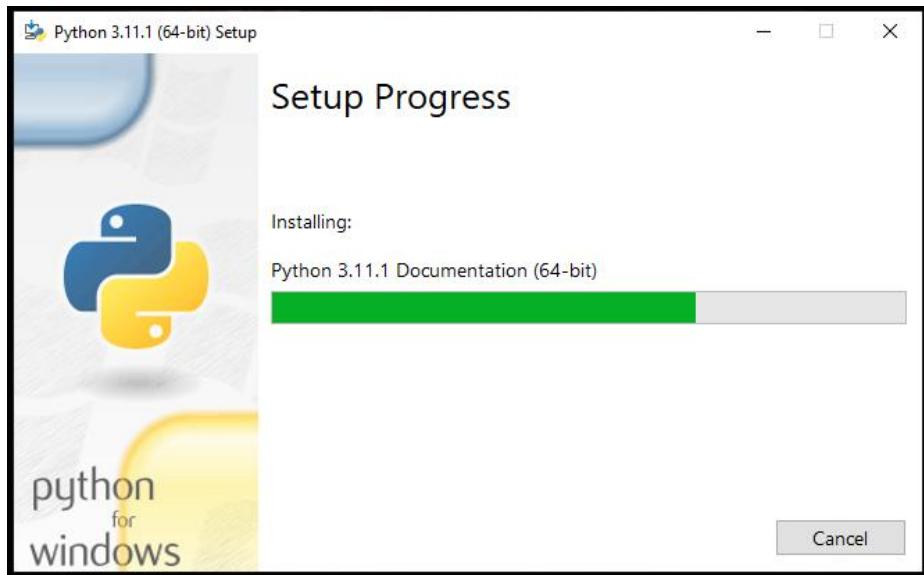
<https://www.python.org/downloads/>

3. Important step while installing software

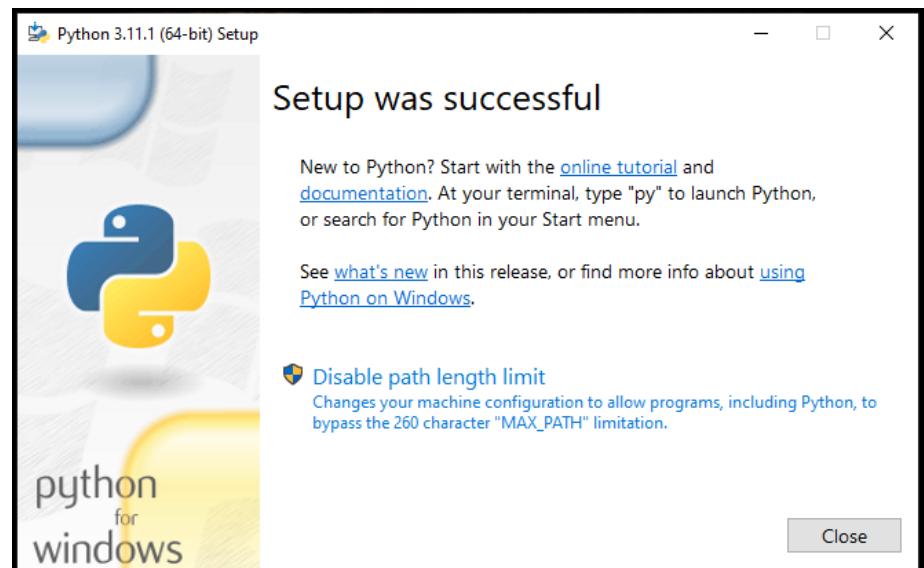
- ✓ Once python software downloaded then we can install by clicking the software.
- ✓ While installing we should check the checkbox of,
 - Add python 3.11.1 to PATH
- ✓ You can follow the below screenshot

Data Science – Python Installation



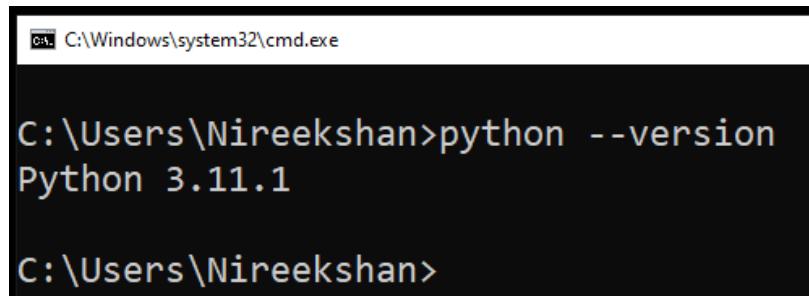


- ✓ Once we check the checkbox then we need to click on **Install Now**.
- ✓ Finally we will get, **Setup was successful** screen



4. Check the python version

- ✓ Open a **NEW** command prompt in your laptop/computer.
- ✓ Type **python --version** command in command prompt and check it.



```
C:\Windows\system32\cmd.exe
C:\Users\Nireekshan>python --version
Python 3.11.1
C:\Users\Nireekshan>
```

1. DATA SCIENCE FUNDAMENTALS

Table of Contents

1. Data Introduction	2
2. What is Data?	2
3. Information.....	2
4. Data Processing	2
5. Data, Information Examples	3
5.1 Example.....	3
5.2 Example.....	3
6. What is Data Science?	4
6.1 Definition 1.....	4
6.2 Definition 2.....	4
6.3 Scientist.....	4
6.4 Nature of Scientist	4
6.5 Process behind the scientist	4
7. BigData.....	6
8. Data Measurement table	6
9. Types of BigData.....	7
9.1 Structured	7
9.2 Semi structured.....	7
9.3 Unstructured data.....	7
10. V specialty in BigData.....	9
10.1 Volume	9
10.2 Variety	9
10.3 Velocity	9
11. Data Analytics.....	10
12. Different types of Analytics	10
12.1 Descriptive Analytics.....	10
12.2 Diagnostic Analytics	10
12.3 Predictive Analytics.....	10
12.4 Prescriptive Analytics	11

1. DATA SCIENCE FUNDAMENTALS

1. Data Introduction

- ✓ Currently we are living in the data world.
- ✓ Everyone is communicating by using devices and social networks, due to this huge amount of data is generating.
- ✓ All applications are generating data.
 - Ecommerce applications
 - Banking applications
 - Social network etc.

2. What is Data?

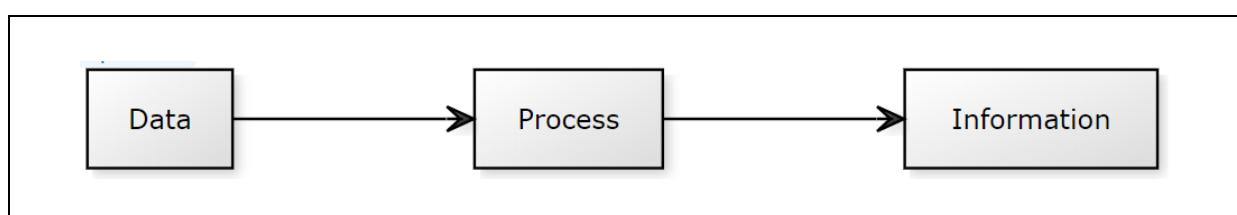
- ✓ Data is a collection of Facts.
- ✓ Facts,
 - Numbers
 - Alphabets
 - Alphanumeric
 - Symbols
 - Images
 - Audio
 - Video & etc

3. Information

- ✓ Data we may not understand clearly.
- ✓ The processed data is called as information.

4. Data Processing

- ✓ Converting the raw data into meaningful information

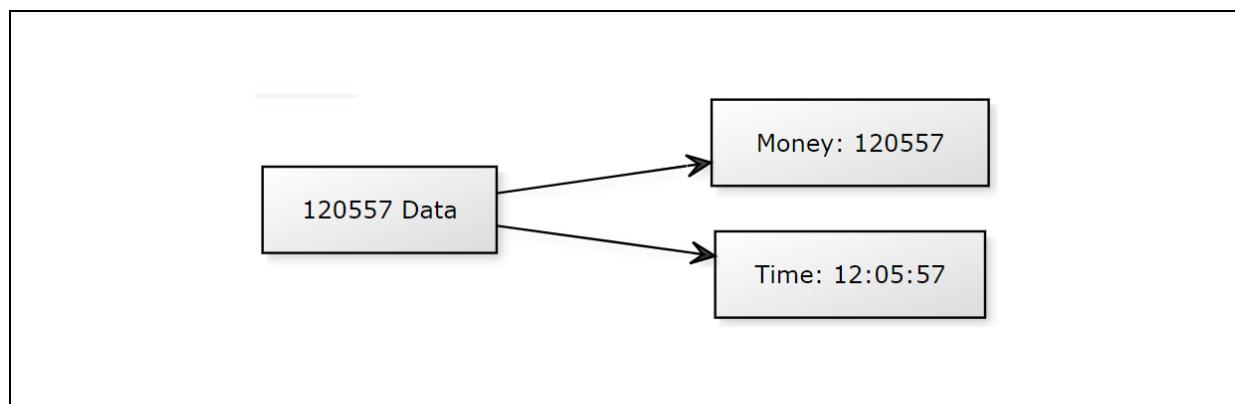


5. Data, Information Examples

5.1 Example

Data	Information
Daniel, Data Scientist, age is sweet sixteen, Bengaluru, India	Name : Daniel Profession : Data Scientist Age : 16 Location : Bengaluru Country : India

5.2 Example



6. What is Data Science?

6.1 Definition 1

- ✓ Data science is a combination of scientific methods, algorithms to extract knowledge and insights from data.

6.2 Definition 2

- ✓ Data science is all about how we take data,
 - Process data
 - Understand the past/present conditions
 - Decision making process
 - Predicting the future results
 - Create new industries/products condition

6.3 Scientist

- ✓ Scientist is a person who is having expert knowledge on specific scenarios.

6.4 Nature of Scientist

- ✓ A scientist used to conduct more research to achieve results.

6.5 Process behind the scientist

- ✓ In simple words a scientist is someone who,
 - Systematically gathers information
 - Doing research
 - Getting evidence
 - Test the evidence
 - Gaining knowledge
 - Understanding the knowledge.
 - Sharing knowledge



7. BigData

- ✓ Extremely large data sets is called as a BigData.

8. Data Measurement table

Data Measurement Chart		
(Types of Various Units of Memory)		
Bit	----	Single Binary Digit (1 or 0)
Byte	----	8 Bits
Kilo Byte	(KB)	1,024 Bytes
Mega Byte	(MB)	1,024 Kilobytes
Giga Byte	(GB)	1,024 Megabytes
Tera Byte	(TB)	1,024 Gigabytes
Peta Byte	(PB)	1,024 Terabytes
Exa Byte	(EB)	1,024 Petabytes
Zetta Byte	(ZB)	1,024 Exabytes
Yotta Byte	(YB)	1,024 Zettabyte

9. Types of BigData

- ✓ There are mainly 3 types of big data.
 - Structured
 - Semi-structured
 - Unstructured

9.1 Structured

- ✓ Structured data will be stored in table formats like rows and columns.
- ✓ It is very easy to understand.
- ✓ Examples like, a table in database, excel file, csv file.

9.2 Semi structured

- ✓ Semi structured may not in proper format
- ✓ This type of data having some properties or tags.
- ✓ Example
 - Xml file
 - JSON file

9.3 Unstructured data

- ✓ Unstructured data have no proper format.
- ✓ In this world mostly unstructured data exists.
- ✓ Example
 - log file
 - audio, video
 - e-mail & etc

Data Science - Fundamentals

Structured data				Semi-structured data	Unstructured data
1	John	18	B.Sc.	<pre><University> <Student ID="1"> <Name>John</Name> <Age>18</Age> <Degree>B.Sc.</Degree> </Student> <Student ID="2"> <Name>David</Name> <Age>31</Age> <Degree>Ph.D. </Degree> </Student> </University></pre>	The university has 5600 students. John's ID is number 1, he is 18 years old and already holds a B.Sc. degree. David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

10. V specialty in BigData

10.1 Volume

- ✓ Volume of the data means, size of the data.
- ✓ Based on size only we can measure as the data is in big data or not.

10.2 Variety

- ✓ Variety of data means, the generated data may exist in different formats like
- ✓ It can be,
 - Structured
 - Semi-structured
 - Unstructured.

10.3 Velocity

- ✓ Velocity of data means, speed of the data generating and processing from the source points.

In short

- Volume = Size
- Variety = Types
- Velocity = Speed

11. Data Analytics

- ✓ All companies are analysing big data to improve the business.
- ✓ From the result of analysis, companies used to reach customers to understand them clearly.

11.1 Every company's target

- ✓ Ultimate target of any company is, wanted to stand in best position, and should be competitive with other companies.

12. Different types of Analytics

- ✓ There are mainly four types of analytics,
 1. Descriptive Analytics
 2. Diagnostic Analytics
 3. Predictive Analytics
 4. Prescriptive Analytics

12.1 Descriptive Analytics

- ✓ Descriptive analytics helps to understand like **what is happening?**
- ✓ Descriptive analytics provide the information about happened situations.
- ✓ Examples
 - Amazon offered one product then how the customers are buying that specific product.

12.2 Diagnostic Analytics

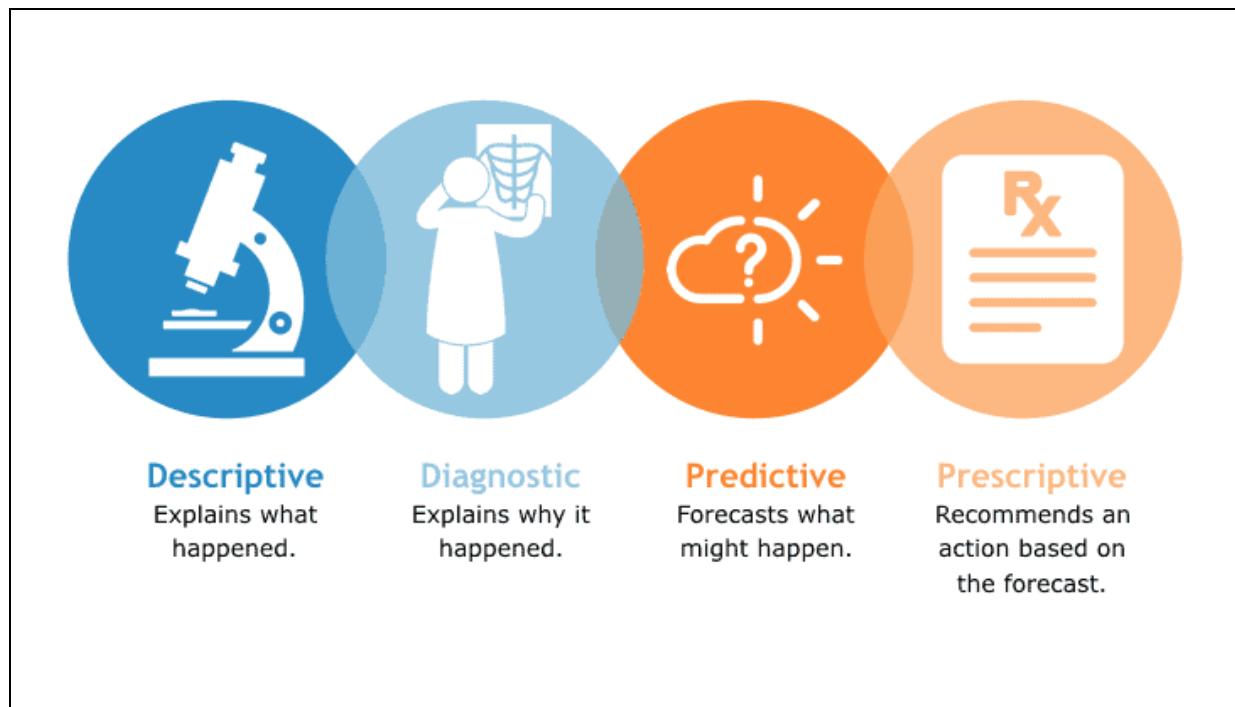
- ✓ Diagnostic analytics helps to understand like **why did it happen?**
- ✓ It is used to do a deeper understanding about the data to find out the cause of events and behaviours.

12.3 Predictive Analytics

- ✓ Predictive analytics helps to understand like **what will happen in future?**
- ✓ It is used to understand the future expectations, trends and provide the information about what will be happen in future.

12.4 Prescriptive Analytics

- ✓ Prescriptive analytics helps to understand like **what is the next best action?**
- ✓ It is used to apply the best decision actions for improvement.



1. PYTHON – INTRODUCTION

Table of Contents

1. What is Python?.....	2
2. Python programming applications	2
3. History of Python?	3
4. Why the name was python?.....	3
5. Python logo.....	4
6. Current python version	4
7. Python supports.....	4

1. PYTHON INTRODUCTION

1. What is Python?

- ✓ Python is a general purpose and high-level programming language.
- ✓ General purpose means, currently all companies are using python programming language to develop, test and deploy the software applications.
- ✓ High level programming means, it's a human readable language and easy to understand.

2. Python programming applications

- ✓ By using python programming we can develop,

Type of application	Purpose of the application
1. Standalone application	✓ An application which needs to install on every machine to work with that application.
2. Web applications	✓ An application which follows client-server architecture.
3. Database applications	✓ An application which perform curd (create, update, retrieve and delete) operations in database.
4. Bigdata applications	✓ An application which can process the BigData. (like pyspark)
5. Machine learning	✓ An application which enables computers to learn automatically from past data.

Make a note

- ✓ By using python we can even implement many of other applications too.

3. History of Python?

- ✓ Python was created by **Guido Van Rossum** in the year of 1991.
- ✓ Python is open source software means we can download freely from www.python.org website and customise the code as well.



Guido Van Rossum

4. Why the name was python?

- ✓ A TV show Monty Python's Flying Circus was very much popular fun show in 1970's.
- ✓ So, Guido likes this show and given this name to his programming language.

5. Python logo



6. Current python version

- ✓ While preparing this document the current python version is,
 - Python 3.11.1

7. Python supports

- ✓ Python supports Functional and Object oriented programming approach
 - Python = Functional Programming + Object Oriented Programming

2. PYTHON - KEYWORDS

Contents

1. What is a keyword in python?	2
2. How many keywords in PYTHON-3.11.1 version?.....	2
3. All keywords in one simple table	2

2. PYTHON KEYWORDS

1. What is a keyword in python?

- ✓ The word which is reserved to do a specific purpose, that word is called as keyword or reserved word.

2. How many keywords in PYTHON-3.11.1 version?

- ✓ There are total **35** keywords are available in python **3.11.1** version.
- ✓ All keywords list we can check from below website url
 - https://docs.python.org/3.11/reference/lexical_analysis.html#keywords

3. All keywords in one simple table

- ✓ All keywords we can see in below table.

S.No	1 Flow control	2 Variable	3 Function/Method	4 Object Oriented	5 Module	6 Logical	7 Checking	8 Deleting	9 Builtin constants	10 Exception handling	11 Asynchronous
1	if	global	def	class	import	and	is	del	True	try	async
2	else	nonlocal	with		from	or	in		False	except	await
3	elif		lambda		as	not			None	finally	
4	for		yield							raise	
5	while		return							assert	
6	break										
7	continue										
8	pass										
Total	8	2	5	1	3	3	2	1	3	5	2

Make a note:

- ✓ All keywords in Python contain only alphabets.
- ✓ These are in lowercase except 3 keywords,
 - True
 - False
 - None

3. PYTHON – HELLO WORLD PROGRAM

Table of Contents

1. Ways to write python program	2
2. Python program execution steps	2
3. Understanding first python program	2

3. PYTHON – HELLO WORLD PROGRAM

1. Ways to write python program

- ✓ We can write the python programs in many ways like,
 - By using any notepad, Notepad++, Edit plus, IDLE (Integrated Development Environment), jupyter, PyCharm IDE, Anaconda, Visual Studio, spyder, atom & etc.
- ✓ Initially the best practice is, using notepad to write python program.

2. Python program execution steps

- ✓ Open a new notepad and **write** python program.
- ✓ We can **save** this program with **.py** or **.python** extension.
- ✓ **Run** or execute the program.
- ✓ Finally, we will get **output**.

Program Name	A program to print Hello World
	demo.py

```
print("Hello World")
```

Run	py demo.py
------------	------------

Output	Hello World
---------------	-------------

3. Understanding first python program

- ✓ `print("Hello World")`,
 - `print()` is a predefined function in python.
 - The purpose of `print()` is to display the output.
 - "Hello World" it is a string in python
 - We will learn more about **function** and **string** in upcoming chapters

Data Science – Python Hello World Practice

Program Name Run and check the output
demo1.py

```
print("Hello World")
```

Output

Program Name Run and check the output
demo2.py

```
print("Hello World")
print("How are you doing")
```

Output

Program Name Run and check the output
demo3.py

```
print("Hello World")
print("How are you doing")
    print("This is Daniel")
```

Output

Data Science – Python Hello World Practice

Program Name Run and check the output
demo4.py

```
print()
```

Output

Program Name Run and check the output
demo5.py

```
print("Hello World")
```

Output

Program Name Run and check the output
demo6.py

```
print("This is 10 $ dollars")
```

Output

Data Science – Python Hello World Practice

Program Name Run and check the output
demo7.py

```
print(111)
```

Output

Program Name Run and check the output
demo8.py

```
print("Hello World")
      print("How are you doing")
print("This is Daniel")
```

Output

Program Name Run and check the output
demo9.py

```
Print("Hello World")
```

Output

Data Science – Python Hello World Practice

Program Name Run and check the output
demo10.py

```
PRINT("Hello World")
```

Output

Program Name Run and check the output
demo11.py

```
print("Hello World")
```

Output

Program Name Run and check the output
demo12.py

```
print("ello orld")
```

Output

Data Science – Python Hello World Practice

Program Name Run and check the output
demo13.py

```
print("Hello World")
```

Output

4. PYTHON – NAMING CONVENTIONS

Table of Contents

1. Identifier.....	2
2. Why should we follow naming conventions?	2
3. Points to follow for identifiers in Python.....	3
# Point 1	3
# Point 2	4
# Point 3	5
# Point 4	6
# Point 5	6
Examples.....	7
Common error.....	7
4. Python identifiers table.....	8
5. Comments in program.....	10
Purpose of comments.....	10
1. Singe line comments.....	10
2. Multi line comments.....	11

4. PYTHON - NAMING CONVENTIONS

1. Identifier

- ✓ A name in a python program is called **identifier**.
- ✓ This name can be,

- Package name
- Module name
- Variable name
- Function name
- Class name
- Method name

- ✓ Python creator made some suggestions to the programmers regarding how to write identifiers in a program.

