Prime Numbers

```
# Return list of primes < n

def primes_under(n):
    primes = [2]           # create list of 1 item
    for i in range(3, n):   # i = 3, 4, ..., n-1
        for p in primes:    # p = 2, 3, ...
            if i%p == 0:    # if it has a divisor
                break       # stop inner loop
            if p*p > i:     # if we're through
             primes.append(i) # add i to the list
                break       # and stop
    return primes

print primes_under(100)
```


Find Symbolic Links

```
import os # Built-in O/S interface

def findlinks(dirname):
    links = []
    names = os.listdir(dirname)
    names.sort()
    for name in names:
        if name in ('.', '..'): continue
        fullname = dirname + '/' + name
        if islink(fullname):
            links.append(fullname)
        elif isdir(fullname):
            links = links + findlinks(fullname)
    return links
```


Find Symbolic Links (cont'd)

```
from stat import S_ISLNK, S_ISDIR, ST_MODE

def islink(path):
    try:
        st = os.lstat(path)
    except (os.error, AttributeError):
        return 0
    return S_ISLNK(st[ST_MODE])

def isdir(path):
    try:
        st = os.stat(path)
    except os.error:
        return 0
    return S_ISDIR(st[ST_MODE])
```


A Date Class

```
import time
DAY = 24*3600
class Date:
  def __init__(self, date):
      # Represent a date as seconds since epoch
      self.date = date
  def __repr__(self):
      # Print a date as "Fri Oct 21 1994"
      s = time.ctime(self.date)
      return s[:11] + s[-4:]
  def tomorrow(self): return self + 1
  def yesterday(self): return self - 1
```

A Date Class (cont'd)

```
    def __add__(self, ndays):
        return Date(self.date + ndays*DAY)

    __radd__ = __add__

    def __sub__(self, other):
        if hasattr(other, 'date'):
            # Assume other is a date too
            return int(self.date/DAY) - \
                    int(other.date/DAY)
        else:
            # Assume other is a number of days
            return self.__add__(-other)

today = Date(time.time())
print today.tomorrow() - today.yesterday()
```


Interfacing to Tk

```
from Tkinter import *
import string

class App(Frame):

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.e = Entry()
        self.e.pack()
      self.e.bind('<Key-Return>', self.callback)

    def callback(self, event):
        print self.entrythingy.get()

root = App()
root.master.title("Foo")
root.mainloop()
```


How Fast Is It?

Run this on an idle machine:

```
import time

def test(n):
    t1 = time.time()
    while n > 0:
        n = n-1
    t2 = time.time()
    return t2-t1

test(100000)
```

- ~2 seconds on an SGI Indigo or Sparc 10
- ~14 seconds on a Power Mac 8100 or Mac IIfx
- ~32 seconds on a Macintosh PowerBook 170

Bytecode Spaghetti

```
import dis
dis.disco(test.func_code)

     0 SET_LINENO          2
     3 RESERVE_FAST        3 (('n', 't1', 't2'))
     6 UNPACK_ARG          1
     9 STORE_FAST          0

    12 SET_LINENO          3
    15 LOAD_GLOBAL         1 (time)
    18 LOAD_ATTR           1 (time)
    21 BUILD_TUPLE         0
    24 BINARY_CALL
    25 STORE_FAST          1
```

```
    28 SET_LINENO          4
    31 SETUP_LOOP         34 (to 68)

>>  34 SET_LINENO          4
    37 LOAD_FAST           0
    40 LOAD_CONST          1 (0)
    43 COMPARE_OP          4
    46 JUMP_IF_FALSE      17 (to 66)
    49 POP_TOP

    50 SET_LINENO          5
    53 LOAD_FAST           0
    56 LOAD_CONST          2 (1)
    59 BINARY_SUBTRACT
    60 STORE_FAST          0
    63 JUMP_ABSOLUTE      34
>>  66 POP_TOP
    67 POP_BLOCK
```



```
>>  68 SET_LINENO          6
    71 LOAD_GLOBAL         1 (time)
    74 LOAD_ATTR           1 (time)
    77 BUILD_TUPLE         0
    80 BINARY_CALL
    81 STORE_FAST          2

    84 SET_LINENO          7
    87 LOAD_FAST           2
    90 LOAD_FAST           1
    93 BINARY_SUBTRACT
    94 RETURN_VALUE
    95 LOAD_CONST          0 (None)
    98 RETURN_VALUE
```