CIND830 - Python Programming for Data Science

Assignment 2 (15% of the final grade)

Due on Dec 12, 2022 11:59 PM

This is a Jupyter Notebook document that extends a simple formatting syntax for authoring HTML and PDF. Review this website for more details on using Jupyter Notebooks.

Consider using a Jupyter Notebook platform to complete this assignment. Ensure using **Python 3.7 release or higher** then complete the assignment by inserting your Python code wherever seeing the string `#INSERT YOUR ANSWER HERE.`

You are expected to submit the notebook file (in IPYNB format) and the exported version (either in PDF or HTML) in the same Assignment link in D2L. Use these guidelines to submit **both** the IPYNB and the exported file (HTML or PDF). Failing to submit both files will be subject to mark deduction.

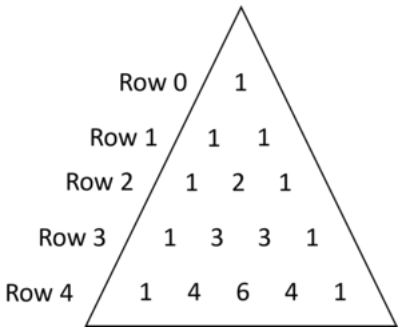
Please be advised that you cannot get more than 100% in this assignment, and the **BONUS** question (if there is any) will only be graded if all other questions have been submitted.

Coverage:

1. Lists, Tuples and Dictionaries
2. Functions and Classes
3. Searching and Sorting
4. Arrays and Grids
5. Stacks, Queues, and Lists

Question 1 **[30 pts]**:

Pascal's triangle is a triangular array that can be used to find combinations. Each number in the triangle is the sum of the two numbers above it. For example, the value 4 in row 4 is the sum of 1 and 3 in the row above. The first and last number in any row will always be 1.



The value of a row and column in Pascal's Triangle can be calculated using the following formula:

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

where n and r, are the row and column indices.

For example, $C(4,2) = 6$ corresponds to row index 4 and column index 2.

Question 1.a [10 pts]: Design a function that takes the row id and column id as arguments and returns the corresponding value from Pascal's triangle. Hint: You can use the math.factorial() function or design one to compute the factorials.

```
#Question 1(a)
# Function for "n" input
def get_input_n():
    """
  Function to ensuring that the user enters an positive integer.
```

```
    """
    #loops to ensures the user enters an input
    inputFetched = False
    while inputFetched == False:

        #this try-except statement validates the entry of a string
        try:
            n = int(input("Please enter the row id: n = "))

            #to validate the number inputted is positive
            if n <0:
                print("Please enter a positive number")
            else:
                inputFetched = True
                return n
        except:
            print("Please enter a number")

# Function for "k" input
def get_input_k():
    """
  Function to ensuring that the user enters an positive integer.
    """
    #loops to ensures the user enters an input
    inputFetched = False
    while inputFetched == False:

        #this try-except statement validates the entry of a string
        try:
            k = int(input("Please enter the column id: k = "))

            #to validate the number inputted is positive
            if k <0:
                print("Please enter a positive number")
            else:
                inputFetched = True
                return k
        except:
            print("Please enter a number")

# Function for calculating binomial cofficient
def binom(n, k):
    import math
    return (int((math.factorial(n))/(math.factorial(k)* math.factorial(n - k))))

# Function to return binomial cofficient C(n,k) : with given "n" and "k"
def binomial_coefficient(n,k):
 if  k == 1 or k == n:
    print("Binomial Cofficient C(n,k) of given 'n' and 'k' is :", 1)
 elif k > n:
  print("There is no Binomial Cofficient at the given 'n' and 'k', it is empty space")
 else:
   print("Binomial Cofficient C(n,k) of given 'n' and 'k' is :", binom(n, k))
def main():
 n = get_input_n()
 k = get_input_k()
 binomial_coefficient(n,k)
main()


     Please enter the row id: n = 4
     Please enter the column id: k = 2
     Binomial Cofficient C(n,k) of given 'n' and 'k' is : 6
```

