16 Nov 2020 E, 20 Feb 2021 E, 9 April 2021 E, 20 Feb 2022 E

We know this famous painter Piet Mondrian whose paintings are partitioned into rectangles of colours like red, blue and yellow. We would like to draw something similar with white, yellow and red. Partition the painting into vertical strips of width A1, A2, A3, . . . , An and horizontal strips of height B1, B2, B3, . . . , Bn centimetres for some positive integer n > 0. These stripes split the painting into n × n rectangles. The intersection of vertical stripe i and horizontal stripe j has colour number (i + j − 2) mod 3 for all 1 ≤ i, j ≤ n. To prepare the painting, we want to know how much paint we need for each colour. We will measure it by the area of each colour in square centimetres.

In [1]:
```
def calculate_areas(w_list, h_list):

    n = len(w_list)
    whiteArea = 0
    redArea = 0
    yellowArea = 0

    widthSum1 = 0
    widthSum2 = 0
    widthSum3 = 0

    heightSum1 = 0
    heightSum2 = 0
    heightSum3 = 0

    for i in range(0,n,3):

        widthSum1 += w_list[i]
        heightSum1 += h_list[i]

    for i in range(1,n,3):

        widthSum2 += w_list[i]
        heightSum2 += h_list[i]

    for i in range(2,n,3):

        widthSum3 += w_list[i]
        heightSum3 += h_list[i]

    whiteArea = heightSum1*widthSum1 + heightSum2*widthSum3 + heightSum3*widthSum2
    redArea = heightSum1*widthSum3 + heightSum2*widthSum2 + heightSum3*widthSum1
    yellowArea = heightSum1*widthSum2 + heightSum2*widthSum1 + heightSum3*widthSum3

    return (whiteArea, yellowArea, redArea)
```

In [2]:
```
l1 = [6,2,4,5,1,1,4]
l2 = [2,5,1,4,2,3,4]
calculate_areas(l1,l2)
```

Out[2]:
```
(197, 155, 131)
```

In [3]:
```
l4 = [i for i in range(100000)]
calculate_areas(l4,l4)
```

Out[3]: (8333166668611088889, 8333166665277822222, 8333166668611088889)

In [4]:
```
l4 = [i for i in range(100000)]
calculate_areas(l4,l4[::-1])
```

Out[4]: (8333166669722177778, 8333166666388911111, 8333166666388911111)

In some culture, some digits are considered to be bad luck. For instance, the digit 4 is considered bad luck in the far east. When a digit is considered bad luck, some people do not want them to appear in any number such as phone numbers or license plate numbers. Write the function auspicious_number(n, bad) that returns how many auspicious numbers are there with exactly n digits and do not contain any numbers given in the list (list) of integer (int) bad .

In [5]:
```
def auspicious_number(n, bad):

    good = []

    for num in range(0,10):

        if num not in bad:
            good.append(num)

    total = 1

    for i in range(n-1):

        total = total*len(good)

    if 0 in bad:

        return total*(len(good)-1)

    else:

        return total*(len(good)-1)
```

In [6]:
```
auspicious_number(3, [4])
```

Out[6]: 648

In [7]:
```
auspicious_number(2, [4])
```

Out[7]: 72

In [8]:
```
auspicious_number(2, [1, 3])
```

Out[8]: 56

In [9]:
```
auspicious_number(3, [1, 3])
```

Out[9]: 448

In [10]:
```
auspicious_number(50, [4])
```

Out[10]:
```
458113351761787849810187670902774464624095575112
```

Given two height h1 and h2, write function magicPotionTreatement to return string of potion processes to take h1 to h2. Potion A increases height by 1 and Potion B halves height if height is even number.

In [11]:
```
def recursiveTreatement (h1,h2,letters):

    if h1 == h2:

        return

    elif h1 == h2 -1:

        letters.append('A')
        return

    elif h1 > 2*h2 and h1%2 == 0:

        letters.append('B')
        recursiveTreatement (h1/2,h2,letters)

    elif h1 > 2*h2 and h1%2 == 1:

        letters.append('A')
        letters.append('B')
        recursiveTreatement ((h1+1)/2,h2,letters)

    elif h1 > h2 and h1 < 2*h2 and h1%2 == 0:

        letters.append('B')
        recursiveTreatement (h1/2,h2,letters)

    elif h1 > h2 and h1 < 2*h2 and h1%2 == 1:

        letters.append('A')
        letters.append('B')
        recursiveTreatement ((h1+1)/2,h2,letters)

    else:

        letters.append('A')
        recursiveTreatement (h1+1,h2,letters)
```