2. Why should we follow naming conventions?

- ✓ While writing the program if we follow the naming conventions then the written code is,
 - Easy to understand.
 - Easy to read.
 - Easy to debug.

3. Points to follow for identifiers in Python

- ✓ We need to follow few points to define an identifiers,

Point 1

- ✓ While writing an identifier we can use,
 - Alphabets, either upper case or lower case
 - Numbers from 0 to 9
 - Underscore symbol (_)
- ✓ If we are using any other symbol then we will get syntax error.

Program Name Creating a valid identifier
demo1.py

```
student_id = 101
print(student_id)
```

Output
101

Program Name Creating an invalid identifier
demo2.py

```
$student_id = 101
print($student_id)
```

Error

SyntaxError: invalid syntax

Point 2

- ✓ We can write an identifier with number but identifier should not start with digit.

Program Name Creating a valid identifier
demo3.py

```
student_id123 = 101
print(student_id123)
```

Output
101

Program Name Creating an invalid identifier
demo4.py

```
123student_id = 101
print(123student_id)
```

Error
SyntaxError: invalid decimal literal

Point 3

- ✓ Identifiers are case sensitive.

Program Name Creating a valid identifier
demo5.py

```
value = 10
print(value)
```

Error
10

Program Name Identifier is a case sensitive
demo5.py

```
value = 10
print(VALUE)
```

Error
NameError: name 'VALUE' is not defined

Point 4

- ✓ We cannot use keywords as identifiers.

Program Name We should not use keywords to create an identifiers
demo6.py

```
if = 10
print(if)
```

Error **SyntaxError:** invalid syntax

Point 5

- ✓ Spaces are not allowed in between the identifier.

Program Name Spaces not allowed between identifier
demo7.py

```
student id = 101
print(student id)
```

Error **SyntaxError:** invalid syntax

Examples

- ✓ 435student # invalid
- ✓ student564 # valid
- ✓ student565info # valid
- ✓ \$student # invalid
- ✓ _student_info # valid
- ✓ class # invalid
- ✓ def # invalid

Common error

- ✓ **SyntaxError**: invalid syntax

4. Python identifiers table

- ✓ This table we can understand while studying upcoming topics.

Identifier	Conventions to follow for identifiers
1. class	<ul style="list-style-type: none"> ✓ In python, a class name should start with upper case and remaining letters are in lower case. ✓ If name having multiple words, then every nested word should start with upper case letter. <p style="margin-left: 40px;">○ Example: StudentInfo</p> <ul style="list-style-type: none"> ✓ Info: This rule is applicable for classes created by users only; the in-built class names used all are in lower-case.
2. package	
3. module	
4. variable	<ul style="list-style-type: none"> ✓ Names should be in lower case. ✓ If name having multiple words, then separating words with underscore (_) is good practice.
5. function	<p style="margin-left: 40px;">○ Example: student_id</p>
6. method	
7. Non-public instance variables	<ul style="list-style-type: none"> ✓ Non-public instance variables should begin with underscore (_), we can say private data. <p style="margin-left: 40px;">○ Example: _balance</p>
8. constants	<ul style="list-style-type: none"> ✓ Names should be in upper case. ✓ If name having multiple words, then separating words with underscore (_) is good practice.

Data Science – Python Naming Conventions

	<ul style="list-style-type: none">○ Example: IN_PROGRESS
9. Non-accessible entities	<ul style="list-style-type: none">✓ Few variables, class constructors (topic in object oriented programming) names having two underscores symbols starting and ending <ul style="list-style-type: none">○ Example: __init__(self)

5. Comments in program

- ✓ There are two types of comments

1. Single line comments
2. Multi line comments

Purpose of comments

- ✓ Comments are useful to describe about the code in an easy way.
- ✓ Python ignores comments while running the program.

1. Singe line comments

- ✓ By using single line comment, we can comment only a single line.
- ✓ To comment single line, we need to use hash symbol #

Program Name	A program with single line comment demo8.py
	#This is Basic program in python print("Welcome to python programming")
output	Welcome to python programming

2. Multi line comments

- ✓ By using multi line comment we can comment multiple lines.
- ✓ To comment multiple lines, we need to use triple double quotes symbol.

Program Name A program with multi line comments
demo9.py

```
"""Author Daniel  
Project Python project  
Location Bengaluru"""
```

```
print("Welcome to python programming")
```

output
Welcome to python programming

5. PYTHON - VARIABLES

Contents

1. Variable	2
2. Purpose of the variable	2
3. Properties of variable.....	2
4. Naming convention to a variable	2
5. Creating a variable	2
6. Creating multiple variables in single line	5
7. Single value for multiple variables.....	6
8. Variable re-initialization.....	7

5. PYTHON - VARIABLES

1. Variable

- ✓ Variable is a reserved memory location to store values.

2. Purpose of the variable

- ✓ The purpose of variable is to represent the **data**
- ✓ Data means a collection of **facts**
- ✓ Facts can be,
 - Alphabets.
 - Numbers.
 - Alphanumeric
 - Symbols

3. Properties of variable

- ✓ Every variable has its own properties,
- ✓ Every variable can contain,
 - Name
 - Type
 - Value

4. Naming convention to a variable

- ✓ Name should start with lower case.
- ✓ If name having multiple words then separating every word with underscore symbol is a good practice

5. Creating a variable

- ✓ We need to follow below syntax to create a variable.

Syntax

```
name_of_the_variable = value
```

Data Science – Python Variables

Program Name Creating a single variable
demo1.py

```
age = 16  
print(age)
```

output
16

Program Name Creating two variables
demo2.py

```
name = "Daniel"  
age = 16  
  
print(name)  
print(age)
```

output
Daniel
16

Program Name Creating two variables and using one print
demo3.py

```
name = " Daniel"  
age = 16  
  
print(name, age)
```

output
Daniel 16

Program Name Creating a variable with meaningful text message
demo4.py

```
name = "Daniel"  
  
print("I am", name)
```

output
I am Daniel

Important note

- ✓ Python having dynamic data type nature.
- ✓ It means, while creating variable we no need to provide the data type explicitly in python.
- ✓ Regarding data type we will learn in Data type chapter.

6. Creating multiple variables in single line

- ✓ We can creating multiple variables in single line
- ✓ # Rule
 - There should be same number on the left and right-hand sides.
Otherwise we will get error.

Program Name Creating multiple variables in single line
demo5.py

```
a, b, c = 1, 2, 3
```

```
print(a)  
print(b)  
print(c)
```

output

```
1  
2  
3
```

7. Single value for multiple variables

- ✓ We can assign a single value to multiple variables simultaneously.

Program Name Assign single value to multiple variables
demo6.py

```
a = b = c = 3
```

```
print(a)  
print(b)  
print(c)
```

Output

```
3  
3  
3
```

8. Variable re-initialization

- ✓ Based on requirement we can re-initialize existing variable value.
- ✓ In variable re-initialization old values will be replaced with new values.

Program Name	Creating a variable demo7.py
	<pre>sal = 10000 print("My salary is:", sal)</pre>
Output	My salary is: 10000

Program Name	Re-initialising variable demo8.py
	<pre>sal = 10000 sal = 12000 print("My salary is:", sal)</pre>
Output	My salary is: 12000

6. PYTHON – DATA TYPES

Table of Contents

1. What is a Data Type in python?.....	2
2. type(p) function.....	3
3. Different types of data types	4
3.1 Built-in data types:	4
1. Numeric types.....	5
2. bool data type (boolean data type).....	7
3. None data type	8
4. Sequences in Python	9
4.1 string data type	10
4.2 list data structure	11
4.3 tuple data structure	11
4.4 set data structure	12
4.5 dictionary data structure	12
4. 6 range data type	13
3.2 User defined data types	15

6. PYTHON – DATA TYPES

1. What is a Data Type in python?

- ✓ A data type represents the type of the data stored into a variable or memory.

Program Name print different kinds of variables
demo1.py

```
emp_id = 1  
name = "Daniel"  
salary = 10000.56
```

```
print("My employee id is: ", emp_id)  
print("My name is: ", name)  
print("My salary is: ", salary)
```

Output

```
My employee id is: 1  
My name is: Daniel  
My salary is: 10000.56
```

Note

- ✓ By using type(p) function we can check the data type of each variable.

2. type(p) function

- ✓ type(p) is predefined function in python.
- ✓ By using this we can check the type of the variables.

Program Name Check the type of variables
demo2.py

```
emp_id = 1
name = "Daniel"
salary = 10000.56

print("My employee id is: ", emp_id)
print("My name is: ", name)
print("My salary is: ", salary)
print()
print("emp_id type is: ", type(emp_id))
print("name type is: ", type(name))
print("salary type is: ", type(salary))
```

Output

```
My employee id is: 1
My name is: Daniel
My salary is: 10000.56
```

```
emp_id type is: <class 'int'>
name type is: <class 'str'>
salary type is: <class 'float'>
```

3. Different types of data types

- ✓ There are two type of data types.
 1. Built-in data types
 2. User defined data types

3.1 Built-in data types:

- ✓ The data types which are already existing in python are called built-in data types.
 1. Numeric types
 - int
 - float
 2. bool (boolean type)
 3. None
 4. Sequence
 - str
 - list
 - tuple
 - set
 - dict
 - range

1. Numeric types

- ✓ The numeric types represent numbers, these are divided into three types,
 1. int
 2. float
 3. complex

1.1 int data type

- ✓ The int data type represents a number without decimal values.
- ✓ In python there is no limit for int data type.
- ✓ It can store very large values conveniently.

Program Name To print integer value
demo3.py

```
a = 20
print(a)
print(type(a))
```

output
20
<class 'int'>

Program Name int data type can store bigger values too
demo4.py

```
b = 99999999999999999999
print(b)
print(type(b))
```

output
99999999999999999999
<class 'int'>

1. 2. float data type

- ✓ The float data type represents a number with decimal values.

Program Name To print float value and data type
demo5.py

```
salary = 10000.56
print(salary)
print(type(salary))
```

Output

```
10000.56
<class 'float'>
```

2. bool data type (boolean data type)

- ✓ bool data type represents boolean values in python.
- ✓ bool data type having only two values those are,
 - **True**
 - **False**

Make a note

- ✓ Python internally represents,
 - **True** as 1
 - **False** as 0

Program Name boolean values
demo6.py

```
a = True
b = False
```

```
print(a)
print(b)
```

output

```
True
False
```

3. None data type

- ✓ **None** data type represents an object that does not contain any value.
- ✓ If any object having no value, then we can assign that object with **None** data type.

Program Name None data type
demo7.py

```
a = None
print(a)
print(type(a))
```

output
None
<class 'NoneType'>

Note

- ✓ A function and method can return **None** data type.
- ✓ This point we will understand more in functions and oops chapters.

4. Sequences in Python

- ✓ Sequence means an object.
- ✓ Sequence object can store a group of values,
 1. string
 2. list
 3. tuple
 4. set
 5. dict
 6. range

Make a note

- ✓ Regarding sequences like `string`, `list`, `tuple`, `set` and `dict` we will discuss in upcoming chapters
 - Python String chapter
 - Python List Data Structure chapter
 - Python Tuple Data Structure chapter
 - Python Set Data Structure chapter
 - Python Dictionary Data Structure chapter

4.1 string data type

- ✓ A group of characters enclosed within single quotes or double quotes or triple quotes is called as string.

Program Name	Creating a string demo8.py
	<pre>name1 = 'Daniel' name2 = "Daniel" name3 = '''Daniel''' name4 = """"Daniel""" print(name1) print(name2) print(name3) print(name4) print(type(name1)) print(type(name2)) print(type(name3)) print(type(name4))</pre>

Output

	Daniel
	Daniel
	Daniel
	Daniel
	<class 'str'>

4.2 list data structure

- ✓ We can create list data structure by using square brackets []
- ✓ list can store a group of values.

Program Name Creating a list data structure
demo9.py

```
values = [10, 20, 30, 40]
print(values)
print(type(values))
```

Output

```
[10, 20, 30, 40]
<class 'list'>
```

4.3 tuple data structure

- ✓ We can create tuple data structure by using parenthesis symbol ()
- ✓ tuple can store a group of values.

Program Name Creating a tuple data structure
demo10.py

```
values = (10, 20, 30, 40)
print(values)
print(type(values))
```

Output

```
(10, 20, 30, 40)
<class 'tuple'>
```

4.4 set data structure

- ✓ We can create set data structure by using curly braces {}
- ✓ set can store a group of values.

Program Name Creating a set data structure
demo11.py

```
values = {10, 20, 30, 40}  
print(values)  
print(type(values))
```

Output

```
{40, 10, 20, 30}  
<class 'set'>
```

4.5 dictionary data structure

- ✓ We can create dictionary data structure by using curly braces {}
- ✓ Dictionary can store a group of values in the form of key value pair.

Program Name Creating a dictionary data structure
demo12.py

```
details = {1: "Jeswanth", 2: "Kumari", 3: "Prasad", 4: "Daniel"}  
print(details)  
print(type(details))
```

Output

```
{1: 'Jeswanth', 2: 'Kumari', 3: 'Prasad', 4: 'Daniel'}  
<class 'dict'>
```

4. 6 range data type

- ✓ range is a data type in python.
- ✓ Generally, range means a group of values from starting to ending.

Creating range of values

- ✓ We can create range of values by using range(p) predefined function

1. range(p) function

- ✓ As discussed, we can create range of values by using range(p) function.
- ✓ Here p should be integer, otherwise we will get error.

Program Name creating a range of values 0 to 4
demo13.py

```
a = range(5)
print(a)
print(type(a))
```

Output
range(0, 5)
<class 'range'>

Note:

- ✓ If we provide range(5), then range object holds the values from 0 to 4

2. range(start, end) function

- ✓ As discussed we can create range of values by using range(start, end) function.
- ✓ Here start means starting value and end means till to end-1 value

Program Name creating a range of values 1 to 9
demo14.py

```
a = range(1, 10)  
print(a)
```

Output
range(1, 10)

Note:

- ✓ If we provide range(1, 10), then range object holds the values from 1 to 9

Accessing range values by using for loop

- ✓ We can access range values by using for loop.

Program Name Access elements from range data type
demo15.py

```
r = range(10, 15)
```

```
for value in r:  
    print(value)
```

output

```
10  
11  
12  
13  
14
```

3.2 User defined data types

- ✓ Data types which are created by programmer.
- ✓ The datatype which are created by the programmers are called ‘user-defined’ data types, example is class, module, array etc.
- ✓ We will discuss about in OOPS chapter.

Question 1 Is Python case sensitive when dealing with identifiers?

Answer Yes

Question 2 What is the maximum possible length of an identifier?

Answer There is no length limit for identifier

Program Printing name, id, salary

Name demo1.py

```
name = "Daniel"  
emp_id = 11  
emp_salary = 1000.123  
  
print(name)  
print(emp_id)  
print(emp_salary)
```

Output

Program Name Printing name, id, salary with meaningful info
demo2.py

```
name = "Daniel"  
emp_id = 11  
emp_salary = 1000.123  
  
print("Emp name is: ",name)  
print("Emp id is: ",emp_id)  
print("Emp salary is: ",emp_salary)
```

Output

Program Name below variable names are valid or invalid
demo3.py

```
a=1  
_a = 2      # single underscore  
__a = 3     # two underscore  
__a__ = 4   # two underscores starting and ending of the  
variable  
  
print(a)  
print(_a)  
print(__a)  
print(__a__)
```

Output

Program Name below variable names are valid or invalid
demo4.py

```
name = "Daniel"  
name_1='Abhishek'  
_ = 'Ramesh'  
  
print(name)  
print(name_1)  
print(_)
```

Output

Program Name below variable names are valid or invalid
demo5.py

```
name = "Daniel"  
name_1 = "Naresh"  
1_name = "Srihari"  
_ = "Ramesh"  
  
print(name)  
print(name_1)  
print(1_name)  
print(_)
```

Output

Program Name below variable names are valid or invalid
demo6.py

```
name = "Daniel"  
name_1 = "Abhishek"  
$_name = "Srihari"  
_ = "Ramesh"  
  
print(name)  
print(name_1)  
print(1_name)  
print(_)
```

Output

Program Name below variable name is valid or invalid
demo7.py

```
abc = 1,000,000  
print(abc)
```

Output

Data Science – Python Variables Practice

Program Name below variable name is valid or invalid
demo8.py

```
abc = 1,000,000
print(abc)
print(type(abc))
```

Output

Program Name below variable names are valid or invalid
demo9.py

```
a b c = 1000 2000 3000
print(a b c)
```

Output

Program Name below variable names are valid or invalid
demo10.py

```
a b c = 1000 2000 3000
print(a b c)
print(type(a b c))
```

Output

Program Name below variable names are valid or invalid
demo11.py

```
a, b, c = 1000, 2000, 3000
print(a, b, c)
```

Output

Program Name below variable names are valid or invalid
demo12.py

```
a, b, c = 1000, 2000, 3000
print(a, b, c)
print(type(a, b, c))
```

Output

Program Name below variable names are valid or invalid
demo13.py

```
a, b, c = 1000, 2000, 3000
print(a, b, c)
```

```
print(type(a))
print(type(b))
print(type(c))
```

Output

Program Name below variable name is valid or invalid
demo14.py

```
a_b_c = 1,2,3
print(a_b_c)
```

Output

Data Science – Python Variables Practice

Program below variable name is valid or invalid

Name demo15.py

```
a_b_c = 1,2,3  
print(a_b_c)  
  
print(type(a_b_c))
```

Output

Program below program valid or invalid

Name demo16.py

```
10  
print("hello")
```

Output

Program below program valid or invalid

Name demo17.py

```
a  
print("hello")
```

Output

Invalid cases for variables

1. While creating a variable,

- Variable name should be written in left hand side.
- Value should be written in right hand side
- Otherwise we will get syntax error.

Program Name Creating variable in wrong direction
demo18.py

```
10 = emp_id  
print(emp_id)
```

Error
SyntaxError: can't assign to literal

2. variables names,

- should not give as keywords names, otherwise we will get error

Program Name keywords we should not use as variable names
demo19.py

```
if = 10  
print(if)
```

Error
SyntaxError: invalid syntax

3. Variables name without value is invalid.

Program Variable without value is invalid.

Name demo20.py

```
a
```

```
print(a)
```

Error

NameError: name 'a' is not defined

7. PYTHON – OPERATORS

Table of Contents

1. Operator	2
2. Type of operators	2
1. Arithmetic Operators: (+, -, *, /, %, **, //).....	3
2. Assignment operator: (=, +=, -=, *=, /=, %=, **=, //=)	5
3. Unary minus operator: (-)	6
4. Relational operators (>, >=, <, <=, ==, !=).....	7
5. Logical operators (and, or, not)	9
6. Membership operators(in, not in)	11

7. PYTHON – OPERATORS

1. Operator

- ✓ An operator is a symbol that performs an operation.
- ✓ An operator acts on some variables, those variables are called as operands.
- ✓ If an operator acts on a single operand, then it is called as **unary** operator
- ✓ If an operator acts on 2 operands, then it is called as **binary** operator.
- ✓ If an operator acts on 3 operands, then it is called as **ternary** operator.

Program Name operator and operands
demo1.py

```
a = 10  
b = 20  
c = a + b  
print(c)
```

output
30

- ✓ Here a, b and c are called as operands.
- ✓ + symbol is called as operator.

2. Type of operators

✓ Arithmetic Operators:	+ , - , * , / , % , ** , //
✓ Assignment Operator	=
✓ Unary minus Operator	-
✓ Relational Operator	> , < , >= , <= , == , !=
✓ Logical Operators	and , or , not
✓ Membership operators	in , not in

Make a note:

- ✓ Python does not have **increment** and **decrement** operators.

1. Arithmetic Operators: (+, -, *, /, %, **, //)

- ✓ These operators will do their usual job, so please don't expect any surprises.

Assume that,

```
a = 13  
b = 5
```

Operator	Meaning	Example	Result
+	Addition	$a + b$	18
-	Subtraction	$a - b$	8
*	Multiplication	$a * b$	65
/	Division	a / b	2.6
%	Modulus (Remainder of division)	$a \% b$	3
**	Exponent operator (exponential power value)	$a ** b$	371293

//	Integer division (gives only integer quotient)	a // b	2

Make a note

- ✓ Division operator / always performs floating point arithmetic, so it returns float values.
- ✓ Floor division (//) can perform both floating point and integral as well,
 - If values are int type, then result is int type.
 - If at least one value is float type, then result is float type.

Program Name Arithmetic Operators
demo2.py

```
a = 13
b = 5
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a%b)
print(a**b)
print(a//b)
```

Output

```
18
8
65
2.6
3
371293
2
```

2. Assignment operator: (=, +=, -=, *=, /=, %=, **=, //=)

- ✓ By using these operators, we can assign values to variables.

Assume that,

a = 13

Operator	Example	Equals to	Result
=	a = 13	a = 13	13
+=	a += 6	a = a + 6	19
-=	a -= 6	a = a - 6	7

Program Name Assignment operator
demo3.py

```
a = 13
print(a)
```

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

Output

```
13
18
```

3. Unary minus operator: (-)

- ✓ Symbol of unary minus operator is -

Program Name Unary minus operator
demo4.py

```
a = 10
print(a)
print(-a)
```

Output

```
10
-10
```

4. Relational operators ($>$, \geq , $<$, \leq , $==$, $!=$)

- ✓ By using these operators, we can create **simple conditions**.
- ✓ These operators are used to **compare two values**.
- ✓ These operators' returns result as **Boolean type** either True or False.

Assume that,

$a = 13$

$b = 5$

Operator	Example	Result
$>$	$a > b$	True
\geq	$a \geq b$	True
$<$	$a < b$	False
\leq	$a \leq b$	False
$==$	$a == b$	False
$!=$	$a != b$	True

Program Name Relational operators
demo4.py

```
a = 13
b = 5

print(a > b)
print(a >= b)
print(a < b)
print(a <= b)
print(a == b)
print(a != b)
```

Output

```
True
True
False
False
False
True
```

5. Logical operators (and, or, not)

- ✓ In python there are three logical operators those are,
 - and
 - or
 - not
- ✓ Logical operators are useful to create **compound conditions**.
- ✓ Compound condition is a **combination** of more than one simple condition.
- ✓ Each simple condition brings the Boolean result finally the total compound condition evaluates either **True** or **False**.

