CSC1010H Tutorial 4 – Data Structures and Problem Solving

Introduction

Learning outcomes

Skills

- Be able to apply a variety of strategies and techniques to problem solving, including divide-and-conquer (breaking the problem into sub problems), and simplification (solving a simplified version first).
- Construct and reason about solutions to programming problems requiring the use of data structures.

Knowledge

- The features of Python lists and dictionaries, and the form of expressions used to construct, deconstruct, and access such structures.

Question 1

Create a program called *tracking.py* that will be used to log information on endangered animals being tracked for a conservation study. The program will prompt the user for animal identity codes and then iteratively respond to the following commands:

'`print`' – print the last known location of each animal.

'`log` *`<animal id> <x ordinate> <y ordinate>`*' – record that the animal with the given identity code was recently observed at the given *(x ordinate, y ordinate)* location.

'`quit`' – terminate the program.

Here is a sample of expected program behaviour:

```
Please enter the animal identity codes. (Press return when done.)
Animal no. 1:
3e46fa
Animal no. 2:
432639
Animal no. 3:
fe9810
Animal no. 4:

Commands: print, log <animal id> <x ord> <y ord>, quit.
>print
Animal 3e46fa cannot be located.
Animal 432639 cannot be located.
Animal fe9810 cannot be located.
>log 432639 340 462
>log fe9810 001 151
>log 432639 395 507
>print
Animal 3e46fa cannot be located.
Animal 432639 last seen at (395, 507).
Animal fe9810 last seen at (1, 151).
```

```
>log 3e46fa 2381 viey
The ordinates should be integers.
log <id> <x ord> <y ord>
>log 3e46fa 2381 81
>pront
Could not interpret command.
Commands: print, log <animal id> <x ord> <y ord>, quit.
>print
Animal 3e46fa last seen at (2381, 81).
Animal 432639 last seen at (395, 507).
Animal fe9810 last seen at (1, 151).
>quit
```

How to solve it?

How will you answer the question? Can we make some suggestions?

The program has a well-defined structure that lends itself to the problem solving strategies of simplification, and divide and conquer.

Applying divide and conquer, here are some of the sub problems that must be tackled:

- Write code to read in a series of identity codes.
- Write code that repeatedly asks the user to enter a command until that command is 'quit'.
- Write code that can split a command up into parts.
- Write code that figures out if a command is 'print' or 'log', or unrecognisable.
- Write code that creates a data structure suitable for storing last known locations of animals.
- Write code that, given an animal identity code and a location, updates the data structure.
- Write code that can print the last known location of an animal.
- Write code that print all last known locations.

We've euphemistically used terms such as 'series' and 'data structure'. Another sub problem is to decide how identity codes and locations should be stored in the program: list(s) and/or dictionary? A list could store a collection of identity codes, a dictionary could store identity codes and locations, a tuple could be used to represent a location. We might be able to do everything just using a dictionary.

Based on these sub problems, we can apply a simplification strategy. We can start with a program that solves one sub problem and then gradually add in code that solves other sub problems, eventually arriving at an overall solution.

A useful tip for simplifications is that you can embed data in your program. For example, say we want to focus on writing code that goes through a dictionary of identity codes and locations, and prints them out, we can start with code like this:

```
# Print a dictionary
def main():
    # Fake data dictionary for our code:
    locations = {'0d33a': (1, 10), '872343': (0, 23)}

    # Put code here to go through dictionary and print out details.

main()
```

HINTS:

- Main program loop:

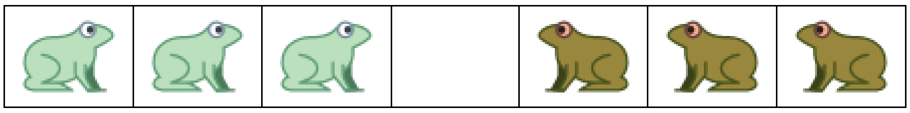
```
# get command
while not command=='quit':
      # process command.
      # get command.
```

- Splitting a string up into a list of parts:

