PLTW COMPUTER SCIENCE

Activity 1.3.2

Python Variables and Functions

Introduction

Could you make a program in Scratch™ programming language that will check several websites each day for merchandise on sale, compare the merchandise to artificial intelligence about your taste and price range, and then text you if anything exciting comes along?

You could with *Python*® programming language! If computers can do it, your team can make it happen with *Python*. From landing a rover on Mars to predicting the weather, *Python* is a versatile language that can get the job done.

Let's start by making the computer perform some arithmetic.

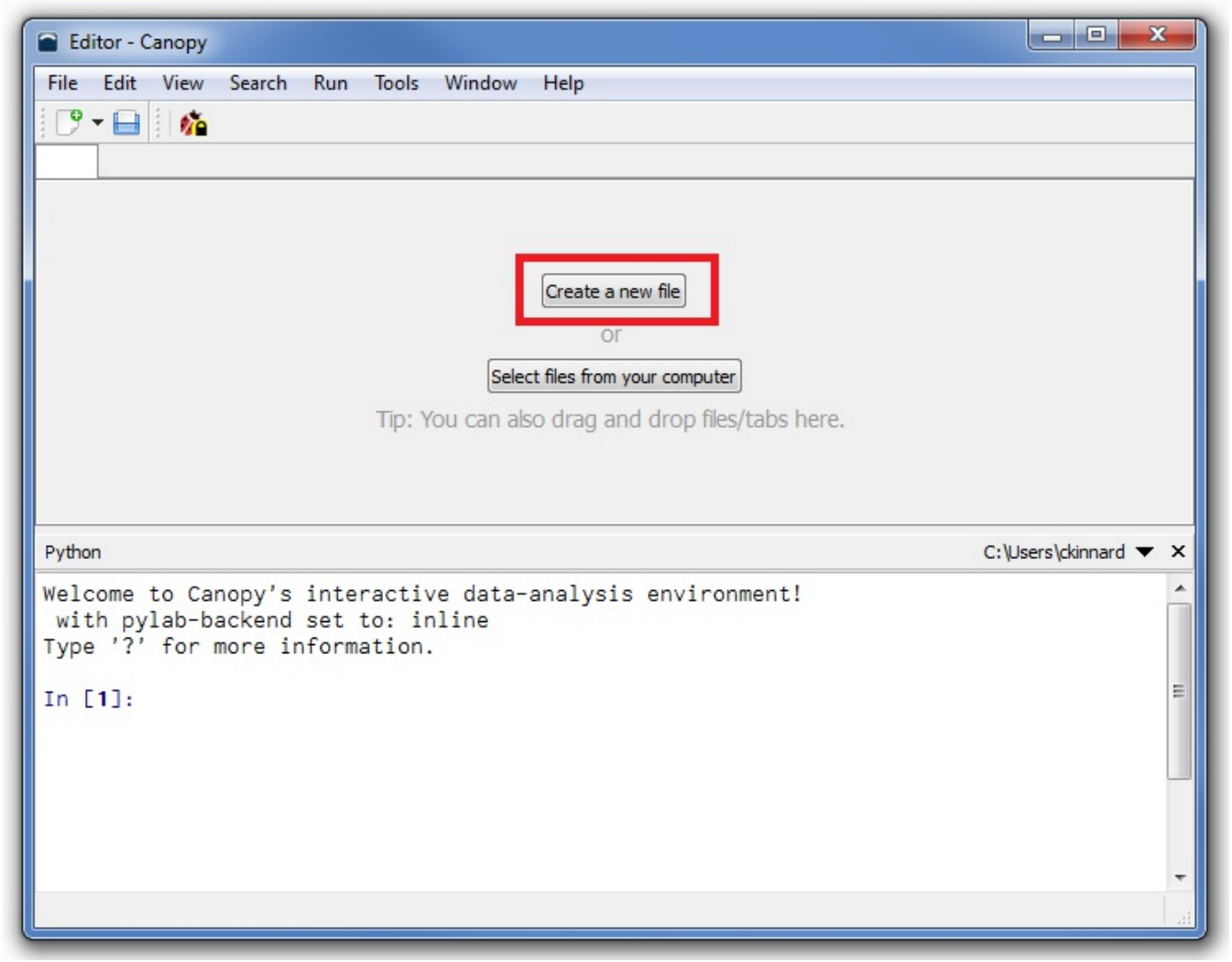
Materials

- Computer with Enthought Canopy distribution of *Python*® programming language

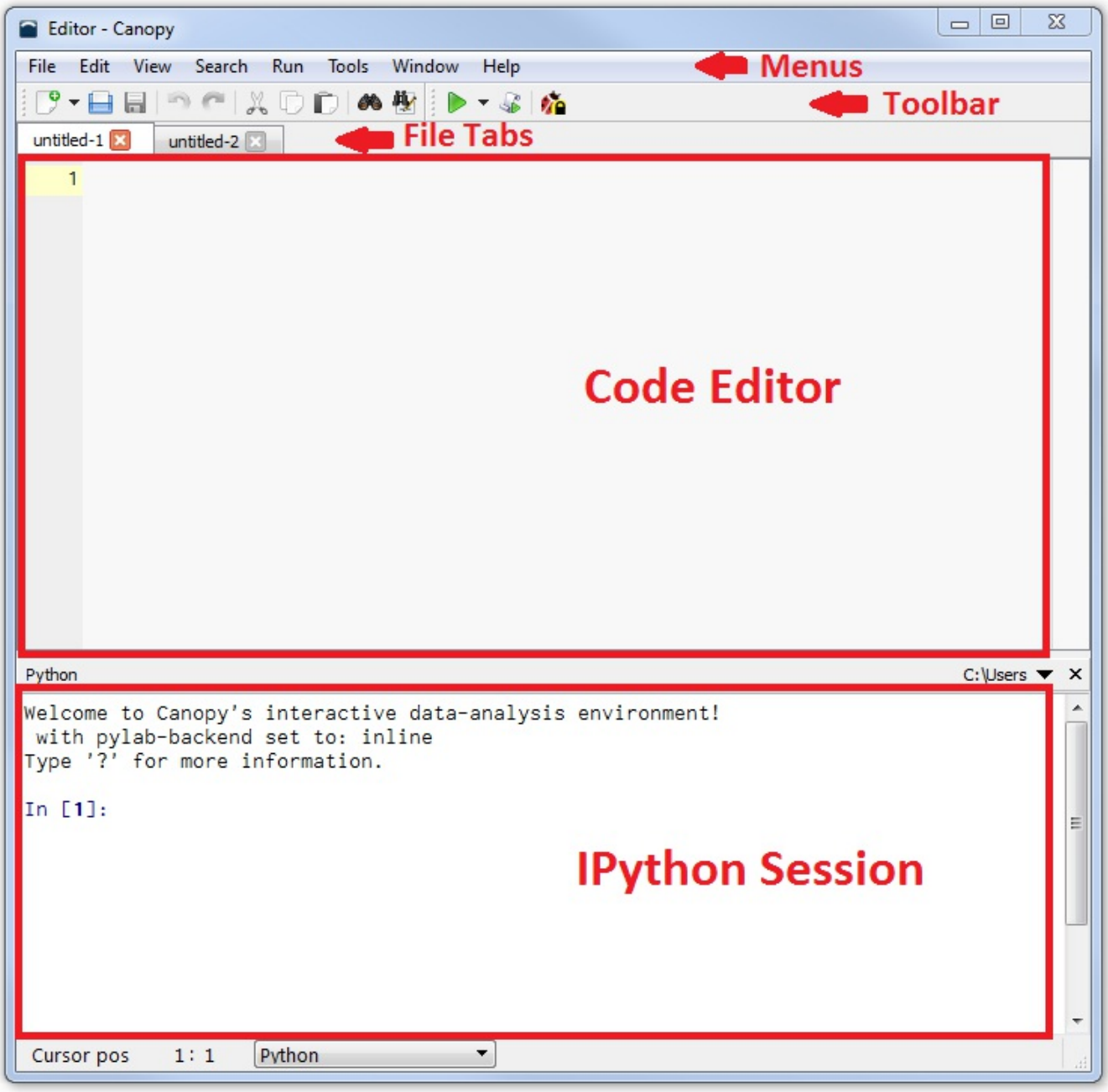
Procedure

1. Form pairs as directed by your teacher. Meet or greet each other to practice professional skills.
2. Launch Python using the Canopy distribution from Enthought.
3. Open the code editor window by clicking the **Editor** button.

