**FSDS MAY BATCH 2022(Python Basics 4)**

**Submitted by: Shubham Tiwari**

Q1. What exactly is []?

Ans: [] is an empty list in Python, which means it contains no items or elements. It is used to store a collection of items or elements of any data type (e.g. integers, strings, floats, etc.). Lists are declared by enclosing elements separated by commas within square brackets.

Q2. 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.)

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

Ans: spam[2] = "hello"

Q3. What is the value of spam[int(int(‘3’ \* 2) / 11)]?

Ans: The value of spam[int(int('3' \* 2) / 11)] is "d".

Explanation:

* '3' \* 2 = '33'
* int('33') = 33
* int(33 / 11) = 3
* spam[3] = "d".

Q4. What is the value of spam[-1]?

Ans: The value of spam[-1] is "d"

Q5. What is the value of spam[:2]?

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

Ans: The value of spam[:2] is ["a", "b"].

Q6. What is the value of bacon.index(‘cat’)?

Ans: The value of bacon.index('cat') is 2.

Q7. How does bacon.append(99) change the look of the list value in bacon?

Ans: After executing "bacon.append(99)", the list value in bacon becomes [3.14, 'cat', '11', 'cat', 'True', 99].

Q8. How does bacon.remove(‘cat’) change the look of the list in bacon?

Ans: After executing "bacon.remove('cat')", the list value in bacon becomes [3.14, '11', 'cat', 'True']. It only removes the first occurrence of the value 'cat' in the list.

Q9. What are the list concatenation and list replication operators?

Ans: The list concatenation operator is **+** and the list replication operator is **\*** in Python.

The **+** operator is used to concatenate two or more lists, creating a new list that contains all elements from the original lists.

The **\*** operator is used to replicate a list multiple times, creating a new list that contains the repeated elements.

For example:

**spam = [1, 2, 3]**

**eggs = [4, 5, 6]**

**ham = spam + eggs**

**print(ham)**

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

**cheese = [7, 8, 9]**

**toast = cheese \* 3**

**print(toast)**

**[7, 8, 9, 7, 8, 9, 7, 8, 9]**

In summary, the **+** operator concatenates two or more lists to create a new list, while the **\*** operator replicates a list multiple times to create a new list.

Q10. What is difference between the list methods append() and insert()?

Ans: The **append()** method is used to add an element to the end of the list, whereas the **insert()** method is used to add an element at a specific index in the list. For example:

**spam = [1, 2, 3, 4]**

**spam.append(5)**

**print(spam)**

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

**spam.insert(2, 6)**

**print(spam)**

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

So, in short, **append()** is used to add elements at the end of the list, while **insert()** is used to add elements at a specific position in the list.

Q11. What are the two methods for removing items from a list?

Ans: The two methods can be:

1. **remove** method
2. **del** statement

Q12. Describe how list values and string values are identical.

Ans: Both lists and strings are sequences in Python, meaning they are ordered collections of values, and each value can be accessed by an index.

Q13. What’s the difference between tuples and lists?

Ans: Tuples and lists are both data structures in Python used to store collections of items, but there are a few key differences between them:

1. **Mutability:** Lists are mutable, meaning you can add, remove, or modify elements after the list is created. Tuples are immutable, meaning their elements cannot be changed once the tuple is created.
2. **Syntax:** Lists are created with square brackets **[]**, while tuples are created with parentheses **()**.
3. **Performance:** Tuples are faster than lists in terms of indexing and iteration speed. This is because tuples are stored in memory as a single block of data, while lists are stored as separate objects that require more memory to access.
4. **Use Cases**: Lists are more commonly used for general-purpose storage of items, whereas tuples are typically used to store fixed, unchanging collections of items, such as the coordinates of a point in space.

For example:

**spam = [1, 2, 3, 4]**

**spam[2] = 5**

**print(spam)**

**[1, 2, 5, 4]**

**eggs = (1, 2, 3, 4)**

**eggs[2] = 5**

**Traceback (most recent call last):**

**File "<stdin>", line 1, in <module>**

**TypeError: 'tuple' object does not support item assignment**

In summary, the main difference between tuples and lists is mutability: lists are mutable and can have their elements changed, while tuples are immutable and cannot have their elements changed once created.

Q14. How do you type a tuple value that only contains the integer 42?

Ans: A tuple value that only contains the integer 42 can be written as: **(42,)**.

Note the comma after 42. In Python, a tuple with only one item must have a comma after the item, otherwise it will be treated as an expression with parentheses, not a tuple.

Q15. 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 get a list value's tuple form using the **tuple()** function. To get a tuple value's list form, you can use the **list()** function.

Example:

**spam = [1, 2, 3, 4]**

**eggs = tuple(spam)**

**print(eggs)**

**(1, 2, 3, 4)**

**bacon = (5, 6, 7, 8)**

**ham = list(bacon)**

**print(ham)**

**[5, 6, 7, 8]**

Q16. Variables that “contain” list values are not necessarily lists themselves. Instead, what do they contain?

Ans: Variables that "contain" list values actually contain references to the lists. In Python, variables do not store the values directly, but instead store references to the objects that hold the values.

So, when a list is assigned to a variable, the variable actually refers to the list object in memory, rather than containing the list itself. When the list is modified, the changes are reflected in the reference, and thus in any variables that refer to the same list.

Q17. How do you distinguish between copy.copy() and copy.deepcopy()?

Ans: In Python, **copy.copy()** and **copy.deepcopy()** are both functions from the **copy** module that are used to make copies of objects. However, they differ in the type of copy they create.

**copy.copy()** creates a shallow copy of an object. A shallow copy creates a new object with a new reference, but the elements inside the object are still referenced to the original object. If the original object contains any mutable objects, such as lists or dictionaries, changes to those objects will be reflected in both the original object and the shallow copy.

**copy.deepcopy()** creates a deep copy of an object. A deep copy creates a new object with a new reference and all elements inside the object are also new objects with new references. Changes to the elements inside the object will not be reflected in the original object.

Example:

import copy

spam = [1, 2, [3, 4]]

eggs = copy.copy(spam)

print(eggs)

[1, 2, [3, 4]]

spam[2].append(5)

print(eggs)

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

bacon = [1, 2, [3, 4]]

ham = copy.deepcopy(bacon)

print(ham)

[1, 2, [3, 4]]

bacon[2].append(5)

print(ham)

[1, 2, [3, 4]]

In summary, **copy.copy()** creates a shallow copy of an object, while **copy.deepcopy()** creates a deep copy of an object, including all elements inside the object.