In [12]:
```
def magicPotionTreatment(h1, h2):

    letters = []
    recursiveTreatement (h1,h2,letters)

    return ''.join(letters)
```

In [13]:
```
magicPotionTreatment(123,5)
```

Out[13]:
```
'ABBABBBA'
```

In [14]:
```
magicPotionTreatment(4,3)
```

Out[14]:
```
'BA'
```

In [15]:
```
magicPotionTreatment(9,2)
```

Out[15]:
```
'ABABAB'
```

In [16]:
```
print(len(magicPotionTreatment(10**100+17,11)))
```

```
357
```

In [17]:
```
magicPotionTreatment(134217729,3)
```

Out[17]:
```
'ABABABABABABABABABABABABABABABABABABABABABABABABABABABAB'
```

Write an iterative version of the function per_cipher_i(s,n) to encrypt a string s with an interval n as mentioned above. In this task, you cannot use any recursion.

In [18]:
```
def per_cipher_i(iString,num):

    length = len(iString)
    final = []

    for i in range(0, length-length%num, num):

        result = ''

        for j in range(num):

            result += iString[i+num-j-1]

        final.append(result)

    end = ''

    if(length%num != 0):

        for i in range(length%num):

            end += iString[-1-i]

        final.append(end)

    finalResult = ''

    for x in final:

        finalResult += x

    return finalResult
```

In [19]:
```
print(per_cipher_i('12345678910',3))
```

```
32165498701
```

In [20]:
```
print(per_cipher_i(per_cipher_i('12345678910',3),3))
```

```
12345678910
```

In [21]:
```
 print(per_cipher_i('PE Part 1 is supposed to be easy',7))
```

```
traP EPs si 1 desoppu eb ot ysae
```

Write a recursion version of the function per_cipher_r(s,n)with the same functionality in Part 1 Task 1. However, you cannot use any loops or list comprehension in this task.

In [22]:
```
def per_cipher_r(s,num):
    if len(s) < num:
        return s[-1::-1]
    else:
        return  s[num-1:0:-1] + s[0] + per_cipher_r(s[num:],num)
```

In [23]:
```
print(per_cipher_r('12345678910',3))
```

```
32165498701
```

In [24]:
```
print(per_cipher_r(per_cipher_i('12345678910',3),3))
```

```
12345678910
```

In [25]:
```
print(per_cipher_r('PE Part 1 is supposed to be easy',7))
```

```
traP EPs si 1 desoppu eb ot ysae
```

Write a function sum_of_3(L,n) to return True if there exists 3 numbers in L with their sum equals to n, and return False otherwise.

In [26]:
```
def sum_of_3(array,target):
    refDict = {}
    for elt in array:
        refDict[elt] = True
    length = len(array)
    for i in range(length):
        for j in range(i+1,length):
            twoSum = array[i] + array[j]
            if refDict.get(target-twoSum) != None:
                return True
    return False
```

In [27]:
```
print(sum_of_3(tuple(range(1,1000)),2500))
```

```
True
```

In [28]:
```
print(sum_of_3(tuple(range(1,1000)),2998))
```

```
False
```

In [29]:
```
 print(sum_of_3(tuple(range(1,4000)),11994))
```

```
True
```

In [30]:
```
print(sum_of_3(tuple(range(1,4000)),11995))
```

```
True
```

The child dna is digit product of his parent. Write a function child_DNA(d) to return the child DNA from the parent.

In [31]:
```
def child_DNA(dna):

    dnaString = str(dna)

    result = 1

    for elt in dnaString:

        result = result*int(elt)

    return result
```

In [32]:
```
 print(child_DNA(262))
```

```
24
```

In [33]:
```
print(child_DNA(987161))
```

```
3024
```

In [34]:
```
def parent_mutated_DNA(dna):

    target = child_DNA(dna)
    result = ''

    if target < 10 :

        return target

    while target > 1:

        for i in range(9,1,-1):

            if target % i == 0:
                result =  str(i) + result
                target = int(target/i)
                break

    return result
```

In [35]:
```
print(parent_mutated_DNA(262))
```

```
38
```

In [36]:
```
print(parent_mutated_DNA(12131))
```

```
6
```

Implement a function isMartian(d) to return True if there exists a possible parent for the creature with DNA d, or False otherwise.

In [37]:
```
def isMartian(d):

    target = d

    while target > 1:

        flag = False
```

```
        for i in range(9,1,-1):

            if target % i == 0:
                flag = True
                target = int(target/i)
                break

        if flag == False:

            return False

    return True
```

In [38]:
```
print(isMartian(3024))
```

```
True
```

In [39]:
```
print(isMartian(16632))
```

```
False
```