Question 1.b [10 pts]: Design a function that takes an integer n as input and displays the first n rows of Pascal's triangle. For example, if the user calls the function with 6 as an argument, the following output should be displayed.

```
       1
      1 1
     1 2 1
    1 3 3 1
   1 4 6 4 1
  1 5 10 10 5 1
```

Hint: You can use the function you designed in part a to compute the triangle values.

```
#Question 1(b)
def get_input():
    """
  Function to ensuring that the user enters an positive integer.
    """
    #loops to ensures the user enters an input
    inputFetched = False
    while inputFetched == False:

        #this try-except statement validates the entry of a string
        try:
            numRows = int(input("How many rows of Pascal's Triangle do you need?----"))

            #to validate the number inputted is positive
            if numRows <0:
                print("Please enter a positive number")
            else:
                inputFetched = True
                return numRows
        except:
            print("Please enter a number")
def pascal_tri(numRows):
  from math import factorial
  '''Print Pascal's triangle with numRows.'''
  for i in range(numRows):
    # loop to get leading spaces
      for j in range(numRows-i+1):
          print(end=" ")
    # loop to get elements of row i
      for j in range(i+1):
          # nCr = n!/((n-r)!*r!)
          print(factorial(i)//(factorial(j)*factorial(i-j)), end=" ")
     # print each row in a new line
      print()
def main():
 numRows = get_input()
 pascal_tri(numRows)
main()
```

```
     How many rows of Pascal's Triangle do you need?----6
            1
           1 1
          1 2 1
         1 3 3 1
        1 4 6 4 1
       1 5 10 10 5 1
```

Question 1.c [10 pts]: Design a function that flips the Pascal triangle. For example, if the user calls the function with 6 as an argument, the output would be as follows.

```
 1 5 10 10 5 1
   1 4 6 4 1
    1 3 3 1
     1 2 1
      1 1
       1
```

Hint: You can use the function you designed in part a to compute the triangle values.

```
#Question 1(c)
def get_input():
    """
  Function to ensuring that the user enters an positive integer.
    """
    #loops to ensures the user enters an input
    inputFetched = False
    while inputFetched == False:

        #this try-except statement validates the entry of a string
        try:
            numRows = int(input("How many rows of Inverted Pascal's Triangle do you need?----"))
```

```
            #to validate the number inputted is positive
            if numRows <0:
                print("Please enter a positive number")
            else:
                inputFetched = True
                return numRows
        except:
            print("Please enter a number")
def inverted_pascal_tri(numRows):
  from math import factorial
  '''Print Inverted Pascal's triangle with numRows.'''
  k = 0
  for i in range(numRows, -1, -1):
    # loop to get leading spaces
      for j in range(k, 0, -1):
          print(end=" ")
      k+=1
    # loop to get elements of row i
      for j in range(0, i+1):
          # nCr = n!/((n-r)!*r!)
          print(factorial(i)//(factorial(j)*factorial(i-j)), end=" ")
     # print each row in a new line
      print("")
def main():
 numRows = get_input()
 inverted_pascal_tri(numRows)
main()
```

```
How many rows of Inverted Pascal's Triangle do you need?----6
1 6 15 20 15 6 1
 1 5 10 10 5 1
  1 4 6 4 1
   1 3 3 1
    1 2 1
     1 1
      1
```

Question 2 **[30 pts]**:

Define four classes, `Polygon`, `Rectangle`, `Square`, and `Triangle`. `Polygon` is the parent class of `Rectangle`, which is the parent class of `Square`. Polygon also is the parent class of the `Triangle` class.

