10 Jun 2017, 9 Jun 2018, 16 June 2018 X, 6 June 2020

Write a function factors that returns a list of the factors for a positive integer n > 1. The factors should be ordered in ascending order.

In [1]:
```
def factors(n):

    result = [1]

    for i in range(2, n+1):

        if n % i == 0:

            result.append(i)

    return result
```

In [2]:
```
factors(2)
```

Out[2]:
```
[1, 2]
```

In [3]:
```
factors(13)
```

Out[3]:
```
[1, 13]
```

In [4]:
```
factors(18)
```

Out[4]:
```
[1, 2, 3, 6, 9, 18]
```

Write a function repetitions that given a string, returns the maximum number of repeated substrings within the string. How this works should be clear from the following examples.

In [5]:
```
def repetitions(word):

    current = [word[0]]
    currentIndex = 0
    currentLength = len(current)


    wordLength = len(word)
    wordIndex = 0

    count = 0


    while wordIndex < wordLength:


        if word[wordIndex] != current[currentIndex]:

            count = 1

            current.clear()

            for i in range(wordIndex+1):

                current.append(word[i])
```

```
            currentLength = len(current)
            currentIndex = 0
            wordIndex += 1

        else:

            if currentIndex == currentLength - 1:

                currentIndex = 0
                count += 1
                wordIndex += 1


            else:


                currentIndex += 1
                wordIndex += 1

    return count
```

In [6]:
```
repetitions("aaaaa")
```

Out[6]: 5

In [7]:
```
repetitions("ababab")
```

Out[7]: 3

In [8]:
```
repetitions("abababc")
```

Out[8]: 1

In [9]:
```
repetitions("abadbabcabadbabc")
```

Out[9]: 2

Write a function digit product that takes in a non-negative integer n and returns the product of all the digits of the integer.

In [10]:
```
def digit_product(n):

    numString = str(n)
    product = 1

    for character in numString:

        product *= int(character)


    return product
```

In [11]:
```
digit_product(1111)
```

Out[11]: 1

In [12]:
```
digit_product(123)
```

Out[12]:
```
6
```

In [13]:
```
digit_product(123041)
```

Out[13]:
```
0
```

In [14]:
```
def numArray(num,length,final):

    if len(num) == length:

        array = []
        for character in num:
            array.append(character)
        final.append(array)

    else:

        for i in range(len(num)):

            numCopy = num.copy()
            numCopy.pop(i)
            numArray(numCopy,length,final)
```

In [15]:
```
def max_digit_product(n,length):

    numString = str(n)
    start = []
    final = []
    maxNum = 0


    for character in numString:
        start.append(character)

    numArray(start,length,final)

    for array in final:

        product = 1

        for num in array:

            product *= int(num)

        maxNum = max(maxNum,product)

    return maxNum
```

In [16]:
```
max_digit_product(11123,1)
```

Out[16]:
```
3
```

In [17]:
```
max_digit_product(112311,2)
```

Out[17]:
```
6
```

In [18]:
```
max_digit_product(1111111,5)
```

Out[18]:
```
1
```

In [19]:
```
max_digit_product(189113451,2)
```

Out[19]:
```
72
```

Write a function pythagoras that takes in 3 points, which are pairs and return True if the 3 points make up a right angle triangle.

In [20]:
```
def  pythagoras(coord1,coord2,coord3):

    xval1 = (coord1[0] - coord2[0])**2
    xval2 = (coord2[0] - coord3[0])**2
    xval3 = (coord1[0] - coord3[0])**2

    yval1 = (coord1[1] - coord2[1])**2
    yval2 = (coord2[1] - coord3[1])**2
    yval3 = (coord1[1] - coord3[1])**2

    if xval1 + yval1 + xval2 + yval2 == xval3 + yval3:

        return True

    elif xval2 + yval2 + xval3 + yval3 == xval1 + yval1:

        return True

    elif xval1 + yval1 + xval3 + yval3 == xval2 + yval2:

        return True

    else:

        return False
```

In [21]:
```
pythagoras((0,0),(1,2),(1,1))
```

Out[21]:
```
False
```

In [22]:
```
pythagoras((0,0),(1,0),(0,1))
```

Out[22]:
```
True
```

In [23]:
```
pythagoras((0,0),(2,0),(0,2))
```

Out[23]:
```
True
```

Write a function count triangles that takes in a list of points and returns the number of right angled triangles that can be formed with the points in the list. You may assume that all the points are distinct.

