1. Differentiate between lists and tuples.

|  |  |  |
| --- | --- | --- |
| **Topic** | **List** | **Tuple** |
| **Mutability** | Mutable (You can add, remove, or modify items in a list.) | Immutable(You cannot add, remove, or modify items in a tuple.) |
| ****Syntax**** | **Lists** are defined using square brackets **[]** | **Tuples** are defined using parentheses **()** |
| ****Performance**** | * allows for operations like append and extend. | * **Tuples** are generally more memory-efficient and can be faster than lists due to their immutability and fixed size. |
| ****Use Cases**** | **Lists** are typically used when the collection of items needs to be modified (added, removed, or changed) throughout the program. | **Tuples** are used when the collection of items should remain constant and not change throughout the program, often for representing fixed collections of items or for use as keys in dictionaries. |
| ****Functions and Methods**** | **Lists** have a variety of methods for modifying their content(append, remove, sort) | **Tuples** have fewer methods since they cannot be modified(count, index) |
| ****Nested Structures**** | Both **lists** and **tuples** can contain other lists or tuples as their elements, allowing for the creation of complex data structures. | |
| ****Hashability**** | **Lists** are not hashable because they are mutable. This means lists cannot be used as keys in dictionaries.  # This will raise a TypeError  # my\_dict = {[1, 2, 3]: 'value'} | **Tuples** are hashable if all their elements are hashable. This allows tuples to be used as keys in dictionaries.  my\_dict = {(1, 2, 3): 'value'} |

2. What are negative indices?

Negative indices in Python are a way to access elements of a sequence (such as lists, tuples, and strings) from the end rather than the beginning. Negative indexing starts from **-1**, which corresponds to the last element of the sequence, **-2** for the second-to-last element, and so on.

my\_string = "Hello"

print(my\_string[-1]) # Output: 'o'

print(my\_string[-2]) # Output: 'l'

print(my\_string[-3]) # Output: 'l'

my\_tuple = (100, 200, 300, 400)

print(my\_tuple[-1]) # Output: 400

print(my\_tuple[-2]) # Output: 300

print(my\_tuple[-3]) # Output: 200

3. How long can an identifier be in Python?

Although Python does not impose a specific limit on the length of an identifier, it is best practice to use names that are long enough to be descriptive but short enough to be manageable and readable. Following Python's naming conventions and guidelines will help ensure that your code is both effective and maintainable.

4. How would you convert a string into lowercase?

you can convert a string to lowercase using the **lower()** method. This method returns a new string where all the uppercase characters in the original string are converted to lowercase.

# Original string

original\_string = "Hello, World!"

# Convert to lowercase

lowercase\_string = original\_string.lower()

print(lowercase\_string) # Output: "hello, world!"

5. What is the pass statement in Python?

The **pass** statement in Python is a null operation; it does nothing when executed. It is used as a placeholder in situations where syntactically, a statement is required, but you do not want any code to be executed.

def my\_function():

pass

# Call the function

my\_function()

The **pass** statement is a useful tool in Python for writing placeholders in your code where an action is syntactically required but not yet implemented. It allows you to write code that is syntactically correct and can be extended later with the actual implementation.

6. Explain help() and dir() functions in Python.

|  |  |
| --- | --- |
| Help() | Dir() |
| The **help()** function is used to display the documentation of modules, classes, functions, keywords, etc. It provides a convenient way to access Python's documentation directly from the interpreter. | The **dir()** function is used to list the attributes and methods of an object. It returns a list of valid attributes for that object. When called without arguments, it returns the list of names in the current local scope. |
| **Function**  help(len)  **Module**  import math  help(math) | Object  -my\_list = [1, 2, 3]  print(dir(my\_list))  module-  import math  print(dir(math)) |
| * **help():** Provides detailed documentation about Python objects, modules, functions, etc. It is useful for understanding what an object or module does and how to use it. | * **dir():** Lists the attributes and methods of an object. It is useful for exploring the capabilities and properties of an object. |

7.How do you get a list of all the keys in a dictionary?

To get a list of all the keys in a dictionary in Python, you can use the **keys()** method, which returns a view object containing the keys of the dictionary. You can then convert this view object to a list if needed.

my\_dict = {

'name': 'Alice',

'age': 25,

'city': 'New York'

}

# Getting the keys view

keys\_view = my\_dict.keys()

print(keys\_view) # Output: dict\_keys(['name', 'age', 'city'])

# Converting the keys view to a list

keys\_list = list(keys\_view)

print(keys\_list) # Output: ['name', 'age', 'city']

* Use the **keys()** method to get a view object of the dictionary keys.
* Convert the view object to a list using the **list()** function if you need the keys in a list format.