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 [29]:
```
ans = 42
another_name_ans = 42
```

Get and note the ID numbers for both.

In [30]:
```
print(id(ans), id(another_name_ans))
```

```
4396991096 4396991096
```

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

In [31]:
```
ans = 30
another_name_ans = 30
print(id(ans), id(another_name_ans))
```

```
4396990712 4396990712
```

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

In [32]:
```
whos
```

```
Variable           Type    Data/Info
-------------------------------------
alphabet           str     abcde
another_name_ans   int     30
ans                int     30
letter_b           str     b
new_list           list    n=4
new_variable       int     2
original_ans       int     42
```

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

In [33]:
```
original_ans = 42
id(original_ans)
```

Out[33]:
```
4396991096
```

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

The ID for 'original_ans' is the same ID number for the original value 42. This means that the ID number of an object remains the same even when the assigned name changes.

2.

Convert both possible Boolean values to strings and print them.

In [11]:
```
bool1 = True
bool2 = False

to_str1 = str(bool1)
to_str2 = str(bool2)

print(to_str1, to_str2)
```

```
True False
```

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

In [34]:
```
new_str1 = 'Hi'
new_str2 = '72'
new_str3 = 'This is a sentence.'
new_str4 = ''

print(bool(new_str1), bool(new_str2), bool(new_str3), bool(new_str4))
```

```
True True True False
```

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

Empty strings convert to False and strings that contain something is True when converted to boolean.

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 [50]:
```
Bool = True
num = 23
float_num = 5.7
```

In [51]:
```
Bool + num
```

Out[51]:
```
24
```

In [52]:
```
num + Bool
```

Out[52]:
```
24
```

In [53]:
```
num + float_num
```

Out[53]:
```
28.7
```

In [54]:
```
float_num + num
```

Out[54]:
```
28.7
```

In [55]:
```
Bool + float_num
```

Out[55]:
```
6.7
```

In [56]:
```
float_num + Bool
```

Out[56]:
```
6.7
```

What is the rule for adding numbers of different types?

The rule is that any combinations of the variables, no matter of the order, can be added together.

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 [25]:
```
num = 45
num_str = '64'
```

In [26]:
```
# adding them
num + num_str
```

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

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

In [29]:
```
# adding them converting the number to a string
to_str = str(num)
to_str + num_str
```

Out[29]:
```
'4564'
```

In [30]:
```
# adding them converting the string to a number
to_int = int(num_str)
num + to_int
```

Out[30]:
```
109
```

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

In [33]:
```
letter_str = 'forty two'
convert_to_int = int(letter_str)
convert_to_int
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[33], line 2
      1 letter_str = 'forty two'
----> 2 convert_to_int = int(letter_str)
      3 convert_to_int

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

Did that work?

No it didn't work.

5.

Make a variable that is a 5 element tuple.

In [35]:
```
tup = (1, 2, 'three', 4.0, True)
```

Extract the last 3 elements.

In [36]:
```
tup[2:]
```

Out[36]:
```
('three', 4.0, True)
```

6.

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

In [37]:
```
tup2 = (1, 2, 3, 4, 5)
tup + tup2
```

Out[37]:
```
(1, 2, 'three', 4.0, True, 1, 2, 3, 4, 5)
```

Make two list variables and add them.

In [38]:
```
lst = [1, 2, 3, 4, 5]
lst2 = [6, 7, 8, 9, 10]

lst + lst2
```

Out[38]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

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

In [39]:
```
tup + lst
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[39], line 1
----> 1 tup + lst

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

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

I was unable to add the tuple to the list, but I was able to add a bool to a float. This is probably because True is equal to 1, so the boolean was able to add to a float. Tuple is immutable and list is mutable, which is why I couldn't add them together. However, they can be added with alike data types, meaning tuple + tuple or list + list.

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 [57]:
```
this_is_int = 45       # integers are whole numbers
this_is_float = 4.5    # floats are decimals
this_is_string = 'hi' # strings have parentheses
this_is_bool = True    # bools are either True or False
```

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 [58]:
```
my_lst = ['hello', 5, [2, 4, 6, 8], 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 [61]:
```
nested_lst = my_lst[2]
nested_lst[1]
```

Out[61]:
```
4
```

Now see if you can do this in one step.

In [62]:
```
my_lst[2][1]
```

Out[62]:
```
4
```

9.

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

In [65]:
```
Dict = {'College': 'COLA',
        'major': ['Psychology', 'Sociology', 'Economics', 'Government']}
```

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

In [66]:
```
Dict['major'][0]
```

Out[66]:
```
'Psychology'
```

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 [18]:
```
new_list = [2, 3, 4, 5]

# unique IDs
print(id(new_list[0]),id(new_list[1]))
```

```
4396989816 4396989848
```

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?

In [19]:
```
# assigned an element from new_list to a new variable

new_variable = new_list[0]
print(id(new_variable))
```

```
4396989816
```

The IDs are still the same. IDs don't change when I assigned a list element to a new 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 [40]:
```
alphabet = 'abcde'
print(id(alphabet[1]))
```

```
4397035880
```

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

In [41]:
```
letter_b = 'b'
print(id(letter_b))
```

```
4396734712
```

What happened?

'b' in the two strings do not have the same ID even though they are technically the same object.