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.

8. What is slicing?

Slicing is a powerful technique to efficiently access subsets of data in sequence data structures

slicing is performed using the colon (**:**) operator within square brackets **[]**.

sequence[start:stop:step]- Syntax

9. How would you declare a comment in Python?

comments are declared using the hash symbol (**#**). Any text following the **#** on the same line is ignored by the Python interpreter. Comments are typically used to explain code, make notes, or temporarily disable code.

10. How will you check if all characters in a string are alphanumeric?

To check if all characters in a string are alphanumeric in Python, you can use the **isalnum()** method. This method returns **True** if all characters in the string are alphanumeric (consisting of letters and numbers) and there is at least one character, otherwise it returns **False**.

# Define a string

my\_string = "Hello123"

# Check if all characters are alphanumeric

if my\_string.isalnum():

print("All characters are alphanumeric.")

else:

print("Not all characters are alphanumeric.")

11. How will you capitalize the first letter of a string?

In Python, you can capitalize the first letter of a string using the **capitalize()** method. This method returns a new string with the first character converted to uppercase and the rest of the characters converted to lowercase.

# Define a string

my\_string = "hello world"

# Capitalize the first letter

capitalized\_string = my\_string.capitalize()

# Print the result

print(capitalized\_string) # Output: "Hello world"

12. With Python, how do you find out which directory you are currently in?

In Python, you can find out which directory you are currently in by using the **os** module, which provides a way to interact with the operating system. The specific function to get the current working directory is **os.getcwd()**.

import os

# Get the current working directory

current\_directory = os.getcwd()

# Print the current working directory

print("Current working directory:", current\_directory)

13. How do you insert an object at a given index in Python?

In Python, you can insert an object at a given index in a list using the **insert()** method. This method takes two arguments: the index at which you want to insert the object and the object itself.

list.insert(index, object)

* **index**: The position at which the object should be inserted. If the index is out of range, it will add the object at the beginning or the end of the list.
* **object**: The object to be inserted into the list.

To insert an object at a specific index in a list in Python, use the **insert()** method. This method is straightforward and allows for flexible manipulation of lists by specifying the exact position where the new element should be added.

14. How do you reverse a list?

### Using the reverse() Method - my\_list.reverse()

### Using Slicing-reversed\_list = my\_list[::-1]

### Using the reversed() Function-reversed\_list = list(reversed(my\_list))

### Using a Loop-reversed\_list = [my\_list[i] for i in range(len(my\_list)-1, -1, -1)]

* **reverse() method**: Modifies the original list in place.
* **Slicing ([::-1])**: Creates a new reversed list without modifying the original.
* **reversed() function**: Provides an iterator to reverse the list, which can be converted back to a list.
* **Loop**: Manually constructs a reversed list using a loop or list comprehension.

15. What is the Python interpreter prompt?

* The Python interpreter prompt is the interface you interact with when you are using the Python interactive shell (also known as the REPL, which stands for Read-Eval-Print Loop). The prompt typically appears as three greater-than signs (**>>>**) and indicates that the interpreter is ready to accept and execute Python code.
* When you enter a block of code that spans multiple lines, such as a function or a loop, the interpreter uses a secondary prompt to indicate that you are still within the same block. The secondary prompt is typically three dots (**...**):
* The Python interpreter prompt (**>>>**) is where you enter Python commands in an interactive session. The secondary prompt (**...**) is used when entering multi-line constructs. This interactive environment is useful for testing code snippets, learning Python, and debugging.

16. How does a function return values?

* The **return** statement is used to exit a function and send a value back to the caller.
* A function can return multiple values as a tuple.
* If a function does not explicitly return a value, it returns **None** by default.
* The **return** statement can be used within loops and conditionals to exit the function early.
* Functions can return any type of value, including lists, dictionaries, and objects.

17. How would you define a block in Python?

Blocks are defined by their indentation level, which is a key aspect of Python's syntax. Indentation is used to define the scope of loops, conditionals, functions, classes, and other constructs.

