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 [2]:
```
answer = 32
answered = 32
```

Get and note the ID numbers for both.

In [3]:
```
id(answer)
```

Out[3]:
```
140720760854280
```

In [12]:
```
id(answered)
```

Out[12]:
```
140720760854152
```

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

In [10]:
```
answer = 30
answered = 28
id(answer)
```

Out[10]:
```
140720760854216
```

In [11]:
```
id(answered)
```

Out[11]:
```
140720760854152
```

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

In [13]:
```
whos
```

```
Variable   Type    Data/Info
--------------------------
answer     int     30
answered   int     28
```

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

In [15]:
```
ans = 30
id(ans)
```

Out[15]:
```
140720760854216
```

In [16]:
```
ansd = 28
id(ansd)
```

Out[16]:
```
140720760854152
```

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

2.

Convert both possible Boolean values to strings and print them.

In [17]:
```
mytrueval = True
mytval = "tval"
print(mytval)
print(type(mytval))
```

```
tval
<class 'str'>
```

In [26]:
```
myfalseval = False
myfalseval = ""
print(myfalseval)
print(type(myfalseval))
```

```
<class 'str'>
```

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

In [18]:
```
myboolt = bool(mytval)
print(type(myboolt))
print(myboolt)
```

```
<class 'bool'>
True
```

In [27]:
```
myboolf = bool(myfalseval)
print(myboolf)
print(type(myboolf))
```

```
False
<class 'bool'>
```

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

In [30]:
```
print(f"Any strings that are anything but 0 or have anything in the "" will be True an
```

```
Any strings that are anything but 0 or have anything in the  will be True and vice ve
rsa for False
```

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 [31]:
```
t = 6
f = 10
bool (f > t)
```

Out[31]:
```
True
```

In [32]:
```
beet = 70
```

In [33]:
```
tea = 4.15
```

In [34]:
```
beet + tea
```

Out[34]:
```
74.15
```

In [38]:
```
beet + bool (f > t)
```

Out[38]:
```
71
```

In [39]:
```
tea + bool (f > t)
```

Out[39]:
```
5.15
```

What is the rule for adding numbers of different types?

In [ ]:
```
The rule is that they must all be true numbers?
```

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 [40]:
```
love = 2
lstr = "60"
```

In [41]:
```
love + lstr
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[41], line 1
----> 1 love + lstr

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

In [43]:
```
love = "luv"
love + lstr
```

Out[43]:
```
'luv60'
```

In [44]:
```
lstr = 60
love + lstr
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 2
      1 lstr = 60
----> 2 love + lstr

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

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

In [47]:
```
um = "three"
um = 3
three = 3
```

Did that work?

5.

Make a variable that is a 5 element tuple.

In [49]:
```
myTup = (3,6,9,12,15)
myTup
```

Out[49]:
```
(3, 6, 9, 12, 15)
```

Extract the last 3 elements.

In [80]:
```
myTup[2],
myTup[3],
myTup[4]
```

Out[80]:
```
15
```

6.

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

In [85]:
```
teacup = (2,4,6,8,10)
print(teacup)
```

```
(2, 4, 6, 8, 10)
```

In [86]:
```
(myTup + teacup )
```

Out[86]:
```
(3, 6, 9, 12, 15, 2, 4, 6, 8, 10)
```

Make two list variables and add them.

In [91]:
```
lit = [1,
       3,
       5,
       7]
print(lit)

mit = [10,
       20,
       30,
       40]
print(mit)
```

```
[1, 3, 5, 7]
[10, 20, 30, 40]
```

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

In [92]:
```
myTup + lit
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[92], line 1
----> 1 myTup + lit

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

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

In [94]:
```
print(f"It didn't work. Adding a bool to float will work because it equates to an inte
```

```
It didn't work. Adding a bool to float will work because it equates to an integer (0,
1) to another number.
```

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.

In [97]:
```
#You can tell if it is an integer if when your print it it comes with a number only li
print(tea)
```

```
4.15
```

In [99]:
```
#You can tell if it is a list by how the numbers come out in a square bracket vs a tup
print(lit)
print(myTup)
```

```
[1, 3, 5, 7]
(3, 6, 9, 12, 15)
```

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 [101…
```
listed = ['hola', [5,4,3,2], True, 'run']
```

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 [110…
```
listed[1]
extract = listed[1]
extract
```

Out[110]:
```
[5, 4, 3, 2]
```

Now see if you can do this in one step.

In [112…
```
extract[2]
```

Out[112]:
```
3
```

9.

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

In [114…
```
dict = {'love': 'cookies',
        'list1': [1,2,3,4]
       }
```

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

In [118…
```
dict['love']
```

Out[118]:
```
'cookies'
```

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 [132…
```
listagain = ['hi', [2,3,4], 'bye', False]
id(False)
id('bye')
```

Out[132]:
```
1574970953200
```

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 [134…
```
same = 'bye'
id(same)
```

Out[134]:
```
1574970953200
```

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

In [ ]:
```
The Id's are the same
```

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 [138…
```
alphabet = 'abcde'
id('b')
```

Out[138]:
```
140720723323960
```

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

In [140…
```
lastthing = 'badbunny'
id('b')
```

Out[140]:
```
140720723323960
```

What happened?

In [ ]:
```
The id stayed the same
```