Solution to Homework 3
Ethan Bolker
March 17, 2015

1 Python's built in documentation

Just type **help** to get help on a function or module:

```
PS C:\eb\python\hw3> python
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>> import eulerphi
eulerphi returned ['passed', 100, 'nothing useful yet']
>>> help(eulerphi)
Help on module eulerphi:

NAME
    eulerphi

FUNCTIONS
    eulerphi(b=100)
        Given a positive integer b,
        returns a list of the integers between 1 and b
        that are relatively prime to b.

        For example, eulerphi(15) returns the list
            [1,2,4,7,8,11,13,14]

        Use the function gcd from the fractions library.

FILE
    c:\eb\python\hw3\eulerphi.py


>>> help(eulerphi.eulerphi)
Help on function eulerphi in module eulerphi:

eulerphi(b=100)
    Given a positive integer b,
    returns a list of the integers between 1 and b
    that are relatively prime to b.

    For example, eulerphi(15) returns the list
        [1,2,4,7,8,11,13,14]

    Use the function gcd from the fractions library.
```

```
>>>
```

I waited for someone in the class to figure this out.

2 Eulerphi

Here's my test of the `eulerphi()` function:

```
myshell> python eulerphi.py
eulerphi(15) returned [1, 2, 4, 7, 8, 11, 13, 14]
eulerphi(16) returned [1, 3, 5, 7, 9, 11, 13, 15]
eulerphi(17) returned [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

This was pretty easy and I didn't need to learn anything new.
Here's the code:

```
import fractions

def eulerphi( b=100 ):
    """ Given a positive integer b,
    returns a list of the integers between 1 and b
    that are relatively prime to b.

    For example, eulerphi(15) returns the list
        [1,2,4,7,8,11,13,14]

    Use the function gcd from the fractions library.
    """
    relprimes = [1]
    for i in range(2,b):
        if fractions.gcd(i,b) == 1:
            relprimes.append(i)
    return relprimes

# Fake the default implicit main method that tells python
# where to start execution — the right trick for testing inside a module.
#
if __name__ == '__main__':
    print("eulerphi(15) returned " + str(eulerphi(15)))
    print("eulerphi(16) returned " + str(eulerphi(16)))
    print("eulerphi(17) returned " + str(eulerphi(17)))
```

3 Primes less than b

We did this in class — the sieve of Eratosthenes does the job. For fun I decided to do it differently for homework. The web page `primes.utm.edu/lists/small/10000.txt` lists the first 10,000 primes, 10 to a line. I found out how to read that page in Python from `http://stackoverflow.com/questions/1843422/get-webpage-contents-with-python`.

I read the web page using this Python code (borrowed from `echo2.py` from the second homework):

```
def primeslessthan(b):
    """ get the first b primes from the web page
    http://primes.utm.edu/lists/small/10000.txt
    """
    page = urllib.request.urlopen('http://primes.utm.edu/lists/small/10000.txt')
    for line in page:
        print(line)
    return
```

and saw that

```
b'                        The First 10,000 Primes\r\n'
b'                       (the 10,000th is 104,729)\r\n'
b'         For more information on primes see http://primes.utm.edu/\r\n'
b'\r\n'
b'      2      3      5      7     11     13     17     19     23     29 \r\n'
b'     31     37     41     43     47     53     59     61     67     71 \r\n'
```

So now I need to skip the first four lines of that file, strip a few characters from the beginning and the end of the next lines, and grab as many numbers as I need.

I googled "python get number from string" and found a wonderful pythonic solution at `stackoverflow.com/questions/4289331/python-extract-numbers-from-a-string`. It uses `split` to separate string into substrings separated by whitespace, then `isdigit` to find the numbers.

The code below works, when I'm connected to the internet. When the url is wrong or the internet connection fails the **try** block doesn't work as I think it should. I'm not going to spend any more time on this problem — I will use the local copy of the list of primes when I need it.