For Boolean types behaviour

- ✓ and
 - If both arguments are True, then only result is True
- ✓ or
 - If at least one argument is True, then result is True
- ✓ not
 - complement

Program Name Logical operators.
demo5.py

```
a = True  
b = False  
  
print(a and a)  
print(a or b)  
print(not a)
```

Output

```
True  
True  
False
```

Program Name	Logical operators demo7.py
	<pre>a = 1 b = 2 c = 3 print((a>b) and (b>c)) print((a<b) and (b<c))</pre>
Output	False True

6. Membership operators (in, not in)

- ✓ There are two membership operators,
 - **in**
 - **not in**
- ✓ Membership operators are useful to check whether the given value is available in sequence or not. (It may be string, list, set, tuple and dict)

The **in** operator

- ✓ **in** operator returns **True**, if element is found in the collection or sequences.
- ✓ **in** operator returns **False**, if element is not found in the collection or sequences.

The **not in** operator

- ✓ **not in** operator returns **True**, if an element is not found in the sequence.
- ✓ **not in** operator returns **False**, if an element is found in the sequence.

Program Name in, not in operators
 demo8.py

values = [10, 20, 30]

```
print(20 in values)
print(22 in values)
```

output

True
False

Program Name in, not in operators
demo9.py

```
text = "Welcome to python programming"

print("Welcome" in text)
print("welcome" in text)
print("nireekshan" in text)
print("Daniel" not in text)
```

output

```
True
False
False
True
```

8. PYTHON – INPUT AND OUTPUT

Table of Contents

1. Input and output	2
2. Hard coding.....	2
3. input(p) function	3
4. Convert from string to int	5
5. Convert from string type to other type	7
6. Convert float data type value into int data type.....	9
7. List out few type conversion functions in python.....	12

8. Input and Output

1. Input and output

- ✓ Input represents data given to the program.
- ✓ Output represents the result of the program.

2. Hard coding

- ✓ Till now we have executed examples by hard coding values to variables
- ✓ In this chapter we will learn how to take values at run time.

Program Name	Hard coding the values demo1.py
	age = 16 print(age)
output	16

- ✓ Based on requirement we can take values at runtime or dynamically as well.

Please enter the age: 16 Entered value is: 16
--

3. **input(p)** function

- ✓ **input(p)** is a predefined function.
- ✓ This function accepts input from the keyboard.
- ✓ This function takes a value from keyboard and returns it as a string type.
- ✓ Based on requirement we can convert from string to other types.

Program Name Printing name by taking value at run time
demo2.py

```
name = input("Enter the name:")  
print("You entered name as:", name)
```

Run py demo2.py

Output
Enter the name: Daniel
You entered name as: Daniel

Program Name Printing name and age by taking value at run time
demo3.py

```
name = input("Enter the name: ")  
age = input("Enter the age: ")  
  
print("You entered name as: ", name)  
print("You entered age as: ", age)
```

Run py demo3.py

Output
Enter the name: Prasad
Enter the age: 16

```
You entered name as: Prasad  
You entered age as: 16
```

Program Name Checking return type value for `input()` function
`demo4.py`

```
value = input("Enter the value ")  
print("Entered value as: ", value)  
print("type is: ", type(value))
```

Run `py demo4.py`

Output
Enter the value: **Daniel**
Entered value as: **Daniel**
type is: <class 'str'>

Run `py demo4.py`

Output
Enter the value:**123**
Entered value as: **123**
type is: <class 'str'>

Run `py demo4.py`

Output
Enter the value:**123.456**
Entered value as: **123.456**
type is: <class 'str'>

4. Convert from string to int

- ✓ We can convert the string value into int value.
- ✓ int(p) is a predefined function
- ✓ This function converts to int data type.

Program Name Converting from string to int data type
demo5.py

```
a = "123"  
print(a)  
print(type(a))
```

```
b = int(a)  
print(b)  
print(type(b))
```

Output

```
123  
<class 'str'>  
123  
<class 'int'>
```

Program Name Converting from string to int data type
demo7.py

```
a = input("Enter a value:")
print("Your value is:", a)
print("data type is:", type(a))

b = int(a)
print("After converting the value is:", b)
print("data type is:", type(b))
```

Run py demo5.py

Output

```
Enter a value:123
Your value is: 123
data type is: <class 'str'>
After converting the value is: 123
data type is: <class 'int'>
```

5. Convert from string type to other type

- ✓ We can convert the string value into float value.
- ✓ float(p) is a predefined function
- ✓ This function converts to float data type.

Program Name Converting from string to float data type
demo8.py

```
a = "123.99"  
print(a)  
print(type(a))
```

```
b = float(a)  
print(b)  
print(type(b))
```

Output

```
123.99  
<class 'str'>  
123.99  
<class 'float'>
```

Program Name Converting from string to float data type
demo9.py

```
a = input("Enter a value:")
print("Your value is:", a)
print("data type is:", type(a))

b = float(a)
print("After converting the value is:", b)
print("data type is:", type(b))
```

Run py demo9.py

Output

```
Enter a value:123.99
Your value is: 123.99
data type is: <class 'str'>
After converting the value is: 123.99
data type is: <class 'float'>
```

6. Convert float data type value into int data type

- ✓ `int(p)` is predefined function in python.
- ✓ This function converts value into int data type

Program Name Converting from float to int data type
demo10.py

```
a = 10000.45
print(a)
print(type(a))
```

```
b = int(a)
print(b)
print(type(b))
```

Output

```
10000.45
<class 'float'>
10000
<class 'int'>
```

Program Name + operator concatenates/joins two string values
demo11.py

```
p1 = input("Enter First product price:")  
p2 = input("Enter Second product price:")
```

```
print("Total price is:", p1+p2)
```

Run py demo11.py

Output

```
Enter First product price:111  
Enter Second product price:222  
Total price is: 111222
```

Program Name Converting string to int and perform + operator
demo12.py

```
p1 = input("Enter First product price:")  
p2 = input("Enter Second product price:")  
  
a = int(p1)  
b = int(p2)  
  
print("Total price is:", a+b)
```

Run py demo12.py

Output

```
Enter First product price:111  
Enter Second product price:222  
Total price is: 333
```

7. List out few type conversion functions in python

1. int(p) – converts other data type into integer type
2. float(p) – converts other data type into float type
3. str(p) – converts other data type into a string.
4. bool(p) – converts other data type into boolean type
5. list(p) – converts sequence to list
6. tuple(p) – converts sequence to a tuple.
7. set(p) – converts sequence to set.
8. dict(p) – converts a tuple of order (key, value) into a dictionary.

9. PYTHON – FLOW CONTROL

Table of Contents

1. Flow control.....	2
2. Types of the execution	2
1. Sequential	2
2. Conditional.....	2
3. Looping.....	3
3. Sequential statements	3
4. Indentation.....	4
4.1 IndentationError	4
5. Conditional or Decision-making statements	5
5.1 if statement.....	5
5.2 if else statement	7
5.3 if elif else statement	9
6. Looping	11
6.1 for loop.....	11
6.2 while loop.....	12
7. break statement.....	14
8. continue statement.....	16
9. pass statement	17

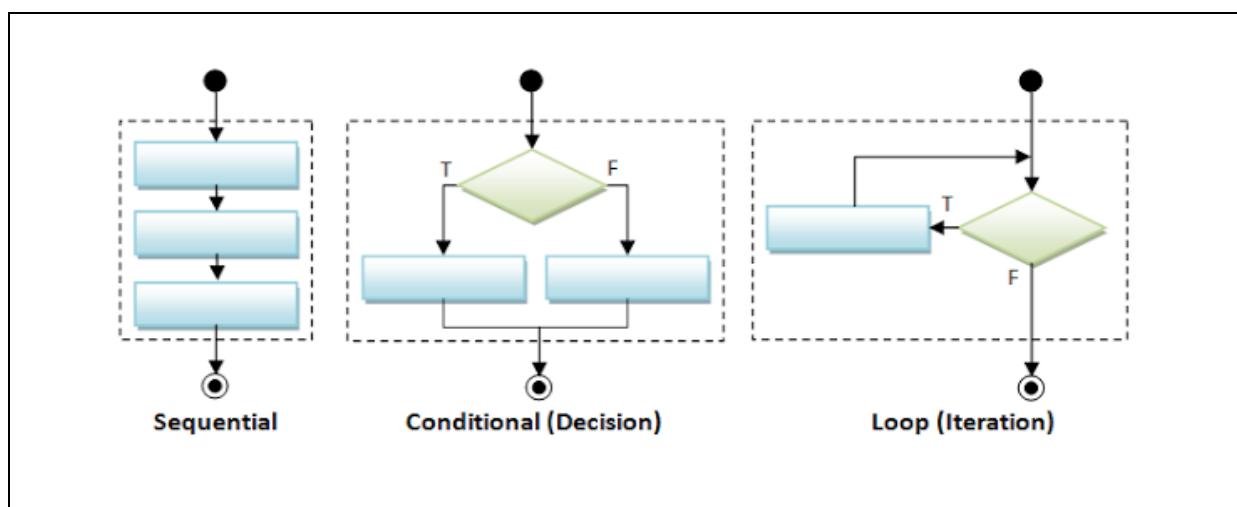
9. PYTHON – FLOW CONTROL

1. Flow control

- ✓ The order of statements execution is called as flow of control.

2. Types of the execution

- ✓ Based on requirement the programs statements can execute in different ways like sequentially, conditionally and repeatedly etc.
- ✓ In any programming language, statements will be executed mainly in three ways,
 - Sequential execution.
 - Conditional execution.
 - Looping execution.



1. Sequential

- ✓ Statements execute from top to bottom, means one by one sequentially.
- ✓ By using sequential statement, we can develop only simple programs.

2. Conditional

- ✓ Based on conditions, statements used to execute.
- ✓ Conditional statements are useful to develop better and complex programs.

3. Looping

- ✓ Based on conditions, statements used to execute randomly and repeatedly.
- ✓ Looping execution is useful to develop better and complex programs.

3. Sequential statements

Program Name sequential statement: executes from top to bottom
demo1.py

```
print("one")
print("two")
print("three")
```

output

```
one
two
three
```

2. Conditional or Decision-making statements

- | | |
|---|---|
| <ul style="list-style-type: none">○ if○ if else○ if elif else○ if elif elif else | <ul style="list-style-type: none">- valid combination- valid combination- valid combination- valid combination |
|---|---|

3. Looping

- ✓ for loop
- ✓ while loop

4. Other keywords

- ✓ break
- ✓ continue
- ✓ pass

4. Indentation

- ✓ Python uses indentation to indicate a block of code.
- ✓ Indentation refers to adding white space before a statement to a particular block of code.
- ✓ Python uses 4 spaces as indentation by default.
 - However, the number of spaces is up to you, but a minimum of 1 space has to be used.

4.1 IndentationError

- ✓ If we didn't follow the proper indentation then we will get error as **IndentationError**: expected an indented block

5. Conditional or Decision-making statements

5.1 if statement

syntax

```
if condition:  
    if block statements  
  
    out of if block statements
```

- ✓ **if** is a keyword in python
- ✓ **if** statement contains an expression/condition/value.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws syntax **error**.
- ✓ After **if** statement we need to follow indentation otherwise it throws **IndentationError**.
- ✓ Condition gives the result as a bool type, means either **True** or **False**.



- ✓ If the condition result is **True**, then **if** block statements will be executed
- ✓ If the condition result is **False**, then **if** block statements won't execute.

Program Name Executing if block statements by using **if** statement
demo2.py

```
x = 1
y = 1

print("x==y value is: ", (x==y))
if x == y:
    print("if block statements executed")
```

output
x==y value is: **True**
if block statements executed

Program Name Executing out of if block statements
demo3.py

```
x = 1
y = 2

print("x==y value is: ", (x==y))
if x == y:
    print("if block statements executed")
print("out of if block statements")
```

output
x==y value is: **False**
out of if block statements

5.2 if else statement

syntax

```
if condition:  
    if block statements1  
  
else:  
    else block statements2
```

- ✓ **if** and **else** are keywords in python
- ✓ **if** statement contains an expression/condition.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws syntax **error**.
- ✓ After **if** and **else** statements we need to follow indentation otherwise it throws **IndentationError**.
- ✓ Condition gives the result as bool type, means either **True** or **False**



- ✓ If the condition result is **True**, then **if** block statements will be executed
- ✓ If the condition result is **False**, then **else** block statements will be executed.

Program Name Executing if block statements by using **if else** statement
demo4.py

```
x = 1
y = 1

print("x==y value is: ", (x==y))

if x == y:
    print("if block statements executed")
else:
    print("else block statements executed")
```

output
x==y value is: **True**
if block statements executed

Program Name printing else block statements
demo5.py

```
x = 1
y = 2

print("x==y value is: ", (x == y))

if x == y:
    print("if block statements executed")
else:
    print("else block statements executed")
```

output
x==y value is: **False**
else block statements executed

5.3 if elif else statement

syntax

```
if condition1:  
    if block statements  
  
elif condition2:  
    elif block1statements  
  
elif condition3:  
    elif block2statements  
  
else:  
    else block statements
```

- ✓ **if, elif and else** are keywords in python
- ✓ **if** statement contains an expression/condition.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws error.
- ✓ After **if, elif** and **else** statements we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as bool type, means either **True** or **False**



- ✓ If the condition result is **True**, then any matched **if** or **elif** block statements will execute.
- ✓ If all **if** and **elif** conditions results are **False**, then **else** block statements will execute.

Make a note

- ✓ Here, else part is an optional

Program Name printing corresponding value by using if, elif, else statements
demo6.py

```
print("Please enter the values from 0 to 4")
x = int(input("Enter a number: "))

if x == 0:
    print("You entered:", x)

elif x == 1:
    print("You entered:", x)

elif x == 2:
    print("You entered:", x)

elif x == 3:
    print("You entered:", x)

elif x == 4:
    print("You entered:", x)

else:
    print("Beyond the range than specified")
```

output

```
Enter a number: 1
You entered: 1

Enter a number: 100
Beyond the range than specified
```

6. Looping

- ✓ If we want to execute a group of statements in multiple times, then we should go for looping kind of execution.
 - for loop
 - while loop

6.1 for loop

- ✓ **for** is a keyword in python
- ✓ Basically, **for** loop is used to get or iterate elements one by one from sequence like string, list, tuple, etc...
- ✓ While iterating elements from sequence we can perform operations on every element.

Syntax

```
for variable in sequence:  
    statements
```

Program Name Using for loop printing value from list
demo7.py

```
values = [10, 20, 30, "Daniel"]  
for value in values:  
    print(value)
```

output

```
10  
20  
30  
Daniel
```

6.2 while loop

- ✓ **while** is a keyword in python
- ✓ If we want to execute a group of statements repeatedly until the condition reaches to **False**, then we should go for **while** loop.

Syntax

Initialization
while condition:
 while block statements
 increment/decrement

- ✓ **while** loop contains an expression/condition.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws syntax error.
- ✓ After **while** loop we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as bool type, means either **True** or **False**



- ✓ If the condition result is **True**, then while loop executes till the condition reaches to **False**.
- ✓ If the condition result is **False**, then **while** loop execution terminates.

Conclusion

- ✓ Till condition is **True** the **while** loop statements will be executed.
- ✓ If the condition reaches to **False**, then **while** loop terminate the execution.

Program Name Printing numbers from 1 to 5 by using while loop
demo8.py

```
x = 1
while x <= 5:
    print(x)
    x = x+1

print("End")
```

output

```
1
2
3
4
5
End
```

7. break statement

- ✓ **break** is a keyword in python
- ✓ The **break** statement can be used inside the loops.
- ✓ By using break we can break the execution based on some condition.
- ✓ Generally, **break** statement is used to terminate **for** and **while** loops.

Program Name while loop without break
demo9.py

```
x = 1
while x<=10:
    print("x=", x)
    x = x + 1

print("out of loop")
```

output

```
x= 1
x= 2
x= 3
x= 4
x= 5
x= 6
x= 7
x= 8
x= 9
x= 10
out of loop
```

Program Name printing just 1 to 5 by using while loop and break
demo10.py

```
x = 1
while x <= 10:
    print("x=", x)
    x = x+1
    if x == 5:
        break

print("out of the loop")
```

output

```
x= 1
x= 2
x= 3
x= 4
out of the loop
```

Program Name break without loop
demo11.py

```
x = 1
if x <= 10:
    print(x)
    break
```

output

```
SyntaxError: 'break' outside loop
```

8. continue statement

- ✓ **continue** is a keyword in python
- ✓ We can use **continue** statement to skip current iteration and continue next iteration.

Program Name continue statement
demo12.py

```
cart = [10, 20, 500, 700, 50, 60]
for item in cart:
    if item == 500:
        continue
    print(item)
```

output

```
10
20
700
50
60
```

Program Name continue without loop
demo13.py

```
x = 1
if x <= 10:
    continue

print(x)
```

output

SyntaxError: 'continue' not properly in loop

9. pass statement

- ✓ **pass** is keyword in python.
- ✓ The pass statement is used as a placeholder for future code.
- ✓ It is useful as a placeholder when a statement is required syntactically, but no code needs to be executed.
- ✓ pass is a null operation, when it is executed, nothing happens.
- ✓ We can define an empty function, class, method with pass statement.

Program Name Function without pass statement
demo14.py

```
def upcoming_sales():

    upcoming_sales()
```

output

IndentationError: expected an indented block

Program Name Function with pass statement
demo15.py

```
def upcoming_sales():
    pass

    upcoming_sales()
```

output

Make a note

- ✓ We can even define a method or block of code with pass statement.

10. Python - String

Table of Contents

1. Gentle reminder	2
2. What is a string?	2
2.1. Definition 1.....	2
2.2. Definition 2.....	2
2.3. String is more popular.....	2
3. Creating string	3
3.1. When should we go for triple single and triple double quotes?	5
4. Mutable	8
5. Immutable	8
6. Strings are immutable.....	8
7. Mathematical operators on string objects.....	10
7.1. Addition (+) operator with string.....	10
7.2. Multiplication (*) operator with string	10
8. Length of the string	11
9. Membership operators (in, not in)	12
9.1. in operator	12
9.2. not in operator.....	13
10. Methods in str class.....	14
11.1. upper() method.....	17
11.2. lower() method	18
11.3. strip() method	19
11.4. count(p) method	20
11.5. replace(p1, p2) method	21
11.6. split(p) method	23

10. Python - String

1. Gentle reminder

- ✓ We already learnt first Hello world program in python.
- ✓ In that program we just print a group of characters by using print(p) function.
- ✓ That group of characters are called as a string.

Program Name	printing Welcome to python programming demo1.py
Output	print("Hello world") Hello world

2. What is a string?

2.1. Definition 1

- ✓ A group of characters enclosed within single or double or triple quotes is called as string.

2.2. Definition 2

- ✓ We can say string is a sequential collection of characters.

2.3. String is more popular

- ✓ In any kind of programming language, mostly usage data type is string.

3. Creating string

Syntax 1 With single quotes

```
name1 = 'Daniel'
```

Syntax 2 With double quotes

```
name2 = "Daniel"
```

Syntax 3 With triple single quotes

```
name3 = '''Daniel'''
```

Syntax 4 With triple double quotes

```
name4 = """Daniel"""
```

<p>Program Name</p> <p>Creating string by using all possibilities demo2.py</p> <pre>name1= 'Daniel' name2 = "Prasad" name3 = '''Mouli''' name4 = """Veeru"""\n print(name1, "name is created by using single quotes") print(name2, "name is created by using double quotes") print(name3, "name is created by using triple single quotes") print(name4, "name is created by using triple double quotes")</pre> <p>output</p> <p>Daniel name is created by using single quotes Prasad name is created by using double quotes Mouli name is created by using triple single quotes Veeru name is created by using triple double quotes</p>

Make a note

- ✓ Generally, to create a string mostly used syntax is double quotes syntax.

3.1. When should we go for triple single and triple double quotes?

- ✓ If you want to create multiple lines of string, then triple single or triple double quotes are the best to use.

Program Name printing Employee information
demo3.py

```
loc1 = '''TCS company
          White Field
          Bangalore'''
```

```
loc2 = """TCS company
          Bangalore
          ITPL tech park"""
```

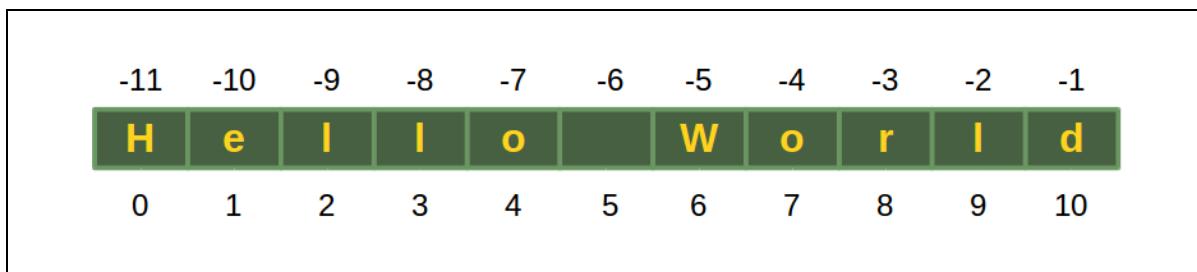
```
print(loc1)
print(loc2)
```

output

```
TCS company
White Field
Bangalore
```

```
TCS company
Bangalore
ITPL tech park
```

Diagram representation



Program Name Accessing string by using index
demo4.py

```
wish = "Hello World"
```

```
print(wish[0])  
print(wish[1])
```

Output

```
H  
e
```

Program Name Accessing string by using slicing
demo5.py

```
wish = "Hello World"
```

```
print(wish[0:7])
```

Output

```
Hello W
```

Program Name Accessing string by using for loop
demo6.py

```
wish = "Hello World"
```

```
for char in wish:  
    print(char)
```

Output

```
H  
e  
l  
l  
o
```

```
W  
o  
r  
l  
d
```

4. Mutable

- ✓ Once if we create an object then the state of existing object can be change/modify/update.
- ✓ This behaviour is called as mutability.

5. Immutable

- ✓ Once if we create an object then the state of existing object cannot be change/modify/update.
- ✓ This behaviour is called as immutability.

6. Strings are immutable

- ✓ String having immutable nature.
- ✓ Once we create a string object then we cannot change or modify the existing object.

Program Name Printing name and first index in string
demo7.py

```
name = "Daniel"  
print(name)  
print(name[0])
```

output

```
Daniel  
D
```

Program Name string having immutable nature
demo8.py

```
name = "Daniel"
```

```
print(name)
print(name[0])
name[0] = "X"
```

output

```
Daniel
```

```
D
```

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

7. Mathematical operators on string objects

- ✓ We can perform two mathematical operators on string.
- ✓ Those operators are,
 - Addition (+) operator.
 - Multiplication (*) operator.