Python uses indentation to determine the grouping of statements.

* In Python, blocks of code are defined by their indentation level.
* Consistent indentation is crucial for defining the scope and grouping of statements.
* Blocks are used with various control structures like **if**, **for**, **while**, **def**, and **class**.
* Proper indentation makes the code more readable and maintains the logical structure.

18. Why do we need break and continue in Python?

1. **break:**
   * Terminating a loop early when a specific condition is met.
   * Searching for an item in a list and stopping the search once the item is found.
   * Breaking out of nested loops when a certain condition is satisfied.
   * Break is used to exit the loop immediately when a condition met
2. **continue:**
   * Skipping iterations of a loop based on specific criteria.
   * Filtering elements in a loop and processing only those that meet certain conditions.
   * Handling special cases or edge cases within a loop.
   * It’s used to skip the current iteration and proceed to the next iteration of the loop

19. In one line, show us how you’ll get the max alphabetical character from a string.

max\_char = max(my\_string, key=ord)

You can get the maximum alphabetical character from a string in Python using the **max()** function with the **key** parameter set to **ord**. Here's the one-liner:

Replace **my\_string** with your actual string. This one-liner will return the maximum alphabetical character from the string based on their Unicode code points. If the string contains only non-alphabetical characters, it will return the highest ASCII character.

20. Can you name ten built-in functions in Python and explain each in brief?

1. **print():**
   * Usage: **print(value1, value2, ...)**
   * Explanation: Used to print the specified values to the standard output (usually the console). It can take multiple arguments separated by commas and converts each value to a string before displaying it.
2. **len():**
   * Usage: **len(sequence)**
   * Explanation: Returns the length (number of items) of the specified sequence (such as a string, list, tuple, or dictionary).
3. **input():**
   * Usage: **input(prompt)**
   * Explanation: Reads input from the user via the keyboard and returns it as a string. Optionally, you can provide a prompt message to display to the user.
4. **range():**
   * Usage: **range(start, stop, step)**
   * Explanation: Generates a sequence of numbers from **start** (inclusive) to **stop** (exclusive), with an optional **step** size. Useful for creating lists, iterating over a sequence of numbers, and defining loop ranges.
5. **type():**
   * Usage: **type(object)**
   * Explanation: Returns the type of the specified object. It can be used to check the type of variables, functions, and other objects.
6. **sorted():**
   * Usage: **sorted(iterable, key=None, reverse=False)**
   * Explanation: Returns a new sorted list from the elements of the specified iterable. Optionally, you can specify a **key** function to customize the sorting, and set **reverse=True** to sort in descending order.
7. **sum():**
   * Usage: **sum(iterable, start=0)**
   * Explanation: Returns the sum of all elements in the specified iterable (such as a list or tuple). Optionally, you can specify a **start** value to begin the summing.
8. **max():**
   * Usage: **max(iterable, \*[, key, default])** or **max(arg1, arg2, \*args[, key])**
   * Explanation: Returns the maximum value from the specified iterable or sequence of arguments. Optionally, you can specify a **key** function to customize the comparison, and **default** to specify a default value if the iterable is empty.
9. **min():**
   * Usage: **min(iterable, \*[, key, default])** or **min(arg1, arg2, \*args[, key])**
   * Explanation: Returns the minimum value from the specified iterable or sequence of arguments. Optionally, you can specify a **key** function to customize the comparison, and **default** to specify a default value if the iterable is empty.
10. **abs():**
    * Usage: **abs(x)**
    * Explanation: Returns the absolute value of the specified number **x**. For integers and floats, it returns the positive distance from zero.

21.How will you convert a list into a string?

