(https://www.edx.org)

MITx: 6.00.1x Introduction to Computer Science and Programming Using Python

sandipan_dey (/dashboard) ▾

Courseware (/courses/MITx/6.00.1_4x/3T2014/courseware) Updates & News (/courses/MITx/6.00.1_4x/3T2014/info)

Calendar (/courses/MITx/6.00.1_4x/3T2014/89309559b0414f6d8cbef9e48ca19f4b/) Wiki (/courses/MITx/6.00.1_4x/3T2014/course_wiki)

Discussion (/courses/MITx/6.00.1_4x/3T2014/discussion/forum) Progress (/courses/MITx/6.00.1_4x/3T2014/progress)

L12 PROBLEM 6 (5/5 points)

This problem will ask a series of questions about generators.

1. Thinking about the `genPrimes` generator from the last problem, which of the following can be done only by using a generator, instead of defining a function (that uses any type of construct we've learned about, except generators)?

   ○ Return 1000000 prime numbers

   ○ Print every 10th prime number, until you've printed 20 of them

   ○ Keep printing the prime number until the user stops the program

   ● Everything that can be done with generator can be done with a function ✓

EXPLANATION:

We could write a function that does any of the choices. However a generator is nice because we can ask the generator for the next item, one at a time, and we don't waste time computing values that we don't ultimately want (or won't want for a long time).

Here are some examples of how one might code a function for each of the above options without a generator:

```
def genPrimesFn():
    '''Function to return 1000000 prime numbers'''
    primes = []   # primes generated so far
    last = 1      # last number tried
    while len(primes) < 1000000:
        last += 1
        for p in primes:
            if last % p == 0:
                break
        else:
            primes.append(last)
    return primes

def genPrimesFn():
    '''Function to print every 10th prime
    number, until you've printed 20 of them.'''
    primes = []   # primes generated so far
    last = 1      # last number tried
    counter = 1
    while True:
        last += 1
        for p in primes:
            if last % p == 0:
                break
        else:
            primes.append(last)
            counter += 1
            if counter % 10 == 0:
                # Print every 10th prime
                print last
            if counter % (20*10) == 0:
                # Quit when we've printed the 10th prime 20 times (ie we've
                # printed the 200th prime)
                return



def genPrimesFn():
    '''Function to keep printing the prime number until the user stops the program.
    This way uses user input; you can also just run an infinite loop (while True)
    that the user can quit out of by hitting control-c'''
    primes = []   # primes generated so far
    last = 1      # last number tried
    uinp = 'y'    # Assume we want to at least print the first prime...
    while uinp != 'n':
        last += 1
        for p in primes:
            if last % p == 0:
                break
            else:
                primes.append(last)
                print last
                uinp = raw_input("Print the next prime? [y/n] ")
                while uinp != 'y' and uinp != 'n':
                    while uinp
                    print "Sorry, I did not understand your input. Please enter 'y' for yes, or 'n' for
no."
                    uinp = raw_input("Print the next prime? [y/n] ")
                if uinp == 'n':
                    break
```

2. Every procedure that has a `yield` statement is a generator.

● True ✓

○ False

EXPLANATION:

See http://docs.python.org/release/2.3.5/ref/yield.html (http://docs.python.org/release/2.3.5/ref/yield.html). The Python documentation is always your friend!

3. If a procedure has only one `yield` statement, but that statement will never be executed, then the procedure is not a generator.

   ○ True
   ● False ✓

EXPLANATION:

Examine the following code; play around with it in Python.

```
def generator1():
    if True:
        yield 1

def generator2():
    if False:
        yield 1

g1 = generator1()
g2 = generator2()

print type(g1)
print type(g2)
print g1.next()
print g2.next()
```

4. Suppose we wanted to iterate over a million numbers using a 'for/in' loop. If we use the code `for x in range(1000000)`, how many numbers do we need to store in memory at once?

   ○ 1
   ○ 2
   ○ 1000
   ● 1000000 ✓
   ○ Don't need to store anything in memory

EXPLANATION:

We need to store 1000000 numbers. This for loop first makes a list of 1000000 integers, *then* iterates over that list. So the entire list of 1000000 numbers must be saved in memory until the iteration has completed.

5. If we were to use a generator to iterate over a million numbers, how many numbers do we need to store in memory at once?

   ○ 1
   ● 2 ✓
   ○ 1000
   ○ 1000000
   ○ Don't need to store anything in memory

EXPLANATION:

We need to store 2 numbers - one for the current value, and one for the max value.

```
def genOneMillion():
    maxNum = 1000000
    current = -1
    while current < maxNum:
        current += 1
        yield current
```

Python actually provides this! The `xrange` function, while not really a generator, has the same benefits of using a generator. You can substitue `xrange` any place in your code that uses `range`. It behaves the same way, but stores much less information in memory so can cause your code to execute somewhat faster.

For the following tasks, would it be best to use a generator, a standard function, or either?

1. Finding the nth Fibonacci number

   ○ Generator
   ● Standard function ✓
   ○ Either a generator or standard function is fine

2. Printing out an unbounded sequence of Fibonacci numbers

   ● Generator ✓
   ○ Standard function
   ○ Either a generator or standard function is fine

3. Printing out a bounded sequence of prime numbers, where the prime numbers are successively computed by division by smaller primes

   ○ Generator
   ○ Standard function
   ● Either a generator or standard function is fine ✓

4. Printing out an unbounded sequence of prime numbers, where the prime numbers are successively computed by division by smaller primes

   ● Generator ✓
   ○ Standard function
   ○ Either a generator or standard function is fine

5. Finding the score of a word from the 6.00x Word Game of Pset 4

   ○ Generator
   ● Standard function ✓
   ○ Either a generator or standard function is fine

6. Iterating over a sequence of numbers in a random order, where no number is repeated

   ○ Generator
   ● Standard function ✓
   ○ Either a generator or standard function is fine

Check | Hide Answer

Help

Show Discussion

New Post

EdX is a non-profit created by founding partners Harvard and MIT whose mission is to bring the best of higher education to students of all ages anywhere in the world, wherever there is Internet access. EdX's free online MOOCs are interactive and subjects include computer science, public health, and artificial intelligence.