# SHRI GNANAMBICA DEGREE COLLEGE: MADANAPALLE
## (AUTONOMOUS)

**COURSE 9: PYTHON PROGRAMMING**
**(SEMESTER IV)**
**(W.E.F.2024-2025)**
**PROGRAM: BCA(ALL GROUPS)**

---

**Unit 1**

**Getting Started with Python:** Introduction to Python, Python Keywords, Identifiers, Variables, Comments, Data Types, Operators, Input and Output, Type Conversion, Debugging. Flow of Control, Selection, Indentation, Repetition, Break and Continue Statement, Nested Loops. **Strings-**String Operations, Traversing a String, String handling Functions.

**Unit 2**

**List:** Introduction to List, List Operations, Traversing a List, List Methods and Built-in Functions.

**Tuples and Dictionaries:** Introduction to Tuples, Tuple Operations, Tuple Methods and Built in Functions, Nested Tuples. Introduction to Dictionaries, Dictionaries are Mutable, Dictionary Operations, Traversing a Dictionary, Dictionary Methods and Built-in functions.

**Unit 3**

**Functions:** Functions, Built-in Functions, User Defined Functions, recursive functions, Scope
of a Variable **Python and OOP:** Defining Classes, Defining and calling functions passing arguments, Inheritance, polymorphism, Modules– date time, math, Packages.

**Exception Handling**- Exception in python, Types of Exception, User-defined Exceptions.

**Unit 4**

**Introduction to NumPy:** Array, NumPy Array, Indexing and Slicing, Operations on Arrays,
Concatenating Arrays, Reshaping Arrays, Splitting Arrays, Statistical Operations on Arrays.

**Data Handling:** Introduction to Python Libraries, Series, Data Frame, Importing and Exporting Data between CSV Files and Data Frames.

**Unit 5**

**Plotting Data using Matplotlib:** Introduction, Plotting using Matplotlib – Line chart, Bar
chart, Histogram, Scatter Chart, Pie Chart. **Database Connectivity:** Importing MySQL for Python, Connecting with a database, Forming a query in MySQL, Passing a query to MySQL.

# Unit 1

**Getting Started with Python:** Introduction to Python, Python Keywords, Identifiers, Variables, Comments, Data Types, Operators, Input and Output, Type Conversion, Debugging. Flow of Control, Selection, Indentation, Repetition, Break and Continue Statement, Nested Loops. Strings-String Operations, Traversing a String, String handling Functions.

# COMPUTER LANGUAGES

A **computer language** is  a way of communicating with the computer.

- A language acts as a bridge between humans and computers. helps us create software, websites, mobile apps ,database, AI systems.
- programming languages are used to create programs that control the behaviour of a system, to express algorithms or as a mode of human communication.

## PROGRAMMING LANGUAGES:-

Programming = Programming is the process of giving instructions to a computer to perform a task or solve the problem.

Languages = Language is a system of communication used to share ideas, thoughts, instructions, or information.

**LANGUAGE = A WAY TO COMMUNICATE**

Every language has words, grammar and rules, without language we cannot share information. **Programming languages** are special languages used to communicate with computers.

## WHAT IS A PROGRAMMING LANGUAGE ?

a programming language is a special language used to write instructions that a computer can understand and execute.  **(OR)** It is a way for humans to communicate with computers and tell them what to do.

- It is used to write programs.
- It follows specific rules and syntax**.**
- Computers cannot understand human languages, so programming languages act as a bridge.
- Examples: Python, C, Java, JavaScript, C++.

## TYPES OF PROGRAMMING LANGUAGES:-

**1.LOW LEVEL  LANGUAGE:-** A Low-Level Language is a type of programming language that is very close to the computer's hardware.

It is difficult for humans to read, write, and understand, but very easy for the computer (CPU) to understand.

## WHY IT IS CALLED "LOW LEVEL" LANGUAGE ?

Because:

- It is very close to the computer hardware.
- It is not similar to human language.
- It is hard for humans to read and write.

## TYPES OF LOW LEVEL LANGUAGES:-

### A) MACHINE LEVEL LANGUAGE:-

Machine Level Language is the lowest level programming language and the only language that the computer's CPU can understand directly. It is written completely in binary digits (0s and 1s).

Ex:- 10100011 11001010 , 00001111 00110011

machine level language is used in microcontrollers, embedded systems(washing machines ,Smart tv, remote controls, AC Controller ), firm ware (permanent software-

BIOS used to start the computer and run the program), device drivers, robotics, CPU Testing, and computer boot process(booting is the process of starting the computer).

**PROCESS OF BOOTING OF THE COMPUTER:-**

1. power on.
2. POST test(Power on self test-post checks CPU, RAM, Keyboard, Hard disk).
3. BIOS runs(Basic input output system,).
4. Bootloader loads(once BIOS finds a bootable device, it loads the bootloader).
5. Operating system loads into RAM(system files load).
6. computer is ready.

**B) ASSEMBLY LEVEL LANGUAGE:- (SECOND GENERATION LANGUAGE)**

assembly language developed in the mid-1950s was a great leap forward.it used symbolic codes, also known as mnemonic codes, which are easy to remember abbreviations, rather than numbers. It uses mnemonics (symbols and short codes) instead of binary. It is easier than machine language but still hardware-dependent. It needs an assembler to convert into machine code.

**EX:-**

ADD for adding, CMP for compare and MUL for multiply.
MOV AX , 4 (stores the value 4 in the AX register of the CPU) MOV means Move
MOV BX , 6 (stores the value 6 in the Bx register of the CPU
ADD AX , BX (adds the content of the AX and BX registers and stores the result in the AX register).

**2.HIGH LEVEL LANGUAGES:- (THIRD GENERATION LANGUAGES)**

These languages are close to human language, easy to read/write. These languages require compiler/interpreter to convert into machine language.

**EXAMPLES OF HIGH LEVEL LANGUAGES:-**

Python, java, C++, C, JavaScript, PHP, Ruby, Swift, Kotlin, GO.

**WHERE ARE HIGH LEVEL LANGUAGES USED:-**

Website designing (JavaScript, PHP), Mobile applications (Java, Kotlin, Swift), AI & Machine Learning (Python), Games (C++, C#), Banking systems(Java), & Desktop apps.

These languages are translated into machine code using compilers or interpreters.

**WHAT IS A COMPILER ?**

A **Compiler** is a software program that converts the entire source code into machine code at once and then executes it.

**EXAMPLE LANGUAGES THAT USE A COMPILER:-**

C, C++, Go, Swift, Java (partly compiler).

**HOW IT WORKS ?**

1. You will write a program.
2. The compiler checks the whole code.

3. It converts everything into machine code.
4. And then the program runs.

**ADVANTAGES:-**

- Very fast execution.
- Errors are shown together after compiling.
- Better for large and complex programs.

**WHAT IS AN INTERPRETER ?**

An **Interpreter** is a software program that converts and executes the code line by line.

**EXAMPLE LANGUAGES THAT USE AN INTERPRETER:-**

Python, JavaScript, PHP, Ruby.

**HOW IT WORKS ?**

1. It reads the first line and executes it.
2. Reads the second line and executes it.
3. Reads the third line and executes it.

(Continues like this, which means that it does line by line execution and it does not move to the next line if it finds errors in the present execution line).

**ADVANTAGES:-**

- Finding errors is easy, as it stops at the line that contains error.
- It is good for beginners.
- It supports in faster development.

**CLASSIFICATION OF FEATURES IN COMPILER AND INTERPRETER :-**

| FEATURES | COMPILER | INTERPRETER |
|---|---|---|
| **EXECUTION** | Whole program at once. | Line by Line. |
| **SPEED** | Faster. | Slower. |
| **ERROR DISPLAY** | All errors after compiling. | Error immediately at that line. |
| **OUTPUT** | Creates an executable file. | Does not create an executable file. |
| **EXAMPLES** | C, C++. | Python, JavaScript. |

**BENEFITS OF LEARNING PYTHON PROGRAMMING :-**

Python is one of the most popular, powerful and beginner-friendly programming languages.

**1)Easy to learn (Beginner Friendly) :-**

- Python has simple English-like syntax.
- It is very easy to read & write.
- It is one of the best language for beginners.
- Errors in Python are easy to understand.

**2)High Job demand (great career opportunities) :-**

**Python Developers** are required in many fields like **Python developer, Web developer, Data Analyst, Data Scientist, Machine Learning Engineer, AI Engineer, Cyber Security Analyst.**

**3) Used everywhere in real world :-**

**PYTHON** is used in almost all the modern technologies, some of them are **Artificial Intelligence (AI), Machine Learning, Data Science & Big Data, Web Development (Django, Flask), Automation & Scripting, Cyber Security, Game Development, Mobile Applications, Cloud computing, IOT** and much more.

**4) High Salary Packages :-**

**PYTHON** related jobs are well paid, Freshers are expected to claim upto **3 to 6 LPA,** while the experienced once are drawing almost **10 to 25+ LPA.** The employees working in **AI/ ML/ Data Science** fields are earning **20 to 50 LPA** in approx.

