Programming Basics

Michael Herzog, Sarah Mann, and Qiyam Tung

January 23, 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://download.enthought.com/epd_7.2/`.
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
```

Comments, Random Numbers, and Rock-Paper-Scissors

Michael Herzog, Qiyam Tung and Sarah Mann

January 31, 2012

1 Comments

Comments are lines in your code that don't do anything. They allow you to leave notes for yourself about what your code does. In Python, any line that starts with a # is a comment, as is anything appearing after a # on a line. Here is an example:

```
x = 0
# This is a comment
print x + 1 # I can also put a comment here
```

I would recommend putting comments at the top of your file to list important information, such as author, date, and the purpose of the program.

```
# Author: Qiyam Tung
# Date: Jan 19, 2011
# This program doesn't do much yet.
print "Beginning program..."
```

You can also comment code that you may want to use later.

```
x = 0
# print "Not sure if I want this line in the program yet"
print x
```

Code that has been commented well is much easier to read than code without comments. Often, you will write some code then come back a few days latter and have no idea how your code works. Comments are the solution to this problem! A good rule of thumb is that you should write as many lines of comments as you write lines of code. Let me repeat:

Your programs should always contain as many lines of comments as lines of code!

2 Rock-Paper-Scissors

2.1 Programming Assignment

Write a Python program that will play rock-paper-scissors. Your program should:

1. Ask the user to choose either rock, paper, or scissors.
2. Randomly choose either rock, paper, or scissors for itself (more on how to do this in a moment).
3. Print out both what the user chose and what the computer chose.
4. Determine a winner for the game, and print that as well.

Remember, rock beats scissors, scissors beats paper, and paper beats rock. Below is an example of how my version of rock-paper-scissors looks when I run it. You're doesn't need to be identical, but it should be similar.

```
Welcome to Rock, Paper, Scissors!
Please select either Rock, Paper, or Scissors:  rock
You selected rock
I selected Scissors.
You win!
```

2.2 Random Numbers

The computer's "strategy" in rock-paper-scissors should be to randomly chose either rock, paper, or scissors each time. The easiest way to do this is to use a random number generator. Here is an example of how to do this:

```
# Sarah Mann
# January 3, 2012
# This program is an example of randomly generating a number

# First, I have to import the "random" library.  This adds a bunch of extra commands
# to python that generate random numbers
import random

# Now I will randomly choose 0, 1, or 2 and store it in the variable my RandNum.
```

```
# random.randrange(a, b) randomly choses an integer that is greater than or equal to a
# and less than b.  In the example below, I will get either 0,1, or 2.
myRandNum = random.randrange(0,3)

# print the result!
print myRandNum
```

In your program, if the random number generator gives you a 0, you should treat it like the computer chose rock. A 1 would mean paper, and a 2 would mean scissors.

Math Functions and Quadratic Functions

Sarah Mann

February 2, 2012

1 Math Functions

Basic Python only knows how to do a few things mathematically. It's kind of like a fifth grader in that way. It can add, subtract, multiply, and divide, but it doesn't know about square roots or logarithms or other useful stuff. Unlike a fifth grader, it does not take 4 years to teach Python about these other mathematical functions; all you have to do is import the math library. Any time you want to use a more advanced math function, you should include at the top of your program (or type into the IDLE)

```
import math
```

This gives Python knowledge of a bunch more mathematical function. Here are a few examples of new things you can do once you import the math library. Try this in your IDLE!

```
>>> import math
>>> math.sqrt(4)
2.0
>>> math.log(8,2)  # this finds the log base 2 of 8
3.0
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

You can find a full list of math functions at `http://docs.python.org/library/math.html`. You will need to use other functions from this library in future programming assignments.

2 Programming Assignment: Quadratic Functions

2.1 Programming Assignment

Write a program that finds the (real) zeros and the vertex of a quadratic function, $f(x) = ax^2 + bx + c$. You will need to research appropriate equations to calculate the zeros and vertex. Your program should:

1. Ask the user to provide values of a, b, and c.
2. Determine how many (real) zeros the function has. Remember, a quadratic function may have zero, one, or two zeros.
3. Calculate the zero(s) and print them for the user.
4. Calculate the vertex and print it for the user.
5. Contain at least as many comments as lines of code.