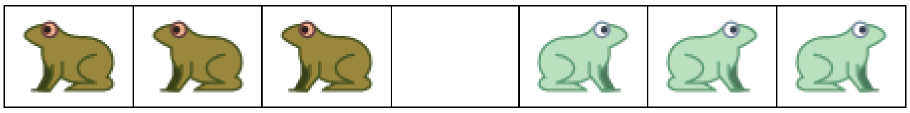
```
>>>'log e3232 32 42'.split()
['log', 'e3232', '32', '42']
>>>'print'.split()
['print']
```

Question 2

'Frogs and Toads' is a simple puzzle game involving a row of squares, the leftmost of which contain frogs and the rightmost of which contain toads e.g.


(Icon courtesy of icons8.com.)

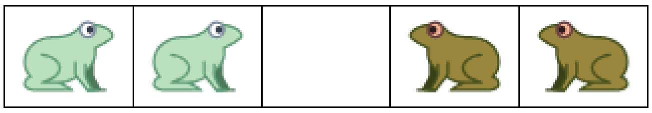
There is an empty square between the frogs and the toads. The challenge is to move the creatures, one at a time such that the positions are reversed (all the frogs on the right, and all the toads on the left) e.g.

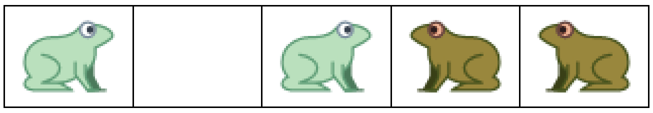


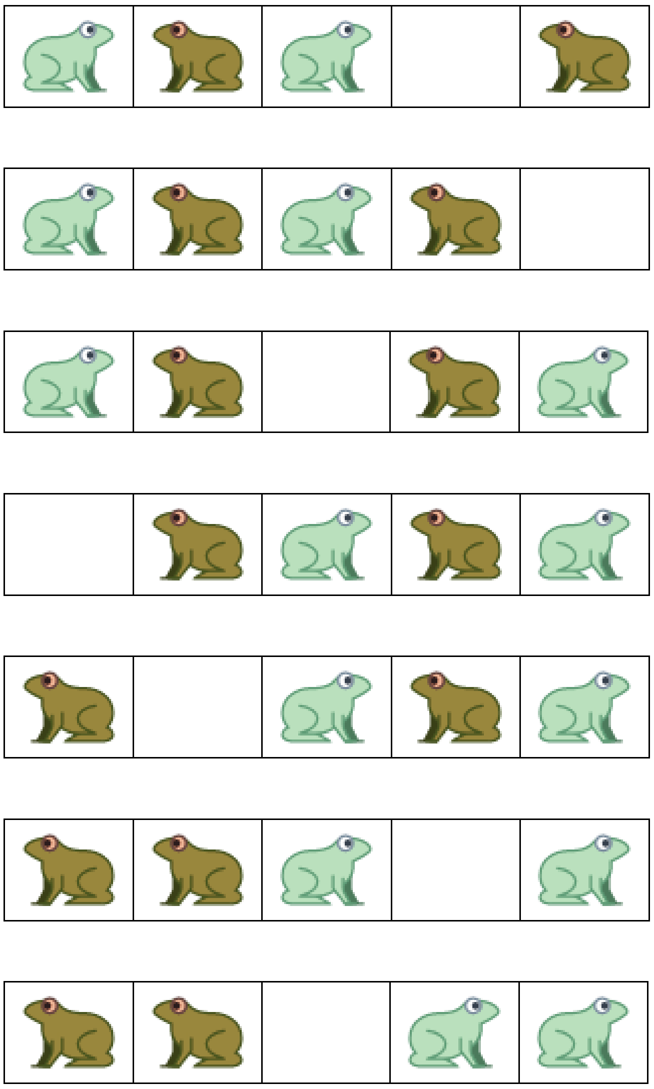
Here are the rules:

- There is only ever one empty square.
- A square is either the empty square, or it contains exactly one frog or toad.
- A frog or toad facing the empty square can hop into it.
- A frog or toad can hop over another frog or toad if the empty square is on the other side.
- A frog can only hop from left to right.
- A toad can only hop from right to left.

