PROGRAMMING IN PYTHON

IT-605 [Lab]

0103IT201111
NAVEEN SINGH

Information Technology

IT 605 [Programming in Python]

1. Python –Overview

   Introduction, History, Features

2. Python –Environment Setup

   Local Environment Setup, Getting Python, Installation of Python, Use of IDE

3. Python –Basic Syntax

   Python Identifiers, Reserved Words, Lines & Indentation, Multiline Statements, Quotation in Python, Comments & other useful constructs

4. Python –Variables

   Assigning Values to Variables, Multiple Assignment, Standard Data Types

5. Python Numbers

   Python Strings, Python Lists, Python Tuples, Dictionary, DataType Conversion

6. Python –Basic Operators

   Types of Operators- Arithmetic Operators, Assignment Operators, Comparison Operators, Logical Operators, Bitwise Operators; Operator Precedence.

7. Python –Decision Making & Loops

   Flowchart, If statement Syntax

8. Python-Functions

   Syntax for defining a function, Calling a Function, Function Arguments, Anonymous Functions

   Python-Applications & Further Extensions

1.Python –Overview

Introduction, History, Features

1.1 Introduction

What is Python?

- Python is a widely-used general-purpose, high-level programming language. Or it is a popular programming language.
- It was created by Guido van Rossum, and released in 1991.

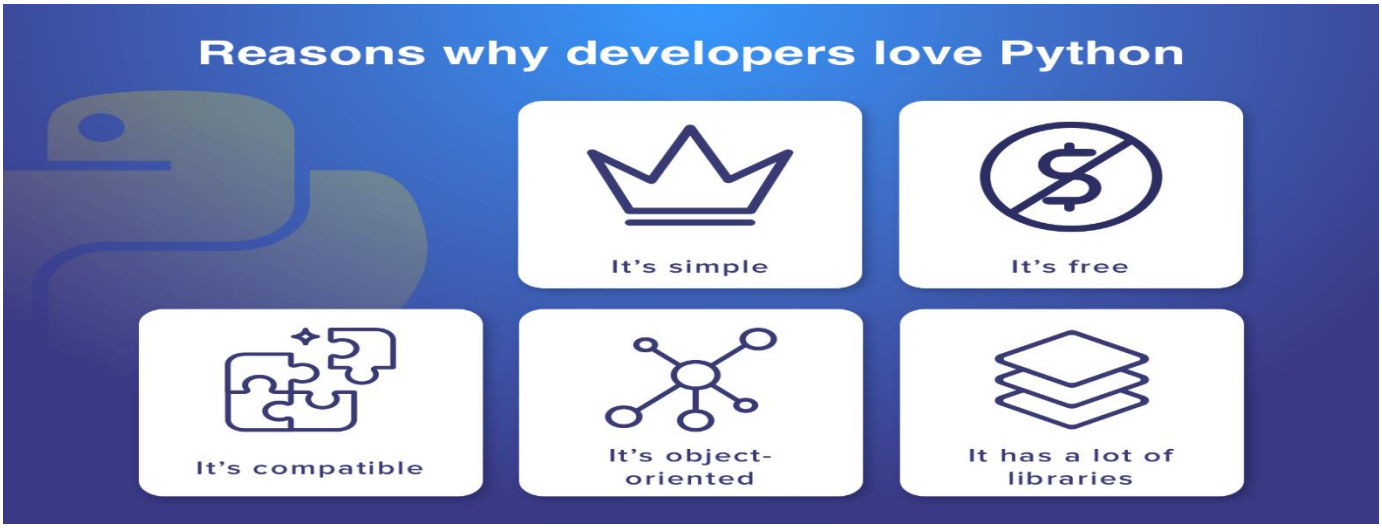
It is used for:

- web development (server-side),
- software development,
- handle big data and perform complex mathematics
- system scripting.
- Python can be used for rapid prototyping, or for production-ready software development.

NOTE:-

- *The most recent major version of Python is Python 3. However, Python 2 is still quite popular.*
- *In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.*

Why/Advantage of Python?

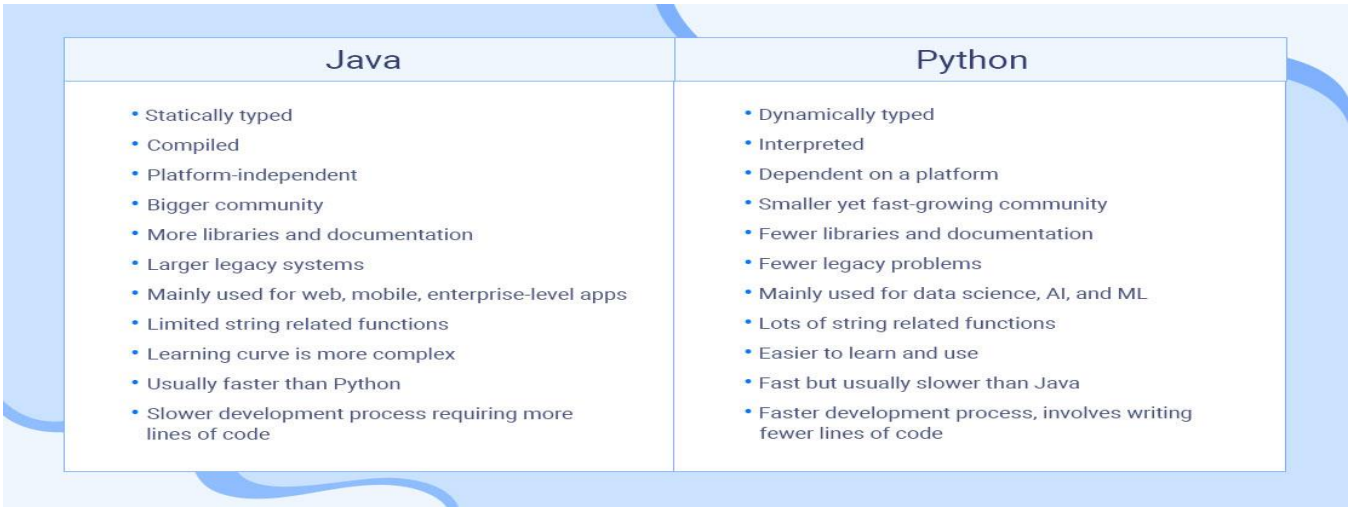


- **Simplicity**- we can say that python is a minimalistic language. It is very easy to write and read. / Python has a simple syntax similar to the English language. Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- **It's free & Open source**- Python is free and open source. It means that the developer don't have to pay for anything. They can share , copy, and change it.
- **Compatibility**- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- **Object-oriented-** Python supports procedure-oriented as well as object-oriented programming.

