UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Examination in:	INF1100 — Introduction to programming with scientific applications
Day of examination:	Thursday, October 6, 2014
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 is printed in the terminal window when the programs below are run?

(a)

```
a = 4
b = a
a = 2
print 'b:', b
```

(Continued on page 2.)

Solution:

```
b: 4
```

(b)
```
a = 2
for i in range(2):
    a += 2
print a
```

Solution:

```
6
```

(c)
```
A = [1, 2, 3, -1]
del A[-1]
A.append(-2)
print A
```

Solution:

```
[1, 2, 3, -2]
```

(d)
```
import sys
a = 4
b = 6
c = a + b
a = sys.argv[1]
b = sys.argv[2]
print c
```

Execution:

```
Terminal> python myprog.py 1 2
```

Solution:

```
10
```

(e)
```
import numpy as np
x = np.linspace(0, 4, 3)
y = x**2
for x_, y_ in zip(x, y):
    print '%.1f %.1f' % (x_, y_)
```

(Continued on page 3.)

Solution:

```
0.0 0.0
2.0 4.0
4.0 16.0
```

(f)
```
def P(x):
    return x + 1

def Q(y):
    return 2*y

print Q(P(Q(3)))
```

Solution:

```
14
```

(g)
```
import sys
A = [1, 7, 15]

try:
    r = float(A[3])
except IndexError:
    print 'A has length %g' % len(A)
except ValueError:
    print 'Cannot convert %g to float!' % A[3]
```

Solution:

```
A has length 3
```

(h)
```
u = [0, 1]
v = [2, 3]
print u + v
from numpy import array
u = array(u)
v = array(v)
print u + v
```

Solution:

```
[0, 1, 2, 3]
[2 4]
```

(Continued on page 4.)

(i)
```
def plus1(x):
    return x + 1

def test_plus1():
    x = 4.5
    exact = 5.5
    computed = plus1(x)
    tol = 1E-14
    success = abs(exact - computed) < tol
    msg = 'exact=%g, computed=%g' % (exact, computed)
    assert success, msg

test_plus1()
```

Solution: Nothing gets printed since `success` is `True`.

(j)
```
for i in range(-1, 1, 2):
    for j in range(3):
        if i == j:
            print i, j
```

Solution:

`Nothing gets printed.`

Exercise 2 (3 points)

Write a Python function for computing the sum

$$K(x) = \sum_{k=0}^{N} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

The factorial $(2k+1)!$ can be computed by `math.factorial(2k+1)`. Print out $K(\pi)$ for $N = 4$.

Solution: The most straightforward solution goes as follows.

```
from math import factorial

def K(x, N):
    x = float(x)  # avoid integer division
    s = 0  # summation variable
    for k in range(N+1):
        s += (-1)**k*x**(2*k+1)/factorial(2*k+1)
    return s
```

(Continued on page 5.)

```
from math import pi
print K(pi, 4)
```

As pointed out in Exercise A.14 (`sin_Taylor_series_diffeq.py`) it can be more efficient to compute $K(x)$, which is the Taylor polynomial for $\sin(x)$, via a system of two difference equations:

$$e_n = e_{n-1} + a_{n-1}, \quad e_0 = 0,\ a_0 = x$$
$$a_n = \frac{-x^2}{2n(2n+1)} a_{n-1}$$

A corresponding Python implementation reads

```
from numpy import zeros

def K(x, N):
    a = zeros(n+1)
    s = zeros(n+1)
    a[0] = x
    s[0] = 0
    for n in range(1, N+1):
        s[n] = s[n-1] + a[n-1]
        a[n] = a[n-1]*(-1)*x**2/((2*n+1)*(2*n))
    return s[n]

from math import pi
print K(pi, 4)
```

A more memory efficient implementation is to store only the two most recent values in the difference equations (we are only interested in the final value s_n anyway and we do not need to plot the sequence).

```
def K(x, N):
    sn_prev = 0  # here s[0], stores in general s[n-1]
    an_prev = x  # here a[0], stores in general a[n-1]
    for n in range(1, N+1):
        sn = sn_prev + an_prev
        an = an_prev*(-1)*x**2/((2*n+1)*(2*n))
        # change contents (be ready for next pass in the loop):
        sn_prev = sn
        an_prev = an
    return sn

from math import pi
print K(pi, 4)
```

(Continued on page 6.)

