Programming and Data Structures in Python, 2023

Graded Assignment 1, 20 Sep 2023, due 27 Sep 2023

Write four Python functions as specified below. Combine the text for all four functions together into a single file. Your function will be called automatically with various inputs and should return values as specified. Do not write commands to read any input or print any output.

- You may define additional auxiliary functions as needed.
- In all cases you may assume that the value passed to the function is of the expected type, so your function does not have to check for malformed inputs.

Note

- Test on Swayam portal
- Official submissions on Moodle

---

1. Write a function `delchar(s,c)` that takes as input strings `s` and `c`, where `c` has length 1 (i.e., a single character), and returns the string obtained by deleting all occurrences of `c` in `s`. If `c` has length other than 1, the function should return `s`

   Here are some examples to show how your function should work.

   ```
   >>> delchar("banana","b")
   'anana'

   >>> delchar("banana","a")
   'bnn'

   >>> delchar("banana","n")
   'baaa'

   >>> delchar("banana","an")
   'banana'
   ```

2. Write a function `nestingdepth(s)` that takes as input a string `s` and computes the maximum nesting depth of brackets if `s` has properly nested brackets. If the string is not properly matched, your function should return `-1`.

   Hint: Use the function `matched()` from the practice assignment.

   Here are some examples to show how your function should work.

   ```
   >>> nestingdepth("zb%78")
   0

   >>> nestingdepth("(7)(a")
   -1

   >>> nestingdepth("a)*(?")
   -1

   >>> nestingdepth("((jkl)78(A)&l(8(dd(FJI:),):)?)")
   4
   ```

3. Write a function `accordian(l)` that takes as input a list of integer `l` and returns `True` if the absolute difference between each adjacent pair of elements alternates between increasing strictly and decreasing strictly.

   Here are some examples of how your function should work.

   ```
   >>> accordian([1,5,1])
   False
   ```

   Explanation: Differences between adjacent elements are `5-1 = 4`, `5-1 = 4`, which are equal.

   ```
   >>> accordian([1,5,2,8,3])
   True
   ```

   Explanation: Differences between adjacent elements are `5-1 = 4`, `5-2 = 3`, `8-2 = 6`, `8-3 = 5`, so the differences decrease, increase and then decrease.

```
>>> accordian([-2,1,5,2,8,3])
True
```

Explanation: Differences between adjacent elements are `1-(-2) = 3`, `5-1 = 4`, `5-2 = 3`, `8-2 = 6`, `8-3 = 5`, so the differences increase, decrease, increase and then decrease.

```
>>> accordian([1,5,2,8,1])
False
```

Explanation: Differences between adjacent elements are `1-(-2) = 3`, `5-1 = 4`, `5-2 = 3`, `8-2 = 6`, `8-1 = 7`, so the differences increase, decrease, increase and then increase again.

4. A square n×n matrix of integers can be written in Python as a list with n elements, where each element is in turn a list of n integers, representing a row of the matrix. For instance, the matrix

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

would be represented as `[[1,2,3], [4,5,6], [7,8,9]]`.

Write a function `rotate(m)` that takes a list representation m of a square matrix as input, and returns the matrix obtained by rotating the original matrix clockwise by 90 degrees. For instance, if we rotate the matrix above, we get

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

Your function should *not* modify the argument m provided to the function `rotate()`.

Here are some examples of how your function should work.

```
>>> rotate([[1,2],[3,4]])
[[3, 1], [4, 2]]
```

Explanation:

```
   1  2    becomes     3  1
   3  4                4  2
```

```
>>> rotate([[1,2,3],[4,5,6],[7,8,9]])
[[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

Explanation:

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

```
>>> rotate([[1,1,1],[2,2,2],[3,3,3]])
[[3, 2, 1], [3, 2, 1], [3, 2, 1]]
```

Explanation:

```
   1  1  1    becomes   3  2  1
   2  2  2              3  2  1
   3  3  3              3  2  1
```