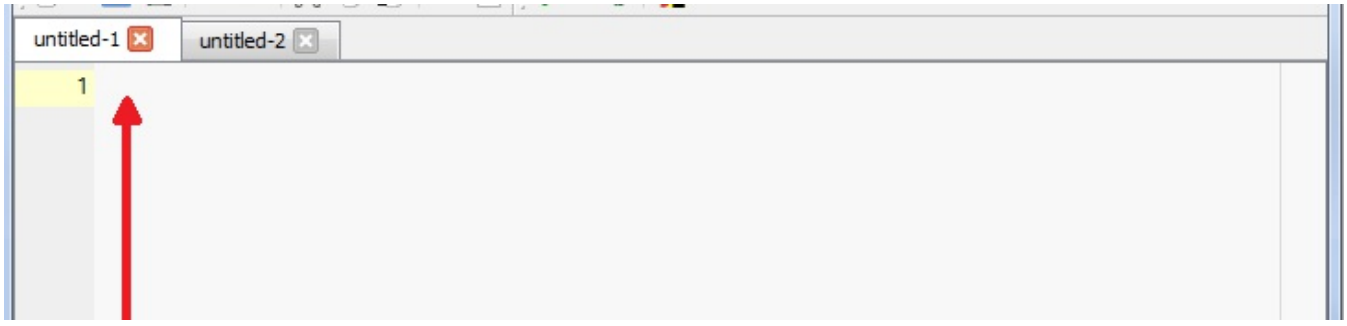
To open a new file, click **Create a new file**.



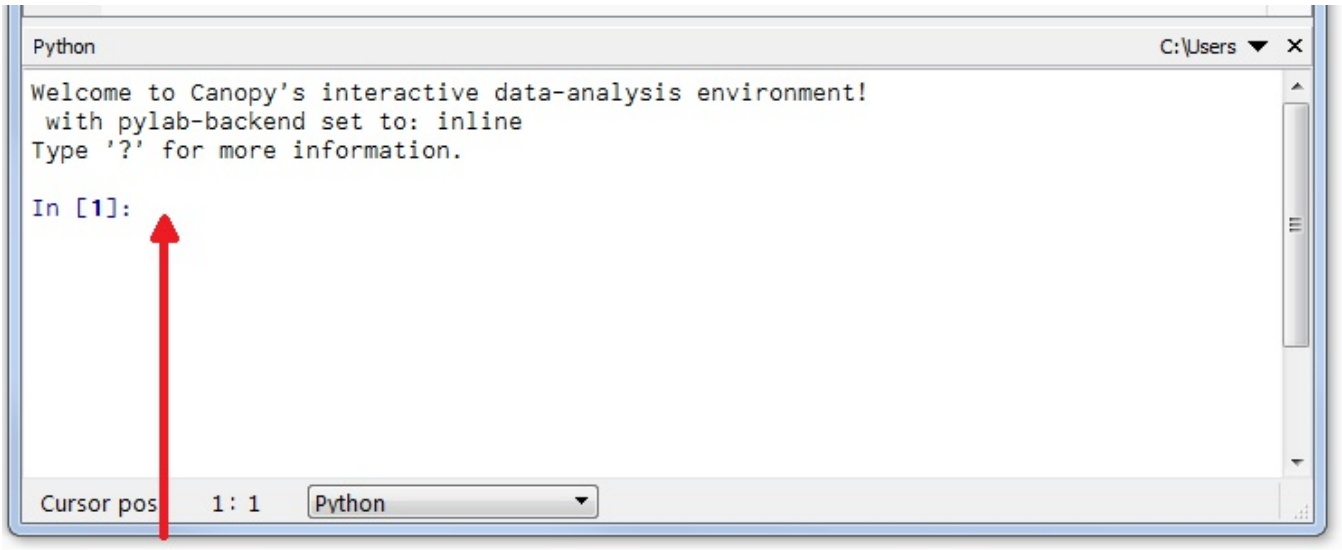
4. Shown below is Canopy's editor window. The editor window has two main parts: the editor (the "code editor") and the interactive command line session (the "IPython session.")



