6.00 Introduction to Computer Science and Programming
Fall 2008

6.00: Introduction to Computer Science and Programming

Problem Set 1

Handed out: Tuesday, September 9, 2008.
Due: 11:59pm, Friday, September 12, 2008.

Introduction

This problem set will introduce you to using control flow in Python and formulating a computational solution to a problem. In this problem set, you will design and write a simple Python program, test it, and hand it in. ***Be sure to read this problem set thoroughly, especially the sections of Collaboration and the Handin Procedure.***

Collaboration
You may work with other students. However, each student should write up and hand in his or her assignment separately. *Be sure to indicate with whom you have worked.* For further detail, please review the collaboration policy as stated in the syllabus.

Computing Prime Numbers

A common type of computation is the *generate-and-test* method, in which one systematically generates potential solutions to a problem, and then applies a sequence of one or more tests to determine if the proposed solution is in fact valid. While one could in principle (and under some circumstances one must) generate potential solutions randomly or according to some probability distribution, often it is more efficient to devise a systematic method for generating all candidate solutions.

Problem 1.

Write a program that computes and prints the 1000^{th} prime number.

Hints:

To help you get started, here is a rough outline of the stages you should probably follow in writing your code:

1. Initialize some state variables
2. Generate all (odd) integers > 1 as candidates to be prime
3. For each candidate integer, test whether it is prime
    1. One easy way to do this is to test whether any other integer > 1 evenly divides the candidate with 0 remainder. To do this, you can use modular arithmetic, for example, the expression a%b returns the remainder after dividing the integer a by the integer b.
    2. You might think about which integers you need to check as divisors – certainly you don't need to go beyond the candidate you are checking, but

how much sooner can you stop checking?

4. If the candidate is prime, print out some information so you know where you are in the computation, and update the state variables

5. Stop when you reach some appropriate end condition. In formulating this condition, don't forget that your program did not generate the first prime (2).

Use these ideas to guide the creation of your code.

If you want to check that your code is correctly finding primes, you can find a list of primes at http://primes.utm.edu/lists/small/1000.txt.

The Product of the Primes

There is a cute result from number theory that states that for sufficiently large *n* the product of the primes less than *n* is less than or equal to *e****n* and that as *n* grows, this becomes a tight bound (that is, the ratio of the product of the primes to *e****n* gets close to 1 as *n* grows).

Computing a product of a large number of prime numbers can result in a very large number, which can potentially cause problems with our computation. (We will be talking about how computers deal with numbers a bit later in the term.) So we can convert the product of a set of primes into a sum of the logarithms of the primes by applying logarithms to both parts of this conjecture. In this case, the conjecture above reduces to the claim that the sum of the logarithms of all the primes less than *n* is less than *n*, and that as *n* grows, the ratio of this sum to *n* gets close to 1.

To compute a logarithm, we can use a built in mathematical functions from Python. To have access to these functions, you need to get them into your environment, which you can do by including the

```
from math import *
```

statement at the beginning of your file. This will allow you to use the function `log` in your code, e.g. `log(2)` will return the logarithm base e of the number 2.

Problem 2.

Write a program that computes the sum of the logarithms of all the primes from 2 to some number `n`, and print out the sum of the logs of the primes, the number `n`, and the ratio of these two quantities. Test this for different values of `n`.

You should be able to make only some small changes to your solution to Problem 1 to solve this problem as well.

Hints:

While you should see the ratio of the sum of the logs of the primes to the value n slowly get closer to 1, it does not approach this limit monotonically.

Hand-In Procedure

1. Save

Save your solution to Problem 1 as ps1a.py and your solution to Problem 2 as ps1b.py. *Do not ignore this step or save your file(s) with different names.*

2. Time and Collaboration Info

At the start of each file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of the people you collaborated with. For example:

```
# Problem Set 1
# Name: Jane Lee
# Collaborators: John Doe
# Time: 1:30
#
... your code goes here ...
```