Programming Basics

Michael Herzog, Sarah Mann, and Qiyam Tung

February 17, 2012

1 Introduction

This semester, we will try to do interesting projects with programming. However, this will require you to learn how to do some basic programming. We will not have time to cover everything in full detail. If you want to learn more or want extra practice, you can check out the great tutorials they have online at `http://docs.python.org/tutorial/`. If you have trouble with any of this code, feel free to e-mail me at `smann@math.arizona.edu`.

2 Objectives

1. Download and install the Python interpreter
2. Learn basic operations (print output, arithmetic, etc.)
3. Conditional statements (making a choice)
4. Hints for debugging (what to do when it doesn't work)

3 Downloading and Installing Python

We will be using a packaging of Python provided by Enthought Scientific Computing Solutions. To download Python,

1. Go to `http://www.enthought.com/products/edudownload.php`, and request an academic license. Follow their instructions to download.
2. The download's page has a variety of download's of this software for a range of operating systems.
   - **For Windows XP Users**: First, we need to find out what type of computer you have.
     (a) Click on the Start button
     (b) Click on the "cmd" button
     (c) Type "winmsd.exe" and hit OK.
     (d) Look for the item in the right window "Processor"
     (e) If it has 64 anywhere in its value, then download the 64-bit version `epd-7.2-2-win-x86_64.msi`.

(f) Otherwise, download the 32-bit version `epd-7.2-2-win-x86.msi`.

- **For Windows 7 Users**: First, we need to find out what type of computer you have.
  (a) Click on the Start button
  (b) Type "system" in the start search box.
  (c) If under System, it says 64-bit, then download the 64-bit version `epd-7.2-2-win-x86_64.msi`.
  (d) Otherwise, download the 32-bit version `epd-7.2-2-win-x86.msii`.
- **For Mac Users**: regardless of your processor, download the 32-bit version `epd-7.2-2-macosx-i386.dmg`.

3. Install the file you downloaded in whatever manner is appropriate for your operating system.

4 Basic Operations

4.1 Opening the Editor

The Enthought Python installation comes with the Python IDLE editor and shell. To start using Python, open the program "IDLE". This should be in the Enthought Application folder. Search for it if you have trouble finding it. Opening the IDLE should open a new window labeled "Python Shell" which will have a few lines of text at the top of it then a line that looks like this:

```
>>>
```

This is the Python interpreter in Interactive Mode. You can type commands here, and as soon as you are done issuing a command, it will do what you tell it to. The disadvantage is that you won't have any commands saved. We'll get to that later.

4.2 Hello, world!

Traditionally, your first program is to print "Hello, world!" to the screen. We'll do that now.

1. Type the following (just the text after the >>>)

```
>>> print 'Hello, world!'
Hello, world!
```

2. Hit Enter and the program should print out the message (we call it a string).

Congratulations! You've told the computer to print the message "Hello, world!". Now try asking Python to print some other message.

4.2.1 Notes from Hello, world!

1. Anything between the single quotes (they can also be double quotes) is considered a **string**. It is the textual representation of anything in the program.
2. Python commands, such as `print`, are **case-sensitive**. In other words, typing `Print` as a command instead of `print` will give you an error, as you can see below. Try this yourself!

```
>>> Print 'Hello World'
  File "<stdin>", line 1
    Print 'Hello World'
                       ^
SyntaxError: invalid syntax
```

3. As you can see from the above example, Python's error's can be hard to understand. You will get better at reading them as you get more experience with Python. We'll get to some tips on what to do if you get an error at the end of this document.
4. When you are working with Python in interactive mode, you usually won't need to explicitly tell it to print things. For example try typing `>>> 'Hello, world!'` into Python. You should get exactly the same result as when you typed `>>> print 'Hello, world!'`. For now, we won't use `print` much, but we will need it again later.

4.3 Basic Arithmetic

Python is good at working with numbers, and can do most things your calculator can do, and many more things your calculator can't do. Here, we will explore some basic arithmetic.

Type each of the following commands into your Python interpreter, and write down the results you get.

```
>>> 2

>>> print 2

>>> 5+2

>>> 2*3

>>> 10/2

>>> 5**2
```

Notice that the `**` operator means "to the power of", so when we type `5**2` into Python, we ask it to compute 5^2.

4.4 Types

In the last command, your output was 6. What if you used

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

What did you get?

The reason it wasn't 6 was because the type of data you gave it was string. In other words, Python only sees it as text, not numbers. There are many different types of data in Python, but for now let's just worry about `numeric` and `string` types.

