Q1. Why do we call Python as a general purpose and high-level programming language?

Python is considered a general-purpose and high-level programming language due to several key characteristics:

General-purpose: Python is designed to be versatile and adaptable, allowing it to be used for a wide range of applications and domains. It provides robust support for various programming paradigms, including procedural, object-oriented, and functional programming. Whether you're working on web development, data analysis, scientific computing, artificial intelligence, or scripting tasks, Python can be effectively utilized.

High-level: Python is a high-level language, which means it abstracts away many low-level details and provides a more user-friendly syntax and semantics. It offers a rich set of built-in data structures, such as lists, dictionaries, and tuples, making it easier to express complex algorithms and manipulate data without needing to write extensive low-level code. This high-level nature simplifies development, enhances readability, and boosts productivity.

Readability and simplicity: Python is renowned for its emphasis on code readability. Its syntax is designed to be clean and intuitive, favoring human readability over machine optimization. Python enforces indentation and uses English-like keywords, making the code more readable and reducing syntax errors. This simplicity and readability make Python an excellent choice for beginners and maintainability in large-scale projects.

Extensive standard library: Python comes w

Q2. Why is Python called a dynamically typed language?

Python is called a dynamically typed language because the type of a variable is determined and checked at runtime, rather than at compile time. In other words, variables in Python can hold values of different types, and their type can be changed during the execution of the program.

In a dynamically typed language like Python, you can assign different types of values to the same variable without explicitly specifying its type. For example, you can assign an integer value to a variable and later assign a string value to the same variable. This flexibility allows for more concise and flexible code.

Here's an example in Python:

```
x = 5  # x is an integer

print(x)
```

```
x = "Hello"  # x is now a string

print(x)
```

In this example, the variable x initially holds an integer value (5), and later it is assigned a string value ("Hello"). The Python interpreter dynamically determines the type of x based on the value assigned to it at runtime.

Dynamic typing can be powerful, as it allows for more flexible programming and quicker prototyping. However, it also requires careful attention to type compatibility and can potentially lead to runtime errors if variables are not used consistently.

In contrast, statically typed languages, such as C++ or Java, require variables to have a specific type declared at compile time. The type of a variable is fixed, and it cannot be changed during runtime. Statically typed languages provide stricter type checking, which can catch certain errors at compile time, but they can be more verbose and less flexible in terms of variable usage.ith a vast standard library that provides numerous pre-built modules and functions for common programming tasks. The standard library covers a broad range of areas, such as file I/O, networking, regular expressions, math, and more. This extensive collection of modules enables developers to leverage existing code, saving time and effort in implementing functionality.

Q3. List some pros and cons of Python programming language?

Python is a popular programming language known for its simplicity, readability, and versatility. Here are some pros and cons of Python:

Pros of Python:

Readability: Python's syntax is designed to be easy to read and understand, making it an excellent choice for beginners and experienced programmers alike. It emphasizes code readability, which leads to more maintainable and less error-prone code.

Large and active community: Python has a vast and active community of developers who contribute to its libraries, frameworks, and tools. This means you can easily find support, resources, and solutions to your programming problems.

Extensive libraries and frameworks: Python provides a wide range of libraries and frameworks, such as NumPy, Pandas, Django, and Flask, which help streamline development tasks and accelerate the process. These libraries cover

various domains like data analysis, web development, machine learning, and more.

Cross-platform compatibility: Python programs can run on multiple platforms, including Windows, macOS, Linux, and even mobile devices, with minimal or no modifications. This portability makes it convenient for developing applications that need to run on different operating systems.

Integration capabilities: Python has strong integration capabilities, allowing developers to easily integrate it with other programming languages such as C/C++, Java, and .NET. This enables leveraging existing codebases and taking advantage of Python's simplicity and ease of use.

Rapid development: Python's concise syntax and extensive standard library enable developers to write code quickly and efficiently. It emphasizes code reusability, which results in faster development cycles and reduced time-to-market for projects.

Cons of Python:

Performance limitations: Compared to lower-level languages like C or C++, Python can be slower in terms of execution speed. This is due to its interpreted nature, automatic memory management, and other high-level abstractions. However, for most applications, Python's performance is sufficient, and performance-critical code can be optimized using techniques like leveraging libraries written in other languages.

