UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Examination in:	IN1900 — Introduction to programming with scientific applications
Day of examination:	Tuesday, October 10, 2017
Examination hours:	9.00 – 13.00

This examination set consists of 9 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 28 points. The maximum number of points is listed for each question. For questions with sub-questions ((a),(b), etc), each sub-question has the same score.

(Continued on page 2.)

Problem 1.

7 What is printed in the terminal window when the programs below are run?

(a)

```
a = 4
b = a
a = a+b
print(a)
```

(b)

```
A = [[-1,0,1],[0,0,0],[10,9,8]]
print(A[1][-1])
```

(c)

```
x = 6
y = -2
c = x >= 10 or y != -2
```

(d)

```
import numpy as np
a = [1,2,3]
a_np = np.array(a)
print(a*2)
print(a_np*2)
```

(e)

```
S = 0
for i in range(3):
    S += i**2

print(S)
```

(Continued on page 3.)

(f)

```
import sys
A = [['-1','0','1'],['0','0','0'],['10','9','8']]

try:
    b = int(A[2])
except IndexError:
    print('A has length %d' %len(A))
    sys.exit(1)
except TypeError:
    print('Cannot convert %s to int' %A[2])
    sys.exit(1)
```

(g)

```
def poly(x,k):
    s = 0
    for i in range(k+1):
        s = s+x**i
    return s

def test_poly():
    k = 2
    x = 2.0

    tol = 1e-14
    success = abs(poly(x,k)-7.0) < tol
    assert success

test_poly()
```

Solution:

```
a:
8
b:
```

(Continued on page 4.)

```
0
c:
False
d:
[1, 2, 3, 1, 2, 3]
[2 4 6]
e:
5
f:
Cannot convert ['10', '9', '8'] to int
g:
(Nothing is printed since the test passes)
```

Problem 2.

6

(a) You have a file named `data.txt` that contains weather data for 25 december at Blindern, in the format listed below. The file starts with data from 1950 and continues to 2008 (the table below does not show the entire file).

```
# Station, year, mean-temp, min-temp, max-temp
18700  1950  -5.9  -8.3 -2.6
18700  1951  6.0     4.5    7.3  0
18700  1952  -1.4    -1.8    -0.2
18700  1953  -0.2    -1.2    6.0  0
18700  1954  -8.5    -9.9    -3.8
18700  1955  -4.0    -7.7    -0.4
18700  1956  -4.9    -5.8    -4.7
18700  1957  -0.4    -1.7    1.9  0
18700  1958  -0.2    -1.0    0.9  7
18700  1959  2.1    0.8    3.5  0
18700  1960  -3.6    -9.7    -2.6
18700  1961  -6.0    -9.0    -2.1
18700  1962  -9.5    -11.3    -7.7
```

(Continued on page 5.)

```
18700  1963  -2.7    -4.1    -0.9
18700  1964  -8.0    -9.5    -6.0
18700  1965  -2.0    -3.5    -1.0
18700  1966  -6.2    -10.6    -1.4
18700  1967  -6.5    -9.3    -5.2
18700  1968  -1.5    -2.8    0.1
```

Write a program that reads data from the file data.txt and makes four lists of data from the columns 'year', mean-temp, max-temp and min-temp. Include necessary imports.

(b) Extend the program from question 2a, and plot the three temperatures 'mean_temp', 'min_temp' and 'max_temp' as a function of year. The axes of the plot shall be marked with 'Year' and 'Temperature', and there shall be a legend to explain each curve. Include necessary imports.

You may assume that you write the code for the plot in the same file as the code in question 2a, so the four lists are already available.

Solution:

```
a:
infile = open('data.txt','r')

infile.readline()

year = []
mean_t = []
min_t = []
max_t = []

for line in infile:
    words = line.split()
    year.append(int(words[1]))
    mean_t.append(float(words[2]))
    min_t.append(float(words[3]))
    max_t.append(float(words[4]))
```

(Continued on page 6.)

```
b:
import matplotlib.pyplot as plt

plt.plot(year,mean_t,label= 'mean')
plt.plot(year,min_t, label='min')
plt.plot(year,max_t,label='max')

plt.legend()
plt.xlabel('Year')
plt.ylabel('T (degrees)')

plt.show()
```

Problem 3.

