Mehdi Maboudi: m.maboudi@tu-bs.de

Technical University of Braunschweig
Institute of Geodesy and Photogrammetry
***

Complete and send this completed worksheet (including its outputs and any supporting code outside of the worksheet(if any)) with your assignment submission.

This exercise is a starter to:

- Check and maybe extend your knowledge about **Python data structures**
  Learn more: Python Official Tutorial
- Check and maybe extend your knowledge about **Numpy**
  Quick overview of algebra and arrays in NumPy: NumPy quickstart
- Get familiar with **Functions** in Python
  Learn more: Python Official Tutorial about functions

Python data structures

String

Write your First Name in the string called `first_name` in lower case
Write your Last Name in the string called `last_name` in lower case

```
first_name = 'Nghi'
last_name = 'Trang'
```

E1.1 (1 pnt)

Change the first letter of both to capital

```
### Start of your code ##
First_name = first_name
Last_name = last_name

### End   of your code ##

print(First_name)
print(Last_name)
```

```
Nghi
Trang
```

Mehdi Maboudi

E1.2 (1 pnt)

Then put First_name and Last_name together in another string called `name`

```
### Start of your code ##
name = First_name +' '+ Last_name
### End   of your code ##

name

'Nghi Trang'
```

'Mehdi Maboudi'

E1.3 (1 pnt)

change all the letters of `name` to uppercase

```
### Start of your code ##
NAME = name.upper()
### End   of your code ##

NAME

'NGHI TRANG'
```

'MEHDI MABOUDI'

E1.4 (2 pnts)

count the number of letters (do not count white spaces) and put it in a variable called "occurrences"
check whether there is `I` (defines as "selected_character") in your NAME or not.

```
### Start of your code ##
def name_info(info,specific_character):
    stripped_name = info.replace(" ","") #Remove white spaces
    letter_count = len(stripped_name) #Count letter excluding white space
    specific_count = stripped_name.lower().count(specific_character.lower()) #Format all to lowercase and count specific number
    return letter_count, specific_count #Return
selected_character = "I"
```

```
NAME_count, occurrences = name_info(NAME,selected_character)

### End   of your code ##


print(NAME,'has',NAME_count,'letters')

# you can see here 3 different types of string formatting in Python
print('%d  %s in:  %s '%((0 if occurrences==-1 else occurrences),selected_character,NAME))  # % Operator
print("The first letter of your name is {}.".format(NAME[0])) # str.format
print(f"The last letter of your name is {NAME[-1]}.") # f-Strings (Python 3.6+)

#f-string formatting is modern and looks more readable
TUBS_email = 'tu-bs.de'
print(f"\n{First_name[0]}.{Last_name}@{TUBS_email} could be a proper email-address for you.")
```

```
NGHI TRANG has 9 letters
1  I in:  NGHI TRANG
The first letter of your name is N.
The last letter of your name is G.

N.Trang@tu-bs.de could be a proper email-address for you.
```

MEHDI MABOUDI has 12 letters
2 I in: MEHDI MABOUDI
The first letter of your name is M.
The last letter of your name is I.

M.Maboudi@tu-bs.de could be a proper email-address for you.

Set

Sets are `Unordered` collections of `unique` elements
Let's create some sets

```
Scientific={'Numpy','Scipy'}
Editors = set(['Spyder','vsCode'])
Visualization=set()
Visualization.add('Gluviz')
Visualization.update(['Matplotlib'])
```

E1.5 (2 pnts)

Check whether 'seaborn' is in 'Visualization' set or not: use a boolean variable called "check_seaborn"
if it is not in 'Visualization' set, add it to the list

```
print(f"Visualization is now:{Visualization}")

### Start of your code ##
check_seaborn = 'seaborn' in Visualization
if not check_seaborn:
    Visualization.add('seaborn')
### End   of your code ##


print(f"seaborn was {'not' if check_seaborn == False else '' } in the
Visualization list.")
print(f"Visualization is now:{Visualization}")

Visualization is now:{'Matplotlib', 'Gluviz'}
seaborn was not in the Visualization list.
Visualization is now:{'Matplotlib', 'Gluviz', 'seaborn'}
```

