Overview Of Languages

What is Program?

A series of coded software instructions to control the operation of a computer or other machine.

What is Language?

They are the ways to transfer every instruction or solution to a particular problem (a means of communication) Eg. C, C++, Java, Python, etc.

Main three types of Languages are:

1. **Procedure Oriented Language**
2. **Object Oriented Language**
3. **Machine Level Language**

Procedure Oriented Language:

- Procedural programming uses a list of instructions to tell the computer what to do step-by-step.
- Procedures, also known as routines, or functions.
- If you want a computer to do something, you should provide step-by-step instructions to computer, how to do it.
- Examples of procedural languages include FORTRAN, COBOL, C and PASCAL.

Object Oriented Language:

- Object Oriented programming (OOP) is a programming language that relies on the concept of classes and objects.
- A Class in object-oriented programming is a blueprint or prototype that defines the variables and the methods (functions) common to Objects of a certain kind.
- An object in OOPS is a specimen of a class.
- **Example:**
- Class: Car (Attributes: Color, Brand, Model and Methods: repaint())
- Object: My Car (Attributes: Color = Blue, Brand = Alto, Model = Charger

  Methods: repaint())
- Object: My Friend's Car (Attributes: Color = Red, Brand = Nissan, Model = Ultima

  Methods:repaint())
- There are many object-oriented programming languages including JavaScript, C++, Java.

Machine Level Language:

- The machine-level language is a language that consists of a set of instructions that are in the binary form 0 or 1 which is also known as binary language or low level language.
- As we know that computers can understand only machine instructions, which are in binary digits, i.e., 0 and 1, so the instructions given to the computer can be only in binary codes.

- Creating a program in a machine-level language is a very difficult task as it is not easy for the programmers to write the program in machine instructions.
- It is error-prone as it is not easy to understand, and its maintenance is also very high.

Example of Procedure Oriented Language (POP):

```
#travel
Global Data
travel{
OpenApp( )
bookCab( )
Wait for the Cab( )
Sit in the Cab( )
reachdestination( )
PayCabFare( )
}
```

Example of Object Oriented Language (OOP):

```
Class Cab{
Cabservice,makelocation,number plate #data
}
Class cabdriver{
Name,employeeid #data
openDoor( ), drive( )  #methods
}
Class Passenger{
name,address  #data
openApp(), bookCab(), walk( ) #methods
}
```

Interpreter:

- Python interpreter is software logic between code and computer hardware. It takes a python program (like *.py file) and run the program line by line from start to bottom.

Compiler:

- A compiler is a computer program which helps you transform source code written in a high-level language into low-level machine language.

Difference between Compiler and Interpreter:

Compiler	Interpreter
Compiler takes an entire program at a time.	Interpreter takes a single line of code at a time.
Compiler works fast.	Interpreter works slower comparatively.
Compiler display all errors after compilation, all at the same time.	Interpreter displays errors of each line one by one.
Error Detection is difficult compared to Interpreter.	Error Detection is easy compared to the Compiler.
Generates intermediate object code which further requires linking, hence requires more memory.	No intermediate object code is generated, hence are memory efficient.
Example: C, C++, uses a compiler.	Example: PHP, Pearl, Python uses Interpreter.

Unit 1: Introduction To Python And Jupyter Notebooks

Introduction to Python:

- Python is a general-purpose high-level programming language.
- Python was developed by Guido Van Rossam in 1989 while working at National Research Institute at Netherlands.
- But officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.
- Python is recommended as first programming language for beginners.

Example1: To print Hello world

- **C:**
- #include<stdio.h>
- void main()
- {
- printf("Hello world");
- }

- **Python:**
- print("Hello World")

- The name Python was selected from the TV Show "The Complete Monty Python's Circus", which was broadcasted in BBC from 1969 to 1974.
- Guido developed Python language by taking almost all programming features from different languages

Where we can use Python?

We can use everywhere. The most common important application areas are

- Desktop Applications
- Web Applications
- Database Applications
- Network Programming
- Developing games
- Data Analysis Applications
- Machine Learning
- Artificial Intelligence Applications
- IoT

Features of Python:

- **Simple and easy to learn:** Python is a simple programming language.
- When we read Python program, we can feel like reading English statements.

