# UNIVERSITETET I OSLO

## Det matematisk-naturvitenskapelige fakultet

| | |
|---|---|
| Examination in: | INF1100 — Introduction to programming with scientific applications |
| Day of examination: | Thursday, October 11, 2012 |
| Examination hours: | 15.00 – 19.00. |
| This examination set consists of 6 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)

```
a = 2
b = a
a = 3
print b
```

*(Continued on page 2.)*

(b)

```
from numpy import linspace
t = linspace(0, 1, 3)
y = t**2
for t_, y_ in zip(t, y):
    print '%.1f %.1f' % (y_, t_)
```

(c)

```
def g(x):
    return 1 - x/4

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

(d)

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

(e)

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

(f)

```
def iterate(f, x, dfdx, tolerance=1.0E-2, max_n=5):
    n = 0
    while abs(f(x)) > tolerance and n <= max_n:
        x = x - f(x)/dfdx(x)
        n += 1
    if n > max_n:
        raise ValueError('Iteration did not converge')
    else:
        return x, f(x)

def g(t):
    return (1-x)*(2-x)  #2 -3x +x^2
```

*(Continued on page 3.)*

```
def dgdt(t):
    return 2*x - 3

def g(t):
    return 1-t

def dgdt(t):
    return -1

print iterate(g, 1, dgdt)
print iterate(g, 12.5, dgdt)
```

(g)

```
import numpy as np
x = np.linspace(1, 5, 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_
```

(h)

```
A = [[0, 0], [0, -1], [1, 3], [2, 4], [0, -2]]
print A[2]
print A[3][1]
print A[2:]
```

(i)