Below are three examples of how my version of this program looks when I run it. Your's doesn't need to be identical, but it should be similar. Remember to comment your code well! Your future self will thank you for it.

```
This program can find the zeros of quadratic functions.
If your function is written in the form ax^2 + bx + c,
what is a?1
what is b?1
what is c?1
There are no real zeros.
The vertex is (-0.5,0.75)
>>> ========== RESTART ==========
>>>
This program can find the zeros of quadratic functions.
If your function is written in the form ax^2 + bx + c,
what is a?1
what is b?4
what is c?1
There are two zeros,-0.267949192431 and -3.73205080757.
The vertex is (-2.0,-3.0)
>>> ========== RESTART ==========
>>>
This program can find the zeros of quadratic functions.
If your function is written in the form ax^2 + bx + c,
what is a?1
what is b?-4
what is c?4
The only zero is 2.0.
The vertex is (2.0,0.0)
```

A Few Short Programs

Sarah Mann

February 19, 2012

1 Introduction

This week, you will write a few short programs using the skills you already know. You should try to write these programs on your own with as little help as possible. Look at previous programs you have written and the previous handouts we have provided. Next week we move on to bigger things, so be sure you know how to do these basic tasks!

Here's the twist: for each program I am going to tell you the maximum number of lines of codes you may use to write your program. You will receive bonus points for writing your program using fewer lines of code.

Remember, you must comment your code! Comments do NOT count against your allotted number of lines.

2 Heads or Tails

Write a program that prints out "Heads" or "Tails" each time it is run. Your program should randomly choose which message it prints on each run. You should save your program as `Heads_Tails.py`. You are allowed 6 lines of code.

3 Greetings

This program will ask for the user's name and favorite number, then print that number and inform the user of the computer's favorite number. If the user's favorite number is x then the computer's favorite number is $2x + 1$. Here is how this program will look when it is run:

```
Hello! What is your name? Sarah
Nice to meet you, Sarah. What is your favorite number? 7
Sarah's favorite number is 7.0 but my favorite number is 15.0.
```

Blue text was printed by the program; black text was inputted by the user. Your program should replicate this *exactly*! You are allotted 4 lines of code. Save your file as `Greetings.py`.

4 Circles

This program will ask the user for the radius of a circle, then compute and print the area and diameter of the circle. Here is how this program will look when it is run:

```
This program calculate the area and diameter of a circle given its radius.
What is the radius of your circle?  1
The area of the circle is 3.14159265359 and the diameter is 2.0.
```

Blue text was printed by the program; black text was inputted by the user. Your program should replicate this *exactly*! You are allotted 5 lines of code. Save your file as `Circles.py`.

Lists

Sarah Mann

February 26, 2012

What follows below is only a brief introduction to lists in Python. For more informations, see `http://effbot.org/zone/python-list.htm` or `http://docs.python.org/tutorial/datastructures.html`.

Assignment *In this document, I will ask you a few questions about how to do things with lists. Your answer* should be in the form of a single line of code. Create a file (word document would be fine) that contains all your answers to turn in.

Note: All example code in this document can by typed directly into the IDLE. You do not need to save the code in a separate file. Text that appears after a >>> is entered into the IDLE, text in blue is Python's response.

1 Basics

Lists allow you to collect a bunch of related things together in one place. Lists are easiest to understand by example, so lets start there. To create a list containing the integers 1 through 5 and store it in the variable MyList you can type

```
>>> MyList = [1, 2, 3, 4, 5]
```

The square brackets indicate that this is a list, and the commas separate elements of the list. We can print the list:

```
>>> print MyList
[1, 2, 3, 4, 5]
```

We can access individual elements of the list. To print the first element of your list you can type

```
>>> print MyList[0]
1
```

Notice that the first element in our list was element 0. This is a quirk of computer science: computers count starting with 0! This means the third element of my list is `MyList[2]`.

Question 1: How do you print the 5th element of your list?

Python can tell us how many elements are in our list. This is called the length of the list.

```
>>> len(MyList) # get the length of MyList
5
```

2 Altering Lists

You can change an element in a list:

```
>>> MyList = [1, 2, 3, 4, 5] # Create MyList
>>> print MyList
[1, 2, 3, 4, 5]
>>> MyList[0] = 7 # Change the first element of MyList to 7
>>> print MyList
[7, 2, 3, 4, 5]
```

Question 2: How can you make the 4th element of MyList be a 9?

You can also add a new element to the end of your list:

```
>>> MyList = [1, 2, 3, 4, 5] # Create MyList
>>> print MyList
[1, 2, 3, 4, 5]
>>> MyList = MyList + [6] # Add a 6 to the end of MyList
>>> print MyList
[1, 2, 3, 4, 5, 6]
```

Question 3: How can you add a 7 to the end of MyList?

