Assignment 4

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

Ans. In Python, square brackets [] have two main purposes:  
**a) List creation:** They are used to define lists, which are ordered collections of items. We place the items we want in the list separated by commas within the brackets.  
For example:

fruits = ["apple", "banana", "cherry"]

numbers = [1, 2, 3, 4]

mixed\_list = [True, 3.14, "hello"]

**b) Accessing elements:** Square brackets are also used to access elements within a list or string (which is a sequence of characters). Each element in a list has a specific index, starting from 0. We can use the index within square brackets to retrieve or modify an element.  
For example:  
  
fruits = ["apple", "banana", "cherry"]

first\_fruit = fruits[0] # first\_fruit will be "apple"

last\_fruit = fruits[2] # last\_fruit will be "cherry"

fruits[1] = "mango" # This changes the second element to "mango"

**Q.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)**

Ans. In Python, lists are mutable, meaning we can modify their elements after creation. Using square brackets [] and the index of the element (which starts from 0), we can assign a new value to the desired position in the list.  
  
spam = [2, 4, 6, 8, 10]

spam[2] = "hello" # Assigning "hello" at index 2

print(spam) # Output: [2, 4, "hello", 8, 10]

In this example, spam[2] refers to the third element (because indexing starts from 0) in the spam list, which originally has the value 6. Assigning "hello" to spam[2] changes the value at that index to "hello".

Note : Let's Pretend the spam includes the list ['a','b','c','d'] for the next three queries  
  
**Q.3 What is the value of spam[int(int('3'\*2)/11)]?**Ans. let's analyze the expression spam[int(int('3'\*2)/11)] assuming spam is the list ['a', 'b', 'c', 'd']:  
Steps:   
a) '3'\*2: This part multiplies the string '3' by 2. In Python, string multiplication concatenates the string with itself the specified number of times. So, '3'\*2 becomes '33'.

b) int('33'): Here, we convert the string '33' obtained in the previous step into an integer. '33' can be easily converted to the integer 33.

c) int(33 / 11): We are dividing 33 by 11. Since we're using the int function, this performs integer division, which discards the remainder and gives us 3. Note that in Python 2, this might result in a float (3.0) due to default division behavior. However, most modern Python uses integer division by default.

d) spam[3]: Finally, we use the index 3 (obtained in the previous step) to access an element within the list spam. we know indexing starts from 0 in Python.  
  
Result : Following the steps, spam[int(int('3'\*2)/11)] translates to spam[3]. Since spam is ['a', 'b', 'c', 'd'], spam[3] will access the element at index 3, which is 'd'.

Therefore, the value of the expression spam[int(int('3'\*2)/11)] is 'd'.

**Q.4 What is the value of spam[-1] ?  
 Ans.**  Assuming spam is the list ['a', 'b', 'c', 'd'], then spam[-1] will evaluate to 'd'.  
 In Python, negative indexing allows you to access elements from the end of the list. -1 refers to the last element, -2 refers to the second-to-last element, and so on.

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

**Ans.** Here, spam[:2] will return a sub-list containing the elements from the beginning of the list spam up to, but not including, index.  
So, spam[:2] will evaluate to ['a', 'b'].  
The colon : indicates slicing, and without a starting index before it, it defaults to the beginning (index 0). Specifying 2 after the colon tells Python to include elements up to, but not including, I index 2.

**Q.6 Let’s Pretend bacon has the list [3.14, ‘cat’, 11, ‘cat’, True] for the next three questions.  
What is the value of bacon.index(‘cat’)?**Ans. The value of bacon.index('cat') is 1.   
Explanation : The index() method is used with lists in Python to find the index of the first occurrence of a specific value within the list.

In this case, we're calling bacon.index('cat'). This searches for the first element in the bacon list that has the value 'cat'.

Since 'cat' is the second element (at index 1) in the bacon list, bacon.index('cat') returns 1.  
  
(The index() method only returns the index of the first occurrence of the value.)  
  
**Q.7 How does bacon.append(99) change the look of the list value in bacon?**Ans. The bacon list will become [3.14, 'cat', 11, 'cat', True, 99] Calling bacon.append(99) modifies the original list stored in bacon by adding the value 99 to the end of the list.

Explanation :  
bacon.append(99): This expression calls the append() method on the list bacon. The append() method is used to add an element to the end of a list. In this case, we're adding the value 99.

Modification: Since lists in Python are mutable (changeable), append() directly modifies the existing list bacon. It doesn't create a new copy.  
  
**Q.8 How does bacon.remove(‘cat’) change the look of the list in bacon?**|  
Ans. The bacon list will become [3.14, 11, 'cat', True].  
Explanation :   
bacon.remove('cat'): This expression calls the remove() method on the list bacon. The remove() method searches for the specified value (in this case, 'cat') and removes the first occurrence of that value from the list.

Removing element: If the value 'cat' is found in the list, it's removed. However, it's important to note that remove() only removes the first occurrence. If there are multiple 'cat' elements in the list, only the first one will be removed.  
If 'cat' is not found in bacon then no change occurs to the list.  
  
**Q.9 What are the list concatenation & list replication operators** **?**

Ans. These are the two main operators working with lists:  
**Concatenation (+)**

Used to join multiple lists into a single new list.

The order of the lists matters. The elements from the left operand are placed before the elements from the right operand in the resulting list.  
For Example:  
list1 = [1, 2, 3]  
list2 = [4, 5, 6]   
combined\_list = list1 + list2 # combined\_list will be [1, 2, 3, 4, 5, 6]  
  
  
**Replication (\*)**

Used to create a new list by repeating an existing list a specified number of times.

