1. What exactly is []?

Ans:- [] is used to represent an empty list, It has no elements .

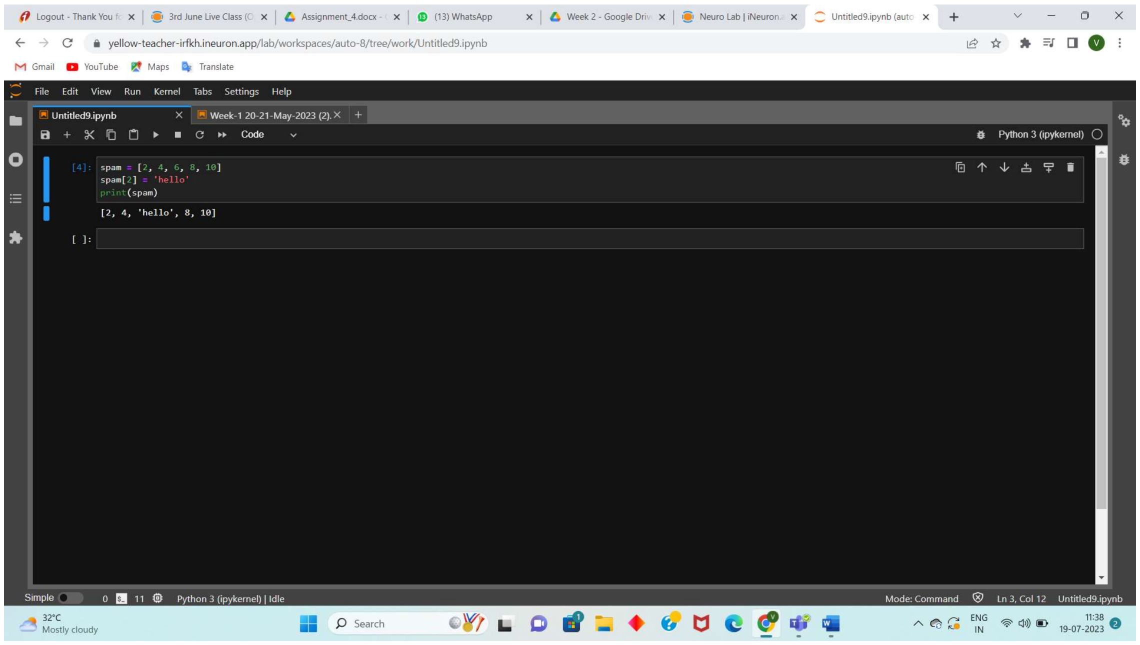
2. In a list of values stored in a variable called spam, how would you assign the value 'hello' as the third value? (Assume [2, 4, 6, 8, 10] are in spam.)

Ans:-

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

spam[2] = 'hello'