1. The `Polygon` should have the following methods:
   - A method called `WhoAmI` that prints out `Triangle` if the current object is a triangle or `Square` if the current object is a square.
   - A method called `side_lengths` to print the side lengths of the polygon.
   - A method called `findArea()` that can be overridden to compute and return the `Polygon` area.
2. When creating `Rectangle` objects, the program should generate two random integers as the sides of the rectangle. Similarly, the program should randomly generate one random integer denoting one side for `Square` objects. The side lengths can be any value between 1 and 10, inclusively.
3. Each of the `Rectangle`, `Triangle` and `Square` classes should have a method called `findArea()` that computes and returns the respective area according to the following formulae:

$SquareArea = side^2$

$RectangleArea = side_1 \times side_2$

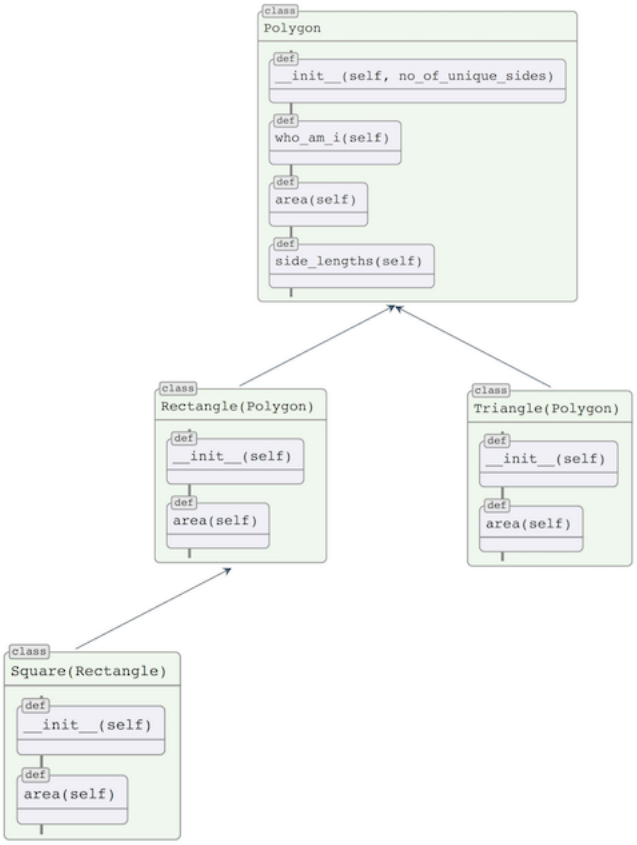
$TriangleArea = \sqrt{p(p-side_1)(p-side_2)(p-side_3)},\ \ where\ p = \frac{side_1+side_2+side_3}{2}$

4. To compute the triangle area, ensure that all side lengths are positive and the sum of any two side lengths is greater than the third side length.
5. The program should automatically create 3 Triangle and 3 Square objects and print out all their properties.

For example, the program would return the following output upon execution:

```
Square
Sides: [2]
Area: 4
---------------------------------------
Triangle
Sides: [7, 2, 9]
Area: 0.0
---------------------------------------
Square
Sides: [6]
Area: 36
---------------------------------------
Triangle
Sides: [9, 8, 6]
Area: 23.53
---------------------------------------
Square
Sides: [7]
Area: 49
---------------------------------------
Triangle
Sides: [9, 3, 9]
Area: 13.31
---------------------------------------
```

