**1. Write four functions that directly mutate a list:**

1. repeat(lst, n): Repeat lst n times.
2. add(lst, x): Adds x to the end of the lst.
3. remove(lst, m, n): Removes all elements between indices m and n inclusive in lst.
4. concat(lst, x): concatenates lst with x (another list). **Examples:**  
   lst = [1, 2, 3, 4]  
   repeat(lst, 3) ➞ [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]  
   add(lst, 1) ➞ [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1]  
   remove(lst, 1, 12) ➞ [1]  
   concat(lst, [3, 4]) ➞ [1, 3, 4]

In [1]:

lst **=** [1, 2, 3, 4]

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

**global** lst

lst **=** lst**\***in\_num

**return** lst

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

**global** lst

lst**.**append(in\_num)

**return** lst

**def** remove(start,end):

**global** lst

**for** ele **in** lst[start:end**+**1]:

lst**.**remove(ele)

**return** lst

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

**global** lst

lst **=** lst**+**in\_list

**return** lst

print(f'repeat(3) ➞ {repeat(3)}')

print(f'add(1) ➞ {add(1)}')

print(f'remove(lst, 1, 12) ➞ {remove(1,12)}')

print(f'concat([3, 4]) ➞ {concat([3, 4])}')

repeat(3) ➞ [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]

add(1) ➞ [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1]

remove(lst, 1, 12) ➞ [1]

concat([3, 4]) ➞ [1, 3, 4]

**2. The classic game of Mastermind is played on a tray on which the Mastermind conceals a code and the Guesser has 10 tries to guess it. The code is a sequence of 4 (or 6, sometimes more) pegs of different colors. Each guess is a corresponding sequence of 4 (or more) pegs of different colors. A guess is "correct" when the color of every peg in the guess exactly matches the corresponding peg in the Mastermind's code.**

After each guess by the Guesser, the Mastermind will give a score comprising black & white pegs, not arranged in any order:

* Black peg == guess peg matches the color of a code peg in the same position.
* White peg == guess peg matches the color of a code peg in another position.

Create a function that takes two strings, code and guess as arguments, and returns the score in a dictionary.

* The code and guess are strings of numeric digits
* The color of the pegs are represented by numeric digits
* no "peg" may be double-scored

**Examples:**  
guess\_score("1423", "5678") ➞ {"black": 0, "white": 0}  
guess\_score("1423", "2222") ➞ {"black": 1, "white": 0}  
guess\_score("1423", "1234") ➞ {"black": 1, "white": 3}  
guess\_score("1423", "2211") ➞ {"black": 0, "white": 2}

In [2]:

**def** guess\_score(code,guess):

output **=** {"black":0,"white":0}

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

**if** code[ele] **==** guess[ele]:

output['black'] **+=** 1

**elif** code[ele] **in** guess **and** ele **!=** guess**.**index(code[ele]):

output['white'] **+=** 1

print(f'guess\_score{code,guess} ➞ {output}')

guess\_score("1423", "5678")

guess\_score("1423", "2222")

guess\_score("1423", "1234")

guess\_score("1423", "2211")

guess\_score('1423', '5678') ➞ {'black': 0, 'white': 0}

guess\_score('1423', '2222') ➞ {'black': 1, 'white': 0}

guess\_score('1423', '1234') ➞ {'black': 1, 'white': 3}

guess\_score('1423', '2211') ➞ {'black': 0, 'white': 2}

**3. Create a function that takes a list lst and a number N and returns a list of two integers from lst whose product equals N.**

**Examples:**  
two\_product([1, 2, -1, 4, 5], 20) ➞ [4, 5]  
two\_product([1, 2, 3, 4, 5], 10) ➞ [2, 5]  
two\_product([100, 12, 4, 1, 2], 15) ➞ None

In [3]:

**def** two\_product(in\_list,in\_num):

output **=** **None**

**for** num\_1 **in** in\_list:

**for** num\_2 **in** in\_list:

**if** num\_1**\***num\_2 **==** in\_num:

output **=** sorted([num\_1,num\_2])

**break**

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

two\_product([1, 2, **-**1, 4, 5], 20)

two\_product([1, 2, 3, 4, 5], 10)

two\_product([100, 12, 4, 1, 2], 15)

two\_product([1, 2, -1, 4, 5]) ➞ [4, 5]

two\_product([1, 2, 3, 4, 5]) ➞ [2, 5]

two\_product([100, 12, 4, 1, 2]) ➞ None

**4. In this challenge, sort a list containing a series of dates given as strings. Each date is given in the format DD-MM-YYYY\_HH:MM:**

"12-02-2012\_13:44"

The priority of criteria used for sorting will be:

* Year
* Month
* Day
* Hours
* Minutes

Given a list lst and a string mode, implement a function that returns:

* if mode is equal to "ASC", the list lst sorted in ascending order.
* if mode is equal to "DSC", the list lst sorted in descending order.