Visualization is now:{'Gluviz', 'Matplotlib'} seaborn was not in the Visualization list. Visualization is now:{'Gluviz', 'seaborn', 'Matplotlib'}

E1.6 (1 pnts)

Put all in one set called `tools`

```
### Start of your code ##
tools = set()
tools.update(Scientific,Visualization,Editors)
### End   of your code ##

tools

{'Gluviz', 'Matplotlib', 'Numpy', 'Scipy', 'Spyder', 'seaborn',
'vsCode'}
```

{'Gluviz', 'Matplotlib', 'Numpy', 'Scipy', 'Spyder', 'seaborn', 'vsCode'}

let's create 'Browser_based' list

```
Browser_based = set(['Jupyterlab','Jupyter
Notebook','Colab','pandas'])
print(f"Browser_based set is now:{Browser_based}")

Browser_based set is now:{'Jupyter Notebook', 'Colab', 'pandas',
'Jupyterlab'}
```

E1.7 (2 pnts)

Ooops! remove 'pandas' from 'Browser_based' set and then add 'Browser_based' to tools

```
### Start of your code ##
Browser_based.remove('pandas')
tools.update(Browser_based)
### End   of your code ##

print(f"Browser_based set is now:{Browser_based}")
print(f"tools set is updated to:{tools}")
```

```
Browser_based set is now:{'Jupyter Notebook', 'Colab', 'Jupyterlab'}
tools set is updated to:{'Colab', 'Jupyter Notebook', 'vsCode',
'Matplotlib', 'Spyder', 'Gluviz', 'Numpy', 'Scipy', 'seaborn',
'Jupyterlab'}
```

Browser_based set is now:{'Jupyterlab', 'Colab', 'Jupyter Notebook'} tools set is updated to: {'Scipy', 'Numpy', 'Gluviz', 'Jupyter Notebook', 'Jupyterlab', 'seaborn', 'Colab', 'vsCode', 'Matplotlib', 'Spyder'}

List

List is a collection which is ordered and changeable. Allows duplicate members.
you can store any type inside a list

E1.8) (2 pnts)

Given the list of integers B below, find the most frequent integer in that list and store it in `most_frequent` variable

```
import random
random.seed(57)
B = [random.randint(0,10) for i in range(20)]
print(f'B={B}')

### Start of your code ##
def highest_frequent_num(num_list):
    if not num_list:    #In case list is empty
        return None
    frequency = {}  #Set frequency dictionary for tracking
    max_count = 0   #Set max_count variable to store highest frequency
    most_frequent = num_list[0] #Set first element as starting point
    for num in num_list:
        if num in frequency:    #Add to value of num in frequency
            frequency[num] += 1
        else:                       #If not exist, set value of 1 for
num in frequency
            frequency[num] = 1
        if frequency[num] > max_count:  #Set the new max count to
compare and update most frequent num
            max_count = frequency[num]
            most_frequent = num
    return most_frequent
most_frequent = highest_frequent_num(B)
### End   of your code ##

print(f'most_frequent number is = {most_frequent}')

B=[0, 5, 9, 9, 0, 3, 8, 5, 7, 4, 6, 10, 6, 0, 2, 8, 9, 5, 6, 6]
most_frequent number is = 6
```

B=[0, 5, 9, 9, 0, 3, 8, 5, 7, 4, 6, 10, 6, 0, 2, 8, 9, 5, 6, 6] most_frequent number is =6

E1.9) (2 pnts)

Given the list C below

- count number of occurrences of each value in list C (without using any external library)

```
random.seed(57)
C = [random.randint(0,10) for i in range(20)]
print(sorted(C))

### Start of your code ##
# hint: use list comprehension
def occur(num_list):
    unique_numbers = sorted(set(num_list))  #Extract the unique number
    frequency = [num_list.count(num) for num in unique_numbers] #Count
frequency of unique number in num list
    return frequency
### End   of your code ##
count_C = occur(C)
count_C

[0, 0, 0, 2, 3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 8, 8, 9, 9, 9, 10]

[3, 1, 1, 1, 3, 4, 1, 2, 3, 1]
```

[0, 0, 0, 2, 3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 8, 8, 9, 9, 9, 10] [3, 1, 1, 1, 3, 4, 1, 2, 3, 1]