Global Interpreter Lock (GIL): Python's Global Interpreter Lock is a mechanism that prevents multiple native threads from executing Python bytecodes simultaneously. This limitation can impact the performance of multi-threaded programs, particularly in CPU-bound scenarios. However, Python offers alternative solutions such as multiprocessing or asynchronous programming to overcome this limitation.

Mobile app development: While Python can be used for developing mobile applications using frameworks like Kivy or BeeWare, it is not as commonly used in mobile app development compared to languages like Java or Swift. Native mobile development is generally favored for performance and platform-specific features.

Mobile platform limitations: Python's support for mobile platforms, especially iOS, has some limitations. Apple's App Store guidelines and restrictions make it challenging to use Python for certain types of applications or impose additional complexities.

Version compatibility: Python has undergone significant updates over the years, with major releases like Python 2 and Python 3. Although Python 3 is the recommended version, there are still existing codebases and libraries that rely on Python 2, which can create compatibility issues when working with different versions.

Overall, Python's advantages, such as its readability, extensive libraries, and supportive community, make it an excellent choice for various applications. However, it's essential to consider the potential drawbacks, such as performance limitations and compatibility issues, depending on the specific requirements of your project.

Q4. In what all domains can we use Python?

Python is a versatile programming language that can be used in a wide range of domains. Here are some of the main domains where Python is commonly used:

Web Development: Python is widely used for web development. Popular web frameworks like Django and Flask are built using Python, allowing developers to create robust and scalable web applications.

Data Science and Machine Learning: Python has become the de facto language for data science and machine learning. Libraries like NumPy, Pandas, and SciPy provide powerful tools for data manipulation, analysis, and scientific computing. Additionally, popular machine learning frameworks such as TensorFlow and PyTorch are primarily written in Python.

Scientific Computing: Python is extensively used in scientific computing due to its ease of use and availability of numerous libraries. Scientists and researchers can utilize libraries like Matplotlib and SciPy to perform complex scientific calculations, visualize data, and conduct simulations.

Artificial Intelligence: Python is widely adopted in the field of artificial intelligence (AI) and natural language processing (NLP). Libraries such as NLTK (Natural Language Toolkit) and spaCy provide functionalities for processing and analyzing text data, while frameworks like Keras and scikit-learn enable the development and deployment of AI models.

Automation and Scripting: Python's simplicity and readability make it an excellent choice for automation and scripting tasks. It is commonly used to write scripts for automating repetitive tasks, system administration, and workflow automation.

Game Development: Python is utilized in game development, particularly for prototyping and scripting. Libraries like Pygame offer tools and functionality

for creating 2D games, while engines such as Panda3D and Unity allow developers to build more complex games using Python.

Internet of Things (IoT): Python is widely used in the IoT domain due to its simplicity and flexibility. It can be employed for building IoT applications, integrating devices, and analyzing data collected from sensors and connected devices.

Networking and Network Automation: Python's extensive networking libraries, such as requests and sockets, make it a popular choice for network programming and automation tasks. It enables developers to build network applications, interact with APIs, and automate network-related tasks.

These are just a few examples of the domains where Python is commonly used. Python's versatility, extensive libraries, and active community support make it applicable in many other areas, including finance, cybersecurity, system administration, and more.

Q5. What are variable and how can we declare them?

In programming, a variable is a named storage location used to hold a value that can change during the execution of a program. It represents a piece of memory that can store different types of data, such as numbers, characters, or more complex structures.

The process of creating a variable in a programming language is known as variable declaration. The exact syntax and rules for declaring variables can vary depending on the programming language, but I'll provide you with a general idea.

Here are the basic steps to declare a variable:

Choose a name for the variable: The name should be meaningful and descriptive, indicating the purpose or content of the variable. It should follow the naming conventions of the programming language you are using (e.g., avoiding spaces and special characters).

Determine the data type: Variables have a data type associated with them, which defines the kind of data they can store. Common data types include integers, floating-point numbers, characters, strings, booleans, etc. The choice of data type depends on the nature of the data you want to store.

