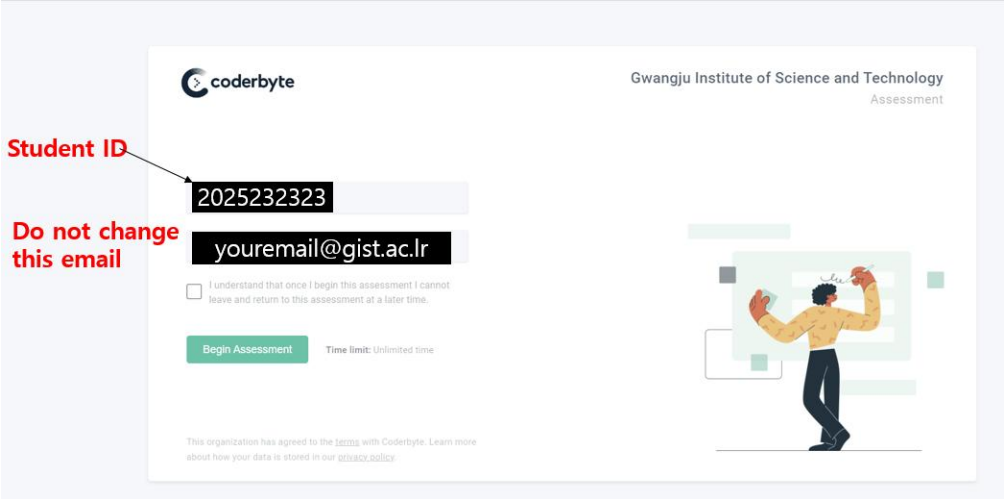
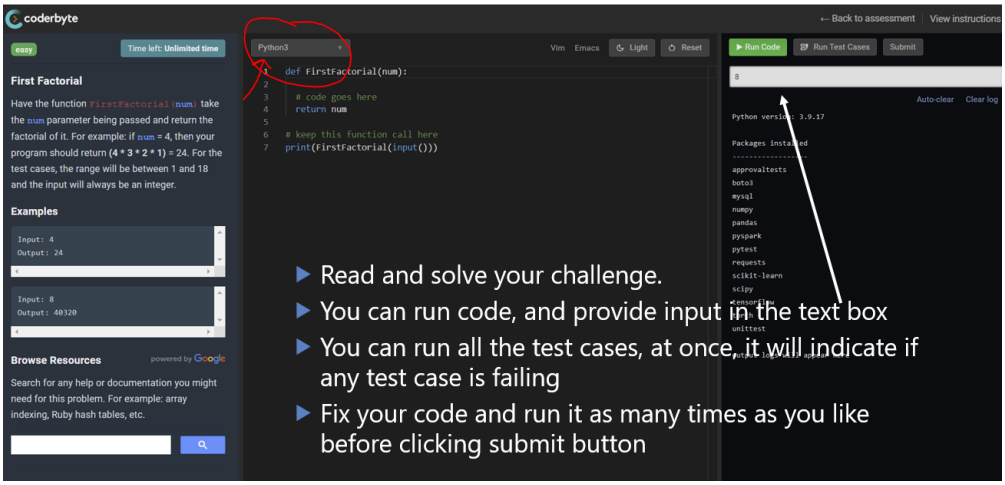
Algo Assignment 5: DP2

General Instruction to submit the assignment

1. Open the link sent in email and enter your **Student ID** and **ZEUS email address**



2. You must change the programming language to Python3 before starting to code. There is a drop down button above the coding window.



3. If you submit and try to resubmit etc., the system will not let you do that. (**once submitted you can not update your code**)
4. Hence **solve this question before hand and then enter your code in Coderbyte system**. This way you will save a lot of hassle. The questions will also be released in the pdf format. Make sure your code is giving right results for all the given test cases before you submit your code.
5. You may ignore warning messages about cheating when you paste your code to Coderbyte.
6. Remember total points from the assignments are only **10%** of your grades. Hence these are mainly for practicing for mid-term and final-term coding test.
7. The time complexity of your submissions will be tested only after 4th Assignment.

BowlingPin

You will get score[*i*–1] * score[*i*] * score[*i*+1]. If *i*–1, or *i*+1 goes out of bounds of the array, then treat it as if there is a pin with a score 1 painted on it.

Return the maximum score you can collect by hitting the Pins correctly.