- The syntaxes are very simple and only 30+ keywords are available.
- When compared with other languages, we can write programs with very a smaller number of lines. Hence more readability and simplicity.
- **Freeware and Open Source:** We can use Python software without any license and it is open source.
- **High Level Programming language:** Python is high level programming language and hence it is programmer friendly language.
- Being a programmer, we are not required to concentrate low level activities like memory management and security etc.
- **Platform Independent:** Once we write a Python program, it can run on any platform without rewriting once again.
- Internally PVM is responsible to convert into machine understandable form.
- **Portability:** Python programs are portable. ie we can migrate from one platform to another platform very easily.
- Python programs will provide same results on any platform.
- **Dynamically Typed:** In Python we are not required to declare type for variables.
- Whenever we are assigning the value, based on value, type will be allocated automatically.
- **Both Procedure Oriented and Object Oriented:** Python language supports both Procedure oriented (like C, pascal etc) and object oriented (like C++, Java) features.
- Hence, we can get benefits of both like security and reusability etc.
- **Interpreted:** We are not required to compile Python programs explicitly.
- Internally Python interpreter will take care that compilation. If compilation fails interpreter raised syntax errors.
- Once compilation success then PVM (Python Virtual Machine) is responsible to execute.
- **Extensible:** We can use other language programs in Python.
- **Embedded:** We can use Python programs in any other language programs. i.e. we can embed Python programs anywhere.
- **Extensive Library:** Python has a rich inbuilt library.
- Being a programmer, we can use this library directly and we are not responsible to implement the functionality.

Python - IDEs:

- There are many free and commercial IDEs available for Python.
- Jupyter Notebook
- Visual Studio Code
- Spyder
- PyCharm

Unit 1: Introduction To Python And Jupyter Notebooks

Jupyter Notebook:

- A Jupyter notebook is a file that contains both programming code and text descriptions.
- Jupyter notebooks can also contain embedded charts, plots, images, videos, and links.
- Jupyter notebooks run in a web browser like Firefox or Google Chrome.
- A Jupyter notebook is an application that can run Python code, display plots, show equations and contain formatted text.

Installing Jupyter Notebook:

- Jupyter installation requires Python 3.3 or greater, or Python 2.7.
- As an existing Python user, you may wish to install Jupyter using Python's package manager, pip.
- First, ensure that you have the latest pip; older versions may have trouble with some dependencies:
- pip3 install --upgrade pip
- Then install the Jupyter Notebook using: pip3 install jupyter

Opening and Creating Jupyter Notebook:

- Follow the steps given below to open the Jupyter notebook:
- Open the juputer on Windows start menu and select --> [Jupyter]
- This action opens the Jupyter file browser in a web browser tab.
- In the upper right select [New] --> [Python 3]
- A new notebook will open as a new tab in your web browser.
- Try typing the code below in the first cell in the notebook to the right of the In [ ]: prompt:
- Then click the run button in the middle of the menu at the top of the notebook.

Saving and downloading Jupyter Notebook:

- Jupyter notebooks can be saved using the save icon in the upper menu or by pressing [Ctrl] + [s].

- Jupyter notebooks can also be saved as a copy, similar to the Save As command common in many programs. To make a copy of a Jupyter notebook use File --> Make a Copy.
- Jupyter notebooks can be downloaded and saved using File --> Download As --> Notebook (.ipynb). Selecting this menu option will download the notebook as a .ipynb file.
- Jupyter notebooks can be saved in other formats besides the native .ipynb format.
- These formats can be accessed using the [File] --> [Download As] menu.
- The available file download types are:
- Notebook (.ipynb) - The native jupyter notebook format

- Python (.py) - The native Python code file type.
- HTML (.html) - A web page
- Markdown (.md) - Markdown format
- reST (.rst) - Restructured text format
- LaTeX (.tex) - LaTeX Article format
- PDF via LaTeX (.pdf) - a pdf exported from LaTeX, requires a converter

Python - Shell:

- Python is an interpreter language. It means it executes the code line by line. Python provides a Python Shell, which is used to execute a single Python command and display the result.
- It is also known as REPL (Read, Evaluate, Print, Loop), where it reads the command, evaluates the command, prints the result, and loop it back to read the command again.
- To run the Python Shell, open the command prompt or power shell on Windows write python and press enter. A Python Prompt comprising of three greater-than symbols >>> appears.
- Now, you can enter a single statement and get the result. For example, enter a simple expression like 3 + 2 press enter and it will display the result in the next line.

- What is REPL? REPL is the language shell, the Python Interactive Shell. The REPL acronym is short for Read, Eval, Print and Loop.
- The Python interactive interpreter can be used to easily check Python commands. To start the Python interpreter, type the command python without any parameter and hit the “return” key.
- The process is:
- Read: take user input.
- Eval: evaluate the input.
- Print: shows the output to the user.
- Loop: repeat.
- Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
- Type "help", "copyright", "credits" or "license" for more information.
- >>> print("Hello, World!")
- Hello, World!

