Here are the answers to your questions:

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

- `[]` represents an empty list in Python. It is a list object with 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`.)\*\*

- You can assign `'hello'` as the third value by indexing into the list and setting the value:

```python

spam[2] = 'hello'

```

- This changes the list to `[2, 4, 'hello', 8, 10]`.

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

- Breaking it down:

- `'3' \* 2` results in `'33'`.

- `int('33')` converts `'33'` to `33`.

- `33 / 11` results in `3.0`, and `int(3.0)` converts it to `3`.

- So, `spam[3]` is the value at index 3 in `spam`, which is `'d'` if `spam` is `['a', 'b', 'c', 'd']`.

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

- `spam[-1]` retrieves the last element of the list `spam`. If `spam` is `['a', 'b', 'c', 'd']`, then `spam[-1]` is `'d'`.

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

- `spam[:2]` returns a slice of the list from the beginning up to but not including index 2. For `spam = ['a', 'b', 'c', 'd']`, `spam[:2]` is `['a', 'b']`.

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

- `bacon.index('cat')` returns the index of the first occurrence of `'cat'` in the list `bacon`. Given `bacon = [3.14, 'cat', 11, 'cat', True]`, the value is `1`.

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

- `bacon.append(99)` adds the value `99` to the end of the list. After the operation, `bacon` would look like `[3.14, 'cat', 11, 'cat', True, 99]`.

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

- `bacon.remove('cat')` removes the first occurrence of `'cat'` from the list. After the operation, `bacon` would look like `[3.14, 11, 'cat', True, 99]`.

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

- List concatenation is done with the `+` operator. For example: `[1, 2] + [3, 4]` results in `[1, 2, 3, 4]`.

- List replication is done with the `\*` operator. For example: `[1, 2] \* 3` results in `[1, 2, 1, 2, 1, 2]`.

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

- `append()` adds an element to the end of the list. For example: `list.append(5)`.

- `insert()` adds an element at a specific position. For example: `list.insert(2, 5)` inserts `5` at index `2`.

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

- `remove(value)` removes the first occurrence of `value` from the list.

- `pop(index)` removes and returns the item at the specified index.

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

- Both lists and strings are sequences that support indexing, slicing, and iteration. They are both ordered collections of elements.

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

- Lists are mutable (they can be modified), while tuples are immutable (they cannot be modified once created). Lists use square brackets `[]`, while tuples use parentheses `()`.

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

- You need a trailing comma to define a single-element tuple:

```python

single\_element\_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 a list to a tuple: `tuple(list\_value)`.

- To convert a tuple to a list: `list(tuple\_value)`.

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

- Variables that contain list values actually hold references to list objects, not the list objects themselves.

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

- `copy.copy()` performs a shallow copy, which copies the object but not the objects it contains (nested objects are still referenced).

- `copy.deepcopy()` performs a deep copy, which copies the object and all objects nested within it, creating entirely new copies.