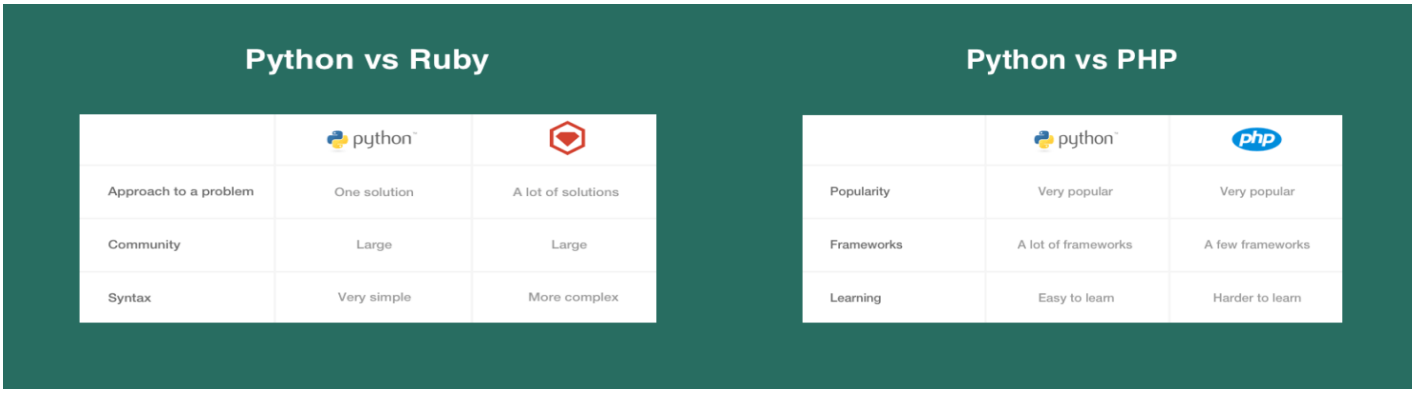
- **Libraries-** Python community has created a huge pile of various libraries for Python.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.



S.NO	PYTHON	C++
1	Python is typically an "interpreted" language	C++ is typically a "compiled" language
2	Python is a dynamic-typed language	C++ is compiled statically typed language
3	Data type is not required while declaring variable	Data type is required while declaring variable
4	It can act both as scripting and general purpose language	It is a general purpose language



1.2 History

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, **Guido Van Rossum** published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
- Python 2.0 added new features such as list comprehensions, garbage collection systems.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.
- *ABC programming language* is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
- The following programming languages influence Python:
    - ABC language.
    - Modula-3

Why the Name Python?

There is a fact behind choosing the name Python. **Guido van Rossum** was reading the script of a popular BBC comedy series "**Monty Python's Flying Circus**". It was late on-air 1970s.

Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the **"Monty Python's Flying Circus"** for their newly created programming language.

The comedy series was creative and well random. It talks about everything. Thus it is slow and unpredictable, which made it very interesting.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.8	October 14, 2019

1.3 Features

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don't need to declare the type of variable because it is a dynamically typed languag**Features in Python**

There are many features in Python, some of which are discussed below as follows:

1. Free and Open Source

Python language is freely available It means that the developer don't have to pay for anything. It is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

2. Easy to code

Python is very easy to learn the language as compared to other languages .It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

As you will see, learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. Large Standard Library

Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Portable

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

8. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called **bytecode**.

9. Extensible feature

Python is an **Extensible** language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

10. Python is an Integrated language

Python is also an Integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. Dynamically Typed Language

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable.

12. Allocating Memory Dynamically

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

2.Python –Environment Setup

Local Environment Setup, Getting Python, Installation of Python, Use of IDE

2.1 Local Environment Setup

Open a terminal window and type "python" to find out if it is already installed and which version is installed. If Python is already installed then you will get a message something like as follows:

```
$ python
Python 3.6.8 (default, Sep 10 2021, 09:13:53)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2.2 Getting Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

You can download Python documentation from https://www.python.org/doc/. The documentation is available in HTML, PDF, and PostScript formats.

2.3 Installation of Python

Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.

If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in terms of choice of features that you require in your installation.

Here is a quick overview of installing Python on various platforms –

Unix and Linux Installation

Here are the simple steps to install Python on Unix/Linux machine.

- Open a Web browser and go to https://www.python.org/downloads/.
- Follow the link to download zipped source code available for Unix/Linux.
- Download and extract files.
- Editing the *Modules/Setup* file if you want to customize some options.

Now issue the following commands:

```
$ run ./configure script
$ make
$ make install
```

This installs Python at standard location */usr/local/bin* and its libraries at */usr/local/lib/pythonXX* where XX is the version of Python.

Windows Installation

Here are the steps to install Python on Windows machine.

- Open a Web browser and go to https://www.python.org/downloads/.

- Follow the link for the Windows installer *python-XYZ.msi* file where XYZ is the version you need to install.
- To use this installer *python-XYZ.msi*, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
- Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

Setting path at Unix/Linux

To add the Python directory to the path for a particular session in Unix −

- **In the csh shell** − type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.
- **In the bash shell (Linux)** − type export PATH="$PATH:/usr/local/bin/python" and press Enter.
- **In the sh or ksh shell** − type PATH="$PATH:/usr/local/bin/python" and press Enter.
- **Note** − /usr/local/bin/python is the path of the Python directory

Setting path at Windows

To add the Python directory to the path for a particular session in Windows −

At the command prompt − type path %path%;C:\Python and press Enter.

Note − C:\Python is the path of the Python directory

2.4 IDE

An integrated development environment (IDE) is a software application that helps programmers develop software code efficiently.

It combines common developer tools into a single graphical user interface (GUI). An IDE typically consists of:

- **Source code editor**: A text editor that can assist in writing software code with features such as syntax highlighting with visual cues, providing language specific auto-completion, and checking for bugs as code is being written.
- **Local build automation**: Utilities that automate simple, repeatable tasks as part of creating a local build of the software for use by the developer, like compiling computer source code into binary code, packaging binary code, and running automated tests.
- **Debugger**: A program for testing other programs that can graphically display the location of a bug in the original code.

Use of IDE:-

Code editing automation-Programming languages have rules for how statements must be structured. Because an IDE knows these rules, it contains many intelligent features for automatically writing or editing the source code.

Syntax highlighting-An IDE can format the written text by automatically making some words bold or italic, or by using different font colors. These visual cues make the source code more readable and give instant feedback about accidental syntax errors.

Intelligent code completion-Various search terms show up when you start typing words in a search engine. Similarly, an IDE can make suggestions to complete a code statement when the developer begins typing.

Refactoring support-Code refactoring is the process of restructuring the source code to make it more efficient and readable without changing its core functionality. IDEs can auto-refactor to some extent, allowing developers to improve their code quickly and easily. Other team members understand readable code faster, which supports collaboration within the team.

Local build automation-IDEs increase programmer productivity by performing repeatable development tasks that are typically part of every code change. The following are some examples of regular coding tasks that an IDE carries out.

Compilation-An IDE compiles or converts the code into a simplified language that the operating system can understand. Some programming languages implement just-in-time compiling, in which the IDE converts human-readable code into machine code from within the application.

Testing- The IDE allows developers to automate unit tests locally before the software is integrated with other developers' code and more complex integration tests are run.

Debugging-Debugging is the process of fixing any errors or bugs that testing reveals.

Your first Python Program

Now that we have Python up and running, we can write our first Python program.

Let's create a very simple program called `Hello World`. A **"Hello, World!"** is a simple program that outputs `Hello, World!` on the screen. Since it's a very simple program, it's often used to introduce a new programming language to beginners.

Type the following code in any text editor or an IDE and save it as `hello_world.py`

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

then, run the file. You will get the following output.

