CIND830 - Python Programming for Data Science

Assignment 3 (10% of the final grade)

Due as per D2L

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 Notebook.

Use the JupyterHub server on the Google Cloud Platform, provided by your designated instructor, for this assignment. Ensure using **Python 3.7.6** release then complete the assignment by inserting your Python code wherever seeing the string "#INSERT YOUR ANSWER HERE."

When you click the `File` button, from the top navigation bar, then select `Export Notebook As ...`, a document (PDF or HTML format) will be generated that includes both the assignment content and the output of any embedded Python code chunks.

Using these guidelines, submit **both** the IPYNB and the exported file (PDF or HTML). Failing to submit both files will be subject to mark deduction.

Question 1 **[25 pts]**:

Write a class **SpecialList** that can hold only strings.

Note. Only Python built-in functions are allowed to be used in this question. Do not use a third-party library for any part of the code. Importing any third-party library will result in marks deduction

a. Users of the class should be able to instantiate an object of the **SpecialList** class with a list of strings. Name the list as *items*. Ensure you check that the input values are strings and if not throw an error. **(5 Points)**

b. Implement a *sort* method as a member of the **SpecialList** class. The sort method should sort the *items* in ascending order. Use bubble sort or insertion sort algorithm for this task. **(10 Points)**

For example, if objSpcList = SpecialList([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10]) then after invoking the `sort` method, the `items` attribute will be [-10, 2, 3, 3, 3, 4, 4, 5, 5, 6, 7, 8, 9, 20]

c. Implement a *contains* method as a member of the **SpecialList** class. It should take a parameter x. If x is in the *items*, the *contains* method should return True otherwise False. Use binary search algorithm to implement this method. Remember that binary search works on sorted lists **(10 Points)**

For example, if `objSpcList = SpecialList([5, 3, 7, 3, 6, 4, 5, 8, 2, 9, 3, 4, 20, -10])` and the user invoked the the `contains()` method with `8`, then the method will return `True

```
#Part A
class SpecialList:
    def __init__(self, items):
        for element in items: #name the list as items and go through each
            if type(element) is not str: #checking for lack of strings
                print("Please change this to a string. We can only take strings.")
                quit()
            else:
                self.items = items

#Part B
#used insertion sort algo
    def sort(self):
        itemlist = self.items
        for x in range(1, len(itemlist)): #go through each item in list
          y = x-1
          currentitem = itemlist[x]
          if currentitem < itemlist[y]:
            if not y < 0:
              if currentitem < itemlist[y] and y >= 0:
                while currentitem < itemlist[y] and y >= 0: #use a while loop to keep moving it backward if necessary and ch
                    itemlist[y+1] = itemlist[y] #if item in list is greater than currentitem, move them one position forward
                    y = y - 1
                itemlist[y+1] = currentitem

#Part C
#Binary Search Algo - sort, then search for the middle of each new batch and then go left/right depending on if larger or sma

    def contains(self, x):
        itemlist = self.items
        self.sort() #sort the items using Part B algo
        #Initialize all variables first last and middle
        first = 0
        middle = 0
        last = len(itemlist) - 1
        for i in itemlist:
            middle = int((first + last)/2) #divide as an integer
            if itemlist[middle] > x: #if element in middle > x then search item must be in 2nd half so narrow search using th
                last = middle - 1 #shrink
            elif itemlist[middle] < x: #if element in middle < x then search item must be in 1st half so narrow search using
                first = middle + 1 #shrink
            elif itemlist[middle] == x: #element in list is in the middle itself so return True
                return True

        if first > last:
          return False #if no match then return False

#Testing
objSpecialList = SpecialList(
    ["5", "3", "7", "4", "6", "4", "5", "8", "2", "9", "3", "4", "20", "-10"])
print(objSpecialList.contains("8"))
```

```
True
```

Question 2 **[25 pts]**:

Infix expression example: 3+4*5/6 (operators are between numbers)

Infix expression evaluation: 6.3333

Postfix version of the infix expression: 345*6/+

Question:

Define a class **InfixExpression**

- Assume for this question the infix expression is a list that can only contain numbers (integers or floats), "+", "*", "-", and "/"
- Remember the precendence rules of operations when writing the functions.