9

(a) Write a python-function `piecewise(x,a,b)` that implements the function:

$$f(x) = \begin{cases} 0.0 & \text{for} \quad x \leq a \\ \frac{x-a}{b-a} & \text{for} \quad a < x \leq b \\ 1.0 & \text{for} \quad x > b \end{cases}$$

Here x, a, and b are scalar values (numbers, not arrays or lists).

(b) Write a test function `test_piecewise()` that tests the function from question 3a. Set a = 0 and b = 1, and choose three different values for x; -1.0, 0.5, and 1.5. The result of these three arguments shall be compared with the expected values 0.0, 0.5 and 1.0.

You can assume that the function `piecewise()` is available in the same file as the test function, so you don't have to write it again.

(c) Write a program that reads the values x, a, and b from the command line, calls the function from 3a with these parameters, and prints the result to the screen. Include a try-except block that

(Continued on page 7.)

handles the two cases that there are not enough input arguments, and that the input arguments have the wrong format. In both cases the program shall print an error message and exit. The error message shall be different for the two types of errors. Include necessary imports.

You can assume that the function from 3a is available in the same file as your program, so you don't have to import it or write it again.

Solution:

```
a:
def piecewise(x,a,b):
    if x < a:
        return 0.0
    elif x < b:
        return (x-a)/(b-a)
    else:
        return 1.0


b:
"""
There are many different ways to test several values
in a single test function. Here are two of the simpler
alternatives, both would give full score on the exam.
"""
def test_piecewise():
    a = 0.0; b = 1.0;
    x1 = -1; x2 = 0.5; x3 = 1.5
    e1 = 0.0; e2 = 0.5; e3 = 1.0
    tol = 1.0e-10
    success1 = abs(piecewise(x1,a,b) - e1) < tol
    success2 = abs(piecewise(x2,a,b) - e2) < tol
    success3 = abs(pieceswise(x3,a,b) - e3) < tol
    assert success1 and success2 and success3

def test_piecewise():
    a = 0.0; b = 1.0;
```

(Continued on page 8.)

```
    x1 = -1.0; x2 = 0.5; x3 = 1.5
    e1 = 0.0; e2 = 0.5; e3 = 1.0
    tol = 1.0e-10
    c1 = piecewise(x1,a,b)
    c2 = piecewise(x2,a,b)
    c3 = piecewise(x3,a,b)
    success = abs(c1 - e1) < tol and abs(c2 - e2) < tol \
      and abs(c3 - e3) < tol
    msg = """
Test failed, computed %g, %g, %g,
expected %g, %g, %g""" %(c1,c2,c3, e1,e2,e3)
    assert success, msg
```

```
c:
import sys

try:
    x, a, b = sys.argv[1:]
    x = float(x)
    a = float(a)
    b = float(b)
except IndexError:
    print('You need to provide three command line arguments.')
    sys.exit(1)
except ValueError:
    print('The command line arguments must be numbers.')
    sys.exit(1)

print(piecewise(x,a,b))
```

Problem 4.

6

(Continued on page 9.)

(a) Write a Python function `pi_approx(n)`, which implements the sum

$$s_n = 4\sum_{k=1}^{n} \frac{(-1)^{k+1}}{2k-1}$$

Write code for calling the function for $n = 10$ og $n = 100$ and printing the result to the screen.

(b) Write a program that generates a list of n-values from 1 to 50, calls the function from 4a for all the values, and plots the function values as a function of n. Include necessary imports. You can assume that the function from 4a is available in the same file as your program, so you don't have to import it or write it again.

```
a:
def pi_approx(n):
    a = 0
    for k in range(1,n+1):
        a += (-1)**(k+1)/(2*k-1)
    return 4*a

print(pi_approx(10), pi_approx(100))

b:
"""You can use lists or arrays,
but in this case lists are simpler."""
n_list = range(1,51)
a_list = []
for i in n_list:
    a_list.append(pi_approx(i))

plt.plot(n_list,a_list)
plt.show()
```

END