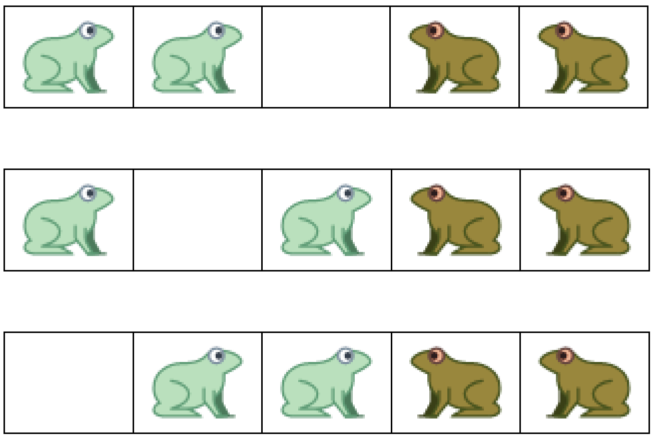
Here is a sample game sequence involving two frogs and two toads:







The game can be lost by arriving at a state where nothing can move e.g.



Write a program called frogsandtoads.*py* that allows a person to play a game of Frogs and Toads using the computer. The program should ask for the number of frogs and the number of toads, and then repeatedly print the puzzle state and ask the user for the index of a frog or toad to move.

The game ends when either the user enters ‘quit’, or succeeds in moving all the frogs to the right and all the toads to the left, or arrives at a point where no more moves can be made.

Here’s a sample of expected program behaviour:

```
Enter the number of frogs:
1
Enter the number of toads:
2
  1   2   3   4
Frog    ToadToad
>3
  1   2   3   4
FrogToad    Toad
>1
  1   2   3   4
    ToadFrogToad
>2
  1   2   3   4
Toad    FrogToad
>4
  1   2   3   4
ToadToadFrog
>3
  1   2   3   4
ToadToad    Frog
Congratulations, you've won!
```

And here’s a sample in which the player loses:

```
Enter the number of frogs:
2
Enter the number of toads:
3
  1   2   3   4   5   6
FrogFrog    ToadToadToad
>4
  1   2   3   4   5   6
FrogFrogToad    ToadToad
>5
  1   2   3   4   5   6
FrogFrogToadToad    Toad
>6
  1   2   3   4   5   6
FrogFrogToadToadToad
Sorry, you've lost.
```

And finally, here’s a sample in which the player quits:

```
Enter the number of frogs:
5
Enter the number of toads:
```

```
8
  1    2    3    4    5    6    7    8    9   10   11   12   13   14
FrogFrogFrogFrogFrog     ToadToadToadToadToadToadToadToad
>7
  1    2    3    4    5    6    7    8    9   10   11   12   13   14
FrogFrogFrogFrogFrogToad     ToadToadToadToadToadToadToad
>quit
```

How to solve it?

As with question 1, we can divide, simplify, conquer. It's your turn to decide how to apply the strategies; to decide which sub problems and simplifications will get you to your goal.

To get you started, however, here's one way in which you can set up the initial puzzle:

```
num_frogs = int(input('Enter the number of frogs:\n'))
num_toads = int(input('Enter the number of toads:\n'))
state = ['Frog']*num_frogs+['']+['Toad']*num_toads
```

The idea is that the sequence of puzzle squares is represented using a list of strings.

- If a square contains a frog then the corresponding location in the list contains the string `'Frog'`.
- If a square contains a toad then the corresponding location in the list contains the string `'Toad'`.
- An empty string, `''`, is stored in the location corresponding to the empty square.

