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.

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.)

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

Get and note the ID numbers for both.

In [2]:
```
print(id(theAnswer),id(anotherNameForTheAnswer))
```

```
4338872712 4338872712
```

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

In [3]:
```
theAnswer = 4
anotherNameForTheAnswer = 2

print(id(theAnswer),id(anotherNameForTheAnswer))
```

```
4338871496 4338871432
```

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

In [4]:
```
whos
```

```
Variable                   Type    Data/Info
--------------------------------------------
anotherNameForTheAnswer    int     2
theAnswer                  int     4
```

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

In [5]:
```python
new42 = 42
id(new42)
```

Out[5]:
```
4338872712
```

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

We see that the ID of new42 is the ID originally grabbed from our second code cell

---

2.

Convert both possible Boolean values to strings and print them.

In [8]:
```python
print(str(True),str(False))
```

```
True False
```

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

In [9]:
```python
print(bool(0),bool(1),bool(-5),bool(5))
```

```
False True True True
```

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

If the int is 0, return false else true.

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.

In [12]:
```
trueBool = True
int42 = 42
float42 = 42.00

print(trueBool + int42)
print(float42 + int42)
print(trueBool + float42)
```

```
43
84.0
43.0
```

What is the rule for adding numbers of different types?

in priority:

boolean < int < float

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

In [13]:
```
num_str = '64'
num_int = 64

print(num_int + num_str)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 4
      1 num_str = '64'
      2 num_int = 64
----> 4 print(num_int + 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.

In [15]:
```
fortytwo = 'forty two'
print(int(fortytwo))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[15], line 2
      1 fortytwo = 'forty two'
----> 2 print(int(fortytwo))

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

Did that work?

no it did not

5.

Make a variable that is a 5 element tuple.

In [16]:
```
fiveTuple = (0,1,2,3,4)
```

Extract the last 3 elements.

In [18]:
```
fiveTuple[2:]
```

Out[18]:
```
(2, 3, 4)
```

6.

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

In [20]:
```
otherTuple = (4,3,2,1,0)

print(otherTuple + fiveTuple)
```

```
(4, 3, 2, 1, 0, 0, 1, 2, 3, 4)
```

Make two list variables and add them.

In [21]:
```
fiveList = [0,1,2,3,4]
otherList = [4,3,2,1,0]

fiveList + otherList
```

Out[21]:
```
[0, 1, 2, 3, 4, 4, 3, 2, 1, 0]
```

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

In [22]:
```
fiveList + fiveTuple
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 fiveList + fiveTuple

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

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

there is no equivalent since one form is mutable and one isn't

---

7.

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

You are able to see it through how it is viewed within your IDE. Like how a float will always have its integers contain a .00 while int variables will just say the int value. Floats also will show the '' around them

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

8.

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

```
In [23]: myList = ['hi', [3, 5, 7, 11], False]
```

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.

```
In [27]: withinList = myList[1]
         withinList[3]

Out[27]: 11
```

Now see if you can do this in one step.

```
In [28]: myList[1][3]

Out[28]: 11
```

9.

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

```
In [34]: exDict = {'name': 'zane',
                   'values': [222,333,444]}
```

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

```
In [35]: exDict['values'][0]

Out[35]: 222
```

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.

```
In [38]: listVar = 33
         lllist = [22,11,listVar]
         print(id(lllist[0]),id(lllist[2]))

4338872072 4338872424
```

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?

```
In [40]: lllist[0] = 381
         id(lllist[0])

Out[40]: 4541294608
```

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

It creates a new ID that is assigned to the variable

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').

```
In [41]: alphabet = 'abcde'
         id(alphabet[1])

Out[41]: 4338917496
```

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

```
In [43]: bStr = 'b'
         id(bStr)
```

```
Out[43]: 4338616328
```

What happened?

The IDs are different