- **PYTHON = HIGH SALARY + FASTER GROWTH IN CAREER.**

**5) Fast Development (Saves time) :-**

**PYTHON** requires very fewer lines of code, while makes the development **faster.** Here, many built-in libraries are available, And hence many companies prefer python as it save lots of time.

**HOW TO LEARN PYTHON ?**

**1)Start with basics :-**

First learn the fundamental concepts like :

- What is PYTHON ?
- Variables in PYTHON.
- Data types in PYTHON.
- INPUT / OUTPUT.
- Operators in PYTHON.

**2) Learn Control Statements :-**

To control the program flow, we have to use :-

- If, else, elif.
- For loop.
- While loop.
- Break, continue.

Practise small programs daily.

**3) Learn Collections (Very IMP) :-**

**COLLECTIONS** in **PYTHON** are **LIST, TUPLE, SET** and **DICTIONARY.**

- These store multiple values.
- Most job questions come from here.

**4) Functions :-**

Learn how to reuse code by working with

- Function declaration.
- Return values.
- Arguments.

- Lambda functions.

Functions make your code clean.

**5) Learn File Handling :-**

READ / WRITE files:

- Open files.
- Read, write them.
- Learn Handling text files, CSV files.

File Handling is useful for automation and projects.

**6) Learn Object-Oriented Programming (OOP) :-**

Important for interviews and projects:

- Learn the concepts like **Class, Object, Inheritance, Polymorphism & Encapsulation.**

**7) Learn Modules & Packages :-**

We have to learn **Importing external modules** and **Using python libraries.**

**EX :-** import math , import random.

**8) Practice Small Projects :-**

Practice improves coding skills, try projects like :

**Calculator, Student management system, Guessing game, To-do list** and **Contact book.**

**9) Learn Frameworks (Optional but Useful) :-**

After basics, learn any of these:

- **Django & Flask for Web Development.**
- **Numpy, Pandas, Matplotlib & Scikit-learn for Data Science.**
- **Selenium, OpenPyXL for Automation.**

**10) Practice Daily (Very Important) :-**

Even 20–30 minutes daily is enough. Consistency makes you perfect.

**APPLICATIONS OF PYTHON :-**

**1)Web Applications :-** You can build complete websites using python. We use **Django, and Flask** to work with web frameworks.

**Ex of websites created using python are, Instagram, Youtube, Spotify.**

You can also create College websites, E-Commerce websites, Student Result Portals, News websites, etc…

**2)Mobile Applications :-**

Python can be used to build simple mobile apps using **Kivy** (python library that helps you create mobile apps), **BeeWare.**

You can create Calculator apps, To-do list apps (add, edit tasks , delete tasks, view pending work)CRUD Operation, Simple learning apps.

**3)Desktop Applications :-**

Python can create computer applications using **Tkinter, PyQt,** and **wxPython.**

You can create Attendance software, Billing systems and Student management systems.

**4)Machine Learning & Artificial Intelligence Applications :-**

Python is No.1 in AI/ML.

It uses libraries like **NumPy, Pandas, TensorFlow** and **Scikit-Learn** to create applications like Face detection systems, Voice recognition, Chatbots, and Recommendation systems (like YouTube, Amazon).

**5)Data Science & Data Analytics Applications :-**

In python we can Analyze data, Visualize charts, and Predict future trends by using the libraries like **Matplotlib, Seaborn,** and **Pandas.**

These are highly helpful in banking, hospitals, marketing and business analytics.

**6)Game Development :-**

Python can create mini-games using **Pygame**. Some of the games created are Car racing game, Snake game, Shooting game and Puzzle games.

## INTRODUCTION TO PYTHON

Python is a **high-level, interpreted, general-purpose** programming language used to develop different types of applications easily and quickly, which was introduced by **Guido-van Rossum** in **1991**. python is used for web development from server side, software development etc.

**1) Why python is a high-level language ?**

Python is called as high-level language because it is close to human language and is easy to read, write & understand. (It is very similar to our simple English).

**2) Why python is an interpreted language ?**

Python is called an interpreted language because here code is executed line by line, due to which error are shown immediately. python uses an interpreter because reads one line execute it then goes to next line.

**3) Why python is a General-Purpose Language ?**

General Purpose means a language that can be used for many types of applications. Here python is said to be as general purpose because it is used in many applications such as

*** web applications**
*** mobile applications**
*** desktop software**
*** AI & ML applications**
*** Games etc.**

## Python Syntax Compared to Other Programming Languages:-

**What is Syntax?**

A Syntax is the set of rules that defines how a program must be written so the computer can understand it.

Syntax = grammar of a programming language.

**C Programming :-**

```
#include <stdio.h>
int main() {
   printf("Hello");
   return 0;
}
```

**Java Programming :-**

```
class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World");
   }
}
```

**Python programming :-**

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

Python syntax is simple and readable compared to other programming languages like C and Java. It uses indentation instead of braces, does not require semicolons, and does not need data type declarations. Due to its English-like syntax and less code, Python is easy to learn and widely used, especially by beginners.

**Python installation and setup :-**

Python installation means installing python software on a computer so that we can write and run python programs.

**steps to install python (windows) :-**

1). open browser
2). go to official website (www.python.org)
3). click download python (latest version)
4). run installer
5). installation complete

**how to check python installation :-**

we can check whether python is installed in our system or not, by using command prompt

1. press window + R
2. type cmd --> Enter
3. Type (python --version)

**Python IDLE (Integrated Development and Learning Environment) :-**

it is default editor that comes with python to write and execute programs.
Some of the other optional editors and IDEs for python are **VS CODE, PYCHARM & JUPYTER NOTEBOOK,** etc…

**How to create a python file :-**

We create a Python file by writing code and saving it as a **".py"** file or extension. It can be run using Python in the command prompt or an IDE.

**Valid Python File Names,**

- Must end with **".py".**
- Can use letters, numbers, and underscore _.
- Should not start with a number.

**Ex :-** hello.py, test.py, my_program.py, python1.py, student_data.py, calculator_app.py.

**invalid Python File Names :-**

- missing **".py"** extension (Ex:- hello, program.txt , etc..).
- spaces in file name (Ex:- my program.py, hello world.py).
- starting with a number (Ex:- 12test.py, 278.py).
- special characters ( my@code.py, test#.py, etc.).
- using python keywords ( print.py, if.py, for.py, class.py).

**Basics Functions in python :-**

**print() :-**

it is a predefine function in python and it is used to display output on the screen.

**Ex:-**

- print('computer science')
- print("sri venkateswara university")
- print("'sgdc"')
- ab=1202
  print(ab)

**id() :-** (Shows memory address)

it is a predefine function in python. the main objective of id() function is to know the address of variable or identifier where our value is stored at memory allocation.

**Ex 1:-**

```
eid=1003
print(eid)
print(type(eid))
print(id(eid))
```

**Ex 2:-**

```
a = 10
b = 10
print(id(a))
print(id(b))
```

**Ex 3:-**

```
a = 10
b = 20
```

print(id(a), id(b))

**type():-**

it is a predefine function in python. it tells the data type of a value or variable.

**Ex 1:-**

```
a = 10
print(type(a))
```

**Ex 2:-**

```
b = 10.5
print(type(b))
```

**Ex 3:-**

```
c = "Python"
print(type(c))
```

**Ex 4:-**

```
print(type(100))
print(type("Hello"))
print(type(3.14))
```

**what is predefine function ?**

Predefined functions are built-in functions provided by Python. These functions are already defined during the development of Python and are available to the programmer for direct use without writing their definitions. (or)
During the development of the Python programming language, some useful functions were already created and provided by the Python developers. These functions are called predefined functions or built-in functions.

**Ex:-**

print() -- display output.
input() -- take input from user.
type() -- find data type.
id() -- memory address.
len() -- length of data.
sum() -- total of values.
max() -- biggest value.
min() -- smallest value.

**Variables :-**

a variable in python is a name that stores a value in memory. it works like a container or box where you keep some information.

Ex:-

**x = 10**

Here:

**x** is a **variable,** and **10** is the **value** stored in that variable.

**creating a variable and assigning a value :-**

we can assign a value to the variable at the time variable is created. we can use the assignment operator = to assign a value to a variable.

**Ex:-**

```
name="john"
age=19
salary=31000
print(name)
print(age)
print(salary)
```

in the above example, "john",19,31000 are values that are assigned to name, age and salary respectively.

