ECM1408 Programming for Science
Continuous Assessment 1

Date set: Monday 13th October, 2014
Hand-in date: **12:00 Friday 24th October, 2014**

This continuous assessment (CA) comprises 15% of the overall module assessment.

This is an **individual** exercise and your attention is drawn to the College and University guidelines on collaboration and plagiarism, which are available from the College website.

Note that both paper (BART) and electronic submissions are required.

This CA tests your knowledge of the programming in Python that we have covered in the first three weeks of term, particularly functions, loops, lists and conditionals.

Make sure that you lay your code out so that it is readable and you comment the code appropriately.

Exercises

1 Syntax errors

Identify the *syntax* errors in the following Python code, stating why each is an error.

```
1  define find-first(L, x):
2      """Return the index of the first
3         occurrence of x in L"
4      if not isinstance(L, list):
5           return FALSE
6      index = 0
7      for item in L
8          if item = x:
9             return index
10       index = index + 1
11
12 x = 10
13 y = -5
14 M = [1, 2, 'three', 4x + y, 'thr' + 'e'*2]
15
16 print[find_first(M, 'three')]
```

Line numbers on the left are merely to help you refer to the lines.

You should submit your answers on paper via BART. **[10 marks]**

2 Diagonally

Write a function `diagonal(text, right_to_left=False)` to print a string diagonally. The argument `right_to_left` should control whether the string `text` is printed from left to right or right to left. So, for example:

```
>>> diagonal("Diagonal text")
D
 i
  a
   g
    o
     n
      a
       l

          t
           e
            x
             t
>>> diagonal("Diagonal text", right_to_left=True)
             D
            i
           a
          g
         o
        n
       a
      l

   t
  e
 x
t
>>>
```

Use your function to print the word *slantwise* diagonally from left to right and from right to left.

Even if you know about indexing in Python, your answer should **not** use indexing. There are no marks for answers that do.

Your code should be in a file called `diagonal.py`. You should submit:

- The output from your program showing *slantwise* printed in both directions (paper via BART).
- Hardcopies of your program `diagonal.py` (paper via BART).
- Your program in a file `diagonal.py` (electronic submission).

[15 marks]

3 Making change

Suppose you buy an item costing £6.34 and pay with a £20 note. The change you are owed is £13.66 and good shopkeepers will return the change using the as many high-value coins or notes as possible. For example, it is better to give change as 1 × £10 note, 1 × £2 coin, 1 × £1 coin, 1 × 50p coin, 1 × 10p coin, 1 × 5p coin and a 1p coin rather than 13 × £1 coins, 6 × 10p coins and 3 × 2p coins (and much better than 1366 × 1p coins!).

Suppose that `coins` is a list of the denominations (in pennies) of coins and notes that is available. For example

```
coins = [1000, 500, 100, 50, 10, 5, 2, 1]
```

means that £10, £5 notes and £1, 50p, 10p 5p, 2p, and 1p coins are available (but no 20p coins).

Write a function `make_change(amount, coins)` that will print the how many of each denomination note or coin should be given if the value of the change is `amount` (in pennies). If change cannot be made using the available coins, your function should print "Sorry. Change not available".

You may assume that there are as many coins as necessary of each of the available denominations. Also you may assume that the available denominations are listed in descending order as above.

Use your function in a program to print what change should be given for the following combinations of amount and coins:

Amount	Available denominations
1366	[1000, 500, 100, 50, 20, 10, 5, 2, 1]
512	[1000, 500, 100, 50, 20, 10, 5, 2, 1]
9	[1000, 500, 100, 50, 20, 10, 5, 2, 1]
999	[1000, 500, 100, 20, 50, 10, 5, 2, 1]
1689	[1000, 500, 100, 50, 20, 10, 5, 2, 1]
0	[1000, 500, 100, 50, 20, 10, 5, 2, 1]
1689	[1000, 500, 100, 50, 10, 2, 1]
89	[1000, 500, 100, 50, 20, 10, 5, 2]

Put your code in a file named `change.py`.

Hints

- Recall that the `//` operator performs integer division. The `%` operator, known as the *modulus* operator, finds the remainder when the lefthand argument is divided by the righthand argument. For example:

```
>>> 14 // 5
2
>>> 14 % 5
4
>>> 15 % 5
0
```

You should submit:

- Hardcopy of the output of your program showing what change should be given for the various amounts (paper via BART).
- Hardcopy of your program `change.py` (paper via BART).
- Your program in a file `change.py` (electronic submission).

[25 marks]

4 Ring of stars

The aim of this question is to draw a ring of stars like the European Union flag using the turtle graphics in `exturtle`.
First write a function `star(turtle, x, y, points, R, r)` which uses turtle `turtle` to draw a star centred at (`x`, `y`). The `points` argument should specify the number of points the star has. The `R` and `r` arguments specify the outer and inner radii of the star respectively. The outer radius `R` is the distance from the centre of the star to the points and the inner radius `r` is the distance from the centre to the corners in between the points. The Appendix gives some more information on the geometry of stars and hints for drawing one.

