**1. Create a function that takes a list and string. The function should remove the letters in the string from the list, and return the list.**

**Examples:**  
remove\_letters(["s", "t", "r", "i", "n", "g", "w"], "string") ➞ ["w"]  
remove\_letters(["b", "b", "l", "l", "g", "n", "o", "a", "w"], "balloon") ➞ ["b", "g", "w"]  
remove\_letters(["d", "b", "t", "e", "a", "i"], "edabit") ➞ []

In [1]:

**def** remove\_letters(in\_list,in\_string):

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

**for** ele **in** in\_string:

**if** ele **in** in\_list:

in\_list**.**remove(ele)

print(f'remove\_letters{in\_list\_copy,in\_string} ➞ {in\_list}')

remove\_letters(["s", "t", "r", "i", "n", "g", "w"], "string")

remove\_letters(["b", "b", "l", "l", "g", "n", "o", "a", "w"], "balloon")

remove\_letters(["d", "b", "t", "e", "a", "i"], "edabit")

remove\_letters(['s', 't', 'r', 'i', 'n', 'g', 'w'], 'string') ➞ ['w']

remove\_letters(['b', 'b', 'l', 'l', 'g', 'n', 'o', 'a', 'w'], 'balloon') ➞ ['b', 'g', 'w']

remove\_letters(['d', 'b', 't', 'e', 'a', 'i'], 'edabit') ➞ []

**2. A block sequence in three dimensions. We can write a formula for this one:**

Create a function that takes a number (step) as an argument and returns the amount of blocks in that step.

**Examples:**  
blocks(1) ➞ 5  
blocks(5) ➞ 39  
blocks(2) ➞ 12

In [2]:

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

depth **=** in\_num**\***3**+**((in\_num)**-**1)**\***1

height **=** [x **for** x **in** range(2,in\_num**+**2)]

print(f'blocks({in\_num}) ➞ {depth**+**sum(height)}')

blocks(1)

blocks(2)

blocks(3)

blocks(4)

blocks(5)

blocks(1) ➞ 5

blocks(2) ➞ 12

blocks(3) ➞ 20

blocks(4) ➞ 29

blocks(5) ➞ 39

**3. Create a function that subtracts one positive integer from another, without using any arithmetic operators such as -, %, /, +, etc.**

**Examples:**  
my\_sub(5, 9) ➞ 4  
my\_sub(10, 30) ➞ 20  
my\_sub(0, 0) ➞ 0

In [3]:

**from** operator **import** sub

**def** my\_sub(in\_one,in\_two):

output **=** sub(in\_one,in\_two) **if** in\_one **>=** in\_two **else** sub(in\_two,in\_one)

print(f'my\_sub{in\_one,in\_two} ➞ {output}')

my\_sub(5, 9)

my\_sub(10, 30)

my\_sub(0, 0)

my\_sub(5, 9) ➞ 4

my\_sub(10, 30) ➞ 20

my\_sub(0, 0) ➞ 0

**4. Create a function that takes a string containing money in dollars and pounds sterling (seperated by comma) and returns the sum of dollar bills only, as an integer.**

For the input string:

1. Each amount is prefixed by the currency symbol: $ for dollars and £ for pounds.
2. Thousands are represented by the suffix k. i.e. $4k = $4,000 and £40k = £40,000

**Examples:**  
add\_bill("d20,p40,p60,d50") ➞ 20 + 50 = 70  
add\_bill("p30,d20,p60,d150,p360") ➞ 20 + 150 = 170  
add\_bill("p30,d2k,p60,d200,p360") ➞ 2 \* 1000 + 200 = 2200

In [4]:

**def** add\_bill(in\_string):

out\_num **=** 0

**for** ele **in** in\_string**.**split(","):

**if** 'd' **in** ele:

**if** 'k' **in** ele:

out\_num **+=** int(ele**.**replace('d','')**.**replace('k',''))**\***1000

**else**:

out\_num **+=** int(ele**.**replace("d",''))

print(f'add\_bill({in\_string}) ➞ {out\_num}')

add\_bill("d20,p40,p60,d50")

add\_bill("p30,d20,p60,d150,p360")

add\_bill("p30,d2k,p60,d200,p360")

add\_bill(d20,p40,p60,d50) ➞ 70

add\_bill(p30,d20,p60,d150,p360) ➞ 170

add\_bill(p30,d2k,p60,d200,p360) ➞ 2200

**5. Create a function that flips a horizontal list into a vertical list, and a vertical list into a horizontal list.**

In other words, take an 1 x n list (1 row + n columns) and flip it into a n x 1 list (n rows and 1 column), and vice versa.

**Examples:**  
flip\_list([1, 2, 3, 4]) ➞ [[1], [2], [3], [4]] # Take a horizontal list and flip it vertical.  
flip\_list([[5], [6], [9]]) ➞ [5, 6, 9] # Take a vertical list and flip it horizontal.  
flip\_list([]) ➞ []

In [5]:

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

**if** len(in\_list) **>** 0:

output **=** [ele[0] **for** ele **in** in\_list] **if** isinstance(in\_list[0],list) **else** [[ele] **for** ele **in** in\_list]

**else**:

output **=** []

print(f'flip\_list({in\_list}) ➞ {output}')

flip\_list([1,2,3,4])

flip\_list([[5],[6],[9]])

flip\_list([])

flip\_list([1, 2, 3, 4]) ➞ [[1], [2], [3], [4]]

flip\_list([[5], [6], [9]]) ➞ [5, 6, 9]

flip\_list([]) ➞ []