**Ex:-** (initializing multiple variables)

```
x,y,z=10,"sgdc",123.34
print(x)
print(y)
print(z)
```

**Ex:-**

```
a1="python is a high level programmig language"
b1=52.32
c1=89
print(a1)
print(b1)
print(c1)
```

**Ex:-** showing variable and its type

```
ab="python"
bc=56
cd=56.22
print(ab)
print(type(ab))
print(bc)
print(type(bc))
print(cd)
print(type(cd))
```

(or)

```
ab="python"
bc=56
cd=56.22
print(ab, type(ab))
print(bc, type(bc))
```

```
print(cd, type(cd))
```

## Python Variable Rules :-

**Rule-1 :-** a variable name must start with a letter or underscore(_).

**Valid :-** **name, age, salary.**

**Invalid :-** **1name** ( because, it is starting with a number).

**Ex:-**

```
abc_ABC_123=1200
print(abc_ABC_123)
print(type(abc_ABC_123))
print(id(abc_ABC_123))
```

**Ex:-**

```
__=1200
print(__)
print(type(__))
print(id(__))
```

**Ex:-**

```
@=24
print(@)
```

**output:-**

**SyntaxError: invalid syntax**

**Rule-2 :-** a variable should not start with number or decimal number from 0-9.

**Ex:-**

```
1234_abc=123
print(1234_abc)
print(type(123_abc))
```

**output:-**

SyntaxError: invalid decimal literal

**Rule-3 :-** there is no length limit for an variable.

**Ex:-**

```
namenamenamenamenameamename="Shri Gnanambica"
print(namenamenamenamenameamename)
print(type(namenamenamenamenameamename))
```

**Rule-4 :-** an identifier or variable are case sensitive and as well as case in sensitive.

**case-sensitive :-** python is a case-sensitive language.

- capital letters (A-Z) and small letters (a-z) are treated as Different.
- python treats uppercase and lowercase letters as different.

**In Python,**

- **a is not the same as A.**
- **name is not the same as Name.**
- **value is not the same as VALUE.**

**Ex :-**

```
A=126
a=120
print(A,a)
print(type(A),type(a))
print(id(A),id(a))
```

**case-insensitive :-** capital and small letters are treated as same.

**Ex :-**

```
A=1200
A=1600
A=5000
print(A)
print(A)
print(A)
print(id(A),id(A),id(A))
print(type(A),type(A),type(A))
```

**output:-**

```
5000
5000
5000
3075226681744 3075226681744 3075226681744
<class 'int'> <class 'int'> <class 'int'>
```

(The variable A is reassigned multiple times, but Python stores only the latest value. Therefore, printing A multiple times prints the same last value (5000) every time).

A=1200 (variable A is created. it points to value 1200).

A=1600 (now A is reassigned, old value 1200 is removed and A now points to 1600).

A=5000 (again reassignment happens, old value 1600 is removed and A now points to 5000--> only the latest value survives).

**why does 5000 prints three times ?**

```
print(A)
print(A)
print(A)
```

**at this moment.**

- A contains only 5000
- you are printing the same variable A three times.

so output will be :

```
5000
5000
5000
```

**Rule-5 :-** an identifier or variable should not start with reserved keyword.

**Reserved keyword :-** a reserved keyword in python is a special word that has a fixed meaning already defined by python or already reserved by python for its own use. These words are used by python to write programs. We cannot use them as variable names, function names, or identifiers.

**how to see all keywords in python ?**

a reserved keyword in python is a special word that has a fixed meaning already defined by python or already reserved by python for its own use.

- These words are used by python to write programs.
- We cannot use them as variable names, function names, or identifiers.

**how to see all keywords in python ?**

```
import keyword
print(keyword.kwlist)
```

**output:-**

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

- (kwlist means all keyword list)
- in python we do have 33 or 35 keywords.

**Comments in python :-**

comments are non-executable lines in a python program. they are written to explain the code and help humans understand what the program is doing. Python ignores comments while running the program.

**Types of comments in python :-**

**1). single-line comment :-** written using the # (hash) symbol

**Ex :-**

```
# This is a single-line comment
print("Hello World") # This prints output
```

**2). multi-line comment:-** written using triple quotes

**Ex :-**

```
""" - This is a multi-line comment, Used to explain big programs.
```

**Data Types in python:-**

Data types specify the type of data stored in a variable. python supports **numeric, string, Boolean, list, tuple, set, dictionary** and **none** data types.

**Data type= type of data stored in a variable**.

**1). Numeric Data Types :-** It is used to store numbers.

**a). int(integer):-** int data type is used to store whole numbers in python. it can be represent as positive number or negative number.

**Ex:-**

```
x1= 1600
print(x1, type(x1))
x2= -455
print(x2,type(x2))
```

**b). float:-(decimal numbers) :-**It can be represent as decimal point number or floating point number. it may be either positive float decimal or negative floating decimal number.

**Ex:-**

```
x1=123.36
print(x1,type(x1),id(x1))
x2=-561.54
print(x2,type(x2),id(x2))
```

**Ex:-**

```
x1=1.3*10*10
print(x1,type(x1), id(x1))
```

**Ex:-**

```
x1=1.3e2
print(x1, type(x1), id(x1))
```

**c). complex:-**

Complex data type in python is used to represent numbers having real part and an imaginary part.

It is written in the form a +bj, where j represents the imaginary unit. python allows only j or J, not i.

**Ex:-** x1=100+500j

100 ----> real part

500j ---> imaginary part

**Ex:-**

```
x1=200+5j
x2=-250-5j
print(x1, type(x1))
print(x2, type(x2))
```

**Ex:- (finding real and imaginary part)**

```
x1=500+20.21j
print("Real Part:",x1.real)
print("imaginary part:",x1.imag)
```

**Ex:-**

```
x1 = 2j + 5J
print(x1, "real part", x1.real, "imaginary part", x1.imag)
```

**output:-**

7j real part 0.0 imaginary part 7.0

| Part | Value | Reason |
|---|---|---|
| `x1` | `7j` | 2j + 5j = 7j |
| `x1.real` | `0.0` | No real number present |
| `x1.imag` | `7.0` | Imaginary value |

0 + 2j
0 + 5j
---
0 + 7j

**If a complex number does not contain a normal number,**
**Python automatically sets the real part to 0.0.**

**input() :-**

The input() function in Python is used to take input from the user during program execution.

**Ex :-**

**Syntax of input() :-**

variable_name = input("message")

**Example :-**

```
name = input("Enter your name: ")
print(name)
```

**2). string data type (str) :-**

String is a data type in Python used to store text such as characters, words, or sentences.

Strings are written inside single quotes, double quotes, or triple quotes.

- while working with string data type space also consider as one character.
- python supports positive index which starts from 0 to end -1. it is also known as forward direction.
- python also supports negative direction which starts from -1 to end +1. it is also known as backword direction.
- string is immutable data type

**Ex:-**

```
str1='core python'
str2="Advance Python"
str3='''Django'''
str4="""javascript"""
str5='120'
```

```
str6="123.67"
print(str1,str2,str3,str4,str5,str6)
```

**Accessing Characters in String (Indexing) :-**

Indexing means accessing individual characters from a string using their position number.

In Python, every character in a string has a position (index).

- Indexing starts from 0, not from 1
- Spaces are also counted as characters

**Positive Indexing :-**

Positive indexing starts from 0 (left to right).

**Ex:-**

name = "PYTHON"

| Index | Character |
|---|---|
| 0 | P |
| 1 | Y |
| 2 | T |
| 3 | H |
| 4 | O |
| 5 | N |

**How to Access a Character**

**Syntax :-**

string_name[index]

**Examples:-**

```
name = "PYTHON"
print(name[0])
print(name[3])
print(name[5])
```

**Example with Sentence (Including Space) :-**

```
text = "Hello World"
print(text[0])
print(text[5])
print(text[6])
print(text[10])
```

| Index | Character |
|---|---|
| 0 | H |
| 1 | e |
| 2 | l |
| 3 | l |
| 4 | o |
| 5 | (space) |
| 6 | W |
| 7 | o |
| 8 | r |
| 9 | l |
| 10 | d |

## Negative Indexing:-

Negative indexing means counting characters from the end of the string (right to left). Python gives minus ( - ) numbers for characters from the end.

**Ex:-**

P Y T H O N

-6 -5 -4 -3 -2 -1

**Ex:-**

```
name = "PYTHON"
print(name[-1])
print(name[-2])
print(name[-3])
```

```
print(name[-6])
```

## Slice operator in string :-

The main objective of slice operator is to make the pieces of string object. slice operator is applicable for positive direction and negative direction as well.

**Slice Operator Syntax :-**

string[start : end : step]

start → where to begin (included)

end → where to stop (excluded)

step → jump count (optional)

**Ex:-**

```
s = "PYTHON"
print(s[0:4])
```

**Ex:-**

```
s="PYTHON"
print(s[2:5])
```

**Ex:-** (Starts from beginning automatically.)

```
s="PYTHON"
print(s[:3])
```

**Ex:-**

```
s="PYTHON"
print(s[3:])
```

## Form1 :-

str1[begin:]

here end(end-1) and step value is optional.

**Ex :-**

```
bca="PYTHON"
print(bca[3:])
```

**Ex :-**

```
bca="PYTHON"
print(bca[0:])
```

## Form 2:-

str1[:end(end-1)]

str1[:7] --> 7-1=6

here begin and step value is optional. the output of the string from the indexing position which starting from 0 to end-1 the position.