To figure out what the third statement does, try entering it in the Python shell (the bottom panel of the Wing 101 IDE) and experimenting e.g.

```
>>> num_frogs = 3
>>> num_toads = 2
>>> state = ['Frog']*num_frogs+['']+['Toad']*num_toads
>>> print(state)
['Frog', 'Frog', 'Frog', '', 'Toad', 'Toad']
>>> state[3] = state[2]
>>> print(state)
['Frog', 'Frog', 'Frog', 'Frog', 'Toad', 'Toad']
>>> state[2] = ''
>>> print(state)
['Frog', 'Frog', '', 'Frog', 'Toad', 'Toad']
>>>
```

Question 3

Using the Problem Solving techniques dealt with in class, solve the following two problems. Show any working, calculations, or visualisations you have used to get your result. Save your solutions in an MSWord document called Question3Solutions.doc or docx.

a) The UCT Frigo café commissioned a small survey of the muffin preferences of students majoring in computer science, physics and mathematics.

   Unfortunately, the survey report got damaged. (Someone spilt coffee on it.) The staff have managed to piece together some of the results. It looks as if it might be possible to deduce the rest. This is what we know:

"Overall, one hundred and sixty students were surveyed, and of these, thirty eight were studying Physics.

The students were asked if they preferred either banana, chocolate, or chocolate chip.

The majority of students, one hundred and eleven of them, preferred one of the chocolate flavours. Chocolate chip in particular was the most popular flavour amongst computer scientists and mathematicians. Forty one mathematicians and thirty three computer scientists preferred it.

Curiously, the second most popular flavour for computer scientists and mathematicians was also the same - twenty one computer scientists and nineteen mathematicians preferred banana.

None of the mathematicians liked chocolate flavour. For Physicists, however, this was the most popular. Twenty two preferred it."

Can you help piece together the missing information?

- How many students were surveyed from each major?
- For each flavour of muffin, overall, how many expressed a preference?
- What's the breakdown of preferences by major? (That is, how many physicists preferred banana, how many preferred chocolate, …, how many computer scientists preferred banana, …, etc.)

b) A remarkable species of mongoose has been discovered, remarkable because it is extremely long-lived and because it has strangely regimented breeding behaviour:

- A new born mongoose takes three months to reach sexual maturity.
- On reaching sexual maturity, a mongoose immediately finds a mate.
- A mongoose mates for life.
- A breeding pair of mongooses (mongeese? mongoosia?) gives birth to a new pair of mongooses (exactly one male and one female) every month after reaching sexual maturity.

Say that we have a new born pair of these mongooses, ignoring the risks of unnatural death and inbreeding, how do we express P(n), the population size in n months?

HINT: Work out P(n) for the first few cases, e.g. P(1) = 2.

Your solution should be a function in the form 'P(n)=????'.

Question 4

A great deal of Computer Science involves doing research, experimenting and familiarising yourself with new tools.

For students who have started CSC1010H in the first semester:

So far you've learnt the basics of Python Turtle. Now you're expected to learn more about Python Turtle. Find all the Python Turtle functions in the following way. From the Windows Start Menu find the Python 3.x installation. Open the Python Manuals and search for 'turtle module' and you should find the page with all the Python Turtle functions.