3 Constructing Lists: Range

The range function allows your to construct lists quickly and will be far more convenient, typically, than typing in the full list yourself. Range has three different usages:

1. `range(stop)` creates the list of integers from 0 to `stop`, not including `stop`.

```
>>>range(5)
[0, 1, 2, 3, 4]
```

2. `range(start, stop)` creates the list of integers from `start` to `stop`, not including `stop`.

```
>>> range(1, 5)
[1, 2, 3, 4]
```

3. `range(start, stop, step)` creates the list of integers from `start` to `stop`, not including `stop`, counting by `step`.

```
>>> range(1, 20, 3)
[1, 4, 7, 10, 13, 16, 19]
```

`step` can also be negative:

```
>>> range(0, -10, -1)
[0, -1. -2, -3, -4, -5, -6, -7, -8, -9]
```

For each of the following questions, construct the given list using the range function.
Example: `[5, 6, 7, 8]`
Answer: `range(5, 9)`
Question 4: `[10, 11, 12, 13, 14, 15, 16]`
Question 5: `[10, 12, 14, 16]`
Question 6: `[-35, -36, -37, -38, -39]`
Question 7: The list of the multiples of 5 that are less than 100: `[5, 10, 15, ..., 95]`

4 Strings are Lists

Strings are lists of characters. This means we can access the individual letters in a word:

```
>>> S = "Computer" # Store the string "Computer" in the variable S
>>> print S[0] # Print the first character in the string S
'C'
```

Question 8: How do you print the 5th letter in "Computer"?
Question 9: How do you print the 'r' in "Computer"?
Question 10: How do you find the number of characters in "Comptuer"?

Loops

Sarah Mann

March 1, 2012

Computers are really good at doing repetitive tasks. Everything we have asked of Python so far have been things that you can do easily without a computer. The power of computers is their ability to do boring things over and over and over again without whining! Now, we'll learn how to make them do repetitive tasks by using loops.

We'll discuss loops together in class, so I'm only going to provide examples here and some programming assignments. Save each example in this section as a `.py` file, then run them.

Assignment: There are both programming assignments and questions in this document. Create a .py file for each programming assignment, save all assignments in one folder, and email the folder to Mr. Herzog. **Be sure to comment your code!**. Put your answers to all questions in a document and turn that in as well.

1 Example: Printing elements in a list

This example should print each element from a list on a new line. Try it!

```
MyList = ["Welcome", "to", "loops", "in", "Python!"]
for MyString in MyList:
    print MyString
```

2 Example: Printing elements in a list, take 2

This example should print each element from a list on a new line. This should do exactly the same thing as the previous example, but here we don't define the list before the `for` statement. Try it!

```
for MyString in ["Welcome", "to", "loops", "in", "Python!"]:
    print MyString
```

3 Question 1

The below is identical to the code in Example: Printing elements in a list, take 2 above except it has one more print statement. How is the behavior of this code different from in the previous example? Why?

```
for MyString in ["Welcome", "to", "loops", "in", "Python!"]:
    print MyString
print MyString
```

4 Example: Printing "Hello, World!"

This example should print "Hello, World!" five times. Remember that `range(5)` creates the list `[0, 1, 2, 3, 4]`. Try it!

```
for i in range(5):
    print "Hello, World!"
```

5 Example: Printing Numbers

This example should print the integers 1 through 5. Try it!

```
for i in range(5):
    print i
```

6 Programming Assignment 1

Write a short program (two lines of code) that prints the integers 10 through 16.

7 Programming Assignment 2

Write a short program (two lines of code) that prints the even integers 10 through 16.

8 Programming Assignment 3

Write a short program (two lines of code) that prints the integers -35 through -39.

9 Programming Assignment 4

Write a short program (two lines of code) that prints the multiples of 5 that are less than 100.

10 Example: Summing Numbers

This example sums the numbers 1 through 100. Try it!

```
sum = 0;
for i in range(1, 101):
    sum = sum + i
print sum
```

11 Programming Assignment 5

Write a program that asks the user for two inputs, n and m, then computes the sum of the integers n through m, and prints out the result. Here is an examples of how this program should look when it is run:

```
This program sums the integers n through m.
n: 5
m: 8
The sum of 5 through 8 is 26
```

Your program should behave similarly, but feel free to write your own messages to the user.

12 Programming Assignment 6

Write a program that asks the user for two inputs, n and m, the computes the product of the integers n through m, and prints out the result. Here is an examples of how this program should look when it is run:

```
This program multiplies the integers n through m.
n: 5
m: 8
The product of 5 though 8 is 1680
```

13 Question 2:

There is exactly one difference between the code below and the code in "Example: Summing Numbers" given earlier. What is it, and what effect does it have on the behavior of the program?

```
sum = 0;
for i in range(1, 101):
    sum = sum + i
    print sum
```

14 Example: Printing Strings

This example prints a string one character at a time. Try it!

```
MyString = "Computer"
for i in range(len(MyString)):
    print MyString[i]
```

15 Programming Assignment 7

Write a program that asks the user for a word then prints that word out one letter at a time. Here is an example:

```
What is your favorite word? onomatopoeia
o
n
o
m
a
t
o
p
o
e
i
a
```

16 Programming Assignment 8

Write a program that asks the user for a word then prints every other letter in that word. Here is an example:

```
What is your favorite word? onomatopoeia
o
o
a
o
o
i
```

17 Programming Assignment 9

Write a program that asks the user for a word then prints the word out one letter at a time backwards. Here is an example:

```
What is your favorite word? onomatopoeia
a
i
e
o
p
o
t
a
m
o
n
o
```

Loops

Sarah Mann

March 1, 2012

Computers are really good at doing repetitive tasks. Everything we have asked of Python so far have been things that you can do easily without a computer. The power of computers is their ability to do boring things over and over and over again without whining! Now, we'll learn how to make them do repetitive tasks by using loops.

