Lesson 2: Introduction to Python

This is a class in Scientific Computing and its main target is to provide you with a set of concepts, tools, and tricks that will allow you to write an efficient scientific code in any language. In order to transform abstract lessons into practical examples that are easy to understand and remember, this class will be taught using Python as its official language.

1 Using python as a calculator

You should have already downloaded and installed the Enthought Python Distribution package Canopy. To begin using Python, simply open your Canopy program. You should then see the Welcome to Canopy window (Figure 1). If the Canopy Editor window is not already

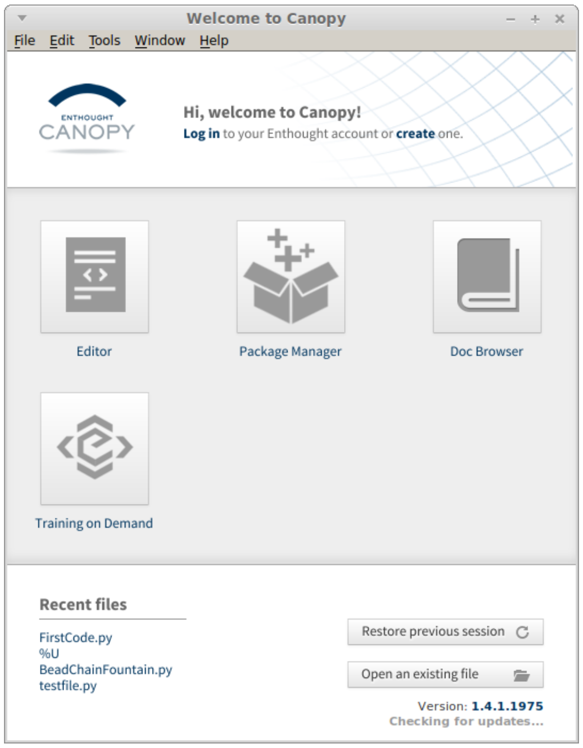
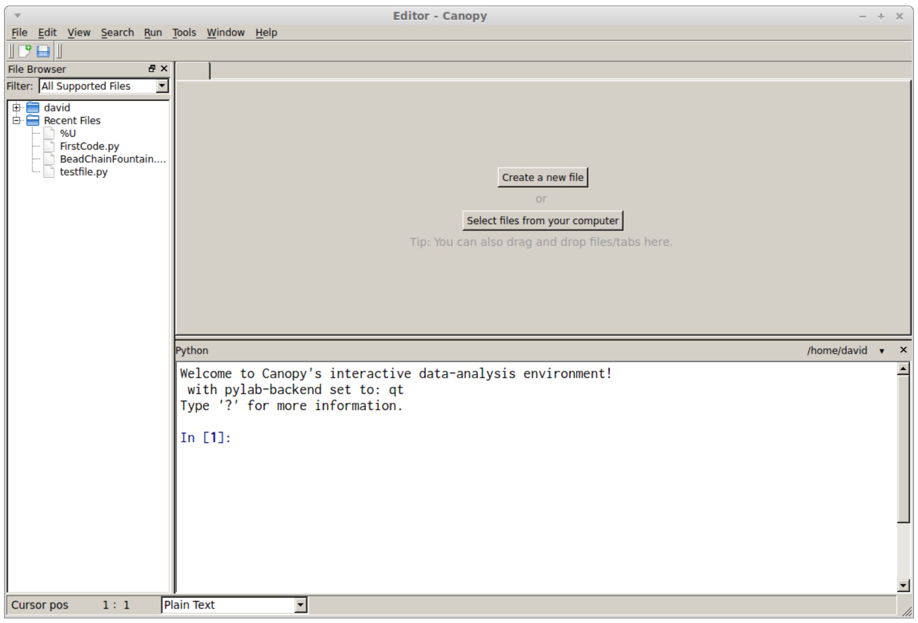


Figure 1: Canopy Welcome Screen.

open, click the Editor icon. You should now have the Canopy Editor shown in Figure 2. Along the left side is a file browser where you can search for Python files. The main area in the center is used for creating and editing Python files. Underneath this area is your Python



Tip: You can also drag and drop files/tabs here.

```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [1]:
```

Figure 2: Canopy editor.

environment. Here, you can use Python interactively with a command-line interface. For the moment, we can use Python as a calculator. While Python itself can only do a few basic mathematical operations, when Canopy starts it automatically loads a library of functions that include many higher mathematical functions used by scientists. We need to remember this for later, because we will need to explicitly import this library when we write our own Python programs. For now we can easily perform such mathematical operations as:

1. addition: $3 + 2$
2. subtraction: $3 - 2$
3. multiplication: $3 * 2$
4. divsion: $3/2$
5. powers: $3 * *2$

A more complex calculation, involving more than one operator, can be performed, but we must pay attention to how we write it. For example, the expression:

$$1 + 2 * 3 * *4 \tag{1}$$

will give as a result 163. Python performs calculations using the standard orders of operations, giving the highest priority to powers, then multiplication and division, and finally addition and subtraction. Therefore the expression above is equivalent to:

$$1 + (2 * (3 * *4)) \tag{2}$$

but not

$$(1 + 2) * 3 * *4 \tag{3}$$

or

$$1 + (2 * 3) * *4 \tag{4}$$

Exercise I: Evaluate all of the above expressions in python.

Particular attention must be paid when using the division operator /. Python uses this operator in different ways for integers and real numbers. The division between two integers will be always rounded to an integer. If asked to compute 9/5, python will return 1 as a result. A strong recommendation is to never use integers in your python programs, unless explicitly required. Instead of writing 9/5 you should write 9.0/5.0. Then python returns the correct result 1.8.

2 Writing it all in a routine

The first step to make Python become more than a calculator is to write a routine, or a series of commands that Python executes in sequence. For that we need a text editor, that is, a program that allows us to write expressions into a file. The main area of the Canopy editor serves as a text editor. (If you prefer, you can use a different text editor. However, it will be important to be consistent and stick with one choice. Also note that a word processor, such as microsoft word, cannot be used.)

A Python routine is a file containing a series of commands. It is typically saved in a file with the .py extension. To create a python file, start by clicking on the Create a new file button in the Canopy editor. At the top of the new file type the following:

```
# your name
# the date
# Python code for PY251, lesson 2
```

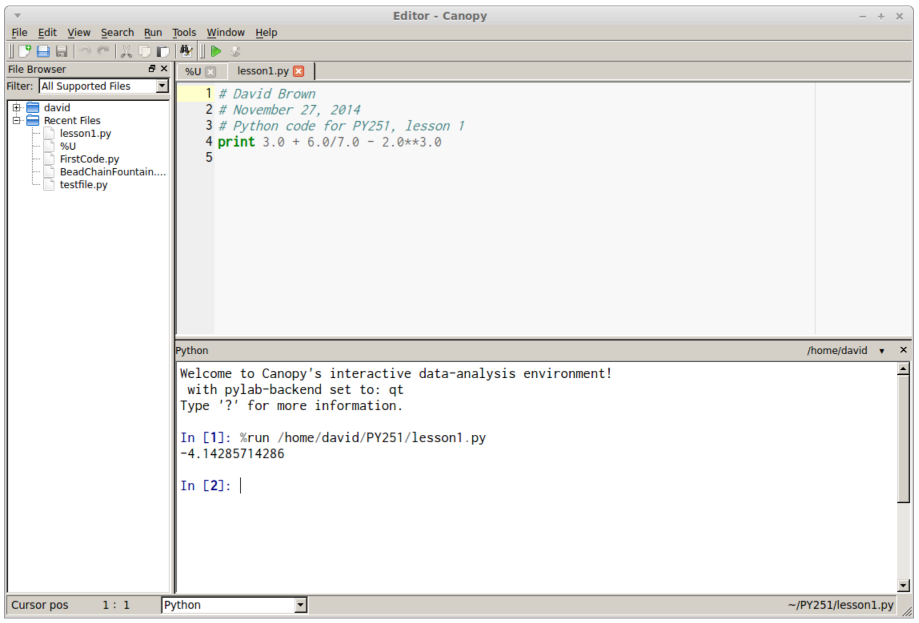
(Replace "your name" with your name and "the date" with the date.) These lines form the header. They are comment lines (lines that begin with the "#" character) that contain a brief explanation of what the routine does. Comments are an important part of readable code. Remember, computer codes should be written so that others can easily read it, and comments are vital to this goal. Every program you write should include a header and should be reasonably well commented throughout.

So far this program is only comments, so it doesn't actually do anything. Add the following line:

```
3.0 + 6.0/7.0 - 2.0**3.0
```

If you run the program as is, python will compute the result but it will not display it. You need to explicitly tell Python to print the result with the print command. Edit the above line to this:

```
print 3.0 + 6.0/7.0 - 2.0**3.0
```



```
# David Brown
# November 27, 2014
# Python code for PY251, lesson 1
print 3.0 + 6.0/7.0 - 2.0**3.0
```

```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [1]: %run /home/david/PY251/lesson1.py
-4.14285714286

In [2]: 
```

Figure 3: Canopy editor with a simple routine.