7.1. Addition (+) operator with string

- ✓ The + operator works like concatenation or joins the strings.

Program Name	+ works as concatenation operator demo9.py
	a = "Python" b = "Programming" print(a+b)
output	PythonProgramming

7.2. Multiplication (*) operator with string

- ✓ This operator works with string to do repetition.

Program Name	* operator works as repetition in strings demo10.py
	course="Python" print(course*3)
output	PythonPythonPython

8. Length of the string

- ✓ We can find number of characters in string by using len() function

Program Name length of the string
demo11.py

```
course = "Python"  
print(len(course))
```

Output

```
6
```

9. Membership operators (in, not in)

Definition 1

- ✓ We can check, if a string or character is a member of string or not by using **in** and **not in** operators

Definition 2

- ✓ We can check, if string is a substring of main string or not by using **in** and **not in** operators.

9.1. in operator

- ✓ **in** operator returns **True**, if the string or character found in the main string.

Program
Name

in operator
demo12.py

```
print('p' in 'python')
print('z' in 'python')
print('on' in 'python')
print('pa' in 'python')
```

output

```
True
False
True
False
```

9.2. not in operator

- ✓ The **not in** operator returns opposite result of **in** operator.
- ✓ **not in** operator returns True, if the string or character not found in the main string.

Program not in operator

Name demo13.py

```
print('b' not in 'apple')
```

output

```
True
```

10. Methods in str class

- ✓ As discussed, str is a predefined class.
- ✓ So, str class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(parameter1)` predefined function.
- ✓ So, internally str class contains two types of methods,
 - With underscore symbol methods.
 - We no need to focus
 - Without underscore symbol methods.
 - We need to focus much on these

Program Name Printing str class methods by using `dir(str)` function
`demo14.py`

```
print(dir(str))
```

output

[

```
__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize',
```

Important methods

'count', 'upper', 'lower', 'replace', 'split', 'strip',

]

Important point

- ✓ As per object-oriented principle,
 - If we want to access instance methods, then we should access by using object name.
- ✓ So, all str class methods we can access by using str object

Important methods in str class

- ✓ upper()
- ✓ lower()
- ✓ strip()
- ✓ count(p)
- ✓ replace(p1, p2)
- ✓ split(p)

11.1. upper() method

- ✓ upper() is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ This method converts lower case letters into upper case letters

Program Name Converting from lowercase to uppercase
demo15.py

```
name = "daniel"  
print("Before converting: ", name)  
print("After converting: ", name.upper())
```

Output
Before converting: daniel
After converting: DANIEL

11.2. lower() method

- ✓ lower() is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ This method converts upper case letters into lower case letters

Program Name Converting from uppercase to lowercase
demo16.py

```
name = "DANIEL"
print("Before converting: ", name)
print("After converting: ", name.lower())
```

Output
Before converting: DANIEL
After converting: daniel

11.3. strip() method

- ✓ strip() is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ This method removes left and right side spaces of string

Make a note

- ✓ This method will not remove the spaces which are available middle of string object.

Program removing spaces in starting and ending of the string

Name demo17.py

```
course = "Python      "
print("with spaces course length is: ",len(course))
x = course.strip()
print("after removing spaces, course length is: ",len(x))
```

Output

```
with spaces course length is: 18
after removing spaces, course length is: 6
```

11.4. count(p) method

- ✓ count(p) is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ By using count() method we can find the number of occurrences of substring present in the string

Program Name counting sub string by using count() method
demo18.py

```
s="Python programming language, Python is easy"  
print(s.count("Python"))  
print(s.count("Hello"))
```

output

```
2  
0
```

11.5. replace(p1, p2) method

- ✓ replace(p1, p2) is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ We can replace old string with new string by using replace(p1, p2) method.

Program Name replacing string by using replace method
demo19.py

```
s1 = "Java programming language"  
s2 = s1.replace("Java", "Python")  
  
print(s1)  
print(s2)
```

output
Java programming language
Python programming language

Replace method returns new string object

- ✓ As we know string is immutable, so replace(p1, p2) method never perform changes on the existing string object.
- ✓ replace(p1, p2) method creates new string object, we can check this by using id(p) function

Program Name replacing string by using replace(p1, p2) method
demo20.py

```
s1 = "Java programming language"  
s2 = s1.replace("Java", "Python")
```

```
print(s1)  
print(s2)
```

```
print(id(s1))  
print(id(s2))
```

output

```
Java programming language  
Python programming language  
49044256  
48674600
```

String objects are immutable, Is replace(p1, p2) method will modify the string objects?

- ✓ Once we create a string object, we cannot change or modify the existing string object.
- ✓ This behaviour is called as immutability.
- ✓ If we are trying to change or modify the existing string object, then with those changes a new string object will be created.
- ✓ So, replace(p1, p2) method will create new string object with the modifications.

Splitting of Strings:

11.6. split(p) method

- ✓ split(p) is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ The default separator is space.
- ✓ We can split the given string according to specified separator by using split(p) method.
- ✓ split(p) method returns list.

Program splitting string by using split() method

Name demo21.py

```
s = "Python programming language"  
n = s.split()
```

```
print("Before splitting:", s)  
print("After splitting: ", n)
```

output

```
Before splitting: Python programming language  
After splitting: ['Python', 'programming', 'language']
```

Program Name splitting string by using split(p) method
demo22.py

```
s = "This is, Python programming, language "
n = s.split(",")
```

```
print("Before splitting:", s)
print("After splitting: ", n)
```

output

```
Before splitting: This is, Python programming, language
After splitting: ['This is', ' Python programming', ' language']
```

Program Name Please execute the program and check the output
demo1.py

```
a = 33  
b = 200
```

```
if b > a:  
    print("b is greater than a")
```

Output

Program Name Please execute the program and check the output
demo2.py

```
a = 33  
b = 200
```

```
if b > a:  
    print("b is greater than a")
```

Output

Program Name Please execute the program and check the output
demo3.py

a = 33
b = 33

```
if b > a:  
    print("b is greater than a")  
  
elif a == b:  
    print("a and b are equal")
```

Output

Program Name Please execute the program and check the output
demo4.py

a = 200
b = 33

```
if b > a:  
    print("b is greater than a")  
  
elif a == b:  
    print("a and b are equal")  
  
else:  
    print("a is greater than b")
```

Output

Program Name Please execute the program and check the output
demo5.py

```
a = 200  
b = 33
```

```
if b > a:  
    print("b is greater than a")  
  
else:  
    print("b is not greater than a")
```

Output

Program Name Please execute the program and check the output
demo6.py

```
a = 200  
b = 33
```

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

Output

Program Name Please execute the program and check the output
demo7.py

```
a = 200  
b = 33
```

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

Output

Program Name Please execute the program and check the output
demo8.py

```
flag = True
```

```
if flag==True:  
    print("Welcome")  
    print("To")  
    print("Python programming language" )
```

Output

Program Name Please execute the program and check the output
demo9.py

flag = **False**

```
if flag==True:  
    print("Welcome")  
    print("To")  
    print("Python programming language" )
```

Output

Program Name Please execute the program and check the output
demo10.py

i = 20;

```
if (i < 15):  
    print ("i is smaller than 15")  
    print ("i'm in if Block")
```

```
else:  
    print ("i is greater than to 15")  
    print ("i'm in else Block")
```

```
print ("i'm not in if and not in else Block")
```

Output

Program Name

Please execute the program and check the output
demo11.py

```
i = 10
if (i == 10):

    if (i < 15):
        print ("i is smaller than 15")

    if (i < 12):
        print ("i is smaller than 12 too")

    else:
        print ("i is greater than 15")
```

Output

Program Name

Please execute the program and check the output
demo12.py

```
x = int(input("Enter x value :"))
y = int(input("Enter y value :"))
z = int(input("Enter z value :"))

if x>y and x>z:
    print("x is greater than remaining two values")

elif y>z:
    print("y is greater than remaining two values")

else:
    print("z is greater than remaining two values ")
```

Output

Program Name Calculating the marks and check the output
demo13.py

```
math = int(input("Enter your maths marks:"))
sci = int(input("Enter your science marks:"))
kan = int(input("Enter your Kannada marks:"))

a = (math + sci +kan)/3

i = int(a)

print("The total average 3 subjects are :",i,"%")

if i>90 and i<=100:
    print("A+ bro")

elif i>80 and i<=90:
    print("A bro")

elif i>60 and i<=80:
    print("B+ bro")

elif i>50 and i<=60:
    print("B bro")

elif i>=30 and i<=35:
    print("C+ bro just pass")

elif i<35:
    print("Sorry boss, you failed")
```

Output

Program Name Please execute the program and check the output
demo14.py

```
print("Welcome to DANIEL Airport")
weight = float(input("How many pounds does your suitcase weigh? "))

if weight > 50:
    print("There is a $25 charge for luggage and already deducted from your credit card, don't get surprise.")

print("Thank you and Enjoy the Journey Boss.")
```

Output

Program Name Please execute the program and check the output
demo15.py

```
balance = -5

if balance < 0:
    print("Balance is in negative mode please add funds now or you will be charged a penalty.")
```

Output

Program Name Please execute the program and check the output
demo16.py

```
num = 22
if num % 2 == 0:
    print("Even Number")
else:
    print("Odd Number")
```

Output

Program Name Please execute the program and check the output
demo17.py

```
x=10

print(isinstance(x, int))
print(isinstance(x, str))
```

Output

Program Name Please execute the program and check the output
demo18.py

```
name="Daniel"

print(isinstance(name, str))
print(isinstance(name, int))
```

Output

Program Name Please execute the program and check the output
demo19.py

x=10

```
if isinstance(x, int):
    print("integer data type")

else:
    print("Its other type")
```

Output

Program Name Please execute the program and check the output
demo20.py

x=10

```
if isinstance(x, str):
    print("string data type")

else:
    print("It is an integer type")
```

Output

Program Name Please execute the program and check the output
demo21.py

```
name="daniel"

if isinstance(name, str):
    print("string data type")

else:
    print("It is other type")
```

Output

Program Name Please execute the program and check the output
demo22.py

```
name="Daniel"

if isinstance(xyz, int):
    print("integer data type")

else:
    print("It is string data type")
```

Output

**Program
Name**

Please execute the program and check the output
demo23.py

```
answer = input("Is Python teaching by Daniel, Yes or Not >> ")

if answer == "Yes":
    print("Welcome to Python sessions by Daniel, please don
't sleep")

else :
    print("I guess you are not student of Daniel.")

print("Thanks!")
```

Output

Program Name

Please execute the program and check the output
demo24.py

```
print("welcome to AiNextGen company...")

allowed_users = ['daniel', 'chiru', 'balayya']
username = input("What is your login name? : ")

if username in allowed_users:
    print("Access granted, Enjoy Guru :")

else:
    print("Sorry Boss, access denied: Contact Admin team.
Namaskaar")
```

Output

Program Name

Please execute the program and check the output
demo25.py

```
number = 4
guess = int(input('Guess number between 1 to 10 : '))

if guess == number:
    print("Congratulations Dude, you guessed it but no prizes
        :) ")

elif guess < number:
    print('No, it is a little higher than guessed number')

else:
    print('No, it is a little lower than guessed number')
```

Output

Program Name Please execute the program and check the output
demo26.py

```
x = int(input("What is the time?"))

if x < 10:
    print("Good morning")

elif x<12:
    print("Soon time for lunch")

elif x<18:
    print("Good day")

elif x<22:
    print("Good evening")

elif x>=22 and x<=24:
    print("Good night")

else:
    print("Dude, please Buy clock and learn timings, Good day")
```

Output

Program Name Please execute the program and check the output
demo27.py

```
marks=int(input("Enter marks: "))

if marks >= 90:
    print("A grade")

elif marks >=80:
    print("B grade")

elif marks >=70:
    print("C grade")

elif marks >= 65:
    print("D grade")

elif marks >= 55:
    print("E grade")

elif marks >= 35:
    print("Congrats for great achievement, hope you are from
Back bench")

else:
    print("Congratulation Dude, you have successfully failed")
```

Output

Program Name Please execute the program and check the output
demo28.py

```
marks=int(input("Enter marks: "))

if marks >= 90:
    if marks > 96:
        print("A++")

    elif marks > 93 and marks <= 96:
        print("A+")

    elif marks >= 90:
        print("A grade")

elif marks >=80:
    print("B grade")

elif marks >=70:
    print("C grade")

elif marks >= 65:
    print("D grade")

elif marks >= 55:
    print("E grade")

elif marks >= 35:
    print("Congrats for great achievement, hope you are from
Back bench")

else:
    print("Congratulation Dude, you have successfully failed")
```

Output

Program Name Please execute the program and check the output
demo29.py

```
while True:  
    n = input('Please enter 4 to quit application: ')  
    if n == '4':  
        break  
  
    print("Thank you")
```

Output

Program Name Please execute the program and check the output
demo30.py

```
number = 4
running = True

while running:
    guess = int(input('Please guess number between 1 to 10 : '))
    if guess == number:
        print('Congratulations Dude, you guessed it but no prizes :)')
        running = False
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')

else:
    print('The while loop is over.') 
```

Output

Program Name Please execute the program and check the output
demo31.py

```
for x in range(1, 10):
    if (x%2==0):
        print(x)
```

Output

Program Name Please execute the program and check the output
demo31.py

```
for x in range(1, 20):
    if (x%2==0) and (x%4==0):
        print(x)
```

Output

Program Name Please execute the program and check the output
demo32.py

```
data = [12, 45.678, 1+2j, True, 'daniel', [1, 2, 3]]
```

```
for item in data:
    print("Type of ",item, " is ", type(item))
```

Output

Program

Accepts a sequence of lines and print those lines by using list, when entered blank line then program terminates

Name demo33.py

```
lines = []
print(lines)

while True:
    l = input()

    if l:
        lines.append(l)
    else:
        break

print(lines)
```

Output

Program Name

Please execute the program and check the output
demo34.py

```
s = input("Enter string, mixed with numbers: ")  
l = 0  
for c in s:  
    if c.isalpha():  
        l = l+1  
  
print("Characters count from entered text:", l)
```

Output

Program Name

Please execute the program and check the output
demo35.py

```
s = input("Enter string, mixed with numbers: ")  
l = 0  
for c in s:  
    if c.isdigit():  
        l = l+1  
  
print("Numbers count from entered text:", l)
```

Output

Program Name Please execute the program and check the output
demo36.py

```
I = input("Enter any alphabet letter: ")

if I in ('a', 'e', 'i', 'o', 'u'):
    print(I, " is a vowel.")

else:
    print(I, " is a consonant.")
```

Output

Program Name Please execute the program and check the output
demo37.py

```
I = input("Enter any alphabet letter: ")

if len(I)==1:

    if I in ('a', 'e', 'i', 'o', 'u'):
        print(I, " is a vowel.")

    else:
        print(I, " is a consonant.")

else:
    print("Invalid input")
```

Output

Program Name Please execute the program and check the output
demo38.py

```
print("List of months: January, February, March, April, May,  
June, July, August, September, October, November, December")  
  
month_name = input("Enter name of Month: ")  
  
if month_name == "February":  
    print("No. of days: 28/29 days")  
  
elif month_name in ["April", "June", "September", "November"]:  
    print("No. of days: 30 days")  
  
elif month_name in ["January", "March", "May", "July",  
"August", "October", "December"]:  
    print("No. of days: 31 day")  
  
else:  
    print("Any month name like", month_name, "?")
```

Output

Program Name Please execute the program and check the output
demo39.py

```
n = int(input("Enter a number for table: "))

for i in range(1,11):
    print(n, 'x', i,'=',n*i)
```

Output

Program Name Please execute the program and check the output
demo40.py

```
x = ['Daniel', 'abhishek']
for i in x:
    print(i)
```

Output

Program Name Please execute the program and check the output
demo41.py

```
x = ['Daniel', 'abhishek']
for i in x:
    print(i.upper())
```

Output

Program Name Please execute the program and check the output
demo42.py

```
x = ['DANIEL', 'ABHISHEK']
for i in x:
    print(i)
```

Output

Program Name Please execute the program and check the output
demo43.py

```
x = ['DANIEL', 'ABHISHEK']
for i in x:
    print(i.lower())
```

Output

Program Name Please execute the program and check the output
demo44.py

```
x = [1, 2, 3]
for i in x:
    print(i.upper())
```

Output

Program Name Please execute the program and check the output
demo45.py

```
i = 1
while i<=20:
    if i%2 == 0:
        break
    print(i)
    i += 2
```

Output

Program Name Please execute the program and check the output
demo46.py

```
i = 1
while i<=20:
    if i%3 == 0:
        break
    print(i)
    i += 2
```

Output

Program Name Please execute the program and check the output
demo47.py

```
i = 1
while False:
    if i%3 == 0:
        break
    print(i)
    i += 2
```

Output

Program Name Please execute the program and check the output
demo48.py

```
True = False
while True:
    print(True)
    break
```

Output

Program Name Please execute the program and check the output
demo49.py

```
i = 0
while i < 5:
    print(i)
    i += 1
    if i == 3:
        break
else:
    print(0)
```

Output

Program Name Please execute the program and check the output
demo50.py

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print(0)
```

Output

Program Name Please execute the program and check the output
demo51.py

```
x = "python"
for i in x:
    print(i)
```

Output

Program Name Please execute the program and check the output
demo52.py

```
x = "python"  
for i in x:  
    print(i, end= "")
```

Output

Program Name Please execute the program and check the output
demo53.py

```
x = "python"  
for i in x:  
    print(i, end= " ")
```

Output

Program Name Please execute the program and check the output
demo54.py

```
x = "python"  
for i in x:  
    print(i+"Z")
```

Output

Program Name Please execute the program and check the output
demo55.py

```
x = "python"  
if i in x:  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo56.py

```
x = "python"  
while i in x:  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo57.py

```
x = "python"  
i="a"  
while i in x:  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo58.py

```
x = 'abcd'  
for i in range(10):  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo59.py

```
x = 'abcd'  
for i in range(x):  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo60.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo61.py

```
x = 'abcd'  
for i in range(x):  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo62.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(i.upper())
```

Output

Program Name Please execute the program and check the output
demo63.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(x[i])
```

Output

Program Name Please execute the program and check the output
demo64.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(i[x])
```

Output

Program Name Please execute the program and check the output
demo65.py

```
x = 123
for i in x:
    print(i)
```

Output

Program Name Please execute the program and check the output
demo66.py

```
for i in range(0):
    print(i)
```

Output

Program Name Please execute the program and check the output
demo67.py

```
for i in range(2):
    print(i)
```

Output

Program Name Please execute the program and check the output
demo68.py

```
for i in range(2.0):  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo69.py

```
for i in range(int(2.0)):  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo70.py

```
for i in "abcd":  
    print(i)
```

Output

Program Name Please execute the program and check the output
demo71.py

```
for i in " ":
    print(i)
```

Output

Program Name Please execute the program and check the output
demo72.py

```
x = 2
for i in range(x):
    x += 1
    print (x)
```

Output

Program Name Please execute the program and check the output
demo73.py

```
x = 2
for i in range(x):
    x -= 2
    print (x)
```

Output

Program Name Please execute the program and check the output
demo74.py

```
for i in range(10):
    if i == 5:
        break
    else:
        print(i)
else:
    print("no value")
```

Output

Program Name Please execute the program and check the output
demo75.py

```
for i in range(5):
    if i == 5:
        break
    else:
        print(i)
else:
    print("no value")
```

Output

Program Name Please execute the program and check the output
demo76.py

```
text = "my name is python"  
for i in text:  
    print (i, end=",")
```

Output

11. Python - Functions - Part - 1

Table of Contents

1. Function	2
2. Types of function.....	2
2.1. Pre-defined or built-in functions.....	2
2.2. User Defined Functions:.....	3
3. Function related terminology	3
4. Function definition	4
4.1. Creating a function.....	4
4.2. Calling a function	6
6. A Function can call other function.....	10
5. Based on Parameters: Functions are two types.....	12
5.1. Function without parameters	12
5.2. Function with parameters.....	13
7. return keyword in python.....	17
7.1. Function without return.....	18
7.2. Function with return	19
7.3. return vs None	25
7.4. A function can return multiple values	26

11. Python - Functions - Part - 1

1. Function

- ✓ A function can contain group of statements which performs the task.

Advantages

- ✓ Maintaining the code is an easy way.
- ✓ Code reusability.

Make a note

- ✓ `print()` is predefined function in python which prints output on the console.

2. Types of function

- ✓ There are two types of functions,
 - Pre-defined or built-in functions
 - User-defined functions

2.1. Pre-defined or built-in functions

- ✓ The functions which are already existing in python are called as predefined function

Examples

- `print(p)`
- `type(p)`
- `input(p)`

2.2. User Defined Functions:

- ✓ Based on requirement a programmer can create his own function, these functions are called as user defined functions.
- ✓ So, practically we will see how to define and use, user defined functions.

3. Function related terminology

- ✓ If we want to understand function concept in better way then we need to focus on function related terminology,
 - **def** keyword
 - name of the function
 - parenthesis ()
 - parameters (if required)
 - colon symbol:
 - function body
 - **return** type (optional)

4. Function definition

- ✓ A function can contains group of statements.
- ✓ The purpose of function is to perform an operation.
- ✓ A function can contain mainly two parts,
 1. Creating a function
 2. Calling a function

4.1. Creating a function

- ✓ Very first step is we need to create a function.
- ✓ We need to use **def** keyword to create a function.
- ✓ After **def** keyword we should write name of the function.
- After function name, we should write parenthesis **()**
 - This parenthesis may contain parameters.
 - Parameters are just like variables which receive the values
 - If function having parameters, then we need to provide the values while calling.
 - We will learn more in parameterized function
- After parenthesis we should write colon : **symbol**
- After : symbol in next line we should provide indentation
- ✓ Function body.
 - Actual logic contains by function body
 - This function body helps to perform the operation.
- ✓ Before closing the function, function may contain return type.

Syntax

```
def functionname():
    """ doc string"""
    Body of the function to perform operation
```

A naming convention to define a function

- ✓ As discussed in Naming convention chapter, function name should be in lower case.
- ✓ If name having multiple words, then separating words with underscore (_) symbol is a good practice.

Program Name Creating a function
demo1.py

```
# function creation
def display():
    print("Welcome to function")
```

output

Make a note

- ✓ When we execute above program, then function body not executed.
 - ✓ To execute function body, we need to call the function.
-

4.2. Calling a function

- ✓ After function is created then we need to call that function to execute the function body.
- ✓ While calling the function, function name should be match otherwise we will get error.

