CISS451/MATH451: Cryptography and Computer Security
Assignment 6

Python programming

Here's a quick-and-dirty tutorial on Python programming. This is not a complete introduction to Python programming. I'll only talk enough to do basic computations, especially relevant to number crunching. (By the way, Python is used in scientific computations in research and in the industry. However I will go into special libraries which are optimized for speed. If you want to find out more, you can google for scipy and numpy.)

Open a text editor (example: xemacs) and edit a file say `x.py` and write this:

```
# my first program
print "hello world"
```

Now run the program from the shell with this command

```
python x.py
```

and you will get this response:

```
hello world
```

Now modify `x.py`:

```
print "hello world"

x = 1
print "x =", x

# here's a for-loop
for i in [5,6,7,8,9,10]:
    print ">>>>> ....."
    print "i =", i, i + 2, i - 2, i * 2, i % 2, i / 2
    print "..... <<<<<"
    print

# here's a for-loop that does the same thing
for i in range(5, 11):
    print ">>>>> ....."
    print "i =", i, i + 2, i - 2, i * 2, i % 2, i / 2
    print "..... <<<<<"
    print

# here a while loop
i = 0
while i < 10:
    print "i =", i
    print "incrementing i ..."
    i += 1

# you can assign multiple variables
a, b, c = 1, 2, 3
print a, b, c
```

As you can see most of the operators are what you have seen before in C++/Java/etc. No surprises here. Likewise you also have `<, <=, >, >=, ==,` =!.

Here's how you do functions:

```
# here's a function
def f(a, b, c):
    z = a + b + c
    return z

# here's calling the above function
print f(1, 2, 3)

# you can return multiple values
def g(a, b, c):
    return a + b, a * c, b / c

# ... and of course receive multiple values
i, j, k = g(1, 2, 3)
print i, j, k
```

You also have arrays:

```
# you have python lists which are like arrays
x = []
```

```
print x
x = [42, 43, 44]
print x
x.append(100)
print x
del x[1]
print x
print 42 in x
print 45 in x

x = [5, 6, -5, 8, 2, -10, 11, 9, 13, -1]
for y in x:
    print y
print x[0]
print x[2]
print x[-1] # the last element
print x[-2] # the second last element
x.sort()
print x
print x[0]  # smallest
print x[-1] # largest

y = x[2:5] # a slice of x
print y

# be careful when doing = on lists ...
x = [1,2,3,4,5]
y = x          # y and x references the same value
x[0] = 100
print x
print y # y references the *same* value as x

# if you want a *copy*, you do this:
import copy
x = [1,2,3,4,5]
y = copy.deepcopy(x)
x[0] = 100
print x
print y
```

Here's a simple function that computes divides:

```
def div(a):
    xs = []
    for d in range(1, a + 1):
        if a % d == 0:
            xs.append(d)
    return xs

print div(100)
```

Here's one that compute common divisors:

```
def common_div(a, b):
    div_a = div(a)
    div_b = div(b)
    xs = []
    for d in div_a:
        if d in div_b:
            xs.append(x)
    return xs

print common_div(100, 30)
```

Here's one that computes gcd is an extremely inefficient way:

```
def slow_gcd(a, b):
    xs = common_div(a, b)
    xs.sort()
    return xs[-1]

print slow_gcd(100, 30)
```

To time your code you can do this:

```
import time
start = time.clock()
g = slow_gcd(100000000, 200000)
end = time.clock()
print "time taken:", end - start
```

Here's a self–exercise:

A positive integer n if said to be *perfect* if it is the sum of the divisors of n less than n. For instance 6 is a perfect number since the divisors of 6 which are less than 6 are

$$1, 2, 3$$

and the sum is

$$1 + 2 + 3 = 6$$

Write a program that prompts the user for n and prints all perfect numbers up to n.

(You will discover the first 4 perfect numbers very quickly. The fifth one takes a while.)

```
def isperfect(n):
    sum = 0
    # CODE TO COMPUTE THE SUM OF DIVISORS OF n WHICH ARE < n
    return sum == n

n = input("enter n: ")
for i in range(1, n + 1):
    if i % 100000 == 0:
        print "testing", i
    if isperfect(n):
        print "***** FOUND", i
```

The answer is on the next page ...

SPOILERS!!!

SPOILERS!!!

SPOILERS!!!

```
def isperfect(n):
    sum = 0
    for x in div(n):
        if x < n:
            sum += x
    return sum == n
```

Of course building a python list is a waste of resources. You might as well just compute the divisors and add. So here's another exercise on the improved `isperfect` function.

```
def isperfect(n):
    sum = 0
    # YOUR CODE TO COMPUTE THE SUM OF DIVISORS DIRECTLY WITH
    # USING THE div FUNCTION.
    return sum == n
```

Time it and see if you can improve the performance.

Can you further improve it? [HINT: if $1 < d < n$ is a divisor of n, then you have already found another which mimght be distinct from d.]

Also, since you're scanning a range of values for perfect numbers, some computations actually repeat. For instance if you already know all divisors of 10, how would you compute the divisors of 30?

With some improvements, you should be able to compute the 5th perfect number and beat the ancient greek mathematicians.

Q1. This is a freebie ... just to check that you understood the above information on python programming ...

Implement the extended Euclidean algorithm so that given a, b, it computes x, y such that

$$ax + by = \gcd(a, b)$$

(See the notes.)

(a) Compute the gcd of of the following two numbers:

392071526476 538733716422

For this question, modify your code so that is prints the values of variables c, d, c', d', r, r', including the initiazation and the changed within the loop. The following is the output for $a = 100, b = 30$.

```
1 ... 1 0 0 1 100 30
2 ... 3 0 1 1 -3 30 10
3 ... 3 1 -3 -3 10 10 0
gcd = 10
x = 1
y = -3
```

Note the line number in the output. It must be present. The above example, the initialization is labeled line 1. The loop executes twice. You must make sure your output matches the output below. Here's another

```
1 ... 1 0 0 1 123 57
2 ... 2 0 1 1 -2 57 9
3 ... 6 1 -2 -6 13 9 3
4 ... 3 -6 13 19 -41 3 0
gcd = 3
x = -6
y = 13
```

(Can you tell what is a and b?) In this case, line 1 is the initialization and the loop executes three times.

(To check for correctness, I will only look at *one* line. That's why I need the line numbers)

It's a good idea to compare the above computation of the gcd of 392071526476 and 538733716422 against the `slow_gcd`.

(b) Do the same for

392071526473 538733716422

SOLUTION.

```
REPLACE THIS WITH YOUR PROGRAM'S OUTPUT
```

Q2. Write a python function that compute the inverse of a mod n if the inverse exists. Your code should look like this:

```
def inv(a, n):
    # call eea with gcd,x,y = eea(....., .....)
    if gcd == 1:
        return 1 # CORRECT THIS
    else:
        return None
```

(a) Does 392071526473 have a multiplicative inverse mod 8387337716424? If so, what is the inverse? (Write it in the range 0, ..., 838733716423.)

(a) Does 392071526473 have a multiplicative inverse mod n = 838733716424? If so, what is the inverse? (Write it in the range $0, ..., n-1$.)

(b) Does 392071526473 have a multiplicative inverse mod n = 8387337164349231 23? If so, what is the inverse? (Write it in the range $0, ..., n-1$.)

SOLUTION.