This might be helpful to visualize the hierarchy of these classes.




```
#Question 2
import random
import math

#Define parent class polygon
class polygon:

  def __init__(self, no_unique_sides):
    self.no_unique_sides = no_unique_sides
```

```
  def who_am_i(self):
    print("The name of the polygon is {}".format(self.shape))


  def side_lengths(self):
    print("The side(s) of {} are {}".format(self.shape,self.sides))

  def findArea1(self, area):
    self.area = area
    print("The area of the {} is {}".format(self.shape, self.area))


#define sub class triangle
class triangle(polygon):
  pass

#define sub parent class rectangle
class rectangle(polygon):

  def __init__(self):
    self.shape = 'Rectangle'
    self.sides = rec_lst

  def findArea(self):
    rec_area = rec_lst[0] * rec_lst[1]
    self.findArea1(rec_area)


#rec_lst = random.sample(range(1,12), 2)


#define sub class square
class square(rectangle):
  def __init__(self):
    self.shape = 'Square'
    self.sides = sqr_lst

  def findArea(self):
    sqr_area = sqr_lst ** 2
    self.findArea1(sqr_area)


#sqr_lst = random.randint(1,11)

#define class for triangle
class triangle(polygon):

  def __init__(self):
    self.shape = 'Triangle'
    self.sides = tri_lst

  def findArea(self):
    if tri_lst[0]<0 or tri_lst[1]<0 or tri_lst[2]<0 or tri_lst[0]+tri_lst[1]<=tri_lst[2] or tri_lst[0]+tri_lst[2]<=tri_lst[1] or tri_lst[1]+t
      print("Invalid Triangle")
    else:
      p = (tri_lst[0] + tri_lst[1] + tri_lst[2]) / 2
      tri_area = math.sqrt(p * (p - tri_lst[0]) * (p - tri_lst[1]) * (p - tri_lst[2]))
      self.findArea1(round(tri_area, 2))

#* (p - tri_lst[1]) * (p - tri_lst[2])) ** 1/2
#tri_lst = random.sample(range(1,12), 3)


#-------------------------------------

#def main():
for i in range (0,4):
  sqr_lst = random.randint(1,11) #random.randint(start, stop)
  sq_obj = square()
  sq_obj.who_am_i()
  sq_obj.side_lengths()
  sq_obj.findArea()
  print('-' * 50)
  tri_lst = random.sample(range(1,12), 3) #random.sample(sequence, k)#range(start, stop+1)
  tri_obj = triangle()
```

```
  tri_obj.who_am_i()
  tri_obj.side_lengths()
  tri_obj.findArea()
  print('-' * 50)
#main()
```

```
    The name of the polygon is Square
    The side(s) of Square are 8
    The area of the Square is 64
    --------------------------------------------------
    The name of the polygon is Triangle
    The side(s) of Triangle are [9, 4, 10]
    The area of the Triangle is 17.98
    --------------------------------------------------
    The name of the polygon is Square
    The side(s) of Square are 9
    The area of the Square is 81
    --------------------------------------------------
    The name of the polygon is Triangle
    The side(s) of Triangle are [11, 10, 6]
    The area of the Triangle is 29.76
    --------------------------------------------------
    The name of the polygon is Square
    The side(s) of Square are 3
    The area of the Square is 9
    --------------------------------------------------
    The name of the polygon is Triangle
    The side(s) of Triangle are [8, 6, 3]
    The area of the Triangle is 7.64
    --------------------------------------------------
    The name of the polygon is Square
    The side(s) of Square are 11
    The area of the Square is 121
    --------------------------------------------------
    The name of the polygon is Triangle
    The side(s) of Triangle are [5, 7, 8]
    The area of the Triangle is 17.32
    --------------------------------------------------
```

Question 3 [40 pts]:

Assume the following class implements the STACK abstract data type (ADT) using the array ADT.

```
class aStack(iArray):
    def __init__(self, capacity = 5):
        self._items = iArray(capacity)
        self._top = -1
        self._size = 0
    def push(self, newItem):
        self._top += 1
        self._size += 1
        self._items[self._top] = newItem
    def pop(self):
        oldItem = self._items[self._top]
        self._items[self._top] = None
        self._top -= 1
        self._size -= 1
        return oldItem
    def peek(self):
        return self._items[self._top]
    def __len__(self):
        return self._size
    def __str__(self):
        result = ' '
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result
```

Question 3.a [10 pts]: Emulate the stack behaviour using the Python list data structure rather than the Array ADT, then list the pros and cons of this approach.

```
class aStack(list):
    def __init__(self, capacity = 5):
        self._items = []
        self._top = -1
        self._size = 0
    def push(self, newItem):
        self._top += 1
        self._size += 1
        self._items.append(newItem)#self._items[self._top] = newItem
    def pop(self):
        oldItem = self._items[self._top]
        self._items[self._top] = None
        self._top -= 1
        self._size -= 1
        return oldItem
    def peek(self):
        return self._items[self._top]
    def __len__(self):
        return self._size
    def __str__(self):
        result = ' '
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result
s = aStack(10)
s.push(10)
s.push(11)
s.push(50)
s.push(13)
s.push(13)
print(s)
s.pop()
print(s)
```

```
 10 11 50 13 13
 10 11 50 13
```

PROS AND CONS OF USING LIST AND ARRAY: Python's builtin data structure list can be used as a stack. Instead of push(), append() is used to add elements to the top of the stack while pop() removes the element in LIFO order.

