KEY_Practice03_Variables_Types

August 29, 2019

1 Practice with Variables and Types

As you've seen, one of the most basic ways to use Python is as a calculator. Let's step up our calculation game by using Python to solve a math problem.

Let's say you opened up a savings account at your local bank and to start you deposited $100. While your money is in the bank it is *accruing interest* at a rate of 10 percent per year. Now, you want to know how much money you will have at the end of 7 years, assuming you never withdraw or deposit anything from your savings account. The formula for calculating this value is:

total = start_balance * (1 + interest_rate)years

[1]:
```
# assign your deposit amount to the variable start_balance
start_balance = 100

# assign your interest rate to the variable interest_rate (TIP: How do we␣
↪represent percentages in math?)
interest_rate = 0.1

# print the types of both start_balance and interest_rate. Make sure they match␣
↪what you expect
print(type(start_balance))
print(type(interest_rate))
```

```
<class 'int'>
<class 'float'>
```

[2]:
```
# assign the number of savings years to the variable years
years = 7

# calculate the result of your savings problem to the variable total.
# HINT 1: Remember your order of operations!
# HINT 2: Use Google for the Python exponent mathematical operator
# Google is your best friend in coding - don't be afraid to use it!
total = start_balance * (1+interest_rate) ** years

# print the value of the total variable
print(total)
```

```
# print the type of the total variable. Is this what we expect? Why?
print(type(total))
```

```
194.87171000000012
<class 'float'>
```

Now that we've had some practice with numerical types, let's get some practice with the other types we learned!

First, let's see if we were able to double our money with our 7 years of interest and patience.

[3]:
```
# determine if total is greater than or equal to double our start balance
# assign this value to the variable doubled and print it
doubled = total >= start_balance*2
print(doubled)

# What type do we expect this to be? Let's print the type of the variable␣
 ↪doubled
print(type(doubled))
```

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

Now, let's play the name game.

[4]:
```
# create a variable called first_name and assign it to your first name as a␣
 ↪string
first_name = "Marlena"

# create a variable called last_name and assign it to your last name as a string
last_name = "Duda"

# Now, concatenate (i.e. add) first_name with last_name and assign it a␣
 ↪variable called full_name
full_name = first_name + last_name

# print full_name. Does this look like you expect? How can you fix the␣
 ↪formatting so it makes sense?
print(full_name)

full_name = first_name + " " + last_name
print(full_name)

# now try to add full_name to the variable start_balance from above. Does this␣
 ↪work?
# read the error message you get carefully!
# learning how to read error messages will give you the keys to fix mistakes (i.
 ↪e. bugs) in your code
```

```
full_name + start_balance
```

```
MarlenaDuda
Marlena Duda


      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-4-6d47cd2526cd> in <module>()
        17 # read the error message you get carefully!
        18 # learning how to read error messages will give you the keys to fix␣
↪mistakes (i.e. bugs) in your code
   ---> 19 full_name + start_balance


       TypeError: must be str, not int
```

Lastly, let's use Python to create a personalized investment report.

We know that we cannot simply add integers, floats, or booleans to strings. But in terms of printing, there are a few ways we can handle the formatting to enable printing of mixed types in the same message.

1.0.1 Method 1 - type conversion

Numeric types and booleans can be converted to strings using the `str()` function.

[5]:
```
print('ABC' + str(123))
print('LMNOP' + str(8.910))
print('XYZ' + str(True))
```

```
ABC123
LMNOP8.91
XYZTrue
```

Some strings can also be converted to other types (if they make sense) using the `int()`, `float()`, and `bool()` functions.

[6]:
```
print(4 + int('5'))
print(1.2 + float("3.4"))
print(False + bool('True'))
```

```
9
4.6
```

1

1.0.2 Method 2 - multiple parameters in `print` function

The `print` function takes multiple parameters (i.e. inputs), which can be of mixed type and are separated by commas.

[7]:
```
print("I have", 1 , "dog and", 2, "cats")
print("I ate", 2.5, "ice cream sandwiches")
print("Those statements are both", True)
```

```
I have 1 dog and 2 cats
I ate 2.5 ice cream sandwiches
Those statements are both True
```

Notice that when using multiple parameters in the `print` functon, Python automatically puts spaces between the different inputs, but when using the type conversion method there are no automatic spaces added.

Now, use either of the above methods to print a report that matches the following format:

```
Customer Name: Marlena Duda
Starting Balance: $100
```

Years Saved: 7 Ending Balance: $194.87 Doubled Starting: False

Remember: Use the variable names we defined above whenever you can!

[8]:
```
# HINT: remember the spacing rules above!
print('Customer Name:', full_name)
print('Starting Balance: $' + str(start_balance))
print('Years Saved:', years)
print('Ending Balance: $' + str(total))
print('Doubled Starting:', doubled)
```

```
Customer Name: Marlena Duda
Starting Balance: $100
Years Saved: 7
Ending Balance: $194.87171000000012
Doubled Starting: False
```

[ ]:
```

```