**Ex :-**

```
str1="Core Python"
print(str1)
print(str1[:7])
```

**Ex :-**

```
str1="Core Python"
print(str1)
print(str1[:9])
```

**Form3 :-**

str1[begin:end(end-1)]

here step is optional. the output of the string would be form giving position to till end-1 the position.

**Ex :-**

```
str1="Core Python"
print(str1)
print(str1[2:9])
```

**Form4 :-**

string[start : stop(end-1) : step]

here output will be generated based on step value.

**Ex:-**

```
str1="Core Python"
print(str1)
print(str1[2:9:1])
```

**Ex:-**

```
str1 = "Core Python"
print(str1)
print(str1[2:9:2])
```

C o r e P y t h o n

0 1 2 3 4 5 6 7 8 9 10

**Now apply it to:**

str1[2:9:2]

| Part | Meaning |
|---|---|
| **`2`** | **Start index (from position 2)** |
| **`9`** | **Stop index (up to 9, but **9 is NOT included**)** |
| **`2`** | **Step (skip 1 character, take every 2nd character)** |

**Select Characters Step by Step :-**

We start at index 2 and move till index 8 (because 9 is excluded), jumping 2 steps each time.

| Index | Character | Taken or Not? |
|---|---|---|
| **2** | **r** | ✅ **Taken** |
| **4** | **(space)** | ✅ **Taken** |
| **6** | **y** | ✅ **Taken** |
| **8** | **o** | ✅ **Taken** |

**Output :-**

Core Python

r yo

**Negative direction :-**

C o r e P y t h o n

0 1 2 3 4 5 6 7 8 9 10

-11-10 -9 -8 -7 -6 -5 -4 -3 -2 -1

**Formula :-(start,stop,step values)**

| Direction | Start | Stop | Step |
|---|---|---|---|
| ****Forward counting**** | **small → big** | **big value** | ****+ (positive)**** |
| ****Backward counting**** | **big → small** | **small value** | ****– (negative)**** |

**Ex :-** Reverse String (Negative Step)

```
str1="Core Python"
print(str1[::-1])
```

**Ex :-**

```
str1 = "Core Python"
print(str1[-6:-1])
```

-6 → character = P

-1 → character = n (but slicing does NOT include end)

Start from -6 (P)

Stop before -1 (n)

**Ex :-**

```
str1 = "Core Python"
print(str1[-11:-7])
```

**Ex :-**

```
str1 = "Core Python"
print(str1[1:8:0])
```

**output :-**

ValueError: slice step cannot be zero

**Ex :-**

```
str1 = "Core Python"
print(str1[-1:-7:-1])
```

## 3). Boolean data type (bool) :-

Used for true or false values.

Boolean data type is used to represent only two values:

True

False

**Boolean = Yes or No**

**Boolean = Correct or Wrong**

**Boolean = On or Off**

**Ex :-**

```
a = True
b = False
print(a)
print(b)
```

**Ex :-**

```
x = 10
y = 5
print(x > y)
print(x < y)
print(x == y)
```

**Ex :-**

```
x1=15
x2=170
print(x1)
print(x2)
x3=x1==x2
print("the result is:",x3)
x4=x1!=x2
print("the result is:",x4)
```

## 4). sequence data types:-

A sequence is an ordered collection of items, which can be of similar or different data types. Sequences allow storing of multiple values in an organized and efficient fashion.

### a). List

List is a collection of different or same type values stored in a single variable, written using square brackets []. It is ordered, indexed, mutable(make the changes), and allows duplicate values.

**mutable object :-(stateful object)**

Mutable data types can be changed after creation.

**Ex :-** list, set,dict,bytearray

**immutable (stateless object) :-**

once we create an object we cannot perform operations or we cannot modify that object then it is said to immutable object.

**Ex:-** int, float, str, complex, tuple, bytes.

**how to create a python list?**

the list can be created using either the list constructor or using square brackets [].

**Syntax of List :-**

list_name = [value1, value2, value3]

**1. using list() constructor :-**

**Ex :-**

```
chars = list("Python")
print(chars)
```

**2. using square brackets[]. the items inside the square brackets :-**

**Ex :-**

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

**Ex :-**

```
a1=[10,"Sgdc",9.24,98,"mpl"]
print(a1)
```

### b). Tuple :-

A tuple is a collection of elements that is:

Ordered

Immutable (cannot be changed)

Allows duplicate values

Can store different data types

Tuple is written using round brackets ( )

**how to create a tuple:-**

1.using parenthesis().

2.using a tuple() constructor.

**Ex :-**

```
t = (10, 20, 30)
print(t)
```

**Ex :-**

```
t = (1, "Python", 3.5, True)
print(t)
```

**Ex :-**

```
n1=(10,20,"python",12.34)
print(n1,type(n1))
```

## c) Range :-

The range() function in Python is used to generate a sequence of numbers within a specified range.

It is most commonly used in loops, especially for loops, to repeat an action a specific number of times.

range() returns a sequence of numbers starting from a value and ending before a value.

**Syntax :-**

```
for x1 in range(begin,end(end-1),step):
```

**Ex :-**

```
for x1 in range(10):
  print(x1)
```

**Ex :-**

```
for x1 in range(10):
  print(x1,end=" ")
```

**Ex :-**

```
for a1 in range(3,16):
  print(a1,end=" ")
```

**Ex :-**

```
for i in range(5):
  print(i, end=" ")
```

## How it works

Starts from 0

Ends at 5 − 1

Increases by 1 (default step)

**Ex :-**

```
for i in range(5, 20):
  print(i, end=" ")
```

**Ex :-**

```
for y1 in range(1,15,1):
  print(y1)
print()
```

**Ex :-**

```
for y1 in range(0,15,2):
  print(y1,end=" ")
```

**Ex :- finding the index values(positional values in string)**

```
s="Shri Gnanambica"
for i in range(len(s)):
  print(i, s[i])
```

## 5). set data type :-

A set in Python is used to store a collection of items with the following properties:

**No duplicate elements :-**

A set does not allow duplicate values. If we try to insert an element that already exists in the set, it is ignored (it does not overwrite any existing element).

**Unordered collection :-**

A set does not maintain any order of elements. When we access the elements of a set, they are returned in an arbitrary order.

Because of this, indexing and slicing are not supported in sets.

**Mutable collection :-**

A set is mutable, meaning we can add or remove elements after its creation.

However, individual elements of a set must be immutable (like integers, strings, or tuples). Elements cannot be modified directly.

## Creating a Set

### 1) Using curly braces { }

**Ex :-**

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

### 2) Using set() constructor

**Ex :-**

```
s = set([10, 20, 30, 40])
print(s)
```

**1. Unordered**

- Elements have no fixed position
- Indexing and slicing are not allowed

**Ex :-**

```
s = {10, 20, 30}
print(s[0])
```

**output :-**

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

**2. No Duplicate Values**

```
s = {1, 2, 2, 3, 3}
print(s)
```

**3. Mutable:-**

**Adding elements**

```
s = {10, 20, 30}
s.add(40)
print(s)
```

**Removing elements**

```
s = {10, 20, 30}
s.remove(20)
print(s)
```

## 6). Dictionary data type :-

A Python dictionary is a data structure that stores data in the form of key : value pairs.
Keys must be unique and immutable
Values can be of any data type and can be duplicated.

### Creating a Dictionary

**1) Using curly braces { }**

```
d1 = {1: 'Shri', 2: 'Gnanambica', 3: 'Degree'}
print(d1)
```

**2) Using dict() constructor**

```
d2 = dict(a="Shri", b="Gnanambica", c="Degree")
print(d2)
```

**Ex :-**

```
Product_info={"pid":1001,"pname":"laptop", "price":45000}
print(Product_info, type(Product_info))
```

**Ex :-**

```
n1={0:100,1:200,2:300,3:400}
print(n1, type(n1))
```

## 7) None data type :-

The None data type in Python represents the absence of a value or no value at all.
It is commonly used to indicate nothing, null, or no result.

**Ex :-**

```
x = None
print(x)
print(type(x))
```

**Ex :-**

```
e1=35000
print("emp salary:",e1)
e2=None
print(e2)
```

## 8). Bytes data type :-

Bytes data type in Python is used to store binary data in the form of values from 0 to 255.

It is immutable and mainly used in file handling, networking, and hardware communication.

Bytes can be created using the bytes() constructor or by using 'b' prefix.

**ASCII values**

Capital letters: A–Z → 65–90

Small letters: a–z → 97–122

Digits: 0–9 → 48–57

## How to Create Bytes Data Type?

There are two common ways.

### 1) Using bytes() constructor :-

Important Rules of bytes

✅ **Allowed**

✓ Integers from 0 to 255

✓ Byte literals using b" "

✓ Used for binary data

**Note :-** The bytes() constructor accepts only integer values between 0 and 255. Float values and strings are not allowed

❌ **Not Allowed**

❌ Strings inside list

❌ Numbers > 255

❌ Characters directly

**Ex :-**

```
b = bytes([65, 66, 67, 68])
print(b)
```

**Output :-**

```
b'ABCD'
```

**Explanation :-**

65 → A
66 → B
67 → C
68 → D
These numbers are converted using ASCII values.