We'll discuss loops together in class, so I'm only going to provide examples here and some programming assignments. Save each example in this section as a `.py` file, then run them.

Assignment: There are both programming assignments and questions in this document. Create a .py file for each programming assignment, save all assignments in one folder, and email the folder to Mr. Herzog. **Be sure to comment your code!**. Put your answers to all questions in a document and turn that in as well.

1 Example: Printing elements in a list

This example should print each element from a list on a new line. Try it!

```
MyList = ["Welcome", "to", "loops", "in", "Python!"]
for MyString in MyList:
    print MyString
```

2 Example: Printing elements in a list, take 2

This example should print each element from a list on a new line. This should do exactly the same thing as the previous example, but here we don't define the list before the `for` statement. Try it!

```
for MyString in ["Welcome", "to", "loops", "in", "Python!"]:
    print MyString
```

3 Question 1

The below is identical to the code in Example: Printing elements in a list, take 2 above except it has one more print statement. How is the behavior of this code different from in the previous example? Why?

```
for MyString in ["Welcome", "to", "loops", "in", "Python!"]:
    print MyString
print MyString
```

4 Example: Printing "Hello, World!"

This example should print "Hello, World!" five times. Remember that `range(5)` creates the list `[0, 1, 2, 3, 4]`. Try it!

```
for i in range(5):
    print "Hello, World!"
```

5 Example: Printing Numbers

This example should print the integers 1 through 5. Try it!

```
for i in range(5):
    print i
```

6 Programming Assignment 1

Write a short program (two lines of code) that prints the integers 10 through 16.

7 Programming Assignment 2

Write a short program (two lines of code) that prints the even integers 10 through 16.

8 Programming Assignment 3

Write a short program (two lines of code) that prints the integers -35 through -39.

9 Programming Assignment 4

Write a short program (two lines of code) that prints the multiples of 5 that are less than 100.

10 Example: Summing Numbers

This example sums the numbers 1 through 100. Try it!

```
sum = 0;
for i in range(1, 101):
    sum = sum + i
print sum
```

11 Programming Assignment 5

Write a program that asks the user for two inputs, n and m, then computes the sum of the integers n through m, and prints out the result. Here is an examples of how this program should look when it is run:

```
This program sums the integers n through m.
n: 5
m: 8
The sum of 5 through 8 is 26
```

Your program should behave similarly, but feel free to write your own messages to the user.

12 Programming Assignment 6

Write a program that asks the user for two inputs, n and m, the computes the product of the integers n through m, and prints out the result. Here is an examples of how this program should look when it is run:

```
This program multiplies the integers n through m.
n: 5
m: 8
The product of 5 though 8 is 1680
```

13 Question 2:

There is exactly one difference between the code below and the code in "Example: Summing Numbers" given earlier. What is it, and what effect does it have on the behavior of the program?

```
sum = 0;
for i in range(1, 101):
    sum = sum + i
    print sum
```

14 Example: Printing Strings

This example prints a string one character at a time. Try it!

```
MyString = "Computer"
for i in range(len(MyString)):
    print MyString[i]
```

15 Programming Assignment 7

Write a program that asks the user for a word then prints that word out one letter at a time. Here is an example:

```
What is your favorite word? onomatopoeia
o
n
o
m
a
t
o
p
o
e
i
a
```

16 Programming Assignment 8

Write a program that asks the user for a word then prints every other letter in that word. Here is an example:

```
What is your favorite word? onomatopoeia
o
o
a
o
o
i
```

17 Programming Assignment 9

Write a program that asks the user for a word then prints the word out one letter at a time backwards. Here is an example:

```
What is your favorite word? onomatopoeia
a
i
e
o
p
o
t
a
m
o
n
o
```

Nesting Loops

Sarah Mann

March 7, 2012

You can nest an if statement or for loop inside another if statement or for loop. This documents explores how to do that. Answer the questions below in a document and turn them in. Complete the programming assignments below, save each of them as `.py` files and turn them in as well.

