**1. For this challenge, forget how to add two numbers together. The best explanation on what to do for this function is this meme:**

**Examples:**  
meme\_sum(26, 39) ➞ 515  
# 2+3 = 5, 6+9 = 15  
# 26 + 39 = 515  
meme\_sum(122, 81) ➞ 1103  
# 1+0 = 1, 2+8 = 10, 2+1 = 3  
# 122 + 81 = 1103  
meme\_sum(1222, 30277) ➞ 31499

In [1]:

**def** meme\_sum(a,b):

a,b **=** str(a),str(b)

output **=** ''

**while** len(a) **!=** len(b):

**if** len(a) **<** len(b):

a **=**'0'**+**a

**else**:

b**=**'0'**+**b

**for** ele **in** range(len(a)):

output **+=** str(int(a[ele])**+**int(b[ele]))

print(f'meme\_sum{a,b} ➞ {output}')

meme\_sum(26, 39)

meme\_sum(122, 81)

meme\_sum(1222, 30277)

meme\_sum('26', '39') ➞ 515

meme\_sum('122', '081') ➞ 1103

meme\_sum('01222', '30277') ➞ 31499

**2. Given an integer, create a function that returns the next prime. If the number is prime, return the number itself.**

**Examples:**  
next\_prime(12) ➞ 13  
next\_prime(24) ➞ 29  
next\_prime(11) ➞ 11  
# 11 is a prime, so we return the number itself.

In [2]:

**def** next\_prime(in\_num):

in\_num\_clone **=** in\_num

**while** **True**:

**if** (in\_num**-**1)**%6** == 0 or (in\_num+1)%6 ==0 :

temp **=** in\_num

**for** ele **in** range(3,in\_num**+**1):

**if** in\_num**%ele** == 0 and ele != in\_num:

in\_num **=** in\_num**+**1

**break**

**if** temp **==** in\_num:

**break**

**else**:

in\_num **+=** 1

print(f'next\_prine({in\_num\_clone}) ➞ {in\_num}')

next\_prime(12)

next\_prime(24)

next\_prime(11)

next\_prine(12) ➞ 13

next\_prine(24) ➞ 29

next\_prine(11) ➞ 11

**3. If a person traveled up a hill for 18mins at 20mph and then traveled back down the same path at 60mph then their average speed traveled was 30mph.**

Write a function that returns the average speed traveled given an uphill time, uphill rate and a downhill rate. Uphill time is given in minutes. Return the rate as an integer (mph). No rounding is necessary.

**Examples:**  
ave\_spd(18, 20, 60) ➞ 30  
ave\_spd(30, 10, 30) ➞ 15  
ave\_spd(30, 8, 24) ➞ 12

In [3]:

**def** ave\_spd(up\_time,up\_speed,down\_speed):

distance **=** up\_speed**\***(up\_time**/**60)

down\_time **=** distance**/**down\_speed

output **=** (2**\***distance)**/**((up\_time**/**60)**+**down\_time)

print(f'ave\_spd{up\_time,up\_speed,down\_speed} ➞ {int(output)}')

ave\_spd(18, 20, 60)

ave\_spd(30, 10, 30)

ave\_spd(30, 8, 24)

ave\_spd(18, 20, 60) ➞ 30

ave\_spd(30, 10, 30) ➞ 15

ave\_spd(30, 8, 24) ➞ 12

**4. The Kempner Function, applied to a composite number, permits to find the smallest integer greater than zero whose factorial is exactly divided by the number.**

kempner(6) ➞ 3 1! = 1 % 6 > 0 2! = 2 % 6 > 0 3! = 6 % 6 === 0

kempner(10) ➞ 5 1! = 1 % 10 > 0 2! = 2 % 10 > 0 3! = 6 % 10 > 0 4! = 24 % 10 > 0 5! = 120 % 10 === 0

A Kempner Function applied to a prime will always return the prime itself.  
kempner(2) ➞ 2  
kempner(5) ➞ 5

Given an integer n, implement a Kempner Function.

**Examples:**  
kempner(6) ➞ 3  
kempner(10) ➞ 5  
kempner(2) ➞ 2

In [4]:

**def** kempner(in\_num):

**def** factorial(in\_num):

**if** in\_num **==** 1:

**return** 1

**else**:

**return** in\_num **\*** factorial(in\_num**-**1)

**for** ele **in** range(1,in\_num**+**1):

**if** factorial(ele)**%in\_num** == 0:

output **=** ele

**break**

print(f'kempner({in\_num}) ➞ {output}')

kempner(6)

kempner(10)

kempner(5)

kempner(2)

kempner(6) ➞ 3

kempner(10) ➞ 5

kempner(5) ➞ 5

kempner(2) ➞ 2

**5. You work in a factory, and your job is to take items from a conveyor belt and pack them into boxes. Each box can hold a maximum of 10 kgs. Given a list containing the weight (in kg) of each item, how many boxes would you need to pack all of the items?**

**Examples:**  
boxes([2, 1, 2, 5, 4, 3, 6, 1, 1, 9, 3, 2]) ➞ 5  
# Box 1 = [2, 1, 2, 5] (10kg)  
# Box 2 = [4, 3] (7kg)  
# Box 3 = [6, 1, 1] (8kg)  
# Box 4 = [9] (9kg)  
# Box 5 = [3, 2] (5kg)

In [5]:

**def** boxes(in\_list):

in\_list\_clone **=** in\_list**.**copy()

output **=** []

temp\_box **=** []

**while** **True**:

**if** len(in\_list) **!=** 0:

**if** sum(temp\_box) **<=** 10:

temp\_box**.**append(in\_list**.**pop(0))

**else**:

in\_list**.**insert(0,temp\_box**.**pop())

output**.**append(temp\_box)

temp\_box **=** []

**else**:

output**.**append(temp\_box)

temp\_box **=** []

**break**

print(f'boxes({in\_list\_clone}) ➞ {output} ➞ {len(output)}')

boxes([2, 1, 2, 5, 4, 3, 6, 1, 1, 9, 3, 2])

boxes([5, 5, 5, 5, 5, 5, 2, 3, 4, 5, 6])

boxes([2, 1, 2, 5, 4, 3, 6, 1, 1, 9, 3, 2]) ➞ [[2, 1, 2, 5], [4, 3], [6, 1, 1], [9], [3, 2]] ➞ 5

boxes([5, 5, 5, 5, 5, 5, 2, 3, 4, 5, 6]) ➞ [[5, 5], [5, 5], [5, 5], [2, 3, 4], [5, 6]] ➞ 5