PythonAssignment4

February 28, 2023

1 Python Assignment 4

1. What exactly is []?
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.) 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)]?
4. What is the value of spam[-1]?
5. 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.
6. What is the value of bacon.index('cat')?
7. How does bacon.append(99) change the look of the list value in bacon?
8. How does bacon.remove('cat') change the look of the list in bacon?
9. What are the list concatenation and list replication operators?
10. What is difference between the list methods append() and insert()?
11. What are the two methods for removing items from a list?
12. Describe how list values and string values are identical.
13. What's the difference between tuples and lists?
14. How do you type a tuple value that only contains the integer 42?
15. How do you get a list value's tuple form? How do you get a tuple value's list form?
16. Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?
17. How do you distinguish between copy.copy() and copy.deepcopy()?

1.0.1 1. What exactly is [] ?

[] is what you can assign to a variable in order to create an empty list. Furthermore, if you use it to create lists by inserting comma separated values or objects and assigning to a variable.

1.0.2 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.)

[1]:
```
# Assign 'hello' to the third value

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

spam[2] = 'hello'
```

```
spam
```

```
[1]: [2, 4, 'hello', 8, 10]
```

1.0.3 Questions 3 -5

```
[2]: spam = ['a', 'b', 'c', 'd']



     # 3
     spam[int(int('3'*2)/11)]

     # First, the int('3'*2) returns and integer 6
     # Second, 6 is divided by 11, this will be a float, but it is type cast as an␣
      ↪integer
     # 6/11 is approximately
```

```
[2]: 'd'
```

```
[3]: #4 Acccess the last element in the list
     spam[-1]
```

```
[3]: 'd'
```

```
[4]: #5 Access from the 0th to, but not including the 2nd index
     spam[:2]
```

```
[4]: ['a', 'b']
```

#6 Bacon and Cats

```
[13]: bacon = [3.14, 'cat', 11, 'cat', True]
```

```
[6]: #6 the index() function will return the location
     #of the argument passed
     bacon.index('cat')
```

```
[6]: 1
```

```
[15]: #7 The append() function will add the passed
      # argument to the end of the iterable



      bacon.append(99)
      bacon
```

```
[15]: [3.14, 'cat', 11, 'cat', True, 99]
```

[16]:
```
#8 The remove() function removes the past argument from
#the iterable
bacon.remove('cat')
```

[17]:
```
bacon
```

[17]:
```
[3.14, 11, 'cat', True, 99]
```

[37]:
```
#9 What are the list concatenation and ilst replication operators?

l1 = ['this', 'list', 'will', 'be']
l2 = ['concatenated', 'with', 'this', 'list']
```

[2]:
```
#we can concatenate a list simliar to a string
l1 + l2
```

[2]:
```
['this', 'list', 'will', 'be', 'concatenated', 'with', 'this', 'list']
```

[3]:
```
# I can replicate a list like this:
l1_1 = l1.copy()
```

[4]:
```
l1
```

[4]:
```
['this', 'list', 'will', 'be']
```

[5]:
```
l1_1
```

[5]:
```
['this', 'list', 'will', 'be']
```

[38]:
```
#10 What is the difference between the list methods append() and insert()
#If I try to append a list with an object, it goes to the end of the list

l1.append(l2)
```

[39]:
```
l1
```

[39]:
```
['this', 'list', 'will', 'be', ['concatenated', 'with', 'this', 'list']]
```

[40]:
```
#If I try to insert() the list it asks for the index which I would like to␣
 ↪insert the object
l1.insert(0,l2)
```

[41]:
```
l1
```

[41]:
```
[['concatenated', 'with', 'this', 'list'],
 'this',
 'list',
 'will',
 'be',
 ['concatenated', 'with', 'this', 'list']]
```

[42]:
```
# 11 What are the methods for removing items from a list
#Remove will take off the first instance of the item
l1.remove(l2)
```

[43]:
```
l1
```

[43]:
```
['this', 'list', 'will', 'be', ['concatenated', 'with', 'this', 'list']]
```

[44]:
```
# Pop will remove an item at a specified index
l1.pop(4)
```

[44]:
```
['concatenated', 'with', 'this', 'list']
```

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

A list a list of strings (unless it is integers or floats that are not stored as strings). A list can be interated over and indexed and sliced

A string a string of strings (unless it is integers or floats that are not stored as strings). A string can be interated over and indexed and sliced

[45]:
```
#Examine the attributes of each object

a = 'my string'

dir(a)
```

[45]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
 'partition',
 'removeprefix',
  'removesuffix',
  'replace',
  'rfind',
  'rindex',
  'rjust',
  'rpartition',
  'rsplit',
  'rstrip',
  'split',
  'splitlines',
  'startswith',
  'strip',
  'swapcase',
  'title',
  'translate',
  'upper',
  'zfill']
```

[46]:
```
b = ['my', 'list']

dir(b)
```

[46]:
```
['__add__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

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

1. The primary diffeence between a list and a tuple is that a tuple is immutable and a list is mutable. Since this is the case, a list will use more memory than a tuple. Subsequently, tuples will be faster than lists. The other difference in this the syntax to create each one

a_list = ['The List uses square brackets']

a_tuple = ('The tuple uses round brackets')

It is a good idea to use tuples when you have data that you do not want to be changed.

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

Typically, you cannot create a tuple of a single element. However, you can do it if you put a comma after the value.

[2]:
```
only_forty_two = (42,)
only_forty_two
```

[2]:
```
(42,)
```

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

In either case you simply wrap the object in list() or tuple() as the case may be.

```
[7]: a_list = ['a','b','c','d',1,2,3]

     print(a_list)
```

```
['a', 'b', 'c', 'd', 1, 2, 3]
```

```
[11]: a_tuple = tuple(a_list)
      print(a_tuple)
```

```
('a', 'b', 'c', 'd', 1, 2, 3)
```

```
[13]: a_list_again = list(a_tuple)

      print(a_list_again)
```

```
['a', 'b', 'c', 'd', 1, 2, 3]
```

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

If a variable stores a list object, then it acts as a pointer to the location of the list rather than storing the contents of the list itself.

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

1. copy.copy() is create shallow copy. This means that if any changes are made to either the original or copy, the changes will be reflected in both.
2. copy.deepcopy() is a copy that will not be changed, i.e. it is independent of the original.