Write the declaration statement: In most programming languages, you declare a variable by specifying its name and the data type. Here's a general syntax:

rust

Copy code

data_type variable_name;

For example, in many programming languages, to declare an integer variable named age, you would write:

arduino

Copy code

int age;

Optionally, assign an initial value: After declaring a variable, you can optionally assign an initial value to it. This is done using an assignment operator (=). The initial value should match the data type of the variable. For example:

arduino

Copy code

int age = 25;

If you don't assign an initial value, the variable will contain a default value specific to its data type (e.g., 0 for numbers, empty string for strings, false for booleans).

It's important to note that the specific syntax and rules for variable declaration can vary across programming languages. Additionally, some languages have more advanced features, such as type inference, where the data type can be automatically determined based on the assigned value.

Q6. How can we take an input from the user in Python?

In Python, you can take input from the user using the input() function. The input() function reads a line of text entered by the user and returns it as a string.

Here's an example of how to use the input() function:

python

Copy code

name = input("Enter your name: ")

print("Hello, " + name + "!")

age = input("Enter your age: ")

```
print("You are " + age + " years old.")
```

In this example, the input() function is used to prompt the user for their name and age. The user's input is stored in the variables name and age, respectively. The values entered by the user are then used in the subsequent print statements.

When you run this code, it will display the prompts and wait for the user to enter their input. After the user enters their input and presses Enter, the program will proceed to the next line of code.

Note that the input() function always returns a string, even if the user enters a number. If you need to perform numerical operations on the input, you may need to convert the input to the appropriate data type using functions like int() or float().

Q7. What is the default datatype of the value that has been taken as an input using input() function?

In Python, the input() function is used to take user input as a string. Therefore, the default datatype of the value returned by the input() function is always a string. Regardless of whether the user enters a number, text, or any other type of input, it will be treated as a string by default. If you want to use the input value as a different datatype, such as an integer or a float, you need to explicitly convert it using appropriate conversion functions like int() or float().

Q8. What is type casting?

Type casting, also known as type conversion, is the process of converting one data type into another. It allows you to treat a variable as a different type temporarily, allowing for operations or assignments that are not normally allowed between the original and target types.

In programming, type casting is particularly common in statically-typed languages like C, C++, Java, and others, where variables have specific data types determined at compile-time. The need for type casting arises when you want to perform operations on variables of different types, or when you want to store a value of one type into a variable of another type.

Type casting can be divided into two categories: implicit casting (also known as widening or automatic casting) and explicit casting (also known as narrowing or manual casting).

Implicit casting: This type of casting occurs automatically by the compiler when there is no loss of information or precision. For example, converting an integer

to a float or a float to a double. Since there is no loss of data, the conversion is done implicitly without any explicit code from the programmer.

Explicit casting: This type of casting requires the programmer to explicitly specify the conversion using special syntax. It is typically used when there is a possibility of data loss or when converting from a larger type to a smaller type. For example, converting a double to an int or an int to a char. Explicit casting involves placing the desired target type in parentheses before the variable or value being cast.

Q9. Can we take more than one input from the user using single input() function? If yes, how? If no, why?

No, we cannot take multiple inputs from the user using a single input() function in Python. The input() function is designed to read a single line of user input as a string. It waits for the user to enter a value and press the "Enter" key, and then it returns the entered value as a string.

If you want to take multiple inputs from the user, you can use the input() function multiple times, each time prompting the user for a different input. For example:

python

Copy code

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

age = input("Enter your age: ")


print("Your name is", name)

print("Your age is", age)
```

In the above code, we are using the input() function twice to get the user's name and age separately. Each time the input() function is called, it waits for the user to enter a value.

If you need to process multiple inputs on a single line, you can use the split() method to split the input string into multiple values. For example:

python

Copy code