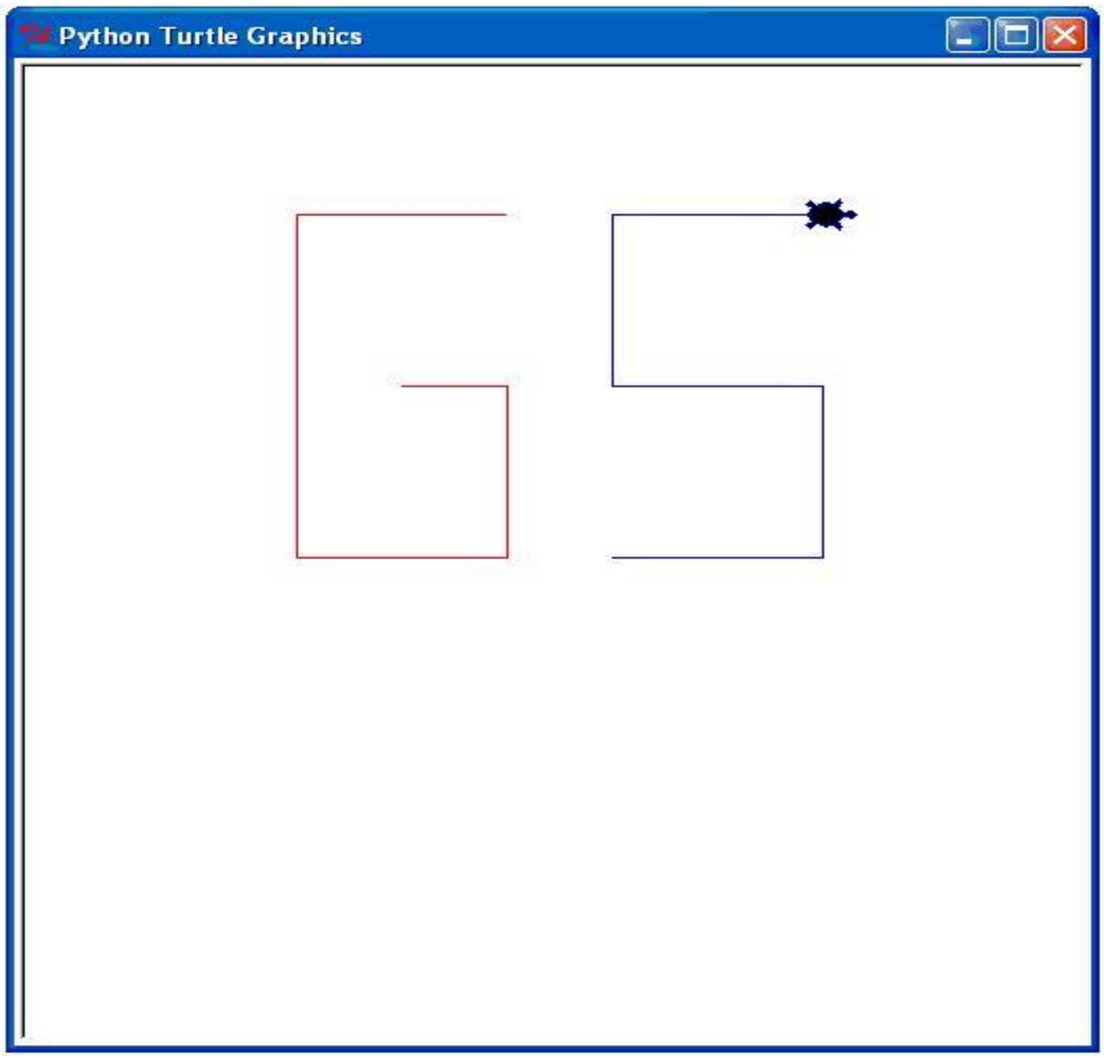
Learn how to use at least 5 new useful Turtle functions we haven't used in class or in the course notes so far. Use these, along with the functions you've already learnt about, to draw a picture of an actual turtle.

(Don't worry, it doesn't have to be completely accurate, a rough approximation will do!)

For students who have started CSC1010H in the second semester:

Python Turtle is a teaching tool for learning programming which has been used in the course in the first semester. Familiarise yourself with how to use it by going through the notes PythonTurtle1, PythonTurtle2 and PythonTurtle3 in the Resources>LectureNotes section on Vula, and do the following exercise.

Write a program using Python Turtle in a file called initials.py which draws your own initials, i.e. the first character of your name and the first character of your surname, using block letters. If they are the same character use the first two characters of your name. Use a different colour for each character, other than black. Also horizontally centre the letters and ensure that they are 200 steps high. Here is an example of what it should look like:



k

Marking and Submission

Submit the *tracking.py*, *frogsandtoads.py*, *Question3Solutions.doc*, and Python turtle program files contained within a single .ZIP folder to the automatic marker. The zipped folder should have the following naming convention:

yourstudentnumber.zip

Marking Guide

Question 1 Tracking	25
Question 2 Frogs and Toads	20
Question 3a Vida Café	12
Question 3b Mongeese	13
Question 4 Python Turtle	30
Total	**100**

Note that questions 3 and 4 will be marked manually.