**Interview Questions:**

**Python:**

1. Who is the creator of Python?

Solution- Python was created by Guido van Rossum

1. In which year was Python first released?

Solution- Python was created by Guido van Rossum, and first released on February 20, 1991.

1. What inspired the name "Python"?

Solution- Python was created by Guido van Rossum, and first released on February 20, 1991. While you may know the python as a large snake, the name of the Python programming language comes from an old BBC television comedy sketch series called Monty Python's Flying Circus.

1. Explain the characteristics of Python as a general-purpose language.

Solution- Python is a high-level programming language that is widely used for developing software applications and web applications. It is known for its simplicity, flexibility, and ease of use

1. What does it mean for Python to be open-source software?

Solution- Python is developed under an OSI-approved open source license, making it freely usable and distributable, even for commercial use.

1. Differentiate between interpreted and compiled languages.

Solution- Interpreted languages are executed line by line

Examples of interpreted languages include Python, JavaScript, and Ruby

Compiled languages undergo a separate compilation phase where the source code is translated into machine code or bytecode by a compiler.

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

1. Why is Python considered an interpreted language?

Solution-Python is called interpreted language because it executed line by line

1. Describe the concept of interactive programming in Python.

Solution- Interactive programming in Python refers to the process of writing and executing code

1. How is Python used for scripting purposes?

Solution- Python is widely used as a scripting language due to its simplicity, readability, and versatility.

1. What is dynamic programming in Python?

Solution- Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems and solving each subproblem only once, storing the solutions to subproblems in a table (usually an array or dictionary) to avoid redundant computations

1. Name the main built-in data types in Python.

Solution- In Python, the main built-in data types include:

**Integer (int)**

**Float (float)**

**Boolean (bool)**

**String (str)**

**List (list)**

**Tuple (tuple)**

**Set (set)**

**Dictionary (dict)**

1. Explain the difference between mutable and immutable objects in Python.

Solution- **Mutable Objects**:

* Mutable objects can be modified after they are created
* Examples of mutable objects in Python include lists, dictionaries, sets, and objects of custom classes where their attributes can be modified.

**Immutable Objects**:

* Immutable objects, on the other hand, cannot be modified after they are created.
* Examples of immutable objects in Python include integers, floats, strings, tuples, and frozensets.

1. How are variables defined in Python?

Solution- In Python, variables are defined by simply assigning a value to a name using the equal sign (**=**).

1. What are the rules for naming variables in Python?

Solution- In Python, variable names must adhere to certain rules and conventions

1. **Valid Characters**: Variable names can contain letters (both uppercase and lowercase), digits, and underscores (**\_**). They must start with a letter or an underscore, but not with a digit.
2. **Case Sensitivity**: Python is case-sensitive. This means that **my\_variable**, **My\_Variable**, and **MY\_VARIABLE** are all different variables.
3. **Reserved Words**: You cannot use Python's reserved words (keywords) as variable names. Examples of reserved words include **if**, **else**, **for**, **while**, **True**, **False**, **None**, etc.

1. How do you check the type of a variable in Python?

Solution- In Python, you can check the type of a variable using the built-in **type()** function

1. What is the difference between local and global variables in Python?

Solution- **Local Variables**:

* Local variables are defined within a function and are only accessible within that function's scope.
* They are created when the function is called and are destroyed when the function exits.
* Local variables cannot be accessed from outside the function in which they are defined

**Global Variables**:

* Global variables are defined outside of any function and are accessible throughout the entire program.
* They can be accessed and modified from any part of the code, including inside functions.
* Global variables persist throughout the lifetime of the program, until explicitly destroyed.

1. Can you use Python keywords as variable names?

Solution- No, Python keywords cannot be used as variable names. Attempting to use a Python keyword as a variable name will result in a syntax error.

1. Explain the purpose of arithmetic operators in Python.

Solution- Arithmetic operators in Python are used to perform mathematical operations on numerical values. They allow you to manipulate numbers in various ways.

1. Describe the usage of comparison operators in Python.

Solution- Comparison operators in Python are used to compare two values and determine the relationship between them. These operators return Boolean values (**True** or **False**) based on whether the comparison is true or false.

1. How do assignment operators work in Python?

Solution- Assignment operators in Python are used to assign values to variables. They combine the assignment (**=**) operator with other arithmetic or bitwise operators to perform an operation and assign the result to a variable

1. What are logical operators used for in Python?

Solution- Logical operators in Python are used to combine multiple conditions or expressions and determine the overall truth value of the compound expression. Python provides three main logical operators: **and**, **or**, and **not.**

1. Define identity and membership operators in Python.

Solution- **Identity Operators**:

* **is**: The **is** operator checks if two variables refer to the same object in memory. If they do, it returns **True**; otherwise, it returns **False**.
* **is not**: The **is not** operator checks if two variables do not refer to the same object in memory. If they do not, it returns **True**; otherwise, it returns **False**.

**Membership Operators**:

* **in**: The **in** operator checks if a value exists in a sequence (such as a string, list, tuple, or dictionary).
* **not in**: The **not in** operator checks if a value does not exist in a sequence.

1. Explain the concept of bitwise operators.

Solution- Bitwise operators in Python are used to perform operations on individual bits of integers. These operators treat operands as sequences of binary digits (bits) and perform operations on corresponding bits of the operands.

1. What are conditional statements in Python?

Solution- Conditional statements in Python are used to make decisions in code based on certain conditions. They allow you to execute different blocks of code depending on whether a condition is true or false.

1. Differentiate between `if`, `elif`, and `else` statements.

Solution- Use if-else statements when the alternatives are mutually exclusive. This means that if one alternative is true, the other alternatives must be false. Use if-elif-else statements when the alternatives are not mutually exclusive

1. What is the purpose of loops in Python?

Solution- Looping means repeating something over and over until a particular condition is satisfied. A for loop in Python is a control flow statement that is used to repeatedly execute a group of statements as long as the condition is satisfied.

1. How do `for` loops and `while` loops differ?

Solution- The major difference between for loop and while loop is that for loop is used when the number of iterations is known, whereas execution is done in a while loop until the statement in the program is proved wrong.

1. What is exception handling in Python?

Solution- All exception-handling blocks in Python begin with the "try" keyword. It is used to check the code for errors. Programmers write only those codes within this block, which might raise an exception. If the code in the try block is error-free, the try block executes, and the subsequent except block is skipped.

1. Explain the purpose of `try`, `except`, `else`, and `finally` blocks.

* Solution- **try block**: This is where you place the code that might raise an exception. It is essentially the "attempt" part of the code. If an exception occurs within this block, Python immediately jumps to the corresponding **except** block.
* **except block**: This block catches and handles exceptions that occur within the **try** block. You can specify which exceptions to catch, or you can use a general **except** block to catch all exceptions. If an exception is raised in the **try** block, Python looks for an appropriate **except** block to handle it.
* **else block**: This block is optional and is executed if the code in the **try** block does not raise any exceptions. It's often used for code that should only run if no exceptions occurred. If any exceptions are raised, the **else** block is skipped.
* **finally block**: This block is also optional and is executed regardless of whether an exception occurred or not. It's typically used for code that should run no matter what, such as releasing external resources like files or database connections. The **finally** block is executed even if there is a **return** statement in the **try** block.

1. How do you raise custom exceptions in Python?

Solution- In Python, you can raise custom exceptions using the **raise** statement. This allows you to create your own exception classes to represent specific errors or exceptional conditions in your code.