- count number of occurrences of each value in list C (use any library you like)

```
### Start of your code ##
from collections import Counter
count_C = [count for n, count in sorted(Counter(C).items())]
#Create sorted list of tuples containing unique number and count and
extracts the frequency to count_C
### End   of your code ##

print(sorted(C))
count_C

[0, 0, 0, 2, 3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 8, 8, 9, 9, 9, 10]
```

```
[3, 1, 1, 1, 3, 4, 1, 2, 3, 1]
```

[0, 0, 0, 2, 3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 8, 8, 9, 9, 9, 10] array([3, 0, 1, 1, 1, 3, 4, 1, 2, 3, 1], dtype=int64)

E1.10) (3 pnts)

Suppose y_true is a list (that contains true class labels), and
y_pred is another list (with predicted class labels from some machine learning task.)
Calculate the **prediction accuracy in percent** (without using any external libraries like NumPy or scikit-learn).
Hint: scikitlearn help example

```
y_true = [1, 2, 0, 1, 1, 2, 3, 1, 2, 1]
y_pred = [1, 2, 1, 1, 1, 0, 3, 1, 2, 1]


correct = 0
total_elements = 0

### Start of your code ##
total_elements = len(y_true)
for t, p in zip(y_true,y_pred):
    if t == p:
        correct += 1
accuracy = (correct/total_elements)*100
### End   of your code ##


print(f'accuracy: {accuracy:.2f}')
```

```
accuracy: 80.00
```

accuracy: 80.00

Dictionary

A dictionary is a collection which is unordered, changeable and indexed.
In Python dictionaries are written with curly brackets, and they have keys and values. thisdict = { "brand": "BMW", "model": "X3", "year": 2005 }

E1.11) (3 pnts)

Given the list B below, create a dictionary
and save each member of B and its frequency (number of occurances) as one item in a dictionary called `B_and_freq`

```
random.seed(57)
B = [random.randint(0,10) for i in range(25)]
print(f'B={B}')
```

```
### Start of your code ##
B_and_freq = {}
for num in sorted(B):
    if num in B_and_freq:
        B_and_freq[num] += 1
    else:
        B_and_freq[num] = 1

### End   of your code ##

print(f'B and frequencies ={B_and_freq}')
```

```
B=[0, 5, 9, 9, 0, 3, 8, 5, 7, 4, 6, 10, 6, 0, 2, 8, 9, 5, 6, 6, 1, 10,
9, 10, 7]
B and frequencies ={0: 3, 1: 1, 2: 1, 3: 1, 4: 1, 5: 3, 6: 4, 7: 2, 8:
2, 9: 4, 10: 3}
```

B=[0, 5, 9, 9, 0, 3, 8, 5, 7, 4, 6, 10, 6, 0, 2, 8, 9, 5, 6, 6, 1, 10, 9, 10, 7] B and frequencies ={0: 3, 1: 1, 2: 1, 3: 1, 4: 1, 5: 3, 6: 4, 7: 2, 8: 2, 9: 4, 10: 3}

E1.12) (3 pnts)

now find the most frequent integer (as "most_frequent") and its number of occurrences (as "occurrences")

```
### Start of your code ##
most_frequent = max(B_and_freq,key=B_and_freq.get)
occurrences = B_and_freq[most_frequent]
### End   of your code ##

print(f'most_frequent number is {most_frequent} which is repeated
{occurrences} times')
```

```
most_frequent number is 6 which is repeated 4 times
```

E1.13 (1 pnts)

Check whether "most_frequent" is an integer

```
### Start of your code ##
if isinstance(most_frequent,int):
    data_type = 'Yes it is'
else:
    data_type = 'No it is not'
### End   of your code ##

print(f'Is {most_frequent} an integer?  {data_type}')
```

```
Is 6 an integer?  Yes it is
```

Numpy

some basics

E2.1) (1 pnt)

Import the NumPy library to create a 3x3 array with values ranging 0-8.
Then add an operation to get the output (H) which should look as follows:

array([[10, 11, 12],
[13, 14, 15],
[16, 17, 18]])

```
import numpy as np
### Start of your code ##
array = np.arange(9).reshape(3,3)
H = array + 10
### End   of your code ##

H
```

```
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

E2.2) (2 pnts)

create a 3x3 NumPy array (I) with random values in range [0,5] drawn from a uniform distribution using the random seed 57.

