Our Second Python Program
Our second programming problem

Primality Testing

Given a positive integer (> 1), determine whether it is a prime number or not.

Examples:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>prime</td>
</tr>
<tr>
<td>2001</td>
<td>composite</td>
</tr>
<tr>
<td>987654321</td>
<td>composite</td>
</tr>
</tbody>
</table>
Algorithmic Idea

- Generate all “candidate” factors of n, namely 2, 3, ..., n-1
- For each generated “candidate” factor, check if n is evenly divisible by the 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.
Algorithm in pseudocode

1. Input n
2. For each factor = 2, 3, ..., n-1 do the following
3. if n is evenly divisible by factor 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
number = int(raw_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"
Discussing this code

- Boolean variables are quite useful for remembering situations that occurred in the program, for later reference.

- What happens if we get rid of the initialization:
  ```java
  isPrime = true
  ```

- Could we have used a boolean variable called `isComposite` instead?
The importance of primality testing

- From time to time you may hear in the news about the new largest prime
- Large primes are the basis of modern day cryptography.
- Cryptography is the mathematical and computational study of how to encode a message so that only the intended receiver can understand the message.
- Without cryptography online business (think Amazon, eBay, etc.) would not be possible.
Improving the efficiency of our program

1. A number $n$ does not have any factors larger than $n/2$, except itself.

2. We know $\sqrt{n} \times \sqrt{n} = n$. Hence, if $n$ is a factor larger than $\sqrt{n}$, then it has a factor smaller than $\sqrt{n}$ also.

This means that only factors 2, 3, ..., $\text{floor}(\sqrt{n})$ need to be considered.
Example

- Say n = 123.
- \(\sqrt{123} = 11.090536506409418\).
- So if 123 has a factor greater than 11.09, then it has factor less than 11.09.
- This means in looking at “candidate” factors, we only need to look at numbers 2, 3, ..., 11.
```python
import math
number = int(input("Enter a positive integer: "))

factor = 2
isPrime = True
factorBound = math.sqrt(number)
while(factor <= factorBound):
    if(number % factor == 0):
        isPrime = False
        factor = factor + 1

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