Python_Variable_HW

January 26, 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.)

[1]:
```
theAnswer = 42
anotherNameForTheAnswer = 42
```

Get and note the ID numbers for both.

[2]:
```
id(theAnswer)
```

[2]:
```
140617648840272
```

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

[3]:
```
140617648840272
```

- `Wow! They have the same ID.`

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

[4]:
```
theAnswer = 90
anotherNameForTheAnswer = 100
```

[5]:
```
id(theAnswer)
```

[5]:
```
140617649030288
```

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

[6]:
```
140617649030608
```

- Now, they are different.

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

[8]:
```
whos
```

```
Variable                   Type    Data/Info
-------------------------------------------
anotherNameForTheAnswer    int     100
theAnswer                  int     90
```

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

[9]:
```
thenewanswer = 42
id(thenewanswer)
```

[9]:
```
140617648840272
```

- Although, they are the same answers of 42. But, they do not have the same ID.

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

1.0.2 2.

Convert both possible Boolean values to strings and print them.

[10]:
```
boolean1 = True
```

[11]:
```
str(boolean1)
```

[11]:
```
'True'
```

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

[16]:
```
st = 'Hello'
st2 = bool(st)
st2
```

[16]: True

What string(s) convert to True and False? In other words, what is the rule for str -> bool conversion? * You can still convert str to bool. But, the 'Hello' is now convert to 'True'.

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.

[17]:
```
bool3 = True
int3 = 5
flt3 = 3.4444
```

[18]:
```
int3 + flt3
```

[18]: 8.4444

[19]:
```
flt3 + bool3
```

[19]: 4.4444

[20]:
```
bool3 + int3
```

[20]: 6

What is the rule for adding numbers of different types?

- You can add them all together without any error showing up. However, Booleans will be counted as '1'.

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

[22]:
```
int4 = 10
str4 = '67'
```

[23]:
```
int4 + str4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [23], in <cell line: 1>()
----> 1 int4 + str4

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

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

[27]:
```
str4_2 = 'forty two'
int(str4_2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [27], in <cell line: 2>()
      1 str4_2 = 'forty two'
----> 2 int(str4_2)

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

Did that work?

- No, it does not work.

1.0.5 5.

Make a variable that is a 5 element tuple.

[4]:
```
tup5 = (1,2,3,4,5)
```

Extract the last 3 elements.

[5]:
```
tup5[-3:5]
```

[5]:
```
(3, 4, 5)
```

1.0.6 6.

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

[2]:
```
tup6 = (6,7,8,9,10)
```

Make two list variables and add them.

[6]:
```
tup5 + tup6
```

[6]:
```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

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

[7]:
```
tup6 + tup5[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 tup6 + tup5[0]

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

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

- Although we can add the whole list of tuple together, I cannot add individual value from one list to another.

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.

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]`).

[10]:
```
list8 = ['Jay', [1,2,3,4,5], True]
list8
```

[10]:
```
['Jay', [1, 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.

[14]:
```
list8[0]
```

[14]:
```
'Jay'
```

[13]:
```
list8[1]
```

[13]:
```
[1, 2, 3, 4, 5]
```

[15]:
```
list8[2]
```

[15]:
```
True
```

[17]:
```
list8_1 = list8[1]
list8_1
```

[17]:
```
[1, 2, 3, 4, 5]
```

[18]:
```
list8_1[0]
```

[18]:
```
1
```

Now see if you can do this in one step.

[20]:
```
list8[1][0]
```

[20]:
```
1
```

1.0.9 9.

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

[22]:
```
dict9 = {'Name': 'Jay',
         'Age': 22,
         'List': [1,2,3,4,5]}
dict9
```

[22]:
```
{'Name': 'Jay', 'Age': 22, 'List': [1, 2, 3, 4, 5]}
```

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

[24]:
```
dict9['List'][0]
```

[24]:
```
1
```

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.

[28]:
```
list10 = [1,2,3,'Jay',True]
list10_1 = list10[0]
list10_2 = list10[3]
```

[30]:
```
id(list10)
```

[30]: 140460042125568

[31]:
```
id(list10_2)
```

[31]: 140460035815408

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? * `The new ID was created for the extracted value from the list.`

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

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').

[34]:
```
str11 = 'abcde'
```

[37]:
```
id(str11[1])
```

[37]: 140459964003120

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

[38]:
```
str11_2 = 'b'
```

[39]:
```
id(str11_2)
```

[39]: 140459964003120

What happened?

- `Although they have different object name, the object that contain 'b' within them have the same ID.`