Term Exam 1: Wednesday February 10, 2016
Instructor : Dr. Osmar Zaïane or Dr. Anup Basu

Student Name :_____

Student ID :_____

- Do not open this exam until you are instructed to do so. Read the instructions.
- The duration of the exam is 50 minutes.
- There are 4 sections worth a total of 100 points.
- This midterm exam will count for 15% of the overall course grade.
- Read all questions carefully. Do not read in diagonal. You may miss things.
- Use a pen not a pencil. You can use a pencil but then can't challenge the marking.
- For full grades, answer all parts of all questions.
- Be concise and give clear and legible answers.
- Non-legible answers will not be marked.
- Cheating is a serious offence in the code of student behaviour.
- No books, notes, phones, or other aids are permitted during the exam.
- Good luck!

Section 1	Section 2	Section 3	Section 4	**Total**

Section 1: Tracing and Algorithm Complexity [25 points]

Consider the following function which receives an Integer and prints a number of lists. Trace this python program to answer the following questions.

```
def myFunction(n):
    row = []
    for i in range(n):
       for j in range(n):
           if (i==j):
              row.append(1)
           else:
               row.append(0)
        print(row)
     return
```

What would be the output of **`myFunction(3)`**
You can use the back of the sheet as draft.

Evaluate the big-O time complexity of this function myFunction assuming the input is N. Justify your answer.

Big-O time complexity:

Justify your answer:

The purpose of the function above was to write the IDENTITY MATRIX, which has 1s on the diagonal and 0 elsewhere. In case it does not produce the desired result, can you move 1 line of code to a different location to achieve this?
(Just circle the line of code and draw an arrow to indicate where it should be moved.)
The result should be in this format

. . .
. . .
. . .

```
def myFunction(n):

    Row = []

    for i in range(n):

       for j in range(n):

           if (i==j):

              Row.append(1)

           else:

               Row.append(0)

       print(Row)

    return
```

Section 2: Multiple choice [30 points]

1- What is the output of the following python statement:

```
print([i*i for i in range(4)])
```

- ☐ a. 0, 1, 4, 9
- ☐ b. 1 2 3 4
- ☐ c. [0, 1, 4, 9]
- ☐ d. 0*0 1*1 2*2 3*3

2- What would be the values of L1 and L2 after these statements?

```
L1=[1,2,3]
L2=L1
L1.append(4)
```

- ☐ a. L1== [1,2,3] L2== [1,2,3,4]
- ☐ b. L1== [1,2,3,4] L2== [1,2,3,4]
- ☐ c. L1== [1,2,3,4] L2== [1,2,3]
- ☐ d. L1== [1,2,3] L2== [1,2,3]

3- What is the output of the following python code?

```
v1=50
v2=v1
v1=v1/10
print (v1, v2)
```

- ☐ a. 5, 50
- ☐ b. 50, 5.0
- ☐ c. 5.0, 50
- ☐ d. 50 50

4- What does the following python statement produce?

```
"a,b,,,,c".split(',') produces
```

- ☐ a. raises an exception because there are several commas next to each other
- ☐ b. ['a', 'b', 'c',]
- ☐ c. ['a', 'b', '', '', '', 'c']
- ☐ d. ['a', 'b', '', 'c']

5- What would be the value of x after the following statements?

```
T= (20, True, "Hat", [1,2,3], 5)
x=T.pop()
```

- ☐ a. 20
- ☐ b. no value. pop() would raise an exception
- ☐ c. 5
- ☐ d. True

6- What is the complexity of the function s?

```
def s(n):
    return n*(n+1)*(n+2)/2
```

- ☐ a. linear time
- ☐ b. constant time
- ☐ c. logarithmic time
- ☐ d. polynomial time

7- What would be returned with the following python code?

```
def s(n):
    return n*(n-1)/2

s(10)
```

- ☐ a. would generate an error
- ☐ b. 45
- ☐ c. would loop indefinitely because recursion lacks a stop condition
- ☐ d. 45.0

8- The complexity of the algorithm to compute x^n using the divide and conquer strategy is $\log_2(n)$ because:

- ☐ a. we divide and multiply n times
- ☐ b. we only consider the top right corner of the matrix
- ☐ c. we generate a tree of multiplication
- ☐ d. the number of times we need to divide n by 2 to get 1 is $\log_2(n)$

9- An algorithm is said to be polynomial for which of these complexities?

- ☐ a. $O(n^2)$
- ☐ b. $O(3^n)$
- ☐ c. $O(\log_2 \log_2 n)$
- ☐ d. $O(n \log_2(n))$

10- The postfix notation of the following infix expression is:

3 + 6 * 7 - (2 - 4) * 3 + 2

- ☐ a. 3 6 7 * 2 4 - 3 * - + 2 +
- ☐ b. 3 6 7 2 4 3 2 + * - - * +
- ☐ c. - * + 3 6 7 * - 2 4 + 3 2
- ☐ d. - * + * - + 3 6 7 2 4 3 2

Section 3: Exceptions [15 points]

1- a-We would like the output of the following function to handle all cases that can occur when we divide by 0: (i) If the Numerator is greater than 0, we want to print "+Infinity"; (ii) If the Numerator is less than 0, we want to print "-Infinity"; and (iii) If the Numerator is equal to 0, we want to print "UNDEFINED: Zero divided by Zero". Fill in the missing code below to make this happen.

```
def DivZERO(a,b):
    try:
        c = a/b
        print(c)
    except ZeroDivisionError:




    return
```

2- a- During the execution of the following code no exception is raised. What are the executed statements?

```
try
    Statement 1
    Statement 2
except ValueError:
    Statement 3
except Exception:
    Statement 4
else:
    Statement 5
finally:
    Statement 6
```

Section 4: Python programming [30 points]

1- Write the code in python to read as input the character Y or N for the question "Do you want to play again? (Y/N)". The input should be validated.
Hint: You need to iterate until a valid input (either Y or N) is entered. The user should be able to enter either uppercase or lowercase. No exception handling required.

2- Given the following algorithm, write the equivalent code in python. No exception handling required.

```
open file records.txt for reading
read each line
  strip the line
  id, name ← split line on ;
  display id and name
close file
```

3- In Lab 4, you were given a python file maze.py in which 2 classes were defined: **Maze** and **MazeSquare**. The constructor of Maze read a file containing the definition of a maze. Moreover the class Maze provided the method **get_start_square()** that returned the starting square of class MazeSquare. It also provided a method **is_finish_square(*square*)** that returned True if *square* (of class MazeSquare) is the finish square of the maze. The MazeSquare class represents a single square in the maze and provides a method **get_legal_moves()** that returns a list of MazeSquare squares that are legal moves from the square this method was invoked.

Complete the following python program based on the following algorithm that uses a stack to test whether a maze is solvable. (lines with . at the start need to be completed)

1-Add the start square to the stack
2-Repeat the following as long as the stack is not empty
2.1-Pop a square off the stack and make it the current square
2.2-If the current square is the finish square then solution found
2.3-Otherwise get the squares which can be moved to from the current square and add them to the stack

```
from maze import Maze
from maze import MazeSquare
from stack import Stack

# This function returns True if the maze has a solution.
def test_maze(maze):
    stack = Stack()
.


.   while
.       current_square =

        if maze.is_finish_square(current_square):
            return True

.       for
.




    return False

print("Is it solvable?", test_maze(Maze("mazefile.txt")))
```