```
values = input("Enter multiple values separated by space: ").split()
```

```
print("You entered:", values)
```

In this case, the user can enter multiple values separated by spaces, and the split() method is used to split the input string into a list of individual values.

10. What are keywozrds?

Keywords are words or phrases that serve as important descriptors or identifiers for a particular topic, concept, or content. In the context of search engines and digital marketing, keywords are the terms that users enter into search engines to find relevant information, products, or services. They play a crucial role in search engine optimization (SEO) and pay-per-click (PPC) advertising campaigns.

Keywords help search engines understand the content of web pages and match them with relevant search queries. By including relevant keywords in web page titles, headings, meta tags, and throughout the content, website owners can increase the likelihood of their pages appearing in search engine results pages (SERPs) when users search for those keywords.

Keywords can be broad or specific, depending on the intent of the user and the context of the content. For example, a broad keyword like "shoes" may have a high search volume but could be more competitive to rank for, while a specific keyword like "running shoes for women" may have a lower search volume but a higher likelihood of attracting targeted traffic.

Keyword research is an important part of digital marketing strategies, as it helps identify the most relevant and effective keywords to target in order to optimize web pages and improve organic search rankings. It involves analyzing search volume, competition, and relevance to select the best keywords for a particular website or campaign.

Q11. Can we use keywords as a variable? Support your answer with reason.

No, keywords cannot be used as variables in most programming languages. Keywords are reserved words that have predefined meanings and functionalities in the language syntax. They serve specific purposes and cannot be redefined or used as identifiers for variables or other elements in the code.

For example, in Python, keywords like "if," "for," "while," "def," and "class" have specific roles in the language syntax. If you try to use them as variable names, you will encounter a syntax error because the language expects these keywords to be used in a particular way.

Using keywords as variables would lead to ambiguity and confusion in the code. It would make it difficult to understand the code's intention and would violate the rules and conventions of the programming language. Therefore, it is generally not allowed to use keywords as variables in programming languages.

Q12. What is indentation? What's the use of indentaion in Python?

In programming, indentation refers to the spacing or tabs used at the beginning of a line of code to indicate its level of nesting within a block of code. It is commonly used to structure and organize code in a readable and logical manner.

In Python, indentation is not just a matter of style but is actually a part of the language's syntax. Python uses indentation to define blocks of code, such as loops, conditional statements, and function definitions. By convention, four spaces or one tab is typically used for each level of indentation.

The use of indentation in Python serves several purposes:

Readability: Indentation enhances the readability of code by providing visual cues to the structure of the program. It helps to easily identify the hierarchy and relationships between different sections of code.

Block delimiters: In many programming languages, curly braces ({}) or keywords are used to define blocks of code. In Python, indentation serves as the block delimiter. The lines of code indented at the same level are considered to be part of the same block.

Code execution: Indentation is crucial for Python's interpreter to determine which statements are grouped together in a block. The interpreter uses indentation to understand the flow of control and execute the code accordingly.

Here's an example to illustrate the use of indentation in Python:

python

Copy code

```
if x > 5:
    print("x is greater than 5")  # This line is indented under the "if" statement
    print("Hello")                # This line is also indented under the "if" statement
else:
    print("x is not greater than 5")  # This line is indented under the "else" statement
```