print(spam)



```
[4]: spam = [2, 4, 6, 8, 10]
     spam[2] = 'hello'
     print(spam)

     [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)]?

Ans:-

('3' * 2) results in string '33'

Int('33') convert the string '33' in integer which is 33

Int (33/11) resulting 3

Finally we have spam[3]. So in spam=['a', 'b', 'c', 'd'] the value at index 3 is ' d'

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

Ans:- In spam=['a', 'b', 'c', 'd']

Spam[-1] is ‘d’

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

Ans:- In spam=['a', 'b', 'c', 'd']

The value of spam[:2] is [‘a’,’b’]

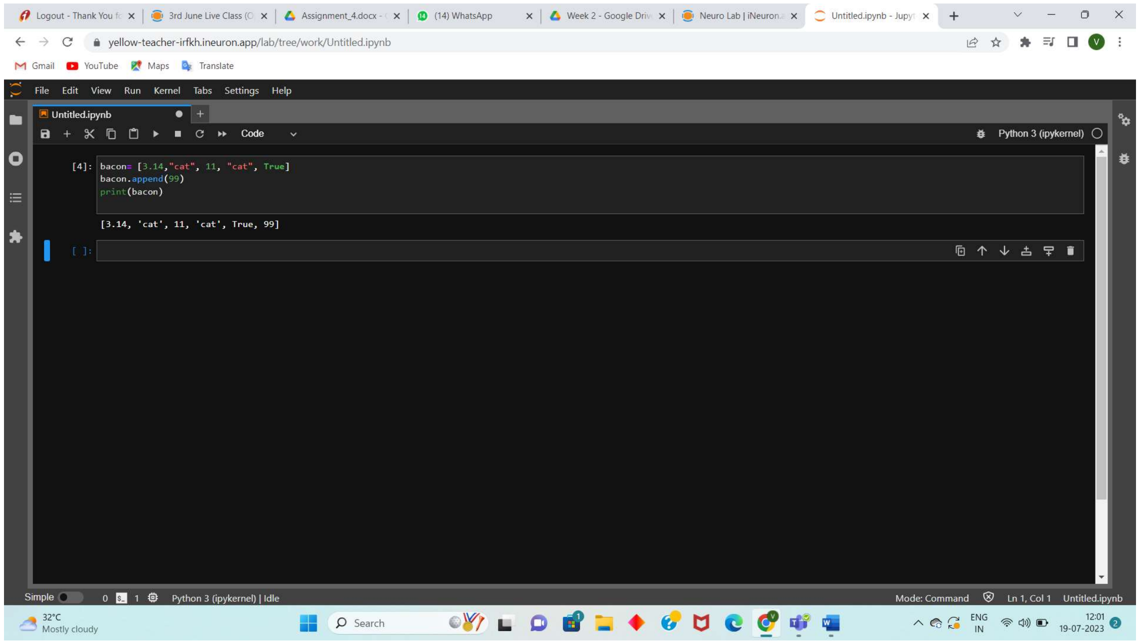
Let's pretend bacon has the list [3.14, 'cat,' 11, 'cat,' True] for the next three questions.

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

Ans:- The value of bacon.index('cat') is 1

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

Ans:- Updated list will be bacon= [3.14, 'cat,' 11, 'cat,' True,99]

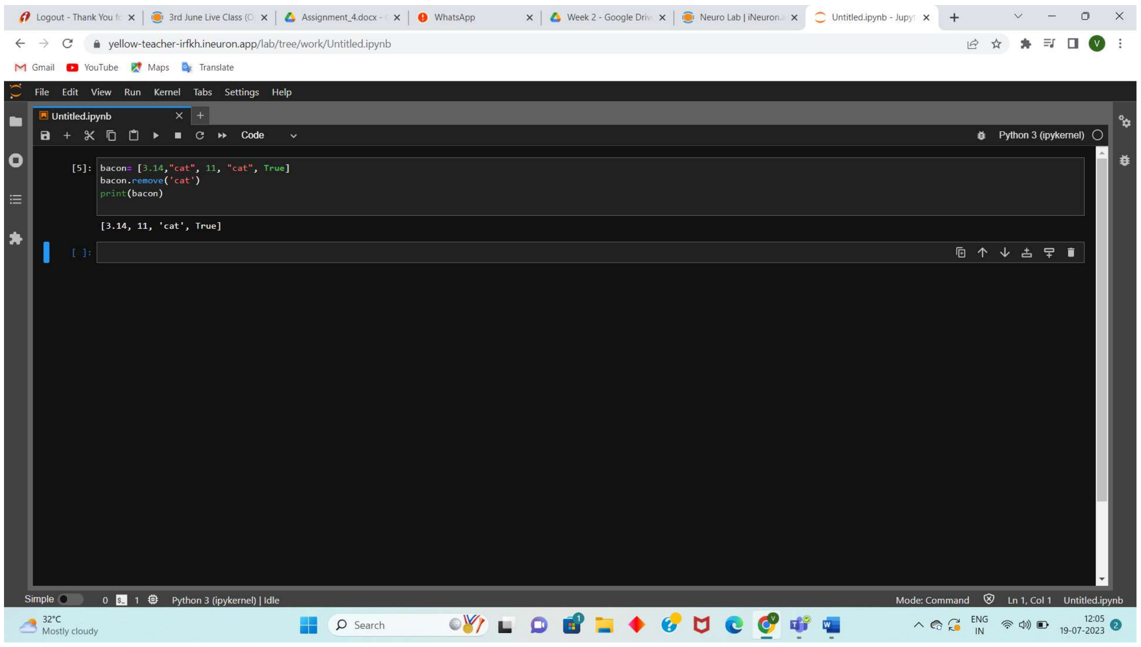


```
[4]: bacon= [3.14,"cat", 11, "cat", True]
     bacon.append(99)
     print(bacon)

