Week 1 Live Coding Solution

Live Coding Problem 1

A positive integer `m` is a prime product if it can be written as `p × q`, where `p` and `q` are both primes.

.

Write a Python function **prime_product(m)** that takes an integer `m` as input and returns `True` if `m` is a prime product and `False` otherwise. (If `m` is not positive, function should return `False`.)

Sample Input

```
6
```

Output

```
True
```

Solution

```
# Solution
def factors(n):
    factorlist = []
    for i in range(1,n+1):
        if n%i == 0:
            factorlist.append(i)
    return(factorlist)
def isprime(n):
    return(factors(n) == [1,n])

def prime_product(n):
    for i in range(1,n+1):
        if n%i == 0:
            if isprime(i) and isprime(n//i):
                return(True)
    return(False)



# suffix (Visible)
n = int(input())
print(prime_product(n))
```

Public Test case

Input 1

```
6
```

Output

```
True
```

Input 2

```
12
```

Output

```
False
```

Input 3

```
58
```

Output

```
True
```

Private Test case

Input 1

```
35
```

Output

```
True
```

Input 2

```
100
```

Output

```
False
```

Input 3

```
-12
```

Output

```
False
```

Input 4

```
77
```

Output

```
True
```

Live Coding Problem 2

Write a function **del_char(s,c)** that takes strings `s` and `c` as input, 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`.

Sample input-1

```
banana
b
```

Output

```
anana
```

Sample input-2

```
banana
an
```

Output

```
banana
```

Solution

```python
# Solution
def del_char(s,c):
    if len(c) != 1:
        return(s)
    snew = ""
    for char in s:
        if char != c:
            snew = snew + char
    return(snew)


# Suffix (Visible)
s = input()
c = input()
print(del_char(s,c))
```

Public Test case

Input 1

```
banana
b
```

Output

```
anana
```

Input 2

```
banana
an
```

Output

```
banana
```

Input 3

```
data structure
u
```

Output

```
data strctre
```

Private Test case

Input 1

```
this is pdsa course
s
```

Output

```
thi i pda coure
```

Input 2

```
this is pdsa course
is
```

Output

```
this is pdsa course
```

Input 3

```
data structure
a
```

Output

```
dt structure
```

Input 4

```
apple
p
```

Output

```
ale
```

Live Coding Problem 3

Write a function **shuffle(l1,l2)** that takes two lists, `l1` and `l2` as input, and returns a list consisting of the first element in `l1`, then the first element in `l2`, then the second element in `l1`, then the second element in `l2`, and so on. If the two lists are not of equal length, the remaining elements of the longer list are appended at the end of the shuffled output.

Sample Input

```
[0,2,4]
[1,3,5]
```

Output

```
[0, 1, 2, 3, 4, 5]
```

Sample Input

```
[0,2,4]
[1]
```

Output

```
[0, 1, 2, 4]
```

Solution

```
# Solution
def shuffle(l1,l2):
    if len(l1) < len(l2):
        minlength = len(l1)
    else:
        minlength = len(l2)
    shuffled = []
    for i in range(minlength):
        shuffled.append(l1[i])
        shuffled.append(l2[i])
    shuffled = shuffled + l1[minlength:] + l2[minlength:]
    return(shuffled)


# Suffix code (visible)
L1 = eval(input())
L2 = eval(input())
```

```
print(shuffle(L1,L2))
```

Public Test case

Input 1

```
[0,2,4]
[1,3,5]
```

Output

```
[0, 1, 2, 3, 4, 5]
```

Input 2

```
[0,2,4]
[1]
```

Output

```
[0, 1, 2, 4]
```

Input 3

```
[0]
[1,3,5]
```

Output

```
[0, 1, 3, 5]
```

Private Test case

Input 1

```
[1,3,5,7,9]
[2,4,6,8,10]
```

Output

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Input 2

```
[1,3,5,7,9]
[1,3,5,7,9]
```