```
rng = np.random.RandomState(57)
### Start of your code ##
I = rng.uniform(0,5,(3,3))

### End   of your code ##

I
```

```
array([[0.43674821, 1.1523855 , 2.05530535],
       [1.55391349, 2.82977945, 2.72531852],
       [4.03549721, 4.59077554, 2.61045376]])
```

array([[0.43674821, 1.1523855 , 2.05530535], [1.55391349, 2.82977945, 2.72531852], [4.03549721, 4.59077554, 2.61045376]])

E2.3) (3 pnts)

let's create an array like A,
then use the NumPy slicing to put the 2x2 lower-right corner of A in another array called B

A = array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])

```
### Start of your code ##
A = np.arange(1, 17).reshape(4,4)
B = A[2:4,2:4]
### End   of your code ##

print(f'A=\n{A}\n')
print(f'B=\n{B}')
```

```
A=
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]

B=
[[11 12]
 [15 16]]
```

A= [[1 2 3 4] [5 6 7 8] [9 10 11 12] [13 14 15 16]]

B= [[11 12] [15 16]]

E2.4) (3 pnts)

Let's multiply all elements of A by 7
then print A and B again

```
A*=7

print(f'A=\n{A}\n')
print(f'B=\n{B}')
```

```
A=
[[  7  14  21  28]
 [ 35  42  49  56]
 [ 63  70  77  84]
 [ 91  98 105 112]]

B=
[[ 77  84]
 [105 112]]
```

What happened to B? Write Your Explanation.
This link may help you!

$\color{blue}{\textit Your Explanation:Because in numpy the syntax "B = A[2:4,2:4] which means B is a slice view of A so the data of A and B is shared, so when modify A, B also changes}$

E2.5) (2 pnts)

Change the way that you defined B to solve the issue inprevious part.

```
### Start of your code ##
B = np.copy(A[2:4,2:4])
### End   of your code ##

A*=7
print(f'A=\n{A}\n')
print(f'B=\n{B}')
```

```
A=
[[ 49  98 147 196]
 [245 294 343 392]
 [441 490 539 588]
 [637 686 735 784]]

B=
[[ 77  84]
 [105 112]]
```

A= [[7 14 21 28] [35 42 49 56] [63 70 77 84] [91 98 105 112]]

B= [[11 12] [15 16]]

E2.6) (2 pnts)

Given two arrays D and E below

- change the shape of E as D (row-major)
- change the shape of E as D (column-major)

```
rng = np.random.RandomState(57)
D = rng.randint(0, 10, (5,4))
E = rng.randint(0, 10, 20)
print(f'D=\n{D}\n')
print(f'E=\n{E}')
### Start of your code ##

E_row_major = E.reshape(D.shape,order='C') # row-major code here

E_col_major = E.reshape(D.shape,order='F') # column-major code here

### End   of your code ##
print(f'\nafter row-major reshape:\nE_row_major=\n{E_row_major}')
print(f'\nafter column-major reshape:\nE_row_major=\n{E_col_major}')
```

```
D=
[[6 5 6 8]
 [2 8 8 1]
 [7 9 5 8]
 [8 1 1 2]
 [0 1 6 4]]

E=
[5 9 6 2 8 5 2 0 1 7 2 2 1 3 5 7 5 8 0 6]

after row-major reshape:
E_row_major=
[[5 9 6 2]
 [8 5 2 0]
 [1 7 2 2]
 [1 3 5 7]
 [5 8 0 6]]

after column-major reshape:
E_row_major=
[[5 5 2 7]
 [9 2 2 5]
 [6 0 1 8]
 [2 1 3 0]
 [8 7 5 6]]
```

Results should look like this: ``` D= [[6 5 6 8] [2 8 8 1] [7 9 5 8] [8 1 1 2] [0 1 6 4]]

E= [5 9 6 2 8 5 2 0 1 7 2 2 1 3 5 7 5 8 0 6]

after row-major reshape: E_row_major= [[5 9 6 2] [8 5 2 0] [1 7 2 2] [1 3 5 7] [5 8 0 6]]

after column-major reshape: E_col_major= [[5 5 2 7] [9 2 2 5] [6 0 1 8] [2 1 3 0] [8 7 5 6]] ```

E2.7) (2 pnts)

Given the arrays F below, collapse array elements into one dimension.

- row-major in another array called `F_flattened_row_major` -column-major in another array called `F_flattened_column_major`