```
Hello, world!
```

Congratulations! You just wrote your first program in Python.

3-Python –Basic Syntax

Python Identifiers, Reserved Words, Lines & Indentation, Multiline Statements, Quotation in Python, Comments & other useful constructs

3.1 Python Identifiers

Identifier is a name used to identify a variable, function, class, module, etc.

Rules for Naming an Identifier

- Identifiers cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter or _. The first letter of an identifier cannot be a digit.
- It's a convention to start an identifier with a letter rather _.
- Whitespaces are not allowed.
- We cannot use special symbols like **!**, @, #, **$**, and so on.

Some Valid and Invalid Identifiers in Python

Valid Identifiers	Invalid Identifiers
score	@core
return_value	return
name1 var1 _var1 _1_var var_1	1name !var1 1var 1_var var#1
convert_to_string	convert to_string

3.2 Keyword/ Reserved word

Keywords are some predefined and reserved words in python that have special meanings.

- Keywords are used to define the syntax of the coding.
- The keyword cannot be used as an identifier, function, and variable name.
- All the keywords in python are written in lower case except True and False.

There are 33 keywords in Python 3.7 let's go through all of them one by one:-

Keywords	Description
and	This is a logical operator it returns true if both the operands are true else return false.
Or	This is also a logical operator it returns true if anyone operand is true else return false.
not	This is again a logical operator it returns True if the operand is false else return false.
if	This is used to make a conditional statement.
elif	Elif is a condition statement used with an if statement the elif statement is executed if the previous conditions were not true
else	Else is used with if and elif conditional statement the else block is executed if the given condition is not true.
for	This is created for a loop.
while	This keyword is used to create a while loop.
break	This is used to terminate the loop.
as	This is used to create an alternative.
def	It helps us to define functions.
lambda	It is used to define the anonymous function.
pass	This is a null statement which means it will do nothing.
return	It will return a value and exit the function.
True	This is a boolean value.
False	This is also a boolean value.
try	It makes a try-except statement.
with	The with keyword is used to simplify exception handling.
assert	This function is used for debugging purposes. Usually used to check the correctness of code

Keywords	Description
class	It helps us to define a class.
continue	It continues to the next iteration of a loop
del	It deletes a reference to an object.
except	Used with exceptions, what to do when an exception occurs
finally	Finally is use with exceptions, a block of code that will be executed no matter if there is an exception or not.
from	The form is used to import specific parts of any module.
global	This declares a global variable.
import	This is used to import a module.
in	It’s used to check if a value is present in a list, tuple, etc, or not.
is	This is used to check if the two variables are equal or not.
None	This is a special constant used to denote a null value or avoid. It’s important to remember, 0, any empty container(e.g empty list) do not compute to None
nonlocal	It’s declared a non-local variable.
raise	This raises an exception
yield	It’s ends a function and returns a generator.

3.3 Lines & Indentation

We can print a string in a new line in 3 ways in Python:

1. Multiple print statements
2. Using '\n.'
3. Using multi-line strings.

These three ways might be useful for different needs, but programmers mostly use **'\n'** to print a new line because it is **the most commonly accepted method** due to its simplicity.

Using '\n,' we can:

1. Print a string in multiple lines.
2. Keep the code short and simple.
3. Customize the positions of the characters of a string.
4. Leave a blank line.

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.

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

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

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

Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

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

Example

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

3.4 Multi-line Statement in Python:

In Python, the statements are usually written in a single line and the last character of these lines is newline. To extend the statement to one or more lines we can use braces {}, parentheses (), square [], semi-colon ";", and continuation character slash "\". we can use any of these according to our requirement in the code. With the line continuation character, we can explicitly divide a long statement into numerous lines (\).

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

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

Code:
```
# Breaks the lines using continuation character

g = "geeks\
for\
geeks"
print(g)
```
Output: `geeksforgeeks`

Line continuation are divided into two different ways:

- Explicit line continuation- In this type of multi-line statement, we will be using the line continuation character (\) to split a statement into multiple lines.
- Implicit line continuation- In this type of multi-line statement, Implicit line continuation is used when you split a statement using either parentheses ( ), brackets [ ], and braces { }.

3.5 Quotation in Python

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

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

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

3.6 Python Comment

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Example

```
#This is a comment
print("Hello, World!")

print("Hello, World!") #This is a comment
```

Multi Line Comments

Python does not really have a syntax for multi line comments.

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

Example

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

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

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

Useful constructs

Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

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

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

Note: cd command us

Python Basic Input and Output

➢ Python Output

- In Python, we can simply use the `print()` function to print output. For example,

```
print('Python is powerful')

# print with end whitespace or print() with end Parameter
print('Good Morning!', end= ' ')
print('It is rainy today')     // Good Morning! It is rainy today
# Python print() with separated parameter
print('New Year', 2023, 'See you soon!', sep= '. ')
# Print Concatenated Strings
print('Programiz is ' + 'awesome.')    //Programiz is awesome.
```

➢ Python Input

- In Python, we can use the `input()` function.
  example,

```
# using input() to take user input
num = input('Enter a number: ')         // Enter a number: 10
print('You Entered:', num)              // You Entered: 10
print('Data type of num:', type(num))   // Data type of num: <class 'str'>
```

To convert user input into a number we can use `int()` or `float()` functions as:

```
num = int(input('Enter a number: '))
```

4-Variables

Assigning Values to Variables, Multiple Assignment, Standard Data Types

4.1 Assigning Values to Variables

Variables- Variables are containers for storing data values.

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

```
x = 5
y = "John"
print(x)
print(y)
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

Get the Type

You can get the data type of a variable with the `type()` function.

Example

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

Example

```
x = "John"
# is the same as
x = 'John'
```

Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

4.2 Multiple Assignment

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Note: Make sure the number of variables matches the number of values, or else you will get an error.

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

4.3 Standard Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

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

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

Print the data type of the variable x:

```
x = 5
print(type(x))
```

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int

`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool

`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

5-Python Numbers & Data Type

Python Strings, Python Lists, Python Tuples, Dictionary, DataType Conversion

Python Numbers

- The number data types are used to store the numeric values.
- Python supports integers, floating-point numbers and complex numbers. They are defined as `int`, `float`, and `complex` classes in Python.

- `int` - holds signed integers of non-limited length.
- `float` - holds floating decimal points and it's accurate up to **15** decimal places.
- `complex` - holds complex numbers.

Let's see an example,

```
num1 = 5
print(num1, 'is of type', type(num1))
num2 = 5.42
print(num2, 'is of type', type(num2))
num3 = 8+2j
print(num3, 'is of type', type(num3))
```

String

- A **string** is a sequence of characters that can be a combination of letters, numbers, and special characters. Or
- **Strings** are arrays of bytes representing Unicode characters.
- It can be declared in python by using single quotes, double quotes, or even triple quotes.
- Strings are **immutable**, i.e., they cannot be changed.
- Each element of the string can be accessed using indexing or slicing operations.
- To concatenate, or combine, two strings you can use the + operator.

NOTE- Python does not have a character data type, a single character is simply a string with a length of 1.

Example: a = 'This is a string'

print (a) //This is a string

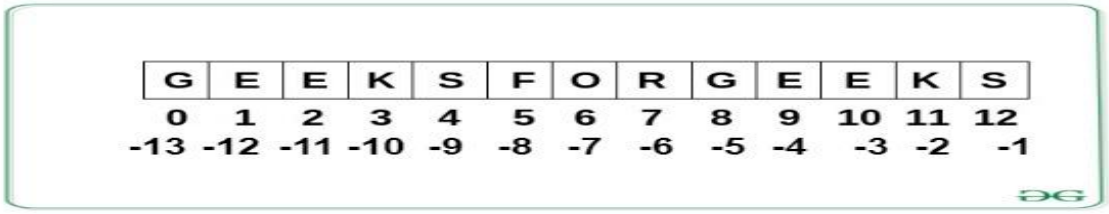
Accessing characters in Python String

1. Indexing.
2. Slicing

Indexing.

- In Python, individual characters of a String can be accessed by using the method of Indexing.

- Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.
- While accessing an index out of the range will cause an **IndexError**.
- Only Integers are allowed to be passed as an index, float or other types that will cause a **TypeError**.




```
Example:-
String1 = "GeeksForGeeks"
# Printing First character
print(String1[0])
# Printing Last character
print(String1[-1])
```

Slicing

- To access a range of characters in the String, the method of slicing is used.
- Slicing in a String is done by using a Slicing operator (colon).
- Example:-

```
String1 = "GeeksForGeeks"
# Printing 3rd to 12th character
print(String1[3:12])
```

Check String

- To check if a certain phrase or character is present in a string, we can use the keyword in. Ex:-
```
txt = "The best things in life are free!"
print("free" in txt)
      //or
if "free" in txt:
print("Yes, 'free' is present.")
```

Check if NOT

- To check if a certain phrase or character is NOT present in a string, we can use the keyword not in. Ex:-
```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

String Methods

- Python has a set of built-in methods that you can use on strings.
- All string methods return new values. They do not change the original string.

Method	*Description*	
upper()	*Converts a string into upper case*	`a = "Hello, World!"` `print(a.upper())`

strip()	*Returns a trimmed version of the string*

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

capitalize()	*Converts the first character to upper case*
casefold()	*Converts string into lower case*
center()	*Returns a centered string*
count()	*Returns the number of times a specified value occurs in a string*
encode()	*Returns an encoded version of the string*
endswith()	*Returns true if the string ends with the specified value*
find()	*Searches the string for a specified value and returns the position of where it was found*
format()	*Formats specified values in a string*
index()	*Searches the string for a specified value and returns the position of where it was found*
isalnum()	*Returns True if all characters in the string are alphanumeric*
isalpha()	*Returns True if all characters in the string are in the alphabet*
isdecimal()	*Returns True if all characters in the string are decimals*
isdigit()	*Returns True if all characters in the string are digits*
isidentifier()	*Returns True if the string is an identifier*
islower()	*Returns True if all characters in the string are lower case*
isnumeric()	*Returns True if all characters in the string are numeric*
isspace()	*Returns True if all characters in the string are whitespaces*
isupper()	*Returns True if all characters in the string are upper case*
join()	*Joins the elements of an iterable to the end of the string*
lower()	*Converts a string into lower case*
lstrip()	*Returns a left trim version of the string*
maketrans()	*Returns a translation table to be used in translations*
partition()	*Returns a tuple where the string is parted into three parts*
replace()	*Returns a string where a specified value is replaced with a specified value*
rjust()	*Returns a right justified version of the string*
rpartition()	*Returns a tuple where the string is parted into three parts*
rsplit()	*Splits the string at the specified separator, and returns a list*
rstrip()	*Returns a right trim version of the string*
split()	*Splits the string at the specified separator, and returns a list*
splitlines()	*Splits the string at line breaks and returns a list*
startswith()	*Returns true if the string starts with the specified value*
swapcase()	*Swaps cases, lower case becomes upper case and vice versa*
title()	*Converts the first character of each word to upper case*
translate()	*Returns a translated string*

Python Collections (Arrays) Datatype

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

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

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

1. List

- A list is a collection of things, enclosed in [ ] and separated by commas.
- Lists are used to store multiple items in a single variable.
- Lists are sequenced data type *which is used to store the collection of data. Tuples and String are other types of sequence data types.*
- **Python Lists** are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java).
- Lists need not be homogeneous always which makes it the most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects
- Lists are one of the most powerful data structures in python.
- In Python, an empty list is created using list() function. But the most powerful thing is that list need not be always homogeneous.
- A single list can contain strings, integers, as well as other objects.
- Lists can also be used for implementing stacks and queues.
- Lists are **mutable**, i.e., they can be altered once declared.
- The elements of list can be accessed using indexing and slicing operations
- *Complexities for Creating Lists*
  Time Complexity: O(1)
  Space Complexity: O(n)

Creating a list- A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

```
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]          # Having duplicate values

List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks'] # mixed type of valuesHaving no.and strings
```

Accessing elements-

1.Indexing-In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. (`print(List[0])`)

2.Slicing- In Python List, there are multiple ways to print the whole list with all the elements, but to print a specific range of elements from the list, we use the Slice operation. (`Sliced_List = List[3:8]`)

- Slice operation is performed on Lists with the use of a colon(:).
- To print elements from beginning to a range use**:** *[: Index]*
- To print elements from end-use: *[:-Index]*
- To print elements from a specific Index till the end use *[Index:]*
- To print the whole list in reverse order, use *[::-1]*

  Note – To print elements of List from rear-end, use Negative Indexes

- Python len() is used to get the length of the list:- `print(len(List))`

```
# Declaring a list
L = [1, "a" , "string" , 1+2]
print L                          // [1, 'a', 'string', 3]
#Adding an element in the list
L.append(6)                      //[1, 'a', 'string', 3, 6]
print L
#Deleting last element from a list
L.pop()                          //[1, 'a', 'string', 3]
print L
#Displaying Second element of the list
print L[1]                       //A
```

List Methods

```
Method     Description
append()   Adds an element at the end of the list
clear()    Removes all the elements from the list
copy()     Returns a copy of the list
count()    Returns the number of elements with the specified value
extend()   Add the elements of a list (or any iterable), to the end
of the current list
index()    Returns the index of the first element with the
specified value
insert()   Adds an element at the specified position
pop()Removes the element at the specified position
remove()   Removes the item with the specified value
reverse()  Reverses the order of the list
sort()     Sorts the list
```

2. Tuples

- A tuple is a sequence of **immutable** Python objects.
- Lists are used to store multiple items in a single variable.
- Tuple items are ordered(order will not change), unchangeable, and allow duplicate values.
- Tuples are just like lists with the exception that tuples cannot be changed once declared.
- The sequence of values stored in a tuple can be of any type, and they are indexed by integers.
- Tuples are usually faster than lists.

Creating a Tuple:- In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping the data sequence.

- Complexities for creating tuples:
  Time complexity: O(1)
  Auxiliary Space : O(n)

Note: *Creation of Python tuple without the use of parentheses is known as Tuple Packing.*

```
# Creating an empty Tuple
Tuple1 = ()
print(Tuple1)         // ()

# Creating a Tuple the use of string
Tuple1 = ('Geeks', 'For')
print(Tuple1)         // ('Geeks', 'For')

# Creating a Tuple with  use of list
list1 = [1, 2, 4, 5, 6]
print(tuple(list1))   // (1, 2, 4, 5, 6)

# Creating a Tuple Mixed Datatype
Tuple1 = (5, 'Welcome', 7, 'Geeks')
print(Tuple1)          // (5, 'Welcome', 7, 'Geeks')
```

Accessing a Tuple:- **Tuples** are immutable, and usually, they contain a sequence of heterogeneous elements that are accessed via **unpacking** or **indexing** and **Slicing.**

- Complexities for accessing elements in tuples:
  Time complexity: O(1)
  Space complexity: O(1)
- ***Note:*** *In unpacking of tuple number of variables on the left-hand side should be equal to a number of values in given tuple a.*

```
# Accessing Tuple with Indexing
Tuple1 = tuple("Geeks")
print(Tuple1[0])          //G