     [3.14, 'cat', 11, 'cat', True, 99]
```

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

Ans- If bacon has the list [3.14, 'cat,' 11, 'cat,' True]

After putting bacon.remove('cat'), Updated value of bacon=[3.14, 'cat,' 11, 'cat,' True,99]

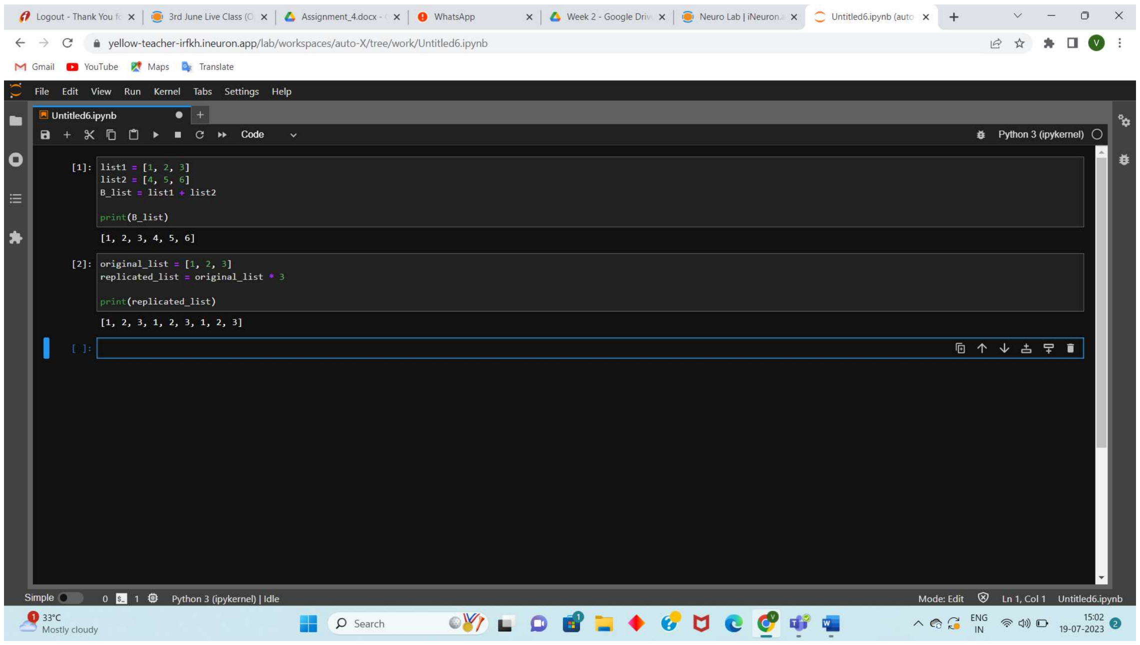


```
[5]: bacon= [3.14,"cat", 11, "cat", True]
     bacon.remove('cat')
     print(bacon)

     [3.14, 11, 'cat', True]
```

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

Ans:- **list concatenation(+):-** The list concatenation+ is used to combine two or more list in to a single list. It create a new list that contains all the elements from the list followed by all the elements from the second list & so on.

list replication operators(*):- list replication operators * is used to create new list by repeating the element of existing list a specifies number of time. It creates a new list where elements from the original list are repeated the given number of times.



```
[1]: list1 = [1, 2, 3]
     list2 = [4, 5, 6]
     B_list = list1 + list2

     print(B_list)

     [1, 2, 3, 4, 5, 6]

[2]: original_list = [1, 2, 3]
     replicated_list = original_list * 3

     print(replicated_list)

     [1, 2, 3, 1, 2, 3, 1, 2, 3]