Program Name	Create and call user defined function demo2.py
	# function creation def display(): print("Welcome to function concept")
	# function calling display()
output	Welcome to function concept

Data Science - Python Functions - Part - 1

Program Name Create and call user defined function
demo3.py

```
# function creation
def display():
    print("Welcome to function concept")

# function calling
display()
display()
```

output

```
Welcome to function concept
Welcome to function concept
```

Program Name Create and call user defined function
demo4.py

```
# function creation
def display():
    print("Welcome to function concept")

# function calling
details()
```

output

```
NameError: name 'details' is not defined
```

Question

- Can i create more than one function in a single python program?

Answer

- Yes, we can
- Based on requirement we can create any number of functions.

Program Name Creating two functions and calling those functions
demo5.py

```
def first():
    print("This is first function")

def second():
    print("This is second function")

first()
second()
```

output
This is first function
This is second function

Program Name Creating two functions and calling those functions
demo6.py

```
def first():
    print("This is first function")

def second():
    print("This is second function")

second()
first()
```

output

```
This is second function
This is first function
```

6. A Function can call other function

- ✓ Based on requirement a function can call another function as well.
- ✓ We can call a function inside another function.

Syntax

```
def first_function():
    body of the first function
    we can call the secondfunction
```

```
def second_function():
    body of the second function
```

first function calling

Program Name Creating two functions
demo7.py

```
def m1():
    print("first function")

def m2():
    print("second function")
```

```
m1()
m2()
```

output

```
first function
second function
```

Program Name One function can call another function
demo8.py

```
def m1():
    print("first function")
    m2()

def m2():
    print("second function")

m1()
```

output

```
first function
second function
```

5. Based on Parameters: Functions are two types

- ✓ Based on parameters, functions can be divided into two types,
 - Function without parameters (**or**) No parameterised function
 - Function with parameters (**or**) Parameterised function

5.1. Function without parameters

- ✓ If a function having no parameters then that function is called as, No parameterized function

Syntax

```
def nameofthefunction():
    body of the function to perform operations

function calling
```

Program Name	Function which having no parameters demo9.py
	# defining a function def display(): print("Welcome to function which having no parameters")
output	>Welcome to function which having no parameters

5.2. Function with parameters

- ✓ If a function having parameters then that function called as parameterised function

Why function having parameters?

- ✓ Function parameters help to process the function operation.
- ✓ When we pass parameters then,
 - Function capture parameters values
 - These values perform the operations.
 - Finally it brings the result.

Syntax

```
def functionname(parameter1, parameter2, ...):  
    body of the function  
function calling
```

Program Name One parameterized function
demo10.py

```
def testing(a):  
    print("one parameterised function:", a)  
  
testing(10)
```

output
one parameterised function: 10

Program Name One parameterized function
demo11.py

```
def testing(a):
    print("one parameterised function:", a)

testing(10.56)
```

output
one parameterised function: 10.56

Program Name One parameterized function
demo12.py

```
def testing(a):
    print("one parameterised function:", a)

testing("Daniel")
```

output
one parameterised function: Daniel

Program Name One parameterized function
demo13.py

```
def testing(a):
    print("one parameterised function:", a)
```

```
x = input("Enter a value:")
testing(x)
```

output

```
Enter a value: 10
one parameterised function: 10
```

Program Name Two parameterized function
demo14.py

```
def testing(a, b):
    print("two parameterised function:", a, b)
```

```
testing(10, 20)
```

output
two parameterised function: 10 20

Program Name Function performing addition operation
demo15.py

```
def addition(a, b):
    print("Addition of two values=", (a+b))

addition(10, 20)
```

output
Addition of two values =30

7. return keyword in python

- ✓ Based on return statement, functions can be divided into two types,
 - Function without return statement
 - Function with return statement
- ✓ return is a keyword in python.
- ✓ We should use return statement with function or method, otherwise we will get error.

Program Name return outside of function which is invalid
demo16.py

```
print('Hello')  
return 100
```

output
SyntaxError: 'return' outside function

7.1. Function without return

- ✓ If a function cannot contains return statement then that function is called as a function without return statement.
- ✓ It's not mandatory to write return statement to a function.
- ✓ A function without return statement is valid.

Program Name Function displaying information
demo17.py

```
def balance():
    print("My bank balance is: ")
```

```
balance()
```

output
My bank balance is:

7.2. Function with return

- ✓ Based on requirement we can write return statement to a function.
- ✓ A function with return statement is valid.

Syntax

```
def nameofthefunction():
    body of the function
    return result
```

Program Name Function with return statement displaying information
 demo18.py

```
def balance():
    print("My bank balance is: ")
    return 100
```

```
balance()
```

output My bank balance is:

Note

- ✓ If a function contains return statement then that function calling we need to assign to a variable.
 - ✓ Daniel why we need to assign to a variable?
 - ✓ Yes, i will explain please wait in another five minutes, then you can understand.
-

Program Name Function with return statement
demo19.py

```
def balance():
    print("My bank balance is: ")
    return 100

b = balance()
print(b)
```

output

```
My bank balance is:
100
```

Why we need to assign a function calling to a variable?

- ✓ If we assign function calling to a variable then that variable holding the variable value.
- ✓ That variable we can use further in our program.

Program Name Function with return statement
demo20.py

```
def balance():
    return 100

b = balance()

if b==0:
    print("Balance is: ", b)

elif b<=0:
    print("Balance is: ", b, " negative please deposit")

else:
    print("Balance is: ", b)
```

output
Balance is: 100

Program Name Function with return statement
demo21.py

```
def balance():
    return 0

b = balance()

if b == 0:
    print("Balance is: ", b)

elif b<=0:
    print("Balance is: ", b, " negative please deposit")

else:
    print("Balance is: ", b)
```

output
Balance is: 0

Program Name Function with return statement
demo22.py

```
def balance():
    return -50

b = balance()

if b == 0:
    print("Balance is: ", b)

elif b <= 0:
    print("Balance is: ", b, "it is negative please deposit")

else:
    print("Balance is: ", b)
```

output
Balance is: -50 it is negative please deposit

Note

- ✓ Below program also valid but not recommended.

Program Function with return statement

Name demo23.py

```
def balance():
    print("My bank balance is: ")
    return 100

print(balance())
```

output

```
My bank balance is:
100
```

Note

- ✓ A method can return a value as well.
- ✓ We will learn this concept again in OOPS chapter.

7.3. return vs None

- ✓ If any function is not return anything, then by default that function returns **None** data type.
- ✓ We can also say as, if we are not writing return statement, then default return value is None

Program Name function returning the None value
demo24.py

```
def m1():
    print("This function is returning nothing")

x = m1()
print(x)
```

output

```
This function is returning nothing
None
```

7.4. A function can return multiple values

- ✓ In python, a function can return multiple values.
- ✓ Based on requirement a function can return multiple values.
 - If function is returning two values then while function calling we need to assign to two variables
 - If function is returning three values then while function calling we need to assign to three variables.
 - If function is returning more than one values, while calling function if we assign with one variable then all values will be stored in tuple.

Syntax

```
def name_of_the_function():
    body of the function
    return value1, value2, value3,...,valueN
```

Program Name Define a function which can return multiple values
demo25.py

```
def m1():
    a = 10
    b = 11
    return a, b

x, y = m1()

print("first value is:", x)
print("second value is:", y)
```

output

```
first value is: 10
second value is: 11
```

Program Name Define a function which can return multiple values
demo26.py

```
def m1():
    a = 10
    b = 11
    c = 12

    return a, b, c

x, y, z = m1()

print("first value is:", x)
print("second value is:", y)
print("third value is:", z)
```

output

```
first value is: 10
second value is: 11
third value is: 12
```

Program Name Define a function which can return multiple values
demo27.py

```
def m1():
    a = 10
    b = 11
    c = 12
    return a, b, c

x = m1()
print("all values:", x)
```

output
all values: (10, 20, 30)

Program Name A function with parameters and return type.
demo28.py

```
def add(x, y, z):
    result = x+y+z
    return result

r = add(10, 20, 30)

print("Addition of values:", r)
```

output
Addition of values: 60

Program Name Please execute the program and check the output
demo1.py

```
s="hello"  
print(s)  
print(type(s))
```

Output

Program Name Please execute the program and check the output
demo2.py

```
s="hello"  
print(length(s))
```

Output

Program Name Please execute the program and check the output
demo3.py

```
s="hello"  
print(len(s))
```

Output

Program Name Please execute the program and check the output
demo4.py

```
s1="hello"  
s2="hello"  
print(id(s1))  
print(id(s2))
```

Output

Program Name Please execute the program and check the output
demo5.py

```
s1="hello"  
s2="hello"  
print(s1 is s2)
```

Output

Program Name Please execute the program and check the output
demo6.py

```
s1="hello"  
s2="hello"  
s3="hi"  
print(s1 is s2)  
print(s2 is s3)
```

Output

Program Name Please execute the program and check the output
demo7.py

```
s1="hello"  
s2="hello"  
s3="hi"  
print(s1 == s2)  
print(s2 == s3)
```

Output

Program Name Please execute the program and check the output
demo8.py

```
print("a"+"bc")
```

Output

Program Name Please execute the program and check the output
demo9.py

```
print("abcd"[2:])
```

Output

Program Name Please execute the program and check the output
demo10.py

```
str1 = 'hello'  
str2 = ','  
str3 = 'world'  
  
print(str1[-1:])
```

Output

Program Name Please execute the program and check the output
demo11.py

```
print(r"\nhello")
```

Output

Program Name Please execute the program and check the output
demo12.py

```
print('new' 'line')
```

Output

Program Name Please execute the program and check the output
demo13.py

```
str1="helloworld"
```

```
print(str1[::-1])
```

Output

Program Name Please execute the program and check the output
demo14.py

```
example = "snow world"
```

```
example[3] = 's'
```

```
print(example)
```

Output

Program Name Please execute the program and check the output
demo15.py

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

```
print(min(l))
```

```
print(max(l))
```

Output

Program Name Please execute the program and check the output
demo16.py

```
q="what are you"  
print(min(q))  
print(max(q))
```

Output

Program Name Please execute the program and check the output
demo17.py

```
example="hello"  
print(example.count("l"))
```

Output

Program Name Please execute the program and check the output
demo18.py

```
example = "helle"  
print(example.find("e"))
```

Output

Program Name Please execute the program and check the output
demo19.py

```
example = "helle"  
print(example.rfind("e"))
```

Output

Program Name Please execute the program and check the output
demo20.py

```
example="helloworld"  
print(example[::-1])
```

Output

Program Name Please execute the program and check the output
demo21.py

```
str1="restart"  
  
char = str1[0]  
str1 = str1.replace(char, '$')  
print(str1)
```

Output

Program Name Please execute the program and check the output
demo22.py

```
str1="restart"  
  
char = str1[0]  
str1 = str1.replace(char, '$')  
str2 = char + str1[1:]  
  
print(str2)
```

Output

Program Name Please execute the program and check the output
demo23.py

```
example="helloworld"  
print(example[::-1].startswith("d"))
```

Output

Program Name Please execute the program and check the output
demo24.py

```
print("hello\example\test.txt")
```

Output

Program Name Please execute the program and check the output
demo25.py

```
print("hello\\example\\test.txt")
```

Output

Program Name Please execute the program and check the output
demo26.py

```
print("hello\"example\"test.txt")
```

Output

Program Name Please execute the program and check the output
demo27.py

```
print("hello"\example"\test.txt")
```

Output

Program Name Please execute the program and check the output
demo28.py

```
print("hello"+1+2+3)
```

Output

Program Name Please execute the program and check the output
demo29.py

```
print("D", end = ' ')
print("C", end = ' ')
print("B", end = ' ')
print("A", end = ' ')
```

Output

Program Name Please execute the program and check the output
demo30.py

```
print("Hello".replace("l", "e"))
```

Output

Program Name Please execute the program and check the output
demo31.py

```
print("abc DEF".capitalize())
```

Output

Program Name Please execute the program and check the output
demo32.py

```
print("abcdef".upper())
```

Output

Program Name Please execute the program and check the output
demo33.py

```
print("ABCDEF".upper())
```

Output

Program Name Please execute the program and check the output
demo34.py

```
print("ABCDEFG".lower())
```

Output

Program Name Please execute the program and check the output
demo35.py

```
print("abcdef".center())
```

Output

Program Name Please execute the program and check the output
demo36.py

```
print("xyzxyzxyz".count('yy'))
```

Output

Program Name Please execute the program and check the output
demo37.py

```
print("xyzxyzxyz".count('yy', 1))
```

Output

Program Name Please execute the program and check the output
demo38.py

```
print("xyzxyzxyz".count('yy', 4))
```

Output

Program Name Please execute the program and check the output
demo39.py

```
print("xyzxyzxyz".endswith("yyy"))
```

Output

Program Name Please execute the program and check the output
demo40.py

```
print("ab\ tcd\ tef")
```

Output

Program Name Please execute the program and check the output
demo41.py

```
print("a\nb")
```

Output

Program Name Please execute the program and check the output
demo42.py

```
print("abcdef".find("cd"))
```

Output

Program Name Please execute the program and check the output
demo43.py

```
print("ccdcddcd".find("c"))
```

Output

Program Name Please execute the program and check the output
demo44.py

```
print("Hello {0} and {1}".format('foo', 'bin'))
```

Output

Program Name Please execute the program and check the output
demo45.py

```
print("Hello {1} and {0}".format('bin', 'foo'))
```

Output

Program Name Please execute the program and check the output
demo46.py

```
print("Hello {} and {}".format('foo', 'bin'))
```

Output

Program Name Please execute the program and check the output
demo47.py

```
print("Hello {name1} and {name2}".format('foo', 'bin'))
```

Output

Program Name Please execute the program and check the output
demo48.py

```
print("Hello {name1} and  
{name2}".format(name1='foo',name2='bin'))
```

Output

Program Name Please execute the program and check the output
demo49.py

```
print('The sum of {0} and {1} is {2}'.format(2, 10, 12))
```

Output

Program Name Please execute the program and check the output
demo50.py

```
print('ab12'.isalnum())
```

Output

Program Name Please execute the program and check the output
demo51.py

```
print('ab,12'.isalnum())
```

Output

Program Name Please execute the program and check the output
demo52.py

```
print('ab'.isalpha())
```

Output

Program Name Please execute the program and check the output
demo53.py

```
print('a B'.isalpha())
```

Output

Program Name Please execute the program and check the output
demo54.py

```
print('0xa'.isdigit())
```

Output

Program Name Please execute the program and check the output
demo55.py

```
print('.isdigit())
```

Output

Program Name Please execute the program and check the output
demo56.py

```
print('my_string'.isidentifier())
```

Output

Program Name Please execute the program and check the output
demo57.py

```
print('$my_string'.isidentifier())
```

Output

Program Name Please execute the program and check the output
demo58.py

```
print('abc'.islower())
```

Output

Program Name Please execute the program and check the output
demo59.py

```
print('a@ 1'.islower())
```

Output

Program Name Please execute the program and check the output
demo60.py

```
print('11'.isnumeric())
```

Output

Program Name Please execute the program and check the output
demo61.py

```
print('HelloWorld'.istitle())
```

Output

Program Name Please execute the program and check the output
demo62.py

```
print('Hello World'.istitle())
```

Output

Program Name Please execute the program and check the output
demo63.py

```
s1=""" hello  
      hi  
      how are you""  
print(s1)
```

Output

Program Name Please execute the program and check the output
demo64.py

```
s1=""" hello  
      hi  
      how are you"""  
print(s1.strip())
```

Output

Program Name Please execute the program and check the output
demo65.py

```
print('abcdef'.partition('cd'))
```

Output

Program Name Please execute the program and check the output
demo66.py

```
print('abcdefcdgh'.partition('cd'))
```

Output

Program Name Please execute the program and check the output
demo67.py

```
print('abcd'.partition('cd'))
```

Output

Program Name Please execute the program and check the output
demo68.py

```
print('abcdef12'.replace('cd', '12'))
```

Output

Program Name Please execute the program and check the output
demo69.py

```
print('abef'.replace('cd', '12'))
```

Output

Program Name Please execute the program and check the output
demo70.py

```
print('abcdefghijklm'.split('cd'))
```

Output

Program Name Please execute the program and check the output
demo71.py

```
print('Ab!2'.swapcase())
```

Output

Program Name Please execute the program and check the output
demo72.py

```
print('ab cd ef'.title())
```

Output

Program Name Please execute the program and check the output
demo73.py

```
print('ab cd-ef'.title())
```

Output

1. Accessing string characters

- ✓ We can access string characters by using,

- Indexing
- Slicing

1.1. Indexing in string

- ✓ Indexing means a position of string's characters where it stores.
- ✓ We need to use square brackets [] to access the string index.
- ✓ String indexing result is string type.

Python support two types of indexes

- ✓ Positive index
- ✓ Negative index

1.2. Positive index

- ✓ The position of string characters can be positive index from **left to right** direction (we can say forward direction).
- ✓ In this way, the starting position is **0**(zero)

1.3. Negative index

- ✓ The position of string characters can be negative index from **right to left** direction (we can say backward direction).
- ✓ In this way, the starting position is **-1**(minus one)
 - Please don't think too much like **-0** (minus zero), there is no minus zero boss.

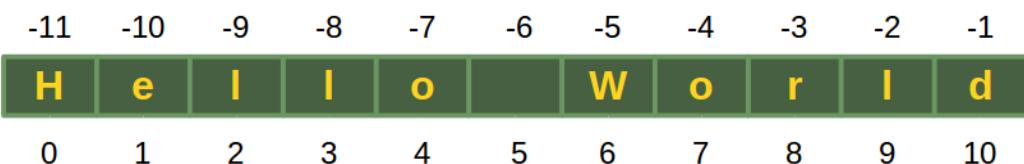
Internal representation

Program Name Printing text message
demo74.py

```
wish = "Hello World"  
print(wish)
```

output
Hello World

Diagram representation



Program Name Accessing string index by using integer values.
demo75.py

```
wish = "Hello World"  
  
print(wish[0])  
print(wish[1])
```

output
H
e

Program Name Accessing string index by using negative values.
demo76.py

```
wish = "Hello World"  
  
print(wish[-1])  
print(wish[-2])
```

output

```
d  
l
```

1.4. Slicing in string

- ✓ A substring of a string is called as a slice.
- ✓ A slice can represent a part of string from string or a piece of string.
- ✓ String slicing result is string type.
- ✓ We need to use square brackets [] in slicing.

Syntax 1:

```
nameofthestring [start: stop]
```

- ✓ **start**
 - It indicates the index where slice can start.
 - Default value is 0
- ✓ **stop**
 - It indicates the index where slice can end.
 - Default value is max allowed index of list i.e. length of the string

Syntax2:

nameofthestring [start : stop :step]

✓ **start**

- It indicates the index where slice can start.
- Default value is 0

✓ **stop**

- It indicates the index where slice can end.
- Default value is max allowed index of list i.e. length of the string

✓ **Step size**

- Increment value.
- Default value is 1

2. String several cases in slicing

Make a note

- ✓ If we are not specifying begin index, then it will consider from beginning of the string.
- ✓ If we are not specifying end index, then it will consider up to end of the string.
- ✓ The default value for step is 1

```
wish = "Hello World"
```

- ✓ wish[:] => accessing from 0th to last
- ✓ wish[::] => accessing from 0th to last
- ✓ wish[2::] => accessing from 2nd position to till last.
- ✓ wish[4::] => accessing from 4th position to till last.
- ✓ wish[:8:] => accessing from 0th to 7th in steps of 1
- ✓ wish[0:9:1] => accessing string from 0th to 8th means (9-1) element.

Program Name slice operator several use cases
demo77.py

```
wish = "Hello World"  
  
print(wish[:])  
print(wish[::])  
print(wish[2::])  
print(wish[4::])  
print(wish[:8:])  
print(wish[0:9:1])
```

Output

```
Hello World  
Hello World  
llo World  
o World  
Hello Wo  
Hello Wor
```

12. Python - Functions - Part 2

Table of Contents

1. Formal and actual arguments	2
2. Types of arguments	3
2.1. Positional arguments	4
2.2. Keyword arguments.....	6
2.3. Default arguments	8
2.4. * args or variable length arguments	10
3. Anonymous functions or Lambdas	13
3.1. map(p1, p2) function	17
3.2. filter(p1, p2) function.....	18
3.3. reduce(p1, p2) function	19

12. Python - Functions - Part 2

1. Formal and actual arguments

- ✓ When a function is defined it may have some parameters.
- ✓ These parameters receive the values.
 - Parameters are called as a '**formal arguments**'
 - When we call the function, we should pass values or data to the function.
 - These values are called as '**actual arguments**'

Program Name Formal and actual arguments
demo1.py

```
def add(a, b):  
    c = a + b  
    print(c)  
  
# call the function  
  
x = 10  
y = 15  
add(x, y)
```

output
25

- ✓ **a** and **b** called as formal arguments
- ✓ **x** and **y** called actual arguments

2. Types of arguments

In python there are 4 types of actual arguments are existing,

1. positional arguments
2. keyword arguments
3. default arguments
4. variable length arguments (*args)

2.1. Positional arguments

- ✓ These are the arguments passed to a function in correct positional order.
- ✓ The number of arguments and position of arguments should be matched, otherwise we will get error.

Program Name Positional arguments
demo2.py

```
def sub(x, y):  
    print(x-y)
```

calling function

```
sub(20, 10)
```

output
10

Make a note

- ✓ If we change the number of arguments, then we will get error.
- ✓ This function accepts two arguments then if we are trying to provide three values then we will get error.

Program Error: Positional arguments
Name demo3.py

```
def sub(x, y):
    print(x-y)

# calling function

sub(10, 20, 30)
```

output

TypeError: sub() takes 2 positional arguments but 3 were given

2.2. Keyword arguments

- ✓ Keyword arguments are arguments that recognize the parameters by the name of the parameters.
- ✓ We can use an identifier (name of the variable) to provide values to the function parameters.

Program Name keyword arguments
demo4.py

```
def cart(product, price):
    print("Product is :" , product)
    print("cost is :" , price)

cart(product = "bangles", price = 200000)
```

output

Product is: bangles
cost is: 200000

Program Name keyword arguments
demo5.py

```
def cart(product, price):
    print("Product is :" , product)
    print("cost is :" , price)

cart(product = "handbag ", price = 100000)
```

output

Product is: handbag
cost is: 100000

Program Name keyword arguments
demo6.py

```
def cart(product, price):
    print("Product is:" , product)
    print("cost is:" , price)

cart(price = 1200, product = "shirt")
```

output
Product is: shirt
cost is: 1200

2.3. Default arguments

- ✓ During function definition we can provide default values to function parameters
- ✓ While creating a function, we can provide values to the function parameters.