```python
rng = np.random.RandomState(57)
F = rng.randint(0, 10, (5,4))
print(f'\nF:\n{F}')

### Start of your code ##

# row-major code here
F_flattened_row_major = F.flatten(order='C')

# column-major code here
```

```
F_flattened_column_major = F.flatten(order='F')

### End   of your code ##
print(f'\nFlattened(row-major):\n{F_flattened_row_major}')
print(f'\nFlattened(column-major):\n{F_flattened_column_major}')
```

```
F:
[[6 5 6 8]
 [2 8 8 1]
 [7 9 5 8]
 [8 1 1 2]
 [0 1 6 4]]

Flattened(row-major):
[6 5 6 8 2 8 8 1 7 9 5 8 8 1 1 2 0 1 6 4]

Flattened(column-major):
[6 2 7 8 0 5 8 9 1 1 6 8 5 1 6 8 1 8 2 4]
```

Results should look like this: ``` F: [[6 5 6 8] [2 8 8 1] [7 9 5 8] [8 1 1 2] [0 1 6 4]]

Flattened(row-major): [6 5 6 8 2 8 8 1 7 9 5 8 8 1 1 2 0 1 6 4]

Flattened(column-major): [6 2 7 8 0 5 8 9 1 1 6 8 5 1 6 8 1 8 2 4] ```

Images

Images could be considered as arrays

E2.8) (9pnts)

E2.8.1) grayscale images (2pnts)

`grayscale images` just have one band(or layer), so they are 2D arrays (matrices)
Given the arrays *grayscale_image* below

```
grayscale_image = np.array(range(20)).reshape(4,5)

print(f'This grayscale_image is\
 {grayscale_image.shape[0]}x{grayscale_image.shape[-1]}\
 and\
 has {grayscale_image.ndim} dimenions.')

grayscale_image
```

```
This grayscale_image is 4x5 and has 2 dimenions.

array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
```

```
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

This grayscale_image is 4x5 and has 2 dimenions. array([[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19]])

reshape the image into one row (collapse array elements into one dimension.)

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

```
### Start of your code ##
grayscale_image_flattened = grayscale_image.flatten(order='C')
### End   of your code ##
grayscale_image_flattened

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
       17, 18, 19])
```

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

E2.8.2) RGB images (3pnts)

`RGB images` have 3 band(or layers), so they are 3D arrays (tensors)

Given the arrays *RGB_image* below

```
RGB_image = np.array(range(12)).reshape(3,2,2)
print(f'This RGB_image is\
 {RGB_image.shape[1]}x{RGB_image.shape[2]}\
 and\
 has {RGB_image.ndim} dimenions.')
RGB_image

This RGB_image is 2x2 and has 3 dimenions.

array([[[ 0,  1],
        [ 2,  3]],

       [[ 4,  5],
        [ 6,  7]],

       [[ 8,  9],
        [10, 11]]])
```

This RGB_image is 2x2 and has 3 dimenions. array([[[0, 1], [2, 3]],

```
   [[ 4,  5],
    [ 6,  7]],
   [[ 8,  9],
    [10, 11]]]])
```

reshape the image into one row and assign it a variable called `RGB_image_flattened`

```
### Start of your code ##
RGB_image_flattened = RGB_image.flatten(order='C')
### End   of your code ##
RGB_image_flattened

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

E2.8.3) More than one RGB image (4pnts)

Later in this course, we will also need to put many RGB images in a 2D array(Each flattened image as one row)
Given the 4D array `RGB_images` below (2 RGB images (each layar of each image is 2x2)

