Introduction to python

Q 1. What is Python? Why is it so popular?

Python is a high-level, interpreted programming language that is used for a variety of tasks such as web development, data analysis, machine learning, artificial intelligence, and scientific computing. It was first released in 1991 by Guido van Rossum and is now maintained by the Python Software Foundation.

Python is popular for several reasons. First, it is easy to learn and has a relatively simple syntax, making it accessible to beginners. Second, it is open source, which means it is free to use and has a large and active community of developers who contribute to its development and create a wealth of libraries and tools that extend its capabilities. Third, it is highly versatile and can be used for a wide range of applications, from small scripts to large-scale applications. Fourth, it is highly readable and expressive, allowing developers to write code that is easy to understand and maintain. Finally, it has a strong ecosystem of libraries and frameworks that enable developers to quickly and easily build complex applications.

Q 2. What are the key features of Python?

Python has several key features that make it a popular choice among developers. Here are some of the most important ones:

1. Simple and easy to learn: Python has a straightforward syntax and is easy to learn for beginners. Its code is highly readable and can be easily understood by people with little to no programming experience.
2. Interpreted: Python is an interpreted language, which means that code can be executed directly without the need for compilation. This allows for faster development cycles and easier debugging.
3. Dynamic and high-level: Python is a dynamic language, which means that data types are determined at runtime, making it easier to write and read code. It is also a high-level language, meaning that it has a level of abstraction that makes it easier to program complex applications.
4. Portable: Python can be run on a wide range of platforms, including Windows, Linux, and macOS. This makes it easy to develop applications that can run on multiple systems without needing to be modified.
5. Large standard library: Python has a large standard library that provides a wide range of modules and functions for tasks such as web development, data analysis, and machine learning. This reduces the need for developers to write code from scratch and speeds up the development process.
6. Extensible: Python can be easily extended with third-party libraries and modules, which can be easily installed using the package manager pip. This allows developers to take advantage of a wide range of tools and frameworks to build complex applications.

7. Object-oriented: Python supports object-oriented programming, which allows for the creation of reusable code and promotes code organization and readability.
8. Strong community support: Python has a large and active community of developers who contribute to its development, create libraries, and provide support to others. This ensures that Python remains a popular and well-supported language for years to come.

Q 3. What type of language is python? Programming or scripting?

Python is a general-purpose programming language that can be used for a variety of tasks, from small scripts to large-scale applications. It is not limited to a specific area of development and is widely used for tasks such as web development, data analysis, machine learning, scientific computing, and more.

While Python is often used as a scripting language due to its interpreted nature and easy-to-use syntax, it is not limited to just scripting tasks. Python is a fully-featured programming language that supports object-oriented, functional, and procedural programming paradigms, making it a versatile tool for developers.

In summary, Python is both a programming language and a scripting language, depending on the context in which it is used.

Q 4. What is pep 8?

PEP 8 is a coding style guide for Python that provides a set of recommendations and best practices for writing readable and maintainable code. PEP stands for Python Enhancement Proposal, which is a design document that proposes changes or additions to Python's syntax or standard library.

PEP 8 provides guidelines on how to format code, naming conventions, code structure, and more. Some of the key recommendations in PEP 8 include:

- Use four spaces for indentation

- Limit line length to 79 characters

- Use lowercase with underscores for variable and function names

- Use CamelCase for class names

- Surround operators with spaces

- Use docstrings to document code

- Avoid extraneous whitespace

By following the recommendations in PEP 8, developers can write code that is easy to read and understand, making it easier to maintain and update in the future. It is widely considered as the standard for Python code style and is used by many organizations and open-source projects.

Q 5. Python an interpreted language. Explain

Python is an interpreted language, which means that the code is executed directly without the need for compilation. When you run a Python program, the source code is first parsed by the interpreter, which translates it into bytecode, a low-level, platform-independent representation of the code.

The bytecode is then executed by the Python virtual machine (PVM), which is a software layer that runs on top of the operating system. The PVM reads the bytecode and executes the corresponding instructions, which are then translated into machine code by the system's CPU and executed.

The main advantage of using an interpreted language like Python is that it allows for faster development cycles and easier debugging. With an interpreted language, you can run and test your code as you write it, without needing to compile it first. This makes it easier to catch errors and fix them quickly, without having to go through the time-consuming process of compiling and linking your code.

Another advantage of using an interpreted language is that it makes it easier to write cross-platform code. Since the bytecode generated by the interpreter is platform-independent, you can write code that can run on multiple platforms without needing to modify it.

In summary, Python is an interpreted language, which means that it executes code directly without the need for compilation. This allows for faster development cycles, easier debugging, and cross-platform compatibility.

Q 6. How is memory managed in python?

Memory management in Python is handled automatically by the language's runtime system. Python uses a system of dynamic memory allocation, which means that the memory is allocated and deallocated as needed during program execution.

