FIT1008 Introduction to Computer Science
Practical Session 2

Semester 2, 2014

Objectives of this practical session

- To investigate different approaches for debugging code.
- To be able to write a menu to test different functions.
- To be able to understand lists of lists.

***Task 1** [3 marks]*

Consider the following Python program. This program should read in a positive integer, and print out all the positive primes less than this integer. However, the program contains some bugs. Your task is to find the bugs, state what they are and fix them.
Some approaches you might like to consider are:

- Try some test data.
- Include print statements.
- Work through the program by hand for a small example.
- Run it through Python tutor: `http://www.pythontutor.com/visualize.html`
- Use a debugger.

```
def is_prime(n):
    k = 3

    if (n = 2):
       flag = true
    elif (n % 2 == 1):
       flag = false
    else:
       while (k*k < n):
          if (n % k == 0):
             flag = false
             break

    return flag

n = int(input('Please␣enter␣positive␣integer:␣'))

for i in range(n):
    if (is_prime(i)):
       print(i)
```

Testing

For Tasks 2 and 3, you are required to write down *at least two test cases* per menu command, to show for example that your program can handle both valid and invalid inputs.

Task 2 [3 marks]

Later in this unit you will be implementing various data structures. A useful approach for testing your implementation is to use a menu, which allows you to test each of the functionalities of the data structure. Consider the following code which uses a menu to append items to a list, print a list out, sort a list, and to quit.

```
def print_menu():
    print('\nMenu:')
    print('1. append')
    print('2. sort')
    print('3. print')
    print('4. quit')

my_list = []
quit = False
input_line = None

while not quit:
    print_menu()

    command = int(input("\nEnter command: "))

    if command == 1:
        item = input("Item? ")
        my_list.append(item)
    elif command == 2:
        my_list.sort()
    elif command == 3:
        print(my_list)
    elif command == 4:
        quit = True
```

Extend the above code so a user can perform the following commands on my_list using the menu:

- clear: which removes all the items in the list.
- reverse: which reverses the order of the items in the list.
- pop: which removes the last item of the list, and prints it.
- size: which prints the length of the list
- insert: which inserts an item before a given position in the list.
- find: which prints the first index position in the list of a given value, or prints False if the value is not in the list.

Task 3 [4 marks]

Consider the puzzle of sliding blocks below. Any block adjacent (*left*, *right*, *above* or *below*) the blank spot can be moved into the position of the blank spot. No diagonal moves are allowed.
The object of the puzzle is can you rearrange the numbers into various patterns, e.g., can you put the numbers in reverse order or if the numbers have been jumbled up and you put them back into the standard configuration (see figure below).

The object of this task is to write a Python program to implement a version of the puzzle. We will represent the puzzle in terms of a lists of list, and use 'X' to mark the blank spot.

```
puzzle = [['␣1', '␣2', '␣3', '␣4'],
          ['␣5', '␣6', '␣7', '␣8'],
          ['␣9', '10', '11', '12'],
          ['13', '14', '15', '␣X']]
```

Write a Python program that uses a menu and allows the user to move the blank square *up*, *down*, *left*, and *right*. At each step the program should display the current state of the puzzle.

Advanced Question [Bonus 2 marks]

A given configuration of the puzzle is solvable if and only if the number of swaps to get it back into the standard configuration plus the city-distance (number of rows plus the number of columns) of the empty square from the lower right corner is even. *Please note that these swaps are not just the standard moves, but can be of any two squares in the puzzle.*
Write a Python program to allows the user to enter their own configuration of the puzzle, and informs the user whether or not it is possible to put it back into the standard configuration.

Hall of Fame

There are many Python modules available that you will find extremely useful. One such module is `random`. Using this module extend the program you wrote for **Task 3** so that the user has the option of starting the puzzle in a random solvable configuration. *A new random configuration should be generated every time the user chooses this option.*