1.Can you explain the difference between a tuple and a list in Python?

A tuple is an immutable sequence of elements, while a list is a mutable sequence of elements. This means that once a tuple is created, its elements cannot be modified, added or removed, whereas elements in a list can be changed. Additionally, tuples are typically faster than lists in Python.

2. How do you handle missing or null values in a dataset when using Python?

There are several ways to handle missing or null values in a dataset when using Python, such as using the `fillna()` function in Pandas or replacing the missing or null values with a default value. One can also use `dropna()` method to drop the missing values.

3. Can you explain how Python's memory management works?

Python uses a private heap space to maintain all its objects and data structures. The memory manager in Python is responsible for allocating and deallocating memory for the heap. The allocation of

heap space for Python objects is done using the built-in `memory manager` and `garbage collector`.

4. How would you debug a Python script?

There are several ways to debug a Python script, including:

- Using the `print()` function to print variable values at different points in the script.
- Using a `debugger` like `pdb` or `ipdb`.
- Using an integrated development environment (IDE) that has a built-in debugger.

5. Can you explain how to use Python's built-in `map()` and `filter()` functions?

The `map()` function applies a given function to all elements of an input list and returns a new list containing the results. The `filter()` function is used to filter elements from a list based on a given condition, and returns a new list containing only the elements that satisfy the condition.

6. Can you explain the difference between a local and global variable in Python?

In Python, a local variable is a variable that is defined inside a function and can only be accessed within that function. A global variable is a variable that is defined outside of any function and can be accessed from anywhere in the script.

7. How would you go about profiling the performance of a Python script?

There are several ways to profile the performance of a Python script, such as using the `cProfile` module or a third-party library like `line_profiler` or `memory_profiler`. One can also use built-in libraries like `timeit` to measure the execution time of specific portions of the code.

8. Can you explain how to use Python's `try` and `except` statements for error handling?

The `try` and `except` statements in Python are used for error handling. The `try` block contains the code that might raise an exception, and the `except` block contains the code that will be executed if an exception is

raised. You can also use the `finally` block to execute code regardless of whether an exception was raised or not.

9. Can you explain how to use Python's `with` statement?

The `with` statement in Python is used to wrap the execution of a block of code with methods defined by a context manager. This is typically used to open and close resources, such as files or database connections, and ensure that the resources are properly closed, even if an exception is raised.

10. How do you implement object-oriented programming in Python?

In Python, object-oriented programming (OOP) is implemented using classes and objects.

A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).

To create a class in Python, the `class` keyword is used, followed by the name of the class. The class definition is then followed by a block of code, which is indented under the class definition. This block of code contains the member variables and methods of the class.

For example, consider the following Python class definition for a simple "Dog" class:

```
class Dog:

    def __init__(self, name, age):

        self.name = name

        self.age = age

    

    def bark(self):

        print("Woof woof!")
```

To create an object of the class, we can use the class name followed by parentheses, and assign the resulting object to a variable.

```
dog = Dog("Fido", 2)
```

Then we can access the object's attributes and methods using the dot notation.

```
print(dog.name) # "Fido"
```

```
print(dog.age) # 2
```

```
dog.bark() # Woof woof!
```

python also support inheritance, polymorphism, encapsulation and other OOP concepts, which allows you to model real-world objects and situations in your code.