- **Execute Python Script:**

- As you have seen above, Python Shell executes a single statement. To execute multiple statements, create a Python file with extension .py, and write Python scripts (multiple statements).
- For example, enter the following statement in a text editor such as Notepad.
- print ("This is Python Script.")
- print ("Welcome to Python Tutorial")
- Save it as myPythonScript.py, navigate the command prompt to the folder where you have

saved this file and execute the python myPythonScript.py command. It will display the result.

Python - IDLE:

- IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python.
- The Python installer for Windows contains the IDLE module by default.
- To start an IDLE interactive shell, search for the IDLE icon in the start menu and double click on it.
- This will open IDLE, where you can write and execute the Python scripts.
- You can execute Python statements same as in Python Shell.
- To execute a Python script, create a new file by selecting File -> New File from the menu.
- Enter multiple statements and save the file with extension .py using File -> Save. For example, save the following code as MyProgram.py.
- print (“Hello World")
- print ("Welcome to Python Tutorial")
- Press F5 to run the script in the editor window. The IDLE shell will show the output.

Data Types:

- 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

Text Type (str):

- str represents String data type.
- A String is a sequence of characters enclosed within single quotes or double quotes.
- s1=‘Arman’ or s1=“Arman"
- By using single quotes or double quotes we cannot represent multi line string literals.
- For this requirement we should go for triple single quotes (''') or triple double quotes (""")
- s1=′′′Arman

- soft'''
- s1="""Arman
- soft"""
- We can display a string literal with the print() function:
- print("Hello") # Hello
- print('Hello') # Hello
- Assigning a string to a variable is done with the variable name followed by an equal sign and the string:
- a = ''Arman''
- print(a) # Arman

Numeric Type:

- There are three numeric types in Python:

(1) int: Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

- **Example:**
- x = 1
- y = 35656333554887711
- z = -3255544
- print(type(x)) # <class 'int'>
- print(type(y)) # <class 'int'>
- print(type(z)) # <class 'int'>
- **Example:**
- a=10
- print(a) #10

(2) float: Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

- **Example:**
- x = 1.20
- y = 1.00
- z = -34.58
- print(type(x)) <class 'float'>
- print(type(y)) <class 'float'>
- print(type(z)) <class 'float'>
- Float can also be scientific numbers with an "e" to indicate the power of 10.

- **Example:**
- x = 36e3
- y = 11E4
- z = -87.6e100
- print(type(x)) <class 'float'>
- print(type(y)) <class 'float'>
- print(type(z)) <class 'float'>

Sequence Type:

(1) List: Lists are used to store multiple items in a single variable.

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- Lists are created using square brackets:
- **Example:**
- my_list = ["apple", "banana", "cherry"]
- print(my_list) # ['apple', 'banana', 'cherry']
- List items can be of any data type and A list can contain different data types.
- **Example:**
- list1 = ["apple", "banana", "mango"]
- list2 = [1, 6, 7, 9, 3]
- list3 = [True, False, True]
- list4 = ["abcd", 35, False, 50, "female"]
- print(list1) # ['apple', 'banana', 'mango']
- print(list2) # [1, 6, 7, 9, 3]
- print(list3) # [True, False, True]
- print(list4) # ['abcd', 35, False, 50, 'female']
- print(type(list1)) # <class 'list'>

(2) Tuple: tuples are used to store multiple items in a single variable.

- A tuple is a collection which is ordered, unchangeable and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
- Tuples are written with round brackets.
- **Example:**
- my_tuple = ("apple", "banana", "cherry")
- print(my_tuple) # ('apple', 'banana', 'cherry')

- Tuple items can be of any data type and A tuple can contain different data types.
- **Example:**
- tuple1 = ("apple", "banana", “mango“)
- tuple2 = (1, 6, 7, 9, 3)
- tuple3 = (True, False, True)
- tuple4 = ("abcd", 35, False, 50, “female“)
- print(tuple1) # (‘apple’, ‘banana’, ‘mango’)
- print(tuple2) # (1, 6, 7, 9, 3)
- print(tuple3) # (True, False, True)
- print(tuple4) # (‘abcd’, 35, False, 50, ‘female’)
- print(type(tuple1)) # <class ‘tuple’>

(3) range(x): it will give numbers from 0 to x-1.

- **Example:**
- x = range(3)
- print(x) # range(0,3)
- print(type(x)) # <class ‘range’>
- for x in range(3):
- print(x)

Output of for loop:

- 0
- 1
- 2

Mapping Type (dict):

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered, changeable and does not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values.
- **Example:**
- d = {10:’Lucky',20:’Arman', 30:’Dhairya’}
- print(d[10]) # Lucky
- print(d[20]) # Arman
- print(d[30]) # Dhairya

Set Type :

(1) Set: Sets are used to store multiple items in a single variable.

