Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2018

Lab 2, Part 1 - Working with Variables, Visualizing Code Execution

Objectives

- To gain experience translating mathematical formulae into Python statements that use variables.
- To gain experience using the online Python Tutor to visualize the execution of the code.

Grading

After you finish all the exercises, a TA will review your solutions, ask you to demonstrate some of them, and assign a grade. For those who don't finish early, a TA will grade the work you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

Getting Started

Step 1: Create a new folder named Lab 2 on the lab computer's hard disk (on the desktop or in your account's Documents folder) or in the M: drive on the network server.

Step 2: Launch Wing 101. When Python starts, it prints a message in the shell window. Check the message and verify 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.

Step 3: For this lab, you'll use Python Tutor to edit and execute your code. PyTutor is a great tool for visualizing the execution of short pieces of code, but it is not a complete program development environment, and it doesn't provide a way for us to save our code in a file. So, while you're working on the exercises, you'll use Wing's editor to prepare a file that contains a copy of your solutions.

Click the New button in the menu bar. A new editor window, labelled untitled-1.py, will appear.

Step 4: From the menu bar, select File > Save As... A "Save Document As" dialogue box will appear. Navigate to your Lab 2 folder, then type lab2.py in the File name: box. Click the Save button. Wing will create a new file named lab2.py.

Exercise 1: To convert a temperature measured on the Celsius scale to the equivalent Fahrenheit temperature, we multiply the Celsius temperature by $\frac{9}{5}$, then add 32. For example, 20.0 degrees Celsius is equivalent to 68 degrees Fahrenheit.

Find the link Open Python Tutor in a new window in the *Labs* section of the course cuLearn page. Launch Python Tutor, and verify that it is configured this way: "Write code in Python 3.6", "hide exited frames", "render all objects on the heap (Python/Java)" and "draw pointers as arrows". If necessary, select the correct options from the drop-down menus.

In Python Tutor's editor window:

1. Type an assignment statement that creates a new variable named `degrees_c` and binds it to `20.0`.

2. Type an assignment statement that converts the temperature bound to `degrees_c` to the equivalent temperature in degrees Fahrenheit. This temperature should be bound to a new variable named `degrees_f`.

Click the Visualize Execution button. Execute the code, one statement at a time, by clicking the Forward> button. Observe the memory diagram as the code is executed, step-by-step. Make sure you can answer these questions:

- When does variable `degrees_c` appear in the diagram?
- What is the name of the frame containing `degrees_c`?
- What does the arrow that points from `degrees_c` to `20.0` represent?
- When does variable `degrees_f` appear in the diagram?
- What does `degrees_f` contain?

When your code is correct, copy/paste the two assignment statements from PyTutor to Wing's editor window for file lab2.py, then save the file (click the Save button or select File > Save from the menu bar.)

Exercise 2: In some countries, a vehicle's fuel efficiency is measured in miles per gallon. In other countries, the efficiency is measured in litres per 100 km. One Imperial gallon is equal to approximately 4.54609 litres. One mile is equal to approximately 1.60934 km. For example, 32 miles per gallon is equivalent to approximately `8.83` litres per 100 km.

Delete the code in PyTutor's editor, then:

1. Type an assignment statement that creates a new variable named `mpg` and binds it to the value `32` (which represents 32 miles per gallon).

2. Type two assignment statements that create new constants named `LITRES_PER_GALLON` and `KMS_PER_MILE` and binds them to the values `4.54609` and `1.60934`, respectively. (Note that the names of constant values are, by convention, usually written entirely in uppercase.)

3. Type one assignment statement that converts the mileage bound to `mpg` to the equivalent

fuel consumption, measured in litres/100 km. This value should be bound to a variable named `fuel_consumption`.

Click the Visualize Execution button. Execute the code, one statement at a time, and observe the memory diagram as the code is executed, step-by-step. Verify that your code correctly converts 32 mpg into 8.83 litres /100 km.

Make sure you can answer these questions:

- How many variables and objects are created when your code is executed, step-by-step?
- What do the arrows in the memory diagram represent?