1 Example: Nested For Loops

Save this code in a `.py` file, and try running it. What do you see?

```
for i in range(4):
    for j in range(3):
        print("i is " + str(i) + " and j is " + str(j) + ".")
```

Here is another example. Try it. What do you see?

```
for i in range(4):
    print("i is " + str(i) + ".")
    for j in range(3):
        print("    j is " + str(j) + ".")
```

Here is another example. Try it. What do you see?

```
for i in range(4):
    print("i is " + str(i) + ".")
    for j in range(i):
        print("    j is " + str(j) + ".")
```

2 Question

There is exactly one difference between the last example and the example before it. What is this difference, how does it effect the behavior of the code, and why?

3 Example: If statement in a for loop

Save this code in a `.py` file, and try running it. What do you see?

```
MyStr = "Computer"

for i in range(len(MyStr)):
    if MyStr[i] not in ["a", "e", "i", "o", "u"]:
        print MyStr[i]
```

4 Question

Try the following commands in your IDLE:

- `"a" in ["e", "o"]`
- `"a" not in ["e", "o"]`
- `1 in range(5)`
- `8 in range(5)`

What response do you get in each case? What does the line
`if MyStr[i] not in ["a", "e", "i", "o", "u"]:`
in the Example 3 do? Describe what the Example 3 does overall.

5 Assignment

Write a program that asks the user for her favorite word then prints out the vowels in that word in the order in which they appear. Here is an example of my program for this assignment:

```
What is your favorite word? onomatopoeia
o
o
a
o
```

```
o
e
i
a
```

6 Assignment

Write a program that

1. Asks the user for an integer n
2. Flips a fair coin n times and tracks how many times it landed on heads and how many times it landed on tails.
3. Prints the results.

Here is sample output from running my code:

```
How many times would you like to flip the coin? 345
In 345 coin flips, heads came up 168 times, and tails came up 177 times.
Heads came up in 48.6956521739% of the coin flips.
```

7 Question

Use your program for Assignment 6 to fill in the table below:

# of coin tosses	% heads				
	run 1	run 2	run 3	run 4	run 5
10					
100					
1000					
10000					

For each row in this table, run your program five times and record the percent of the coin flips that were heads for each run. What do you notice about this percentage as the number of coin tosses increases? Why?

8 Assignment

Modify your Rock-Paper-Scissors game so that it first asks the user how many games of Rock-Paper-Scissors she would like to play, then plays that many games keeping a score of how many games each player won. At the end, the program should print how many games each player won, how many games were tied, and declare a winner for the tournament. Here's one run of my program:

Welcome to Rock, Paper, Scissors!
How many games of Rock, Paper, Scissors would you like to play? **3**

This is game 1 of 3.
Please select either Rock, Paper, or Scissors: **rock**
You selected rock.
I selected scissors.
You win!

This is game 2 of 3.
Please select either Rock, Paper, or Scissors: **paper**
You selected paper.
I selected scissors.
I win!

This is game 3 of 3.
Please select either Rock, Paper, or Scissors: **scissors**
You selected scissors.
I selected paper.
You win!

I won 1 games, you won 2 games, and we tied 0 games.
You won the tournament!

You may write your own messages to the user, but your program should function similar to mine.

Computing Pi

Sarah Mann

March 20, 2012

In this assignment, you will compute the value of π using a random number generator and basic arithmetic. You will write one program to do this, which you should call `pi.py`. You will write this program in a few stages and will turn in one or more of these stages before your final program is due. You will also answer some questions related to this program and turn these in as well.

1 Circles in Squares

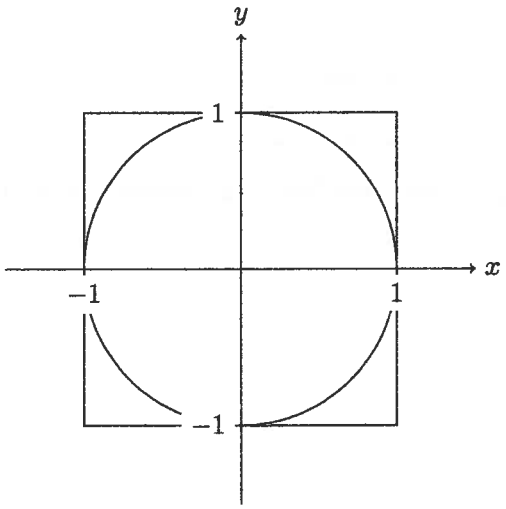


Figure 1: This figure shows a square centered on the origin of a coordinate axis. The square's corners are located at $(1, 1), (1, -1), (-1, -1)$, and $(-1, 1)$. Inside this square is a circle, also centered at the origin, with radius 1.

Question 1: What is the area of the square depicted in Figure 1?

Question 2: What is the area of the circle depicted in Figure 1?

Question 3: If you randomly chose a point inside the square in Figure 1, what is the probability that point will also be inside the circle?

