Python_Variable_HW

February 6, 2023

1 Variables homework

Advice for homework and life: try, play, experiment; don't Google until you get frustrated.

You can't break anything with Python, so, if you want to figure something out, try until you get it to work. You'll learn more and you'll remember it longer and more deeply.

In general, the homework question will consist of the question in a markdown cell, a code cell for you to play in, repeating as needed, and a final markdown cell for your answer or explanation *that will be in italics.*

But do feel free to add or delete cells as you feel appropriate. The important thing is you get your point across!

1.0.1 1.

Make an integer: `theAnswer = 42`. Now make another: `anotherNameForTheAnswer = 42` (You don't have to use these exactly, but make sure you have two different names referring to the same exact value.)

[2]:
```
theans = 88
anotherans = 88
```

Get and note the ID numbers for both.

[3]:
```
id(theans)
```

[3]:
```
4351043184
```

[4]:
```
id(anotherans)
```

[4]:
```
4351043184
```

Assign each name to a completely different number and confirm that each name now refers to an object with a new ID.

[5]:
```
theans = 48
anotherans = 84
```

[6]:
```
id(theans)
```

[6]:
```
4351041904
```

[7]:
```
id(anotherans)
```

[7]:
```
4351043056
```

Do a `whos` to confirm that *no* names refer to the original value.

[8]:
```
whos
```

```
Variable     Type    Data/Info
------------------------------
anotherans   int     84
theans       int     48
```

Finally, assign a new name to the original value, and get its ID.

[9]:
```
anAns = 88
```

[10]:
```
id(anAns)
```

[10]:
```
4351043184
```

Look at this ID and compare it to the original ones. What happened? What does this tell you?

The ID is the same as the original ones. The integer itself still has the ID attached to it so when assigning this new name to the value it still has the same original identification.

1.0.2 2.

Convert both possible Boolean values to strings and print them.

[8]:
```
boolOne = True
boolTwo = False

print(str(boolOne), str(boolTwo))
```

```
True False
```

Now convert some strings to Boolean until you figure out "the rule".

[6]:
```
strOne = 'True'
strTwo = 'False'
strThree = '0'
```

```
print(bool(strOne), bool(strTwo), bool(strThree), bool(''))
```

```
True True True False
```

What string(s) convert to True and False? In other words, what is the rule for str -> bool conversion?

All non-empty strings convert to True. The only False I got was from an empty string.

1.0.3 3.

Make three variables:

- a Boolean equal to True
- an int (any int)
- a float

Try all combinations of adding two of the variables pairwise.

[24]:
```
aBool = True
anInt = 6
aFloat = 6.6
```

[25]:
```
aBool + anInt
```

[25]:
```
7
```

[26]:
```
aBool + aFloat
```

[26]:
```
7.6
```

[27]:
```
aFloat + anInt
```

[27]:
```
12.6
```

What is the rule for adding numbers of different types?

The boolean value equal to True is equal to 1. Both the integers and float values stayed the same.

1.0.4 4.

Make an int (any int) and a string containing a number (e.g. num_str = '64'). Try

- adding them
- adding them converting the number to a string
- adding them converting the string to a number

```
[28]: anyInt = 8
      num_str = '2'
```

```
[29]: anyInt + num_str
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 1
----> 1 anyInt + num_str

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
[30]: str(anyInt) + num_str
```

```
[30]: '82'
```

```
[31]: int(num_str) + anyInt
```

```
[31]: 10
```

Try converting a `str` that is a spelled out number (like 'forty two') to an int.

```
[33]: int('nine')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[33], line 1
----> 1 int('nine')

ValueError: invalid literal for int() with base 10: 'nine'
```

Did that work?

No it did not!

1.0.5 5.

Make a variable that is a 5 element tuple.

```
[34]: colors = ('pink', 'green', 'yellow', 'red', 'purple')
```

Extract the last 3 elements.

```
[36]: colors[2:5]
```

```
[36]: ('yellow', 'red', 'purple')
```

1.0.6 6.

Make two variables containing tuples (you can create one and re-use the one from #5). Add them using "+".

```
[40]: shapes = ('circle', 'square', 'triangle', 'oval', 'diamond')

      colors + shapes
```

```
[40]: ('pink',
       'green',
       'yellow',
       'red',
       'purple',
       'circle',
       'square',
       'triangle',
       'oval',
       'diamond')
```

Make two list variables and add them.

```
[41]: snack = ['peanuts','raisins','chips']
      animals = ['dog','cat','bird']

      snack + animals
```

```
[41]: ['peanuts', 'raisins', 'chips', 'dog', 'cat', 'bird']
```

Try adding one of your tuples to one of your lists.

```
[42]: shapes + snack
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[42], line 1
----> 1 shapes + snack

TypeError: can only concatenate tuple (not "list") to tuple
```

What happened? How does this compare to adding, say, a bool to a float?

It cannot add a list to a tuple. A bool and a float works because the boolean value is converted to an integer which can used in addition.

1.0.7 7.

Can you tell the type of a variable by looking at its value?

If so, how? A couple examples are fine; no need for an exhaustive list.

Yes I can. A boolean variable is easy to recognize because it's either True or False. An integer is a whole number (i.e. 8 or 10), and a float variable includes a decimal (i.e. 8.5 or 0.05). A string variable is a sequence of characters (i.e. 'dog') and is enclosed in a parentheses.

1.0.8 8.

Make a list variable in which one of the elements is itself a list (e.g. `myList = ['hi', [3, 5, 7, 11], False]`).

```
[43]: myList = ['pluto', [2,3,4,5], True]
```

Extract one element of the nested list - the list-within a list. Try it in two steps, by first extracting the nested list and assigning it to a new variable.

```
[46]: nested = myList[1]
      nested[1]
```

```
[46]: 3
```

Now see if you can do this in one step.

```
[47]: myList[1][1]
```

```
[47]: 3
```

1.0.9 9.

Make a `dict` variable with two elements, one of which is a list.

```
[48]: cars_dict = {
          'type':'Jeep',
          'color':['white','black','yellow']
      }
```

Extract a single element from the list-in-a-dict in one step.

```
[50]: cars_dict['color'][0]
```

```
[50]: 'white'
```

10.

Make a list variable. Consider that each element of the list is logically an *object* in and of itself. Confirm that one or two of these list elements has its own unique ID number.

[61]:
```
anotherList = [2,4,6,8]
print(id(anotherList[0]), id(anotherList[3]))
```

```
4354120624 4354120816
```

If you extract an element from your list and assign it to a new variable, are the IDs the same, or is a new object created?

[63]:
```
elem = anotherList[0]
id(elem)
```

[63]:
```
4354120624
```

Are the IDs the same, or is a new object created when you assigned the list element to new variable?

The IDs are the same because the object is unchanged, regardless of the new name assignment.

1.0.10 11.

Make a `str` variable containing the first 5 letters of the alphabet (e.g. `a2e = 'abcde'`). Check the ID of the second (index = 1) element (the 'b').

[75]:
```
ae = 'abcde'
id(ae[1])
```

[75]:
```
4354166688
```

Now make a `str` variable containing the letter 'b'. Check its ID.

[72]:
```
letter = 'b'
id(letter)
```

[72]:
```
4353865568
```

What happened?

The ID is different because one variable is 'abcde'while the other is just considered 'b'. (i'm not sure about this one)