You can enter Python code in both the code editor and in the IPython session windows. To help clarify where you should place code samples, you will see two different styles of code blocks throughout these lessons.



```
# Code intended for the code editor appears in a
# YELLOW code block and includes line numbers
```

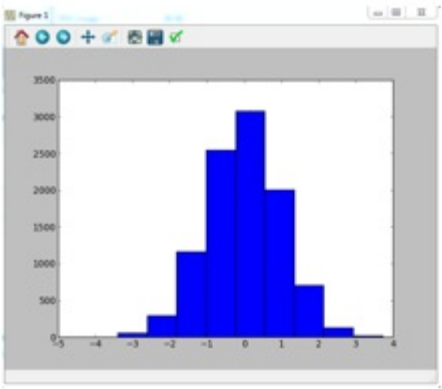


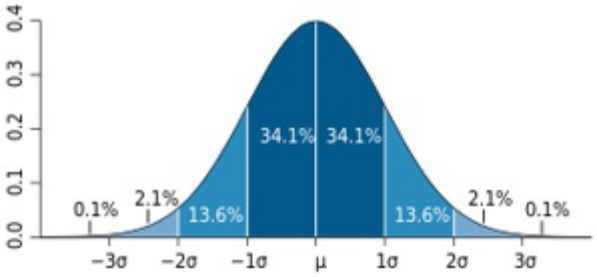

```
In []: # Code intended for the IPython interactive session
In []: # appears in a GREEN code block with In/Out prompts
```

5. *Python* is very powerful because it is easy to incorporate code that other people have already written. These are called modules, and they can be imported from libraries of packages that come with Canopy. Try the following input in the IPython session, which is the lower portion as shown in the previous step.

```
In []: import matplotlib.pyplot as plt
In []: a = randn(10000)
In []: plt.hist(a)
In []: plt.show()
```

You should see a new window titled 'Figure 1' containing output similar to the one shown below at left. This new window might be behind another window.





You have just made *Python* pick 10,000 random numbers from a normal distribution (the "Bell" curve) and plot them in a histogram! Before diving into the power that the *Python* libraries will give you, you'll want to learn the basics of the language. As you begin to learn *Python*, you'll recognize some familiar elements from languages that we have used in the past as well as

some new ones. After you learn a bit of one text-based programming language, learning other languages is much easier.

The difficulties of learning your first text-based language are well worth it. Just follow along in the IPython session.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

6. You can share your work with your teacher electronically. Logging everything that happens in the IPython session can also be a useful tool for you. Begin saving the IPython session as follows.

   - Change the **working directory** to your personal folder as directed by your teacher. A working directory is how we refer to the location in our file structure where a specific application like IPython will look for resources and save files.

     > Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

   - Turn on session logging. Enter the command below using your names in place of `studentNames`.

     ```
     In []: %logstart -ort studentNames_1_3_2.log
     ```

     Beware: the log is saved only later when you use the command `%logstop`.

   - Place a title in the logfile using a comment. In *Python*, the # symbol comments out the rest of the line. When you are finished typing the line, press enter.

     ```
     In []: # Jane Doe 1.3.2 IPython log
     ```

   - As directed by your teacher, you might use the log file to turn in your answers to the questions in this activity. If so, you will type your answers as comments in your IPython session, just as you typed the title into the previous step.

Part I: Numeric types

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

7. The IPython session will evaluate expressions, and if you don't tell it to do anything with the expression, IPython will send the result back to you as output. Try adding two integers:

```
In []: 5 + 3
Out[]: 8
```

Adding two integer types results in an **int**, which can only represent an integer. It is of **type** `int`. A variable is a place in memory to hold data, and a variable's type says what sort of things the bits in memory represent. The type `int` is a *Python* **native type**, meaning that the data abstraction is built into the language.

Some *Python* native variable types

Variable Type	Represents	Example Values
`int`	integers	`3, 5, or -7000`
`float`	any real number	`3., 5.2, or 6.3e-4`
`boolean`	True or False	`True`
`str`	any characters	`'hi there'` or `'2?'`

This means 6.3×10^{-4}, scientific notation for 0.00063

If you put a decimal point in a number, that number is identified as the variable type **float**, short for floating point representation. Floats have limited precision but can approximate any real number, including integers. Ignore the last few digits if a lot of digits come after the decimal point!

```
In []: 5 + 3.  # Note the decimal point!
Out[]: 8.0
```

Notice that adding an `int` and a `float` returned a `float`—you can tell by the decimal point. Do the following two inputs give results that are integer types (no decimal point) or floating point types (with decimal point)?

```
In []: 7*2
In []: 7*2.
```

- The kind of number that you get when you multiply depends on whether you used ints or floats. Discuss with your partner and explain.