Output

```
[1, 1, 3, 3, 5, 5, 7, 7, 9, 9]
```

Input 3

```
[1,3,5,7,9]
[2]
```

Output

```
[1, 2, 3, 5, 7, 9]
```

Input 4

```
[2]
[2,3,4,5,6,7,8,9]
```

Output

```
[2, 2, 3, 4, 5, 6, 7, 8, 9]
```

Live Coding Problem 4

Write a function **expanding(L)** that takes a list of integer `L` as input and returns `True` if the absolute difference between each adjacent pair of elements strictly increases.

Sample Input

```
[1,3,7,2,9]
```

Output

```
True
```

Sample Input

```
[1,3,7,2,-3]
```

Output

```
False
```

Solution

```
# Solution
def expanding(l):
    if len(l) <= 2:
        return(True)
    diff = abs(l[1]-l[0])
    return(diff < abs(l[2]-l[1]) and expanding(l[1:]))

def expanding_iterative(l):
    if len(l) <= 2:
        return(True)
    olddiff = abs(l[1]-l[0])
    for i in range(2,len(l)):
        newdiff = abs(l[i]-l[i-1])
        if newdiff <= olddiff:
            return(False)
        olddiff = newdiff
    return(True)

# Suffix (Visible)
L = eval(input())
print(expanding(L))
```

Public Test case

Input 1

```
[1,3,7,2,9]
```

Output

```
True
```

Input 2

```
[1,3,7,2,-3]
```

Output

```
False
```

Input 3

```
[1,3,7,10]
```

Output

```
False
```

Private Test case

Input 1

```
[2,3,5,8,4,9,3]
```

Output

```
True
```

Input 2

```
[2,3,5,8,12,15]
```

Output

```
False
```

Input 3

```
[2,2,2,2,2]
```

Output

```
False
```

Input 4

```
[10,9,7,4,0,-6]
```

Output

```
True
```

Live Coding Problem 5

Write a Python function `sumsquare(l)` that takes a nonempty list of integers as input and returns a list `[odd,even]`, where `odd` is the sum of squares all the odd numbers in `l` and `even` is the sum of squares of all the even numbers in `l`.

Sample Input

```
[1,3,5]
```

Output

```
[35, 0]
```

Sample Input

```
[-1,-2,3,7]
```

Output

```
[59, 4]
```

Solution

```
# Solution
def even(n):
    return(n%2 == 0)
def sumsquare(l):
    oddsum = 0
    evensum = 0
    for n in l:
        if even(n):
            evensum += n*n
        else:
            oddsum += n*n
    return([oddsum,evensum])


# Suffix (Visible)
L = eval(input())
print(sumsquare(L))
```

Public Test case

Input 1

```
[1,3,5]
```

Output

```
[35, 0]
```

Input 2

```
[-1,-2,3,7]
```

Output

```
[59, 4]
```

Input 3

```
[2,4,6]
```

Output

```
[0, 56]
```

Private Test case

Input 1

```
[1,2,3,4,5,6]
```

Output

```
[35, 56]
```

Input 2

```
[1,2,-1,-2]
```

Output

```
[2, 8]
```

Input 3

```
[1,3,5,7,9]
```

Output

```
[165, 0]
```

Input 4

```
[2,4,6,8]
```

Output

```
[0, 120]
```

Live Coding Problem 6

Write a Python function `histogram(l)` that takes as input a list of integers with repetitions and returns a list of pairs as follows:.

- for each number `n` that appears in `l`, there should be exactly one pair `(n,r)` in the list returned by the function, where `r` is the number of repetitions of `n` in `l`.
- the final list should be sorted in ascending order by `r`, the number of repetitions. For numbers that occur with the same number of repetitions, arrange the pairs in ascending order of the value of the number.

