Name: DHWANI GUPTA

Enrollment Number: EBEON0223761985

Python Theory Questions

1) What Is Python? Why It Is So Popular?

- Python is dynamically typed, scripted language and a high-level programming language that supports object-oriented programming approach as well as functional programming approach.
- It was created by Guido Van Rossum in 1989.

2) What Are the Key Features of Python?

- Python has several key features that contribute to its popularity and versatility as a programming language. Here are some of its key features:

i. **Readability**: Python emphasizes code readability and uses a clean and easy-to-understand syntax. It uses indentation instead of braces to define code blocks, making it more readable and visually appealing.

ii. **Simplicity**: Python's syntax and design aim to keep the language simple and concise. It has a minimalistic and intuitive approach, making it easier for beginners to learn and write code.

iii. **Expressive language**: Python allows developers to express concepts and ideas in fewer lines of code compared to other languages. It emphasizes a clear and straightforward coding style, enhancing productivity and reducing development time.

iv. **Versatility**: Python is a versatile language that supports various programming paradigms, including procedural, object-oriented, and functional programming. It offers a wide range of libraries and frameworks, making it suitable for web development, scientific computing, data analysis, artificial intelligence, machine learning, and more.

v. **Large standard library**: Python comes with a vast standard library that provides ready-to-use modules and functions for common tasks. It eliminates the need to build everything from scratch and enables developers to leverage existing code to accelerate development.

vi. **Interpretation**: Python is an interpreted language, which means that code execution occurs line by line without the need for compilation. This feature enables quick prototyping, easy debugging, and a more interactive coding experience.

vii. **Dynamic typing**: Python is dynamically typed, allowing variables to change their data type during runtime. This flexibility simplifies coding and provides more freedom to developers, but it also requires careful attention to data types to avoid potential errors.

viii. **Cross-platform compatibility**: Python code can run on various platforms and operating systems, including Windows, macOS, Linux, and more. It offers excellent portability, allowing developers to write code once and deploy it across multiple platforms.

ix. **Community and ecosystem**: Python has a vibrant and active community of developers worldwide. This community contributes to a vast ecosystem of third-party libraries and frameworks, expanding the functionality and possibilities of Python.

- Summary, Python is easy and simple to understand. It is interpreted and platform independent which makes debugging very easy. Python supports object-oriented programming approach as well as functional programming approach. Python provides very big library support like Numpy, Tensorflow, OpenCV, etc. it is possible to integrate other programming language with python.

3) What Type of Language Is Python? Programming Or Scripting?

- Python is a Scripting language as python is Interpreted Language. Python uses interpreter to translate the code into machine understandable language.

4) What Is Pep 8?

- PEP 8 is a style guide for writing Python code. "PEP" stands for "Python Enhancement Proposal," which is a document that describes new features, processes, or guidelines for the Python programming language. PEP 8 specifically focuses on the recommended coding conventions and style guidelines to promote consistency and readability in Python code.

5) Python An Interpreted Language. Explain.

- Python directly executes the instructions without earlier compiling a program into machine language and that's why it is an **Interpreted Language.**

6) How Is Memory Managed in Python?

- In Python, memory management is handled automatically by the interpreter.
- The interpreter uses a technique called "garbage collection" to automatically free up memory that is no longer needed by the program.

7) What Is Namespace in Python?

- A namespace is a collection of currently defined symbolic names along with information about the object that each name references. You can think of a namespace as a dictionary in which the keys are the object names and the values are the objects themselves.

8) What is PYTHONPATH?

- PYTHONPATH is an environment variable in Python that specifies additional directories where the Python interpreter should look for modules and packages when importing them. It is a way to extend the default search path for Python modules and packages.

- When you import a module or package in Python, the interpreter searches for the module or package in a predefined set of directories that includes the current directory, the built-in modules directory, and the directories listed in the sys.path variable. The PYTHONPATH environment variable allows you to add custom directories to the sys.path list, so that the interpreter can find modules and packages in those directories.

9) Is Python Case Sensitive?

- Yes, Python is a case-sensitive programming language. This means that Python distinguishes between uppercase and lowercase letters in variable names, function names, class names, and other identifiers.

10) Is Indentation required in python?

- Yes, indentation is required in Python. As python uses indentation to indicate the structure and scope of code.

11) What are reserved Words?

- Reserved words, also known as keywords, are words that are part of the programming language's syntax and have predefined meanings or functionalities. These words are reserved and cannot be used as identifiers (e.g., variable names, function names, class names) in the program. And they are always written in lower case. For example, print, and, or, if, else, try, except, finally, etc.

12) What are different data types available in Python?

