UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Examination in:	INF1100 — Introduction to programming with scientific applications
Day of examination:	Thursday, October 10, 2013
Examination hours:	15.00 – 19.00.

This examination set consists of 8 pages.

Appendices:	None.
Permitted aids:	None.

Make sure that your copy of the examination set is complete before you start solving the problems.

- Read through the complete exercise set before you start solving the individual exercises. If you miss information in an exercise, you can provide your own reasonable assumptions as long as you explain that in detail.
- The maximum possible score on the exam is 25 points. The maximum number of points is listed for each exercise (a correct answer of a subquestion ((a), (b), etc.) gives 1 point).

Exercise 1 (10 points)

What will be the output of the `print` statement in the programs below? Assume that the Python codes are run by version 2.x (e.g., version 2.7), not version 3.x.

(a)

```
counter1 = 10
counter2 = counter1
counter1 = 11
print 'counter2=%d' % counter2
```

(Continued on page 2.)

Solution:

```
counter2=10
```

(b)

```
counter1 = 10
counter2 = counter1
counter1 = 11
counter2 += 2 + counter1
print counter2
```

Solution:

```
23
```

(c)

```
A = [1, 2, 3, -1]
if A[2] < 3:
   A[2] = 0
else:
   A[2] = 10
if A[-1] <= 0:
   A.append(4)
print A
```

Solution:

```
[1, 2, 10, -1, 4]
```

(d)

```
B = [2*x+1 for x in range(5)]
print B[1:-1]
```

Solution:

```
[3, 5, 7]
```

(e)

```
from numpy import linspace
x = linspace(0, 2, 3)
v = x + 2
for x_, v_ in zip(x, v):
    print '%.1f %.1f' % (x_, v_)
```

(Continued on page 3.)

Solution:

```
0.0 2.0
1.0 3.0
2.0 4.0
```

(f)

```
def Q(y):
    r = 4*y
    r = r + 1
    return r

x = 2
print 'Q(%g)=%g' % (x, Q(x))
```

Solution:

```
Q(2)=9
```

(g)

```
x = range(1, 17, 5)
y = x
for x_ in x[1:-1]:
    for y_ in y[1:-1]:
        if x_ != y_ and x_ > y_ + 1:
            print x_, y_
```

Solution:

```
11 6
```

(h) The following program, called `prog.py`,

```
import sys

try:
    r1 = float(sys.argv[1])
    print r1
    r2 = float(sys.argv[2])
    print r2
    r3 = float(sys.argv[3])
    print r3
except IndexError:
    print 'Not enough command-line arguments!', sys.argv[1:]
except ValueError:
    print 'Illegal conversion to float!'
```

(Continued on page 4.)

is run as

```
Terminal> python prog.py 3 6 hello world
```

Solution:

```
3
6
Illegal conversion to float!
```

(i)

```
u = [-1, -2]
v = [1, 2]
print u + v
from numpy import array
u = array(u)
v = array(v)
print u + v
```

Solution:

```
[-1, -2, 1, 2]
[0 0]
```

(j)

```
def equal(a, b, eps=1E-14):
    """Test if a==b with tolerance eps."""
    return abs(a - b) < eps

def some_function(Q):
    if Q < 0:
        raise ValueError('Q<0 is not allowed')

    if equal(Q, 0):
        return 0, None
    else:
        return 1, 2

def run_some_function():
    try:
        r1, r2 = some_function(-1)
        print 'Something is wrong with some_function(-1)!'
        ok = False
    except:
```

(Continued on page 5.)

```
            ok = True
        r1, r2 = some_function(0)
        if not equal(r1, 0) or not r2 == None:
            print 'Something is wrong with some_function(0)!'
            ok = False
        r1, r2 = some_function(2)
        if not equal(r1, 1) or not equal(r2, 2):
            print 'Something is wrong with some_function(2)!'
            ok = False
        if ok:
            print 'All tests passed!'
        else:
            print 'Some test(s) failed!'

    run_some_function()
```

Solution:

```
All tests passed!
```

Exercise 2 (3 points)

We want to write a program that can compute values of the function $g(t) = e^{-at}$ and its derivative $g'(t) = -ae^{-at}$, where a is some known parameter. Write a Python function `g(t)` that evaluates and returns the values of $g(t)$ and $g'(t)$. The parameter a can be a global variable. Demonstrate how the function is called and how the returned result can be stored in variables. Write out the computed value of $g'(t)$ to the screen.
Solution:

```
from math import exp

def g(t):
    return exp(-a*t), -a*exp(-a*t)

a = 2
g_value, dg_value = g(0.1)
print dg_value
```

Exercise 3 (3 points)

Extend the program in Exercise 2 such that the t and a values are read from the command line. Add a `try-except` block to handle the cases that

(Continued on page 6.)

the user has failed to provide enough command-line arguments or when the command-line arguments cannot be interpreted as real numbers. Stop the program in those cases.