Sample Input

```
[13,12,11,13,14,13,7,7,13,14,12]
```

Output

```
[(11, 1), (7, 2), (12, 2), (14, 2), (13, 4)]
```

Sample Input

```
[13,7,12,7,11,13,14,13,7,11,13,14,12,14,14,7]
```

Output

```
[(11, 2), (12, 2), (7, 4), (13, 4), (14, 4)]
```

Solution

```
# Solution
def build_table(l):
    # Use a dictionary to build a frequency table
    frequency = {}
    # For each number, create a new entry in the table or increment the
frequency
    for n in l:
        if n in frequency.keys():
            frequency[n] = frequency[n] + 1
        else:
            frequency[n] = 1

    return(frequency)

def sort_table(fdict):
```

```
    # First build a list of the form (r,n)
    flist = [ (fdict[n],n) for n in fdict.keys() ]
    # Sort this list using built in sort, which will sort first by frequency,
then by value
    flist.sort()
    # Flip each pair and return
    return( [ (n,r) for (r,n) in flist ])

def histogram(l):
    frequency_table = build_table(l)
    return(sort_table(frequency_table))



# Suffix code(Visible)
L=eval(input())
print(histogram(L))
```

Public Test case

Input 1

```
[7,12,11,13,7,11,13,14,12]
```

Output

```
[(14, 1), (7, 2), (11, 2), (12, 2), (13, 2)]
```

Input 2

```
[13,7,12,7,11,13,14,13,7,11,13,14,12,14,14,7]
```

Output

```
[(11, 2), (12, 2), (7, 4), (13, 4), (14, 4)]
```

Input 3

```
[13,12,11,13,14,13,7,7,13,14,12]
```

Output

```
[(11, 1), (7, 2), (12, 2), (14, 2), (13, 4)]
```

Private Test case

Input 1

```
[1,1,1,3,3,2,3,2,2,4,4,4,4,5,4]
```

Output

```
[(5, 1), (1, 3), (2, 3), (3, 3), (4, 5)]
```

Input 2

```
[1,2,1,2,1,2,1,2,1,2,1,2]
```

Output

```
[(1, 6), (2, 6)]
```

Input 3

```
[1,2,3,4,5]
```

Output

```
[(1, 1), (2, 1), (3, 1), (4, 1), (5, 1)]
```

Input 4

```
[1,1,1,1,1,1]
```

Output

```
[(1, 6)]
```

Live Coding Problem 7

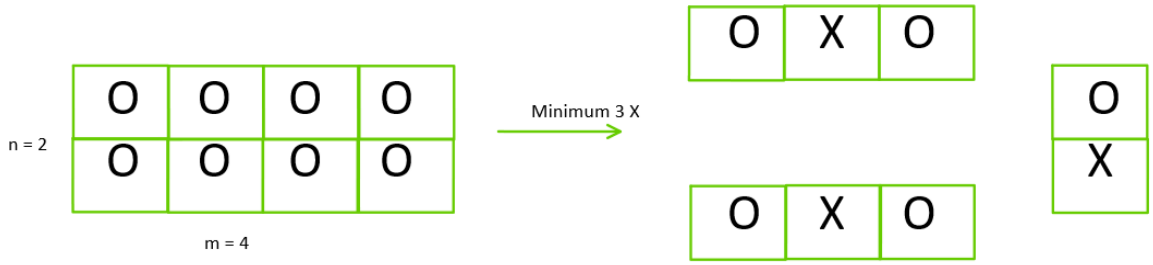
Given a wooden piece , a grid containing **n** rows and **m** columns, each 1 x 1 square containing **O** written inside it. You can cut the original or any other rectangular piece obtained during the cutting into two new pieces along the grid lines. You will obtain a certain number of rectangle pieces after doing the cutting. **1 x 1 is a square, you cannot treat it as a rectangle.**

Your task is to design each rectangular piece obtained in such a way that any pair of adjacent cells have different symbols.