Python uses a technique called reference counting to keep track of the memory used by objects. Each object in Python has a reference count, which keeps track of the number of references to that object. When an object's reference count drops to zero, it is automatically deallocated by the Python interpreter's garbage collector.

In addition to reference counting, Python also uses a garbage collector that runs periodically to identify and deallocate objects that are no longer in use. The garbage collector works by identifying objects that have no active references and deallocating them from memory.

Python also supports a mechanism called the "Python Memory Manager," which is responsible for allocating and deallocating memory for Python objects. The memory manager uses a combination of techniques, including memory pooling, to minimize the overhead of memory allocation and improve performance.

Overall, memory management in Python is handled automatically by the language's runtime system, which uses a combination of techniques, including reference counting, garbage collection, and memory pooling, to optimize performance and ensure efficient use of system resources.

Q 7. What is namespace in python?

A namespace in Python is a mapping between names and objects. It is a way of organizing names in a program, so that names that are used in different parts of the program do not clash with each other.

Python has several different types of namespaces, including:

1. Global namespace: This is the namespace that contains all the names defined at the top level of a module. These names can be accessed from any part of the program that imports the module.
2. Local namespace: This is the namespace that contains the names defined inside a function. These names are only accessible within the function and are deleted when the function returns.
3. Built-in namespace: This is the namespace that contains all the built-in names of Python, such as "print" and "len".

When you use a name in Python, the interpreter first looks for it in the local namespace. If it is not found there, it looks for it in the global namespace. If it is not found there either, it looks for it in the built-in namespace.

You can create your own namespaces in Python using modules or classes. Modules provide a way to organize related names in a program, while classes provide a way to organize names within an object.

Overall, a namespace in Python is a way of organizing names in a program, so that they do not clash with each other and can be accessed from different parts of the program as needed.

Q 8. What is PYTHONPATH?

PYTHONPATH is an environment variable in Python that specifies additional directories to be searched when importing modules. When you import a module in Python, the interpreter looks for it in the directories specified in the sys.path list.

The sys.path list is initialized with several directories, including the directory containing the script that is being executed and the standard library directories. However, you may need to import modules from other directories, such as custom libraries or third-party packages. This is where the PYTHONPATH environment variable comes in.

By setting the PYTHONPATH environment variable, you can add additional directories to the sys.path list, so that Python can find modules located in those directories. The

PYTHONPATH variable is a list of directory names, separated by colons on Unix-based systems or semicolons on Windows systems.

Q 9. Is python case sensitive?

Yes, Python is a case-sensitive language. This means that it distinguishes between identifiers (such as variable names, function names, and class names) that have different capitalization.

For example, in Python, the variables "myVar" and "myvar" are considered to be two different variables. Similarly, the function names "myFunc" and "myfunc" would be treated as two separate functions.

It is important to be consistent with the capitalization of identifiers in your code, to avoid syntax errors and to make your code more readable. It is a common convention in Python to use lowercase letters for variable and function names, and uppercase letters for class names.

Overall, Python is a case-sensitive language, which means that it distinguishes between identifiers with different capitalization.

Q 10. Is indentation required in python?

Yes, indentation is required in Python. In fact, indentation is used to indicate the grouping of statements in Python, unlike other programming languages that use curly braces or other symbols for this purpose.

In Python, the standard convention is to use four spaces for indentation. When you start a new block of code (such as a function, loop, or conditional statement), you should indent all of the statements in that block by four spaces. When you reach the end of the block, you should unindent back to the previous level of indentation.

It is important to be consistent with your indentation in Python, since even a small deviation from the standard convention can cause syntax errors. Many code editors and IDEs have features that automatically format your code with the correct indentation, which can help you to avoid mistakes.

Overall, indentation is required in Python to indicate the grouping of statements in blocks of code, and it is important to be consistent with your indentation to avoid syntax errors.

Q 11. What are reserved words?

Reserved words (also known as keywords) are words in a programming language that have a special meaning and are reserved for specific purposes. These words cannot be used as variable names, function names, or any other identifiers in a program, since they are already used by the language itself for specific operations.

In Python, there are a total of 35 reserved words, which are: False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield.

These words are used for various purposes in Python, such as defining control structures (such as "if", "while", and "for"), defining functions (such as "def" and "lambda"), and defining variable scopes (such as "global" and "nonlocal").

It is important to remember that you cannot use reserved words as variable names or any other identifiers in your program, since doing so will result in a syntax error.

Q 12. What are the different data types available in python?