```
RGB_images = np.array(range(24)).reshape(2,3,2,2)


print(f'There are {RGB_images.shape[0]} RGB_images\
 : each of size\
 {RGB_images.shape[-2]}x{RGB_images.shape[-1]}.\
 So our tensor has {RGB_images.ndim} dimensions(4D).\
\nThe shape of the tensor is: {RGB_images.shape}')

print(f'  The first number  ({RGB_images.shape[0]}): indicated the 
number of images,\
\n  The second number ({RGB_images.shape[1]}): indicated the number of 
layers of images (Red,Green,Blue),\
\n  The third number  ({RGB_images.shape[2]}): indicated the number of 
rows in each layer,\
\n  The last number   ({RGB_images.shape[3]}): indicated the number of 
columns in each layer.\n')

print(f'RGB_images array look like:\n{RGB_images}')

There are 2 RGB_images : each of size 2x2. So our tensor has 4 
dimensions(4D).
The shape of the tensor is: (2, 3, 2, 2)
  The first number  (2): indicated the number of images,
  The second number (3): indicated the number of layers of images 
(Red,Green,Blue),
  The third number  (2): indicated the number of rows in each layer,
  The last number   (2): indicated the number of columns in each 
layer.
```

```
RGB_images array look like:
[[[[ 0  1]
   [ 2  3]]

  [[ 4  5]
   [ 6  7]]

  [[ 8  9]
   [10 11]]]


 [[[12 13]
   [14 15]]

  [[16 17]
   [18 19]]

  [[20 21]
   [22 23]]]]
```

There are 2 RGB_images : each of size 2x2. So our tensor has 4 dimensions(4D). The shape of the tensor is: (2, 3, 2, 2) The first number (2): indicated the number of images, The second number (3): indicated the number of layers of images (Red,Green,Blue), The third number (2): indicated the number of rows in each layer, The last number (2): indicated the number of columns in each layer.

RGB_images array look like: [[[[0 1] [2 3]]

[[4 5] [6 7]]

[[8 9] [10 11]]]

[[[12 13] [14 15]]

[[16 17] [18 19]]

[[20 21] [22 23]]]]

reshape the images into a 2D array where each row corresponds to one image and put it in a variable called `RGB_images_flattened`

```
### Start of your code ##
RGB_images_flattened =
RGB_images.reshape(RGB_images.shape[0],np.prod(RGB_images.shape[1:]))
### End   of your code ##
RGB_images_flattened
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]])
```

RGB_images_flattened = [[0 1 2 3 4 5 6 7 8 9 10 11] [12 13 14 15 16 17 18 19 20 21 22 23]]

Functions

Defining a function

Python official doc

E3.1) (8 pnts)

Write a function called `flatten_images` with:

- input: ND array called images as input
- output: 2D array images_flattened as discussed in E2.8.3

your function should be able to manage 1 or more grayscale or RGB image(s)

```
### Start of your code ##
def flatten_images(images):
    dimension_number = images.ndim
    if dimension_number == 4:
        return
images.reshape(images.shape[0],np.prod(images.shape[1:]))
    else:
        return images.flatten(order = 'C')
### End   of your code ##
```

Test

let's test your function

```
np.equal(grayscale_image_flattened,flatten_images(grayscale_image)).al
l()
```

```
True
```

If your function works correctly (for 1 grayscale image) the output of the line above would be:

True

If you run the next line and your function does not work correctly (for 1 RGB image) you will get an error,
otherwise nothing happens. It means the assertion is done.

```
assert np.equal(RGB_image_flattened ,flatten_images(RGB_image)).all()
== True
```

If you run the next line and your function does not work correctly (for more than 1 RGB images) you will get an error,
otherwise nothing happens. It means the assertion is done.