# Tuple unpacking
Tuple1 = ("Geeks", "For", "Geeks")

# This line unpack values of Tuple1
a, b, c = Tuple1
print(a)                  // Geeks
print(b)                  // For
print(c)                  // Geeks
```

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Tuples VS Lists:

Similarities	**Differences**
Functions that can be used for both lists and tuples: len(), max(), min(), sum(), any(), all(), sorted()	Methods that cannot be used for tuples: append(), insert(), remove(), pop(), clear(), sort(), reverse()
Methods that can be used for both lists and tuples: count(), Index()	we generally use ‘tuples’ for heterogeneous (different) data types and ‘lists’ for homogeneous (similar) data types.
Tuples can be stored in lists.	Iterating through a ‘tuple’ is faster than in a ‘list’.
Lists can be stored in tuples.	‘Lists’ are mutable whereas ‘tuples’ are immutable.
Both ‘tuples’ and ‘lists’ can be nested.	Tuples that contain immutable elements can be used as a key for a dictionary.

3. Python Dictionary

- **Dictionary in Python** is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.
- Dictionary holds **key:value** pair. Key-Value is provided in the dictionary to make it more optimized. Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
  print(Dict) // {1: 'Geeks', 2: 'For', 3: 'Geeks'}

Creating a Dictionary

- In Python, a dictionary can be created by placing a sequence of elements within curly **{}** braces, separated by 'comma'.
- Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its **Key:value**.
- Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be *immutable*.
  Note – *Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.*
- *Time complexity: O(len(dict))*
- *Space complexity: O(n)*

```
# Creating a Dictionary with Integer Keys
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(Dict) // {1: 'Geeks', 2: 'For', 3: 'Geeks'}
# Creating a Dictionary with Mixed keys
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print(Dict)  // {'Name': 'Geeks', 1: [1, 2, 3, 4]}
# Creating an empty Dictionary
Dict = {}
print(Dict)  // {}
# Creating a Dictionary with dict() method
Dict = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})
print(Dict)   // {1: 'Geeks', 2: 'For', 3: 'Geeks'}
# Creating a Dictionary with each item as a Pair
Dict = dict([(1, 'Geeks'), (2, 'For')])
print(Dict)    // {1: 'Geeks', 2: 'For'}

# Creating a Nested Dictionary as shown in the below image
Dict = {1: 'Geeks', 2: 'For',
        3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
print(Dict)    //{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
```

Accessing elements of a Dictionary

- In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets.
- There is also a method called **get()** that will also help in accessing the element from a dictionary.This method accepts key as argument and returns the value.

- **Time complexity:** O(1)
- **Space complexity:** O(1)

```
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using key
print(Dict['name'])           //For
# accessing a element using key
print(Dict[1])                 //Geeks
```

Dictionary methods

- **clear() –** Remove all the elements from the dictionary
- **copy() –** Returns a copy of the dictionary
- **get() –** Returns the value of specified key
- **items() –** Returns a list containing a tuple for each key value pair
- **keys() –** Returns a list containing dictionary's keys
- **pop() –** Remove the element with specified key
- **popitem() –** Removes the last inserted key-value pair
- **update() –** Updates dictionary with specified key-value pairs
- **values() –** Returns a list of all the values of dictionary
-

```
# demo for all dictionary methods
dict1 = {1: "Python", 2: "Java", 3: "Ruby", 4: "Scala"}