- A set is a collection which is both unordered and unindexed.
- Set items are unordered, unchangeable, and do not allow duplicate values.
- Unordered means that the items in a set do not have a defined order.
- Sets are unchangeable, meaning that we cannot change the items after the set has been created.
- Sets cannot have two items with the same value.
- Sets are written with curly brackets.

- **Note**: Set items are unchangeable, but you can remove items and add new items. For this purpose following methods can be used
- Remove(): Remove an element from Set.
- Discard(): Remove an element from Set
- POP(): Remove an arbitrary(random) element
- Clear() :Remove all elements.

- **Example:**
- my_set = {"apple", "banana", "cherry", "apple"}
- print(my_set) # {‘banana’, ‘cherry’, ‘apple’}

(2) frozenset(): It returns an unchangeable frozenset object.

- We can not change the value of a frozenset item.
- **Example:**
- my_list = ['apple', 'banana', 'cherry’]
- y = frozenset(my_list)
- print(y) # frozenset({'apple', 'cherry', 'banana'})

Boolean Type (bool):

- Booleans represent one of two values: True or False.
- In programming you often need to know if an expression is True or False.
- You can evaluate any expression in Python, and get one of two answers, True or False.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer.
- When you run a condition in an if statement, Python returns True or False.
- Almost any value is evaluated to True if it has some sort of content.
- Any string is True, except empty strings.

- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.
- **Example:**
- print(20 > 8) # True
- print(20 == 8) # False
- print(20 < 8) # False
- print(bool("abc")) # True
- print(bool("")) # False
- print(bool(123)) # True
- print(bool(["apple", "cherry", "banana"])) # True
- print(bool(0)) # False

Reserved Words or Keywords:

- In Python some words are reserved to represent some meaning or functionality. Such types of words are called reserved words.
- There are 33 reserved words available in Python.
- True, False, None, and, or ,not,is,if, elif, else, while, for, break, continue, return, in, yield, try, except, finally, raise, assert, import, from, as, class, def, pass, global, nonlocal, lambda, del, with.
- All Reserved words in Python contain only alphabet symbols.
- Except the following 3 reserved words, all contain only lower case alphabet symbols.
- True
- False
- None
- **Example:**
- a= true ×
- a=True √

TYPE CASTING:

- We can convert one type value to another type. This conversion is called Typecasting or Type conversion.
- The following are various inbuilt functions for type casting.
- 1) int() 2) float() 3) complex() 4) bool() 5) str()

1) int(): We can use this function to convert values from other types to int.

- >>> int(123.987) # 123
- >>> int(10+5j) #TypeError: can't convert complex to int
- >>> int(True) #1

- >>> int(False) # 0
- >>> int("10") #10
- >>> int("10.5") #ValueError: invalid literal for int() with base 10: '10.5'
- >>> int("ten") #ValueError: invalid literal for int() with base 10: 'ten'

2) float(): We can use float() function to convert other type values to float type.

- >>> float(10) #10.0
- >>> float(10+5j) #TypeError: can't convert complex to float
- >>> float(True) #1.0
- >>> float(False) #0.0
- >>> float("10") #10.0
- >>>float("10.5") #10.5
- >>> float("ten") # ValueError: could not convert string to float: 'ten'
- **Form-2:** complex(x,y)
- We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.
- Eg: complex(10, -2) # 10-2j
- complex(True, False) #1+0j

3) bool(): We can use this function to convert other type values to bool type.

- bool(0) # False
- bool(1) #True
- bool(10) #True
- bool(10.5) #True
- bool(0.178) #True
- bool(0.0) #False
- bool(10-2j) #True
- bool(0+1.5j) #True
- bool(0+0j) #False
- bool("True") #True
- bool("False") #True
- bool("") #False

4) str(): We can use this method to convert other type values to str type.

