Python Assignment - 4

**1. What exactly is []?**

=> The symbol "[]" typically represents an empty list or array in Python. In programming, a list or an array is a data structure that allows you to store multiple values in a single variable. The square brackets, [], are used to define an empty list or to access elements within a list.

**2. In a list of values stored in a variable called spam, how would you assign the value 'hello' as the third value? (Assume [2, 4, 6, 8, 10] are in spam.)**

=> spam = [2, 4, 6, 8, 10]

Spam[2] = ‘hello’

print(spam)

output: [2, 4, ‘hello’, 8, 10]

**Let's pretend the spam includes the list ['a', 'b', 'c', 'd'] for the next three queries.**

**3. What is the value of spam[int(int('3' \* 2) / 11)]?**

=> The value of above expression is ‘d’

**4. What is the value of spam[-1]?**

=> spam[-1] = ‘d’

**5. What is the value of spam[:2]?**

=> spam[:2] = [‘a’, ‘b’]

**Let's pretend bacon has the list [3.14, 'cat,' 11, 'cat,' True] for the next three questions.**

**6. What is the value of bacon.index('cat')?**

=> The index of cat is ‘1’.

**7. How does bacon.append(99) change the look of the list value in bacon?**

=> bacon = [3.14, ‘cat’, 11, ‘cat’, True, 99]

**8. How does bacon.remove('cat') change the look of the list in bacon?**

=> bacon = [3.14, 11, ‘cat’, True, 99]

**9. What are the list concatenation and list replication operators?**

=>

1. List Concatenation Operator (+): The ‘+’ operator is used to concatenate or join two lists together. When applied to two lists, it creates a new list containing all the elements from both lists in the order they appear.
2. List Replication Operator (\*): The ‘\*’ operator is used to replicate or repeat a list a certain number of times. It creates a new list by repeating the elements of the original list.

**10. What is difference between the list methods append() and insert()?**

=>

1. append():

The ‘append()’ method is used to add an element to the end of a list. It takes a single argument, which is the element to be added. The ‘append()’ method modifies the original list by adding the element at the end.

1. insert():

The ‘insert()’ method is used to insert an element at a specific position in a list. It takes two arguments: the index at which the element is to be inserted, and the element itself.

**11. What are the two methods for removing items from a list?**

=> The two methods for removing items from a list are:

1. remove():

The remove() method is used to remove the first occurrence of a specified value from a list. It takes a single argument, which is the value to be removed. If the value is found in the list, it is removed, modifying the original list. If the value is not present, it raises a ValueError.

2. pop():

The pop() method is used to remove an element from a list at a specific index. It takes an optional index argument. If no index is provided, it removes and returns the last element from the list. If an index is specified, it removes and returns the element at that index, modifying the original list.

**12. Describe how list values and string values are identical.**

=> List values and string values share some similarities, but they are fundamentally different data types in Python. Here's a comparison between list values and string values:

1.Sequence Type:

Both lists and strings are sequence types in Python. They maintain an ordered collection of elements. In lists, the elements can be of different data types, while strings are sequences of characters.

2.Indexing:

Both lists and strings allow accessing individual elements by their index. Indexing starts from 0, where the first element has an index of 0, the second has an index of 1, and so on.

3.Slicing:

Both lists and strings support slicing operations to extract sub-sequences. Slicing allows you to create a new sequence by specifying a range of indices.

4.Iteration:

Both lists and strings can be iterated over using loops, such as the for loop. This allows you to access each element of the sequence one by one.

5.Immutability vs. Mutability:

Strings are immutable, which means their elements cannot be modified once the string is created. In contrast, lists are mutable, and you can modify their elements, add or remove elements, and change their length.

6.Type of Elements:

Lists can store elements of different data types, such as numbers, strings, or even other lists. On the other hand, strings store characters exclusively.

7.Methods and Operations:

Lists and strings have different methods and operations specific to their data types. For example, strings have methods like split(), join(), and string-specific operations like concatenation (+) and repetition (\*). Lists have methods like append(), remove(), and list-specific operations like concatenation (+) and repetition (\*).

**13. What's the difference between tuples and lists?**

=> Tuples and lists are both data structures in Python, but they have some key differences:

**Mutability**: Lists are mutable, which means you can modify, add, or remove elements after creation. Tuples, on the other hand, are immutable, meaning their elements cannot be modified once the tuple is created. You cannot add or remove elements from a tuple.

**Syntax**: Lists are defined using square brackets [ ], with elements separated by commas.

For example: my\_list = [1, 2, 3]. Tuples are defined using parentheses ( ), with elements separated by commas. For example: my\_tuple = (1, 2, 3).

**Usage**: Lists are commonly used when you have a collection of items that may change over time. You can append, insert, or delete elements from a list as needed. Tuples, on the other hand, are typically used when you have a collection of items that should not be changed, such as a set of coordinates or a record of data.

**Performance**: Tuples are generally more memory-efficient and faster to access than lists. Since tuples are immutable, Python can optimize their storage and retrieval. Lists, being mutable, require more memory allocation and incur a small performance overhead for operations like appending or extending.

**14. How do you type a tuple value that only contains the integer 42?**

=> my\_tuple = (42,)

**15. How do you get a list value's tuple form? How do you get a tuple value's list form?**

=> To convert list to tuple.

My\_list = [1,2,3]

My\_tuple = tuple(My\_list)

print(my\_tuple)

To convert tuple to list.

My\_tuple = (1,2,3)

My\_list = list(My\_tuple)

print(My\_list)

**16. Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?**

=> Variables that "contain" list values in Python actually contain references or pointers to the list objects in memory. In Python, objects such as lists are stored separately in memory, and variables hold references to those objects rather than the objects themselves.

When you assign a list to a variable, the variable is assigned a reference to the list object's memory location. This means that multiple variables can refer to the same list object. Modifying the list through one variable will be reflected in all variables that reference the same list object.

**17. How do you distinguish between copy.copy() and copy.deepcopy()?**

=> The **copy.copy()** and **copy.deepcopy()** functions in Python's copy module are used to create copies of objects, but they differ in how they handle nested objects or references within the original object:

**copy.copy()**: This function performs a shallow copy of an object. It creates a new object and populates it with references to the nested objects found in the original. In other words, it creates a new object but does not create copies of the nested objects themselves. Changes made to the nested objects in the copy will also be reflected in the original object and vice versa.

**copy.deepcopy()**: This function performs a deep copy of an object. It creates a completely independent copy of the original object and all its nested objects. It recursively goes through the original object and creates new objects for all the nested objects, effectively duplicating the entire structure. Changes made to the nested objects in the copy will not affect the original object, and vice versa.