When your code is correct, copy/paste the code from PyTutor to Wing's editor window for file lab2.py, then save the file.

Exercise 3: Suppose you have some money (the *principal*) that is deposited in a bank account for a number of years and earns a fixed annual *rate* of interest. The interest is compounded *n* times per year.

The formula for determining the total amount of money you'll have is:

$$amount = principal\left(1 + \frac{rate}{n}\right)^{n \cdot time}$$

where:

- *amount* is the amount of money accumulated after *time* years, including interest.
- *principal* is the initial amount of money deposited in the account
- *rate* is the annual rate of interest, specified as a decimal; e.g, 5% is specified as 0.05
- *n* is the number of times the interest is compounded per year
- *time* is the number of years for which the principal is deposited.

For example, if $1,500.000 is deposited in an account paying an annual interest rate of 4.3%, compounded quarterly (4 times a year), the amount in the account after 6 years is:

$$amount = \$1,500\left(1 + \frac{0.043}{4}\right)^{4 \cdot 6} \approx \$1938.84$$

Delete the code in PyTutor's editor, then:

1. Type assignment statements that create new variables named `principal`, `rate`, `n` and `time`, and bind them to the values `1500`, `0.043`, `4` and `6` respectively.

2. Type <u>one</u> assignment statement that calculates the total amount of money in the account after `time` years. This value should be bound to a new variable named `amount`.

Click the Visualize Execution button. Execute the code, one statement at a time, and observe the memory diagram as the code is executed, step-by-step. Verify that the value bound to `amount` is approximately `1938.84`.

When your code is correct, copy/paste the code from PyTutor to Wing's editor window for file lab2.py, then save the file.

Exercise 4: Consider these assignment statements:

```
a = 9
b = 4
c = a * b
d = b
a = 2
b = 3
c = c + d
```

Without using the Python shell or PyTutor to assist you, predict the values that will be bound to a, b, c and d after Python executes these statements. Type your predictions in lab2.py.

Delete the code in PyTutor's editor, then type the assignment statements (use PyTutor, not the Python shell).

Click the Visualize Execution button. Execute the code, one statement at a time, and observe the memory diagram as the code is executed, step-by-step.

Were your predictions correct?

Exercise 5 starts on the next page.

Exercise 5: This is an edited excerpt from *Practical Programming, 2nd Edition: An Introduction to Computer Science Using Python 3*, Paul Gries, Jennifer Campbell, Jason Montojo, © 2013 The Pragmatic Programmers LLC, Book Version P3.0, January 2016.

In this example, the variable `score` appears on both sides of the assignment statement:

```
>>> score = 50
>>> score
50
>>> score = score + 20
>>> score
70
```

This is so common that Python provides a shorthand notation for this operation:

```
>>> score = 50
>>> score
50
>>> score += 20
>>> score
70
```

An augmented assignment combines an assignment statement with an operator to make the statement more concise. An augmented assignment statement is executed as follows:

1. Evaluate the expression on the right of the = sign to produce a value.
2. Apply the operator attached to the = sign to the variable on the left of the = and the value that was produced in step 1. This produces another value. Store the binding to that value in the variable on the left of the =.

Note that the operator is applied *after* the expression on the right is evaluated:

```
>>> d = 2
>>> d *= 3 + 4  # This is equivalent to d = d * (3 + 4),
                # not d = d * 3 + 4
>>> d
14
```

All of the binary operators, `+`, `-`, `*`, `/`, `//`, `%` and `**`, have shorthand versions. For example, we can square a number by multiplying it by itself:

```
>>> number = 10
>>> number *= number
>>> number
100
```

This code is equivalent to:

```
>>> number = 10
>>> number = number * number
>>> number
100
```

This table contains a summary of the augmented assignment operators:

Symbol	**Example**	**Result**

`+=`	`x = 7`
	`x += 2`
		`x` is bound to `9`

`-=`	`x = 7`
	`x -= 2`
		`x` is bound to `5`

`*=`	`x = 7`
	`x *= 2`
		`x` is bound to `14`

`/=`	`x = 7`
	`x /= 2`
		`x` is bound to `3.5`

`//=`	`x = 7`
	`x //= 2`
		`x` is bound to `3`

`%=`	`x = 7`
	`x %= 2`
		`x` is bound to `1`

`**=`	`x = 7`
	`x **= 2`
		`x` is bound to `49`

<u>Without using the Python shell or PyTutor to assist you</u>, predict the value that will be bound to `x` after Python executes these two statements (type your prediction in lab2.py) :

```
    x = 3
    x += x - x
```

Delete the code in PyTutor's editor, then type the assignment statements (use PyTutor, not the Python shell).