For the above example, you will get the array [3, 1, 5, 8]. According to the above order, if you hit the pin which has score 1, you get 3*1*5=15 points and the remaining pins are [3, 5, 8]. Next, you hit the pin with score 5, get 3*5*8=120 points, and get [3, 8] remaining. If you hit the pin with score 3, you get 1*3*8=24 points and the remaining pin is [8]. You hit the pin, and get 1*8*1 points. In this case, the total score is 15+120+24+8=167 points.

The constraints are given like this:

n = arr.length

1 <= *n* <= 500

0 <= arr[*i*] <= 100

Test cases:

1. For input `[3,1,5,8]`, the correct output is `167`

2. For input `[1,5]`, the correct output is `10`

3. For input `[5,0,8]`, the correct output is `48`

4. For input `[13,7,8,2,14,1,72,31,5]`, the correct output is `50989`

5. For input `[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]`, the correct output is `13680`

6. For input `[15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]`, the correct output is `13680`

7. For input `[18, 92, 40, 53, 26, 16, 64, 70, 68, 53, 32, 49, 73, 82, 98, 67, 98, 77, 91, 27, 10, 30, 8, 7, 100, 6, 45, 77, 63, 37, 95, 10, 84, 66, 3, 87, 76, 35, 13, 22, 54, 9, 33, 91, 60, 51, 21, 51, 61, 51]`, the correct output is `16836552`

8. For input `[12,11,10,11,12]`, the correct output is `4402`

9. For input `[3,2,13,55,55,41,99,5,6]`, the correct output is `604533`

10. For input `[7,0]`, the correct output is `7`

PlateBreaking

Have the function PlateBreaking(arr) with arr = [k, *n*].

You are given *k* identical plates and you have access to a building with *n* floors labeled from 1 to *n*.

You know that there exists a floor *f* where 0 <= *f* <= *n* such that any plate dropped at a floor higher than *f* will break, and any plate dropped at or below floor *f* will not break.

Each move, you may take an unbroken plate and drop it from any floor *x* (where 1 <= *x* <= *n*). If the plate breaks, you can no longer use it. However, if the plate does not break, you may reuse it in future moves.

Return the minimum number of moves that you need to determine with certainty what the value of *f* is.

For example, suppose you have *k*=1 plates and *n*=2 floors. Drop the plate from floor 1. If it breaks, we know that *f*=0. Otherwise, drop the plate from floor 2. If it breaks, we can know that *f*=1, and if it doesn't break, *f*=2. Hence, we need at minimum 2 moves to determine with certainty what the value of f is.

The constraints of k and n will be like this:

1 <= k <= 100

1 <= n <= 1000

Test cases:

1. For input `[2,6]`, the correct output is `3`

2. For input `[3,14]`, the correct output is `4`

3. For input `[4,900]`, the correct output is `13`

4. For input `[5,900]`, the correct output is `11`

5. For input `[2,80]`, the correct output is `13`

6. For input `[3,80]`, the correct output is `8`

7. For input `[4,80]`, the correct output is `7`

8. For input `[5,80]`, the correct output is `7`

9. For input `[1,20]`, the correct output is `20`

10. For input `[10,1000]`, the correct output is `10`

LCS

Have the function LCS(**strArr**) take the **strArr** parameter being passed which will be an array of two strings containing only the characters {a,b,c} and have your program return the length of the longest common subsequence common to both strings. A common subsequence for two strings does not require each character to occupy consecutive positions within the original strings. For example: if **strArr** is ["abcabb","bacb"] then your program should return **3** because one longest common subsequence for these two strings is "bab" and there are also other 3-length subsequences such as "acb" and "bcb" but 3 is the longest common subsequence for these two strings.

Test cases:

1. For input ["abc","cb"], the correct output is 1
2. For input ["abcabb","bacb"], the correct output is 3
3. For input ["bcacb","aacabb"], the correct output is 3
4. For input ["aaac","caaa"], the correct output is 3
5. For input ["cbac","cacbbc"], the correct output is 3
6. For input ["caabbcab","bbabcc"], the correct output is 4
7. For input ["b","a"], the correct output is 0
8. For input ["a","aa"], the correct output is 1
9. For input ["abcabcabc","bbcbbcaa"], the correct output is 5
10. For input ["abccc","caabac"], the correct output is 3