- Python provides several built-in data types to handle different kinds of data. These data types represent the type of values that can be stored and manipulated in variables or data structures. Here are some of the commonly used data types in Python:

• Type	Explanation
1. Numeric Types	**int**: Represents integer numbers, e.g., 42, -17, 0. **float**: Represents floating-point numbers with decimal places, e.g., 3.14, -0.5, 2.0.
2.Boolean Type	**bool**: Represents boolean values that can be either True or False.
3.Sequence Types	**string**: Represents a sequence of Unicode characters, enclosed in single quotes or double quotes, e.g., 'Hello', "World". **list**: Represents an ordered collection of items, enclosed in square brackets, e.g., [1, 2, 3]. **tuple**: Represents an ordered, immutable collection of items, enclosed in parentheses, e.g., (4, 5, 6).
4.Mapping Type	**dict**: Represents a collection of key-value pairs enclosed in curly braces, e.g., {'name': 'John', 'age': 25}.
5.Set Types	**set**: Represents an unordered collection of unique items, enclosed in curly braces, e.g., {1, 2, 3}. **frozenset**: Represents an immutable set, enclosed in parentheses, e.g., frozenset({1, 2, 3}).
6.Sequence of Bytes Type	**bytes**: Represents a sequence of bytes, e.g., b'Hello'.
7.Bytearray Type	**bytearray**: Represents a mutable sequence of bytes.
8.None Type	**None**: Represents the absence of a value or a null value.

13. What is indexing and slicing explain with example.

- Indexing and slicing are techniques used to access specific elements or subsequences within a data structure like a string, list, or tuple in Python.
- **Indexing**: Indexing allows you to retrieve an individual element from a sequence by its position or index. In Python, indexing starts at 0, meaning the first element has an index of 0, the second element has an index of 1, and so on. You can access an element at a specific index using square brackets [].

```
my_list = [10, 20, 30, 40, 50]
print(my_list[0])    # Output: 10
print(my_list[2])    # Output: 30
```

```
10
30
```

- Slicing: Slicing allows you to extract a subsequence or a portion of a sequence by specifying a range of indices. It returns a new sequence containing the selected elements. The slicing syntax uses a colon : inside square brackets [] to define the start, stop, and step values. The general syntax for slicing is: sequence[start:stop:step]
    i. start is the index where the slice begins (inclusive).
    ii. stop is the index where the slice ends (exclusive).
    iii. step is the number of indices to skip between elements (optional, default is 1).

```
my_list = [10, 20, 30, 40, 50]
print(my_list[1:4])      # Output: [20, 30, 40]
print(my_list[::2])      # Output: [10, 30, 50]
```

```
[20, 30, 40]
[10, 30, 50]
```

Indexing and slicing are techniques used to access specific elements or subsequences within a data structure like a string, list, or tuple in Python.

14. Differentiate between rsplit() and split().

- Both split() and rsplit() are string methods in Python that allow you to split a string into a list of substrings based on a specified delimiter. The key difference between split() and rsplit() lies in the direction of splitting and the position of the resulting substrings.
- **split()**: split() is a string method that splits a string into a list of substrings based on a specified delimiter. By default, the split() method splits the string from left to right (i.e., from the beginning).
- **rsplit()**: rsplit() is a string method that splits a string into a list of substrings based on a specified delimiter, but it splits the string from right to left (i.e., from the end).

15. Write the properties of string in python.

Strings in Python have several properties that make them versatile and useful for handling textual data. Here are some key properties of strings:

i. **Immutable**: Strings in Python are immutable, which means they cannot be changed after they are created. Any operation that modifies a string actually creates a new string.

ii. **Sequence of Characters**: A string is a sequence of characters, allowing you to access individual characters by their index. You can iterate over the characters of a string using loops or perform operations on each character individually.

iii. **Concatenation**: Strings can be concatenated using the + operator, which combines two or more strings together. For example, "Hello" + "World" results in the string "HelloWorld".

iv. **Indexing and Slicing**: Strings support indexing and slicing operations. You can access individual characters by their index using square brackets, and you can extract substrings by specifying a range of indices using slicing.

v. **Length**: The len() function can be used to determine the length of a string, which is the total number of characters in the string.

vi. **String Formatting**: Python provides various methods of formatting strings, including using placeholders, such as %s or {}, and the more modern f-string syntax, which allows embedding expressions within curly braces {}.

vii. **String Methods**: Python offers a wide range of built-in string methods that provide various functionalities for manipulating and working with strings. These methods include lower(), upper(), strip(), replace(), split(), and many more.

viii. **Unicode Support**: Strings in Python are Unicode-based, allowing the representation of characters from different writing systems and languages. This enables working with multilingual and international text.

ix. **Escape Characters**: Strings support the use of escape characters, such as \n for a newline, \t for a tab, \" for a double quote, etc. These characters allow representing special characters and formatting within a string.

x. **String Comparison**: Strings can be compared using relational operators (==, !=, <, >, <=, >=). The comparison is based on the lexicographic order of the characters in the strings.

These properties make strings a fundamental and versatile data type in Python, allowing for various operations and manipulations when working with textual data.