Applications: Sieve of Eratosthenes, Recursion

14 Oct 2009
CMPT140
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

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

Recursion

- Recursion is when a function invokes itself
- Classic example: factorial (!)
  - n! = n(n-1)(n-2)(n-3) ... (3)(2)(1)
  - 0! = 1
- Compute recursively:
  - Inductive step: n! = n*(n-1)!
  - Base case: 0! = 1
- Inductive step: assume (n-1)! is calculated correctly; then we can find n!
- Base case is needed to tell us where to start

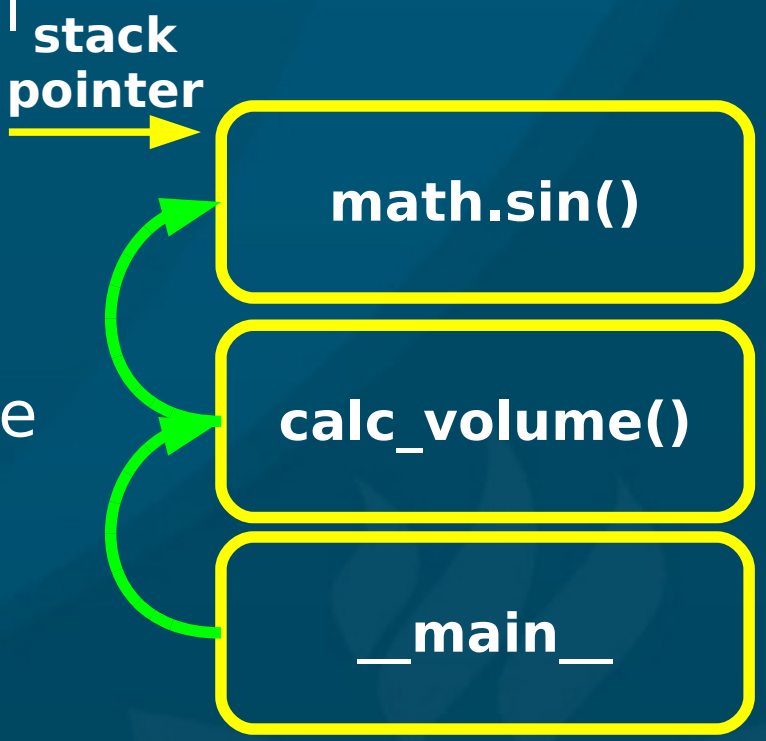
factorial() in Python

```
def factorial(n):
    """Calculate n!.  n should be a positive
      integer."""
    if n == 0:                          # base case
        return 1
    else:                               # inductive step
        return n * factorial(n-1)
```

- Progress is made each time: factorial(n-1)
- Base case prevents infinite recursion
- What about factorial(-1)? Or factorial(2.5)?

The call stack

- When a program is running, an area of memory is set aside to store local variables, the state of the program, etc.
- When a procedure is invoked, the calling context is saved, and a new chunk of memory is allocated for the procedure to use: its stack frame
- When the procedure finishes, its frame is released and control goes back to the calling context
- The stack pointer keeps track of what frame is currently running



Call stack for recursion

```
def factorial(n):
    """Compute the factorial of a
       positive integer."""
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)
```

- If there were any local variables, each frame would have its own instance of the local variables
- When an error (exception) happens, IDLE shows a backtrace: part of the call stack

Recursion example: Fibonacci

- Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34,..
  - Each number is the sum of the two previous

```
def fibonacci(n):
    """Compute the n-th Fibonnaci number.
    pre: n should be a positive integer.
    """
    if n == 0 or n == 1:                # base case
        return 1
    else:                               # inductive step
        return fibonacci(n-2) + fibonacci(n-1)
```

  - Note: very inefficient algorithm!

Computing & Society Paper

- Computing scientist as **G**odly **C**hristian **L**eader:
  - Not just knowledge about tools, but
  - Wisdom of how to use tools
    - To serve others and
    - To give glory to God
- Write a short essay on a topic of your choosing about computers and society:
    - ~ 5 pages typed double-spaced 12pt 1in margins
    - Submit half-page topic by Fri 6Nov
    - Paper due near end of semester (Wed 2Dec)
      - Electronic submission (email, eCourses)

Sample paper topics

- Censorship and free speech
  - Pornography, gambling, hate groups, etc.
- Violence in video games (Columbine etc.)
- Privacy: online banking, ID theft, etc.
- Blogs: effect on politics, social interaction, etc.
- File sharing: Napster, Gnutella, etc.
- Artificial intelligence: the nature of sentience
- Online dating (e.g. eHarmony): pros/cons
- Equity of access / rural digital divide
  - ..... or come up with your own topic!

Tips for essay writing

- Your essay should be a position paper:
  - Topic should have at least two sides (e.g. pro/con)
  - You should state (in the introductory paragraph) what your position is (thesis)
  - You should have at least 2-3 points, each, both for and against your position
    - It is not necessary to rebut every point that contradicts your position:
    - Be honest about faults/limitations of your thesis
  - Summary intro/conclusion paragraphs
  - Proper English (spelling, grammar) is important!

TODO

- HW3 due Mon
  - Ch3 and mostly Ch4
- Lab3 due next week Wed/Thu
  - Full lab write-up required!
    Use “Lab Template” on course webpage