In [24]:
```
def get3(pointList, final):

    if len(pointList) == 3:

        final.append(pointList)
        return

    else:

        for item in pointList:
```

```
            copyList = pointList.copy()
            copyList.remove(item)
            get3(copyList, final)
```

In [25]:
```
def count_triangles(pointTuple):

    pointList = []

    final = []

    count = 0

    for item in pointTuple:

        pointList.append(item)

    get3(pointList, final)
    #print(final)

    for item in final:

        if pythagoras(item[0],item[1],item[2]) == True:
            #print(item)
            count += 1

    return count
```

In [26]:
```
count_triangles(((0,0),(1,0),(0,1)))
```

Out[26]:
```
1
```

In [27]:
```
count_triangles(((0,0),(1,1),(2,1)))
```

Out[27]:
```
0
```

In [28]:
```
count_triangles(((0,0),(1,0),(0,1),(-1,-2)))
```

Out[28]:
```
2
```

In [29]:
```
count_triangles(((0,0),(1,0),(0,1),(1,1)))
```

Out[29]:
```
4
```

Your first task is to write a function ET numbers that takes in two parameters: number and mapping. The function must return the 'ET number' representation of number (which is a decimal integer) based on the mapping passed to it.

In [30]:
```
def ET_number(target, inputTuple):

    length = len(inputTuple)
    refDic = {}
    result = ""
    refTarget = target

    for i in range(length):

        refDic[i] = inputTuple[i]
```

```
    while refTarget > 0:

        modulo = refTarget%length
        result = refDic[modulo] + result
        refTarget = refTarget - modulo
        refTarget = refTarget/length

    return result
```

In [31]:
```
ET_number(5, ("0","1","2","3","4","5","6","7","8","9"))
```

Out[31]:
```
'5'
```

In [32]:
```
ET_number(20, ("9","8","7","6","5","4","3","2","1","0"))
```

Out[32]:
```
'79'
```

In [33]:
```
ET_number(10, ("0","1","2","3","4","5"))
```

Out[33]:
```
'14'
```

In [34]:
```
ET_number(6, ("0","4"))
```

Out[34]:
```
'440'
```

In [35]:
```
ET_number(5, ("1","0"))
```

Out[35]:
```
'010'
```

In [36]:
```
ET_number(10, ("a","b","c"))
```

Out[36]:
```
'bab'
```

Your second task is to write the function max ET number that takes a tuple of ET numbersand their corresponding mapping and returns the ET number with the highest value. You can assume that each ET number in ET numbers is of some non-negative value when converted into a decimal integer.

In [37]:
```
def max_ET_number(tuple1,tuple2):

    refDic = {}
    length = len(tuple2)
    count = 1
    maxValue = 0
    maxElt = ""

    for i in range(length):

        refDic[tuple2[i]] = i

    #print(refDic)
    for elt in tuple1:

        value = 0

        for i in range(len(elt)):


            value = value + ((refDic[elt[i]])*((length)**(len(elt)-1-i)))
```

```
        if value > maxValue:

            maxValue = value
            maxElt = elt

    return maxElt
```

In [38]:
```
max_ET_number( ("0","1","2","3","4","5"), ("0","1","2","3","4","5","6","7","8","9")
```

Out[38]: '5'

In [39]:
```
max_ET_number(("12","34","42","58"), ("0","1","8","3","5","4","6","7","2","9"))
```

Out[39]: '42'

In [40]:
```
max_ET_number(("19","20","21"), ("0","2","1","3","4","5","6","7","8","9"))
```

Out[40]: '19'

In [41]:
```
max_ET_number(("14","15"),("0","1","2","3","5","4"))
```

Out[41]: '14'

In [42]:
```
max_ET_number(("707","700","770"),("0","7"))
```

Out[42]: '770'

In [43]:
```
max_ET_number(("0","4","40","44","4004","4040"),("0","4"))
```

Out[43]: '4040'

In [44]:
```
max_ET_number(("317","311","713","413"),("7","1","3","4"))
```

Out[44]: '413'

In [45]:
```
max_ET_number(("aba", "abc", "ca", "cb"), ("a", "b", "c"))
```

Out[45]: 'cb'