# copy() method
dict2 = dict1.copy()
print(dict2)       //{1: 'Python', 2: 'Java', 3: 'Ruby', 4: 'Scala'}

# clear() method
dict1.clear()
print(dict1)      //{}

# get() method
print(dict2.get(1))  //Python

# items() method
print(dict2.items())  //dict_items([(1, 'Python'), (2, 'Java'), (3, 'Ruby'), (4,
'Scala')])

# keys() method
print(dict2.keys())     //dict_keys([1, 2, 3, 4])

# pop() method
dict2.pop(4)
print(dict2)             //{1: 'Python', 2: 'Java', 3: 'Ruby'}

# popitem() method
dict2.popitem()
print(dict2)           //{1: 'Python', 2: 'Java'}

# update() method
dict2.update({3: "Scala"})
print(dict2)            //{1: 'Python', 2: 'Java', 3: 'Scala'}
# values() method
print(dict2.values())          //dict_values(['Python', 'Java', 'Scala']
```

4. Python Set

- Sets are used to store multiple items in a single variable.
- Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.
- A set is a collection which is *unordered*, *unchangeable**, o not allow duplicate values, and *unindexed*.

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

- Sets are written with curly brackets. `set2 = {1, 5, 7, 9, 3}`
- A set can contain different data types. `set1 = {"abc", 34, True, 40, "male"}`

Access Items

- You cannot access items in a set by referring to an index or a key.
- But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

Add Items

- Once a set is created, you cannot change its items, but you can add new items.
- To add one item to a set use the `add()` method. `thisset.add("orange")`

Remove Item

- To remove an item in a set, use the `remove()`, or the `discard()` method. `thisset.discard("banana")`
- You can also use the `pop()` method to remove an item, but this method will remove the *last* item.
  Note: If the item to remove does not exist, `remove()` will raise an error.

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set

discard() Remove the specified item
intersection() Returns a set, that is the intersection of two other sets
intersection_update() Removes the items in this set that are not present in other, specified set(s)
isdisjoint() Returns whether two sets have a intersection or not
issubset() Returns whether another set contains this set or not
issuperset() Returns whether this set contains another set or not
pop() Removes an element from the set
remove() Removes the specified element
symmetric_difference() Returns a set with the symmetric differences of two sets
symmetric_difference_update() inserts the symmetric differences from this set and another
union() Return a set containing the union of sets
`set3 = set1.union(set2)`
update() Update the set with the union of this set and others

Data type Conversion

Key Points to Remember

1. Type Conversion is the conversion of an object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of objects are converted using predefined functions by the user.
5. In Type Casting, loss of data may occur as we enforce the object to a specific data type.

There are two types of Type Conversion in Python:

1. **Implicit Type Conversion**:- the Python interpreter automatically converts one data type to another without any user involvement.
2. **Explicit Type Conversion**:- the data type is manually changed by the user as per their requirement. With explicit type conversion, there is a risk of data loss since we are forcing an expression to be changed in some specific data type.

Function	Converting what to what	Example

`int()`	string, floating point → integer	`>>> int('2014')`
		`2014`
		`>>> int(3.141592)`
		`3`

`float()`	string, integer → floating point number	`>>> float('1.99')`
		`1.99`
		`>>> float(5)`
		`5.0`

`str()`	integer, float, list, tuple, dictionary → string	`>>> str(3.141592)`
		`'3.141592'`
		`>>> str([1,2,3,4])`
		`'[1, 2, 3, 4]'`

`list()`	string, tuple, set, dictionary → list	`>>> list('Mary')  # list of characters in 'Mary'`
		`['M', 'a', 'r', 'y']`
		`>>> list((1,2,3,4))  # (1,2,3,4) is a tuple`
		`[1, 2, 3, 4]`
		`>>> list({1, 2, 3})  # {1, 2, 3} is a set`
		`[1, 2, 3]`

`tuple()`	string, list, set → tuple	`>>> tuple('Mary')`
		`('M', 'a', 'r', 'y')`
		`>>> tuple([1,2,3,4])  # [ ] for list, ( ) for tuple`
		`(1, 2, 3, 4)`

`set()`	string, list, tuple → set	`>>> set('alabama' # unique character set from a string`
		`{'b', 'm', 'l', 'a'}`
		`>>> set([1, 2, 3, 3, 3, 2 # handy for removing duplicates`
		`{1, 2, 3}`

6- Python –Basic Operators

Types of Operators- Arithmetic Operators, Assignment Operators, Comparison Operators,

Logical Operators, Bitwise Operators; Operator Precedence.

- **Operators** are used to perform operations on variables and values.
- Example- print(10 + 5); we use the + operator to add together two values

Python divides the operators in the following groups:

- ✓ Arithmetic operators
- ✓ Assignment operators
- ✓ Comparison operators
- ✓ Logical operators
- ✓ Bitwise operators
- ○ Identity operators
- ○ Membership operators

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	*Name*	*Example*
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	vision	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators

Assignment operators are used to assign values to variables:

Operator	*Example*	*Same As*
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Comparison operators are used to compare two values:

Operator	*Name*	*Example*
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Logical operators are used to combine conditional statements:

Operator	*Description*	*Example*
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	*Name*	*Description*
&	AND	*Sets each bit to 1 if both bits are 1*

	OR	*Sets each bit to 1 if one of two bits is 1*
^	XOR	*Sets each bit to 1 if only one of two bits is 1*
~	NOT	*Inverts all the bits*
<<	Zero fill left shift	*Shift left by pushing zeros in from the right and let the leftmost bits fall off*
>>	Signed right shift	*Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off*

Operator Precedence.

The operator precedence in Python is listed in the following table. It is in descending order (upper group has higher precedence than the lower ones).

Operators	Meaning
`()`	Parentheses
`**`	Exponent
`+x`, `-x`, `~x`	Unary plus, Unary minus, Bitwise NOT
`*`, `/`, `//`, `%`	Multiplication, Division, Floor division, Modulus
`+`, `-`	Addition, Subtraction
`<<`, `>>`	Bitwise shift operators
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==`, `!=`, `>`, `>=`, `<`, `<=`, `is`, `is not`, `in`, `not in`	Comparisons, Identity, Membership operators
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

7-Python –Decision Making & Loops

Flowchart, If statement Syntax

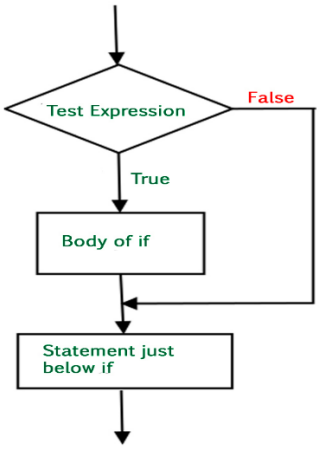
- Decisions in a program are used *when the program has conditional choices* to execute a code block.
- It is the prediction of conditions that occur while executing a program to specify actions.
- Multiple expressions get evaluated with an outcome of either TRUE or FALSE.

Python provides various types of conditional statements:

Statement	Description
if Statements	It consists of a Boolean expression which results are either TRUE or FALSE, followed by one or more statements.
if else Statements	It also contains a Boolean expression. The if the statement is followed by an optional else statement & if the expression results in FALSE, then else statement gets executed. It is also called alternative execution in which there are two possibilities of the condition determined in which any one of them will get executed.
Nested Statements	We can implement if statement and or if-else statement inside another if or if -else statement. Here more than one if conditions are applied & there can be more than one if within elif.

if Statements

The decision-making structures can be recognized and understood using flowcharts.



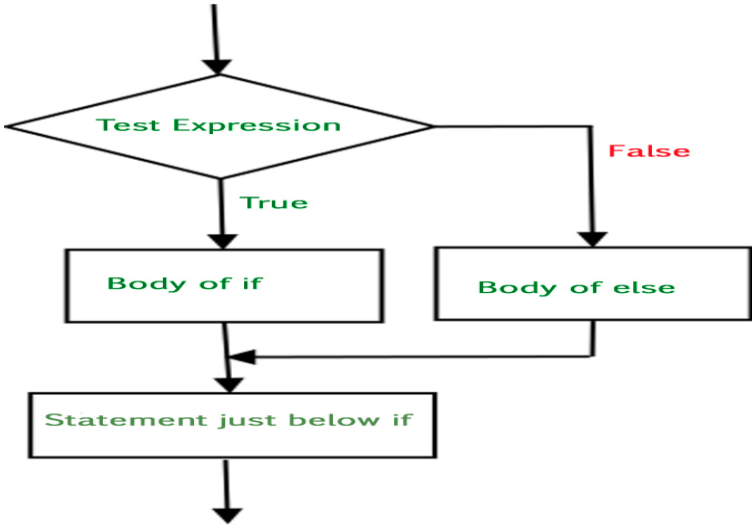

```
Syntax: if expression:
            #execute your code
Example:  a = 15
          if a > 10:
            print("a is greater")                    #Output:  a is greater
```

if else Statements

```
Syntax:if expression:
            #execute your code
          else:
```

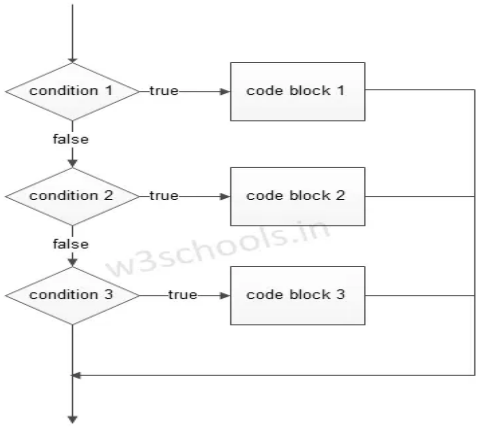
```
        #execute your code
Example:  a = 15, b = 20

        If a > b:
            print("a is greater")
        else:
            print("b is greater")                    #Output:  b is greater
```



Elif(if-elif ladder) Statements

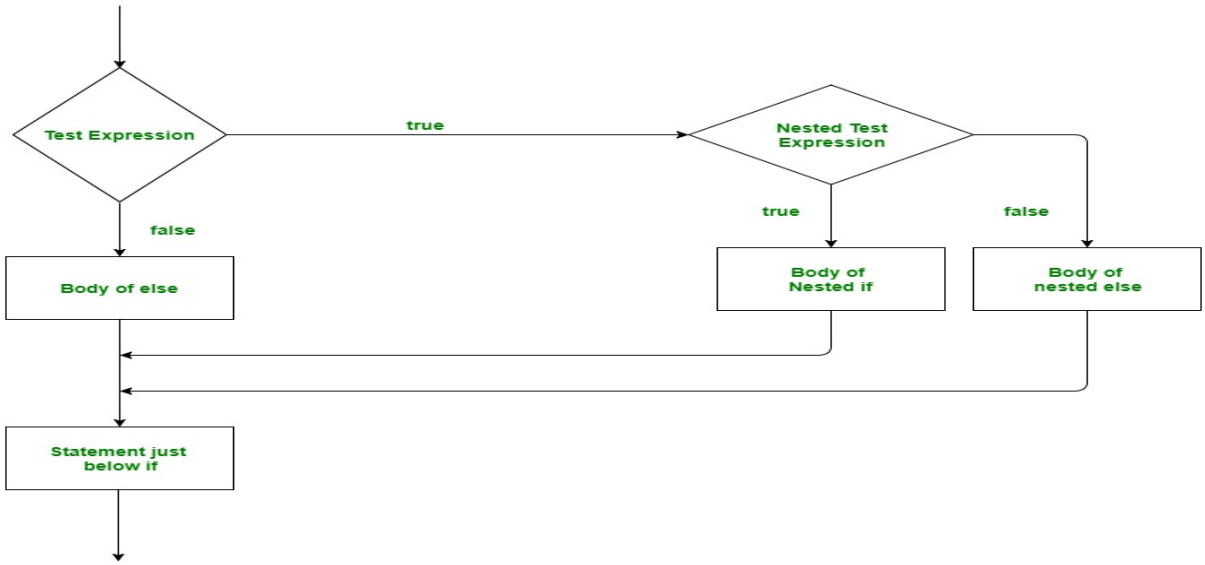
elif - is a keyword used in Python replacement of else if to place another condition in the program. This is called chained conditional.