**2) Using b prefix (byte literal) :-**

In Python, the b prefix is used to represent byte literals. Inside b"", only characters are allowed. To create bytes using numeric values, the bytes() constructor must be used.

**Ex :-**

```
x = b"Gnanambica"
print(x)
print(list(x))
```

In the given code, the string "Gnanambica" is converted into bytes using the b prefix. The list() function converts the bytes object into a list of ASCII values of each character.

**Bytes is IMMUTABLE (cannot be changed)**

**Example :-**

```
b = bytes([65, 66, 67])
b[0] = 90
print(b)
```

**output :-**

**TypeError: 'bytes' object does not support item assignment**

**Accessing Bytes Elements :-**

You can access bytes using indexing.

**Ex :-**

```
b = bytes([65, 66, 67])
print(b[0])
print(b[1])
```

**Python Code to Find ASCII Value of a Character:**

**ord()**

ord = ordinal value
ord() is a built-in Python function that is used to find the ASCII / Unicode value of a character.
- It accepts only one character
- Returns an integer number

**chr()**

chr() is a built-in Python function that is used to convert an ASCII / Unicode value into a character.

chr = character

- It accepts an integer
- Returns a character

**Ex :-** (displaying one character only)

```
ch = input("Enter a character: ")
print("ASCII value is:", ord(ch))
```

**Ex:-** (Multiple Characters)

```
text = input("Enter characters: ")
for ch in text:
  print(ch, " : ", ord(ch))
```

**converting ASCII Value → Character:-**

**Ex :-**

```
num = int(input("Enter ASCII value: "))
print("Character is:", chr(num))
```

**Ex :-**

```
values = input("Enter ASCII values separated by space: ").split()
for v in values:
  print(v, " : ", chr(int(v)))
```

**Why .split() is used**

- input() takes the entire line as one string.
- If the user enters 65 97 48, Python sees it as '65 97 48' (a single string).
- .split() splits it into separate parts: ['65','97','48']

Then we can loop over each number.

**Frozenset data type:-**

A frozenset is a collection of unique elements, just like a set,
but the main difference is:

- Frozenset is **IMMUTABLE** (cannot be changed after creation)
- Frozenset is an immutable version of a set.
- Once created, you cannot add, remove, or modify elements.

## Why the Name "Frozenset"?

**Frozen → cannot be changed**
**Set → collection of unique elements**
**So:**
**Frozenset = Frozen (unchangeable) set**

## How to Create a Frozenset ?

**✓ Using frozenset() constructor :-**

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

**Ex :-**

```
s = {1, 2, 3}
fs = frozenset(s)
print(fs)
```

**Ex :-**

```
fs = frozenset("python")
print(fs)
```

**Ex :-**

```
d = {frozenset([1,2,3]): "Python"}
print(d)
```

## When to Use Frozenset?

✓ You want data that must not change
✓ You need a set as a dictionary key
✓ You want data safety

# Python Operators :-

Operators are special symbols that perform specific operations on one or more operands(values) and then return a result.

```
a = 10
b = 5
c = a + b
```

**+ → operator**
**a and b → operands**

python has seven types of operators that we can use to perform different operations and produce a result.

1). Arithmetic operator
2). Relational operator

3). Assignment operator
4). Logical operator
5). Membership operator
6). Identity operator
7). Bitwise operator.

## 1) Arithmetic operator

Arithmetic operators in Python are used to perform basic mathematical operations such as addition, subtraction, multiplication, division, modulus, floor division, and exponentiation on numeric values.
there are seven arithmetic operators we can use to perform different mathematical operations, such as:

1. + (Addition)
2. - (subtraction)
3. * (multiplication)
4. / (division)
5. // (floor division)
6. % (modulus)
7. ** (Exponentiation)

**Ex :-**

**Addition Operator (+) --> Used to add two or more values.**

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

**Ex :-**

**Subtraction Operator (-) -->Used to subtract one value from another.**

```
a = 20
b = 10
print(a - b)
```

**Ex :-**

**Multiplication Operator (*) --> Used to multiply values.**

```
a = 5
b = 4
print(a * b)
```

**Ex :-**

**Division Operator (/) --> Used to divide two numbers.**

```
a = 10
b = 2
print(a / b)
```

**Ex :-**

**The floor division // rounds the result down to the nearest whole number.**

```
x = 15
y = 2
print(x // y)
```

**ouput:-** 7

**Ex :-**

**% (modulus)--> returns the remainder after division of two numbers.**

```
a=10
b=3
print(a%b)
```

**Ex :-**

**** (Exponentiation)--> used to calculate the power of a number.**

**Syntax**

**result = base ** exponent**

**base → number to be multiplied**

**exponent → how many times it is multiplied**

```
a=2
b=3
print(a**b)
```

**Explanation :-**

2 ** 3 = 2 × 2 × 2 = 8

**Ex :-**

```
# Variables
a = 15
b = 4
# Addition
print("Addition:", a + b)
# Subtraction
print("Subtraction:", a - b)
# Multiplication
print("Multiplication:", a * b)
# Division
print("Division:", a / b)
# Floor Division
print("Floor Division:", a // b)
# Modulus
```

```
print("Modulus:", a % b)
# Exponentiation
print("Exponentiation:", a ** b)
```

**Output:-**

Addition: 19
Subtraction: 11
Multiplication: 60
Division: 3.75
Floor Division: 3
Modulus: 3
Exponentiation: 50625

## 2. Assignment operator :-

In python, assignment operator are used to assigning value to the variable. Assign operator is denoted by = symbol. there are shorthand assignment operators in python, for example a+=2 which is equivalent to a=a+2.

| operator | Example | Same As |
|---|---|---|
| = | a=10 | a=10 |
| += | a+=30 | a=a+30 |
| -= | a-=15 | a=a-15 |
| *= | a*=10 | a=a*10 |
| /= | a/=5 | a=a/5 |
| %= | a %=5 | a=a%5 |
| **= | a**=4 | a=a**4 |
| //= | a//=5 | a=a//5 |
| >>= | a>>=5 | a=a>>5 |
| <<= | a<<=5 | a=a<<5 |

**Ex :- Simple Assignment (=)**

```
marks = 85
print(marks)
```

**Ex :- Add and Assign (+=)**

```
score = 50
score += 20
print(score)
```

**Ex :- Subtract and Assign (-=)**

```
balance = 1000
balance -= 250
print(balance)
```

**Ex :- Multiply and Assign (*=)**

```
salary = 20000
salary *= 2
print(salary)
```

**Ex :- Divide and Assign (/=)**

```
distance = 100
distance /= 4
print(distance)
```

**Ex :- Floor Divide and Assign (//=)**

```
students = 50
groups = 6
students //= groups
print(students)
```

## 3) Relational Operator (comparison operator) :-

Relational operators are also called comparision operators. it performs a comparision between two values. it returns a boolean True or False depending upon the result of the comparision.

**assume variable x holds 10 and variable y holds 5.**

| Operator | Description | Example |
|---|---|---|
| >(greater than) | It returns true if the left. Operand is greater than the right. | x>y Result is true. |
| <(less than) | It returns true if the left operand is less than the right. | x < y result is false. |
| ==(equal to) | it returns true if both operands are equal. | x==y (false) |

| | | |
|---|---|---|
| **!= (not equal to)** | **it returns false if both operands are equal.** | **x!=y (true)** |
| **>= (greather than or equal to)** | **It returns true if the left operand is greater than or equal to the right.** | **x>=y (true)** |
| **<=(less than or equal to)** | **it returns true if the left operand is less than or equal to the right.** | **x<=y (false)** |

**Ex :- Equal to (==) --> Checks whether both values are equal**.

```
a = 10
b = 10
print(a == b)
```

**Ex :- Not Equal to (!=) --> Checks whether values are different.**

```
a = 10
b = 5
print(a != b)
```

**Ex :- Greater Than (>)--> Checks if the left value is greater than the right value.**

```
a = 20
b = 10
print(a > b)
```

**(or)**

```
a1=int(input("Enter the first number:"))
a2=int(input("Enter the second nuber:"))
print(a1 > a2)
```

**Ex :- Less Than (<) --> Checks if the left value is smaller than the right value.**

```
a = 5
b = 10
print(a < b)
```

**Ex :- Greater Than or Equal to (>=) --> Checks if the value is greater or equal.**

```
marks = 35
print(marks >= 35)
```

**Ex :- Less Than or Equal to (<=) --> Checks if the value is less or equal.**

```
age = 18
```

```
print(age <= 18)
```

## 4) Logical operator :-

Logical operators are useful when checking a condition is true or not. all logical operator returns a boolean value true or false depending on the condition in which it is used.

