UNIVERSITY OF OSLO

Faculty of Mathematics and Natural Sciences

Exam in	**INF3331**
Day of exam:	**2012-12-05**
Exam hours:	**4**

This examination paper consists of 15 pages including 11 pages of appendices.

Appendices:	**2 (Regex syntax and csv documentation)**
Permitted materials:	**None**

Make sure that your copy of this examination paper is complete before answering.

It is possible to answer the exam in either Norwegian or English. For all the exercises it is important to specify any assumptions you make, in particular in cases where you feel the exercise text is unclear.

1: Vectorization (5 points)

Vectorization plays an important role in high-level languages like Python. Explain why, and provide an example of a vectorized operation using NumPy arrays. The example should be a complete Python script, including necessary imports.

This is the exact same exercise as last year. As last year, I do not expect a long and detailed answer. The explanation and the example should be scored as follows:

- *Up to two points for a valid explanation, which includes the main points that loops in Python are slow and vectorization runs the loops in C or similar.*
- *Up to three points for a working example. It can be simple, such as array-array or array-scalar multiplication.*
    - *Scripts with insignificant syntax errors (i.e. typos) should get full score.*
    - *Mostly correct, but missing import or similar can be given two points*
    - *Wrong syntax, but getting parts of the idea; one point*

2. Regular expressions (5 points)

The following Python commands do not give the desired output, which is to extract the two intervals specified in the string (one with real numbers and one with integers). Explain why the regular expression fails. Suggest a regular expression that works.

```
>>> import re
>>> l = re.search(r"\[(.*),(.*)\]", " [-3.2E+01,0.11  ] and [-4,8]").groups()
>>> l
('-3.2E+01,0.11  ] and [-4', '8')
```

There were a lot of questions about this one, since the question text is not really clear. Many students wondered whether the regex should pick up the two numbers of one interval, or pick up both intervals in a single go (i.e. four numbers). Of course the structure of the suggested regex clearly points to the first interpretation, but we should still show some flexibility here. Again, I suggest to give a maximum of two points for the explanation and three for the solution:

- *Two points for any reasonable explanation that gets the point of greedy vs non-greedy. One point for other explanations that are close but not* *quite get the main point.*

- *Three points to any working (or very close) solution that picks up the two numbers in a single interval, or the four numbers of the two intervals. Two points for solutions that are look ok but are not quite there. One point for solutions that don't really work but catches some of the idea, for instance a regex that only works for the integer intervals.*

3. Monte Carlo simulation (10 points)

Write a Python script that applies Monte Carlo simulation to estimate the probability of getting at least three 6's when you throw 10 dice. (Monte Carlo simulations are performed by letting the program "throw dice" a large number of times, and the count the outcomes of interest.) All the dice are regular 6-sided with sides from 1-6. You can set directly how many experiments to run, or let this be the choice of the user.

Remember that random.randint(n,m) returns an random integer in the range from and including n, up to and including m. Include necessary imports.

This is a relatively simple script, and very close to some of the exercises given through the fall. One possible solution is shown below. It does not matter if the task is solved using NumPy vectorization or not. The script is relatively simple, so scoring is a little strict

10 points: A working script with no bugs and no awkward programming, i.e. a reasonably elegant and compact solution

8-9: Minor bugs such as missing imports, or slightly cumbersome programming style such as redundant lines etc

6-7: Get the idea of the two loops and the bulk of the algorithm right, but quite a few bugs and errors

4-5: Get the main idea of looping and counting outcomes to compute the probability, but major flaws

1-3: Far from the right solution, but pieces of the code are on track

```
import random

N = 1000

ndice = 10

nsix = 3

M=0

for i in range(N):
    sixes = 0
```

```
        for j in range(ndice):
                 r = random.randint(1, 6)
                if r == 6:
                        three += 1
        if three >= nthree:
                M += 1
p = float(M)/N
print 'probability:', p
```

4: Classes (5 points)

Write a class (class Planet) that includes methods returning area and circumference of planet. Each instance of the class should be initialized with the planet's radius. Here are some useful geometric formulae for spheres:

Area = $4\pi r^2$
Circumference = $2\pi r$

Use of the class should look like this:

```
>>> jorda = Planet(6371)  #radius of the earth (jorda) is 6371 km
>>> omkrets = jorda.circumference()
>>> areal = jorda.area()
```

This is a very simple class programming exercise, with three simple functions __init__, circumference and area. A key point is getting the init function right. A possible solution is listed below. Here's a suggestion for scoring:

5 points: absolutely everything correct (except possibly a very minor typo). Full score should be given regardless of using pi from math or simply 3.14

4 points: everything ok, minor typos and syntax errors

3 points: some errors, but the __init__ function needs to be there. For example, forgetting "self" as an argument to the functions, with everything else correct, should give three points

2 points: Major errors, incorrect init function etc

1 point: Completely missing init function, other major errors

```
from math import pi

class Planet:

        def __init__(self, R):

                self.R = R

        def area(self):

                return 4*pi*(self.R)**2

        def circumference(self):

                return 2*pi*self.R
```