To convert a list into a string in Python, you can use the **join()** method of strings. This method concatenates each element of the list into a single string with a specified separator between each element.

my\_list = ['apple', 'banana', 'cherry']

# Convert the list to a string with comma-separated elements

result = ', '.join(my\_list)

print(result) # Output: "apple, banana, cherry"

22. How will you remove a duplicate element from a list?

To remove duplicate elements from a list in Python,

you can convert the list to a set, which automatically removes duplicates due to its property of storing only unique elements, and then convert it back to a list if needed. Here's how you can do it

my\_list = [1, 2, 2, 3, 4, 4, 5]

unique\_list = list(set(my\_list))

print(unique\_list) # Output: [1, 2, 3, 4, 5]

my\_list = [1, 2, 2, 3, 4, 4, 5]

unique\_list = []

for item in my\_list:

if item not in unique\_list:

unique\_list.append(item)

print(unique\_list) # Output: [1, 2, 3, 4, 5]

23. What is a dictionary in Python?

A dictionary in Python is an unordered collection of key-value pairs. Each key-value pair maps the key to its corresponding value. Dictionaries are mutable and can contain heterogeneous data types.

1. **Key-Value Pairs:** Each element in a dictionary consists of a key and its corresponding value. Keys are used to retrieve the associated values quickly.
2. **Unordered:** Unlike sequences like lists or tuples, dictionaries do not maintain any specific order for their elements. Therefore, the order in which elements are inserted into a dictionary is not preserved.
3. **Mutable:** Dictionaries can be modified after creation. You can add, update, or remove key-value pairs from a dictionary.
4. **Indexed Access:** Elements in a dictionary are accessed using their keys rather than numerical indices. This allows for efficient lookup based on unique identifiers.
5. **Dynamic:** Dictionaries can grow or shrink dynamically as key-value pairs are added or removed

my\_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

In this dictionary:

* **'name'**, **'age'**, and **'city'** are keys.
* **'John'**, **30**, and **'New York'** are their corresponding values.

You can access values in a dictionary using square brackets **[]** with the key:

print(my\_dict['name']) # Output: John

24. Explain the //, %, and \*\* operators in Python.

Division

result = 10 // 3 # Output: 3

Modules

remainder = 10 % 3 # Output: 1

Exponentiation

result = 2 \*\* 3 # Output: 8

25. What do you know about relational operators in Python.

26. What are assignment operators in Python?

27. Explain logical operators in Python.

28. Finally, tell us about bitwise operators in Python.

29. What data types does Python support?

30. How would you convert a string into an int in Python?

31. How do you take input in Python?

32. What is recursion?

33. What does the function zip() do?

34. How do you calculate the length of a string?

35. Explain Python List Comprehension.

36. How do you get all values from a Python dictionary?

37. What if you want to toggle case for a Python string?

38. Write code to print everything in the string except the spaces.

39. Now, print this string five times in a row.

40. What is the purpose of bytes() in Python?

41. What is a control flow statement?

42. Create a new list to convert the following list of number strings to a list of numbers.

43. Given the first and last names of all employees in your firm, what data type will you use to store it?

44. How would you work with numbers other than those in the decimal number system?

45. How many arguments can the range() function take?

46. What is PEP 8?

47. What is the best code you can write to swap two numbers?

48. How can you declare multiple assignments in one statement?

49. If you are ever stuck in an infinite loop, how will you break out of it?

50. What are the benefits of using Python?

51. What are python modules? Name some commonly used built-in modules in Python?

52. What are local variables and global variables in Python?

53. What is a lambda function?

54. What is self in Python?

55. How does break, continue and pass work?

56. What does [::-1} do?

57. How can you generate random numbers in Python?

58. What is the difference between range & xrange?

59. How do you write comments in python?

60. How to add values to a python array?

61. What are Python libraries? Name a few of them.

62. How are classes created in Python?

63. What does an object() do?

64. Write a program to produce Fibonacci series in Python.

65. Write a program in Python to check if a number is prime.

66. Write a program in Python to check if a sequence is a Palindrome.

67. Write a one-liner that will count the number of capital letters in a file. Your code should work even if the file is too big to fit in memory.

68. Write a sorting algorithm for a numerical dataset in Python.

69. What is PYTHON PATH?

70. What type of language is python?