Use your function to draw a row of stars with 5, 6, 7 and 8 points.

Write a second function `ring(turtle, cx, cy, Nstars, radius, points, R, r)` which draws a ring of `Nstars` stars, so that each star is a distance `radius` from the centre of the ring at (`cx`, `cy`). The `points` argument should specify the number of points for each star and the arguments `R` and `r` specify the outer and inner radii of each star. Your `ring` function should use your `star` function to draw each star.

Use your `ring` function to draw an EU flag with 12 5-pointed stars.

Your program should be in a file name `ring.py`.

Hints:

- See Appendix for a possible way to draw a single star.
- It will be easier to use the `goto(turtle, x, y)` function rather than the `left()`, `right()`, `forward()`, `backward()` functions. The `goto(turtle, x, y)` makes the turtle draw (if the pen is down) or move (if it's up) from its current location to `(x, y)`.
- You can get the *cos* and *sin* functions from the `math` module with

  ```
  from math import cos, sin
  ```

  Note that `cos` and `sin` expect angles in radians, not degrees.
- If you want to fill your stars with a colour (it's not mandatory), you can use the `begin_fill(turtle)` and `end_fill(turtle)` functions. Find out how to use these using the Python help system to read their doc-strings:

  ```
  >>> from exturtle import *
  >>> help(begin_fill)
  ```

You should submit:

- Screenshots of your row of stars and the ring of stars (paper via BART).
- Hardcopy of your program `ring.py` (paper via BART).
- Your program in a file `ring.py` (electronic submission).

[25 marks]

5 Train Passengers

This problem is taken from the UK and Ireland Programming Contest held this month.

The Nordic Company of Passing Carriages is losing money at an alarming rate because most of their trains are empty. However, on some lines the passengers are complaining that they cannot fit in the carriages and have to wait for the next train!

The authorities want to fix this situation. They asked their station masters to write down, for a given train, how many people left the train at their station, how many went in, and how many had to wait. Then they hired your company of highly paid consultants to assign properly sized trains to their routes.

You just received the measurements for a train, but before feeding them to your optimisation algorithm you remembered that they were collected on a snowy day, so any sensible station master would have preferred to stay inside their cabin and make up the numbers instead of going outside and counting.

Verify your hunch by checking whether the input is inconsistent, i.e., at every time the number of people in the train did not exceed the capacity nor was below 0 and no passenger waited in vain. The train should start and finish the journey empty, in particular passengers should not wait for the train at the last station.

Information about a train is contained in a file. The first line contains two integers C and n ($2 \leq n \leq 100$), the total capacity and the number of stations the train stops in. The next n lines contain three integers each, the number of people that left the train, entered the train, and had to stay at a station. Lines are given in the same order as the train visits each station.

Write a program to read the file and output a single line containing one word: `possible` if the measurements are consistent, `impossible` otherwise.

Here are some sample inputs and the corresponding output. These files are available on the ELE site.

`train1.txt`	`train2.txt`	`train3.txt`	`train4.txt`

`1 2`
`0 1 1`
`1 0 0`
	`1 2`
	`1 0 0`
	`0 1 0`
		`1 2`
		`0 1 0`
		`1 0 1`
			`1 2`
			`0 1 1`
			`0 0 0`

`possible`	`impossible`	`impossible`	`impossible`

Your program will be tested on these data files and some more extensive files. Make sure that it outputs only a single line containing either `possible` or `impossible`.

Hint

- The skeleton file `trains.py` and `read_data.py` on the ELE site shows you how to read the data from a train data file into a list.
- You may find it helpful to write a function that updates the number of people on the train after it visits each station.

You should submit:

- Hardcopy of your program `trains.py` (paper via BART). The initial lines of your program should be comments that explain how your algorithm works.
- Your program in a file `trains.py` (electronic submission). Do not modify the `read_data.py` file and there is no need to submit it.

[25 marks]

[Total 100 marks]

Submitting your work

The CA requires both paper and electronic submissions.

Paper You should submit the answer to question 1 and paper copies of the code and any output for **all** the other questions to the Harrison Student Services Office in the foyer of the Harrison Building by the deadline of **12:00 Friday 24th October, 2014**. Markers will not be able to give feedback if you do not submit hardcopies of your code and marks will be deducted if you fail to do so.

Paper submissions should have the BART cover sheet securely attached to the front and should be anonymous (that is, the marker should not be able to tell you are from the submission). If this is the first time you have used BART, please make sure that you understand the procedure beforehand and leave plenty of time as there are often queues close to the deadline.

Where you are asked for paper copies of the output of your code, please cut and paste the output from the terminal rather than taking a screenshot, because the screenshot is often illegible after printing.

Electronic You should submit the files containing the code for each question via the electronic submission system at `http://empslocal.ex.ac.uk/submit/`. Make sure that your code is in files with the names specified in the questions. Then use `zip` or `rar` or `tar` to compress these into a single file, and upload this file using the submit system. You must do this by the deadline.