Symbols : X and **O**

What would be the minimum number of cells you need to put an **X** on in an **n x m** grid to achieve the desired result ?

Example:- For n = 2 and m = 4, output is 3



Write a function **Min_X(n, m)** that accept the number of rows **n** and number of columns **m** as input and returns the minimum number of cells you need to put an **X** on in an **n x m** grid to achieve the desired result.

Sample Input 1

```
2 #n
4 #m
```

Output

```
3
```

Sample Input 2

```
3
1
```

Output

```
1
```

Solution

Solution Code

```
def Min_X(x,y):
    k = x // 3 * y
    x = x % 3
    k += y // 3 * x
    y = y % 3
    if 1 <= x * y <= 2:
```

```
        return (k + 1)
    elif x * y == 4:
        return (k + 2)
    else:
        return (k)

#Suffix hidden code
n = int(input())
m = int(input())
print(Min_X(n,m))
```

Public Test cases

Input 1

```
3
1
```

Output

```
1
```

Input 2

```
6
3
```

Output

```
6
```

Input 3

```
4
7
```

Output

```
10
```

Input 4

```
5
4
```

Output

```
7
```

Private Test cases

Input 1

```
1
2
```

Output

```
1
```

Input 2

```
2
2
```

Output

```
2
```

Input 3

```
2
5
```

Output

```
4
```

Input 4

```
3
5
```

Output

```
7
```

Live Coding Problem 8

Ram has an list of integers $a_1, a_2, \ldots\ldots a_n$ of size n. Each integer at index i denotes the money placed at that index i. He can do the following **operation exactly once**:

Pick a subsegment of the list and cyclically rotate it in the clockwise direction by any amount. i.e. pick integer l and r such that $1 \leq l \leq r \leq n$, and rotate the list $a_l, a_{l+1}, \ldots\ldots a_r$ in the clockwise direction by any amount. Ram wants the maximum amount of money by performing this particular operation exactly once. After performing the operation , Ram will collect $a_n - a_1$ amount of money.

Determine the maximum value of $a_n - a_1$ that he can obtain.

Write a function **Max_Amount(a)** that accept a list **a** and returns the maximum value of $a_n - a_1$ that Ram can obtain.

Sample Input

```
[1, 9, 8, 4, 6]
```

Output

```
8
```

Sample Input

```
[3, 2, 10, 8]
```

Output:

```
7
```

Solution

Solution Code

```python
def Max_Amount(a):
    n = len(a)
    if n == 1:
        return 0

    ans1 = a[-1] - min(a[:-1])

    ans2 = max(a[1:]) - a[0]

    ans3 = float("-inf")
    for i in range(n):
        ans3 = max(ans3 , a[(i-1) % n] - a[i] )
```

```
    ans = max(ans1,ans2,ans3)
    return ans

a = eval(input())
print(Max_Amount(a))
```

Public Test cases

Input 1

```
[1, 9, 8, 4, 6]
```

Output

```
8
```

Input 2

```
[3, 2, 10, 8]
```

Output:

```
7
```

Input 3

```
[6, 10, 2, 5, 10, 3, 1, 9, 10, 10]
```

Output

```
9
```

Input 4

```
[10, 7, 5, 1, 10, 10, 9, 7, 3, 2]
```

Output:

```
4
```

Input 5

```
[10, 7, 5, 1, 10, 10, 9, 7, 3, 2]
```

Output:

```
4
```

Private Test cases

Input 1

```
[5, 2, 10, 7, 6, 2, 1 ]
```

Output

```
5
```

Input 2

```
[7, 4, 1, 9, 6]
```

Output:

```
5
```

Input 3

```
[7, 8]
```

Output

```
1
```

Input 4

```
[9, 5, 4, 9, 3, 2, 3, 1]
```

Output:

```
6
```

Input 5

```
[10, 7, 5, 1, 10, 10, 9, 7, 3, 2]
```

Output:

```
4
```

###