PYTHON PROGRAMMING
GATE DA & DSA

Topic Breakdown:

1. Getting Started with Python
- 1.1 Introduction to Python
- 1.2 Compiler vs Interpreter Language
- 1.3 Setting up Python work environment
- 1.4 Hello World in Python (running Python codes in 3 ways)
- 1.5 Print Statement
- 1.6 Variables
- 1.7 Comments
- 1.8 Data Types
- 1.9 Getting the input from the user

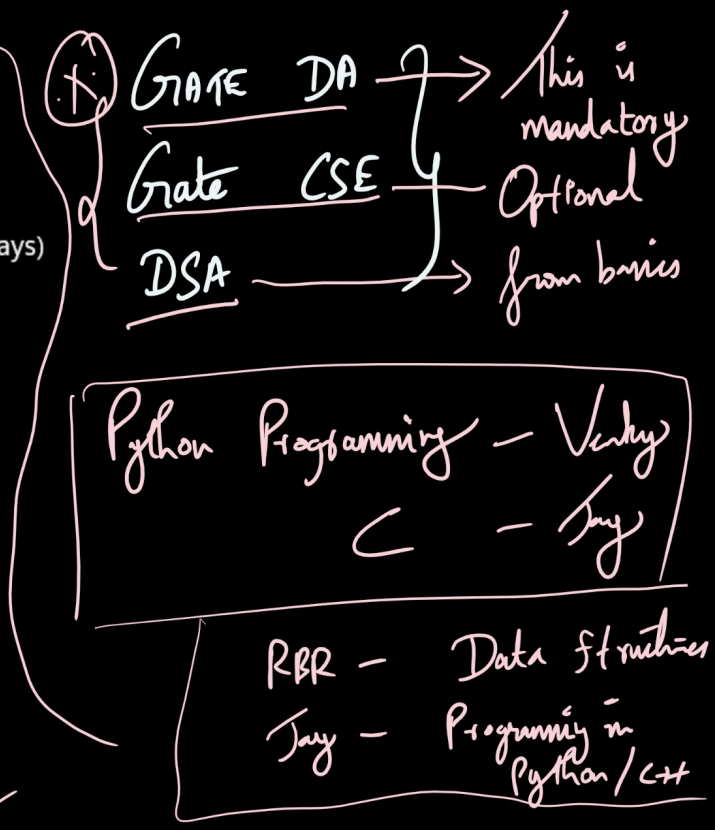
2. Operators in Python
- 2.1 Arithmetic Operators
- 2.2 Comparison Operators
- 2.3 Assignment Operators
- 2.4 Logical Operators
- 2.5 Bitwise Operators
- 2.6 Membership Operators
- 2.7 Operator Precedence
- 2.8 String Operations
- 2.9 Type Casting - Implicit & Explicit Conversions
- 2.10 Unicode Representations

3. Conditional Statements
- 3.1 if else statements
- 3.2 elif statement
- 3.3 Nested if else statements
- 3.4 Ternary Statements

4. Loops in Python
- 4.1 for loop
- 4.2 while loop
- 4.3 break statement
- 4.4 continue statement
- 4.5 pass statement
- 4.6 Nested Loops

5. Lists
- 5.1 Initializing a list
- 5.2 Access a list
- 5.3 Indexing in a list
- 5.4 length of a list
- 5.5 User input of a list
- 5.6 Adding elements to a list
- 5.7 Removing elements from a list
- 5.8 Slicing a list
- 5.9 Built-in-methods for Python Lists
- 5.10 Reversing a list

6. Tuple
- 6.1 Initializing a tuple
- 6.2 Access a tuple



6.3 Indexing in a tuple
6.4 length of a tuple
6.5 Mutability
6.6 How to add elements/remove elements using a tuple?
6.7 Built-in-methods for Python Lists

7. Set
7.1 Initializing a set
7.2 Access a set
7.3 length of a set
7.4 Mutability for set
7.5 How to add elements/remove elements in a set?
7.6 Frozen Sets
7.7 Built-in-methods for Python sets

8. Dictionary
8.1 Initializing a Dictionary
8.2 Access a Dictionary
8.3 Length of a Dictionary
8.4 Mutability for dictionary
8.5 Add/remove items in a dictionary
8.6 Copy a dictionary
8.7 Nested dictionaries
8.8 Built-in-methods for Python dictionary

9. Strings
9.1 What is a string?
9.2 Indexing in a string
9.3 String Reversal
9.4 Escape Sequencing
9.5 Updating/Deleting an element in a string
9.6 Formatting a string
9.7 String operations
9.8 Slicing a string
9.9 Built-in-methods for Python strings