Program Name Default arguments
demo7.py

```
def cart(product, price = 40.0):
    print("Product is :" , product)
    print("cost is :" , price)

# calling function

cart(product = "pen")
```

output
Product is: pen
Cost is : 40.0

Program Name Default arguments
demo8.py

```
def cart(product, price = 40.0):
    print("product is :", product)
    print("cost is :", price)

cart(product = "handbag", price = 10000)
```

output
Product is: handbag
Cost is : 10000

Program Name Default arguments
demo9.py

```
def cart(product, price = 40.0):
    print("Product is:", product)
    print("cost is:", price)

cart(price = 500, product = "shirt")
```

output
Product is: shirt
Cost is : 500

Make a note

- ✓ If we are not passing any value, then only default value will be considered.

2.4. * args or variable length arguments

- ✓ *args also called as variable length arguments.
- ✓ Sometimes our requirement can be like, need to provide more values to function parameter to process the result. Here we can use *args concept.
- ✓ It represents single star symbol before the argument name.
- ✓ Internally the provided values will be stored in a tuple.

Few points

- ✓ If we define one parameterised function then during function calling we need to pass one value.
- ✓ If we define two parameterised function then during function calling we need to pass two values, if we pass more or less than two values then we will get error.

Syntax

```
def nameofthefunction(*x):  
    body of the function
```

- *x is variable length argument
- ✓ Now we can pass any number of values to this *x.
- ✓ Internally the provided values will be represented in tuple.

Program Name Variable length argument
demo10.py

```
def m(x):  
    print(x)
```

```
m(10)
```

output
10

Program Name Variable length argument
demo11.py

```
def m(x):  
    print(x)
```

```
m(10, 20)
```

output
TypeError: m() takes 1 positional argument but 2 were given

Program Name Variable length argument
demo12.py

```
def m(*x):  
    print(x)
```

```
m(10)
```

output
(10)

Program Name Variable length argument
demo13.py

```
def m(*x):  
    print(x)
```

```
m(10, 20)
```

output
(10, 20)

Program Name Variable length argument
demo14.py

```
def m(*x):  
    print(x)
```

```
m(10, 20, 30)
```

output
(10, 20, 30)

3. Anonymous functions or Lambdas

- ✓ Generally we can create normal function by using def keyword
- ✓ lambda is a keyword in python.
- ✓ By using **lambda** keyword we can create lambda function.
- ✓ Lambda function also called as anonymous function.
- ✓ A function without a name is called as anonymous function.
- ✓ Lambda function will process the input and return the result.

Lambda function syntax

```
lambda argument_list: expression
```

Advantage

- ✓ By using Lambda Functions, we can write very concise code so that readability of the program will be improved.

Program anonymous function
Name demo16.py

```
s = lambda a: a*a  
  
x = s(4)  
print(x)
```

output

16

- ✓ Here because of lambda keyword it creates anonymous function.

A simple difference between normal and lambda functions

Program Name To find square by using a normal function
demo17.py

```
def square(t):
    return t*t

s=square(2)
print(s)
```

output
4

Program Name To find sum of two values by using normal function
demo18.py

```
def add(x, y):
    return x + y

b = add(2, 3)
print(b)
```

output
5

Program Name To find sum of two values by using anonymous function demo19.py

```
add = lambda x, y: x+y  
  
result = add(1, 2)  
print("The sum of value is:", result)
```

output

```
3
```

Make notes

- ✓ Lambda Function internally returns expression value and we **no need to write return statement** explicitly.

Where lambda function fits exactly?

- ✓ Sometimes we can pass function as argument to another function. In such cases lambda functions are best choice.
- ✓ We can use lambda functions very commonly with filter(), map() and reduce() functions, these functions expect function as argument.

Lambda functions

- ✓ map(p1, p2) function
- ✓ filter(p1, p2) function
- ✓ reduce(p1, p2) function

3.1. map(p1, p2) function

- ✓ map(fun, iterable) is a predefined function in python.
- ✓ This function takes iterable and apply logic on every item and returns new iterable.

Syntax

```
map(function, sequence)
```

Program Name Find square by using map function

Name demo20.py

```
without_gst_cost = [100, 200, 300, 400]
with_gst_cost = map(lambda x: x+10, without_gst_cost)

x = list(with_gst_cost)
print("Without GST items costs: ", without_gst_cost)
print("With GST items costs: ", x)
```

output

```
Without GST items costs: [100, 200, 300, 400]
With GST items costs: [110, 210, 310, 410]
```

3.2. filter(p1, p2) function

- ✓ filter(fun, iterable) is a predefined function in python.
- ✓ This function takes iterable and apply filtering logic on every item and returns new iterable.

Syntax

```
filter(function, sequence)
```

Program Name example by using filter function
demo21.py

```
items_cost = [999, 888, 1100, 1200, 1300, 777]
gt_thousand = filter(lambda x : x>1000, items_cost)

x = list(gt_thousand)
print("Eligible for discount:", x)
```

output
Eligible for discount: [1100, 1200, 1300]

3.3. reduce(p1, p2) function

- ✓ `reduce(fun, iterable)` is a predefined function existing in `functools` module.
- ✓ This function apply a function of two arguments cumulatively to the items of sequence, from left to right.
- ✓ Finally this function reduce the sequence to a single value.

Syntax

```
reduce(function, sequence)
```

Program

Name reduce function

demo22.py

```
from functools import reduce
```

```
items_cost = [111, 222, 333, 444]
```

```
total_cost = reduce(lambda x, y : x+y, items_cost)  
print(total_cost)
```

output

```
1110
```

- ✓ For example, `reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` calculates $((((1+2)+3)+4)+5)$

13. Python - Module

Table of Contents

1. Modules.....	2
2. import module.....	3
3. Renaming or aliasing a module.....	4
4. from and import keywords	5
5. import * (star symbol).....	6
6. Member aliasing.....	6

13. Python - Module

1. Modules

- ✓ In python a module means, a saved python file.
- ✓ This file can contain a group of classes, methods, functions and variables.
- ✓ Every Python file mean **.py** or **.python** extension file is called as a module.

```
Program    module program
Name      additionmultiplication.py

x = 10

def addition(a, b):
    print("Sum of two values: ", (a+b))

def multiplication(a, b):
    print("Multiplication of two values: ", (a*b))
```

- ✓ Now **additionmultiplication.py** file is a module.
- ✓ **additionmultiplication.py** module contains one variable and two functions.

2. import module

- ✓ **import** is a keyword in python.
- ✓ By using import keyword we can import (get) modules in our program.
- ✓ Once we imported the module then we can use members (variables, functions & etc) of module.

Program Name importing **additionmultiplication** module and calling members
demo1.py

```
import additionmultiplication

print(additionmultiplication.x)
additionmultiplication.addition(1, 2)
additionmultiplication.multiplication(2, 3)
```

output

```
10
Sum of two values: 3
Multiplication of two values: 6
```

Make a note:

- ✓ Whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently.

3. Renaming or aliasing a module.

- ✓ **as** is a keyword in python
- ✓ By using **as** keyword we can rename/alias existing module.

Syntax

```
import additionmultiplication as admul
```

- ✓ Here **additionmultiplication** is module name and alias name is **admul**
- ✓ We can access members by using alias name **admul**

Program Name

```
importing module  
demo2.py
```

```
import additionmultiplication as admul  
  
print(admul.x)  
admul.addition(1,2)  
admul.multiplication(3, 4)
```

output

```
10  
Sum of two values: 3  
Multiplication of two values: 12
```

4. from and import keywords

- ✓ **from** is keyword in python
- ✓ We can import some specific members of module by using **from** keyword.
- ✓ The main advantage of **from** keyword is we can access members directly without using module name.

Program Name from and import keywords
demo3.py

```
from additionmultiplication import x, addition
print(x)
addition(10,20)
```

output
10
Sum of two values: 30

Program Name **NameError:** name 'multiplication' is not defined
demo4.py

```
from additionmultiplication import x, addition
print(x)
multiplication(10,20)
```

Error
10
NameError: name 'multiplication' is not defined

5. import * (star symbol)

- ✓ We can use * symbol to import all members of a module.
- ✓ We can import all members of a module as by using import * (symbol)

<p>Program Name</p> <p>demo5.py</p>	<pre>importing by using * from additionmultiplication import * print(x) addition(10, 20) multiplication(10, 20)</pre>
output	<pre>10 Sum of two values: 30 Multiplication of two values: 200</pre>

6. Member aliasing

- ✓ We can give alias name to the members of a module

<p>Program Name</p> <p>demo6.py</p>	<pre>member aliasing from additionmultiplication import x as y, addition as add print(y) add(10, 20)</pre>
output	<pre>10 Sum of two values: 30</pre>

14. Python Package

Contents

1. What is a package?.....	2
2. __init__.py file.....	2
3. Advantage.....	2

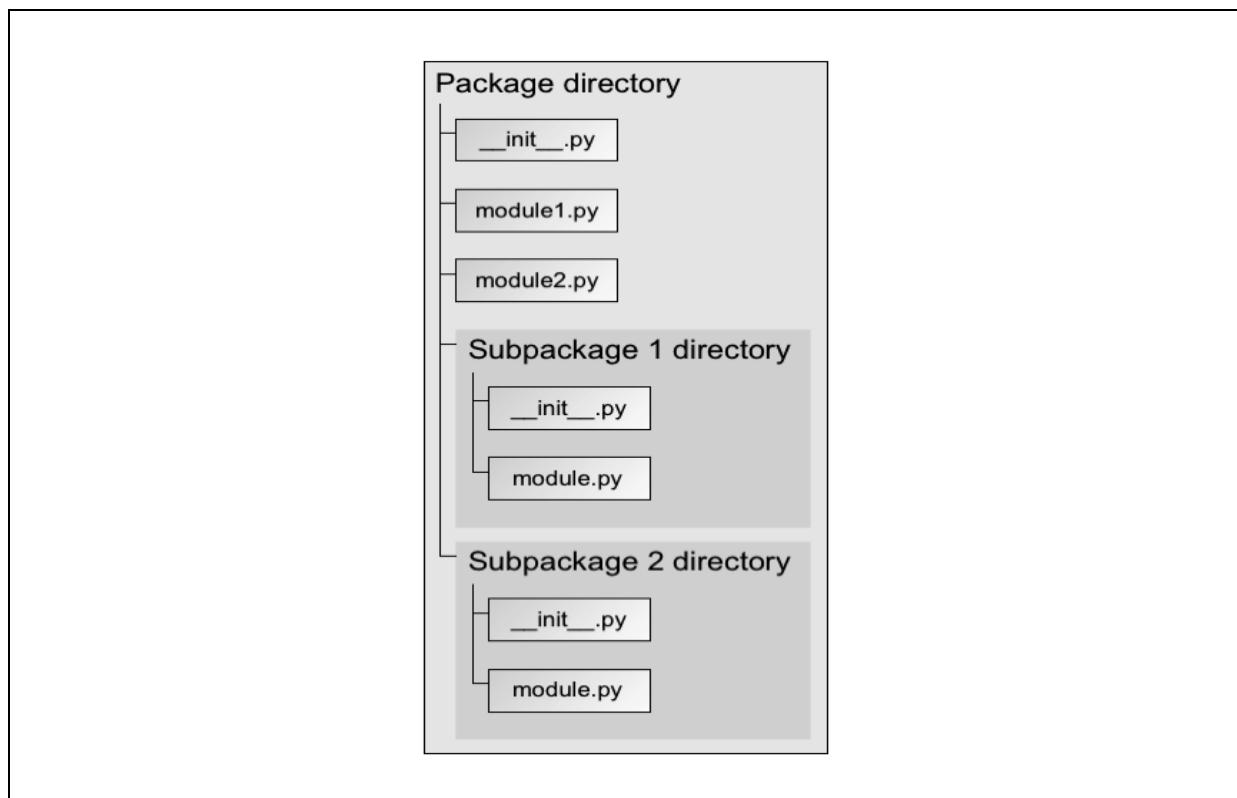
14. Python Package

1. What is a package?

- ✓ A package is nothing but folder or directory which represents collection of python modules(programs)

2. `__init__.py` file

- ✓ Any folder or directory contains `__init__.py` file, is considered as a Python package.
- ✓ `__init__.py` can be empty file.
- ✓ A package can contain sub packages also.



3. Advantage

- ✓ We can resolve naming conflicts.
- ✓ We can identify our components uniquely.
- ✓ It improves the modularity of the application.

Example1

```
|  
|---demo1.py  
|  
|---demo2.py  
|  
|---- __init__.py  
|  
|---pack1  
|  
|---- test1.py  
|  
|---- __init__.py
```

Program Creating __init__.py file
Name __init__.py

Empty file

Program executing a package
Name test1.py

```
def m1():  
    print("Hello this is test1 present in pack1")
```

Program Name executing a package
demo1.py

```
import pack1.test1  
pack1.test1.m1()
```

output
Hello this is test1 present in pack1

Program Name executing a package
demo2.py

```
from pack1.test1 import m1  
m1()
```

output
Hello this is test1 present in pack1

Example2

```
|---demo3.py  
|  
|-----__init__.py  
|  
|---maindir  
|  
|-----test2.py  
|  
|-----__init__.py  
|  
|----- subdir  
|  
|-----test3.py  
|  
|-----__init__.py
```

Program Name package __init__.py

Empty file

Program Name executing a package test2.py

```
def m2():  
    print("This is test2 present in maindir")
```

Program Name executing a package
test3.py

```
def m3():
    print("This is test3 present in maindir.subdir")
```

Program Name executing a package
demo3.py

```
from maindir.test2 import m2
from maindir.subdir.test3 import m3
```

```
m2()
m3()
```

output

```
This is test2 present in maindir
This is test3 present in maindir.subdir
```

Make a note

- ✓ Summary diagram of library, packages, modules which contains functions, classes and variables.
 - Library - A group of packages
 - Package - A group of modules
 - Modules - A group of variables functions and classes.

15. Python - List Data Structure

Table of Contents

1. Why should we learn about data structures?	2
2. Python Data structures	3
3. Sequence of elements	3
4. list data structure	4
5. Creating list	6
5.1. Creating empty list	6
5.2. Creating list with elements	6
5.3. Creating list by using list(p) predefined function.....	9
6. list having mutable nature	10
7. Accessing elements from list	11
7.1. By using index	11
7.2. Slicing	15
7.3. Accessing list by using for loop	17
8. len(p) function	18
9. Methods in list data structure	19
9.1. count(p)method	21
9.2. append(p)method.....	22
9.3. insert(p1, p2) method:.....	23
9.4. remove(p) method:.....	24
9.5. reverse():.....	25
9.6. sort() method:.....	26
10. Mathematical + and * operators	27
10.1. Concatenation operator +.....	27
10.2 Repetition operator *	27
11. Membership operators	28
12. list comprehension	29

15. Python - List Data Structure

1. Why should we learn about data structures?

- ✓ The common requirement in any real time project is like, creating, updating, retrieving, and deleting elements.
- ✓ Few more real times operations are below,
 - Storing
 - Searching
 - Retrieving
 - Deleting
 - Processing
 - Duplicate
 - Ordered
 - Unordered
 - Size
 - Capacity
 - Sorting
 - Un-sorting
 - Random access
 - Keys
 - Values
 - Key – value pairs
- ✓ So, to understand above operations where to use and how to use then we need to learn about data structures.

2. Python Data structures

- ✓ If you wanted to store a group of individual objects in a single entity, then you should go for data structures.

3. Sequence of elements

- ✓ Data structure also called as sequence.
- ✓ A sequence is a datatype that can contains a group of elements.
- ✓ The purpose of any sequence is, to store and process a group of elements.
- ✓ In python, strings, lists, tuples, set and dictionaries are very important sequence datatype.

4. list data structure

- ✓ We can **create** list by using,
 - square brackets [] symbols
 - list() predefined function.
 - ✓ A list can **store** group of objects or elements.
 - A list can store **same** (Homogeneous) type of elements.
 - A list can store **different** type (Heterogeneous) of elements.
 - ✓ A list **size** will increase dynamically.
 - ✓ In list **insertion** order is preserved or **fixed**.
 - If we insert elements into 10, 20, 30 then output also will display as 10, 20, 30 then this is called as insertion order is preserved or fixed
 - Example
 - Input => [10, 20, 30]
 - Output => [10, 20, 30]
 - ✓ **Duplicate** elements are allowed.
 - ✓ List having **mutable** nature.
 - Mutable means once we create a list object then we can change or modify the content of list object.
 - ✓ Store elements by using **index**.
 - A list data structure supports both positive and negative indexes.
 - Positive index means from left to right
 - Negative index means right to left
-

Note:

- ✓ Inside list every object can be separated by comma separator.
-

Make a note

- ✓ list is a predefined class in python.
- ✓ Once if we create list object means internally object is creating for list class.
- ✓ What is a class how class works, we will learn in OOPs chapter.

5. Creating list

- ✓ We can create list by using square brackets []
- ✓ Inside list, elements will be separated by comma separator

5.1. Creating empty list

- ✓ An empty list is valid

Program Name Creating empty list
demo1.py

```
a = []
print(a)
print(type(a))
```

Output
[]
<class 'list'>

5.2. Creating list with elements

- ✓ We can create list directly with elements.

Program Name Creating list with same type of elements
demo2.py

```
numbers = [10, 20, 30, 40]
print(numbers)
```

Output
[10, 20, 30, 40]

Program Name Creating list with same type of elements with **duplicates**
demo3.py

```
numbers = [10, 20, 30, 40, 10, 20, 30, 40]
print(numbers)
```

Output
[10, 20, 30, 40, 10, 20, 30, 40]

Program Name creating list with same type of elements
demo4.py

```
names = ["Daniel", "Prasad", "Ramesh", "Daniel"]
print(names)
```

Output
["Daniel", "Prasad", "Ramesh", "Daniel"]

Program Name Creating list with **different** type of elements
demo5.py

```
student_info = ["Daniel", 10, 35.9]
print(student_info)
```

Output
["Daniel", 10, 35.9]

Note

- ✓ Observe the above programs output,
 - Order is preserved
 - Duplicates are allowed.

5.3. Creating list by using list(p) predefined function

- ✓ list(p) is a predefined function in python.
- ✓ By using this function we can create list.
- ✓ list(p) function takes only one parameter.
- ✓ This parameter should be sequence (range, list, set, tuple, etc...) object otherwise we will get error.

Program Name	Creating list by using list(p) function demo6.py
	<pre>r = range(0, 10) a = list(r) print(a)</pre>
output	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

6. list having mutable nature

- ✓ Once we created a list object then we can change or modify the elements in the existing list object.
- ✓ So, list having mutable nature.

Program Name list having mutable nature
demo7.py

```
a = [1, 2, 3, 4, 5]
print(a)
print("Before modifying a[0] : ", a[0])

a[0] = 20
print("After modifying a[0] : ", a[0])
print(a)
```

output

```
[1, 2, 3, 4, 5]
Before modifying a[0] : 1
After modifying a[0] : 20
[20, 2, 3, 4, 5]
```

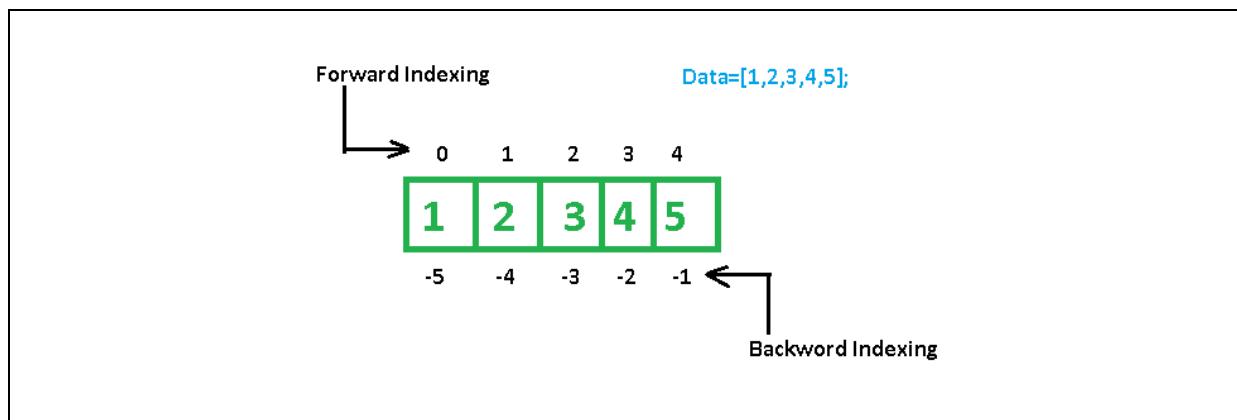
7. Accessing elements from list

- ✓ We can access elements from list by using,

1. Index.
2. Slice operator.
3. loops

7.1. By using index

- ✓ **index** represents accessing the elements by their position numbers in the list.



- ✓ **Indexing** represents accessing the elements by their position numbers in the list.
- ✓ Index starts from **0** onwards.
- ✓ List supports both positive and negative indexes.
 - Positive index represents from left to right direction
 - Negative index represents from right to left.
- ✓ If we are trying to access beyond the range of list index, then we will get error like **IndexError**.