```
assert np.equal(RGB_images_flattened,flatten_images(RGB_images)).all()
== True
```

Defining another function

let's create two numpy arrays like X and y

```
import numpy as np
seed = 57
rng = np.random.default_rng(seed)

num_samples = 12
num_features = 3
num_classes = 2
X = rng.integers(0, 10, (num_samples,num_features))
y =rng.integers(0, num_classes, (num_samples))
print(f'X=\n{X}\n')
print(f'y={y}')
```

```
X=
[[0 6 8]
 [4 8 8]
 [9 3 2]
 [3 6 1]
 [4 4 3]
 [7 8 2]
 [1 3 2]
 [1 3 4]
 [5 9 9]
 [9 9 5]
 [5 6 0]
 [0 5 2]]

y=[0 1 1 0 1 0 0 1 1 1 1 1]
```

X= [[0 6 8] [4 8 8] [9 3 2] [3 6 1] [4 4 3] [7 8 2] [1 3 2] [1 3 4] [5 9 9] [9 9 5] [5 6 0] [0 5 2]]

y=[0 1 1 0 1 0 0 1 1 1 1 1]

E3.2) (15 pnts)

Given the function below called `split_to_2sets`

- Complete the inputs and outputs ducumentation (2pnts)
- Complete function implementation (8pnts)
  Splits each of X,y input arrays to 2 separate arrays with the same orders

Check also the function arguments. It should work properly, when you send different values for the parametrs
first_set_size could be a number in [0,1) or an integer>=1. When it is in [0,1): use it as percentage When it is an integer>=1: use it directly as first_set_size in our example above: if first_set_size=0.25, X1 should be 3x3 and X2 should be 9x3 if first_set_size=5, X1 should be 5x3 and X2 should be 7x3

You can get some hints from following links:
But do not use them directly and write your code just with low level numpy and python commands
numpy.random.choice

numpy.random.Generator.choice

sklearn.model_selection.train_test_split

```
def split_to_2sets(X,y,first_set_size=0.25,shuffle=True,RandomState_seed=57):
    """
    Splits each of X,y input arrays to 2 separate arrays with the same orders
    Inputs:
    - X: A numpy array of shape (N, D)
    - y: A numpy array of shape (N)
    write about next inputs  <<<<<<<<<<<<<<<<<<<<<<<
    - first_set_size: given as a percentage of the array (0<first_set_size<1) or the absolute size of the sample (first_set_size >=1)
    - shuffle: default is True, if set to false,
    - RandomState_seed: a variable to determine the seed for the random number generation for reproducibility purposes if needed
    outputs:
    X1: A numpy array of shape (N1_indices, D) which contains the first set
    X2: write about X2
    write about next outputs <<<<<<<<<<<<<<<<<<<<<<<
    """
#check the inputs
    assert X.shape[0]==y.shape[0]
    if  first_set_size<=0:
        raise ValueError('first_set_size should greater then zero')

    ### Start of your code ##
    #check other inputs to be valid

        # Check if random_state is set for reproducibility
    if RandomState_seed is not None:
        np.random.seed(RandomState_seed)
        # Determine the size of the first set
    if isinstance(first_set_size, float) and 0 <= first_set_size < 1:
        first_set_count = int(len(X) * first_set_size)
    elif isinstance(first_set_size, int) and first_set_size >= 1:
        first_set_count = first_set_size
    else:
        raise ValueError("first_set_size must be a float in [0, 1) or an int >= 1.")
    ### End of your code ##
    N = X.shape[0]  #number of samples
    ### Start of your code ##
    # separate X1,X2,y1,y2
    # Generate a random permutation of indices if shuffle = True, else keep the same order
    if shuffle:
        indices = np.random.permutation(len(X))
    else:
        indices = np.arange(len(X))
    # Split indices into two parts
    first_indices = indices[:first_set_count]
    second_indices = indices[first_set_count:]
    # Use indices to split the arrays
    X1, X2 = X[first_indices], X[second_indices]
    y1, y2 = y[first_indices], y[second_indices]
    ### End of your code ##

    return X1,X2,y1,y2
```

X1,X2,y1,y2 = split_to_2sets(X,y)

X1 = [[2 2 1] [9 6 2] [8 1 1]] X2 = [[9 5 8] [8 2 8] [0 1 7] [6 5 6] [8 1 7] [8 5 2] [3 5 7] [2 0 1] [6 4 5]] y1 = [0 1 0] y2 = [0 0 0 1 0 1 0 0 0]

Test your function

Testing your code is very important.There are many ways to test your code.
more details: this link or this link

Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.

There are many test runners available for Python. The one built into the Python standard library is called unittest. The most popular test runners are:

- pytest,
- unittest,
- nose
  and
- doctest

Relative benefits of pytest, unittest, nose, and doctest are described here ***

Here we do some very simple manual tests.