[ ]:
```

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

Ans:- There are following difference between the list methods append() and insert()?

append():- This method is used to add an element to the end of a list. It takes a single argument, value, which is the element you want to add to the list

insert():-This method is used to add an element at a specific index in the list. It takes two arguments: index , which represents the position where the element should be inserted , & value , which is the element to be added.

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

Ans:- Following are the two methods for removing items from a list:-

1. list.remove(value)
2. list.pop(index)

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

Ans:- They share some similarity:-

1. Sequence Type:- Both list & string are sequence type, which means they represent ordered collection of elements.

2. Indexing & Slicing:- Both supports indexing & slicing operation. Indexing allow accessing individual element by their position, while slicing allow extracting a portion of the sequence based on specified start & end position.
3. Length :- Both have a length, which can be obtained using the len().

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

Ans:- There are some Key difference:-

1. **Mutability:-**
   a) list are mutable, which means their element can be changed after they are created.
   b) tuples are immutable ,which means their element can not be changed after they created
2. **Syntax:-**
   a) Lists are defined using square bracket[]
   b) Tuples are defined using parentheses()

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

Ans:- my_tuple=(42,)

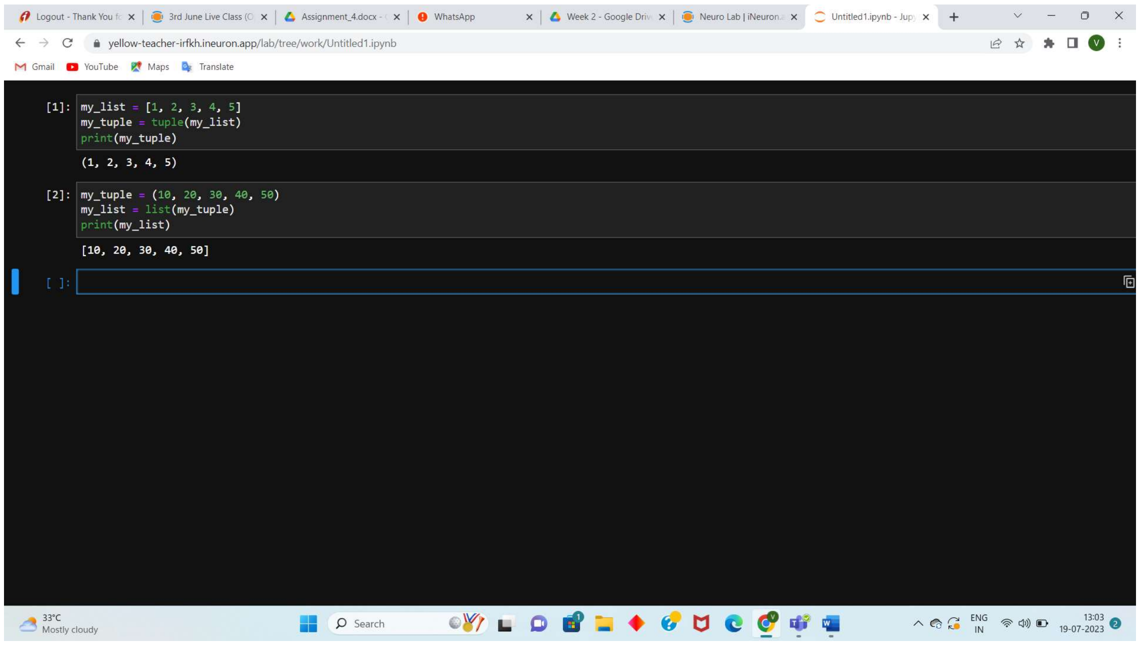
15. How do you get a list value's tuple form? How do you get a tuple value's list form?

Ans:- **List value's tuple form**

```
my_list = [1, 2, 3, 4, 5]

my_tuple = tuple(my_list)

print(my_tuple)
```

Tuple value's list form

```
my_tuple = (10, 20, 30, 40, 50)

my_list = list(my_tuple)

print(my_list)
```



```
[1]: my_list = [1, 2, 3, 4, 5]
     my_tuple = tuple(my_list)
     print(my_tuple)

     (1, 2, 3, 4, 5)

[2]: my_tuple = (10, 20, 30, 40, 50)
     my_list = list(my_tuple)
     print(my_list)

     [10, 20, 30, 40, 50]

[ ]:
```

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

Ans:- Variables that "contain" list values are not necessarily lists themselves. Instead, they contain references to the memory location where the list is stored. List are mutable object & when we assign a list to a variable , the variable does not store the actual list directly, it stores a reference to the memory location where the list data is stored.

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

Ans:- **copy.copy():-** This function creates a shallow copy of an object. A shallow copy means that it duplicates the top -level object & creates new references to the nested abject within it. However , it does not create new copies of the nested object . As a result, change made to the nested object in the copy will be reflected in the original object & vice versa.

copy.deepcopy():-This function creates a deep copy of an object. A deep copy means that it duplicates the top -level object & recursively creates new copies of all nested abject within it. As a result, change made to the nested object in the copy will not affect the original object & vice versa.