**Examples:**

sort\_dates(["10-02-2018\_12:30", "10-02-2016\_12:30", "10-02-2018\_12:15"], "ASC") ➞ ["10-02-2016\_12:30", "10-02-2018\_12:15", "10-02-2018\_12:30"]

sort\_dates(["10-02-2018\_12:30", "10-02-2016\_12:30", "10-02-2018\_12:15"], "DSC") ➞ ["10-02-2018\_12:30", "10-02-2018\_12:15", "10-02-2016\_12:30"]

sort\_dates(["09-02-2000\_10:03", "10-02-2000\_18:29", "01-01-1999\_00:55"], "ASC") ➞ ["01-01-1999\_00:55", "09-02-2000\_10:03", "10-02-2000\_18:29"]

In [4]:

**from** datetime **import** datetime

**def** sort\_dates(in\_list,sort\_by):

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

in\_list\_unix **=** []

**for** ele **in** in\_list:

in\_list\_unix**.**append(datetime**.**strptime(ele, "%d-%m-%Y\_%H:%M")**.**timestamp())

in\_list\_unix **=** sorted(in\_list\_unix) **if** sort\_by **==** 'ASC' **else** sorted(in\_list\_unix, reverse**=True**)

output **=** []

**for** ele **in** in\_list\_unix:

output**.**append(datetime**.**fromtimestamp(ele)**.**strftime("%d-%m-%Y\_%H:%M"))

print(f'sort\_dates{in\_list,sort\_by}➞ {output}')

sort\_dates(["10-02-2018\_12:30", "10-02-2016\_12:30", "10-02-2018\_12:15"], "ASC")

sort\_dates(["10-02-2018\_12:30", "10-02-2016\_12:30", "10-02-2018\_12:15"], "DSC")

sort\_dates(["09-02-2000\_10:03", "10-02-2000\_18:29", "01-01-1999\_00:55"], "ASC")

sort\_dates(['10-02-2018\_12:30', '10-02-2016\_12:30', '10-02-2018\_12:15'], 'ASC')➞ ['10-02-2016\_12:30', '10-02-2018\_12:15', '10-02-2018\_12:30']

sort\_dates(['10-02-2018\_12:30', '10-02-2016\_12:30', '10-02-2018\_12:15'], 'DSC')➞ ['10-02-2018\_12:30', '10-02-2018\_12:15', '10-02-2016\_12:30']

sort\_dates(['09-02-2000\_10:03', '10-02-2000\_18:29', '01-01-1999\_00:55'], 'ASC')➞ ['01-01-1999\_00:55', '09-02-2000\_10:03', '10-02-2000\_18:29']

**5. Write a function that selects all words that have all the same vowels (in any order and/or number) as the first word, including the first word.**

**Examples:**  
same\_vowel\_group(["toe", "ocelot", "maniac"]) ➞ ["toe", "ocelot"]  
same\_vowel\_group(["many", "carriage", "emit", "apricot", "animal"]) ➞ ["many", "carriage", "apricot", "animal"]  
same\_vowel\_group(["hoops", "chuff", "bot", "bottom"]) ➞ ["hoops", "bot", "bottom"]

In [5]:

**def** same\_vowel\_group(in\_list):

vowels **=** ['a','e','i','o','u']

first\_ele **=** sorted(set([x **for** x **in** in\_list[0] **if** x **in** vowels]))

output **=** []

**for** ele **in** range(0,len(in\_list)):

vowels\_in\_word **=** [x **for** x **in** in\_list[ele] **if** x **in** first\_ele]

**if** sorted(first\_ele) **==** sorted(set(vowels\_in\_word)):

output**.**append(in\_list[ele])

print(f'same\_vowel\_group({in\_list}) ➞ {output}')

same\_vowel\_group(["toe", "ocelot", "maniac"])

same\_vowel\_group(["many", "carriage", "emit", "apricot", "animal"])

same\_vowel\_group(["hoops", "chuff", "bot", "bottom"])

same\_vowel\_group(['toe', 'ocelot', 'maniac']) ➞ ['toe', 'ocelot']

same\_vowel\_group(['many', 'carriage', 'emit', 'apricot', 'animal']) ➞ ['many', 'carriage', 'apricot', 'animal']

same\_vowel\_group(['hoops', 'chuff', 'bot', 'bottom']) ➞ ['hoops', 'bot', 'bottom']

**6. Create a function that takes a list of more than three numbers and returns the Least Common Multiple (LCM).**

**Examples:**  
lcm\_of\_list([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) ➞ 2520  
lcm\_of\_list([13, 6, 17, 18, 19, 20, 37]) ➞ 27965340  
lcm\_of\_list([44, 64, 12, 17, 65]) ➞ 2333760

In [6]:

**import** math

**def** lcm\_of\_list(in\_list):

output **=** in\_list[0]

**for** ele **in** range(1,len(in\_list)):

output **=** (output**\***in\_list[ele])**//**math**.**gcd(output,in\_list[ele])

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

lcm\_of\_list([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

lcm\_of\_list([13, 6, 17, 18, 19, 20, 37])

lcm\_of\_list([44, 64, 12, 17, 65])

lcm\_of\_list([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) ➞ 2520

lcm\_of\_list([13, 6, 17, 18, 19, 20, 37]) ➞ 27965340

lcm\_of\_list([44, 64, 12, 17, 65]) ➞ 2333760