  Your teacher might direct you to use the IPython logging session to turn in answers to questions like these. If so, use the same type of inline comment used to title the session log in step 6, but add a backslash (\) character to continue a comment. Inline comments begin with a #, and each additional comment line must start with a #. A sample is shown here. The parentheses contain an instruction to you.

  ```
  In []: # 5a. int*int returns an int \
    ...: # but int*float returns a float \
    ...: (Press enter here to end the continued input)
  ```

- What does division return? Try the following and then explain using an inline comment,

filling in the blanks.

```
In []: 7/2
In []: 7/2.
In []: # 7b. int/int returns _______ but int/float returns _______
```

8. You can assign a value to a variable.

   ```
   In []: a = 2
   ```

   *Python ex*ecutes the assignment command, but nothing is returned. Only the **state** is affected: the variable `a` is now bound to bits in memory that represent the value 2. When you assign a value to a variable, the variable must be on the left side of the `=`. It must not be part of an expression like `x+1`. This is one of the **syntax** rules for *Python*. The syntax of a language specifies how the symbols of the language are allowed to be put together.

   Write code to assign the value 16 to the variable `student_age`. When activities ask you to write some code, you can annotate your work with a comment, using the question number from the activity and/or a description of the task. Begin with the # sign.

   ```
   In []: # 8. Assignment

   In []:(write code to assign value 16 to variable student_age)
   ```

9. You can **evaluate** variable expressions:

   ```
   In []: a
   Out[]: 2

   In []: a * 7 # multiplication
   Out[]: 14

   In []: 3 ** 2 # exponentiation
   Out[]: 9
   ```

   What is 23^{43}? The `L` in the output stands for another native type: long.

10. You can also **call** existing functions and they will return a value:

    ```
    In []: abs(-7) # absolute value function
    Out[]: 7
    ```

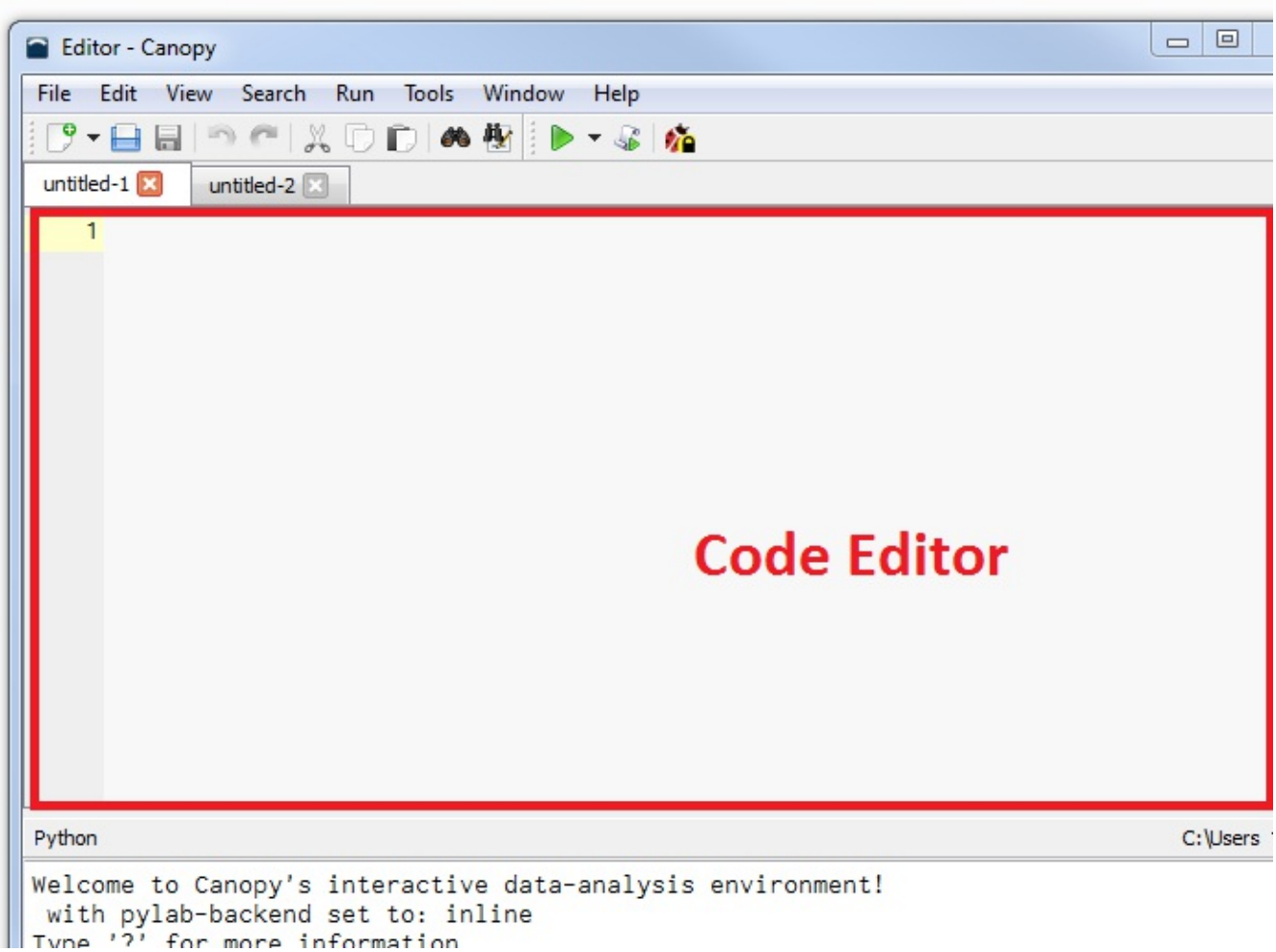
    The function `abs()` is a Python **built-in function**. A full list of the 80 built-in functions is available at **http://docs.python.org/2/library/functions.html**.