- The list is the part of python's syntax so it doesn't need to be declared whereas you have to declare the array before using it.
- You can store values of different data-types in a list (heterogeneous), whereas in Array you can only store values of only the same data-type (homogeneous).
- Unfortunately, the list has a few shortcomings. The biggest issue is that it can run into speed issues as it grows. The items in the list are stored next to each other in memory, if the stack grows bigger than the block of memory that currently holds it, then Python needs to do some memory allocations. This can lead to some append() calls taking much longer than other ones.
- Arrays are stored more efficiently i.e. as contiguous blocks of memory vs. pointers to Python objects. Arrays take less memory compared to lists.
- The main difference between these two data types is the operations you can perform on them. For example, you can divide an array by 3 and it will divide each element of array by 3. Same can not be done with the list.

Question 3.b [10 pts]: Redefine the Stack class methods to push and pop two items rather than one item at a time.

For example, if the stack includes numbers from one to ten: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, then invoking the `pop()` method twice will remove the last four elements and modify the stack elements to be: `[1, 2, 3, 4, 5, 6]`

```
# For List ---Stack class methods to push and pop two items
class aStack_2(list):
    def __init__(self):
        self._items = []
        # self._top = 0
        self._size = 0
    def push(self, newItem, newItem2):
        # self._top += 1
        self._size += 2
```

```
        # self._items[self._top] = newItem
        self._items.append(newItem)
        self._items.append(newItem2)
    def pop(self):
        # oldItem = self._items[self._size]
        # self._items[self._size] = None
        # self._top -= 1
        oldItem = self._items.pop()
        oldItem2 = self._items.pop()
        self._size -= 2
        return (oldItem, oldItem2)
    def peek(self):
        return self._items[self._size-1]
    def __len__(self):
        return self._size
    def __str__(self):
        result = ' '
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result

s = aStack_2()
s.push(1, 2)
s.push(3,4)
s.push(5,6)
s.push(7,8)
s.push(9,10)
print(s)
s.pop()
print(s)
```

```
      1 2 3 4 5 6 7 8 9 10
      1 2 3 4 5 6 7 8
```

```
# For Array------Stack class methods to push and pop two items
#Assume the following class implements the STACK abstract data type (ADT) using the array ADT.
class iArray():
 def __init__(self, capacity, fillValue = None):
  self._items = list()
  for count in range(capacity):
   self._items.append(fillValue)
 def __len__(self):
#Obtaining the size of the array
    return len(self._items)

 def __str__(self):
#Representing the array as String
  return str(self._items)
 def __setitem__(self, index, newItem):
#Replacing an element in the array
#Assigning a value to an element
  self._items[index] = newItem
 def __setitem__(self, index, newItem):
#Replacing an element in the array
#Assigning a value to an element
  self._items[index] = newItem
 def __getitem__(self, index):
#Obtaining an element in the array
  return self._items[index]
 def __iter__(self):
#Traversing elements with loops
  return iter(self._items)
class aStack(iArray):
    def __init__(self, capacity = 5):
        self._items = iArray(capacity)
        self._top = -1
        self._size = 0

    def push(self, newItem1, newItem2):
        self._top += 1
        self._size += 1
        self._items[self._top] = newItem1
        self._top += 1
        self._size += 1
        self._items[self._top] = newItem2
```

```
    def pop(self):
        oldItem1 = self._items[self._top]
        self._items[self._top] = None
        self._top -= 1
        self._size -= 1
        oldItem2 = self._items[self._top]
        self._items[self._top] = None
        self._top -= 1
        self._size -= 1
        return [oldItem1, oldItem2]

    def peek(self):
        return self._items[self._top]

    def _len_(self):
        return self._size

    def _str_(self):
        result = ' '
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result
s = aStack(10)
s.push(1, 2)
s.push(3,4)
s.push(5,6)
s.push(7,8)
s.push(9,10)
print(s)
s.pop()
print(s)
```

```
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
 [1, 2, 3, 4, 5, 6, 7, 8, None, None]
```