- >>> str(10) # '10'
- >>> str(10.5) #'10.5'
- >>> str(10+5j) # '(10+5j)'

- >>> str(True) #'True'

Variables:

- Variables are containers for storing data values.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable is created the moment you first assign a value to it.
- A variable name must start with a letter or the underscore character.
- **Example:**
- x = 5
- print(x) # 5
- _y = “Sai"
- print(_y) # Sai
- A variable name cannot start with a number.
- **Example:**
- 123z=6 # Invalid
- Variable names are case-sensitive (age, Age and AGE are three different variables).
- **Example:**
- x = 1
- print(x) # 1
- X = "Sumit“
- print(X) # Sumit (Here X will not overwrite x)
- Variables do not need to be declared with any particular type, and can even change type after they have been set.
- **Example:**
- x = 2 # x is of type int
- x = "Sai“ # x is now of type str
- print(x) #Sai
- If you want to specify the data type of a variable, this can be done with casting.
- **Example:**
- x = str(7)
- print(x) # x will be ‘7’
- y = int(7)
- print(y) # y will be 7
- z = float(7)

- print(z) # z will be 7.0
- You can get the data type of a variable with the type() function.
- **Example:**
- x = 3
- print(type(x)) #<class 'int'>
- y = "Sai"
- print(type(y)) #<class 'str'>
- String variables can be declared either by using single or double quotes.
- **Example:**
- a = "Johny"
- print(a) # Johny
- #double quotes are the same as single quotes:
- a = 'Johny'
- print(a) # Johny
- Python allows you to assign values to multiple variables in one line. It is called assignment of many values to multiple variables.
- **Example:**
- a, b, c = "Orange", "Banana", "Cherry"
- print(a) # Orange
- print(b) # Banana
- print(c) # Cherry
- Python allows you to assign the same value to multiple variables in one line. It is called assignment of one value to multiple variable.
- **Example:**
- a=b=c= "Apple"
- print(a) # Apple
- print(b) # Apple
- print(c) # Apple
- Python allows you to combine both text and a variable, Python uses the + character for this purpose.
- **Example:**
- a ="awesome"
- print("Python is "+ a) # Python is awesome.
- For numbers, the + character works as a mathematical operator.
- **Example:**

- a=20
- b=30
- print(a+b) # 50
- Python will give you an error, if you will combine string and a number.
- **Example:**
- x=30
- y="$Arman$"
- print(x+y) # TypeError: unsupported operand type(s) for +: 'int' and 'str'
- Python supports 2 types of variables.
- Global Variables
- Local Variables
- The variables which are declared outside of function are called global variables.
- These variables can be accessed in all functions of that module.
- The variables which are declared inside a function are called local variables.
- Local variables are available only for the function in which we declared it. i.e from outside of function we cannot access.

Comments:

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments starts with a # , and Python will ignore them.
- **Example:**
- #This is a comment
- print("Hello, Aayush") # Hello, Aayush
- Comments can be placed at the end of a line, and Python will ignore the rest of the line.
- **Example:**
- print("Hello, Aayush") #This is a comment
- A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code
- **Example:**
- #print("Hello, Aayush")
- print("Hello, World") # Hello, World
- 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, Aayush!") # Hello, Aayush
- 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, Aayush") # Hello, Aayush

Reading Input from Users:

- Python allows for user input.
- That means we are able to ask the user for input.
- Python 3.6 uses the input() method.
- **Example:**
- username = input("Enter username:") # Enter username: Arman
- print("Username is: " + username) # Username is: Arman

Python Operators:

- Operator is a symbol that performs certain operations.
- Python provides the following set of operators:
- Arithmetic Operators
- Relational Operators OR Comparison Operators
- Logical operators
- Ternary Operators OR Conditional Operators
- Bitwise operators
- Assignment operators
- Identity Operators
- Membership Operators

Arithmetic Operators:

❖ +	→	Addition (It is also used for string concatenation)
❖ -	→	Subtraction
❖ *	→	Multiplication (It is also used for string multiplication)
❖ /	→	Division
❖ %	→	Modulus
❖ //	→	Floor Division
❖ **	→	Exponent OR Power

- **Example:**
- a=10
- b=2
- print('a+b=',a+b) # a+b = 12
- print('a-b=',a-b) # a-b= 8
- print('a*b=',a*b) # a*b= 20
- print('a/b=',a/b) # a/b= 5.0
- print('a//b=',a//b) # a//b= 5
- print('a%b=',a%b) # a%b= 0
- print('a**b=',a**b) # a**b= 100
- / operator always performs floating point arithmetic. Hence it will always returns float value.
- But Floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If at least one argument is float type then result is float type.
- We can use +,* operators for str type also.
- If we want to use + operator for str type then compulsory both arguments should be str type only otherwise we will get error.
- **Example:**
- “Arman"+10 # TypeError: must be str, not int
- “Arman"+"10" # ‘Arman10’
- If we use * operator for str type then compulsory one argument should be int and other argument should be str type.
- **Example:**
- 2*“Arman" OR # ArmanArman
- “Arman"*2 # ArmanArman
- 2.5*“Arman" # TypeError: can't multiply sequence by non-int of type 'float’
- “Arman"*“Arman" # TypeError: can't multiply sequence by non-int of type 'str'

