1. Write four functions that directly mutate a list:

repeat(lst, n): Repeat lst n times.
add(lst, x): Adds x to the end of the lst.
remove(lst, m, n): Removes all elements between indices m and n inclusive in lst.
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]

Ans:

In [30]:

```
def repeat(lst,n):
    lst = lst*n
    return lst

def add(lst, x):
    lst.append(x)
    return lst

def remove(lst, m, n):
    for ele in lst[m:n+1]:
        lst.remove(ele)
    return lst

def concat(lst,x):
    lst = lst+x
    return lst
lst = [1, 2, 3, 4]
print(f'repeat(lst, 3) ⟶ {repeat(lst,3)}')
print(f'add(lst, 1) ⟶ {add(lst, 1)}')
print(f'remove(lst, 1, 12) ⟶ {remove(lst, 1, 12)}')
print(f'concat(lst, [3, 4]) ⟶ {concat(lst, [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]
remove(lst, 1, 12) ⟶ [1]
concat(lst, [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}

Ans:

In [31]:
```
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

Ans:

In [48]:
```
def two_product(in_list,N):
    output = None
    for i in in_list:
        for j in in_list:
            if i*j == N:
                output = sorted([i,j])
    print(f'two_product({in_list,N}) → {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], 20)) → [4, 5]
two_product(([1, 2, 3, 4, 5], 10)) → [2, 5]
two_product(([100, 12, 4, 1, 2], 15)) → 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"]

Ans:

In [1]:
```
from datetime import datetime
def sort_dates(in_list,sort_by):
    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"]

Ans:

In [2]:

```python
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']
```