## Types of Logical Operators in Python :-

### And :-

True if both conditions are True

**Truth Table :-**

| condition 1 | condition 2 | Result |
|---|---|---|
| **True(1)** | **True(1)** | **True(1)** |
| **True(1)** | **False(0)** | **False(0)** |
| **false(0)** | **true(1)** | **false(0)** |
| **false(0)** | **false(0)** | **false(0)** |

**Ex :-**

```
username=input("Enter the username:")
password=input("Enter the password:")
if(username=="sgdc" and password=="sgdc@123"):
    print("Welcome to Shri Gnanambica Degree College")
else:
    print("Provide Valid Details")
```

### Or :-

True if any one condition is True

**Truth Table :-**

| condition 1 | condition 2 | result |
|---|---|---|
| **True** | **True** | **True** |
| **True** | **False** | **True** |
| **False** | **True** | **True** |
| **False** | **False** | **False** |

**Ex :-**

```
first_name=input("Enter the First Name:")
Lat_name=input("Enter the last name:")
username=input("Enter the username:")
password=input("Enter the password:")
if(first_name== "shri" or Lat_name=="gnanambica") and (username=="sgdc" or
password=="sgdc123"):
    print("welcome to Shri Gnanambica Degree College")
else:
    print("please provide valid details")
```

**not :-**

The not operator is used to reverse a logical condition and is commonly used to check negative or opposite conditions in Python programs. **(Reverses the result)**

| condition | Result |
|---|---|
| **True** | **False** |
| **False** | **True** |

**Ex :-**

```
s = "Shri Gnanambica"
print(not s)
```

**Ex :-**

```
es = ""
print(not es)
```

**Ex :-**

```
x = 10
print(not (x == 5))
print(x != 5)
```

**Ex :-**

```
marks =int(input("Enter the marks:"))
if not (marks >= 35):
    print("Fail")
else:
    print("Pass")
```

## 5) Membership operator :-

Membership operators are used to test whether a value is present in a sequence such as a list, tuple, string, set, or dictionary.

It checks whether the given value or variable is present in a given sequence. if present, it will returns true else false. in python, there are two membership operators in and not in.

### 1) in operator :-

It returns a result as true if it finds a given object in the sequence. otherwise, it returns false.

**Ex :-**

```
numbers = [10, 20, 30, 40]
print(20 in numbers)
```

**Ex :-**

```
text = "python"
print('p' in text)
```

**Ex :-**

```
fruits = ("apple", "banana", "mango")
print("apple" in fruits)
```

**Ex :-**

```
student = {"name": "Ravi", "age": 20}
print("name" in student)
```

**(Note :- In dictionaries, membership checks only keys, not values.)**

**Ex :-**

```
list1 = [1, 2, 3, 4, 5]
str1 = "Hello World"
dict1 = {1: "Shri", 2: "Gnanambica", 3: "College"}

print(2 in list1)      # checking integer in a list
print('6' in str1)     # checking character in a string
print(3 in dict1)      # checking key in a dictionary
```

### 2) Not in Operator :-

Checks whether a value is not present in a sequence.

**Ex :-**

```
vowels = ['a', 'e', 'i', 'o', 'u']
print('z' not in vowels)
```

**Ex :-**

```
list1 = [1, 2, 3, 4, 5]
str1 = "Hello World"
dict1 = {1: "Shri", 2: "Gnanambica", 3: "College"}

print(2 not in list1)    # integer check in list
print('O' not in str1)   # character check in string
print(3 not in dict1)    # key check in dictionary
```

**Ex :-**

```
my_list=[11,15,21,29,50,70]
number=int(input("Eter the number:"))
if number not in my_list:
   print("Number is not present.")
else:
   print("Number is present")
```

## 6) Identity operator :-

The Identity Operators are used to compare the objects if both objects are actually of same data type and share same memory location. There are different identity operators such as:

### 1) IS  Operator :-

The is operator checks if two variables point to the same object (same memory location).

**Ex :-**

```
a = 10
b = 10
print(a is b)
```

**Ex :-**

```
c1="python"
d1="python"
print(c1 is d1)
```

**Ex :-**

```
s1=(12, "python")
print(id(s1))
s2=(12, "python")
print(id(s2))
print(s1 is s2)
```

**Ex :-**

```
a1=23+3j
```

```
a2=23+3j
print(id(a1))
print(id(a2))
print(a1 is a2)
```

**Ex :-**

```
a1={1:12, 2:34}
print(id(a1))
b1={1:12, 2:34}
print(id(b1))
print(a1 is b1)
```

## 2) is not operator :-

is not is an identity operator in Python.
It checks whether two variables do NOT point to the same object in memory

**Ex :-**

```
a = 10
b = 10
print(a is not b)
```

**Ex :-**

```
a = [1, 2, 3]
b = [1, 2, 3]
print(a is not b)
```

**Ex :-**

```
d1 = {1: 10}
d2 = {1: 10}
print(d1 is not d2)
```

## 7) Bitwise operator :-

In python, bitwise operator are used to performing bitwise operations on integers. to perform bitwise, we first need to convert integer value to binary(0 and 1 ) value.

the bitwise operator operates on values bit by bit, so it is called bitwise. it always returns the result in decimal format.

In python has 6 bitwise operators listed below :-

**1. & bitwise and**
**2. | bitwise or**
**3. ^ bitwise xor**
**4. ~ bitwise 1's complement.or bitwise not**

**5. << bitwise left-sift**
**6. >> bitwise right shift.**

**1) & - bitwise and**
Bitwise AND operator is somewhat similar to logical and operator. It returns True only if both the bit operands are 1 (i.e. True). All the combinations are −

**0 & 0 is 0**
**1 & 0 is 0**
**0 & 1 is 0**
**1 & 1 is 1**

**Ex :-**
```
a = 10
print(bin(a))
b = 6
print(bin(b))
print(a & b)
```
**output :-**
```
10 → 1010
 6 → 0110
------------
0010  → 2
```

- **bin()** - It is used to convert into binary number.

**converting binary number to decimal number :-**
```
b="1010"
decimal=int(b, 2)
print(decimal)
```

**2) | - bitwise or operator :-**
The "|" symbol (called pipe) is the bitwise OR operator. If any bit operand is 1, the result is 1 otherwise it is 0.

**0 | 0 is 0**
**0 | 1 is 1**
**1 | 0 is 1**
**1 | 1 is 1**

**Ex :-**

```
a=10
print(bin(a))
b=6
print(bin(b))
print(a | b)
```

**3) ^ - Bitwise XOR :- (caret symbol)**

The term XOR stands for exclusive OR. It means that the result of OR operation on two bits will be 1 if only one of the bits is 1.

**0 ^ 0 is 0**
**0 ^ 1 is 1**
**1 ^ 0 is 1**
**1 ^ 1 is 0**

**Ex :-**

```
a=10
print(bin(a))
b=6
print(bin(b))
print(a ^ b)
```

**4) ~ - Bitwise not operator :- (tide symbol)**

The bitwise NOT operator (~) is a unary operator used to invert all the bits of a number. It changes every 1 bit to 0 and every 0 bit to 1. This operation is performed on the binary representation of the number.

**~n = -(n + 1)**

**Ex :-**

```
a = 10
print(~a)
```

**5) << - left-shift operator :-**

The left shift operator (<<) is a bitwise operator used to shift the bits of a number to the left side by a specified number of positions. It is written as a << b, where a is the value to be shifted and b is the number of positions.

When the bits are shifted to the left, the empty positions on the right side are filled with zeros (0). The bits that move out from the left side are discarded.

**Ex :-**

```
a = 5
print(a << 1)
```

**Explanation :-**

Binary of 5: 00000101
After left shift by 1 position: 00001010
Decimal value of 00001010 is 10

**Output :-**

10

**6) >> - Right-Shift operator :-**

The right shift operator (>>) is a bitwise operator used to shift the bits of a number to the right side by a specified number of positions. It is written as a >> b, where a is the value to be shifted and b is the number of positions.
When the bits are shifted to the right, the empty positions on the left side are filled with zeros for positive numbers. The bits that move out from the right side are discarded.

**Ex :-**

```
a = 20
print(a >> 1)
```

**Explanation :-**

Binary of 20 : 00010100
After right shift by 1 position: 00001010
Decimal value of 00001010 is 10

**Output :-**

10

## INPUT AND OUTPUT IN PYTHON :-

**Input in Python:-**

Input refers to providing data to a program during execution.
Python uses the input() function to take input from the user.

**Syntax :-**

```
variable = input()
variable = input("Prompt message")
```

**Example :-**

```
name = input("Enter your name: ")
print(name)
```

**Important Rule of input() :-**

**The input() function always returns a STRING**
**Even if the user enters:- Number, Float, Any value**

**Example :-**

```
x = input("Enter a number: ")
print(type(x))
```

**Output :-**

```
<class 'str'>
```

**Output in Python :-**

Output refers to displaying the result or information produced by the program.
- Output means displaying data on the screen.
- Python uses the print() function for output.

