Python_Variable_HW_FINAL

January 26, 2024

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]:
```
4354552440
```

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

[4]:
```
4354552440
```

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

[5]:
```
theAnswer = 7
anotherNameForTheAnswer = 7
```

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

[6]:
```
whos
```

```
Variable                   Type    Data/Info
-----------------------------------------
anotherNameForTheAnswer    int     7
theAnswer                  int     7
```

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

[7]:
```
originalOne = 42
id(originalOne)
```

[7]:
```
4354552440
```

The id's for the original one and the new one made are the same. This shows that integers are immutable–they cannot be changed after they are made.

1.0.2 2.

Convert both possible Boolean values to strings and print them.

[10]:
```
bol1 = True
bol0 = False
str(bol1)
```

[10]:
```
'True'
```

[11]:
```
str(bol0)
```

[11]:
```
'False'
```

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

[12]:
```
bool("Hello There")
```

[12]:
```
True
```

[13]:
```
bool("False")
```

[13]:
```
True
```

[16]:
```
bool("")
```

[16]:
```
False
```

[17]:
```
bool("I love tea")
```

[17]: True

Essentially everything will be True except blank strings they will be False.

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.

[18]:
```
varBoolean = True
varInt = 77
varFlo = 5.236
```

[19]:
```
varBoolean + varInt
```

[19]: 78

[20]:
```
varBoolean + varFlo
```

[20]: 6.236

[21]:
```
varFlo + varInt
```

[21]: 82.236

If you add an integer and a float the results will be a float. If you add a boolean to a float or integer, False would be considered as 0 and True will be considered 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]:
```
anyint = 88
num_str = '10'
```

[24]:
```
anyint + int(num_str)
```

[24]: 98

[25]:
```
str(anyint) + (num_str)
```

[25]:
```
'8810'
```

[26]:
```
anyint + num_str
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 anyint + num_str

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.

[28]:
```
num_spelled = 'forty two'
int(num_spelled)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[28], line 2
      1 num_spelled = 'forty two'
----> 2 int(num_spelled)

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

That did not workout, there was an error message

1.0.5 5.

Make a variable that is a 5 element tuple.

[29]:
```
five_element = (1, 4, 'five', 7, 88)
```

Extract the last 3 elements.

[31]:
```
five_element[-3:]
```

[31]:
```
('five', 7, 88)
```

1.0.6 6.

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

[32]:
```
my_tupple = (6, 7, 8, 19, 66)
my_tupple + five_element
```

[32]:
```
(6, 7, 8, 19, 66, 1, 4, 'five', 7, 88)
```

Make two list variables and add them.

[33]:
```
list1 = (6, 8, 9)
list2 = (9, 76, 89)
list1 + list2
```

[33]:
```
(6, 8, 9, 9, 76, 89)
```

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

[34]:
```
list1 + my_tupple
```

[34]:
```
(6, 8, 9, 6, 7, 8, 19, 66)
```

Essentially, the contents of the tupple's elements get added to the list, whereas with bools and floats the the bool is converted to the integer value and it adds via addition and you get the sum of the two.

1.0.7 7.

Yes, but not everytime, because you can have a string that contains a number but if you print() it comes out as just the integer, so you may not differentiate

x = 4, y = 'Hello', u = 3.14

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]`).

[47]:
```
Listing = ['hello', [3, 6, 9], 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.

[55]:
```
Listing[1]
Listing3 = Listing[1]
```

[56]:
```
print(Listing3)
```

```
[3, 6, 9]
```

Now see if you can do this in one step.

[58]:
```
Listing4 = Listing[1]
print(Listing4)
```

```
[3, 6, 9]
```

1.0.9 9.

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

[59]:
```
mydic = {'one': 'value', 'one2': [1, 5, 6]}
```

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

[61]:
```
anElement = mydic['one2'][1]
print(anElement)
```

```
5
```

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.

[62]:
```
listVariable = [1, "Hello", [3, 88], 7.00]
```

The id's are the same, since they reference the same object

[63]:
```
id(listVariable[0])
```

[63]:
```
4354551128
```

[64]:
```
new = listVariable[0]
id(new)
```

[64]:
```
4354551128
```

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').

[65]:
```
alpha = 'abcde'
id(alpha[1])
```

[65]:
```
4354597224
```

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

[68]:
```
em = 'b'
id(em)
```

[68]: 4354296056

They have different id numbers, each string is immutable so it is different in the memory.