Hint: You may get inspired by code snippets elsewhere in this exam. Even if you did not succeed in writing the function in Exercise 2, you can just assume that it is available as described.

Solution:

```
from math import exp
import sys

def g(t):
    return exp(-a*t), -a*exp(-a*t)

try:
    t = float(sys.argv[1])
    a = float(sys.argv[2])
except IndexError:
    print 'Not enough command-line arguments (two are needed)', sys.argv[1:]
    sys.exit(1)
except ValueError:
    print 'Cannot convert command-line arguments to floats', sys.argv[1:]
    sys.exit(1)

g_value, dg_value = g(t)
print dg_value
```

Exercise 4 (3 points)

Assume that you have the function that evaluates $g(t)$ and $g'(t)$ as specified in Exercise 2. Make a code snippet for creating a plot with the two curves $g(t)$ and $g'(t)$, for $0 \leq t \leq 5/a$. Add a label `t` on the x axis in the plot and include a legend for each curve. Save the plot to a file.

Solution:

```
from numpy import exp # need vectorized exp now

def g(t):
    return exp(-a*t), -a*exp(-a*t)

a = 2
import numpy as np
t = np.linspace(0, 5/float(a), 101)
g_curve, dg_curve = g(t)
```

(Continued on page 7.)

```
# Plotting with matplotlib
import matplotlib.pyplot as plot
plt.plot(t, g_curve, t, dg_curve)
plt.xlabel('t')
plt.legend(['g', "g'"])
plt.savefig('plot.png')
plt.show()

# Alternative plotting with scitools
from scitools.std import *
plot(t, g_curve, t_dg_curve,
     legend=['g', "g'"],
     xlabel='t', savefig='plot.png')
```

Exercise 5 (3 points)

The quadratic equation

$$ax^2 + bx + c = 0$$

has two solutions

$$x_1 = \frac{-b + \sqrt{q}}{2a}, \quad x_2 = \frac{-b - \sqrt{q}}{2a},$$

where $q = b^2 - 4ac$. The two solutions coincide if $q = 0$. Write a Python function that takes a, b, and c as arguments and returns the two solutions x_1 and x_2. Require $q \geq 0$ so that x_1 and x_2 are real numbers. Raise a `ValueError` exception if $q < 0$. Return x_1 and `None` if $x_1 = x_2$.

Write a main program with necessary code for demonstrating how the function can be used to solve the particular quadratic equation $x^2 + 3x = -1$.

Solution:

```
def solve_quadratic_equation(a, b, c):
    q = b**2 - 4*a*c
    if q < -1E-14:    # q < 0 with tolerance
        raise ValueError('q=%g<0 is not allowed' % q)
    x1 = (-b + sqrt(q))/(2*a)
    x2 = (-b - sqrt(q))/(2*a)
    if q > 1E-14:     # q > 0 with tolerance
        return x1, x2
    else:
        return x1, None

from math import sqrt
print solve_quadratic_equation(1, 3, 1)
```

(Continued on page 8.)

Exercise 6 (3 points)

The purpose of this exercise is to test the function developed in the previous exercise. Even if you have not succeeded in writing the function, you can just assume that it is available as specified in Exercise 5.

Write a test function that calls the function from the previous exercise and checks that the results are correct in three test cases: $q > 0$, $q = 0$, and $q < 0$. Write a message to the screen when a test fails, and write `OK` if all tests succeed. Remember to test equality of real numbers with a tolerance.

Hint: You may get inspired by code snippets elsewhere in this exam.

Solution:

```
def equal(a, b, eps=1E-14):
    """Test a==b with tolerance eps."""
    return abs(a - b) < eps

def test_solve_quadratic_equation():
    ok = True    # True if all tests pass
    eps = 1E-14  # tolerance for testing equality of floats
    # Case 1: q > 0
    # x1 = 1, x2 = -1  (x-1)*(x+1)=0 => x**2 - 1 = 0
    x1, x2 = solve_quadratic_equation(1, 0, -1)
    if not equal(x1, 1) or not equal(x2, -1):
       print 'Test with q>0 failed.'
       ok = False

    # Case 2: q = 0
    # x1 = 2, x2 = 2  (x-2)*(x-2)=0 => x**2 - 4*x + 4 = 0
    x1, x2 = solve_quadratic_equation(1, -4, 4)
    if not equal(x1, 2) or not x2 == None:
       print 'Test with q==0 failed.'
       ok = False

    # Case 3: q < 0
    # b=1, a=1, c=1 => b**2 - 4*a*c =  1 - 4 = -3
    try:
        x1, x2 = solve_quadratic_equation(1, 1, 1)
        print 'Test with q<0 failed.'
        ok = False
    except ValueError:
        # Do nothing
        pass # or do print ''
    if ok:
        print 'OK'
```

END