Carleton University
Department of Systems and Computer Engineering
ECOR 1051 - Fundamentals of Engineering I - Fall 2019

Lab 1 - Using Software Experiments to Learn about Python's Datatypes

Objectives

- To learn how to use the Python shell to perform software experiments.
- To learn more about the Python datatype that represent integers.

Overview

In this lab, you'll use the Python shell to explore how Python supports calculations with integers and real numbers. Our approach will use *directed experimentation*; that is, experimentation to learn something that wasn't known before.

For some exercises, you'll be asked to type expressions in the Python shell window, record the results displayed by Python, and then draw conclusions.

For other exercises, you will be asked to devise one or more short experiments. You'll record your experiments (i.e., write the Python expressions that you typed and the results displayed by Python), then write one or two sentences that summarize what you learned.

Learning outcomes: 1; Graduate attributes: 1.3, 5.3 (see the course outline)

Getting Started

Launch Wing 101. One of the tabbed windows is titled Python Shell. When Python starts, it prints a message in this window. The first line should contain "3.7.4", which indicates that Wing is running Python version 3.7. If another version is running, ask a TA for help to reconfigure Wing to run Python 3.7.

Download lab1.txt from cuLearn and open it in Wing 101. (By default, only the names of Python files are listed in the Open File dialogue box. You'll need to change this so that lab1.txt appears. Click on the Files of type drop-down menu and select Plain Text.)

This file is a template in which you will type your solutions to the exercises. Please, do not change the layout of this file; that is, delete any lines in the template. Don't change the formatting; for example, use boldface or italicized text. You will be submitting your solutions as plain text.

We recommend that you bring a printed copy of this handout to the lab. Instead of editing lab1.txt while you do the exercises, you can write your solutions on the ruled lines in the handout, then edit lab1.txt after you've finished all the exercises. Remember, you don't have to finish the lab during the lab period. You have until Sunday, Sept. 15 at 11:59 pm to submit your solutions for grading.

Exercise 1: The shell displays a prompt (`>>>`) when it is ready for you to type a command. After the prompt, type this *expression* (don't type any spaces between the numbers and the plus sign):

```
>>> 1+2
```

The integers 1 and 2 are the expression's *operands* and the `+` is the *addition operator*.

Now press the `Enter` key. This "tells" Python to *read* this expression, verify that it is syntactically correct, then *evaluate* the expression (perform the addition operation on the two operands) and *print* (display) the result in the shell window.

On the ruled line, write the value that is displayed:

```
>>> 1+2
```

_____

Exercise 2: Are spaces in expressions significant? This time, type exactly one space between the operands and the operator, like this:

```
>>> 1 + 2
```

Remember to press `Enter` when you are ready for Python to evaluate the expression.

Write the value that is displayed:

```
>>> 1 + 2
```

_____

Repeat the experiment, but this time, type several spaces between the operands and the operator:

```
>>> 1      +        2
```

Write the value that is displayed:

```
>>> 1      +        2
```

_____

What do you conclude about the significance of spaces in expressions? Does the number of spaces between the operands and operator affect the evaluation? Record your conclusions here:

_____

Exercise 3: What kind of numbers are 1, 2, 0, -1, etc.? Python has a built-in *function* called `type`, which, when *called*, provides information about the *type* of its *argument*. (For those who haven't programmed before, calling a function is analogous to entering a number on your handheld calculator (the argument), then pressing a function key labelled `type`). Type the following *call expressions* and record the information that's displayed:

```
>>> type(1)
```

_____

```
>>> type(2)
```

_____

```
>>> type(0)
```

_____

```
>>> type(-1)
```

_____

```
>>> type(255)
```

_____

```
>>> type(-256)
```

_____

When Python evaluates each of these call expressions, it should display `<class 'int'>`. Did you observe anything different? If not, we can conclude that integers are values of type `int`. (Python types are implemented by a programming construct known as a *class*, so we can also say that Python integers are *instances* of class `int`.)

