> SDEV-220, Fall-2024

Python: numbers, booleans, and variables

Last updated by Khristin Schenk on Aug 26th, 2024

- [x] Module 1 Lab - Collaboration and Github
- [x] Module 1 Programming Assignment - Numbers and Types

> Chapter 3 to the section titled "Things to Do"
> Complete sections 3.1 through 3.6 utilizing your Jupyter Notebook

From textbook:
Introducing Python, 2nd Edition by Bill Lubanovic. Published by O'Reilly Media, Inc.

3.1 Calculate the Number of Seconds in an Hour

An hour has 60 minutes, and each minute has 60 seconds. So the total number of seconds in an hour can be calculated as:

```
seconds_per_hour = 60 * 60
```

3.2 Assign the Result to a Variable

The result is assigned to the variable `seconds_per_hour` in the above step.

3.3 Calculate Seconds in a Day

To find the total number of seconds in a day, you need to multiply the number of seconds per hour by the number of hours in a day (24):

```
seconds_per_day = seconds_per_hour * 24
```

3.4 Calculate Seconds Per Day and Save to a Variable

The calculation is done in the above step, and the result is stored in `seconds_per_day`.

3.5 Divide `seconds_per_day` by `seconds_per_hour` Using Floating-Point Division

To perform floating-point division:

```
floating_point_division = seconds_per_day / seconds_per_hour
```

3.6 Divide `seconds_per_day` by `seconds_per_hour` Using Integer Division

To perform integer division:

```
integer_division = seconds_per_day // seconds_per_hour
```

3.7 Comparison of Floating-Point and Integer Division

The integer division should yield the same quotient as the floating-point division when the result is converted to an integer, excluding any fractional part. (https://www.educative.io/answers/what-is-integer-and-float-division-in-python)

Compared:

```
print(floating_point_division)  # This should give a floating-point result
print(integer_division)         # This should give an integer result

# Check if they agree
print(floating_point_division == integer_division)  # This should be True
```

```
# 3.1 Calculate the Number of Seconds in an Hour
seconds_per_hour = 60 * 60

# 3.3 Calculate Seconds in a Day
seconds_per_day = seconds_per_hour * 24

# 3.5 Divide `seconds_per_day` by `seconds_per_hour` Using Floating-Point Division
floating_point_division = seconds_per_day / seconds_per_hour

# 3.6 Divide `seconds_per_day` by `seconds_per_hour` Using Integer Division
integer_division = seconds_per_day // seconds_per_hour

# Print results
print("Seconds per hour:", seconds_per_hour)
print("Seconds per day:", seconds_per_day)
print("Floating-point division:", floating_point_division)
print("Integer division:", integer_division)

# Check if they agree (aside from the final .0)
print("Do they agree?", floating_point_division == integer_division)
```

Running this code will show that the floating-point division and integer division should agree, with the integer division result being the integer part of the floating-point division result.

Notes From Textbook:

Python's simplest built-in data types

- **Booleans** (which have the value True or False)
- **Integers** (whole numbers such as 42 and 100000000)
- **Floats** (numbers with decimal points such as 3.14159, or sometimes exponents like 1.0e8, which means one times ten to the eighth power, or 100000000.0)

Floats

Integers are whole numbers, but floating-point numbers (called floats in Python) have decimal points:

```
>>> 5.
5.0
>>> 5.0
5.0
>>> 05.0
5.0
```

Floats can include a decimal integer exponent after the letter e:

```
>>> 5e0
5.0
>>> 5e1
50.0
>>> 5.0e1
50.0
>>> 5.0 * (10 ** 1)
50.0
```

You can use underscore (_) to separate digits for clarity, as you can for integers:

```
>>> million = 1_000_000.0
>>> million
1000000.0
>>> 1.0_0_1
1.001
```

Floats are handled similarly to integers:

- You can use the operators (+, –, *, /, //, **, and %) and the divmod() function.
- To convert other types to floats, you use the float() function. As before, booleans act like tiny integers:

```
>>> float(True)
1.0
>>> float(False)
0.0
```

Converting an integer to a float just makes it the proud possessor of a decimal point:

```
>>> float(98)
98.0
>>> float('99')
99.0
```

And you can convert a string containing characters that would be a valid float (digits, signs, decimal point, or an e followed by an exponent) to a real float:

```
>>> float('98.6')
98.6
>>> float('-1.5')
-1.5
>>> float('1.0e4')
10000.0
```

When you mix integers and floats, Python automatically promotes the integer values to float values:

```
>>> 43 + 2.
45.0
```

Python also promotes booleans to integers or floats:

```
>>> False + 0
0
>>> False + 0.
0.0
>>> True + 0
1
>>> True + 0.
1.0
```

Nonzero numbers are considered True:

```
>>> bool(True)
True
>>> bool(1)
True
>>> bool(45)
True
>>> bool(-45)
True
```

And zero-valued ones are considered False:

```
>>> bool(False)
False
>>> bool(0)
False
>>> bool(0.0)
False
```