Python supports several built-in data types, including:

1. Numeric types: -
   - int (signed integers)
   - float (floating-point numbers)
   - complex (complex numbers)
2. Sequence types: -
   - list (ordered, mutable sequences)
   - tuple (ordered, immutable sequences)
   - range (immutable sequences of numbers)
3. Text type:
   - str (string of characters)
4. Mapping type:
   - dict (unordered, mutable mappings of key-value pairs)
5. Set types:
   - set (unordered, mutable collection of unique elements)
   - frozenset (unordered, immutable collection of unique elements)
6. Boolean type:
   - bool (logical values of True or False)
7. Binary types:
   - bytes (immutable sequence of bytes)
   - bytearray (mutable sequence of bytes)
   - memoryview (memory buffer interface)

Each of these data types has its own set of operations and methods for manipulating and working with data. It's worth noting that Python is dynamically typed, which means that the data type of a variable is determined at runtime based on the value that it holds.

Q 13. What is indexing and slicing explain with example.

Indexing and slicing are techniques used to access specific elements of a sequence (such as a string, list, or tuple) in Python.

Indexing:

Indexing refers to accessing a single element of a sequence using its position (or index) within the sequence. In Python, indexing starts at 0 for the first element, and continues sequentially for each subsequent element.

For example, consider the following string:

my_string = "Hello, world!"

To access the first character of the string (which is "H"), we can use indexing as follows:

first_char = my_string[0]

This sets the value of `first_char` to the first character of the string, which is "H". Similarly, to access the second character (which is "e"), we can use:

second_char = my_string[1]

This sets the value of `second_char` to the second character of the string, which is "e". We can continue to use indexing to access any individual element of the string.

Slicing:

Slicing refers to accessing a range of elements from a sequence. In Python, slicing uses the colon (:) operator to specify a range of indices to include.

For example, consider the following list:

my_list = [1, 2, 3, 4, 5]

To access a subset of this list (such as the second, third, and fourth elements), we can use slicing as follows:

subset = my_list[1:4]

This sets the value of `subset` to a new list containing the second, third, and fourth elements of `my_list`, which are `[2, 3, 4]`. The slice notation `1:4` specifies the range of indices to include, starting at index 1 (the second element) and continuing up to, but not including, index 4 (the fifth element).

Slicing can also be used to access a range of characters from a string. For example, consider the following string:

my_string = "Hello, world!"

To extract the substring "world" from this string, we can use slicing as follows:substring = my_string[7:12]

This sets the value of `substring` to the characters of `my_string` starting at index 7 (the "w" character) and continuing up to, but not including, index 12 (the "d" character). The resulting substring is "world".

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

Both `rsplit()` and `split()` are string methods in Python that can be used to split a string into a list of substrings based on a specified delimiter. However, there is one key difference between these methods:

`split()` method splits the string from the beginning (left) based on the delimiter, while `rsplit()` method splits the string from the end (right) based on the delimiter.

Here's a more detailed explanation of each method:

split():

- Splits the string from the beginning (left) based on the delimiter.

- Returns a list of substrings.

- By default, splits the string on whitespace (spaces, tabs, and newlines).

- Takes an optional argument `sep` to specify the delimiter. For example: `my_string.split(‘,’)` splits the string on commas.

Example:

```
my_string = "apple,banana,orange"

split_list = my_string.split(',')  # split the string on commas

print(split_list)
```

Output: ['apple', 'banana', 'orange']

rsplit():

- Splits the string from the end (right) based on the delimiter.

- Returns a list of substrings.

- By default, splits the string on whitespace (spaces, tabs, and newlines).

- Takes an optional argument `sep` to specify the delimiter. For example: `my_string.rsplit(‘,’, 1)` splits the string on the last comma.

Example:

```
my_string = "apple,banana,orange"

rsplit_list = my_string.rsplit(',', 1)  # split the string on the last comma
```

print(rsplit_list)

Output: ['apple,banana', 'orange']

In this example, `rsplit()` splits the string on the last comma and returns a list containing two elements: the first element is "apple,banana" (which is everything before the last comma), and the second element is "orange" (which is everything after the last comma).

Q 15. Write the properties of string in python.

In Python, strings are a sequence of characters, and they have several properties:

1. Immutable: Once a string is created, it cannot be modified. Any operation that appears to modify a string actually creates a new string.
2. Ordered: The characters in a string are ordered, meaning each character has a specific index that can be used to access it.
3. Iterable: You can iterate over the characters in a string using a loop or c omprehension.
4. Concatenation: Strings can be concatenated using the `+` operator.
5. Slicing: You can extract a part of a string by specifying a range of indices using slicing notation `[:]`.
6. Indexing: You can access a specific character in a string using indexing notation `[]`.
7. String methods: Python provides several built-in methods that can be used to manipulate strings, such as `upper()`, `lower()`, `replace()`, `strip()`, `split()`, `join()`, `startswith()`, `endswith()`, and many others.
8. Escape characters: Strings can contain escape characters such as `\n` (newline), `\t` (tab), and `\\` (backslash) that have special meanings.

Overall, strings are a very versatile and important data type in Python, and they are used extensively in most Python programs.