Program Name list indexing
demo8.py

```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)

print(names[0])
print(names[1])
print(names[2])

print(type(names))
```

output

```
['Daniel', 'Prasad', 'Ramesh']
Daniel
Prasad
Ramesh
<class 'list'>
```

Program Name list indexing
demo9.py

```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)

print(names[0])
print(names[1])
print(names[2])

print(type(names))

print(type(names[0]))
print(type(names[1]))
print(type(names[2]))
```

output

```
['Daniel', 'Prasad', 'Ramesh']
Daniel
Prasad
Ramesh
<class 'list'>
<class 'str'>
<class 'str'>
<class 'str'>
```

Program Name **IndexError:** list index out of range
demo10.py

```
names = ["Daniel", "Prasad", "Ramesh"]

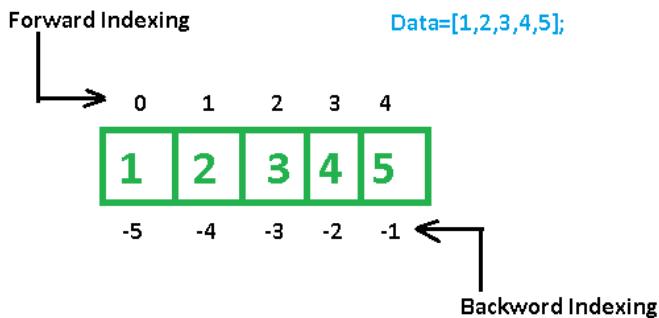
print(names)
print(names[30])
```

output

```
['Daniel', 'Prasad', 'Ramesh']
IndexError: list index out of range
```

7.2. Slicing

- ✓ Slicing represents extracting a piece of the list from already created list



Syntax

[start: stop: stepsize]

✓ start

- It indicates the index where slice can start.
- Default value is 0

✓ stop

- It indicates the index where slice can end.
- Default value is max allowed index of list i.e. length of the list

✓ Step size

- Increment value.
- Default value is 1

Program Name Slice example
demo11.py

```
n = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(n)
print(n[:])
print(n[::])
print(n[0:5:])
```

output

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5]
```

7.3. Accessing list by using for loop

- ✓ We can access elements from list by using **for** loop.

Program Name accessing elements from list by using for loop
demo12.py

```
values = [100, 200, 300, 400]
```

```
for value in values:  
    print(value)
```

output

```
100  
200  
300  
400
```

8. len(p) function

- ✓ By using len(p) predefined function we can find the length of list.
- ✓ This function returns the number of elements present in the list.

Program Name To find length of list
demo13.py

```
values = [10, 20, 30, 40, 50]  
print(len(values))
```

Output
5

9. Methods in list data structure

- ✓ As discussed, list is a predefined class.
- ✓ So, list class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(parameter1)` predefined function.
- ✓ So, internally list class contains two types of methods,
 - With underscore symbol methods.
 - We no need to focus
 - Without underscore symbol methods.
 - We need to focus much on these

Program Name Printing list data structure methods by using `dir(list)` function
`demo14.py`

```
print(dir(list))
```

output

```
[  
'__add__', ..... , '__subclasshook__',
```

Important methods

```
'append', 'count', 'insert', 'remove', 'reverse', 'sort'
```

```
]
```

Important point

- ✓ As per object-oriented principle,
 - If we want to access instance methods, then we should access by using object name.
- ✓ So, all list methods we can access by using list object.

Important methods in list

- ✓ count(p) method
- ✓ append(p) method
- ✓ insert() method
- ✓ remove() method
- ✓ reverse() method
- ✓ sort() method

9.1. count(p)method

- ✓ count(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ This method returns the number of occurrences of specific value in the list.

Program Name To find count of specific value in list
demo15.py

```
n = [1, 2, 3, 4, 5, 5, 5, 3]
print(n.count(5))
print(n.count(2))
```

output

```
3
1
```

9.2. append(p)method

- ✓ append(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ This method adds object or element to the existing list object.
- ✓ This method will add the object to list at the end of the list.

Program Name appending elements into list
demo16.py

```
a = []
a.append(10)
a.append(20)
a.append(30)
print(a)
```

output
[10, 20, 30]

Program Name appending elements into list
demo17.py

```
a = [10, 20, "Daniel"]

a.append("Naresh")
a.append("Veeru")
print(a)
```

output
[10, 20, "Daniel", "Naresh", "Veeru"]

9.3. **insert(p1, p2) method:**

- ✓ insert(p1, p2) is a predefined method in list class.
- ✓ We should access this method by using list object.
- ✓ By using this method we can insert value at specific position in list.

Program Name inserting elements into list
demo18.py

```
a = [10, 20, 30, 40, 50]
```

```
a.insert(0, 76)  
print(a)
```

output

```
[76, 10, 20, 30, 40, 50]
```

append(element)	insert(index, element)
✓ This method adds element at last position.	✓ This method adds element at specific index position.

9.4. remove(p) method:

- ✓ remove(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By using this method we can remove value from list.

Program Name Removing element from list
demo19.py

```
a = [10, 20, 30]
```

```
a.remove(10)  
print(a)
```

output

```
[20, 30]
```

Ordering elements of List:

9.5. reverse():

- ✓ reverse() is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By using this method we can reverse values in list.

Program reverse of the list

Name demo20.py

```
a = [10, 20, 30, 40]
```

```
print(a)
a.reverse()
print(a)
```

output

```
[10, 20, 30, 40]
[40, 30, 20, 10]
```

9.6. sort() method:

- ✓ sort() is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By default insertion order is fixed.
- ✓ By using this method we can sort values in list.
 - For numbers the order is ascending order.
 - For strings the order is alphabetical order

Program Name sorting the numbers and names
demo21.py

```
a = [10, 40, 50, 20, 30]
a.sort()
print(a)

b = ['Daniel', 'Ramesh', 'Arjun']
b.sort()
print(b)
```

output

```
[10, 20, 30, 40, 50]
['Arjun', 'Daniel', 'Ramesh']
```

10. Mathematical + and * operators

10.1. Concatenation operator +

- ✓ '+' operator concatenate two list objects to join them and returns single list.

Program Name + operator concatenates the lists
demo22.py

```
a = [10, 20, 30]  
b = [40, 50, 60]  
c = a + b
```

```
print(c)
```

output
[10, 20, 30, 40, 50, 60]

10.2 Repetition operator *

- ✓ '*' operator works to repetition of elements in the list.

Program Name * operator repetition the lists
demo23.py

```
a = [10, 20, 30]
```

```
print(a)  
print(a*2)
```

output
[10, 20, 30]
[10, 20, 30, 10, 20, 30]

11. Membership operators

- ✓ We can check if the element is a member of a list or not by using membership operators those are,
 - **in** operator
 - **not in** operator
- ✓ If the element is member of list, then **in** operator returns True otherwise False.
- ✓ If the element is not in the list, then **not in** operator returns True otherwise False

Program Name Membership operators
 demo24.py

```
a = [10, 20, 30, 40, 50]

print(20 in a)          # True
print(20 not in a)     # False

print(90 in a)          # False
print(90 not in a)     # True
```

output

```
True
False
False
True
```

12. list comprehension

- ✓ List comprehensions represents creating new lists from Iterable object like a list, set, tuple, dictionary and range.
- ✓ List comprehension takes input as iterable, we can apply conditional logic on every item and returns new list.
- ✓ List comprehensions code is very concise way.

Syntax

```
list = [expression for item1 in iterable1 if statement]
```

- ✓ Here Iterable represents a list, set, tuple, dictionary or range object.
- ✓ The result of list comprehension is new list based on the applying conditions.

Program list comprehension example

Name demo25.py

```
values = [10, 20, 30]
result = [value+2 for value in values]
```

```
print(values)
print(result)
```

output

```
[10, 20, 30]
[12, 22, 32]
```

Program Name list comprehension example
demo26.py

```
values = [10, 20, 30]
result = [value*3 for value in values]

print(values)
print(result)
```

output

```
[10, 20, 30]
[30, 60, 90]
```

Program Name list comprehension example
demo27.py

```
values = [10, 20, 30, 40, 50, 60, 70, 80, 90]
result = [value for value in values if value <= 50]

print(values)
print(result)
```

output

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
[10, 20, 30, 40, 50]
```

Program Name square numbers from 1 to 10 by using list comprehension
demo28.py

```
values = range(1, 11)
squares = [value*2 for value in values]
print(squares)
```

output
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

16. Python – Tuple Data Structure

Table of Contents

1. Tuple data Structure.....	2
2. When should we go for tuple data structure?	3
3. Syntax Surprise 1: Single value tuple.....	4
4. Syntax Surprise 2. Parenthesis is optional for tuple	5
5. Different ways to create a tuple	6
6. Accessing elements of tuple:	8
6.1 Index.....	8
6.2. Slice operator:.....	9
7. Tuple vs immutability:.....	10
8. Mathematical operators on tuple:	11
8.1. Concatenation operator (+):	11
8.2 Multiplication operator (*)	12
9. len(p) function	12
10. Method in tuple data structure	13
10.1. count(p) method	14
10.2. index(p) method	15
12. Differences between List and Tuple:	16
13. Can I add elements to this tuple t = (11, 22, [33, 44], 55, 66)?	17

16. Python – Tuple Data Structure

1. Tuple data Structure

- ✓ We can **create** tuple data structure by using,
 - Parenthesis () symbol.
 - Predefined tuple(p) function.
- ✓ A tuple can **store** group of objects or elements.
 - A tuple can store **same** (Homogeneous) type of elements.
 - A tuple can store **different** (Heterogeneous) type of elements.
- ✓ In tuple insertion **order** is preserved or **fixed**.
 - If we insert elements into 10, 20, 30 then output also will display as 10, 20, 30 then this is called as insertion order is preserved or fixed
 - Example
 - Input => (10, 20, 30)
 - Output => (10, 20, 30)
- ✓ **Duplicate** elements are **allowed**.
- ✓ Tuple having **immutable** nature.
 - Immutable means once we create a tuple object then we cannot change or modify the content of tuple object.
- ✓ Store elements by using **index**.
 - A tuple data structure supports both positive and negative indexes.
 - Positive index means from left to right
 - Negative index means right to left

Note:

- ✓ tuple is a predefined class in python.
- ✓ Once if we create tuple object means internally object is creating for tuple class.

Note:

- ✓ Inside tuple every object can be separated by comma separator.

2. When should we go for tuple data structure?

- ✓ If we are going to define a data which never change over all the period, then we should go for tuple data structure.

Example:

1. Week days names
2. Month names
3. Year names

Program Name Tuple having same type of objects
demo1.py

```
employee_ids = (10, 20, 30, 40, 50)
print(employee_ids)
print(type(employee_ids))
```

Output
(10, 20, 30, 40, 50)
<class 'tuple'>

3. Syntax Surprise 1: Single value tuple

- ✓ If tuple having only one object, then that object should end with comma separator otherwise python internally not considered as it is tuple.

Program Name A single value with tuple syntax, but it's not tuple
demo2.py

```
number = (9)

print(number)
print(type(number))
```

Output
(9)
<class 'int'>

Program Name A single value with tuple syntax, but it's not tuple
demo3.py

```
name = ("Daniel")

print(name)
print(type(name))
```

Output
Daniel
<class 'str'>

Program Name Tuple single value ends with comma separator then it's tuple
demo4.py

```
name = ("Daniel", )  
print(name)  
print(type(name))
```

Output

```
('Daniel')  
<class 'tuple'>
```

4. Syntax Surprise 2. Parenthesis is optional for tuple

- ✓ While creating a tuple parenthesis is optional

Program Name Parenthesis symbol is optional while creating tuple
demo5.py

```
emp_ids = 10, 20, 30, 40  
print(emp_ids)
```

output

```
(10, 20, 30, 40)
```

5. Different ways to create a tuple

1. Empty tuple

- ✓ We can create an empty tuple by using empty parenthesis.

Program Name empty tuple
demo6.py

```
emp_id = ()  
print(emp_id)  
print(type(emp_id))
```

output
(
<class 'tuple'>

2. Tuple with group of values

- ✓ Tuple can contain group of objects; those objects can be same type or different type.

Program Name Tuple example
demo7.py

```
emp_id = (11, 12, 13)  
std_id = 120, 130, 140  
print(emp_id)  
print(std_id)
```

output

```
(11, 12, 13)  
(120, 130, 140)
```

Program Name Tuple example
demo8.py

```
t = (11, 12, 13, "daniel")
print(t)
```

output

```
(11, 12, 13, "daniel")
```

3. By using tuple(p) function

- ✓ We can create tuple by using tuple(p) function.

Program Name Creating tuple by using tuple function
demo9.py

```
a = [11, 22, 33]
t = tuple(a)
print(t)
```

output

```
(11, 22, 33)
```

6. Accessing elements of tuple:

- ✓ We can access tuple elements by using,
 - Index
 - Slice operator

6.1 Index

- ✓ Index means position where element stores

Program Name Accessing tuple by using index
demo10.py

```
t = (10, 20, 30, 40, 50, 60)
```

```
print(t[0])      #    10
print(t[-1])     #    60
```

Output

```
10
60
```

6.2. Slice operator:

- ✓ A group of objects from starting point to ending point

Program Name Accessing tuple by using slice
demo11.py

```
t = (10, 20, 30, 40, 50, 60)
```

```
print(t[2:5])
print(t[2:100])
print(t[::-2])
```

Output

```
30, 40, 50)
(30, 40, 50, 60)
(10, 30, 50)
```

7. Tuple vs immutability:

- ✓ Tuple having immutable nature.
- ✓ If we create a tuple then we cannot modify the elements of existing tuple.

Program Name	Prove tuple having immutable nature demo12.py
	<pre>t = (10, 20, 30, 40) print(t[1]) t[1] = 70</pre>
Output	20 TypeError: 'tuple' object does not support item assignment

8. Mathematical operators on tuple:

- ✓ We can apply plus (+) and Multiplication (*) operators on tuple.
- ✓ + Operator works as concatenation.
- ✓ * Operator works as multiplication.

8.1. Concatenation operator (+):

- ✓ + operator concatenates two tuples and returns single tuple

Program Name Concatenation operator on tuple
demo13.py

```
t1 = (10,20,30)  
t2 = (40,50,60)  
t3 = t1 + t2
```

```
print(t3)
```

Output

```
(10, 20, 30, 40, 50, 60)
```

8.2 Multiplication operator (*)

- ✓ Multiplication operator works as repetition operator

Program Name Repetition operator on tuple
demo14.py

```
t1 = (10,20,30)
t2 = t1*3
print(t2)
```

Output
(10, 20, 30, 10, 20, 30, 10, 20, 30)

9. len(p) function

- ✓ To return number of elements present in the tuple

Program Name len(p) function
demo15.py

```
t = (10,20,30,40)
print(len(t))
```

Output

4

10. Method in tuple data structure

- ✓ As discussed, tuple is a predefined class.
- ✓ So, tuple class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(p)` predefined function.

- ✓ So, internally tuple class contains two types of methods,
 - With underscore symbol methods.
 - We no need to focus
 - Without underscore symbol methods.
 - We need to focus much on these

Program Name Printing tuple data structure methods by using `dir(p)` function
`demo16.py`

```
print(dir(tuple))
```

output

```
[  
    '__add__', ...., '__subclasshook__',  
  
    'count', 'index'  
]
```

Important point

- ✓ As per object-oriented principle,
 - If we want to access instance method then we should access by using object name.
- ✓ So, all tuple methods we can access by using tuple object.

Methods in tuple

1. count(parameter1) method
2. index(parameter1) method

10.1. count(p) method

- ✓ count(p) is a method, we should access this method by using tuple object.
- ✓ This method returns the number of occurrences of specified item in the tuple

Program Name	count(p) method demo17.py
	<pre>t = (10, 20, 10, 10, 20) print(t.count(10))</pre>
output	3

10.2. index(p) method

- ✓ returns index of first occurrence of the given element.
- ✓ If the specified element is not available, then we will get **ValueError**.

Program Name index(p) method
demo18.py

```
t = (10, 20, 30)  
print(t.index(30))
```

Output
2

Program Name index(p) method
demo19.py

```
t = (10, 20, 30)  
print(t.index(88))
```

Output
ValueError: tuple.index(x): x not in tuple

12. Differences between List and Tuple:

- ✓ List and Tuple are exactly same except small difference:
 - List objects are mutable
 - Tuple objects are immutable.
- ✓ In both cases,
 - Insertion order is preserved.
 - Duplicate objects are allowed
 - Heterogeneous objects are allowed
 - Index and slicing are supported.

List	Tuple
<ul style="list-style-type: none"> ✓ List is a Group of Comma separated Values within Square Brackets and Square Brackets are mandatory. ✓ Example: <code>i = [10, 20, 30, 40]</code> 	<ul style="list-style-type: none"> ✓ Tuple is a Group of Comma separated Values within Parenthesis and Parenthesis are optional. ✓ Example: <code>t = (10, 20, 30, 40)</code> ✓ Example: <code>t = 10, 20, 30, 40</code>
<ul style="list-style-type: none"> ✓ List Objects are Mutable i.e. once we create List object we can perform any changes in that Object. ✓ Example: <code>i[1] = 70</code> 	<ul style="list-style-type: none"> ✓ Tuple Objects are Immutable i.e. once we create Tuple object we cannot change its content. ✓ Example: <code>t[1] = 70</code> ✓ TypeError: tuple object does not support item assignment.
<ul style="list-style-type: none"> ✓ If the Content is not fixed and keep on changing, then we should go for List. 	<ul style="list-style-type: none"> ✓ If the content is fixed and never changes then we should go for Tuple.

13. Can I add elements to this tuple t = (11, 22, [33, 44], 55, 66)?

- ✓ Yes we can add elements to list in tuple.
- ✓ In second index position list is available, to that we can add

Program Name tuple data structure can store any data
demo23.py

```
t = (11, 22, [33, 44], 55, 66)
```

```
t[2].append(77)  
print(t)
```

output
(11, 22, [33, 44, 77], 55, 66)

17. PYTHON - SET DATA STRUCTURE

Table of Contents

1. Set data structure	2
2. When should we go for set?	3
3. Creating set by using same type of elements.....	3
4. Creating set by using different type of elements.....	4
5. Creating set by range(p) type of elements.....	4
6. Creating set by using set(p) predefined function.....	5
7. Empty set	6
8. len(p) function.....	7
9. Methods in set data structure	8
9.1. add(p) method:	9
9.2. remove(p) method	10
9.3. clear() method	10
10. Membership operators: (in and not in)	11
11. set comprehensions	12
12. Remove duplicates in list elements	12

17. PYTHON - SET DATA STRUCTURE

1. Set data structure

- ✓ We can **create** set by using,
 - Curly braces {} symbols
 - set() predefined function.
- ✓ A set can **store group** of objects or elements.
 - A set can store **same** (Homogeneous) type of elements.
 - A set can store **different** (Heterogeneous)type of elements.
- ✓ A set size will **increase** dynamically.
- ✓ In set insertion order is not preserved means **not fixed**.
 - If we insert elements into 10, 20, 30, 40 then there is no guarantee for output as 10, 20, 30, 40
 - Example
 - Input => {10, 20, 30, 40}
 - Output => {20, 40, 10, 30}
- ✓ Duplicate elements are **not allowed**.
- ✓ Set having **mutable** nature.
 - Mutable means once we create a set object then we can change or modify the content of set object.
- ✓ Set data structure **cannot** store the elements in index order.

Note:

- ✓ set is a predefined class in python.
- ✓ Once if we create set object means internally object is creating for set class.

Note:

- ✓ Inside set every object can be separated by comma separator.

2. When should we go for set?

- ✓ If we want to represent a group of **unique** values as a single entity, then we should go for set.
- ✓ set cannot store duplicate elements.
- ✓ Insertion order is not preserved.

Note

- ✓ We can create set by using curly braces {} and all elements separated by comma separator in set.

3. Creating set by using same type of elements

Program Name creating same type of elements by using set
demo1.py

```
s = {10, 20, 30, 40}  
print(s)  
print(type(s))
```

Output

```
{40, 10, 20, 30}  
<class 'set'>
```

4. Creating set by using different type of elements

Program Name creating different type of elements by using set
demo2.py

```
s = {10, "Daniel", 30.9, "Prasad", 40}  
print(s)  
print(type(s))
```

Output
{40, 10, 'Daniel', 'Prasad', 30.9}
<class 'set'>

5. Creating set by range(p) type of elements

Program Name creating set by using range(p)
demo3.py

```
s = set(range(5))  
print(s)
```

output
{0, 1, 2, 3, 4}

Program Name set not allowed duplicates
demo4.py

```
s ={10, 20, 30, 40, 10, 10}  
print(s)  
print(type(s))
```

Output
{40, 10, 20, 30}
<class 'set'>

Make a note

- ✓ Observe the above programs output,
 - Order is not preserved
 - Duplicates are not allowed.

6. Creating set by using set(p) predefined function

- ✓ We can create set by using set(p) function.
- ✓ set(p) predefined function will take only one parameter.
- ✓ This parameter should be sequence (range, list, set, tuple, etc...) object otherwise we will get error.