You will be sent an email by the submit system asking you to confirm your submission by following a link. Your submission is not confirmed until you do this. It is best to do it straightaway, but there is a few hours leeway after the deadline has passed. It is possible to unsubmit and resubmit electronic coursework — follow the instructions on the submission website.

Marking criteria

Work will be marked against the following criteria. Although it varies a bit from question to question they all have approximately equal weight.

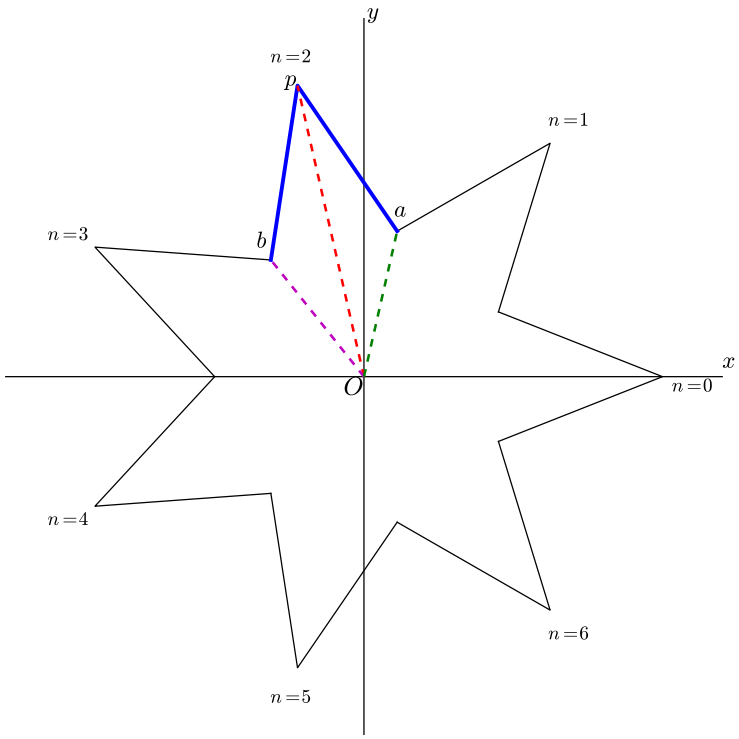
- **Does your algorithm correctly solve the problem?**
  In most of these exercises the algorithm has been described in the question, but not always in complete detail and some decisions are left to you.
- **Does the code correctly implement the algorithm?**
  Have you written correct code?

- **Is the code syntactically correct?**
  Is your program a legal Python program regardless of whether it implements the algorithm?
- **Is the code beautiful or ugly?**
  Is the implementation clear and efficient or is it unclear and inefficient? Is the code well structured? Have you made good use of functions?
- **Is the code well laid out and commented?**
  Is there a comment describing what the code does? Are the comments describing the major portions of the code or particularly tricky bits? Do functions have a docstring? Although Python insists that you use indentation to show the structure of your code, have you used space to make the code clear to human readers?

There are 10% penalties for:

- Not submitting hardcopies of your programs.
- Not naming files as instructed in the questions.

Appendix: Geometry of a star



The diagram shows one possible way of constructing a star; you are welcome to use other constructions if you prefer. The points of the star, such as those labelled $n = 0$, $n = 1$, $n = 2$, etc are all a distance R from the origin, O, so that the dashed red line has length R. The inner "corners", such as those labelled a and b, are a radius r from the origin, so that dashed green and magenta lines have length r.

Recall that if a point is a distance r from the origin and a line from it to the origin makes an angle θ with the x axis, then the coordinates of the point are $(x, y) = (r\cos(\theta), r\sin(\theta))$. Therefore, the coordinates of the points and corners of the star can be found if we know the relevant angles.

It's easiest to label the points starting from 0 and going up to $P - 1$, where P is the number of points in the star ($P = 7$ in the diagram). Focusing on the $n = 2$ point, outlined in blue, we can see that the angle between the x axis and the red dashed line to p is $n \times 2\pi/P$ because the angle between the points is $2\pi/P$. For simplicity let $\Delta = 2\pi/P$. Thus the coordinates of the nth point are

$$(R\cos(\Delta n), R\sin(\Delta n))$$

Notice also that the corners are halfway between the points, so the angle between the x axis and the corner on the clockwise side of the point (a is on the clockwise side of p) is $\frac{2\pi n}{P} - \frac{2\pi}{2P} = (n - \frac{1}{2})\Delta$. Thus the coordinates of corners like a are:

$$\big(r\cos((n - \tfrac{1}{2})\Delta), r\sin((n - \tfrac{1}{2})\Delta)\big)$$

By similar reasoning the coordinates of the corners on the anti-clockwise side of each point (like corner b for point p) are:

$$\big(r\cos((n + \tfrac{1}{2})\Delta), r\sin((n + \tfrac{1}{2})\Delta)\big)$$

Thus one way to draw the star is to have a loop that is traversed once for each point. The body of the loop draws the nth point, starting at the corner like a, drawing to the point like p and then drawing to the corner like b; the next iteration of the loop will draw the two lines for the next point, and so on.