4.5 Numeric types

Numeric types, as you might expect, represent numbers. You know from Algebra that there are many kinds of numbers like integers, real, and complex numbers. In the Python programming language, you call them `int`, `float`, and `complex`, respectively. The one you use the most is `int` because we use them in loops (see later), but since this is a math class, expect to use `float` and `complex` as well. Why do we distinguish between them? Well, try the following code and observe the output.

```
>>> 5.0/2.0

>>> 5/2
```

When your numbers have a point `.` in them, then Python interprets that as a real number. Otherwise, it's an integer.

But notice that the second line gave us 2. That's because if you're doing division with integers, your result cannot be a real number. So, the result is whatever you get when you do real number division and throw away everything after the decimal.

So what if I mix the two together? Will I get an integer or real number? Try it!

4.6 String types

The other data type is called **string**. As mentioned before, Python sees this as text. You can print them, but you can also link two strings together. For example, the `+` operator, when used on two strings, means that you should link (**concatenate**) them into one.

```
>>> 'My name is Bond. ' + 'James Bond.'
My name is Bond. James Bond.
```

Note that it is only defined for strings, so don't try mixing numeric types with strings like below.

```
>>> 'This will not work ' + 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

If you want to print a number with a string, just convert the number to a string. Try this:

```
>>> 'My favorite number is ' + str(3)
```

The `str` operator converts any object type into a string. You can also convert a string (that ony contains a number!) to a number.

```
>>>  int('3') + 3
6
>>> print float('2.0')*5.2
10.4
```

5 Variables

Variables allow you to store answers and keep them for later use. They can also be used as user input.

Suppose I want to be able to evaluate the given statement

$$y = x^2 + 2 \tag{1}$$

for any given value of x. Let's say I want to find y when x is 3. The way to do that would be.

```
>>> x = 3
>>> y = x**2 + 2
>>> print y
11
```

Here, the equal symbol (`=`) means "assign". So, x is assigned a value of 3 and y is assigned a value of $x^2 + 2$ or (in this case) $3^2 + 2$. It is *not* testing equality of two values. We'll do that later. Also, notice that I type `>>> print y` instead of just `y`. Try typing `y` and see what happens. You have to use `print` to see the values of variables.

5.1 Example

The following code asks for your name and stores it in a varible called `name`. Try it.

```
>>> name = raw_input("What's your name, kid!? ")
What's your name, kid!?
```

At this point, you need to type in your name, which will give you.

```
>>> name = raw_input("What's your name, kid!? ")
What's your name, kid!? Qiyam
>>> print name
Qiyam
```

Of course, it would be nice to be able to print something more meaningful than just your name.

```
>>> print name + ' is your name? What a horrible name.'
Qiyam is your name? What a horrible name.
```

Notes on this example:

1. The function `raw_input(mystring as string)` tells python to display the string `mystring` to the user, wait for the user to type something and hit enter, then return whatever the user typed. So the variable `name` will contain whatever the user types in.
2. When we tell Python `print name + ' is your name?...'`, it prints whatever is stored in the variable `name` followed by `' is your name?...'`. We use the `+` here to **concatenate** the two strings together.

6 Running from a file

Interactive Mode is useful for testing out new functions and techniques. However, most of the time, we will want to run many commands at once. While this is possible in interactive, it is far more convenient to write and test it from a file.

This section will teach you to write python code in a simple text file. You can edit it from any text editor (Microsoft Word, Notepad, etc.) but a editor specifically for Python highlights key words and it makes it easier to read.

1. From the IDLE window (the window you're typing Python commands to) go to the menu window and select `New Window`.

2. This opens up a file that you can edit.
3. Type any previous command in this file.
4. You must save the file before you can run it. Save it somewhere convenient and name it `first_program.py`.
5. To run it, select the menu `Run` → `Run Module` or hit `F5`

7 Program Assignment

Write a program that will calculate the value of

$$\frac{x-2}{x^2+3x-10}. \tag{2}$$

Your program must be able to take in x for input and print out the result. Call your program `rational.py`. Hint: use the function `float` to convert user input to a floating point number.

Here is an example of what your program should look like when you run it.

```
$ python rational.py
What is x? 2.5
The result is: 0.133333333333
$ python rational.py
What is x? 2.3
The result is: 0.13698630137
$ python rational.py
What is x? 2.2
The result is: 0.138888888889
$ python rational.py
What is x? 2.1
The result is: 0.140845070423
$ python rational.py
What is x? 2.01
The result is: 0.142653352354
```

8 Conditional Statements

In this section, we'll learn how to use conditional statements, which lets the program do different things depending on the input. From now on, unless stated otherwise, we will be typing our code in a text file, so create a new file called `conditional.py`. Copy the text in the box below into this file.