```
# Get a list of primes
#
# Ethan Bolker
# Spring 2015 for Math 480

# I've already done this with my sieve of Eratosthenes program,
# so I will do it again here by scraping a web page. There's a list of
# the first 10,000 primes, 10 to a line, at
# http://primes.utm.edu/lists/small/10000.txt

import urllib.request
import sys

def primeslessthan(b=20, local=True):
    """ get the primes from the web page
    http://primes.utm.edu/lists/small/10000.txt
    """
    if local:
        page = open("primelist.txt")
    else:
        try:
            page = urllib.request.urlopen('http://primes.utm.edu/lists/small/10000.txt')
        except:
            print("opening local file", file= sys.stderr)
            page = open("primelist.txt")
    linenumber = 0
    primes = []
    for line in page:
        linenumber += 1
        if linenumber < 5: # skip header lines
            continue
        primes += [int(s) for s in line.split() if s.isdigit()]
        if primes[-1] > b: # cool way to get last element of a list
            break
    # now back up to the last prime < b
    lastindex = -1
    while primes[lastindex] >= b:
        lastindex+= -1
    return (primes[:1+lastindex])

for b in range(3,15):
    print("primes less than " + str(b))
    print(primeslessthan(b))
```

```
print(primeslessthan(1000))
print(primeslessthan(100, False))
```

4 Faster fsf

Now I'm ready to improve the function that finds the smallest factor of its input by checking a few primes first, then looping on the numbers relatively prime to their product.

I went over the pseudocode for this in class several times. I thought it was pretty straightforward. But when I actually wrote the code it was much harder to implement than I thought. I had to find and fix several gotchas.

The code, from, `fsf.py` :

```
from eulerphi import eulerphi
from eratosthenes import eratosthenes

def fsf(number, b=10):
    """find smallest factor of number.
    First try the primes less than b,
    then loop on possible factors relatively
    prime their product.
    """
    sqrt = math.sqrt(number)
    primelist = eratosthenes(b);
    L = 1
    for p in primelist:
        if p > sqrt:
            return number
        if number%p == 0:
            return p
        L *= p
    relprimelist = eulerphi(L)
    # relprimelist begins with 1 and ends at L-1.
    # The loop shouldn't begin with 1, so I will
    # strip the 1 and append L+1
    relprimelist = relprimelist[1:] + [L + 1]
    base = 0
    while True: # potential infinite loop!
        for k in relprimelist:
            possible_factor = base + k
            if possible_factor > sqrt:
                return number
            if number%possible_factor == 0:
                return possible_factor
        base += L
    return -1 # should never get here!
```

and the code to test it, from `factor6.py` :

```
def timeall(number):
    print("find smallest factor of " + str(number))
    start = time.time()
    smallest_factor = fsf.fsf0(number)
    elapsed = time.time() - start

    print ("smallest factor: " + str(smallest_factor))
    print( "fsf0: " + str(elapsed)+ " seconds" )

    start = time.time()
    smallest_factor = fsf.fsf1(number)
    elapsed = time.time() - start
```

```
33      print("fsf1: " + str(elapsed)+ " seconds" )
34
35      for b in [4,6,8,12,14,18,20]:
36          start = time.time()
37          smallest_factor = fsf.fsf(number, b)
38          elapsed = time.time() - start
39          print(  "fsf b=" + str(b) + ": " + str(elapsed)+ " seconds" )
40
41  timeall(1000000007)
42  timeall(329841239228791)
```

The timing study turned out interesting, and worth the work. Varying the upper bound b for the list of primes to test first shows clearly that the setup overhead to build the list of small primes and call `eulerphi` grows quickly with b. It's worth paying that price only to factor large numbers.

In the data that follow you can see two experiments factoring two primes (found with Wolfram Alpha). For the one with 10 digits the fastest algorithm is the new one with $b = 6$, so testing just $[2, 3, 5]$ first. For the 15 digit prime $b = 14$ is best, testing $[2, 3, 5, 11, 13]$ first.

```
myshell> python factor6.py
find smallest factor of 1000000007
smallest factor: 1000000007
fsf0: 0.009001016616821289 seconds
fsf1: 0.003000020980834961 seconds
fsf b=4: 0.003999948501586914 seconds
fsf b=6: 0.002000093460083008 seconds
fsf b=8: 0.003000020980834961 seconds
fsf b=12: 0.00400090217590332 seconds
fsf b=14: 0.04400205612182617 seconds
fsf b=18: 0.965054988861084 seconds
fsf b=20: 21.57323384284973 seconds
find smallest factor of 329841239228791
smallest factor: 329841239228791
fsf0: 6.415367126464844 seconds
fsf1: 1.8151040077209473 seconds
fsf b=4: 2.7761590480804443 seconds
fsf b=6: 1.9321098327636719 seconds
fsf b=8: 1.5990920066833496 seconds
fsf b=12: 1.432082176208496 seconds
fsf b=14: 1.3620779514312744 seconds
fsf b=18: 2.170124053955078 seconds
fsf b=20: 22.80830407142639 seconds
```

When $b = 20$ it takes about 20 seconds just to build the list of 1.66 million numbers relatively prime to $L = 2 * 3 * 5 * 7 * 11 * 13 * 17 * 19 = 9699690$. Maybe that could be faster with an iterator.

