Science One CS 2015/2016: Solutions to Term 1 Exam

Question 1. **(30 points)**

Determine and **clearly** indicate the output (exactly what is printed to the screen) of each of the following code snippets when executed with Python 2.7 (the version we have been using in this course), or write "ERROR" if executing the code would cause an error. Be careful – some of these questions are meant to be tricky.

1 pt (a)
```
print 5+2
```

Solution: 7

1 pt (b)
```
print 5+2.0
```

Solution: 7.0

1 pt (c)
```
print 5/2
```

Solution: 2

1 pt (d)
```
print '5'+'2'
```

Solution: 52

1 pt (e)
```
print 1.0+2.0==3.0
```

Solution: True

1 pt (f)
```
print 0.1+0.2==0.3
```

Solution: False

1 pt (g)
```
print 1.0+2.0=3.0
```

Solution: ERROR

2 pts (h)
```
if 2 > 3 or 3 > 2:
    print 'Hello'
else:
    print 'Goodbye'
```

Solution: Hello

2 pts (i)
```
x = 2
if x < 5 and x > 8:
    x = x + 1
```

```
if x > 0:
    x = x + 1
print x
```

Solution: 3

2 pts (j)

```
x = 2
if x < 5 and x > 8:
    x = x + 1
    if x > 0:
        x = x + 1
print x
```

Solution: 2

3 pts (k)

```
x = 2
y = x
x = 3
print y
```

Solution: 2

3 pts (l)

```
def hello(a):
    return a + 5

a = 1
a = hello(a)
a = hello(a)
print a
```

Solution: 11

1 pt (m)

```
def hello(a):
    a = a + 5
    return a

a = 1
hello(a)
print a
```

Solution: 1

4 pts (n)
```
import numpy as np
N = 5
x = np.zeros(N)
x[0] = 1
x[1] = 1
n = 2
while n < N:
    x[n] = x[n-1] + x[n-2]
    n = n + 1
print x[N-1]
```

Solution: 5

1 pt (o)
```
import numpy as np
x = np.zeros(5)
print x[5]
```

Solution: ERROR

2 pts (p)
```
x = 0
for j in range(4):
    x = x + j
print x
```

Solution: 6

3 pts (q)
```
x = 0
for i in range(20):
    for j in range(50):
        x = x + 1
print x
```

Solution: 1000

Question 2. **(4 points)**

2 pts (a) What is the binary representation of the base-10 number 20?

Solution: 10100

2 pts (b) What is the base-10 representation of the binary number 1000011?

Solution: 67

Question 3. **(20 points)**

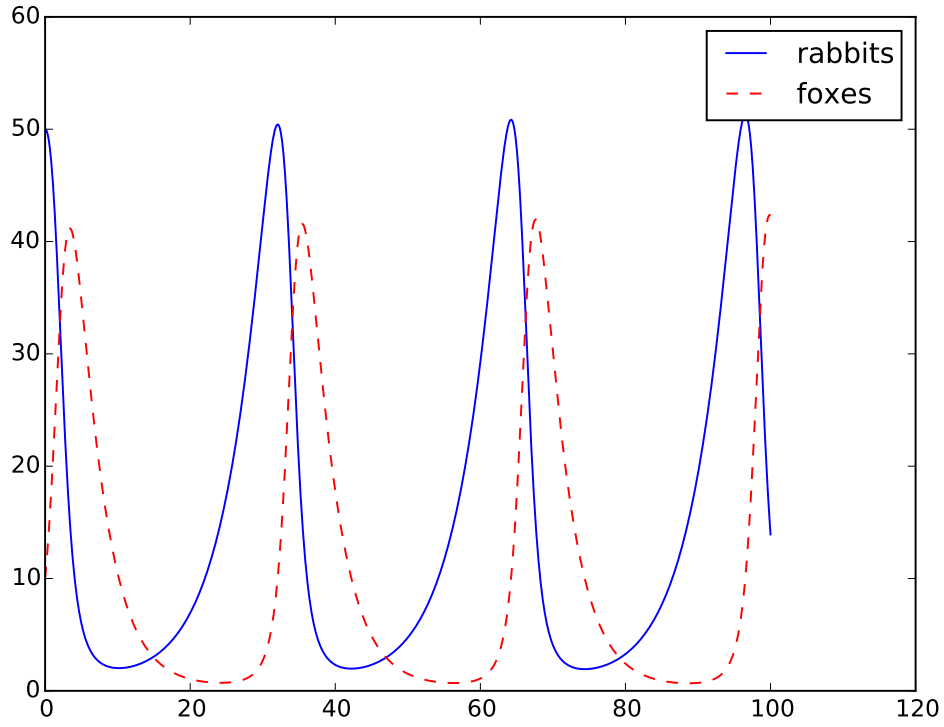
Let x be the number of rabbits and y be the number of foxes in a forest. We will model the interactions between these populations with the following differential equations:

$$\frac{dx}{dt} = 0.2x - 0.02xy$$
$$\frac{dy}{dt} = -0.3y + 0.02xy$$

Just like in the biology and physics tutorials, we can write down discrete versions of these equations using the Euler method with a finite time step Δt:

$$x(t + \Delta t) = x(t) + [0.2x(t) - 0.02x(t)y(t)]\,\Delta t$$
$$y(t + \Delta t) = y(t) + [-0.3y(t) + 0.02x(t)y(t)]\,\Delta t$$

In the space provided on the next page, write a program that simulates how these populations change over time. The first and last few lines of the program are already provided for you; to get full marks, you must make use of the provided code. Below is an example of what the output looks like for $N = 10000$, $\Delta t = 0.01$, $x(0) = 50$, and $y(0) = 10$.



Solution:

```
import sys
import numpy as np
import matplotlib.pyplot as plt

N =  int(sys.argv[1])
dt = float(sys.argv[2])
x0 = int(sys.argv[3])    # initial number of rabbits, x(0)
y0 = int(sys.argv[4])    # initial number of foxes, y(0)

x = np.zeros(N+1)
y = np.zeros(N+1)
t = np.zeros(N+1)
x[0] = x0
y[0] = y0

for n in range(N):
    x[n+1] = x[n] + (0.2*x[n]-0.02*x[n]*y[n])*dt
    y[n+1] = y[n] + (-0.3*y[n]+0.02*x[n]*y[n])*dt
    t[n+1] = t[n] + dt

plt.plot(t, x, 'b')
plt.plot(t, y, '--r')
plt.legend(('rabbits', 'foxes'))
plt.savefig('population.pdf')
```

Question 4. **(15 points)**

Write the `simulate` function from Assignment 2. As a reminder, this function takes as arguments S_0, σ, and T and returns an array of size $T+1$ containing the stock price at each time step. To generate the stock prices, the function simulates the geometric Brownian motion for T steps using the update rule

$$S_t = S_{t-1} \exp\left(-0.5\sigma^2 + \sigma Z\right) ,$$

where Z is a random number drawn from a Gaussian distribution, which you can generate using `numpy.random.randn()`. You can assume `numpy` has already been imported somewhere above your function (and you can refer to it as either `numpy` or `np`). You do not need to write comments. As a reminder, $\exp(x)$ means e^x and can be computed with `numpy.exp(x)`.

Solution:

```
def simulate(initial_price, sigma, T):
    price = np.zeros(T+1)
    price[0] = initial_price
    for t in range(T):
        Z = np.random.randn()
        price[t+1] = price[t]*np.exp(-0.5*sigma**2 + Z*sigma)
    return price
```

Question 5. **(20 points)**

Alice, your collaborator in the Earth, Ocean and Atmospheric Sciences Department at UBC, has provided you with a Python function `simulate` that simulates the temperature in Vancouver based on certain parameters. You are not given the code for this function, but you can see the documentation (comment statement) that Alice wrote:

```
# Predicts the future temperatures in Vancouver using a climate simulation.
# Note: the simulation is random and will give different results each time.
# Inputs: carbon_dioxide (float), the carbon dioxide emission levels
#         green_seats    (int),   number of Green Party seats in parliament
#         initial_temp   (float), the initial temperature
#         num_years      (int),   number of years to simulate
# Output: the simulated temperature over time (array of size num_years+1)
```

Your job is to complete the partially written program on the next page so that it repeatedly calls the `simulate` function and computes the expected *change* in temperature after N years, averaged over the specified number of trials.

We will assume that Alice's function is contained in the `globalwarming` library. Therefore, the top of the program on the next page contains the statement `import globalwarming`. To call the function, use `globalwarming.simulate`. This is just like what you have done in the past, for example when you type `import numpy` and then use `numpy.exp` to access the `exp` function within the `numpy` library.