Program Name creating set by using set(parameter1) function
demo5.py

```
r = range(0, 10)  
l = set(r)  
print(l)  
output  
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

7. Empty set

- ✓ We can create empty set.
- ✓ While creating empty set compulsory we should use **set()** function.
- ✓ If we didn't use set function, then it treats as dictionary instead of set.

Program Name demo6.py

Empty curly braces is not a set, it considers as a dictionary

```
s = {}  
print(s)  
print(type(s))
```

Output

```
{}  
<class 'dict'>
```

Program Name demo7.py

Using **set()** function to create empty set

```
s = set()  
print(s)  
print(type(s))
```

Output

```
set()  
<class 'set'>
```

8. len(p) function

- ✓ By using len(p) predefined function we can find the length of set.
- ✓ This function returns the number of elements present in the set.

Program Name To find length of set
demo8.py

```
n = {10, 20, 30, 40, 50}  
print(len(n))
```

Output
5

Program Name To find length of set
demo9.py

```
n = {10, 20, 30, 40, 50, 10, 10, 10}  
print(len(n))
```

Output
5

9. Methods in set data structure

- ✓ As discussed, set is a predefined class.
- ✓ So, set class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(parameter1)` predefined function.

- ✓ So, internally set class contains two types of methods,
 - With underscore symbol methods.
 - We no need to focus
 - Without underscore symbol methods.
 - We need to focus much on these

Program Name Printing set data structure methods by using `dir(set)` function
`demo10.py`

```
print(dir(set))
```

output

```
[  
    '__and__', ..... '__subclasshook__', '__xor__',
```

Important methods

```
'add', 'clear', 'remove',
```

```
]
```

Important point

- ✓ As per object-oriented principle,
 - If we want to access instance method, then we should access by using object name.
- ✓ So, all set methods we can access by using set object.

Important methods in set:

- ✓ add(p) method
- ✓ remove(p) method
- ✓ clear() method

9.1. add(p) method:

- ✓ add(p) is a method, we should access this method by using set object.
- ✓ This method adds element to the set.

Program Name add(p) function can add element to the set
demo11.py

```
s = {10, 20, 30}  
s.add(40)  
print(s)
```

Output
{40, 10, 20, 30}

9.2. remove(p) method

- ✓ remove(p) is a method in set class, we should access this method by using set object.
- ✓ This method removes specified element from the set.
- ✓ If the specified element not present in the set, then we will get **KeyError**

Program Name remove(p) method in set
demo12.py

```
s = {40, 10, 30, 20}  
s.remove(30)  
print(s)
```

Output
{40, 10, 20}

9.3. clear() method

- ✓ clear() is a method in set class, we should access this method by using set object.
- ✓ This method removes all elements from the set.

Program Name clear() method in set
demo13.py

```
s = {10,20,30}  
print(s)  
s.clear()  
print(s)
```

Output
{10, 20, 30}
set()

10. Membership operators: (in and not in)

- ✓ By using these operators, we can find the specified element is exists in set or not

Program in operator
Name demo14.py

```
s = {1, 2, 3, 'daniel'}
```

```
print(s)
print(1 in s)
print('z' in s)
```

Output

```
{1, 2, 3, 'daniel'}
True
False
```

11. set comprehensions

- ✓ set comprehensions represents creating new set from Iterable object like a list, set, tuple, dictionary and range.
- ✓ set comprehensions code is very concise way.

Program Name set comprehension
demo15.py

```
s = {x*x for x in range(5)}  
print(s)
```

output
{0, 1, 4, 9, 16}

12. Remove duplicates in list elements

- ✓ We can remove duplicates elements which are exists in list by passing list as parameter to set function

Program Name removing duplicates from list
demo18.py

```
a = [10, 20, 30, 10, 20, 40]  
s = set(a)  
print(s)
```

Output
{40, 10, 20, 30}

18. PYTHON - DICTIONARY DATA STRUCTURE

Table of Contents

1. Dictionary data structure	2
2. When should we go for dictionary?.....	3
3. Creating dictionary	4
4. Empty dictionary	4
5. We can add key-value pairs to empty dictionary.....	5
6. Access values by using keys from dictionary.....	6
7. Update dictionary.....	7
7.1.Case 1.....	7
7.2.Case 2	7
8. Removing or deleting elements from dictionary	9
9. len(p) function	11
10. Methods in dict class data structure	12
10.1. clear() method	14
10.2. keys() method	15
10.3. values().....	15
10.4. items()	16
11. Dictionary Comprehension:	17

18. PYTHON - DICTIONARY DATA STRUCTURE

1. Dictionary data structure

- ✓ If we want to represent group of **individual objects** as a single entity then we should go for below data structures,
 - list
 - set
 - tuple
- ✓ If we want to represent a group of objects as **key-value pairs** then we should go for,
 - dict or dictionary
- ✓ We can create dict by using,
 - **Curly braces {} symbols**
 - **dict()** predefined function.
- ✓ Dictionary data structure contains **key, value pairs**.
- ✓ **key-value**
 - In dictionary key, value pairs are **separated** by colon : symbol
 - One key-value pair is called as **item**.
 - In dictionary every item is separated by **comma symbol**.
 - In dictionary **duplicate keys** are **not allowed**.
 - In dictionary **duplicate values** can be **allowed**.
 - A dictionary keys and values can store **same** (Homogeneous) type of elements.
 - A dictionary keys and values can store **different** (Heterogeneous) type of elements.
- ✓ In dictionary insertion order is not preserved means **not fixed**.
- ✓ Dictionary **size** will **increase** dynamically.
- ✓ Dictionary object having **mutable** nature.
- ✓ Dictionary data structure **cannot** store the elements in **index** order.
 - Indexing and slicing concepts are not applicable

Note:

- ✓ dict is a predefined class in python.
- ✓ Once if we create dictionary object means internally object is creating for dict class.

2. When should we go for dictionary?

- ✓ If we want to represent a group of objects as **key-value** pairs then we should go for,
 - dict or dictionary

Symbols to create data structure for specific data structure?

- ✓ To create **list**, we need to use **square bracket symbols** : []
- ✓ To create **tuple**, we need to use **parenthesis symbols** : ()
- ✓ To create **set** we need to use **curly braces** with values : {}
- ✓ So, to create **dict** we need to use **curly braces** : {}

Create dictionary

Syntax

```
d = { key1 : value1, key2 : value2 }
```

3. Creating dictionary

- ✓ We can create dictionary with key, value pairs.

Program Name

creating dictionary

Name demo1.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print(d)
```

output

```
{10: "Ramesh", 20: "Arjun", 30: "Daniel"}
```

4. Empty dictionary

- ✓ We can create empty dictionary.

Program Name

creating empty dictionary

Name demo2.py

```
d = {}  
print(d)  
print(type(d))
```

output

```
{}  
<class 'dict'>
```

5. We can add key-value pairs to empty dictionary

- ✓ As we know we can create an empty dictionary.
- ✓ For that empty dictionary we can add key, value pairs.

Program Name creating empty dictionary and adding elements
demo3.py

```
d = []
d[10] = "Ramesh"
d[20] = "Arjun"
d[30] = "Daniel"
print(d)
```

output
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}

6. Access values by using keys from dictionary

- ✓ We can access dictionary values by using keys
- ✓ Keys play main role to access the data.

Program Name Accessing dictionary values by using keys
demo4.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
  
print(d[10])  
print(d[20])  
print(d[30])
```

output
Ramesh
Arjun
Daniel

Program Name Accessing key and value from dictionary using for loop
demo5.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
  
for k in d:  
    print(k, d[k])
```

output
10 Ramesh
20 Arjun
30 Daniel

7. Update dictionary

- ✓ We can update the key in dictionary.

Syntax

```
d[key] = value
```

7.1.Case 1

- ✓ While updating the key in dictionary, if key is not available then a new key will be added at the end of the dictionary with specified value.

7.2.Case 2

- ✓ While updating the key in dictionary, if key is already existing then old value will be replaced with new value.

Program Name Case 1: Adding key-value pair to dictionary
demo6.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print("Old dictionary: ",d)  
  
d[99] = "John"  
print("Added key-value 99:John pair to dictionary: ", d)
```

output
Old dictionary: {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}

Added key-value 99:John pair to dictionary:
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel', 99: 'John'}

Program Name Case 2: Updating key-value pair in dictionary
demo7.py

```
d = {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
print("Old dictionary:", d)
```

```
d[30] = 'Chandhu'  
print("Updated dictionary 3:Chandhu :", d)
```

output

```
Old dictionary: {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
Updated dictionary 3:Chandhu : {10: 'Ramesh', 20: 'Arjun', 30:  
'Chandhu'}
```

8. Removing or deleting elements from dictionary

- ✓ By using `del` keyword, we can remove the keys
- ✓ By using `clear()` method we can clear the objects in dictionary

8.1. By using `del` keyword

Syntax

```
del d[key]
```

- ✓ As per the syntax, it deletes entry associated with the specified key.
- ✓ If the key is not available, then we will get `KeyError`

Program Name Deleting key in dictionary
demo8.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print("Before deleting key from dictionary: ", d)  
del d[10]  
print("After deleting key from dictionary: ", d)
```

output
Before deleting key from dictionary:
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}

After deleting key from dictionary:
{20: 'Arjun', 30: 'Daniel'}

We can delete total dictionary object

Syntax

```
del nameofthedictionary
```

- ✓ It can delete the total dictionary object.
- ✓ Once it deletes then we cannot access the dictionary.

Program Delete key in dictionary

Name demo9.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print(d)
```

```
del d  
print(d)
```

output

```
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
NameError: name 'd' is not defined
```

9. len(p) function

- ✓ This function returns the number of items in the dictionary

Program Finding length of dictionary

Name demo10.py

```
d = {100: "Ramesh", 200: "Arjun"}  
print("length of dictionary is: ",len(d))
```

output

```
length of dictionary is:2
```

10. Methods in dict class data structure

- ✓ As discussed dict is a predefined class.
- ✓ So, dict class can contain methods because methods can be created inside of class only.
- ✓ We can check these method by using `dir(parameter1)` predefined function.

- ✓ So, internally dict class contains two types of methods,
 - With underscore symbol methods.
 - We no need to focus
 - Without underscore symbol methods.
 - We need to focus much on these

Program Name Printing dict data structure methods by using `dir(dict)` function
`demo11.py`

```
print(dir(dict))
```

output

```
[
```

```
'__class__', .....'__subclasshook__',
```

Important methods

```
'clear', 'items', 'keys', 'values'
```

```
]
```

Important point

- ✓ As per object-oriented principle,
 - If we want to access instance methods, then we should access by using object name.
- ✓ So, all dict methods we can access by using dict object.

Important methods

- ✓ clear() method
- ✓ keys() method
- ✓ values() method
- ✓ items() method

10.1. clear() method

- ✓ clear() is a method in dict class, we should access this method by using dictionary object.
- ✓ This method removes all entries in dictionary.
- ✓ After deleting all entries, it just keeps empty dictionary

Program Name removing dictionary object by using clear() method
demo12.py

```
d = {10: "Ramesh", 20: "Arjun", 30:"Daniel"}  
print(d)  
d.clear()  
print("After cleared entries in dictionary: ", d)
```

output
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}
After cleared entries in dictionary: {}

10.2. keys() method

- ✓ keys() is a method in dict class, we should access this method by using dict object.
- ✓ This method returns all keys associated with dictionary

Program Name keys() method
demo13.py

```
d = {100: "Ramesh", 200: "Daniel", 300: "Mohan"}  
print(d)  
print(d.keys())
```

Output

```
{100: 'Ramesh', 200: 'Daniel', 300: 'Mohan'}  
dict_keys([100, 200, 300])
```

10.3. values()

- ✓ values() is a method in dict class, we should access this method by using dict object.
- ✓ This method returns all values associated with the dictionary

Program Name values() method
demo14.py

```
d = {100: "Ramesh", 200: "Daniel", 300: "Mohan"}  
print(d)  
print(d.values())
```

Output

```
{100: 'Ramesh', 200: 'Daniel', 300: 'Mohan'}  
dict_values(['Ramesh', 'Daniel', 'Mohan'])
```

10.4. items()

- ✓ items() is a method in dict class, we should access this method by using dict object.
- ✓ By using this method we can access keys and values from the dictionary.

Program Name Accessing key and value from dictionary using items() method
demo15.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}
```

```
for k, v in d.items():
    print(k, v)
```

output

```
10 Ramesh
20 Arjun
30 Daniel
```

11. Dictionary Comprehension:

- ✓ A dictionary comprehension represents creating new dictionary from Iterable object like a list, set, tuple, dictionary and range.
- ✓ Dictionary comprehensions code is very concise way.

Program Name Dictionary comprehension
 demo15.py

```
squares = {a: a*a for a in range(1,6)}
print(squares)
```

Output
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

19. PYTHON – OBJECT ORIENTED PROGRAMMING

Table of Contents

1. Is Python follows Functional approach or Object-oriented approach?	2
2. Features of Object-Oriented Programming System	2
3. class:	3
3.1. Def1:.....	3
3.2. Def2:.....	3
4. How to define or create a class?	4
5. Brief discussion about class	4
6. object	7
6.1. Why should we create an object?.....	7
6.2. What is an object?	7
6.3. Syntax to create an object	8
7. Constructor	10
7.1. What is the main purpose of constructor?.....	10
7.2. When constructor will be executed?	11
7.3. How many times Constructor will executes?.....	11
7.4. Types of constructors.....	12
7.5. Constructor without parameters	12
7.6. Creating parameterised constructor.....	13
8. Difference between method and constructor	17
9. Instance variables:	18
9.1. What is instance variable?	18
9.2. Separate copy instance variable for every object.....	18
9.3. Declaring & accessing instance variables.....	19
10. Instance methods	20
11. self pre-defined variable	21

19. PYTHON – OBJECT ORIENTED PROGRAMMING

- ✓ Object-Oriented Programming is a methodology to design software by using classes and objects.
- ✓ It simplifies the software development and maintenance by providing the below features,

1. Is Python follows Functional approach or Object-oriented approach?

- ✓ Python supports both functional and object-oriented programming.

2. Features of Object-Oriented Programming System

- ✓ class
- ✓ object
- ✓ constructor
- ✓ Inheritance & etc...

3. class:

3.1. Def1:

- ✓ A class is a model for creating an object and it does not exist physically.

3.2. Def2:

- ✓ A class is a specification (idea/plan/theory) of properties and actions of objects.

Syntax

```
class NameOfTheClass:  
    1. constructor  
    2. properties (attributes)  
    3. actions (behaviour)
```

- ✓ We can create class by using **class** keyword.
- ✓ class can contain,

- constructor
- properties
- actions

- ✓ Properties also called as variables.
- ✓ Actions also called as methods.

4. How to define or create a class?

- ✓ A python class may contain the below things,

Syntax

```
class NameOfTheClass:  
    """ documentation string ""  
  
    1. Constructor  
  
    2. Variables  
        1. instance variables  
  
    3. Methods  
        1. instance methods
```

5. Brief discussion about class

- ✓ We can create class by using **class** keyword.
- ✓ class keyword follows the **name of the class**.
- ✓ After name of the class we should give **colon :** symbol.
- ✓ After : colon symbol in next line we should provide the **indentation**, otherwise we will get error.
- ✓ class can contain,
 - **Constructor** are used for initialization purpose
 - **Variables** are used to represent the data.
 - **instance** variables
 - **Methods** are used to represent actions.
 - **instance** methods

Class naming convention

- ✓ While writing a class we need to follow the naming convention to meet real time standards,
 - class names should start with upper case and remaining letters are in lower case.
 - **Example:** Student
 - If name having multiple words, then every inner word should start with upper case letter.
 - **Example:** StudentInfo

Note

- ✓ Documentation string represents description of the class. Within the class doc string is always optional.

Program Name	Define a class demo1.py
<pre>class Employee: def display(self): print("Hello My name is Daniel")</pre>	
output	

Make a note

- ✓ In above program, when we run then we will not get any output because we didn't call display method
- ✓ Above program Employee represents a **class** which is defined by developer.
- ✓ Developer defined only one method as display(self)
- ✓ Method we can define by using **def** keyword.
- ✓ Methods means it's just like a functions to perform an operations

Kind info:

- ✓ Writing a class is not enough; we should know how to use the variables and methods.

So,

- ✓ We need to create an object to access instance data(variables/methods) of a class.

6. object

6.1. Why should we create an object?

- ✓ As per requirement we used to define variables and methods in a class.
- ✓ These variables and methods hold the data or values.
- ✓ When we create an object for a class, then only data will be stored for the data members of a class.

6.2. What is an object?

Definition 1:

- ✓ Instance of a class is known as an object.
 - Instance is a mechanism to allocate enough memory space for data members of a class.

Definition 2:

- ✓ Grouped item is known as an object.
 - Grouped item is a variable which stores more than one value.

Definition 3:

- ✓ Real world entities are called as objects.

Make some notes

- ✓ An object exists physically in this world, but class does not exist.

6.3. Syntax to create an object

Syntax

```
nameoftheobject = nameoftheclass()
```

Example

```
emp = Employee()
```

Program

Name

Creating a class and object

demo2.py

```
class Employee:  
    def display(self):  
        print("Hello my name is Daniel")
```

```
emp = Employee()  
emp.display()
```

output

Hello my name is Daniel

Program Name Creating a class and object
demo2.py

```
class Employee:  
    def display(self):  
        print("Hello my name is Daniel")  
  
    def teaching(self):  
        print("I like teaching")  
  
emp = Employee()  
  
emp.display()  
emp.teaching()
```

output
Hello my name is Daniel
I like teaching

Make a note

- ✓ We can create object for class.
- ✓ In the above example **emp** is object name.
 - emp is just like a variable
- ✓ above example, **display(self)** is instance method.
 - To access instance method, we should create an object
 - So, we are accessing instance methods by using object name

7. Constructor

- ✓ Constructor is a special kind of method in python.
- ✓ So, we can create constructor by using **def** keyword
- ✓ The name of the constructor should be **__init__(self)**
 - Two underscore symbols before and after init with self as parameter
- ✓ self should be first parameter in constructor,

Syntax

```
class NameOfTheClass:  
    def __init__(self):  
        body of the constructor
```

7.1. What is the main purpose of constructor?

- ✓ The main purpose of constructor is to initialize instance variables.

7.2. When constructor will be executed?

- ✓ Constructor will be executed automatically at the time of object creation.

Program Name Creating a constructor
demo3.py

```
class Employee:  
    def __init__(self):  
        print("constructor is executed")  
  
emp = Employee()
```

output
constructor is executed

7.3. How many times Constructor will executes?

- ✓ If we create object in two times then constructor will execute two times.

Program Name Creating a constructor
demo3.py

```
class Employee:  
    def __init__(self):  
        print("constructor is executed")  
  
emp1 = Employee()  
emp2 = Employee()
```

output
constructor is executed
constructor is executed

7.4. Types of constructors

- ✓ Based on parameters constructors can be divided into two types,
 1. Constructor without parameters
 2. Constructor with parameters

7.5. Constructor without parameters

- ✓ If constructor having no parameters, then at least it should contain `self` as one parameter.

Syntax

```
class NameOfTheClass:  
    def __init__(self):  
        body of the constructor
```

Program Name Creating a constructor
demo3.py

```
class Employee:  
    def __init__(self):  
        print("constructor is executed")  
  
emp = Employee()
```

output
constructor is executed

7.6. Parameterised constructor

- ✓ Based on requirement constructor can contain any number of parameters.

7.6. Creating parameterised constructor

- ✓ By default, first parameter should be self to constructor.
- ✓ Constructor can contain more parameters along with self
- ✓ If constructor having more parameters, then the first parameter should be self and remaining parameters will be next.

Syntax

```
class NameOfTheClass:  
    def __init__(self, parameter1, parameter2):  
        body of the constructor
```

Note: One parameterised constructor

Program Name One parameterised constructor
demo6.py

```
class Employee:  
    def __init__(self, number):  
        self.number= number  
        print("Employee id is: ", self.number)  
  
e1 = Employee(1)  
e2 = Employee(2)  
e3 = Employee(3)
```

output
Employee id is: 1
Employee id is: 2
Employee id is: 3

Note: One parameterised constructor

- ✓ If constructor having one parameter, then during object creation we need to pass one value.

Can i write a constructor and an instance method in a single program?

- ✓ Yes we can write constructor and instance method both in single program.
- ✓ Here constructor purpose is to initialize instance variables, and method purpose is to perform operations.

Two parameterised constructor

Program Name One parameterised constructor and instance method
demo7.py

```
class Employee:  
    def __init__(self, number):  
        self.number = number  
  
    def display(self):  
        print("Employee id is:", self.number)  
  
e1 = Employee(1)  
e2 = Employee(2)  
e3 = Employee(3)  
  
e1.display()  
e2.display()  
e3.display()
```

output

```
Employee id is: 1  
Employee id is: 2  
Employee id is: 3
```

Note: Access instance variable in instance method

- ✓ Inside instance method we can access instance variables by using self.

Two parameterised constructor

Program Name Two parameterised constructor and instance method
demo8.py

```
class Employee:  
    def __init__(self, number, name):  
        self.number = number  
        self.name = name  
  
    def display(self):  
        print("Hello my id is :", self.number)  
        print("My name is :", self.name)
```

```
e1=Employee(1, 'Daniel')  
e1.display()
```

```
e2=Employee(2, 'Arjun')  
e2.display()
```

Output

```
Hello my id is: 1  
My name is: Daniel
```

```
Hello my id is: 2  
My name is: Arjun
```

Note: Two parameterised constructor

- ✓ If constructor having two parameters, then during object creation we need to pass two values

Three parameterised constructor

Program Name Three parameterised constructor and instance method demo9.py

```
class Employee:  
    def __init__(self, number, name, age):  
        self.number = number  
        self.name = name  
        self.age = age  
  
    def display(self):  
        print("Hello my id is :", self.number)  
        print("My name is :", self.name)  
        print("My age is sweet :", self.age)  
  
e1=Employee(1, 'Daniel', 16)  
e1.display()  
  
e2=Employee(2, 'Arjun', 17)  
e2.display()  
  
e3=Employee(3, 'Prasad', 18)  
e3.display()
```

Output

Hello my id is: 1
My name is: Daniel
My age is sweet: 16

Hello my id is: 2
My name is: Arjun
My age is sweet: 17

Hello my id is :3
My name is: Prasad
My age is sweet: 18

Note: Three parameterised constructor

- ✓ If constructor having three parameters, then during object creation we need to pass three values.

8. Difference between method and constructor

Method	Constructor
✓ Methods are used to perform operations or actions	✓ Constructors are used to initialize the instance variables.
✓ Method name can be any name.	✓ Constructor name should be <code>__init__(self)</code>
✓ Methods we should call explicitly to execute	✓ Constructor automatically executed at the time of object creation.

9. Instance variables:

9.1. What is instance variable?

- ✓ If the value of a variable is changing from object to object such type of variables is called as instance variables.

9.2. Separate copy instance variable for every object

- ✓ For every object a separate copy of instance variables will be created.

Program Instance variables
Name demo10.py

```
class Student:  
    def __init__(self, name, number):  
        self.name=name  
        self.number=number  
  
    s1 = Student('Daniel', 101)  
    s2 = Student('Prasad', 102)  
  
    print("Studen1 info:")  
    print("Name: ", s1.name)  
    print("Id : ", s1.number)  
  
    print("Studen2 info:")  
    print("Name: ", s2.name)  
    print("Id : ", s2.number)
```

Output

Studen1 info:
Name: Daniel
Id: 101

Studen2 info:
Name: Prasad
Id: 102

9.3. Declaring & accessing instance variables

- ✓ We can declare instance variables inside constructor
- ✓ We can access instance variables by using object name

Program Name Initializing instance variables inside Constructor
demo11.py

```
class Employee:  
    def __init__(self):  
        self.eno = 10  
        self.ename = "Daniel"  
        self.esal = 10000  
  
    emp = Employee()  
  
    print("Employee number:", emp.eno)  
    print("Employee name:", emp.ename)  
    print("Employee salary:", emp.esal)
```

output
Employee number: 10
Employee name : Daniel
Employee salary : 10000

10. Instance methods

- ✓ Instance methods are methods which act upon the instance variables of the class.
- ✓ Instance methods are bound with instances or objects, that's why called as instance methods.
- ✓ The first parameter for instance methods is `self` variable.
- ✓ Along with `self` variable it can contains other variables as well.

Program Name	Instance methods demo13.py
<pre>class Demo: def __init__(self, a): self.a=a def m(self): print(self.a) d=Demo(10) d.m()</pre>	

Output

10

11. self pre-defined variable

- ✓ self is a predefined variable in python, this variable belongs to current class object.
 - self variable we can use to create below things,
 - Constructor
 - Instance variable
 - Instance methods
- ✓ Constructor
 - By using self, we can initialize the instance variables inside constructor `__init__(self)`
- ✓ Instance variable
 - By using self, we can declare and access instance variables,
- ✓ Instance methods
 - By using self, we can create instance methods.