2 Sketch of Program

The analysis of Figure 1 suggests a method for computing π. If we randomly choose points inside the square, then $\frac{\pi}{4}$ of those points should be inside the circle as well. Thus we can compute π in the following way:

1. Let `n` be the number of iterations to run, and `InCircle` be the number of points that were inside the circle
2. Do the following things `n` times:
   (a) Randomly select a point `(x,y)` inside the square
   (b) If `(x,y)` is also inside the circle then:
      i. Add one to `InCircle`
3. Compute `ApproxPi`, our approximation of π from `InCircle` and n
4. Compare `ApproxPi` to π

Your goal is to write a program to do this. The rest of this document discusses how to do each of these steps.

3 Selecting a point in the square

We have previously used the `random` library and `random.randrange` to generate random integers. The `random` library has some other useful functions; today we will use `random.uniform(a,b)` to generate random floats. `random.uniform(a,b)` accepts two inputs `a` and `b` and returns a randomly selected float in the range `(a,b)`. Here is an example which you should run from your interactive window:

```
>>> import random
>>> random.uniform(0,1) # try this line a few times
```

What do you see?

3.1 Program Stage 1

Write a short program that randomly chooses a point `(x,y)` inside the square in Figure 1, and prints that point.

4 Is the point inside the circle?

Question 4: Give the formula for the distance between a point (x, y) and the origin.

Question 5: Given two values `x` and `y`, how can you test (in Python!) if the point `(x,y)` is inside the circle in Figure 1. Your answer should include one line of Python code that completes this task, as well as an explanation of your code.

4.1 Program Stage 2

Building on your program for Stage 1, write a program that randomly chooses a point `(x,y)` inside the square in Figure 1, prints that point, then determines if the point is inside the circle in Figure 1 and prints an appropriate message. Here are two examples of what happens when I run my program:

```
(0.4066222280242917, 0.1357168750269946)
The point is in the circle.
```

```
(0.8176168111658206, -0.9989834077422652)
The point is not in the circle.
```

5 Do it n times

Now that we can randomly pick a point in the square and test if it is in the circle, we need to do this a lot of times and keep track of how often the point is in the circle. This will be stage 3 of your programming assignment.

5.1 Program Stage 3

Building on your program for Stage 2, write a program that:

1. Asks the user how many points she wants to choose (we'll call this number n)
2. Does the following n `n` times:
   (a) Randomly selects a point `(x,y)` inside the square
   (b) Checks if `(x,y)` is also inside the circle and counts how often this happens
3. Computes an approximation of π:
   (approximation of π) = $4\times$ (number of times the point was inside the circle)/n
4. Prints n, the number of times the point was inside the circle, and the approximation of π

6 Comparing our approximation of π and π

As the final piece to this project, we would like to know how good our approximation to π is. Recall that the `math` library contains a numerical value for π, `math.pi`. We will compare our approximation to this value. Create a new variable called `Error` by adding this line of code to your program:

```
Error = ApproxPi - math.pi
print "Error is: " + Error
```

Note: You must `include math` at the top of your program. You will need to replace `ApproxPi` with the name of your variable that stores the approximation of π.

`Error` will hold the error of your approximation. However, when we talk about the accuracy of an approximation, we often talk about how many digits the approximation is good to. For example, in one run of my `pi.py` program, I approximated π to be `ApproxPi = 3.1438`. The first few digits of π are 3.14159 so the error in my approximation was `Error = 0.00220734`. The first 2 digits after the decimal point of my approximation of π are the same as those of π, so we say that my approximation is good to two digits. Notice that the first 2 digits after the decimal in `Error` are zero. The goal now is to make our Python program identify and report to us how many digits of accuracy you have in your approximation. Here is how my program looks when its run:

```
How many points would to like to draw: 100000
100000 points drawn, 78620 of which were inside the unit circle.
ApproxPi: 3.1448
Error: 0.00320734641021
Number of Correct Digits: 2
```

We can identify the number of digits in the approximation that are correct by using base 10 logarithms.

Question 6: Define a logarithm. Do so in complete sentences *and* using mathematical formulae. What is a base 10 logarithm? What are the domain and range of the function $\log_{10} x$? (Cite your sources!)

Question 7: For each of the following values of `Error`, count how many zeros there are after the decimal, rewrite `Error` in scientific notation, calculate $\log_{10}|\text{Error}|$, then truncate $\log_{10}|\text{Error}|$ to an integer value by discarding all numbers after the decimal point. I've done the first one as an example.

Error	# of zeros after decimal	Error in Sci. Not.	$\log_{10}\lvert\text{Error}\rvert$	truncated $\log_{10}\lvert\text{Error}\rvert$
0.003207	2	3.207×10^{-3}	-2.4938	-2
0.000887				
-0.021592				
-0.093592				
0.000093				
-0.003192				

Question 8: In question 7, why did I make you calculate $\log_{10}|\text{Error}|$ and not $\log_{10}\text{Error}$?

