1. Create a function that takes three arguments a, b, c and returns the sum of the numbers that are evenly divided by c from the range a, b inclusive ?

Examples:
`evenly_divisible(1, 10, 20) ➞ 0

No number between 1 and 10 can be evenly divided by 20.
evenly_divisible(1, 10, 2) ➞ 30

2 + 4 + 6 + 8 + 10 = 30
evenly_divisible(1, 10, 3) ➞ 18

3 + 6 + 9 = 18`

Ans:

In [3]:
```
a = int(input('Enter a:'))
b = int(input('Enter b:'))
c = int(input('Enter c:'))
def evenly_divisible(i,j,k):
    sumlist=[]
    for n in range(i,j+1):
        if n % c == 0:
            sumlist.append(n)
    print(f'evenly_divisible({i},{j},{k}) ➞ {sum(sumlist)}')
evenly_divisible(a,b,c)
#evenly_divisible(1,10,2)
#evenly_divisible(1,10,3)
```

```
Enter a:1
Enter b:10
Enter c:2
evenly_divisible(1,10,2) ➞ 30
```

2. Create a function that returns True if a given inequality expression is correct and False otherwise ?

Examples:
correct_signs("3 < 7 < 11") ➞ True
correct_signs("13 > 44 > 33 > 1") ➞ False
correct_signs("1 < 2 < 6 < 9 > 3") ➞ True

Ans:

In [4]:
```
def correct_signs():
    string = input('Enter the inequality: ')
    output = eval(string)
    print(f'correct_signs("{string}") ➞ {output}')
for x in range(3):
    correct_signs()
```

```
Enter the inequality: 3 < 7 < 11
correct_signs("3 < 7 < 11") ➞ True
Enter the inequality: 13 > 44 > 33 > 1
correct_signs("13 > 44 > 33 > 1") ➞ False
Enter the inequality: 1 < 2 < 6 < 9 > 3
correct_signs("1 < 2 < 6 < 9 > 3") ➞ True
```

3. Create a function that replaces all the vowels in a string with a specified character ?

Examples:
replace_vowels("the aardvark", "#") ➞ "th# ##rdv#rk"

replace_vowels("minnie mouse", "?") ⟶ "m?nn?? m??s?"
replace_vowels("shakespeare", "*") ⟶ "shksp**r"

2 ** 5 + 1 = 33
2 * 5 + 1 = 11
33 is a multiple of 11
is_curzon(10) ⟶ False

2 ** 10 + 1 = 1025
2 * 10 + 1 = 21
1025 is not a multiple of 21
is_curzon(14) ⟶ True

2 ** 14 + 1 = 16385
2 * 14 + 1 = 29
16385 is a multiple of 29`
Ans:

In [18]:
```
def replace_vowels(string,char):
    vowels = ['a','e','i','o','u','A','E','I','O','U']
    string = input('Enter a string: ')
    char = input('Enter special character: ')
    new = string
    for i in string:
        if i in vowels:
            new = new.replace(i,char)
    print(f'replace_vowels({string}, {char}) ⟶ {new}')
for x in range(3):
    replace_vowels(string,char)
```

```
Enter a string: the aardvark
Enter special character: #
replace_vowels(the aardvark, #) ⟶ th# ##rdv#rk
Enter a string: minnie mouse
Enter special character: ?
replace_vowels(minnie mouse, ?) ⟶ m?nn?? m??s?
Enter a string: shakespeare
Enter special character: *
replace_vowels(shakespeare, *) ⟶ sh*k*sp**r*
```

4. Write a function that calculates the factorial of a number recursively ?

Examples:
factorial(5) ⟶ 120
factorial(3) ⟶ 6
factorial(1) ⟶ 1
factorial(0) ⟶ 1

Ans:

In [4]:

```
n = int(input('Enter number:'))
def factorial(n):
    if n==0:
        return 1
    return n*factorial(n-1)
print(f'factorial({n}) ⟶ {factorial(n)}')
#factorial(n)
```

```
Enter number:5
factorial(5) ⟶ 120
```

5. Hamming distance is the number of characters that differ between two strings ?

To illustrate:
String1: "abcbba"
String2: "abcbda"
Hamming Distance: 1 - "b" vs. "d" is the only difference.
Create a function that computes the hamming distance between two strings.
Examples:
hamming_distance("abcde", "bcdef") ⟶ 5
hamming_distance("abcde", "abcde") ⟶ 0
hamming_distance("strong", "strung") ⟶ 1

Ans:

In [1]:

```
def hamming_distance():
    string1 = input('Enter the String1: ')
    string2 = input('Enter the String2: ')
    if len(string1) == len(string2):
        count = 0
        for i in range(len(string1)):
            if string1[i] != string2[i]:
                count = count+1
        print(f'hamming_distance({string1},{string2}) ⟶ {count}')
for x in range(3):
    hamming_distance()
```

```
Enter the String1: abcde
Enter the String2: bcdef
hamming_distance(abcde,bcdef) ⟶ 5
Enter the String1: abcde
Enter the String2: abcde
hamming_distance(abcde,abcde) ⟶ 0
Enter the String1: strong
Enter the String2: strung
hamming_distance(strong,strung) ⟶ 1
```