**print() Function**

**Syntax :-**

```
print(object)
print(object1, object2)
```

**Example :-**

**Print Message**

```
print("Python Programming")
```

**Print Variables**

```
x = 10
y = 20
print(x, y)
```

## Type Coversion :-

Type conversion in Python is the process of changing one data type into another, which is essential for data manipulation and compatibility in operations. There are two primary types of conversion: implicit (automatic) and explicit (manual, also known as type casting).

**Implicit Type Conversion :-**

In implicit conversion, Python automatically converts one data type into another during expression evaluation.

In implicit type conversion of data types in Python, the Python interpreter automatically converts
one data type to another without any user involvement.

**Ex :-**

```
x = 10
y = 10.6
z = x + y

print("x:", type(x))
print("y:", type(y))
print("z =", z)
print("z :", type(z))
```

**Explicit Type Conversion :-**

Explicit type conversion requires the programmer to manually convert data types using built-in functions. This is necessary when Python cannot perform an automatic conversion or when you need control over the resulting type, though it may result in data loss.

**common types casting functions :-**

| function | converts to |
|---|---|
| **int()** | **integer** |
| **float()** | **float** |
| **str()** | **string** |
| **complex()** | **complex** |
| **list()** | **list** |
| **tuple()** | **tuple** |
| **set()** | **set** |

**Ex :- (String to Integer)**

```
x = "25"
y = int(x)
print(y, type(y))
```

**Ex :-**

```
a = "12.5"
b = float(a)
print(b)
```

**Ex :-**

```
num = 100
print(str(num))
```

**Ex :- (complex conversion)**

```
x = complex(5)
y = complex(2, 3)
print(x)
print(y)
```

**Ex :-**

```
s = "python"
print(list(s))
```

## Debugging in Python :-

Debugging is the process of finding, understanding, and fixing errors (bugs) in a computer program.

### What is a Bug ?

A bug is a mistake in a program that causes: Wrong output, Program crash, Unexpected behavior.

### Types of Errors in Python :-

#### 1) Syntax Errors :-

Errors due to wrong grammar in code.

```
if x > 5
   print("Hello")
```

**Missing (: )→ SyntaxError**

#### 2) Runtime Errors :-

Errors that occur during execution.

**Ex :-**

```
a = 10
b = 0
print(a / b)
```

**ZeroDivisionError: division by zero**

#### 3) Logical Errors :-

A logical error in Python occurs when your code is syntactically correct and runs without crashing, but produces an incorrect or unexpected result. These errors are often the most difficult to debug because the Python interpreter does not provide an error message or "traceback" to indicate where the issue lies.

**Ex :-**

```
marks = 35

if marks > 35:
    print("Pass")
else:
    print("Fail")
```

**Output :-**

Fail

**Why is this a Logical Error ?**

Student with 35 marks should PASS
Condition should be >= 35, not > 35

# Control Statements in Python :-

Control statements are used to control the flow of execution of a program.
They decide which statements run, how many times they run, or when execution should stop or skip.

In Python, control statements are mainly divided into three types :-

**1) Decision making or conditional statements**
**2) iterative statements**
**3) Transfer statements**

## 1) Decision making or conditional statements :-

Decision control statements check a condition and execute a particular block of code only if the condition is satisfied.
Decision Control Statements are used to make decisions in a program.
They allow the program to execute different blocks of code based on conditions (True or False).

**Types of Conditional statements :-**

**- if statement**
**- nested if statement**
**- if-else statement**
**- if-elif-else statement**

**if statement :-**

The if statement checks a condition.
If the condition evaluates to True, the statements inside the if block are executed; otherwise, they are skipped.

**Syntax :-**

```
if condition:
   statement(s)
```

**Ex :-**

```
language=input("Enter the language:")
if(language=="python"):
   print("Python is a Geneal purpose used programming language.")
```

**Ex :-**

```
x = 10
if x > 5:
   print("x is greater than 5")
```

**Ex :-**

```
age = int(input("Enter the age:"))
if age >= 18 and age <= 60:
   print("Eligible to work")
```

**Nested if statement:- (inner if)**

It is also known as inner if statement. inner if statement can be represent as defining or declaring a if statement inside another if statement while working nested or inner if statement outer if statement must be true.

**Syntax :-**

```
if condition1:
   if condition2:
      statement(s)
```

**Ex :-**

```
first_name=input("Enter the first name:")
last_name=input("Enter the last name:")
if(first_name=="Shri"):
   print("my first name is:",first_name)
   if(last_name=="Gnanambica"):
      print("my last name is:",last_name)
```

**Ex :-**

```
username = "admin"
password = "1234"
```

```
if username == "admin":
    if password == "1234":
        print("Login Successful")
```

**if-else statement :-**

An if-else statement checks a condition; if it is true, the if block executes, otherwise the else block executes.

The if-else statement is a decision control statement used to execute one block of code when a condition is true and another block when the condition is false.

**Syntax :-**

```
if condition:
    statement(s)
else:
    statement(s)
```

**Ex :-**

```
p1=input("Enter the password:")
p2=input("Enter the confirm password:")
if(p1==p2):
    print(p1, p2 ,"valid password")
else:
    print(p1,p2, " invalid password")
```

**Ex :-**

```
number=int(input("Enter the number:"))
if(number % 2==0):
    print(number, "Even number")
else:
    print(number, "odd number")
```

**Ex :-**

```
age = 16
if age >= 18:
    print("Eligible to vote")
else:
    print("Not eligible to vote")
```

**Ex :-**

```
marks =int(input("Enter the marks:"))
if marks >= 35:
    print("Pass")
else:
    print("Fail")
```

**Ex :-**

```
str1=input("Enter the object:")
if(str1==str1[::-1]):
   print(str1,"Palindrome number")
else:
   print(str1,"Not palindrome number")
```

**if-elif-else statement :-**

The if-elif-else statement is a decision control statement used to check multiple conditions one by one and execute the block of code whose condition is True.
The if-elif-else statement allows a program to test multiple conditions and execute only one block of code among them.

**Syntax :-**

```
if condition1:
   statement(s)
elif condition2:
   statement(s)
elif condition3:
   statement(s)
else:
   statement(s)
```

**Ex :-**

```
language=input("Enter the technology:")
if(language=="Python"):
   print(language, "Python is a general purpose")
elif(language=="Javascript"):
   print(language, "client side validation")
elif(language=="Django"):
   print(language, "to develop web application")
elif(language=="Reactjs"):
   print(language, " single page application")
elif(language=="CSS"):
   print(language, "creating styles")
else:
   print(language,"Invalid please enter proper language")
```

**Ex :-**

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

if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 60:
    print("Grade C")
else:
    print("Fail")
```

## Iterative Statements :-

Iterative statements are used to repeat a block of code multiple times until a given condition is satisfied. They are also called looping statements.
Iterative statements allow a program to execute a set of instructions repeatedly based on a condition or a sequence.

### Types of Iterative Statements in Python :-

Python has two main iterative statements:

1. **for loop**
2. **while loop**

### 1) for loop :-

The for loop is an iterative (looping) statement used to execute a block of code repeatedly for a fixed number of times or for each element in a sequence (like list, string, tuple, or range).

**Syntax :-**

```
for variable in sequence:
    statement(s)
```

**Ex :-**

```
for i in range(1, 6):
    print(i)
```

**Ex :-**

```
name = "Python"
for ch in name:
    print(ch)
```

**Ex :-**

```
for x1 in range(15):
    print("Python")
```

**Ex :-**

```
numbers = [10, 20, 30, 40]

for num in numbers:
    print(num)
```

**Ex :-**

```
n=int(input("Enter the limit:"))
for i in range(1, n+1):
    if i%2==0:
        print(i, "Even")
    else:
        print(i, "odd")
```

**Ex :-**

```
n1=int(input("Enter a number:"))
for i in [n1]:
    if i % 2==0:
        print("Even")
    else:
        print('odd')
```

**while loop :-**

A while loop is used to repeat a block of code as long as a condition is true. When the condition becomes false, the loop stops.

**Syntax :-**

```
while condition:
    statements
```

**Ex :-**

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

**Explanation :-**

**i = 1 → start value(initialization)**
**i <= 5 → condition (The condition decides how long the loop should run.)**
**print(i) → prints number**
**i = i + 1 → increases value**
**Loop stops when i becomes 6**

**- Initialization is assigning a first value to a variable before using it in a loop or program.**

**Infinite while loop :-**

An infinite while loop is a loop that never stops running because its condition is always True.

**Ex :-**

```
while True:
   print("Hello")
```

**(This loop runs forever.)**

**How to stop an infinite loop :-**

Use **break** statement inside the loop:

**Ex :-**

```
while True:
   print("Hello")
   break   # stops the loop after first iteration
```

**Output :-**

```
Hello
```

**Stop it manually (in console) :-**

Press Ctrl + C in Python terminal or IDE.

**Ex :-**

```
i = 1
while i <= 10:
   if i % 2 == 0:
      print(i,"EVEN")
   i += 1
```

**Ex :-**