Question 9: Consider the table in question 7. State a relationship between the number of zeros after the decimal and the exponent when you represent `Error` in scientific notation? Explain why (justify, prove) this relationship exists.

Question 10: Consider the table in question 7. State a relationship between the truncated $\log_{10}|\text{Error}|$ and the number of zeros after the decimal.

Question 11: Using known properties of logs, prove that $\log_{10} m \times 10^{-a} = (\log_{10} m) - a$. Assume that $0 < m < 10$ and $a > 0$. If you don't know any properties of logs, ask google.

Question 12: For $0 < m < 10$, what is the range of $\log_{10} m$? In other words, fill in blanks: if $0 < m < 10$ then _____$< \log_{10} m <$ _____.

Question 13: Use your answers to questions 10 and 11 to justify (prove) your answer to question 10.

Question 14: How can you calculate (in Python) the number of accurate digits in your approximation of π?

6.1 Program Stage 4

Building on your program for Stage 3, complete your `pi.py` program. For this stage, add to the end of your Stage 3 program code that

1. Calculates the error in your approximation of π
2. Calculates the number of digits that are accurate in your approximation of π.

There is an example on the previous page of how my completed program behaves. Yours should behave similarly.

Question 15: Here is an examples of output from my program:

```
ApproxPi: 3.13924
Error: -0.00235265358979
Number of Correct Digits: 2
```

Notice that my approximation of π only matches $\pi \simeq 3.14159$ at the first digit after the decimal point, yet my program says it is correct to two digits. Here is another example:

```
ApproxPi: 3.14248
Error: 0.000887346410207
Number of Correct Digits: 3
```

Here my answer matches π in the first 2 digits, but my program claims they match in the first 3. Explain what is happening in these examples. Can you justify my program's claim that the first two digits are correct in the first example and that first 3 digits are correct in the second example? If you can't justify it, can you fix it in your own program?

Question 16: How large do you need to make n to get an approximation of π that is accurate to 4 digits? When you run your program with this value of n, does it always produce an approximation of π that is accurate to 4 digits? You should run your program at least 5 times with this value for n and comment on the results.

Final Python Project

Sarah Mann

April 17, 2012

Seniors: Your assignment will be graded out of 35 points and is due Friday, April 27, 2012.
Not Seniors: Your assignment will be graded out of 100 points and is due Friday, May 18, 2012.

This document lists a number of problems which can be solved with the aide of a computer program. Your task is to solve some of them using Python. For each problem you solve, you will earn some points towards this assignment. The number of points earned for each problem is listed in the problem statement. The assignment will be graded out of 100 points; you may earn extra credit by accruing more than 100 points. For full credit on each problem you must write a Python program that

1. is your own original work.
2. is fully commented. Your comments must include
   (a) your name and the date
   (b) a statement of the problem being solved
   (c) a description of the approach you chose to solve the problem
   (d) documentation of the functionality of each portion of your code
3. works correctly on all test cases provided here.
4. work correctly on some other test cases similar to those listed here.

These problems were posed on the site `projecteuler.net`. Solutions to them can be found on the internet. Of course, use of work that is not your own is a violation of the honor code and will result in no credit for this assignment; don't look for code on the internet! You are welcome to discuss these problems with your classmates, but you may not share code.

Notes and Suggestions:

- For each problem, start by solving an easier related problem. Usually, this will mean using a small number for n or reproducing the example given.
- Generalize the procedure you used to solve the easier related problem. Write down a set of instruction (sometimes called "pseudo-code" or an algorithm) of how to solve this type of problem.

- Present your procedure to another classmate, Mr. Herzog, or Sarah. Can this third party follow your instructions to solve this problem correctly?
- Translate your procedure into Python code.
- Sometimes you may need to do things in Python that we have not yet specifically discussed. This is the nature of programming; you never know all the capabilities of a programming language and have to learn new tricks as the need arise. At the end of this document I have include a few new functions in Python that I think you might need. Mr. Herzog and Sarah can suggest other Python functions as needed.

Problem 1 (30 points)

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Write a program that asks the user for a value of n then computes the sum of all the multiples of 3 or 5 below n. Here are three examples:

- The sum of all the multiples of 3 or 5 below 10 is 23.
- The sum of all the multiples of 3 or 5 below 100 is 2318.
- The sum of all the multiples of 3 or 5 below 1500 is 524250.

Problem 2 (30 points)

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be

$$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

The sum of the even-valued terms in the Fibonacci sequence whose value does not exceed 50 is 44. Write a program that asks the user for a value of n, then computes the sum of the even-valued terms in the Fibonacci sequence whose value does not exceed n. Here are three examples:

- The sum of the even-valued terms in the Fibonacci sequence whose value does not exceed 50 is 44.
- The sum of the even-valued terms in the Fibonacci sequence whose value does not exceed 150 is 188.
- The sum of the even-valued terms in the Fibonacci sequence whose value does not exceed 15 million is 19,544,084.