```
    print("World")                # This line is also indented under the "else"
statement
```

In the above example, the indentation helps define the two blocks of code: one for the "if" statement and another for the "else" statement. The indented lines within each block are executed based on the condition being satisfied or not.

It's important to note that consistent indentation is crucial in Python. Mixing different indentation styles or inconsistent indentation levels can lead to syntax errors or unexpected behavior in the program.

Q13. How can we throw some output in Python?

In Python, you can throw output or display information using the print() function. The print() function takes one or more arguments and displays them as output on the console.

Q14. What are operators in Python?

In Python, operators are special symbols or characters that perform specific operations on one or more operands (values, variables, or expressions) and produce a result. Python supports a variety of operators, which can be classified into different categories based on the type of operation they perform. Here are some commonly used operators in Python:

Arithmetic Operators:

Addition (+)

Subtraction (-)

Multiplication (*)

Division (/)

Floor Division (//)

Modulo (%)

Exponentiation (**)

Assignment Operators:

Assignment (=)

Addition Assignment (+=)

Subtraction Assignment (-=)

Multiplication Assignment (*=)

Division Assignment (/=)

Modulo Assignment (%=)

Floor Division Assignment (//=)

Exponentiation Assignment (**=)

Comparison Operators:

Equal to (==)

Not equal to (!=)

Greater than (>)

Less than (<)

Greater than or equal to (>=)

Less than or equal to (<=)

Logical Operators:

Logical AND (and)

Logical OR (or)

Logical NOT (not)

Bitwise Operators:

Bitwise AND (&)

Bitwise OR (|)

Bitwise XOR (^)

Bitwise NOT (~)

Left Shift (<<)

Right Shift (>>)

Membership Operators:

Membership (in)

Not in (not in)

Identity Operators:

Identity (is)

Not identity (is not)

These operators are used in various contexts to perform different operations, such as arithmetic calculations, assignment of values, logical operations, bitwise operations, and more.

Q15. What is difference between / and // operators?

The '/' and '//' operators are both used for division in Python, but they behave differently based on the types of operands involved.

The '/' operator performs regular division and returns a floating-point result, regardless of the operand types. It calculates the quotient and includes the decimal part if necessary. Here's an example:

7 / 2 # returns 3.5

In this case, the result is 3.5 because 7 divided by 2 is 3.5.

On the other hand, the '//' operator performs integer division, also known as floor division. It returns the largest integer that is less than or equal to the division result. The result is always of integer type, regardless of the operand types. Here's an example:

7 // 2 # returns 3

In this case, the result is 3 because 7 divided by 2 is 3.5, and the '//' operator discards the decimal part, giving the floor value.

To summarize, the '/' operator performs regular division and returns a floating-point result, while the '//' operator performs integer division and returns an integer result by discarding the decimal part.

Q16. Write a code that gives following as an output.

```

iNeuroniNeuroniNeuroniNeuron

```

Here's a code for desired output:

python

Copy code

string = "iNeuron"

output = string * 4 + "Neuron"

```
print(output)
```

Q17. Write a code to take a number as an input from the user and check if the number is odd or even.

Certainly! Here's a Python code snippet that takes a number as input from the user and checks if it's odd or even:

```
number = int(input("Enter a number: "))

if number % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

Q18. What are boolean operator?

Boolean operators are operators that are used to manipulate or evaluate Boolean values. Boolean values are binary values that can be either true or false. The three basic Boolean operators are:

AND Operator: The AND operator returns true if both operands are true. It is represented by the symbol "&&" or the word "AND". For example, in the expression "A && B", the result will be true only if both A and B are true.

OR Operator: The OR operator returns true if at least one of the operands is true. It is represented by the symbol "||" or the word "OR". For example, in the expression "A || B", the result will be true if either A or B or both are true.

NOT Operator: The NOT operator returns the opposite of the operand. It is represented by the symbol "!" or the word "NOT". For example, in the expression "!A", the result will be true if A is false, and false if A is true.

These Boolean operators can be combined to form complex expressions and are commonly used in programming and logic to make decisions and control the flow of execution based on conditions.

Q18. What are boolean operator?

Boolean operators are operators that are used to manipulate or evaluate Boolean values. Boolean values are binary values that can be either true or false. The three basic Boolean operators are:

AND Operator: The AND operator returns true if both operands are true. It is represented by the symbol "&&" or the word "AND". For example, in the expression "A && B", the result will be true only if both A and B are true.

OR Operator: The OR operator returns true if at least one of the operands is true. It is represented by the symbol "||" or the word "OR". For example, in the expression "A || B", the result will be true if either A or B or both are true.

NOT Operator: The NOT operator returns the opposite of the operand. It is represented by the symbol "!" or the word "NOT". For example, in the expression "!A", the result will be true if A is false, and false if A is true.

These Boolean operators can be combined to form complex expressions and are commonly used in programming and logic to make decisions and control the flow of execution based on conditions.

Q19. What will the output of the following?

```

1 or 0

0 and 0

True and False and True

1 or 0 or 0
```

The output of the given expressions would be as follows:

1 or 0: This expression evaluates to 1 (True) because the "or" operator returns the first operand that is True. In this case, 1 is True, so the expression results in 1.