```
numbers = (1, 4, 8, 3, 2)
k = numbers[2]
try:
    element = float(numbers[k])
    print 'element=%f' % element
except IndexError:
    print 'Index %d > %d' % (k, len(numbers))
except ValueError:
    print 'Could not convert %d to float' % (numbers[k])
```

(j)

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

*(Continued on page 4.)*

## Exercise 2 (3 points)

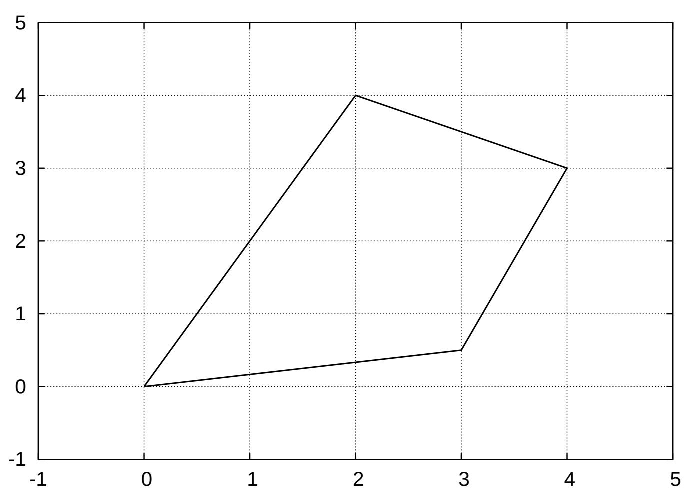
It is known that one inch is 2.54 cm and that one foot equals 12 inches. Make a function `fts2ms(v)` that converts a velocity `v` from feet per second to meter per second. Use the function to convert the velocity 3 ft/s to m/s.

## Exercise 3 (4 points)

An arbitrary triangle can be described by the coordinates of its three vertices: $(x_1, y_1)$, $(x_2, y_2)$, $(x_3, y_3)$. The area of the triangle is given by the formula

$$A = \frac{1}{2}\left(x_2y_3 - x_3y_2 - x_1y_3 + x_3y_1 + x_1y_2 - x_2y_1\right),$$

when $(x_1, y_1)$, $(x_2, y_2)$, $(x_3, y_3)$ are listed in counterclockwise direction. Write a function `area(vertices)` that returns the area of a triangle whose vertices are specified by the argument `vertices`, which is a nested list of the vertex coordinates. For example, `vertices` is `[[0,0], [1,0], [0,2]]` if the three corners of the triangle have coordinates $(0, 0)$, $(1, 0)$ and $(0, 2)$. Show how to use the `area` function to compute the area of the four-sided quadrilateral figure below.




## Exercise 4 (4 points)

The purpose of this exercise is to plot the size of the terms in a Taylor polynomial. We write the polynomial on the form

$$p(x) = \sum_{i=0}^{N} t_i(x). \tag{1}$$

*(Continued on page 5.)*

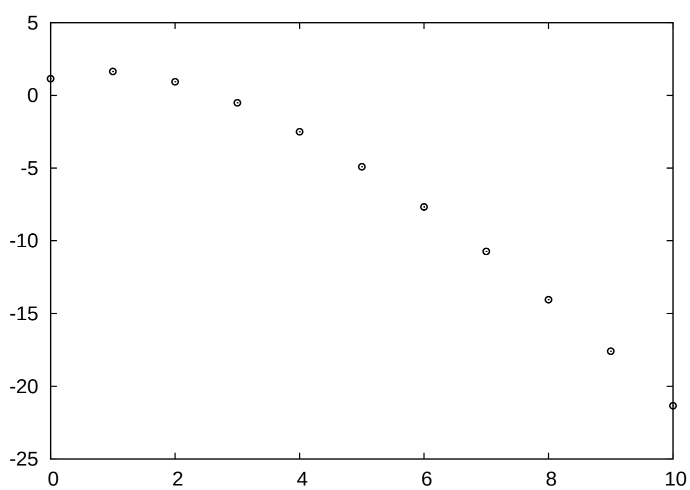
As a specific example, the terms $t_i(x)$ in the Taylor polynomial for $\sin x$ are given as

$$t_i(x) = (-1)^i \frac{x^{2i+1}}{(2i+1)!}.$$

Make a function `terms(ti, x_0, N)` that returns an array of $t_i(x_0)$, $i = 0, \ldots, N$. The argument `ti` is some Python function of `i` and `x` for evaluating $t_i(x)$, `x_0` corresponds to $x_0$ and `N` to $N$. Also write a function `ti_sin(i, x)` for evaluating the specific $t_i(x)$ in the Taylor polynomial for $\sin x$ (given above).

Make another function `visualize(t)` that plots the logarithm of the absolute value of the elements in the `t` array against their indices ($i = 0, \ldots, N$). That is, the function plots `log(abs(t[i]))` versus `i`. Use small circles to visualize the data points (do not draw solid lines between the points).

Demonstrate how to call the `terms` and `visualize` functions for displaying how rapidly the 10 first terms in the Taylor polynomial for $\sin x$ go to zero. The resulting figure when $x_0 = \pi$ is displayed next.




The reason for working with $\ln |t_i(x_0)|$ instead of just $t_i(x_0)$ is that the size of the terms in Taylor polynomials decreases by many orders of magnitude as $i$ grows, and this decrease is not visible in a plot if we do not take the logarithm of the small values. The point with the exercise is to visualize how fast a Taylor series converges.

## Exercise 5 (4 points)

Newton's method for solving a possibly nonlinear algebraic equation $g(x) = 0$ consists of generating a sequence of approximations to a solution:

$$x_n = x_{n-1} - \frac{g(x_{n-1})}{g'(x_{n-1})}.$$

*(Continued on page 6.)*

When $|g(x_n)| \leq \epsilon$, we accept $x_n$ as a good approximation to the solution of $g(x) = 0$.

Implement a function that takes the $g(x)$ function and its derivative $g'(x)$ as parameters, along with $x_0$, a tolerance ($\epsilon$) and a maximum $n$ value. Raise an exception if $n$ exceeds the maximum $n$ value without meeting the convergence criterion $|g(x_n)| \leq \epsilon$. Let the function return the sequences $x_n$ and $g(x_n)$, $n = 0, 1, 2, \ldots$.

Demonstrate how to use the function to solve the equation $\sin x + \cos^2 x = e^x$ if $x_0 = -4$ is the initial guess. Write the last element in the sequence $x_0, x_1, \ldots$ to the terminal window (the last element is usually the best approximation to the root of the equation). Add a plot command to visualize the sequence $x_0, x_1, \ldots$.

END