```
n = int(input("Enter a number: "))
i = 1
while i <= n:
   if i % 2 == 0:
      print(i, "EVEN")
   i += 1
```

## Transfer statements :-

In Python, transfer statements (also known as jump statements or loop control statements) alter the normal flow of program execution.

Transfer statements in Python are statements that change or transfer the normal flow of execution of a program from one place to another.

They are mainly used inside loops and functions to control how and when the execution should move.

**1) break statement**
**2) continue statement**
**3) pass statement**

## break statement :-

The break statement is used to immediately stop a loop when a certain condition becomes true.

When break is executed, the control comes out of the loop and the program continues with the next statement after the loop.

**Ex :-**

```
numbers = [2, 4, 6, 8, 10]
for n in numbers:
    if n == 6:
        print("Number 6 found, stopping the loop")
        break
print(n)
```

**Ex :-**

```
i = 1
while i <= 10:
    print(i)
    if i == 5:
        print("Reached 5, stopping the loop")
        break
    i += 1
```

**Ex :-**

```
names = ["Alice", "Bob", "Charlie", "David"]
for name in names:
    if name == "Charlie":
        print("Found Charlie!")
        break
    print("Checking:", name)
```

**continue statement :-**

The continue statement is used to skip the current iteration of a loop and move to the next iteration immediately. Unlike break, it does not stop the loop; it just skips the remaining statements for the current iteration.

**Ex :-**

```
for i in range(1, 11):
    if i == 6:
        continue
    print(i, end=" ")
```

**Ex :- (Skip a number in a loop)**

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

**Ex :-**

```
for char in "Gnanambica":
    if char == "m":
        continue
    print(char, end=" ")
```

**Ex :-**

```
i=1
while i<=10:
    if i==5:
        i+=1
        continue
    print(i)
    i+=1
```

**pass statement :-**

The pass statement in Python is a null statement.

It does nothing when executed and is used as a placeholder in code.

Python requires at least one statement in a block. If you don't want to write code yet, you can use pass to avoid errors.

**Ex :-**

```
i = 10
if i > 0:
    pass
else:
    print("i is not positive")
```

**Ex :-**

```
for i in range(5):
    pass
```

## Indentation in Python :-

Indentation in Python refers to the spaces given at the beginning of a line of code.
Python uses indentation to define blocks of code.
Indentation is mandatory in Python. It is used in control structures such as if, if-else, for loop, while loop, and functions.
All statements that belong to the same block must have the same level of indentation.
If indentation is not written correctly, Python raises an IndentationError.
The recommended indentation in Python is 4 spaces, and mixing tabs and spaces should be avoided.

**Ex :-**

```
if num > 0:
    print("Positive number")
else:
    print("Negative number")
```

## Selection Statements in Python :-

Selection statements in Python are used to make decisions and control the flow of execution of a program.
They allow the program to choose different paths of execution based on conditions.
Python provides the following selection statements:

- **if statement**
- **if–else statement**
- **if–elif–else statement**
- **nested if statement**

The **if statement** executes a block of code only when the given condition is true.
The **if–else statement** executes one block if the condition is true and another block if it is false.
The **if–elif–else** statement is used to check multiple conditions, where only one block is executed.
A **nested if** statement is an if statement inside another if statement and is used for complex decision-making.
Selection statements help in writing logical, flexible, and decision-based programs and are commonly used in real-life applications like grading systems, voting eligibility, and number checking.

**Example :-**

```
num = 10
```

```
if num > 0:
    print("Positive number")
else:
    print("Negative number")
```

## Repetition in Python :-

Repetition in Python refers to the process of executing a block of code repeatedly until a specified condition is satisfied.

It is also known as iteration or looping.

Python provides two main repetition statements:

- for loop
- while loop

The **for loop** is used when the number of repetitions is known in advance. It iterates over a sequence such as a list, string, or range.

The **while loop** is used when the number of repetitions is not known and the loop runs as long as a condition remains true.

Repetition statements reduce code duplication, improve readability, and make programs efficient.

They are widely used for tasks like printing numbers, searching data, and processing lists.

**Example :-**

```
for i in range(1, 6):
    print(i)
```

## String  Operation in Python :-

A string operation is an action performed on a string (a sequence of characters) to create, modify, access, or analyze text data in a program.

In Python, string operations allow us to combine strings, extract characters, find text, replace text, compare strings, and format output.

### String Concatenation (+) :-

String concatenation means joining two or more strings into a single string.

In Python, this is done using the + operator.

**Ex :-**

```
a = "Hello"
b = "World"
print(a + b)
```

**Ex :-**

```
name = "Srinivas"
course = "Python"
print("My name is " + name + " and I am learning " + course)
```

**String Repetition :-**

String repetition means repeating the same string multiple times using the * operator.

**Ex :-**

```
s1="Shri Gnanambica "
r1=5
a1=s1*r1
print(a1)
```

**Ex :-**

```
a1="Om namo Venkateshaya "
print(a1 * 5)
```

**String Indexing :-**

Indexing is used to access individual characters in a string using position numbers.

- **Index starts from 0.**
- **Negative indexing starts from -1.**

**Ex :-**

```
s = "Python"
print(s[0])
print(s[3])
print(s[-1])
```

**String Slicing :-**

Slicing is used to extract a portion of a string.

**Syntax :-**

string[start : end : step]

**start - starting index (included)**

**end - ending index (excluded)**

**step - jump count**

**Ex :-**

```
s = "Python"
print(s[1:4])
print(s[:3])
print(s[2:])
print(s[::2])
print(s[::-1])
```

**Searching (find()) :-**

**find()** is used to search for a substring and return its index position.

**Ex :-**

```
s = "Python Programming"
print(s.find("Program"))
```

**Ex :-**

```
s = "Python Programming"
print(s.find("Java"))
```

**output :-**

-1

**Replacing (replace()) :-**

**replace()** is used to replace one word or character with another.

**Syntax :-**

```
string.replace(old, new)
```

**Ex :-**

```
s = "I like Java"
print(s.replace("Java", "Python"))
```

**Splitting (split()) :-**

**split()** breaks a string into a list of words.

**Syntax :-**

```
string.split(separator)
```

**Ex :-**

```
s = "Python is very easy"
print(s.split())
```

**Ex :-**

```
s = "Apple,Banana,Orange"
print(s.split(","))
```

**String Formatting :-**

a) Using **format()**(Used to insert values into a string.)

- The format() method formats the specified value(s) and insert them inside the string's placeholder.
- The placeholder is defined using curly brackets: {}. Read more about the placeholders in the Placeholder section below.
- The format() method returns the formatted string.

**Syntax :-**

```
string.format(value1, value2...)
```

**Ex :-**

```
name = "Srinivas"
age = 19
print("My name is {} and my age is {}".format(name, age))
```

**b) Using f-strings :-**

```
name = "Srinivas"
course = "Python"
print(f"My name is {name} and I am learning {course}")
```

## String Handling Functions in Python :-

String handling functions are built-in methods provided by Python to manipulate, modify, search, and analyze strings easily.

**1) upper() :-**

Converts all characters of a string into uppercase.

```
s = "python"
print(s.upper())
```

**2) lower() :-**

Converts all characters into lowercase.

```
s = "PYTHON"
print(s.lower())
```

**3) capitalize() :-**

Converts only the first character of the string to uppercase.

```
s = "python"
print(s.capitalize())
```

**4) title() :-**

Converts the first letter of every word to uppercase.

```
s = "python programming language"
print(s.title())
```

**5) replace(old, new) :-**

Replaces a word or character with another.

```
s = "I like Java"
print(s.replace("Java", "Python"))
```

**6) split() :-**

Splits a string into a list of words.

```
s = "Python is very easy"
print(s.split())
```

**7) join() :-**

Joins list elements into a string.

```
words = ["Python", "is", "easy"]
print(" ".join(words))
```

**8) find() :-**

Finds the index position of a substring.

```
s = "Python Programming"
print(s.find("Program"))
```

**9) count() :-**

Counts how many times a character or word appears.

```
s = "banana"
print(s.count("a"))
```

**10) startswith() & endswith() :-**

```
s = "Python Programming"
print(s.startswith("Python"))
print(s.endswith("ing"))
```

**Output :-**

True
True

**11) strip() :-**

strip() removes spaces from BOTH the left and right sides of a string.

**Syntax :-**

string.strip()

**Example :-**

**Removing spaces**

```
s = "   Python   "
print(s.strip())
```

**lstrip() (Left Strip) :-**

lstrip() removes spaces ONLY from the LEFT side of the string.

**Syntax :-**

string.lstrip()

**Example :-**

```
s = "   Python   "
print(s.lstrip())
```

**rstrip() (Right Strip) :-**

rstrip() removes spaces ONLY from the RIGHT side of the string.

**Syntax :-**

string.rstrip()

**Example :-**

```
s = "   Python   "
print(s.rstrip())
```

---

**B. Devendra Msc CS**
**Dept of Computers**
**Shri Gnanambica Degree College(A), Madanapalle.**