Part II: Function Definitions

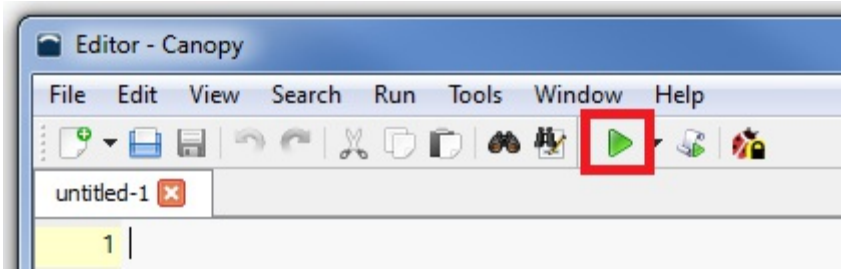
11. Enter the following code into the code editor. Notice the color of the code block is yellow and includes line numbers, indicating the code belongs in the code editor, not the IPython session.

You can use copy and paste, or you can type the code yourself. If typing, note that the editor often indents your code automatically. If you need to manually indent, the indentation is 4 spaces.

```
def add_tip(total, tip_percent):
    ''' Return the total amount including tip'''
    tip = tip_percent*total
    return total + tip
```



Unlike code typed into the IPython session, code in the code editor is not run when you press enter. It is executed when you press the green "Run" button.



What advantage might there be to executing each line of code as you type it in the interactive IPython shell?

What advantage do you think there might be to editing many lines of code in the code editor before execution?

12. *Python* uses indentation to group lines of code into blocks. A block of code refers to a group of lines, like the “chunks” of code in Scratch™ or App Inventor. It is unfortunate that we couldn't call them blocks in all languages and use another word or phrase like “puzzle pieces” for the individual lines of code in the “block” languages.

    The keyword `def` requires a block of code to define a new function. The block begins with the colon at the end of line 1. Lines 2 through 4 are indented four spaces to be part of that block.

    Python uses indentation. What did Scratch or App Inventor use to group together the lines of code in an if-chunk or an else-chunk in an if-else structure?

13. Line 2 begins and ends with `'''` . Between these sets of three single quotes, all text is commented out, just like the #, except that the commenting can continue over multiple lines. We call this a **multi-line comment**. M**ulti-line comment** When used at the beginning of a function definition, the comment is called a **docstring** because it can be accessed by a user with the command docstring because it can be accessed by a user with the command `help(add_tip).help(add_tip).`

    When you define a function with `def`, why would you want to use a docstring?