Exercise 3 (3 points)

Extend the program in Exercise 2 such that the x and N values are read from the command line. Add a `try-except` block to handle the cases that 1) the user has failed to provide enough command-line arguments or 2) the command-line arguments cannot be interpreted as numbers. Write an error message and stop the program in those cases.

Solution:

```
import sys

try:
    x = float(sys.argv[1])
    N = int(sys.argv[2])
except IndexError:
    print 'Not enough command-line arguments! Need x and K.'
    sys.exit(1)
except ValueError:
    print 'Cannot convert %s and %s to numbers' % \
          (sys.argv[1], sys.argv[2])
    sys.exit(1)

print 'K(%g, %d)=%g' % (x, N, K(x, N))
```

Exercise 4 (3 points)

Write the necessary code for plotting the $K(x)$ function on $[0, 2\pi]$ in Exericse 2. Mark the x axis with `x`, the y axis with `K`, introduce a legend reflecting value of N, and save the plot to a PNG file.

Solution:

```
import numpy as np
x = np.linspace(0, 2*np.pi, 101)
N = 5
y = K(x, N)
import matplotlib.pyplot as plt
# import scitools.std as plt
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.legend(['N=%d' % N])
plt.savefig('tmp.png')
plt.show()
```

(Continued on page 7.)

Exercise 5 (3 points)

Write a *test function* for verifying the implementation of the `K(x, N)` function in Exercise 2. Hint: Choose an x and N, compute the result by hand, and let the test function compare this exact result with a call to `K(x, N)`. The test function should follow the standard conventions for such functions (i.e., have a name on the form `test_*()` and perform the test via an `assert` statement).

Solution: With $x = 1/2$ and $N = 1$ we get

$$K(x) = x - \frac{1}{6}x^3 = \frac{1}{2} - \frac{1}{6}\frac{1}{8} = \frac{23}{48}.$$

```
def test_K():
    x = 0.5
    N = 1
    exact = 23.0/48
    computed = K(x, N)
    tol = 1E-14
    success = abs(exact - computed) < tol
    assert success, 'exact=%g, computed=%g' % (exact, computed)
```

Exercise 6 (3 points)

A text file with name `simulations.dat` contains a header line, a blank line, and then three columns with numbers:

```
time    measurement   prediction

0.0     -1.376        -1.233
0.1     -1.091        -1.001
0.2      1.087         0.986
0.3      2.136         3.001
```

The length of the columns with numbers is not known.
You are asked to read the file in a program, store the data in three Numerical Python (`numpy`) arrays and plot the "measurement" and "prediction" values versus the "time" values in the same figure.

Solution:

```
# Read file data into lists
infile = open('simulations.dat', 'r')
infile.readline(); infile.readline() # skip first two lines

# Start with empty lists since we do not know how
```

(Continued on page 8.)

```
# many lines the file has
time = []
measurement = []
prediction = []

for line in infile:
    words = line.split()
    floats = [float(word) for word in words]
    time.append(floats[0])
    measurement.append(floats[1])
    prediction.append(floats[2])
infile.close()

# Convert to numpy arrays
import numpy as np
time = np.array(time)
measurement = np.array(measurement)
prediction = np.array(prediction)

# Plot measurement with red circles and
# prediction with blue solid line
import matplotlib.pyplot as plt
plt.plot(time, measurement, 'ro',
         time, prediction, 'b-')
plt.legend(['measurement', 'prediction'])
plt.xlabel('time')
plt.show()
```

Note: since each number in the file starts at a certain column number in the line string, it is possible to extract the data by string slicing:

```
for line in infile:
    time.append(float(line[0:8]))
    measurement.append(float(line[8:22]))
    prediction.append(float(line[22:]))
```

END