0 and 0: This expression evaluates to 0 (False) because the "and" operator returns the first operand that is False. In this case, both operands are False, so the expression results in 0.

True and False and True: This expression evaluates to False because the "and" operator returns False if any of the operands are False. In this case, the second operand is False, so the expression results in False.

1 or 0 or 0: This expression evaluates to 1 (True) because the "or" operator returns the first operand that is True. In this case, the first operand is 1, so the expression results in 1.

Q20. What are conditional statements in Python?

Conditional statements in Python are used to make decisions in a program based on certain conditions. They allow the program to execute different blocks of code depending on whether a given condition is true or false. The two main types of conditional statements in Python are the "if" statement and the "if-else" statement.

The "if" statement: It allows the program to execute a block of code only if a specified condition is true.

The "if-else" statement: It extends the "if" statement by providing an alternative block of code to be executed when the condition is false.

Additionally, there is also the "if-elif-else" statement, which allows for multiple conditions to be checked. It is used when there are more than two possible outcomes. The basic syntax is as follows:

```
if condition1:
    # code to be executed if condition1 is true
elif condition2:
    # code to be executed if condition1 is false and condition2 is true
else:
    # code to be executed if both condition1 and condition2 are false
```

Conditional statements provide a powerful way to control the flow of execution in a program and make it more flexible and responsive to different situations.

Q21. What is use of 'if', 'elif' and 'else' keywords?

The keywords 'if', 'elif' (short for "else if"), and 'else' are control flow statements used in programming languages to implement conditional logic. These keywords are commonly used in languages like Python.

Here's an explanation of each keyword:

'if': The 'if' statement is used to perform an action or execute a block of code if a certain condition is true. It allows you to specify a condition, and if the condition evaluates to true, the code block associated with the 'if' statement is executed.

'elif': The 'elif' statement is used to specify additional conditions to be checked if the initial 'if' condition is false. It allows you to chain multiple conditions

together and provide alternative actions or code blocks based on different conditions. You can have zero or multiple 'elif' statements following an 'if' statement.

'else': The 'else' statement is used as a catch-all condition that executes a block of code if none of the previous conditions (specified by 'if' or 'elif') evaluate to true. It provides an alternative action when all the preceding conditions are false.

Here's an example in Python to illustrate the usage of these keywords:

python

Copy code

```
age = 25

if age < 18:
    print("You are a minor.")
elif age >= 18 and age < 65:
    print("You are an adult.")
else:
    print("You are a senior citizen.")
```

Q22. Write a code to take the age of person as an input and if age >= 18 display "I can vote". If age is < 18 display "I can't vote"

Certainly! Here's a simple code snippet in Python that takes the age of a person as input and displays whether they can vote or not:

python

Copy code

```
age = int(input("Enter your age: "))

if age >= 18:
    print("I can vote")
else:
    print("I can't vote")
```

Q23. Write a code that displays the sum of all the even numbers from the given list.

```
numbers = [12, 75, 150, 180, 145, 525, 50]
```

```
numbers = [12, 75, 150, 180, 145, 525, 50]
sum_of_evens = 0
for num in numbers:
    if num % 2 == 0:
        sum_of_evens += num
print("Sum of even numbers:", sum_of_evens)
```

Q24. Write a code to take 3 numbers as an input from the user and display the greatest no as output.

```
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))
# Finding the greatest number
if num1 >= num2 and num1 >= num3:
    greatest = num1
elif num2 >= num1 and num2 >= num3:
    greatest = num2
else:
    greatest = num3
# Displaying the greatest number
print("The greatest number is:", greatest)
```

Q25. Write a program to display only those numbers from a list that satisfy the following conditions

- The number must be divisible by five

- If the number is greater than 150, then skip it and move to the next number

- If the number is greater than 500, then stop the loop

```
numbers = [12, 75, 150, 180, 145, 525, 50]
```

```
numbers = [12, 75, 150, 180, 145, 525, 50]
for number in numbers:
    if number > 500:
        break  # Stop the loop if the number is greater than 500
    if number > 150:
        continue  # Skip numbers greater than 150 and move to the next number
    if number % 5 == 0:
        print(number)
```