```
money = 0.0

if money == 0.0:
  print "You have nothing. You're a bum."
```

Here, we give the variable `money` the value 0. Then, if money is zero, we tell the user they are a bum. Notes on this example:

1. The expression `==` is used to compare whether two numbers are equal. If it is true, then it will execute the statements under the `if` statement. Be careful not to mix this up with `=`, which assigns the value of the right side to the variable on the left side.
2. Notice the colon at the end of the if statement. If statements must always end with a colon.
3. Notice that last line is *indented*. Any statements that you want to run after the `if` statement must be indented.

When you run the program, you should get the message "You have nothing. You're a bum." Now edit the file so that `money = 1`. Run it again. Did you get the message?

No, that's because `money` is 1. In other words, your program can do different things in different conditions.

Let's observe some other operators. Copy the text below into a new file, then run the program a few times. Providing a different amount of money each time your run the program. What happens?

```
money = raw_input("How much money do you have? ")

money = float(money)

if money <> 0.0:
  print "Looks like you have some money"
if money < 100.0:
  print "You have less than 100 dollars"
if money >  60.0:
  print "Hey, you can buy a very expensive dinner for one."
```

There are several things to note here:

1. `raw_input` gives us a string result so if we are to compare it to 0, we need to convert it to a number (using the `float()` operator).
2. The `<>` operator tests whether the left hand side is not equal to the right hand side (it is the opposite of the `==` operator).

3. The `>` operator tests whether the left hand side is greater than the right side.
4. The `>=` operator tests whether the left hand side is greater than **or equal** the right side.
5. The `<` operator tests whether the left hand side is less than the right hand side.
6. The `<=` operator tests whether the left hand side is less than **or equal** the right side.

There is one more thing you need to know about `if` statements.

```
money = raw_input("How much money do you have? ")

money = float(money)

if money > 1000000.0:
  print "Wow, you're rich!"
elif money >= 2000.0:
  print "You could buy a really good computer with that."
else:
  print "You have less than 2000 dollars"
```

What the code says here is that you will only ever print one message. That is, if you have more than 1000000, then it will print a message and then skip the other conditional statements `elif` and `else`. However, if that's not true, then it will check to see whether you have 2000+ dollars. If true, it will print a statement. Otherwise, it will simply tell you have less than 2000 dollars.

The `elif` means "else if" or "if the previous thing wasn't true, see if this is true". And `else` simply means "if everything above wasn't true, do this".

9 Program Assignment

Modify your `rational.py` file so that it tells the user whether the given x is undefined at a given point (when you have division by zero operation). For example,

```
$ python rational.py
What is x? 2.0
The function is undefined at 2.0
```

Remember, the function is undefined at two points.

10 OK, but what do I need know by now?

- How to print data (strings and numbers)
- How to use variables (manipulation and assignment)
- How to use `if` statements.

11 Ack! My program won't work!

Programs can fail for many, many reasons. It is the price you pay for being able to generalize a task. Fixing an error takes time and experience to get good at.

However, just remember that a computer is a very logical machine. It complains when you have code that doesn't make sense. Programmers who design programming languages try to give you useful hints as to why you're program failed (e.g. you tried to divide by 0, you referenced a variable that doesn't exist).

Here is a list of common errors.

- You used the wrong number type. For example, in the `rational.py` program, make sure your numbers end with a `.0` (e.g. change `2` to `2.0`) to use real numbers instead of integers.
- Python is case-sensitive. That means that `print` is seen as a different from `Print`.
- No colon (`:`) after a conditional statement.
- Inconsistent indenting. The following code will cause errors because the two statements are not aligned.

```
if number == 0:
  print ‘‘Your number is 0’’
   print ‘‘Bye now’’
```

  To fix this, just make sure they are equally indented. It does not matter how deep they are indented, just so long as they are equally indented.

```
if number == 0:
  print ‘‘Your number is 0’’
  print ‘‘Bye now’’
```

- You are missing a parenthesis, quote, or some other kind of bracket. For example, if you're code doesn't work and looks like this

```
y = (x - 1.0/2.0
```

but what you really meant to say was this: $y = \frac{x-1}{2}$, then the following code is what you should write instead.

```
y = (x - 1.0)/2.0
```