```
Syntax: if expression:
            #execute your code
        elif expression:
            #execute your code
        else:
            #execute your code
Example:  a = 15, b = 15
        If a > b:
            print("a is greater")
        elif a == b:
            print("both are equal")
        else:
            print("b is greater")                    #Output:  both are equal
```

Nested Statements

Nested if statements is an if statement inside another if statement



Syntax:

```
if (expression):
  if(nested expression):
    Statement of nested if
  else:
    Statement of nested if else
  Statement of outer if
Statement outside if block
```

Example :

```
num1 = int( input())
num2 = int( input())
if( num1>= num2):
    if(num1 == num2):
        print(f'{num1} and {num2} are equal')
    else:
        print(f'{num1} is greater than {num2}')
else:
    print(f'{num1} is smaller than {num2}')
```

LOOP

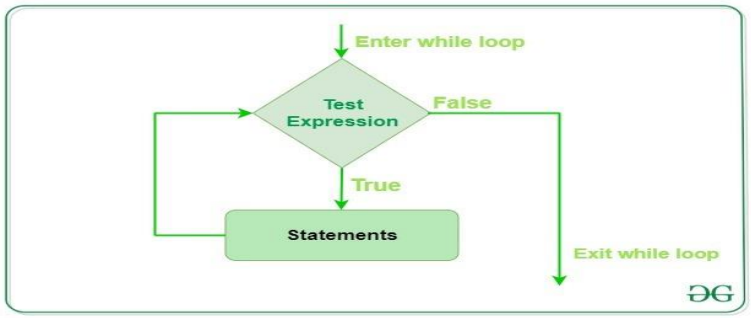
- **While Loops**
- **For loop**
- **Nested loop**

While Loops

- In python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied.when the condition becomes false, the line immediately after the loop in the program is executed

Syntax :

```
while expression:
    statement(s)
```



Example:

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

Using else statement with while loops

Syntax :

```
while condition:
     # execute these statements
else:
     # execute these statements
```

Example:

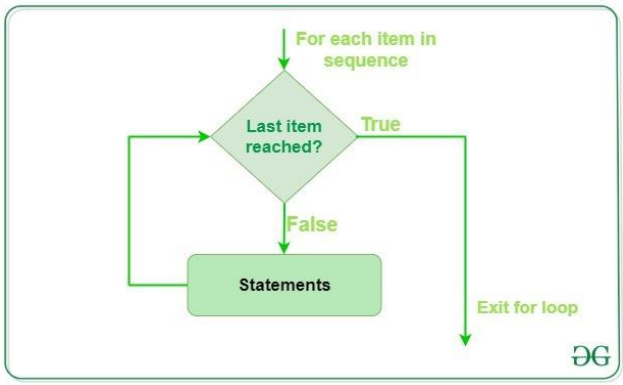
```
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
else:
    print("In Else Block")
```

For Loops

Python For loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, Set or Dictionary.
In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is **“for” loop** which is similar to each loop in other languages. Let us learn how to use for in loop for sequential traversals
Note: In Python, for loops only implements the collection-based iteration.

For Loops Syntax

```
for var in iterable:
    # statements (Here the iterable is a collection of objects like lists, tuples.)
```



Example:

```
# Iterating over a list
print("List Iteration")
l = ["geeks", "for", "geeks"]
for i in l:
    print(i)

# Iterating over a tuple (immutable)
print("\nTuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
    print(i)

# Iterating over a String
print("\nString Iteration")
s = "Geeks"
for i in s :
    print(i)

# Iterating over dictionary
print("\nDictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d :
    print("%s %d" %(i, d[i]))
```

For loop in Python with else

In most programming languages (C/C++, Java, etc), the use of else statements has been restricted with the if conditional statements. But Python also allows us to use the else condition with for loops.

Note: The else block just after for/while is executed only when the loop is NOT terminated by a break statement

```
# Python program to demonstrate
# for-else loop

for i in range(1, 4):
    print(i)
```

```
else:  # Executed because no break in for
    print("No Break\n")
```

Output:

```
1

2

3

No Break
```

Nested Loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

- Nested for loop statement is as follows:

```
for iterator_var in sequence:
   for iterator_var in sequence:
       statements(s)
       statements(s)
```

- Nested while loop statement is as follows:

```
while expression:
   while expression:
       statement(s)
       statement(s)
```

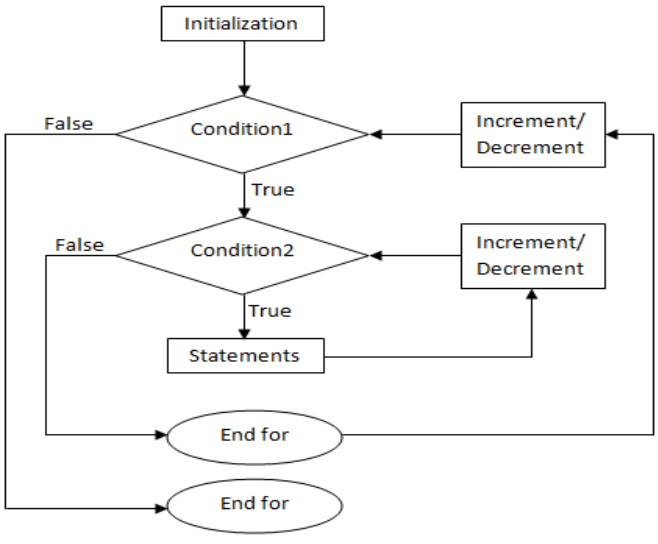


Fig: Flowchart for nested for loop

Example

```
# Python program to illustrate
# nested for loops in Python
from __future__ import print_function
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

Loop Control Statements

Loop control statements change execution from its normal sequence.

1. **Continue Statement**
   Python continue Statement returns the control to the beginning of the loop

Example:

```
# Prints all letters except 'e' and 's'
for letter in 'geeksforgeeks':
    if letter == 'e' or letter == 's':
        continue
    print('Current Letter :', letter)
```

Output:

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

2. **Break Statement in Python**
   Python break statement brings control out of the loop.

Example:

```
for letter in 'geeksforgeeks':
 # break the loop as soon it sees 'e'or 's'
    if letter == 'e' or letter == 's':
        break
 print('Current Letter :', letter)
```

Output:

```
Current Letter : e
```

3. **Pass Statement in Python**
   The pass statement to write empty loops. Pass is also used for empty control statements, functions, and classes.