Relational (Comparison) Operators:

- `>` → Greater than
- `<` → Less than
- `>=` → Greater than or equal to
- `<=` → Less than or equal to
- `==` → Equal to
- `!=` → Not equal to

- **Example:**
- a=10
- b=20
- print("a > b is ",a>b) # a > b is False
- print("a >= b is ",a>=b) # a >= b is False
- print("a < b is ",a<b) # a < b is True
- print("a <= b is ",a<=b) # a <= b is True
- We can apply relational operators for str and bool types also.
- **Example:**
- a="Arman"
- b="Arman"
- print("a > b is ",a>b) # a > b is False
- print("a <= b is ",a<=b) # a <= b is True
- print(True>=True) # True
- print(10 >True) # True
- print(False > True) # False
- Chaining of relational operators is possible. In the chaining, if all comparisons returns True then only result is True. If atleast one comparison returns False then the result is False
- 10<20 # True
- 10<20<30 # True
- 10<20<30<40 # True
- 10<20<30<40>50 # False
- We can apply == and != operators for any type even for incompatible types also.
- 10==20 # False
- 10!= 20 # True
- 10==True # False

- False==False # True
- “Arman"==“Arman"   # True
- Chaining concept is applicable for equality operators. If atleast one comparison returns False then the result is False. Otherwise the result is True.
- 10==20==30==40   # False
- 10==10==10==10   # True

Logical Operators :

- We can apply logical operators (and,or,not) for all types.
- **For Boolean Types Behaviour:**
- and : If both arguments are True then only result is True.
- or: If atleast one arugemnt is True then result is True.
- not: Complement
- **Example:**
- True and False #False
- True or False #True
- not False   #True
- **For Non-Boolean Types Behaviour:**
- 0 means False
- non-zero means True
- Empty string is always treated as False
- x and y: If x is evaluates to false return x otherwise return y
- **Example:**
- 10 and 20 # 20
- 0 and 20 # 0
- If first argument is zero then result is zero otherwise result is y
- x or y: If x evaluates to True then result is x otherwise result is y
- **Example:**
- 10 or 20 # 10
- 0 or 20 # 20
- not x: If x is evaluates to False then result is True otherwise False
- **Example:**
- not 10 # False
- not 0 # True

Ternary OR Conditional Operators :

- Syntax: x = first Value if condition else second Value
- If condition is True then first Value will be considered else second Value will be considered.
- **Example:**
- a,b=10,20
- x=30 if a<b else 40
- print(x) # 30

Assignment Operators:

- We can use assignment operator to assign value to the variable.
- **Example:**
- x = 10
- We can combine assignment operator with some other operator to form compound assignment operator.
- **Example:**
- x += 10 → x = x+10
- The following is the list of all possible compound assignment operators in Python.
- +=
- -=
- *=
- /=
- %=
- //=
- **=
- &=
- |=
- ^=
- >>=
- <<=
- **Example:**
- x=10
- x+=20
- print(x) # 30

Membership Operators:

- We can use Membership operators to check whether the given object present in the given collection. (It may be String, List, Set, Tuple OR Dict)
- in → Returns True if the given object present in the specified Collection.
- not in → Returns True if the given object is not present in the specified Collection.
- **Example:**
- x="hello learning Python is very easy!!!"
- print('h' in x) # True
- print('d' in x) # False
- print('d' not in x) # True
- print('Python' in x) # True

Operator Precedence:

- If multiple operators present then which operator will be evaluated first is decided by operator precedence.
- **Example:**
- print(3+10*2) # 23
- print((3+10)*2) # 26
- The following list describes operator precedence in Python .
- (1) () → Parenthesis
- (2) ** → Exponential Operator
- (3) ~, - → Bitwise Complement Operator, Unary Minus Operator
- (4) *, /, %, // →Multiplication, Division, Modulo, Floor Division
- (5) +, - → Addition, Subtraction
- (6) <<, >> → Left and Right Shift
- (7) & → Bitwise And
- (8) ^ →Bitwise X-OR
- (9) | → Bitwise OR
- (10) >, >=, <, <=, ==, != → Relational OR Comparison Operators
- (11) =, +=, -=, *=... → Assignment Operators
- (12) is , is not → Identity Operators
- (13) in , not in → Membership operators
- (14) not → Logical not
- (15) and → Logical and
- (16) or → Logical or