Now save your program with an easily identifiable file name. If you've created the program using the Canopy editor, then click on the "Run the current file" arrow. Your Canopy editor should now look something like Figure 3.

Exercise II: Write the above program and execute it.

3 Strings and variables

3.1 Strings

Python can deal with much more than just numbers and mathematical operators, it can also deal with lists of characters. A character is anything that you can type on a keyboard. If we enclose characters with a set of single quotes (`'`) or double quotes (`"`), Python will interpret the characters as text, called a string. For example, `'`NC State`'` and `"`University`"` are strings. Most often a string will be used with the print command; for example,

```
print "PY251 is a blast."
```

Strings are often used to prompt the user of the routine to enter information, or to give the user context for output from the routine. As a first step to using strings, write the standard first routine for any language, the "Hello World" routine.

Exercise III: Write a routine that will output the words `Hello World`.

3.2 Variables

Consider the motion of a ball thrown vertically. It's height as a function of time is given by:

$$y(t) = v_0 t - \frac{g}{2} t^2 \tag{5}$$

where we've used the origin of coordinates as the initial height. Suppose we want to compute the position of the ball at $t = 2\,\mathrm{s}$, given an initial velocity of $v_0 = 10\,\mathrm{m/s}$. Here is a simple code that will carry out the calculation:

```
# your name
# the date
# Height of an object thrown vertically upward

t = 2.0    # time at which we want the position (s)
v0 = 10.0  # initial velocity (m/s)
g = 9.8    # acceleration due to gravity (m/s**2)

y = v0*t - 0.5*g*t**2  # compute the vertical position

print y
```

There are few things to note about the above routine. First, comments are used throughout the code to explain what is going on. Commenting your code well will help anyone who reads it later—including yourself. Second, we assigned *variables* to the numbers. This is especially useful when the same quantity (like the time t) will be used multiple times throughout the routine. If you decide to change the value of that quantity, you only have to make the change in one place. Finally, blank lines have been inserted between the logical breaks in the code. The code begins with a header. The next section is where all of the variables are defined. Next is the section where the result is computed. The final section is where the output is returned. Separating the code like this improves readability of the routine.

To further improve things, we can modify the print statement to include some explanation. Change the final line of your routine to something like:

```
print "The height at t = {} seconds is y = {} meters" .format(t,y)
```

There are different ways to create this same output. Try the following:

```
print "The height at t = ", t, "seconds is y = ", y, "meters"
```

Here, the output consists of strings and variables. Here is another method:

```
print "The height at t = "+str(t)+" seconds is y = "+str(y)+" meters"
```

In this case, we have transformed the numbers into strings and then concatenated everything into a single string.

The first option is called printf formatting. Most programmers prefer this method of creating a print statement.

Exercise IV: Write the routine to compute the height of an object. Try all three formatting options for the output.

4 Line input

You will not want to edit your routine every time you change the initial velocity or the time. On the other hand, it's pretty certain that the acceleration of gravity is going to remain nearly constant for quite a while. You can create a routine that, instead of having the time and velocity specified, the user is prompted to type them in. To do this, use the `raw_input()` command.

```
# your name
# the date
# Height of an object thrown vertically upward

# time at which we want the position (s)
t = raw_input('Time in seconds: ')
t = float(t)
# initial velocity (m/s)
v0 = raw_input('Initial velocity in m/s: ')
v0 = float(v0)
# acceleration due to gravity (m/s**2)
g = 9.8

y = v0*t - 0.5*g*t**2  # vertical position

print "The height at t = {} seconds is y = {} meters" .format(t,y)
```

The `raw_input()` command asks for user input and interprets the input as a string. The `float()` command converts the input into a real (float) number. The `int()` command converts the input into an integer. You can convert a number to a string with `str()`.

It is sometimes useful to input more than one string with a single prompt. For example, the command

```
a,b = raw_input('Input two numbers: ').split()
```

allows the user to insert two numbers separated by a blank space. The `raw_input` command will assign the first number to `a` and the second number to `b`.

Homework: Write a routine that asks you for your name, your birth year, and your height in feet and inches. The program should print out:

```
Hello World, my name is [your name] and I am [your age] years old.  I am [your height in meters] meters tall.
```

Your homework should be turned in as a LaTeXdocument. The document should include a title (such as "Lesson 1"), and author (your name). It should include a copy of your python code. You can insert your code into the LaTeXdocument using the "verbatim" environment:

\begin{verbatim}

```
# This is my python code
x = 17
y = 12
print x + y
```

\end{verbatim}