Data Structures & Algorithms(AI) 2020/2021 Semester 2 Mid Semester Test

Section A: MCQ (20 marks)

1. B
2. C
3. C
4. A
5. B
6. B
7. D
8. B
9. C
10. B

Section B: Programming (80 marks)

1. Consider the implementation of a Stack class together with an associated main program and output as provided in Appendix A. You are to complete the below class Stack2 that is derived from class Stack. You must base your implementation on the output of the main program as provided below.

Main program using class Stack2

```
s1 = Stack2('#', '*')
s1.push(100)
s1.push(200)
print('Content of stack =',s1)
s2 = Stack2('[', ']')
for i in range(1,6):
    s2.push(i)
print('Content of stack =',s2)
print('Item at top=',s2.get())
while not s2.isEmpty():
    print(s2.pop())
    print(s2)
```

Output of the Main Program using class Stack2

```
Content of stack = #200-100*
Content of stack = [5-4-3-2-1]
Item at top= 5
5
[4-3-2-1]
```

```
4
[3-2-1]
3
[2-1] 2 [1] 1
[]
```

Create Stack2 class

```
class Stack2(Stack):
    def __init__(self, left, right):
        # 1(a): Complete your code implementation here
    '''
        super().__init__()
        self.left = left
        self.right = right
    '''

    def push(self, item):
        # 1(b): Complete your code implementation here
    '''
        self._Stack__list.insert(0, item)
    '''
    def pop(self):
        if super().isEmpty():
            return None
        else:
            return self._Stack__list.pop()

    def get(self):
        # 1(c): Complete your code implementation here
    '''
        if self.isEmpty():
            return None
        return self._Stack__list[0]
    '''
    def __str__(self):
        '''
            1. len
            2. i
            3. -1
            4. str(item) + '-'
            5. str(item)
            6. self.right
            7. output
        '''
        output = self.left
        for i in range(_(1)_, self._Stack__list):
            item = self._Stack__list[_(2)_]
            if i < len(self._Stack__list) _(3)_:
```

```
                output += _(4)_
            else:
                output += _(5)_
        output += _(6)_
        return _(7)_
```

1(e). Explain briefly whether the class **Stack** is an abstract class or not.

```
It is NOT an Abstract class. We are not able to instantiate an object from
the class.
```

1(f). A data structure such as a Stack is considered to be a *LIFO* data structure. Briefly explain what does *LIFO* stands for?

```
LIFO stands for 'Last in, First Out'. It means that the item that was
added last will be the first one to be removed.
```

1(g). Briefly explain what is a *Deque* data structure?

```
Deque is a double-ended queue. It is a Data Structure where you can add
and remove items from either the front or the back ends of the queue.
```

2. Converting a number from one number system to another can be achieved through either an iterative or a recursive approach. Consider the below iterative function toHexadecimal that converts a decimal number into hexadecimal. A helper function hexDigit, a main program, and output are also provided.

Main program

```
def hexDigit(index):

return['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
[index % 16]
def toHexadecimal(n):
    hexNum= ''
    while n > 0:
        digit = hexDigit( n % 16)
        hexNum =   digit + hexNum
        n= int(n /16)
    return hexNum

print( toHexadecimal(10),end=' ')
print( toHexadecimal(11),end=' ')
```

```
print( toHexadecimal(32),end=' ')
print( toHexadecimal(31),end=' ')
print( toHexadecimal(171),end=' ')
print( toHexadecimal(255),end=' '
```

Output of Main program

```
A  B  20  1F  AB  FF
```

2(a). Rewrite the iterative toHexadecimal function into a recursive function. You must make use of the helper function hexDigit Write your implementation in the below box.

```
# Write your code implementation here.
def toHexadecimal(n):
    if n < 16:
        return hexDigit(n)
    else:
        return toHexadecimal(n // 16) + hexDigit(n % 16)
```

2(b). Mention two key differences between an *iterative* and a *recursive* function.

```
    1. Recursive functions calls itself until the stopping condition is
met. However, an iterative function repeats until a condition fails.

    2. Recursive functions typically uses a branching structure(if-else).
Iterative function uses a looping structure (while-loops, for-loops)
```