Precedance (High to Low)	Operator	Description	Associativity
1	( )	Parentheses	left-to-right
2	**	Exponent	right-to-left
3	* / %	Multiplication/division/modulus	left-to-right
4	+ –	Addition/subtraction	left-to-right
5	<< >>	Bitwise shift left, Bitwise shift right	left-to-right
6	< <=	Relational less than/less than or equal to	left-to-right
7	> >=	Relational greater than/greater than or equal to	
8	== !=	Relational is equal to/is not equal to	left-to-right
9	is, is not	Identity	left-to-right
10	in, not in	Membership operators	
11	&	Bitwise AND	left-to-right
12	^	Bitwise exclusive OR	left-to-right
13	|	Bitwise inclusive OR	left-to-right
14	not	Logical NOT	right-to-left
15	and	Logical AND	left-to-right
16	or	Logical OR	left-to-right
17	=	Assignment	right-to-left
18	+= -=	Addition/subtraction assignment	
19	*= /=	Multiplication/division assignment	
20	%= &=	Modulus/bitwise AND assignment	
21	^= |=	Bitwise exclusive/inclusive OR assignment	
22	<<= >>=	Bitwise shift left/right assignment	

- **Example:**
- a=30
- b=20
- c=10
- d=5

- print((a+b)*c/d) # 100.0
- print((a+b)*(c/d)) # 100.0
- print(a+(b*c)/d) # 70.0

end

Python's print() function comes with a parameter called 'end'. By default, the value of this parameter is '\n', i.e. the new line character.

print("Welcome to", end = ' ')

print("Python for Python", end= ' ')

Output:

print("Python for Python", end= ' ')

Note here it does not take a new line.

Sep

The separator between the arguments to print() function in Python is space by default (softspace feature) , which can be modified and can be made to any character, integer or string as per our choice. The 'sep' parameter is used to achieve the same, it is found only in python 3.x or later. It is also used for formatting the output strings.

Example:

#code for disabling the softspace feature

print('G','F','G', sep='')

#for formatting a date

print('09','12','2016', sep='-')

Output:

GFG

09-12-2016

Escape Characters

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes

Problem 1:

```
txt = "We are the so-called "Vikings" from the north."
```

Output:

File "demo_string_escape_error.py", line 1

txt = "We are the so-called "Vikings" from the north."

^

SyntaxError: invalid syntax

Can be solved by:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Output:

We are the so-called "Vikings" from the north.

Problem 2:

txt = 'It's alright.'

print(txt)

Output:

txt = 'It's alright.'

^

SyntaxError: invalid syntax

:Can be solved by:

txt = 'It\'s alright.'

print(txt)

Output:

It's alright.

Problem 3: (To add word in a new line)

txt = "Hello\nWorld!"

print(txt)

Output:

Hello

World!

Problem 4: (To add space between two words)

txt = "Hello\tWorld!"

print(txt)

Output:

Hello World!

Problem 5: (#This example erases one character (backspace):)

txt = "Hello \bWorld!"

print(txt)

Output:

HelloWorld!

Extra Information (Just for Knowledge):

Note: From this section no question will be asked in exam

PyCharm- IDE:

- PyCharm is the most popular IDE used for Python scripting language.
- Follow the steps given below to install PyCharm on your system.
- Download the required package from the official website of PyCharm https://www.jetbrains.com/pycharm/download/#section=windows. Here you will observe two versions of package for Windows: (a) Professional (b) Community
- The professional package involves all the advanced features and comes with free trial for few days and the user has to buy a licensed key for activation beyond the trial period.
- Community package is for free and can be downloaded and installed as and when required. It includes all the basic features needed for installation. Note that we will continue with community package throughout this tutorial.
- Download the community package (executable file) onto your system and mention a destination folder.
- Now, begin the installation procedure similar to any other software package.

Other Data Types:

We can represent int values in the following ways:

- 1) Decimal form 2) Binary form 3) Octal form 4) Hexa decimal form
- **I) Decimal Form (Base-10):**
- It is the default number system in Python
- The allowed digits are: 0 to 9. Eg: a =10
- **II) Binary Form (Base-2):**
- The allowed digits are : 0 & 1
- Binary value should be prefixed with 0b or 0B
- Eg: a = 0B1111(Valid) , a = 0B123(Invalid)
- **III) Octal Form (Base-8):**
- The allowed digits are : 0 to 7
- Octal value should be prefixed with 0o or 0O.
- Eg: a = 0o123(Valid) , a = 0o786(Invalid)
- **IV) Hexa Decimal Form (Base-16):**
- The allowed digits are: 0 to 9, a-f (both lower and upper cases are allowed)
- Hexadecimal value should be prefixed with 0x or 0X
- Eg: a = 0XFACE, a = 0XBeef, a = 0XBeer
- Being a programmer, we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal form.