Note.You can use any existing library for stack or queue implementation

a. Users of the class should be able to initialize an object of the *InfixExpression* class with a list containing a *infixexpression*. Name the list as *infix*. **(5 Points)**

b. Implement *_check_infix* method as a member of *InfixExpression* class. The method should return True if the given expression is a correctly formatted infix expression else return False. Update the **init** function to use this function to check if the expression initialized is valid. If not throw an error from the **init** function **(5 Points)**

c. Implement a *infixtopostfix* method as a member of the *InfixExpression* class. The method should return the *postfix* form of the *infix* expression.**(7.5 Points)**

- **Algorithm to implement**
    - Initalize an empty stack for the operators
    - Read infix expression list from left to right
        - If element is operand (numbers) then print it out
        - If element is math operator (call it thisOps)
            - While stack is not empty and the top element on the stack has the same or greater precedence as thisOp
                - Pop (and print) from the stack
            - Push thisOps onto the stack
    - If the entire expression has been consumed then pop (and print) out remaining operators from stack

d. Implement *evaluate* method as a member of the *InfixExpression* class. The method should return the evaluated value of the expression. **(7.5 Points)**

- **Algorithm to implement**
    - Initialize two empty stacks: a value stack to hold operands (numbers) and operator stack to hold operators
    - While there are still elements in infix expression to be read in,
        - Get the next element.
        - If the element is:
            - A number: push it onto the value stack.
            - An operator (call it thisOp):
                - While the operator stack is not empty, and the top element on the operator stack has the same or greater precedence as thisOp,
                    - Pop the operator from the operator stack.
                    - Pop the value stack twice, getting two operands.
                    - Apply the operator to the operands, in the correct order. (Remember stack is last in first out)
                    - Push the result onto the value stack.
                - Push thisOp onto the operator stack.
    - While the operator stack is not empty,
        - Pop the operator from the operator stack.
        - Pop the value stack twice, getting two operands.
        - Apply the operator to the operands, in the correct order.
        - Push the result onto the value stack.
    - At this point the operator stack should be empty, and the value stack should have only one value in it, which is the final result.

Helper Libraries

- https://docs.python.org/3/library/operator.html
- https://docs.python.org/3/library/collections.html#collections.deque

```
class InfixExpression:

    operators = ["+", "-", "*", "/"]

    # Part A

    def __init__(self, infix):
        if self._check_infix(infix) == True: #check if expression initialized is valid
            self.infix = infix
        else: #else throw an error
            raise ValueError(
                "Error: The items in the list do not make an infix expression")

    # Part B
    def _check_infix(self, infix):
        isInfix = True
        for i in range(len(infix)):
            if i % 2 == 0 and type(infix[i]) != int:
                isInfix = False
            elif (i % 2 != 0) and infix[i] not in self.operators:
                isInfix = False
        if isInfix == True and len(infix) % 2 == 0:
            isInfix = False
        return isInfix

    # Part C - return postfix form of infix expression
    def infixtopostfix(self):
        postfix = []
        operator_stack = []

        for item in self.infix:
            if type(item) == int:
                postfix.append(item)
            elif item in self.operators:
                thisOp = item
                if thisOp == "*" or thisOp == "/":
                    while len(operator_stack) > 0 and operator_stack[-1] == "*" or len(operator_stack) > 0 and operator_stack
                    while len(operator_stack) > 0 and operator_stack[-1]        or len(operator_stack) > 0 and operator_stac
                        postfix.append(operator_stack.pop())
                    operator_stack.append(thisOp)
                else:
                    while len(operator_stack) > 0:
                        postfix.append(operator_stack.pop())
                    operator_stack.append(thisOp)
        for operator in operator_stack[::-1]:
            postfix.append(operator)
        return postfix

    # Part D
    def evaluate(self):
        value_stack = []
        operator_stack = []

        for item in self.infix:
            if type(item) == int:
                value_stack.append(item)
            elif item in self.operators:
                thisOp = item
                if thisOp == "*" or thisOp == "/":
                    while len(operator_stack) > 0 and operator_stack[-1] == "*" or len(operator_stack) > 0 and operator_stac
                        operator = operator_stack.pop()
                        second_value = value_stack.pop()
                        first_value = value_stack.pop()

                        if operator == "*":
                            value_stack.append(first_value * second_value)
                        else:
                            value_stack.append(first_value / second_value)

                    operator_stack.append(thisOp)
                else:
                    while len(operator_stack) > 0:
                        operator = operator_stack.pop()
                        second_value = value_stack.pop()
                        first_value = value_stack.pop()

                        if (operator == '*'):
```

```
                        if (operator == '*'):
                            value_stack.append(first_value * second_value)
                        elif (operator == '/'):
                            value_stack.append(first_value / second_value)
                        elif (operator == '+'):
                            value_stack.append(first_value + second_value)
                        else:
                            value_stack.append(first_value - second_value)

                    operator_stack.append(thisOp)

        while len(operator_stack) > 0:
            operator = operator_stack.pop()
            second_value = value_stack.pop()
            first_value = value_stack.pop()

            if (operator == '*'):
                value_stack.append(first_value * second_value)
            elif (operator == '/'):
                value_stack.append(first_value / second_value)
            elif (operator == '+'):
                value_stack.append(first_value + second_value)
            else:
                value_stack.append(first_value - second_value)

        return value_stack[0]


test = InfixExpression([3, '+', 4, '*', 5, "/", 6])
print(test.infixtopostfix())
print(test.evaluate())
```

```
[3, 4, 5, '*', 6, '/', '+']
6.333333333333334
```

Question 3 **[25 pts]**:

NOTE: You can use the `numpy` library to solve this question.