Problem 3 (30 points)

The prime factors of 13,195 are 5, 7, 13, and 29. The largest of these is 29. Write a program that asks the user for a value of n, then computes the largest prime factor of n. Here are three examples:

- The largest prime factor of 13,195 is 29.
- The largest prime factor of 4,749,886 is 523.
- The largest prime factor of 155,131,173,990 is 2,381.

Problem 4 (30 points)

The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + ... + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + ... + 10)^2 = 55^2 = 3025$$

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 - 385 = 2640. Write a program that asks the user for a value of n, then computes the difference between the sum of the squares of the first n natural numbers and the square of the sum. Here are three examples:

- The difference between the sum of the squares of the first 10 natural numbers and the square of the sum is 2640.
- The difference between the sum of the squares of the first 50 natural numbers and the square of the sum is 1,582,700.
- The difference between the sum of the squares of the first 120 natural numbers and the square of the sum is 52,124,380.

Problem 5 (35 points)

By listing the first six prime numbers: 2, 3, 5, 7, 11, and 13, we can see that the 6th prime is 13. Write a program that asks the user for a value of n, then computes the nth prime number. Here are three examples:

- The 6th prime number is 13.
- The 100th prime number is 541.
- The 12,000th prime number is 128,189.

Problem 6 (40 points)

The sequence of triangle numbers is generated by adding the natural numbers. So the 7th triangle number would be $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$. The first ten triangle numbers would be:

$$1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...$$

Let us list the factors of the first seven triangle numbers:

triangle number	factors
1	1
3	1,3
6	1,2,3,6
10	1,2,5,10
15	1,3,5,15
21	1,3,7,21
28	1,2,4,7,14,28

We can see that 28 is the first triangle number to have over five divisors. Write a program that asks the user for a value of n, then computes the first triangle number to have over n divisors. Here are three examples:

- The first triangle number to have over 5 divisors is 28.
- The first triangle number to have over 10 divisors is 120.
- The first triangle number to have over 300 divisors is 2,162,160.

Problem 7 (45 points)

The following iterative sequence is defined for the set of positive integers:

$$n \rightarrow \begin{cases} \frac{n}{2}, & n \text{ even} \\ 3n+1, & n \text{ odd} \end{cases}$$

Using the rule above and starting with 13, we generate the following sequence:

$$13, 40, 20, 10, 5, 16, 8, 4, 2, 1$$

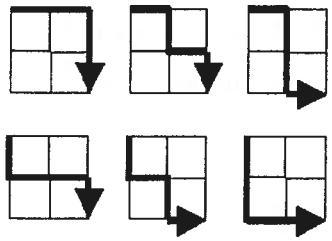
It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Write a program that asks the user for a value of n, then find the starting number less than n that produces the longest chain. NOTE: Once the chain starts the terms are allowed to go above n. Here are three examples:

- The longest sequence starting with a number less that 10 starts with 9 and has length 13.
- The longest sequence starting with a number less that 1,000 starts with 871 and has length 113.
- The longest sequence starting with a number less that 2 million starts with 1,723,519 and has length 349.

Problem 8 (45 points)

Starting in the top left corner of a 2×2 grid, there are 6 routes (without backtracking) to the bottom right corner.



Write a program that asks the user for a value of n, then finds the number of routes from the top left corner to the bottom right corner of and $n \times n$ grid without backtraking.

- There are 6 routes from the top left corner to the bottom right corner of a 2×2 grid.
- There are 184,756 routes from the top left corner to the bottom right corner of a 10×10 grid.
- There are 126,410,606,437,752 routes from the top left corner to the bottom right corner of a 25×25 grid.

Appendix: Useful Functions

This is a list of a few functions in Python you might find useful.

1. **The remainder operator** : $n \% m$ computes the remainder of n divided by m. If you type $8 \% 3$ into your IDLE, it will return 3 because the remainder of 8 divided by 3 is 2. You can test if an number n is even by checking if $n \% 2$ is zero. For example, $6 \% 2 = 0$ (since 6 is even), and $7 \% 2 = 1$ (since 7 is odd). How could you test if an number n is divisible by 3 using the % operator?

2. **The sum function**: `sum(LIST)` returns the sum of all of the items in `LIST`. For example, `sum([4,5,9])` returns 18. `sum(range(10))` returns $1+2+3+4+5+6+7+8+9 = 45$. Try these in your IDLE.

3. **floor and ceiling**: In the `math` library, the function `math.floor(x)` rounds x down to the nearest integer. For example, `math.floor(3.5)` returns 3.0. The function `math.ceil(x)` rounds x up to the nearest integer. For example, `math.ceil(3.5)` returns 4.0.