```
Python 3.4.2
>>> L = 2*3*5*7*11*13*17*19
>>> L
9699690
>>> phiL = 1*2*4*6*10*12*16*18
>>> phiL
1658880
>>>
```

Here is the LaTeX source for this document. You can cut it from the pdf and use it to start your answers. I used the `\jobname` *macro for the source file name, so you can call your file by any name you like.*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% hw3 solution
% Math 480 Spring 2015
%
% No need to read or understand anything before the comment line below
% marked
% %%%%%%%%%%%%%%% start here %%%%%%%%%%%%%%%

\documentclass[10pt]{article}
\usepackage[textheight=10in]{geometry}

\usepackage{verbatim}
\usepackage{amsmath}
\usepackage{amsfonts} % to get \mathbb letters

\usepackage[utf8]{inputenc}
\DeclareFixedFont{\ttb}{T1}{txtt}{bx}{n}{9} % for bold
\DeclareFixedFont{\ttm}{T1}{txtt}{m}{n}{9}  % for normal
% Defining colors
\usepackage{color}
\definecolor{deepblue}{rgb}{0,0,0.5}
\definecolor{deepred}{rgb}{0.6,0,0}
\definecolor{deepgreen}{rgb}{0,0.5,0}

\usepackage{listings}

%Python style from
%http://tex.stackexchange.com/questions/199375/problem-with-listings-package-for-python-syntax-colo
\newcommand\pythonstyle{\lstset{
  language=Python,
  backgroundcolor=\color{white}, %%%%%%%
  basicstyle=\ttm,
  keywordstyle=\ttb\color{deepblue},
  emph={MyClass,__init__},
  emphstyle=\ttb\color{deepred},
  stringstyle=\color{deepgreen},
  commentstyle=\color{red},  %%%%%%%%
  frame=tb,
  showstringspaces=false,
  numbers=left,numberstyle=\tiny,numbersep =5pt
}}

\usepackage{hyperref}

\begin{document}

\pythonstyle{}

%%%%%%%%%%%%%%% start here %%%%%%%%%%%%%%%
\begin{center}
\Large{
Solution to Homework 3 \\
Ethan Bolker \\
\today
}
\end{center}
```

```
\section{Python's built in documentation}

Just type \lstinline!help! to get help on a function or module:

\begin{verbatim}
PS C:\eb\python\hw3> python
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>> import eulerphi
eulerphi returned ['passed', 100, 'nothing useful yet']
>>> help(eulerphi)
Help on module eulerphi:

NAME
    eulerphi

FUNCTIONS
    eulerphi(b=100)
        Given a positive integer b,
        returns a list of the integers between 1 and b
        that are relatively prime to b.

        For example, eulerphi(15) returns the list
            [1,2,4,7,8,11,13,14]

        Use the function gcd from the fractions library.

FILE
    c:\eb\python\hw3\eulerphi.py


>>> help(eulerphi.eulerphi)
Help on function eulerphi in module eulerphi:

eulerphi(b=100)
    Given a positive integer b,
    returns a list of the integers between 1 and b
    that are relatively prime to b.

    For example, eulerphi(15) returns the list
        [1,2,4,7,8,11,13,14]

    Use the function gcd from the fractions library.

>>>
\end{verbatim}
```

```
I waited for someone in the class to figure this out.

\section{Eulerphi}

Here's my test of the \lstinline!eulerphi()! function:

\begin{verbatim}
myshell> python eulerphi.py
eulerphi(15) returned [1, 2, 4, 7, 8, 11, 13, 14]
eulerphi(16) returned [1, 3, 5, 7, 9, 11, 13, 15]
eulerphi(17) returned [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
\end{verbatim}

This was pretty easy and I didn't need to learn anything new.

Here's the code:

\lstinputlisting{eulerphi.py}

\section{Primes less than b }

We did this in class --- the sieve of Eratosthenes does the job. For
fun I decided to do it differently for homework. The web page
\url{primes.utm.edu/lists/small/10000.txt} lists the first 10,000
primes, 10 to a line. I found out how to read that page in Python from
\url{http://stackoverflow.com/questions/1843422/get-webpage-contents-with-python}.

I read the web page using this Python code (borrowed from
\lstinline!echo2.py! from the second homework):

\lstinputlisting[firstnumber=13,firstline=13,lastline=21]{primelist0.py}

and saw that

\begin{verbatim}
b'                          The First 10,000 Primes\r\n'
