Programming Problem 2: Primality testing

FEB 2 2015

Our second programming problem

List Primes

Given a positive integer N, generate all prime numbers less than or equal to N.

Example:

Input: 100

Output: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

Here is a list of Python ideas this example will lead us to…

- More control flow statements: `break`
- Using modules in Python
- The `math` module and useful functions in it
- Boolean operators: `and`, `or`, `not`
- Timing Python programs: `time` module
- Nested loops
- A second look at variable and expressions
- More on numeric data types in Python

Why do computer scientists care about prime numbers?

- Our digital life depends on the *security* of information that we send over the internet.
- Security of information is made possible by *encryption* methods.
- One of the most well known encryption methods is the *RSA algorithm* (R = Ron Rivest, S = Adi Shamir, and A = Leonard Adleman).
- The first step of the RSA algorithm is to find two *large* primes p and q and compute their product n = p*q.
- "Large" here could mean 300 digits or so.
- So *primality testing* (i.e., checking whether a given positive integer is a prime) is a computational problem that has attracted a lot of attention.

Basic Algorithmic Idea

1. Consider each integer n = 2, 3, 4, …, N.
2. Check if n is a prime and if so print it.

“Almost” Python code

```
N = int(input())

n = 2
while n <= N:
    if i is a prime number:
        print(n)
    n = n + 1
```

- This is a standard way of using a while-loop to walk through a sequence of integers.
- If we knew how to check if n is a prime, we would be done. So we should now solve the *primality testing* problem.
- Thus we have *reduced* our original problem into a “smaller” problem. This is a standard algorithmic technique in computer science.

Algorithmic Idea: Primality Testing

- Generate all "candidate" factors of n, namely

  2, 3, …, n-1
- For each generated "candidate" factor, check if n is evenly divisible by the "candidate" factor (i.e., the remainder is 0).
- If a "candidate" factor is found to be a real factor, then n is composite.
- If no "candidate" factor is found to be a real factor, then n is a prime.

Primality testing algorithm in pseudocode

1. Input n
2. For each candidateFactor = 2, 3, …, n-1:
3.   if n is evenly divisible by candidateFactor then
4.   remember that n is a composite

5. If we have detected that n is a composite
6.   output that n is a composite
7. Otherwise output that n is a prime

Python code (Version 0)

```
number = int(input("Enter a positive integer: "))

factor = 2
isPrime = True
while(factor <= number  - 1):
        if(number % factor == 0):
                isPrime = False
        factor = factor + 1

if(isPrime):
        print(number, "is prime")
else:
        print(number, "is composite")
```

Boolean Variables

- This program uses the boolean variable isPrime to remember if the input is a prime.
- Notice that you don't have to say: isPrime == True
- In general, boolean variables are quite useful for remembering situations that occurred in the program, for later reference.
- Questions:
  - What if we had not initialized isPrime to True?
  - Could we have used a boolean variable called isComposite to remember that the input is a composite, rather than a prime?