Application: Sieve of Eratosthenes

10 Oct 2008
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

List operations (Python)

```
myApples = [ "Fuji", "Gala", "Red Delicious" ]
```

- Test for list membership:

```
if "Fuji" in myApples:                    # True
```

- Concatenate:

```
[ 'a', 'b', 'c' ] + [ 'd', 'e' ]
```

- Repeat:

```
[ 'a', 'b', 'c' ] * 2
```

- Modify list entries (mutable):

```
myApples[1] = "Braeburn"
```

- Convert a string to a list of characters:

```
list("Hello World!")      # ['H', 'e', 'l', 'l', 'o', ...]
```

More list operations

- Delete an element of the list:

```
del myApples[1]        # [ "Fuji", "Golden Delicious"
  ]
```

- List slice (start:end):

```
myApples[0:1]      # [ "Fuji", "Gala" ]
```

- Assignment is aliasing:

```
yourApples = myApples      # points to same array
```

- Use a whole-list slice to copy a list:

```
yourApples = myApples[:]
    # [:] is shorthand for [0:-1] or
      [0:len(myApples)-1]
```

Quiz 03: 10 minutes, 20 points

- Define recursion in your own words.
  - Write a short example in Python to illustrate.
    - (It doesn't have to do anything useful.)
- What is the call stack used for?
- What are global variables and why are they bad?
- Write a Python function that returns the sum of a given list.
    - Docstring required! Comment as appropriate.
    - Error-checking or pre-conditions required

Quiz 03: answers #1-2

- What is recursion?
  - A recursive function invokes itself:

```
def countdown(n):
    if n <= 0:
        return 0
    print n,
    return countdown(n-1)
```

- What is the call stack?
  - Keeps track of which procedures are currently running. Made up of stack frames, recording local variables and parameters for each function invocation.

Quiz 03: answers #3-4

- What are global variables?
  - Accessible everywhere in the module: even inside functions defined in the module
  - Functions can modify global variables and cause unintended side-effects
- Calculate the sum of a list:

```
def sum(vec):
    """Sum the input list.
    Vec must be a list of ints or floats."""
    result = 0.0
    for term in vec:
        result += term
    return result
```

Sieve of Eratosthenes

- Problem: list all the prime numbers between 2 and some given big number.
  - You had a homework that was similar: test if a given number is prime, and list its factors
  - How did you solve that?
    - Procedure is_prime() (pseudocode):

```
Iterate for factor in 2 .. sqrt(n):
    If (n % factor == 0), then
        We've found a factor!
```

- But this is wasteful: really only need to test prime numbers for potential factors

Listing all primes

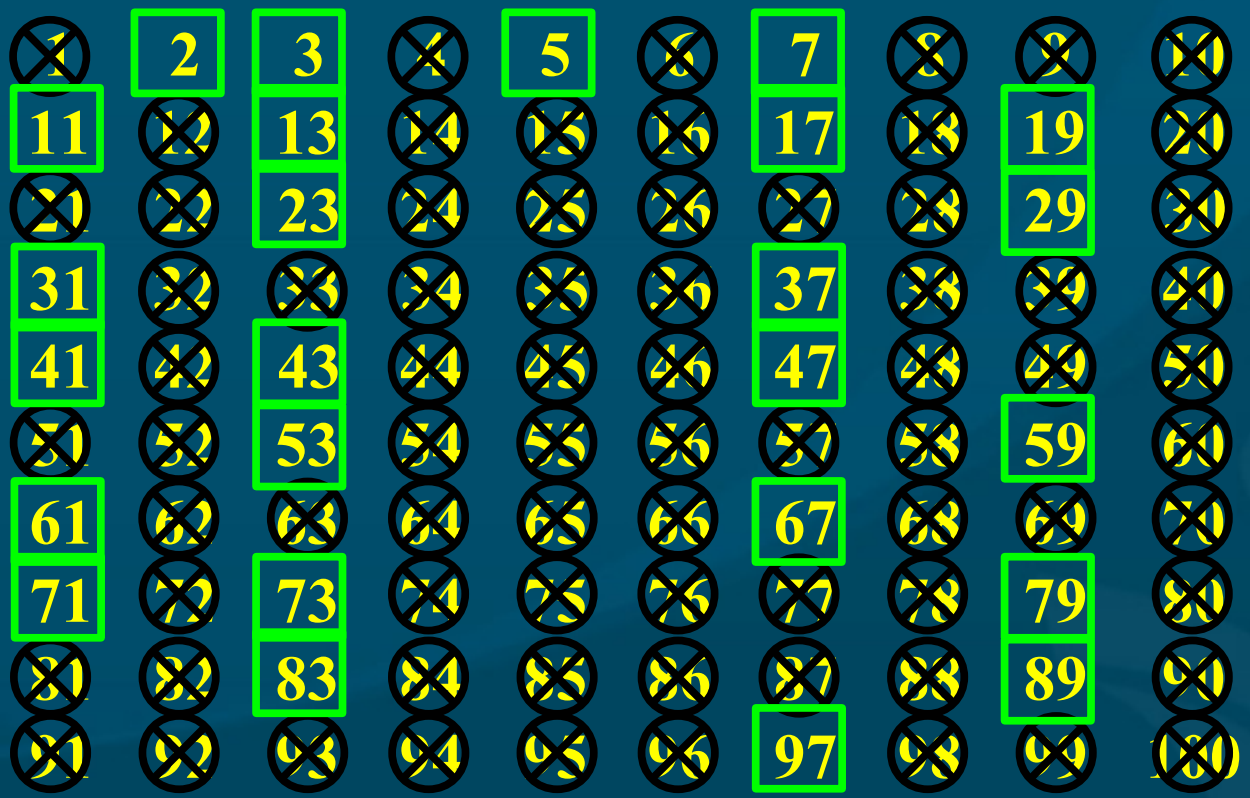
- We could tackle this problem by repeatedly calling is_prime() on every number in turn:

```
for num in range(2, max):
    if is_prime(num) ...
```

- But this could be really slow if max is big
- Is there a smarter way to eliminate non-prime (composite) numbers?

Sieve of Eratosthenes

- The sieve works by a process of elimination: we eliminate all the non-primes by turn:



Prime sieve: pseudocode

1) Create an array of booleans and set them all to true at first. (true = prime)
2) Set array element 1 to false. Now 2 is prime.
3) Set the values whose index in the array is a multiple of the last prime found to false.
4) The next index where the array holds the value true is the next prime.
5) Repeat steps 3 and 4 until the last prime found is greater than the square root of the largest number in the array.

Prime sieve: Python code

```
"""Find all primes up to a given number, using
   Eratosthenes' prime sieve."""
import math                              # sqrt
size = input("Find all primes up to: ")

# Initialize: all numbers except 0, 1 are prime
primeFlags = range(size+1)          # so pF[size] exists
for num in range(size+1):
    primeFlags[num] = True

primeFlags[0] = False
primeFlags[1] = False
```

Prime sieve: Python code (p.2)

```
# Computation: eliminate all non-primes
for num in range(2, int(math.sqrt(size))+1):
    if primeFlags[num]:            # got a prime
        # Eliminate its multiples
        for multiple in range(num**2, size+1, num):
            primeFlags[multiple] = False

# Output
print "Your primes, sir/madam:",
for num in range(2, size+1):
    if primeFlags[num]:
        print num,
```

http://twu.seanho.com/python/primesieve.py