Example:

```
# An empty loop
for letter in 'geeksforgeeks':
    pass
print('Last Letter :', letter)
```

Output:

```
last Letter : s
```

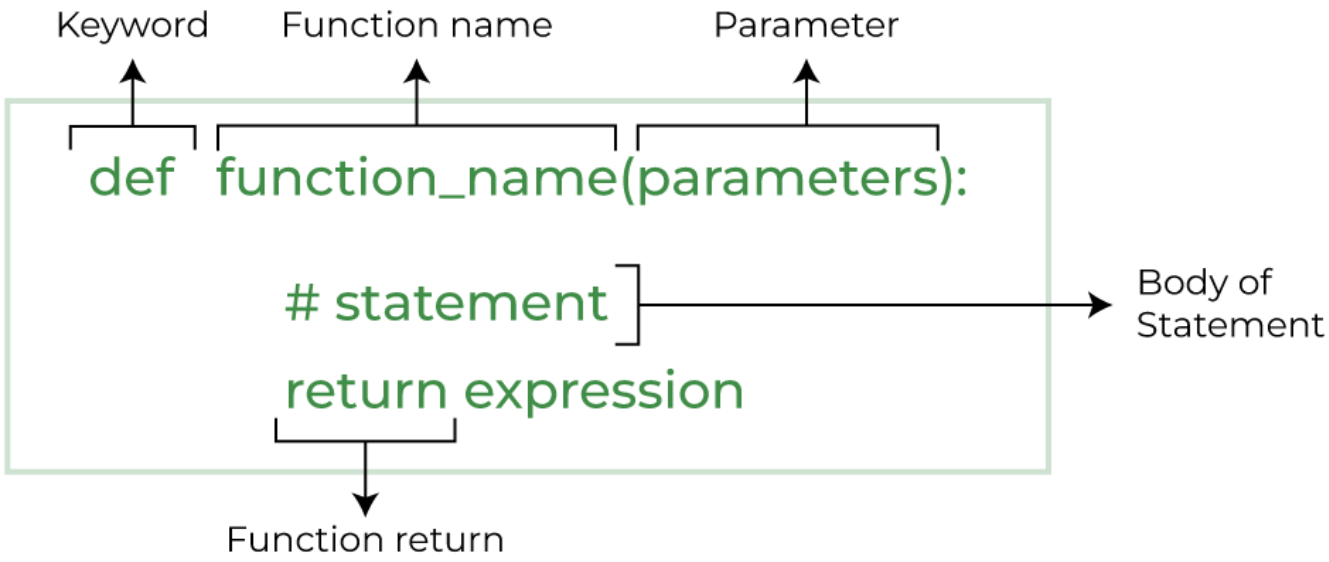
8-Python-Functions

Syntax for defining a function, Calling a Function, Function Arguments, Anonymous Functions

Python-Applications & Further Extensions

Python Functions

- A function is a block of code that return/perform the specific task,which only runs when it is called
- You can pass data, known as parameters, into a function.
- A function can return data as a result.



Benefits of Using Functions

1. Code Reusable - We can use the same function multiple times in our program which makes our code reusable.

2. Code Readability - Functions help us break our code into chunks to make our program readable and easy to understand.

Types of function

There are two types of function in Python programming:

1. **Standard library/built-in functions** - These are built-in functions in Python that are available to use.

   For example,

- `print()` - prints the string inside the quotation marks
- `sqrt()` - returns the square root of a number
- `pow()` - returns the power of a number

These library functions are defined inside the module. And, to use them we must include the module inside our program.

For example, `sqrt()` is defined inside the `math` module.

```
import math

# sqrt computes the square root
square_root = math.sqrt(4)

print("Square Root of 4 is",square_root)

# pow() comptes the power
power = pow(2, 3)

print("2 to the power 3 is",power)
```

2. **User-defined functions** - We can create our own functions based on our requirements.

Creating/Defining a Function

- In Python a function is defined using the **def** keyword:

  Syntax:-
  ```
  def my_function():
      print("Hello from a function")
  ```

Calling a Function

- To call a function, use the function name followed by parenthesis:

  Syntax:-
  ```
  def my_function():
      print("Hello from a function")
  my_function()
  ```

Function Arguments/ Parameters

- The terms *parameter* and *argument* can be used for the same thing: Information that are passed into a function.
- By default, a function must be called with the correct number of arguments.
- From a function's perspective:
  A ***parameter*** is the variable listed inside the parentheses in the function definition.
  An **argument** is the value that is sent to the function when it is called
- Arguments are specified after the function name, inside the parentheses.
- You can add as many arguments as you want, just separate them with a comma.
- Arguments are often shortened to **args** in Python documentations.

Example:-
```
def my_function(fname):          //parameter
    print(fname + " Refsnes")
my_function("Emil")              //argument
```

Types of Arguments

Python supports various types of arguments that can be passed at the time of the function call.

1. **Default arguments /Parameter Value**

- A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument.
- If we call the function without argument, it uses the default value:

```
def my_function(country = "Norway"):
    print("I am from " + country)
my_function("India")
my_function()
```

2. **Keyword Arguments**

- The idea is to allow the caller to specify the argument name with values so that caller does not need to remember the order of parameters.
- You can also send arguments with the *key* = *value* syntax.
- The phrase Keyword Arguments are often shortened to **kwargs** in Python documentations.

- This way the order of the arguments does not matter.
- Example:-

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)
my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

3. **Arbitrary Arguments, *args/ Variable-length arguments**

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a *tuple* of arguments, and can access the items accordingly:
- Example:-

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

Return statement/Values

- To let a function return a value, use the return statement:

```
def my_function(x):
  return 5 * x
print(my_function(3))
```

The pass Statement

- function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

```
def myfunction():
  pass
```

Anonymous Function/ Python Lambda

- In Python, a lambda function is a special type of function without the function name. Example,

```
lambda : print('Hello World')   //Note: This lambda function doesn't have any arguments.
```

- Here, we have created a lambda function that prints `'Hello World'`

Declaration

We use the `lambda` keyword instead of def to create a lambda function. Here's the syntax to declare the lambda function:

```
lambda argument(s) : expression
```

- `argument(s)` - any value passed to the lambda function
- `expression` - expression is executed and returned

Example-1:

```
# declare a lambda function   WITHOUT ARGUMENTS
greet = lambda : print('Hello World')
# call lambda function
greet()
# Output: Hello World
```

Example-2:

```
# declare a lambda function   WITH ARGUMENTS
greet = lambda name : print(' Hey there,', name')
# call lambda function
greet('Naveen')
# Output: Hey there, Naveen
```

- we have assigned a lambda function to the `greet` variable.
- Here, `name` after the `lambda` keyword specifies that the lambda function accepts the argument named `name`.

Python Modules

- Module is a file that contains code to perform a specific task.
- A module may contain variables, functions, classes etc.
- Let us **create a module**. Type the following and save it as `example.py`.

```
# Python Module addition
# Here, we have defined a function add() inside a module named exampl
def add(a, b):
result = a + b
return result
```

Import User Define Modules

- We can import the definitions inside a module to another module.
- We use the `import` keyword to do this. To import our previously defined module `example`, we type the following in the Python prompt.

```
import example
```

- This does not import the names of the functions defined in `example` directly in the current symbol table. It only imports the module name `example` there.
- Using the module name we can access the function using the dot `.` operator. For example:

```
addition.add(4,5) # returns 9
```

Note:

- Python has tons of standard modules. You can check out the full list of Python standard modules and their use cases.
- Standard modules can be imported the same way as we import our user-defined modules.

Import Python Standard Library Modules

- The Python standard library contains well over **200** modules. We can import a module according to our needs.
- Suppose we want to get the value of `pi`, first we import the math module and use `math.pi`. For example,

```
import math                             # import standard math module
print("The value of pi is", math.pi)    # use math.pi to get value of pi
```

➢ Import with Renaming

```
import math as m          # import module by renaming it
print(m.pi)
```

➢ From...import statement

We can import specific names from a module without importing the module as a whole.,

```
from math import pi       # import only pi from math module
//or// from math import pi, sin,cos
print(pi)
```

➢ Import all names

we can import all names(definitions) from a module using the following construct

```
# import all names from the standard module math
from math import *
print("The value of pi is", pi)
```