b'                         (the 10,000th is 104,729)\r\n'
b'         For more information on primes see http://primes.utm.edu/\r\n'
b'\r\n'
b'      2      3      5      7     11     13     17     19     23     29 \r\n'
b'     31     37     41     43     47     53     59     61     67     71 \r\n'
\end{verbatim}

So now I need to skip the first four lines of that file, strip a few
characters from the beginning and the end of the next lines, and grab
as many numbers as I need.

I googled ``python get number from string'' and found a wonderful
pythonic solution at
\url{stackoverflow.com/questions/4289331/python-extract-numbers-from-a-string}.
It uses \lstinline!split! to separate string into substrings separated
by whitespace, then \lstinline!isdigit! to find the numbers.

The code below works, when I'm connected to the internet. When the url
is wrong or the internet connection fails the \lstinline!try! block
doesn't work as I think it should. I'm not going to spend any more
time on this problem --- I will use the local copy of the list of
primes when I need it.
```

```
\lstinputlisting{primelist1.py}

\section{Faster fsf}

Now I'm ready to improve the function that finds the smallest
factor of its input by checking a few primes first, then looping on
the numbers relatively prime to their product.

I went over the pseudocode for this in class several times. I thought
it was pretty straightforward. But when I actually wrote the code it
was much harder to implement than I thought. I had to find and fix
several gotchas.

The code, from, \lstinline!fsf.py! :

\lstinputlisting[firstnumber=55,firstline=55,lastline=87]{fsf.py}

\noindent
and the code to test it, from \lstinline!factor6.py! :

\lstinputlisting[firstnumber=21,firstline=21,lastline=42]{factor6.py}

The timing study turned out interesting, and worth the work. Varying
the upper bound $b$ for the list of primes to test first shows clearly
that the setup overhead to build the list of small primes and call
\lstinline!eulerphi! grows quickly with $b$. It's worth
paying that price only to factor large numbers.

In the data that follow you can see two experiments factoring two
primes (found with Wolfram Alpha). For the one with 10 digits the
fastest algorithm is the new one with $b=6$, so testing just
$[2,3,5]$ first. For the 15 digit prime $b=14$ is best, testing
$[2,3,5,11,13]$ first.


\begin{verbatim}
myshell> python factor6.py
find smallest factor of 1000000007
smallest factor: 1000000007
fsf0: 0.009001016616821289 seconds
fsf1: 0.003000020980834961 seconds
fsf b=4: 0.003999948501586914 seconds
fsf b=6: 0.002000093460083008 seconds
fsf b=8: 0.003000020980834961 seconds
fsf b=12: 0.00400090217590332 seconds
fsf b=14: 0.04400205612182617 seconds
fsf b=18: 0.965054988861084 seconds
fsf b=20: 21.57323384284973 seconds
find smallest factor of 329841239228791
smallest factor: 329841239228791
fsf0: 6.415367126464844 seconds
fsf1: 1.8151040077209473 seconds
fsf b=4: 2.7761590480804443 seconds
fsf b=6: 1.9321098327636719 seconds
fsf b=8: 1.5990920066833496 seconds
fsf b=12: 1.432082176208496 seconds
fsf b=14: 1.3620779514312744 seconds
fsf b=18: 2.170124053955078 seconds
fsf b=20: 22.80830407142639 seconds
\end{verbatim}
```

```
When $b=20$ it takes about 20 seconds just to build the list of
1.66 million numbers relatively prime to $L = 2*3*5*7*11*13*17*19 =
9699690$. Maybe that could be faster with an iterator.

\begin{verbatim}
Python 3.4.2
>>> L = 2*3*5*7*11*13*17*19
>>> L
9699690
>>> phiL = 1*2*4*6*10*12*16*18
>>> phiL
1658880
>>>
\end{verbatim}
\newpage
\emph{
Here is the \LaTeX{} source for this document. You can cut it from the
pdf and use it to start your answers. I used the} \verb!\jobname!
\emph{macro for the source file name, so you can call your file by any
  name you like.}
\verbatiminput{\jobname}

\end{document}
```