UNIVERSITY OF SOUTHAMPTON FEEG1001W1

SEMESTER 1 EXAMINATION 2012-2013

FEEG1001 Computing

Duration: 90 mins.

Attempt **ALL QUESTIONS 1, 2, 3, 4, 5, 6, 7** and **8**.

A total of 90 marks are available for this paper.

Questions 1 to 7 carry 10 marks each.
Question 8 carries 20 marks.

Only University approved calculators may be used.

A foreign language translation dictionary (paper version) is permitted provided it contains no notes, additions or annotations.

The exam is open book.

Marks in brackets are for guidance only.

Question 1 **[10 marks]**

Implement a function `mult3vec(c,v)` that takes as an argument a scalar `c` and a sequence `v` with three elements, i.e. `v = [v0, v1, v2]`. The function should multiply every component of the sequence `v` with the scalar `c`, and return a list with the three elements `[c * v0, c * v1, c * v2]`.

Example:

```
>>> print(mult3vec(3, [0, 1, 2]))
[0, 3, 6]
>>> mult3vec(0.5, [100, 42.5, 3.14])
[50.0, 21.25, 1.57]
```

Question 2 **[10 marks]**

Implement a function `multvec(c,v)` that takes as argument a scalar `c` and a sequence `v` of arbitrary length, i.e. `v = [v0, v1, v2, ...]`. The function should multiply every component of the sequence `v` with the scalar `c`, and return a list of the same length as `v` which contains the products of `c` and the elements of `v`.

Example:

```
>>> multvec(3, [0, 1])
[0, 3]
>>> multvec(3, [0, 1, 2, 3, 4])
[0, 3, 6, 9, 12]
```

Question 3 **[10 marks]**

Implement a function `convert_time(t)` that takes as argument a duration in minutes `t`, and returns a tuple `(d,h,m)` containing the corresponding number of days `d`, hours `h` and minutes `m`. You can assume that the input argument `t` is of type integer and non-negative.

Example:

```
>>> print(convert_time(100))
(0, 1, 40)
>>> convert_time(4000)
(2, 18, 40)
```

Question 4 **[10 marks]**

Implement a function `nearest(xs,a)` which takes as input a list `xs` of numbers and a number `a`. The function should return the element of `xs` that is the nearest to `a`.
You can assume that the list `xs` has at least one element. If there are two or more elements in `xs` that have the same minimum distance to `a`, the `nearest` function can return any one of those elements.

Example:

```
>>> nearest([1, 5, 3, 2, 4], 1.9)
2
>>> nearest([1, 6, 3], 1.9)
1
```

Question 5 **[10 marks]**

Implement a function `derivative(f,xs)` that takes as arguments a function `f` and a list of numbers `xs`. This function should return an approximation of the derivative $f'(x)$ of $f(x)$ at each value x in the list `xs`. The approximation of the derivative $f'(x)$ at position x is calculated using the forward finite difference:

$$f'(x) \equiv \frac{\mathrm{d}f}{\mathrm{d}x}(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

where we choose $\Delta x = 10^{-6}$.

If you can, implement this function using list comprehension.

Example:

```
from math import cos

def f(x):
    return x ** 2

def g(x):
    return cos( 4.0 * x)

>>> derivative(f, [1, 2.2, 4])
[2.0000009999243673, 4.400001000881559,
 8.000000999430767]
>>> derivative(g, [0, -2.3])
[-8.000045070843953e-06, 0.891567455307829]
```

Question 6 **[10 marks]**

Write a function `read(filename)` which reads data from a file with name `filename`, and returns a tuple of two lists containing the data as floating point numbers.
You can assume that the file will contain either a line that starts with a #, or a line containing two numbers that are separated by a comma. The two numbers represent the x and y coordinates of some data. All the x coordinates should be gathered in a list `xs` of floating point numbers and all y coordinates should be gathered in a list `ys` of floating point numbers. If the line starts with a `#`, then the whole line should be ignored.
The function should return a tuple `(xs, ys)` which contains the list `xs` as first element and the list `ys` as the second element.

Example: for a file with the name data.txt that reads

```
# some point data (2d)
1.5,20
10,42
100,-42.5
```

We expect the following behaviour:

```
>>> read('data.txt')
([1.5, 10.0, 100.0], [20.0, 42.0, -42.5])
```

Question 7 **[10 marks]**

The Fibonacci series is defined by the recursion

$$F[n] = F[n-1] + F[n-2] \quad \text{with} \quad F[0] = 0 \text{ and } F[1] = 1$$

The series starts like this: $[0, 1, 1, 2, 3, 5, 8, ...]$.

Write a function `isfib(F)` that returns `True` if the supplied list `F` is a Fibonacci series and `False` otherwise. The function should return `False` if the supplied list has less than two items.

Example:

```
>>> isfib([0,1,1,2,3,5,8])
True
>>> isfib([1,1,2])
False
>>> isfib([4,1,4,5])
False
>>> isfib([0])
False
```

Question 8 **[20 marks]**

1. Write a function `f_from_data(xs,ys)` which accepts tabulated data of a function $y(x)$ where x-positions are given in the first list `xs` and the corresponding values for $y(x)$ are given in the second list of numbers `ys`. The function `f_from_data` should return a callable object `f(x)` that returns the tabulated data for $y(x)$ where it is known, and use linear interpolation for x values between the tabulated positions.

   Example:

```
>>> f = f_from_data([3, 4, 6], [0, 1, 2])
>>> print f(3)
0.0
>>> print f(4)
1.0
>>> print f(3.5)
0.5
>>> print f(5)
1.5
>>> print f(6)
2.0
```

2. Write a function `root(xs,ys)` which accepts tabulated data of a function $y(x)$ where x-positions are given in the first list `xs` and the corresponding values for $y(x)$ are given in the second list of numbers `ys`. The function should return the value x for which $y(x)$ is (approximately) zero, using linear interpolation to obtain values of $y(x)$ between the tabulated positions.

You can assume that the tabulated function $y(x)$ crosses the zero line exactly once, and that the values `ys[0]` and `ys[-1]` have opposite signs. We recommend that you use the `scipy.optimize.brentq` function as part of your solution.

Example:

```
>>> xs = [4, 6, 7]
>>> ys = [-1.0, 1.5, 2]
>>> print(root(xs, ys))
4.8
```

END OF PAPER