Exercise 4: What about other arithmetic operations on integers (including negative integers)? Type these expressions and record the results:

```
>>> 4 - 1
```

_____

```
>>> 4 - 6
```

_____

```
>>> 5 - -9
```

_____

```
>>> -7 - -2
```

_____

```
>>> 2 * 3
```

_____

```
>>> 4 * -3
```

_____

```
>>> -5 * -4
```

_____

It appears that, for integer values, `-` is the subtraction operator and `*` is the multiplication operator. What can you conclude about the types of the values produced by these operations?

_____

_____

_____

_____

Exercise 5: The forward slash character, `/`, is the symbol for the division operator. Type these expressions and record the results:

```
>>> 6 / 2
```

_____

```
>>> -6 / 2
```

_____

```
>>> 6 / -2
```

_____

```
>>> -6 / -2
```

_____

```
>>> 7 / 2
```

_____

```
>>> -7 / 2
```

_____

```
>>> 7 / -2
```

_____

```
>>> -7 / -2
```

_____

```
>>> 10 / 3
```

_____

```
>>> 10 / 6
```

_____

Use the `type` function to determine the types of the values produced when one integer is divided by another. Record your conclusions.

_____

_____

_____

Exercise 6: Does Python support more complicated expressions made up of several operands and operators? Type these expressions and record the results:

```
>>> 1 + 2 + 3
```

_____

```
>>> 5 - 1 - 1 - 1
```

_____

```
>>> 2 * 2 * 3
```

_____

```
>>> 12 / 3 / 2
```

_____

Record your conclusions:

_____

_____

Exercise 7 is on the next page.

Exercise 7: We can easily show that we can mix operators in an expression, but we then have to consider the order in which the operations are performed.

Consider these two expressions: `1 + 2 * 3` and `2 * 3 + 1`.

If the * operator has the same *precedence* as the + operator, the expressions will be evaluated left-to-right; that is, when the first expression is evaluated, the addition will be performed before the multiplication, and when the second expression is evaluated, the multiplication will be performed before the addition.

On the other hand, if the * operator has higher precedence than the + operator, the multiplication will always be performed before the addition, and both expressions will evaluate to the same value.

Type these expressions and record the results:

```
>>> 1 + 2 * 3
```

_____

```
>>> 2 * 3 + 1
```

_____

What do you conclude about the precedence of the * operator relative to the + operator?

_____

Exercise 8: We can use parentheses (round brackets) to change the order of evaluation. Type these expressions and record the results.

```
>>> (1 + 2) * 3
```

_____

```
>>> 2 * (3 + 1)
```

_____

Compare these results to the ones you obtained in Exercise 7.

_____

_____

Exercise 9: We hypothesize that subtraction has lower precedence than multiplication, which means that multiplication will always be performed before subtraction when an expression is evaluated. We also hypothesize that subtraction has the same precedence as addition. This means that an expression containing (only) the `-` and `+` operators will always be evaluated left to right.

Design a few experiments to help you determine if these hypotheses are true. Record the expressions you type and the results displayed by Python. Are these hypotheses correct? Record your conclusions.

Exercise 10: In grade-school mathematics classes, you learned about *implicit multiplication*, where multiplication operators can be purposely excluded from an expression if no misunderstanding is possible. For example, in the expression $2(3 + 4)$, 2 is implicitly multiplied with the sum of 3 and 4. This expression is equivalent to $2 \times (3 + 4)$, which uses the explicit multiplication operator, ×.

Design an experiment to determine if Python supports implicit multiplication. Record the expressions you type and the results displayed by Python. Record your conclusions.

_____

_____

_____

Exercise 11: Can Python expressions have superfluous parentheses? Try these experiments (as always, record the results Python displays):

```
>>> (1 + 2 + 3)
```

_____

```
>>> (1) + 2
```

_____

```
>>> (((1 + 2) + 3) + 4) + 5
```

_____

```
>>> ((((1))))
```

_____

What do you conclude?

