1. Can you differentiate between a List and a Tuple?

Lists and tuples are Python data structures. The list is dynamic and whereas the tuple has static characteristics. They both have various advantages and use cases.

List

The list is the mutable data type, consumes more memory, and it is better for element insertion and deletion. Furthermore, it has several building functions, and the implication of iterations is slower compared to Tuple.

Example:

```
a_list = ["Data", "Camp", "Tutorial"]
```

2. What is __init__() in Python?

It is known as a constructor in OOP terminology. It is used to initiate a state when you create a new object. For example, you can assign values to object properties or run the operations that are necessary when the object is created.

The __init__() method is reserved for Python classes, and it is called automatically when you create a new object.

Example:

We have created a `book_shop` class and added the constructor and `book()` function. The constructor will store the book title name and the `book()` function will print the book name.

To test our code we have initialized the `b` object with “Sandman” and executed the `book()` function.

3. What is the difference between a mutable data type and an immutable data type?

The mutable Python data types can be modified, and they can change at runtime, for example, a List, Dictionary, and Set.

The immutable Python data types can not be changed or modified, and they remain unchanged during runtime, for example, a Numeric, String, and Tuple.

4. Explain List, Dictionary, and Tuple comprehension with an example.

List

List comprehension offers one-liner syntax to create a new list based on the values of the existing list. You can use a `for loop` to replicate the same thing, but it will require you to write multiple lines, and sometimes it can get complex.

List comprehension eases the creation of the list based on existing iterable.

5. What is monkey patching in Python?

Monkey patching in Python is a dynamic technique that can change the behavior of the code at run-time. In short, you can modify a class or module at run-time.

Example:

Let's learn monkey patching with an example.

1. We have created a class `monkey` with a `patch()` function. We have also created a `monk_p` function outside the class.
2. We will now replace the `patch` with the `monk_p` function by assigning `monkey.patch` to `monk_p`.
3. In the end, we will test the modification by creating the object using the `monkey` class and running the `patch()` function.