Here are some example runs showing what your program should print out. (You do not need to understand where the numbers come from, since you cannot see the `simulate` function.) Note that the output specifies whether the temperature increases or decreases; your program should do this too, depending on whether the final temperature is greater than or less than the initial temperature. You can ignore the case of the temperature staying exactly the same.

```
>> python VancouverTemp.py 0.1 1 20 10 1000
The temperature will INCREASE by 0.2 degrees Celsius.

>> python VancouverTemp.py 0.1 300 20 10 1000
The temperature will DECREASE by 25.0 degrees Celsius.
```

question continues on next page

Solution:

```
import sys
import numpy as np    # in case you want to use numpy
import globalwarming # gives us access to globalwarming.simulate

# read in the command-line arguments
CO2     = float(sys.argv[1]) # the carbon dioxide emission levels
green   = int(sys.argv[2])   # the number of Green Party seats
T_0     = float(sys.argv[3]) # the initial temperature
N       = int(sys.argv[4])   # the number of years
trials  = int(sys.argv[5])   # the number of trials

################################
#### WRITE YOUR CODE BELOW ####
################################
average_temp = 0.0
for i in range(trials):
    average_temp += globalwarming.simulate(CO2, green, T_0, N)[N]
average_temp /= trials

if average_temp > T_0:
    print 'The temperature will INCREASE by: '+str(average_temp-T_0)
else:
    print 'The temperature will DECREASE by: '+str(T_0-average_temp)
```