_____

_____

Exercise 12: Python's exponentiation operator is **. For example, the expression that calculates 3^2 ("3, squared") is `3 ** 2`.

- Type the following expression at the shell prompt to calculate 3^2. Write the value Python displays when it evaluates the expression.

  ```
  >>> 3 ** 2
  ```

  _____

- Suppose we want to raise -3 to the power 2; i.e., calculate -3^2 ("-3, squared"), which equals 9.

  Perhaps the Python expression `-3 ** 2` would do this. Type this expression and record the value Python displays. Does this expression correctly calculate -3^2; that is, is the result 9?

  ```
  >>> -3 ** 2
  ```

  _____

- Based on what you observed, which operation does Python perform first when it evaluates the expression `-3 ** 2`: **exponentiation** (i.e., does Python first raise +3 to the power 2, yielding 9, then negate the result, yielding -9) or **negation** (i.e., does Python first negate +3, yielding -3, then raise -3 to the power 2, yielding 9)?

  _____

- Create some experiments to determine the simplest expression that raises -3 to the power 2, yielding 9; i.e., what is the simplest expression that squares -3? (For those of you who are reading ahead, don't use Python's `pow` function.) Record your experiments and your conclusions.

  _____

  _____

  _____

  _____

  _____

Exercise 13: In mathematics, applying the *floor function* to a number x (denoted, $\lfloor x \rfloor$) yields the largest integer less than or equal to x. For example, the floor of 3.25 is 3, the floor of -3.25 is -4, and the floor of 3.0 is 3. For more information, read:
http://mathworld.wolfram.com/FloorFunction.html

Integer division is division in which the fractional part (remainder) is discarded and is sometimes denoted by the symbol, \. Integer division of integers a and b, $a \setminus b$, can be defined as $\lfloor a / b \rfloor$,where / denotes "normal" division. For example, 7 \ 2 is $\lfloor 7 / 2 \rfloor$, which is $\lfloor 3.5 \rfloor$, which equals 3.

Python has a *floor division* operator, which is represented by `//`. When Python evaluates the expression `a // b`, where `a` and `b` are integers, `a` is divided by `b`, then the floor function is applied to the quotient. In other words, when its arguments are values of type `int`, the `//` operator performs integer division. For example,

- `15 // 2` yields 7 (15 / 2 yields 7.5, and $\lfloor 7.5 \rfloor$ is 7).
- `-15 // 2` yields -8 (-15 / 2 yields -7.5, and $\lfloor -7.5 \rfloor$ is -8).

When the operands are integers, the `//` operator always yields the largest integer that is less than or equal to the quotient; that is, it always rounds the quotient towards minus infinity.

Without using Python, predict the values of these expressions. Record your predictions.

```
11 // 4
```

_____

```
-11 // 4
```

_____

```
11 // -4
```

_____

```
-11 // -4
```

_____

Now use the Python shell to evaluate these expressions. Were your predictions correct?

Exercise 14: Recall from elementary school that the result of dividing two integers can be expressed as an integer quotient and an integer remainder. For example, 14 divided by 5 is 2 (the quotient), with a remainder of 4.

We've learned that Python's floor division operator calculates integer quotients when the operands are integers. For example, typing:

```
>>> 14 // 5
```

yields 2, because 14 divided by 5 is 2.8, and the floor function applied to 2.8 yields 2.

Python's *modulo* operator, `%`, calculates the remainder that is "left over" when one integer is divided by another. At the shell prompt, type:

```
>>> 14 % 5
```

The result is 4 (because 14 divided by 5 yields remainder 4).

For any integers *x* and `y`, this identity is always true:

```
x = (x // y) * y + (x % y)
```

(Aside: in this exercise, the symbol `=` is the *equals sign* used in mathematics, and not Python's assignment operator.)

This identity tells us that Python's floor division (`x // y`) and modulo (`x % y`) operations are connected this way: the dividend, *x*, is equal to the integer quotient (`x // y`) multiplied by the divisor, *y*, plus the integer remainder (`x % y`).