5: List slicing (5 points)

A list a is created is created with the command

```
>>>
a=list([range(0,10),range(10,20),range(20,30),range(30,40),range(40,50)])
```

Write the result of the following commands:

```
>>> print a[0][:]
>>> print a[0][:3]
>>> print a[::2][:]
>>> print a[-1][-1]
>>> print a[:][-1]
```

There are two parts to this question. The first is figuring out what kind of list the first command gives. The second is figuring out the slicing. Some were unfamiliar with the first command, which is not so surprising since there is in fact a redundant "list" command in it. However, it should not be very difficult to figure out that the result is a list of lists. The actual print commands also strongly indicate that this is the outcome. The tricky part of the actual print commands is the major difference between for instance a[0] and a[::2]. With a[0] you suddenly have a single list, so a[0][0] is the first element. Using a[::2] you still have a two dimensional list, so adding a second index will actually pick out lists. The solution is listed below.

For scoring I suggest 1 point per correct answer, and 0.5 points per answer that is very slightly off. For instance, if someone forgets that range[0,10] will give a list that ends with 9 and not 10, and everything else is correct, this will give 2.5 points in total.

```
>>>
a=list([range(0,10),range(10,20),range(20,30),range(30,40),range(40,50)]
)
>>> print a[0][:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> print a[0][:3]
[0, 1, 2]
>>> print a[::2][:]
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [20, 21, 22, 23, 24, 25, 26, 27, 28, 29], [40, 41,
42, 43, 44, 45, 46, 47, 48, 49]]
>>> print a[-1][-1]
49
>>> print a[:][-1]
[40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
```

6. CSV file processing (10 points)

You were recently employed in Foomatic inc., and have been asked to correct an error in the company's large collaction of spreadsheets. The same error occurs in hundreds of spreadsheets, and correcting it manually will be too time consuming. Fortunately, Python has a package (csv) for for reading and writing files in the spreadsheet format CSV. Before you can start correcting errors you need to make a couple of functions to handle the files. Use the modules of the csv-package to make the following two functions.

- The first function takes the name of a csv-file as input, checks the format (dialect) of the file, reads the data and stores it as a list of dictionaries. The first row in the spreadsheet contains the column names. These will be the keys of the dictionaries. The remaining rows contain data that is to be stored; one row fills one dictionary.
  Use case example: We have a csv file "input_file.csv" containing the following data:

  ```
  Name, Profession, Phone
  Johnson, carpenter, 12345678
  Anderson, plumber, 87654321
  ```

  Use of the function should now result in the following output:

  ```
  >>> namelist = readcsv(”input_file.csv”)
  >>> namelist
  [{’Name’: ’Johnson’,’Profession’: ’carpenter’, ’Phone’: ’12345678’},
  {’Name’:’Anderson’, ’Profession’: ’plumber’: ’Phone’: ’87654321’}]
  ```

- The second function takes a list of dictionaries and a file name as input, and writes the contents of each dictionary as a line in the file, in csv-format. Applied to the example above, calling
  ```
  >>> writecsv(namelist,”output.csv”)
  ```

should write the contents of namelist to the file ”output.csv”, in the same format and order as in ”input_file.csv”.

Most of the solution here is fairly simple, since the enclosed documentation has a few examples that are very close to what we want. However, some details are a little tricky. The read function must use the class csv.Sniffer to figure out the dialect, and then the class csv.DictReader to read the contents into a dictionary. The write function must then use the class csv.DictWriter. The most (only) challenging part is the write function, which comes with a couple of challenges. The first is to get the column headers right. The second is the result of a slightly ambiguous question text, stating that the format of output should be equal to the format of input. I honestly did not think much of the dialect here, but many students made this assumption. This makes the entire question slightly more difficult, since the read function must store the dialect so it can be used in the write. We should be flexible on this part, and accept solutions that try to get the right dialect and also those that skip this part. A partly functioning solution is listed below. The dialect part does not function correctly for this input file, therefore the read function does not work, but solutions close to this suggestion should get full score.
Here's a suggested scoring scheme:
5 points for each function
readcsv: 2 points for identifying the right classes (sniffer and dictreader), 3 points for using them in more or less the right way, 4 points for solutions with very minor errors (for instance forgetting to reset the file object after sniffing the sample), 5 points for fully correct solutions or just minor typos.
Writecsv: 1 point for getting the class right, 2 points for initializing it correctly with the fieldnames, 3 points for getting both the functions writeheader, writerow, 4 points for using them more or less correctly, and 5 points for any solution that is close to working.

```
import csv
import sys

def readcsv(filename):
   f = open(filename, 'rt')
   dialect = csv.Sniffer().sniff(f.readline())
   f.seek(0)
   entries = []
   reader = csv.DictReader(f,dialect)
   for row in reader:
      entries.append(row)
   f.close()
   return entries
```

```
def writecsv(dictlist,filename):
    f = open(filename, 'wt')
    writer = csv.DictWriter(f, dictlist[0].keys())
    writer.writeheader()
    for row in dictlist:
        writer.writerow(row)
    f.close()
```