10. Functions
10.1 What is a function?
10.2 Function declaration
10.3 Built-in-Functions
10.4 User-defined functions
10.5 Docstring in functions
10.6 Function Arguments
    10.6.1 Default Arguments
    10.6.2 Keyword Arguments
    10.6.3 Positional Arguments
    10.6.4 Arbitrary Arguments
10.7 Nested Functions
10.8 Anonymous Functions
10.9 Return statement
10.10 Pass by Reference & Pass by Value
10.11 Recursive Functions

11. GATE - Problem Solving (2 sessions)

12. DSA - Basic Problem Solving (2 sessions)

PYTHON PROGRAMMING
GATE DA & DSA

Agenda:

- Programming
- print() statement
- Variables
- Comments
- Data Types
- User Input

Programming:
Programming refers to a technological process for telling a computer which tasks to perform to solve problems. You can think of programming as a collaboration between humans and computers, in which humans create instructions for a computer to follow (code) in a language computers can understand.

How does computer programming work?
In very basic terms, programming tells a computer what to do. First, a programmer writes code, a set of letters, numbers, and other characters. Next, a compiler converts each line of code into a language a computer can understand. Then, the computer scans the code and executes it, thereby performing a task or series of tasks. Tasks might include displaying an image on a webpage or changing the font of a section of text.

Types of Programming Languages:

High-level vs. low-level languages

The biggest factor that differentiates high- and low-level programming languages is whether the language is meant to be easily understood by a human programmer or a computer. Low-level languages are machine-friendly, which makes them highly efficient in terms of memory usage but difficult to understand without the help of an assembler. Since they're not very people-friendly, they're also not widely used anymore. Examples include machine code and assembly languages.
High-level languages, on the other hand, are less memory efficient but much more human-friendly. This makes them easier to write, understand, maintain, and debug. Most popular programming languages in use today are considered high-level languages.

Interpreted vs. compiled languages

The distinction between interpreted and compiled languages has to do with how they convert high-level code and make it readable by a computer. With interpreted languages, code goes through a program called an interpreter, which reads and executes the code line by line. This tends to make these languages more flexible and platform-independent.

Examples of interpreted languages include:
1. ***Python***
2. JavaScript
3. PHP
4. Ruby

Compiled languages go through a build step where the entire program is converted into machine code. This makes it faster to execute, but it also means that you have to compile or "build" the program again anytime you need to make a change.

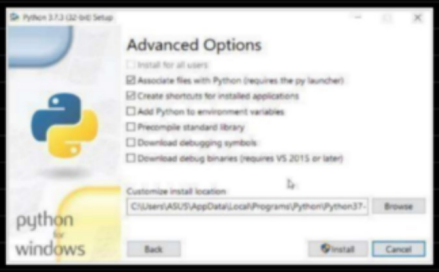
Examples of compiled languages include:
C, C++, and C

2. Rust
3. Erlang

Python supports Multiple Paradigms - Functional, procedural, object-oriented programming

Installing Python:

• Download the latest version of Python (https://www.python.org/downloads/).
• Run the installer file and follow the steps to install Python During the install process, check Add Python to environment variables. This will add Python to environment variables, and you can run Python from any part of the computer. Also, you can choose the path where Python is installed.



• To run the Python code, you can create a new file and save it with the **.py** extension.

```
python firstProgram.py
```

Setting Up Python Work Environment:

• To start working with Google Collaboratory Notebook you first need to log in to your Google account, then go to this link (https://colab.research.google.com.)
• For more details on Google Colab please visit the link below. (https://www.geeksforgeeks.org/how-to-use-google-colab/)

print() function
Syntax : print(value(s), sep= ' ', end = '\n')
Return Type: It returns output to the screen.
(Visit CODE file)

Variables
• Python variables are containers that store values.
• Python is not "statically typed".
• We do not need to declare variables before using them or declare their type.
• A variable is created the moment we first assign a value to it.
• A Python variable is a name given to a memory location. It is the basic unit of storage in a program.

Rules for Python variables
• A Python variable name must start with a letter or the underscore character.
• A Python variable name cannot start with a number.
• A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
• Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
• The reserved words(keywords) in Python cannot be used to name the variable in Python.

input ():
• This function first takes the input from the user and converts it into a string.
• The type of the returned object always will be <class 'str'>.
• It does not evaluate the expression it just returns the complete statement as String.
• For example, Python provides a built-in function called input which takes the input from the user. When the input function is called it stops the program and waits for the user's input. When the user presses enter, the program resumes and returns what the user typed.

Covering it in next class.

Skip it for today class.