- b=0o10
- c=0X10
- d=0B10
- print(b) #8
- print(c) #16
- print(d) #2

Base Conversions:

- **1) bin():** We can use bin() to convert from any base to binary.
- >>> bin(15) # '0b1111'
- >>> bin(0o11) # '0b1001'
- >>> bin(0X10) # '0b10000'
- **2) oct():** We can use oct() to convert from any base to octal.
- >>> oct(10) # '0o12'
- >>> oct(0B1111) # '0o17'
- >>> oct(0X123) # '0o443'
- **3) hex():** We can use hex() to convert from any base to hexa decimal.
- >>> hex(100) # '0x64'
- >>> hex(0B111111) # '0x3f'
- >>> hex(0o12345) # '0x14e5'

Example:

- **Complex:** A complex number is of the form $a+bj$ where $j^2=-1$
  - Where a is a real part and b is an imaginary part.
  - 'a' and 'b' contain Integers OR Floating Point Values.
  - **Example:**
  - x=3 + 5j
  - y=10 + 5.5j
  - z=0.5 + 0.1j
  - print(type(x)) <class 'complex'>
  - print(type(y)) <class 'complex'>
  - print(type(z)) <class 'complex'>

Binary Type:

(2) bytes Data Type:

- bytes data type represents a group of byte numbers just like an array.

- The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.
- Once we creates bytes data type value, we cannot change its values, otherwise we will get TypeError.
- **Example:**
- x = [10,20,30,40]
- b = bytes(x)
- type(b) # bytes
- print(b[0]) # 10
- print(b[-1]) # 40
- **(2) bytearray Data type:**
- bytearray is exactly same as bytes data type except that its elements can be modified.
- **Example:**
- x=[10,20,30,40]
- b = bytearray(x)
- for i in b :
- print(i)
- 10
- 20
- 30
- 40
- b[0]=100
- for i in b:
- print(i)
- 100
- 20
- 30
- 40

Extra Operators:

Bitwise Operators:

- The Bitwise Operators are given below.
- & (And) If both bits are 1 then only result is 1 otherwise result is 0.
- | (OR) If atleast one bit is 1 then result is 1 otherwise result is 0.
- ^ (XOR) If bits are different then only result is 1 otherwise result is 0 .

- ~ bitwise complement operator.
- 1 → 0 & 0 → 1
- << → Bitwise Left Shift
- >> → Bitwise Right Shift
- We can apply these operators bitwise.
- These operators are applicable only for int and boolean types.
- By mistake if we are trying to apply for any other type then we will get Error.

- **Example:**
- print(4&5) # Valid
- print(True & True) # Valid
- print(10.5&5.6) #TypeError: unsupported operand type(s) for &: 'float' and 'float'
- **Bitwise And,OR and XOR:**
- print(4&5) # 4 (100 & 101 = 100)
- print(4|5) #5 (100 | 101 = 101)
- print(4^5) # 1 (100 ^ 101 = 001)
- **Bitwise Complement Operator (~):**
- We have to apply complement for total bits.
- **Example:**
- print(~5) # -6 (~0000 0101=1111 1010)
- Note: The most significant bit acts as sign bit.
- 0 value represents +ve number where as 1 represents -ve value.
- Positive numbers will be represented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form.
- **<< Left Shift Operator:** After shifting we have to fill the empty cells with zero.
- **Example:**
- print(10<<2) # 40 (Multiply 10 by 2^2)
- **>> Right Shift Operator:** After shifting we have to fill the empty cells with zero.
- **Example:**
- print(10>>2) # 2 (Divide 10 by 2^2 and consider integer value only)
- We can apply bitwise operators for boolean types also.
- **Example:**
- print(True & False) # False
- print(True | False) #True

- print(True ^ False) # True
- print(True<<2) # 4
- print(True>>2) # 0

Identity Operators:

- We can use identity operators for address comparison.
- There are 2 identity operators.
- is
- is not
- r1 is r2 returns True if both r1 and r2 are pointing to the same object.
- r1 is not r2 returns True if both r1 and r2 are not pointing to the same object.
- **Example:**
- a=10
- b=10
- print(a is b) # True
- x=True
- y=True
- print(x is not y) # False
- Note: We can use is operator for address comparison where as == operator for content comparison.

Type casting of complex(): We can use complex() function to convert other types to complex type.

- **Form-1:** complex(x) We can use this function to convert x into complex number with real part x and imaginary part 0.
- complex(10)==>10+0j
- complex(10.5)===>10.5+0j
- complex(True)==>1+0j
- complex(False)==>0j
- complex("10")==>10+0j
- complex("10.5")==>10.5+0j
- complex("ten") ==> ValueError: complex() arg is a malformed string