Click the Visualize Execution button. Execute the code, one statement at a time, and observe the memory diagram as the code is executed, step-by-step.

Was your prediction correct?

Wrap-up

1. Remember to have a TA review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the grading/signout sheet.
2. All files you've created on the hard disk will be deleted when you log out. Remember to backup your project folder before you leave the lab; for example, copy it to a flash drive and/or a cloud-based file storage service.

Posted: Sept. 23, 2018; updated and reposted: Sept. 24, 2018

Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2018

Lab 2, Part 2 - Using Software Experiments to Learn about Python's `str` Datatype

Objectives

- To gain more experience using the Python shell to build software experiments.
- To learn about the `str` datatype provided by Python.

Overview

In Lab 1, you used software experiments to learn how Python supports calculations with integers (values of type `int`) and real numbers (values of type `float`). In this lab's exercises, you'll explore the computation Python can perform with character strings (values of type `str`).

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.

Getting Started

Launch Wing 101. One of the tabbed windows is titled Python Shell. When the shell starts, Python prints a message in this window. The first line should contain "v3.7.0", 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.

For this lab, you'll do all of your work in the shell. You won't be using an editor to write complete Python scripts (programs), so you don't need to open a new editor window.

Exercise 1: A Python *string* is a collection of characters. Literal strings are enclosed in pairs of single quotes or double quotes; for example, `'Hello, world!'` and `"Hello, world!"` are the same string. This syntax is particularly useful if you want a string literal that contains single or double quotes; for example, `"haven't"` or `'"Spam, spam, spam," he said.'`

Type the following expressions and record the results:

```
>>> type('Hello')
```

____

```
>>> type("goodbye")
```

____

```
>>> type("Python programming is fun, isn't it")
```

____

```
>>> type('a')
```

____

`>>> type("")` *(Type two double quotes with no spaces between them.)*

____

The information displayed by Python after each of these function calls should be `<class 'str'>`. Did you observe anything different? If not, we can conclude that character strings are values of type `str`.

Exercise 2: What operations are permitted with values of type `str`?

When used with strings, `+` is the *concatenation operator*. At the shell prompt, type:

```
>>> 'Hello, ' + 'world!'
```

The result is a new string, `'Hello, world!'`

Are `*`, `/`, `//` and `-` valid operators when both operands are strings? If so, what operations do they perform? What is the type of each result? Perform some experiments to find out. On the ruled lines, record the expressions you type, the values that Python displays when it evaluates the expressions, and your conclusions.

____

____

____

____

____

____

____

Exercise 3: Type the following expressions and record the results:

```
>>> "Spam" * 3
```

____

```
>>> 3 * "Spam"
```

____

```
>>> "Spam" * 1
```

____

```
>>> 1 * "Spam"
```

____

```
>>> "Spam" * 0
```

____

```
>>> 0 * "Spam"
```

____

```
>>> "Spam" * -3
```

____

```
>>> -3 * "Spam"
```

____

What operation does * perform when one operand is a string and the other operand is an integer?

____

Is the order in which the operands appear important; i.e., does it matter whether the string is the left-hand or right-hand operand of *?

____

Exercise 4: Are `+`, `-`, `/`, and `//` valid operators when one operand is a string and the other operand is an integer? If so, what operations do they perform? Does it matter if the string is the left-hand or the right-hand operand? Perform some experiments to find out. On the ruled lines, record the expressions you type, the values that Python displays when it evaluates the expressions, and your conclusions.