Let's verify this for the case where `x` is `14` and `y` is `5`. At the shell prompt, type:

```
>>> (14 // 5) * 5 + (14 % 5)
```

As expected, the result is 14. (If you're not sure why, evaluate this expression without using Python.) So, we've verified that this identity is true:

```
14 = (14 // 5) * 5 + (14 % 5)
```

Things get interesting when one or both operands are negative. At the shell prompt, type:

```
>>> -14 // 5
```

The result is -3 (because -14 divided by 5 is -2.8, which is then rounded towards minus infinity, yielding -3.) So, what is the remainder; i.e., what is `-14 % 5`?

We know that the identity `x = (x // y) * y + (x % y)` must be satisfied, so substitute -14 and 5 for *x* and *y* and simplify:

```
-14 = (-14 // 5) * 5 + (-14 % 5)
    = -3 * 5 + (-14 % 5)
    = -15 + (-14 % 5)
```

To satisfy the identity, `-14 % 5` must evaluate to 1.

Let's verify this. At the shell prompt, type:

```
>>> (-14 // 5) * 5 + (-14 % 5)
```

The result is -14, so clearly, the identity is satisfied.

At the shell prompt, type:

```
>>> -14 % 5
```

The result is 1, as we predicted.

Let's repeat this experiment for (14, -5). Type these expressions and record the results:

```
>>> 14 // -5
```

_____

```
>>> 14 % -5
```

_____

```
>>> (14 // -5) * -5 + (14 % -5)
```

_____

Can you conclude that the identity `14 = (14 // -5) * -5 + (14 % -5)` is true?

_____

Let's repeat this experiment for (-14, -5). Type these expressions and record the results:

```
>>> -14 // -5
```

_____

```
>>> -14 % -5
```

_____

```
>>> (-14 // -5) * -5 + (-14 % -5)
```

_____

Can you conclude that the identity `-14 = (-14 // -5) * -5 + (-14 % -5)` is true?

_____

Make sure you understand that, when `x` and `y` are integers, the expression:

```
(x // y) * y + (x % y)
```

always evaluates to `x`. If this isn't clear, go back and repeat the exercise to this point.

Without using Python, predict the values of these expressions. Record your predictions.

```
(22 // 7) * 7 + (22 % 7)
```

_____

```
22 // 7
```

_____

```
(22 // 7) * 7
```

_____

```
22 % 7
```

_____

This exercise continues on the next page.

```
(-22 // 7) * 7 + (-22 % 7)
```

_____

```
-22 // 7
```

_____

```
(-22 // 7) * 7
```

_____

```
-22 % 7
```

_____

```
(22 // -7) * -7 + (22 % -7)
```

_____

```
22 // -7
```

_____

```
(22 // -7) * -7
```

_____

```
22 % -7
```

_____

This exercise continues on the next page.

```
(-22 // -7) * -7 + (-22 % -7)
```

_____

```
-22 // -7
```

_____

```
(-22 // -7) * -7
```

_____

```
-22 % -7
```

_____

Now use the Python shell to determine if your predictions were correct.

Summary: given two integers *j* and *k*, record the steps you should follow (without using the Python interpreter) to predict the value Python will calculate when it evaluates the expression: *j % k*

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Wrap Up

1. After you've typed your solutions in lab1.txt, login to cuLearn.
2. Click the link Submit Lab 1 for grading.
3. Click the Add submission button.
4. Copy/paste your solutions from lab1.txt to the Online text box, then click the Save changes button. The status of your submission is now *Draft (not submitted)*, which means that you can make changes.
5. Review the online text. If you want to make changes to your submission, click the Edit submission button. When you are ready to finish submitting your lab work, click the Submit assignment button.
6. When the Confirm submission page appears, click the Continue button to submit your work. The status of your submission will change to *Submitted for grading*. If you've changed your mind, click the Cancel button. This will return you to the page where you can edit your submission.

Edited: Sept. 2, 2019

Updated (corrections to Exercise 14) and reposted: Sept. 7, 2019