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]: rnd = 7
        rndw = 7
        rnd

Out[1]: 7
```

```
In [2]: rndw

Out[2]: 7
```

Get and note the ID numbers for both.

In [3]:
```
id(rnd)
```
Out[3]:
```
4528066072
```

In [4]:
```
id(rndw)
```
Out[4]:
```
4528066072
```

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

In [5]:
```
rnd = 8
rndw = 8
rnd
```
Out[5]:
```
8
```

In [6]:
```
rndw
```
Out[6]:
```
8
```

In [8]:
```
id(rnd)
print(id(rndw))
```
```
4528066104
```

In [9]:
```
id(rndw)
```
Out[9]:
```
4528066104
```

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

In [11]:
```
whos
```
```
Variable   Type    Data/Info
----------------------------
rnd        int     8
rndw       int     8
```

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

In [12]:
```
rndoh = 7
rndoh
id(rndoh)
```
Out[12]:
```
4528066072
```

Look at this ID and compare it to the original ones. What happened? What does this tell you?
it looks like changing the name assign to a number or value doesn't make a new id based on how many times we do it.

2.

Convert both possible Boolean values to strings and print them.

In [13]:
```
w = "True!"
w
```
Out[13]:
```
'True!'
```

In [14]:
```
e = "False!"
e
```
Out[14]:
```
'False!'
```

In [15]:
```
print(e)
```
```
False!
```

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

In [ ]:
```

```

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

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 [16]: t = True
         a = 1
         b = 2.3

         a + b
Out[16]: 3.3
```

```
In [17]: t + a
Out[17]: 2
```

```
In [18]: t + b
Out[18]: 3.3
```

```
In [19]: a + a
Out[19]: 2
```

```
In [20]: t + t
Out[20]: 2
```

```
In [21]: b + b
Out[21]: 4.6
```

What is the rule for adding numbers of different types?

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 [22]:
```
r = 9
num_str = "10"
```

In [23]:
```
r + num_str
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 r + num_str

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

In [24]:
```
r = "9"
r + num_str
```

Out[24]:
```
'910'
```

In [25]:
```
num_str = 10
num_str + r
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[25], line 2
      1 num_str = 10
----> 2 num_str + r

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

In [26]:
```
r = 9
r + num_str
```
Out[26]:
```
19
```

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

In [29]:
```
str = 'twelve'
```

Did that work? yes

5.

Make a variable that is a 5 element tuple.

In [30]:
```
mytuple = (1, 4, 6, 7, 9)
mytuple
```
Out[30]:
```
(1, 4, 6, 7, 9)
```

Extract the last 3 elements.

In [31]:
```
mytuple[2:5]
```
Out[31]:
```
(6, 7, 9)
```

6.

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

In [32]:
```




tuple_1 = (2, 3, 7)
tuple_2 = (5, 8, 9)
```

In [33]:
```
tuple_1 + tuple_2
```

Out[33]:
```
(2, 3, 7, 5, 8, 9)
```

Make two list variables and add them.

In [34]:
```
list_1 = ['moi', 'ofcourse', 'two', 21]
```

In [35]:
```
list_2 = [2, 'ok', '3.8']
```

In [36]:
```
list_1+list_2
```

Out[36]:
```
['moi', 'ofcourse', 'two', 21, 2, 'ok', '3.8']
```

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

In [37]:
```
list_1 + tuple_1
```

```
---------------------------------------------------------------------
------
TypeError                                 Traceback (most recent call
last)
Cell In[37], line 1
----> 1 list_1 + tuple_1

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

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

i believe that it is the same thing

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, if it's a number then I know that it is an Integer.
- If it's a True or false then I know that it is a Bool.
- If it has a decimal then it is a float.
- A list contains multiple inputs.
- Tuples are like lists but contain numbers

```
In [87]: type(b)
Out[87]: float
```

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 [40]: myList = ['hola',['rnd', 'lola', 7, 'utaustin'], 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.

```
In [41]: nested_list = myList[1]
```

```
In [48]: name = nested_list[1]
```

```
In [49]: print(name)
lola
```

In [52]:
```



number = nested_list[2]
```

In [53]:
```
print(number)
```

```
7
```

Now see if you can do this in one step.

In [54]:
```
element = myList[1][2]
```

In [55]:
```
print(element)
```

```
7
```

9.

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

In [56]:
```
myDict = {'name': 'rnd', 'scores': [65, 75, 85, 95]}
```

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

In [59]:
```
element = myDict['scores'][2]
print(element)
```

```
85
```

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 [62]:
```
myList = [{'name': 'rndoh', 'age': 20}, [1, 2, 3, 4, 5], 7, 'hols']

id_1 = id(myList[0])
id_2 = id(myList[1])

print(f"ID of the first element: {id_obj1}")
print(f"ID of the second element: {id_obj2}")
```

```
ID of the first element: 4594399360
ID of the second element: 4595238400
```

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?

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

IDs are not 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 [63]:
```
mystr = 'abcde'

id_b = id(mystr[1])

print(f"ID of 'b': {id_b}")
```

```
ID of 'b': 4528111976
```

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

In [64]:
```
string_b = 'b'

id_b = id(string_b)

print(f"ID of 'b': {id_b}")
```

```
ID of 'b': 4527810808
```

What happened?

different because the first b was a part of a string and this b was on a string on it's own