14. Line 3 assigns the value calculated by `tip_percent*total` to a new variable, `tip`. Because tip is created within a block of code, it has a **local scope**. That means that the variable can only be used inside that block. Once the block is done executing, the variable tip will no longer be referenceable by future program statements and will be deleted automatically during **garbage collection**. Garbage collection refers to the important task of releasing reserved memory once it is no longer needed by the program. The value of `total+tip` will be returned as a value to the IPython session or to whatever code called the function.

    What might be some advantages of destroying local variables once the block is done executing?

15. Run the code in the code editor by clicking the green arrow. You won't observe any output, but your code has now defined a new function, binding the function name `add_tip()` to the code, and adding the function name to the **namespace**. The namespace is the collection of words that have meaning to the **interpreter** or **compiler**.

    The interpreter or compiler translates the computer programming language into machine code for execution by the processor. Compilers perform this translation before executing the program, while interpreters translate into machine code a little bit at a time while the program executes. Generally, compiled languages are faster when executing, but interpreted languages allow you to work more easily with the code while you are creating it. *Python* is partly compiled and partly interpreted.

    To understand the compiler/interpreter difference, consider this analogy: If you were negotiating with a business partner who spoke another language, it would be possible to translate from one language to another a single sentence at a time and get reactions from the business partner after each sentence. This method would function like an interpreter. What would a compiler be like?

16. Once a function definition is executed, you can use the function you've defined as many times

as you wish. As shown in the two examples below, a value returned by a function can be used anywhere the value could have been used.

```
In []: a = add_tip(20, 0.15)
In []: a
Out[]: 23.0
In []: add_tip(30, 0.15)
Out[]: 34.5
```

17. Try this, discuss the output with your partner, and log a comment about it.

```
In []: add_tip(30)
```

Complete one or more of the following problems (a, b, and/or c) as directed by your teacher. You can create your functions by continuing the same *Python* file containing your definition of the `add_tip()` function. Make sure to strategize before you code, deciding what your algorithm will do. As you pair program, don't forget the steps of the software design process.

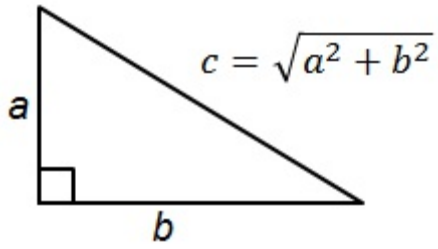
Normally, functions should **validate** input **arguments**. Validating input means checking that the input is what you expect, such as positive numbers for the lengths of a triangle's sides. Also, testing your code should usually involve writing a **test suite**, which is a program that does the testing for you. We'll come back to those ideas. For now, just demonstrate a working function in the IPython session as shown below.

Save your code by choosing **File > Save As…**. Use a filename like `JDoeJSmith_1_3_2.py` (but replace JDoe and JSmith with your names). Make sure to save the file in your directory and to name the file with a `.py` extension.

- Define a function `hyp(leg1, leg2)` that returns the length of the hypotenuse of a right triangle.

  Hint: `number**0.5` *takes the square root of a number*.

  ```
  In []: # 17a. Hypotenuse test
  In []: hyp(3,4)
  Out[]: 5.0
  ```



- Define a function `mean(a, b, c)` that returns the mean of three numbers.

  Hint: *divide by 3. to get a float*.

  ```
  In []: # 17b. Mean test
  In []: mean(3,4,7)
  Out[]: 4.666666666666667
  ```

- Define a function `perimeter(base, height)` that returns the perimeter of a rectangle with side lengths `base` and `height`.

```
In []: # 17c. Perimeter test
In []: perimeter(3,4)
Out[]: 14
```

18. Save your IPython session. It will be stored in the working directory of the IPython shell from when you started logging. Share the file with your teacher if directed to do so.

```
In []: %logstop
```

Conclusion

1. Read the introduction to this activity again. Describe something you would like to have automated by a program.
2. What are the native data types you learned about in this activity?
3. What are some differences between the command line of the interpretive IPython session and the code editor where you edit a file of code?
4. What do you think might be some of the advantages of putting code inside of a function definition?