Question 3.c [10 pts]: Add a new method called `getMax` that returns the greatest stack data element.

```
class aStack(list):
    def __init__(self, capacity = 5):
        self._items = []
        self._top = -1
        self._size = 0
        self.trackStack = []
    def push(self, newItem):
        self._top += 1
        self._size += 1
        self._items.append(newItem)#self._items[self._top] = newItem
        if (len(self._items) == 1):
            self.trackStack.append(newItem)
            return

        # If current element is greater than
        # the top element of track stack,
        # append the current element to track
        # stack otherwise append the element
        # at top of track stack again into it.
        if (newItem > self.trackStack[-1]):
            self.trackStack.append(newItem)
        else:
            self.trackStack.append(self.trackStack[-1])
    def getMax(self):
        return self.trackStack[-1]

    def pop(self):
        oldItem = self._items[self._top]
        self._items[self._top] = None
        self._top -= 1
        self._size -= 1
        self.trackStack.pop()
        return oldItem
    def peek(self):
        return self._items[self._top]
    def __len__(self):
        return self._size
    def __str__(self):
        result = ' '
```

```
        for i in range(len(self)):
            result += str(self._items[i]) + ' '
        return result

s = aStack()
s.push(8)
print(s.getMax())
s.push(10)
print(s.getMax())
s.push(50)
print(s.getMax())
s.pop()
print(s.getMax())
```

```
8
10
50
10
```

Question 3.d [10 pts]: Design a function that returns whether an expression has an **unnecessary** parenthesis or not. A set of parenthesis is unnecessary if multiple sets of parenthesis surround the whole expression or one of its subexpressions.

For example, These expressions have unnecessary parenthesis:

1. `(a+b)+((c+d))` The subexpression `c+d` is surrounded by two pairs of brackets.
2. `((a+(b)))+(c+d)` The subexpression `a+(b)` is surrounded by two pairs of brackets.
3. `(((a+b)+(c+d)))` Two pairs of brackets surround the whole expression.
4. `((a+(((b)))))+(c+d)` Each of the two subexpressions `(b)` and `a+(b)` is surrounded by two pairs of brackets.

However, these expressions do not have any unnecessary parenthesis.

1. (a+b)+(c+d)
2. a+(b+c+d)

Note: Please do not use a third-party library for any part of the code, as only Python built-in functions are permitted in this question.

```
# Function to find duplicate parenthesis
# in a balanced expression
def findDuplicateparenthesis(string):

    # create a stack of characters
    Stack = []

    # Iterate through the given expression
    for ch in string:

        # if current character is
        # close parenthesis ')'
        if ch == ')':

            # pop character from the stack
            top = Stack.pop()

            # stores the number of characters between
            # a closing and opening parenthesis
            # if this count is less than or equal to 1
            # then the brackets are redundant else not
            elementsInside = 0
            while top != '(':

                elementsInside += 1
                top = Stack.pop()

            if elementsInside < 1:
                return True

        # push open parenthesis '(', operators
        # and operands to stack
        else:
            Stack.append(ch)

    # No duplicates found
    return False
```

```
# Driver Code
if __name__ == "__main__":

    # input balanced expression
    string = "(a+b)+(c+d)"

    if findDuplicateparenthesis(string) == True:
        print("Unnecessary parenthesis Found")
    else:
        print("No Unnecessary parenthesis Found")


 No Unnecessary parenthesis Found
```

This is the end of assignment 2