- https://pypi.org/project/numpy/

a) (7.5 Points) Create a function that creates and prints a two dimensional 2x2 array (i.e. grid). The elements of the array should all be zeros.

```
#pip install numpy
import numpy

def gridprinter():
  a = numpy.zeros((2,2))
  print(a)

gridprinter()
```

```
[[0. 0.]
 [0. 0.]]
```

b) (7.5 Points) Replace the elements of the grid with randomly selected numbers from the inclusive interval $[-5, +5]$

```
a = numpy.zeros((2,2))
print(a)
a = numpy.random.randint(low = -5, high = 5, size = (2,2))
print(a)

#this is another way to do it
#print (numpy.where(a==0, numpy.random.randint(low = -5, high = 5), a))
```

```
[[0. 0.]
 [0. 0.]]
[[-5 -5]
 [-2 -4]]
```

c) (10 Points) Compute the determinant of the grid. For example, if the grid is equal to:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\det A = ad - bc$$

```
print(numpy.linalg.det(a))
```

```
-8.000000000000002
```

Question 4 **[25 pts]**:

a) (10 Points) Complete the methods of the following `Stack` class according to their description

```
class Stack:
  def __init__(self):
    """ Initialize a new stack """
    self.elements = []
  def push(self, new_item):
    """ Append the new item to the stack """
    ## CODE HERE ###
  def pop(self):
    """ Remove and return the last item from the stack """
    ## CODE HERE ###
  def size(self):
    """ Return the total number of elements in the stack """
    ## CODE HERE ###
  def is_empty(self):
    """ Return True if the stack is empty and False if it is not empty """
    ## CODE HERE ###
  def peek(self):
    """ Return the element at the top of the stack or return None if the stack is empty """
    ## CODE HERE ###
```

```
class Stack:
  def __init__(self):
    """ Initialize a new stack """
    self.elements = []
  def push(self, new_item):
    """ Append the new item to the stack """
    self.elements.append(new_item)
  def pop(self):
    """ Remove and return the last item from the stack """
    return self.elements.pop()
  def size(self):
    """ Return the total number of elements in the stack """
    return len(self.elements)
  def is_empty(self):
    """ Return True if the stack is empty and False if it is not empty """
    if self.elements == []:
      return True
    else:
      return False
  def peek(self):
    """ Return the element at the top of the stack or return None if the stack is empty """
    if self.is_empty() == True:
      return None
    else:
      return self.elements[len(self.elements)-1]
```

Use the Stack class you defined in `Q1a` to solve the following problem.

b) (7.5 Points) Write a function to detect whether the order of brackets is correct using stacks. Some examples are given below:

```
exp1 = "(2+3)+(1-5)" # True
```

```
exp2 = "((3*2))*(7/3))" # False
exp3 = "(3*5))" # False


def is_valid(exp):
  """ Check the orders of the brackets
      Returns True or False
  """
  balanced = "Y"
  for char in exp:
    if char == "(" : #if opening bracket
      s.push(char)
    elif char == ")": #if closing bracket
      if s.is_empty() == True:
        balanced = "N"
      if s.is_empty() == False: #if something there
        s.pop()

  if balanced == "Y": #if there is an empty list aka it is balanced then
    return True
  elif balanced == "N":
    return False

is_valid(exp3)
```

```
     False
```

c) (7.5 Points) Create a MinStack class inherited from the Stack class defined in `Q1a` . The MinStack class has an additional function called get_min() that returns the minimum element in the stack

```
class MinStack(Stack):
  def get_min(self):
    min = s.peek()
    for i in range(len(s.elements)):
      if min > s.elements[len(s.elements)-2 - i]:
        min = s.elements[len(s.elements)-2 - i]
    return min
```

```
#test
s = MinStack()
s.elements = []
s.push(5)
s.push(3)
s.push(6)
s.push(0)

s.get_min()
```

```
     0
```

This is the end of assignment 3

✓ 0s completed at 11:34 PM ● ×