The left operand is the list to be repeated, and the right operand specifies the number of repetitions.  
For Example:  
numbers = [10, 20]

repeated\_list = numbers \* 3 # repeated\_list will be [10, 20, 10, 20, 10, 20]  
  
some key points :  
Both operators create new lists. They don't modify the original lists.

Concatenation works with lists of any data types (strings, integers, etc.), as long as they are sequences.

Replication only works with sequences, not with other data types like integers or booleans.  
  
**Q.10 What is difference between the list methods append() and insert()?**Ans**.** The key difference between the append() and insert() methods in Python lists lies in where they add elements:

**append(element)**

Adds a new element to the end of the list.

Takes only one argument, which is the element to be added.  
For example:   
my\_list = [1, 2, 3]

my\_list.append(4) # Add 4 to the end of the list

print(my\_list) # Output: [1, 2, 3, 4]  
  
**insert(index, element)**  
Inserts a new element at a specific index within the list.

Takes two arguments:

index: The index where you want to insert the element. (Remember indexing starts from 0).

element: The element to be inserted.

For Example:  
my\_list = [1, 2, 3]

my\_list.insert(1, 5) # Insert 5 at index 1 (before the element at index 1)

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

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

**Ans**. There are several methods for removing items from lists in Python, but here are two commonly used approaches:

**Using the remove() method:**

The remove() method removes the first occurrence of a specified value from the list. It's efficient for removing elements by their value.

For example :

fruits = ["apple", "banana", "cherry", "banana"]

fruits.remove("banana") # Removes the first "banana"

print(fruits) # Output: ["apple", "cherry", "banana"]

**Using slicing with deletion:**

Slicing allows you to extract a portion of the list. You can combine slicing with the del keyword to remove elements from specific positions or based on conditions.

For Example :

numbers = [1, 2, 3, 4, 5]

del numbers[1:3] # Removes elements from index 1 (inclusive) to 3 (exclusive)

print(numbers) # Output: [1, 4, 5]

Use remove() when you want to remove an element by its value (especially if there might be duplicates).

Use slicing with del for targeted removal based on position or conditions.

**Q.12 describe how list values & string values are identical?**

**Ans**. While lists and strings can hold sequences of elements in Python, they have some key differences in how they store and manipulate data:

**Similarities:**

Ordered sequences: Both lists and strings are ordered collections of items. Elements are accessed by their position (index), starting from 0.

Indexing and slicing: You can use square brackets [] to access elements by index and slicing to extract sub-sequences.

**Differences:**

Mutability:  
Lists are mutable, meaning you can modify their elements after creation. You can add, remove, or change elements within a list.

Strings are immutable. Once a string is created, its content cannot be directly changed.

Data types of elements:  
Lists can hold elements of various data types (integers, strings, booleans, even other lists) within the same list.

Strings can only contain characters (including letters, numbers, symbols, and whitespace).

In essence, lists are versatile for storing and managing collections of different data types, while strings are specifically designed for handling sequences of characters.

**Q. 13. What's the difference between tuples & list?  
 Ans.**

|  |  |  |
| --- | --- | --- |
| Feature | List | Tuple |
| Mutability | Mutable (elements can be changed) | Immutable (elements cannot be changed) |
| Creation | Created with square brackets [] | Created with parentheses () |
| Use cases | General purpose collections | Data integrity, function arguments, etc. |

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

**Ans.** There are two ways to create a tuple containing a single integer 42 in Python:  
 a) Tuple with comma:  
 my\_tuple = (42,) # The comma after 42 is essential to define it as a tuple  
 In Python, a single value within parentheses might be interpreted as a parenthesized expression, not a tuple. Adding a comma after the value explicitly creates a tuple with one element.

b) Using the tuple() constructor:

my\_tuple = tuple([42]) # Create a list with 42 and convert it to a tuple

Here, we create a temporary list containing 42 using square brackets []. Then, we use the tuple() constructor to convert the single-element list into a tuple.

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

Ans.

In Python, you can convert between lists and tuples using built-in functions:

**Converting a list to a tuple:**

Using the tuple() function: This is the most straightforward way. The tuple() function can take any iterable (including lists) as input and create a tuple containing the elements.

For Example :  
my\_list = [1, 2, 3, "apple"]

my\_tuple = tuple(my\_list)

print(my\_tuple) # Output: (1, 2, 3, 'apple')

**Converting a tuple to a list:**

Using the list() function: Similar to tuple(), the list() function can convert any iterable, including tuples, into a list.

For Example :

my\_tuple = (4, 5, 6)

my\_list = list(my\_tuple)

print(my\_list) # Output: [4, 5, 6]

**Q.16 Variables that “contain” list values are not necessarily lists themselves. Instead what do they contain?**Ans.   
Variables that contain list values contain references to the lists, not the lists themselves.  
Because

* Lists can store multiple values in an ordered collection.
* Variables act as named containers that hold the value of something else.
* When you assign a list to a variable, the variable doesn't copy the entire list. Instead, it gets a reference (like an address) pointing to the actual list in memory.

This means any changes made to the list through the variable will affect the original list, and vice versa.

**Q.17 How do you distinguish between copy.copy() and copy.deepcopy()?**  
Ans. Both copy.copy() and copy.deepcopy() are functions in Python's copy module used for creating copies of objects. However, they differ in how deeply they copy the object's contents:

**Shallow Copy (copy.copy())**

1. Creates a new object of the same type but only copies the top-level elements.
2. If the original object contains references to other objects (like lists or nested objects within a list), those references are copied directly into the new object.
3. Any changes made to the nested objects in the original or the copy will be reflected in both since they share the same reference.

**Deep Copy (copy.deepcopy())**

1. Creates a completely independent copy of the original object.
2. Recursively copies all nested objects within the original object, creating new copies for each level.
3. Changes made to the nested objects in the original or the copy won't affect each other because they are separate objects.