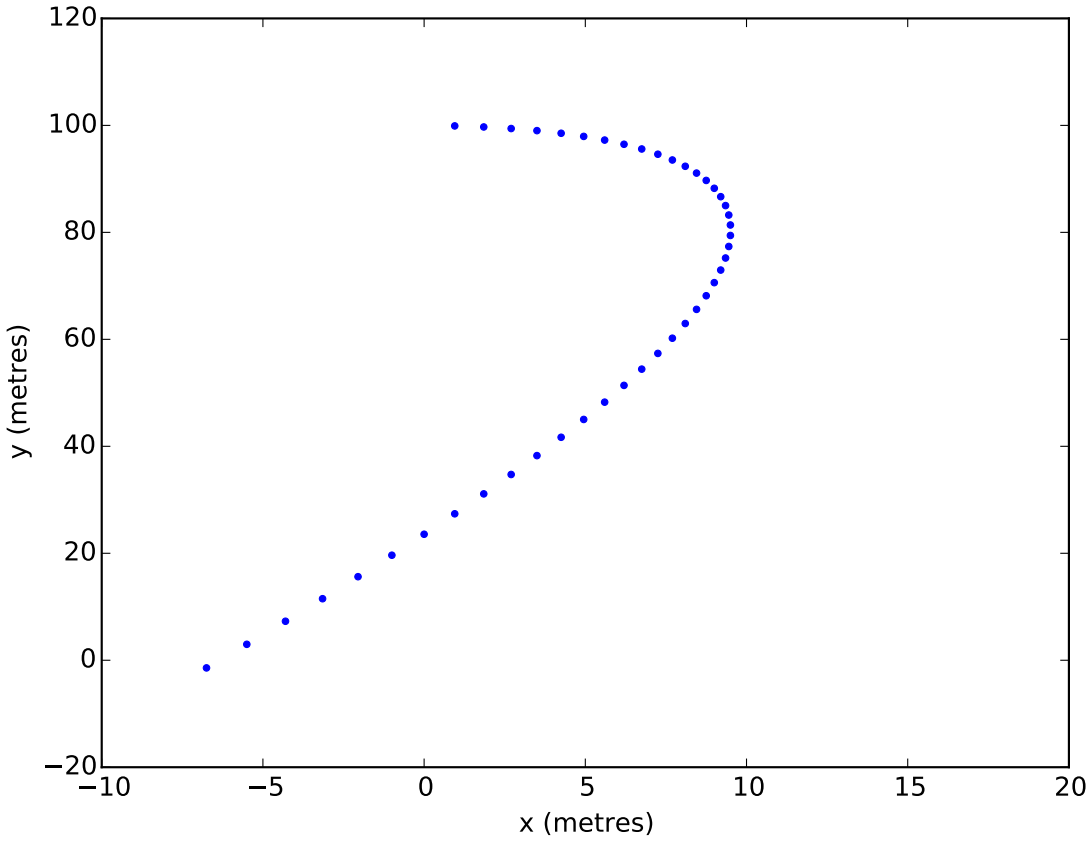
Question 6. **(11 points)**

A Science One student named James writes the following code for the Euler method tutorial. The lines are numbered along the left-hand side for your reference.

```
1  # Author:      James Charbonneau
2  # Description: Use the Euler method to simulate a parcel dropped
3  #              from an airplane moving in 2-D subject to drag
4  #              until it hits the ground.
5
6  import matplotlib.pyplot as plt
7  import math
8
9  dt    =  0.1  # step size (s)
10 g     = -9.8  # acceleration due to gravity (m/s^2)
11 gamma = 0.05  # drag coefficient (kg/m)
12
13 x = 0.0       # initial x-position (m)
14 y = 100.0     # initial y-position (m)
15
16 vx = 10.0     # initial horizontal velocity (m/s)
17 vy = 0.0      # initial vertical velocity (m/s)
18
19 # compute accelerations
20 v2 = vx*vx+vy*vy
21 v  = math.sqrt(v2)
22 ax = -gamma*v*vx
23 ay = g-gamma*v*vy
24
25 while y > 0:
26     # update velocities
27     vx = vx + ax*dt
28     vy = vy + ay*dt
29
30     # update positions
31     x = x + vx*dt
32     y = y + vy*dt
33
34     plt.plot(x, y, '.b')
35
36 plt.xlabel('x (metres)')
37 plt.ylabel('y (metres)')
38 plt.ylim(-20,120) # sets the y limits of the figure to [-20,120]
39 plt.xlim(-10, 20) # sets the x limits of the figure to [-10, 20]
40 plt.savefig('euler_buggy.pdf')
```

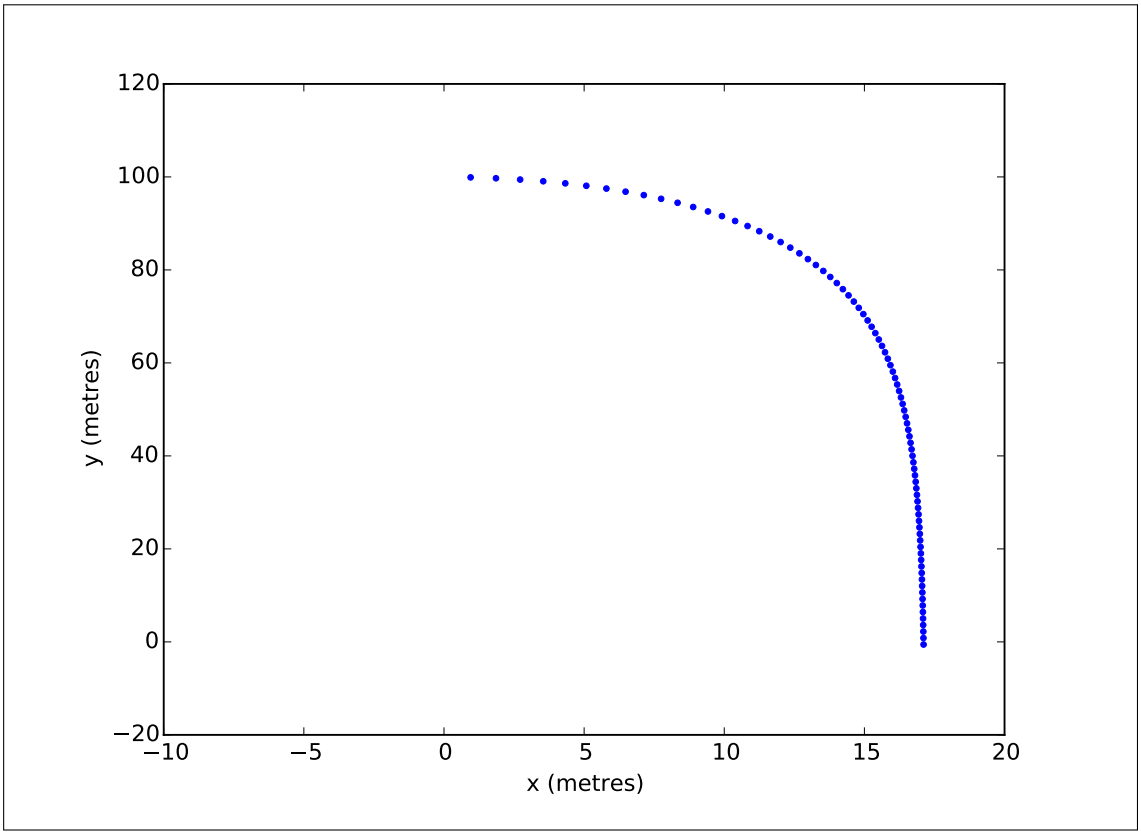
question continues on next page

However, when James runs his code, he obtains the following strange-looking plot:



3 pts (a) On the axes above, draw a curve representing what the plot should look like. You only need the very general shape; no need to spend time trying to get this curve just right.

Solution: Here is the correct plot:



4 pts (b) Briefly explain the problem with James's code (1 sentence maximum). You may refer to specific line numbers if that is helpful.

Solution: The acceleration calculation on lines 19-23 needs to be inside the loop. (Note: a smaller issue is that the positions should be updated before the velocities.)

4 pts (c) What would happen if line 25 were changed to `while y != 0`? (1 sentence maximum.) As a reminder, the `!=` operator means "not equal to".

Solution: The code would get stuck in an infinite loop.