```
# test_00 to test_03: testing the size of outputs in different
situations

# test_00: test with default keyword arguments
X1,X2,y1,y2 = split_to_2sets(X,y)
assert  (X1.shape[0]+X2.shape[0]==X.shape[0]
         and
         y1.shape[0]+y2.shape[0]==y.shape[0])


# test_01: test changing RandomState_seed
X1,X2,y1,y2 = split_to_2sets(X,y,RandomState_seed=402)
assert (X1.shape[0]+X2.shape[0]==X.shape[0]
        and
        y1.shape[0]+y2.shape[0]==y.shape[0])


# test_02: test shuffle
X1,X2,y1,y2 = split_to_2sets(X,y,RandomState_seed=402,shuffle=False)
assert (X1.shape[0]+X2.shape[0]==X.shape[0]
        and
        y1.shape[0]+y2.shape[0]==y.shape[0])


# test_03: test first_set_size [0,1]
X1,X2,y1,y2 =
split_to_2sets(X,y,RandomState_seed=402,shuffle=False,first_set_size=.
3)
assert (X1.shape[0]+X2.shape[0]==X.shape[0]
        and
        y1.shape[0]+y2.shape[0]==y.shape[0])

# test_04 to test_06: testing edge cases for "first_set_size"

# test_04: test first_set_size to be 1
X1,X2,y1,y2 =
split_to_2sets(X,y,RandomState_seed=402,shuffle=False,first_set_size=1
)
assert (X1.shape[0]+X2.shape[0]==X.shape[0]
        and
        y1.shape[0]+y2.shape[0]==y.shape[0]
        and
        X1.shape[0]==1)

# test `edge cases` of `first_set_size`

# test_05: test first_set_size to be 0 : We considered this case in
our function
# We should get a ValueError like "first_set_size should greater then
```

```
zero"
X1,X2,y1,y2 =
split_to_2sets(X,y,RandomState_seed=402,shuffle=False,first_set_size=0
)
assert (X1.shape[0]+X2.shape[0]==X.shape[0]
        and
        y1.shape[0]+y2.shape[0]==y.shape[0])



# test_06: test first_set_size to be a negative value :We considered
this case in our function
# We should get a ValueError like "first_set_size should greater then
zero"
X1,X2,y1,y2 =
split_to_2sets(X,y,RandomState_seed=402,shuffle=False,first_set_size=-
1)
assert (X1.shape[0]+X2.shape[0]==X.shape[0]
        and
        y1.shape[0]+y2.shape[0]==y.shape[0])
```

```
----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[159], line 5
      1 # test `edge cases` of `first_set_size`
      2 
      3 # test_05: test first_set_size to be 0 : We considered this
case in our function
      4 # We should get a ValueError like "first_set_size should
greater then zero"
----> 5 X1,X2,y1,y2 =
split_to_2sets(X,y,RandomState_seed=402,shuffle=False,first_set_size=0
)
      6 assert (X1.shape[0]+X2.shape[0]==X.shape[0]
      7         and
      8         y1.shape[0]+y2.shape[0]==y.shape[0])
     12 # test_06: test first_set_size to be a negative value :We
considered this case in our function
     13 # We should get a ValueError like "first_set_size should
greater then zero"

Cell In[157], line 19, in split_to_2sets(X, y, first_set_size,
shuffle, RandomState_seed)
     17 assert X.shape[0]==y.shape[0]
     18 if  first_set_size<=0:
---> 19     raise ValueError('first_set_size should greater then
zero')
     21 ### Start of your code ##
     22 #check other inputs to be valid
     23 
     24     # Check if random_state is set for reproducibility
     25 if RandomState_seed is not None:

ValueError: first_set_size should greater then zero
```

```
#check an example outputs
Xt = np.array([[1,2],[3,4],[5,6]])
yt = np.array([0,1,1])

# test_07:
Xt1,Xt2,yt1,yt2 = split_to_2sets(Xt,yt,first_set_size=2,shuffle=False)
assert np.equal(Xt1,np.array([[1,2],[3,4]])).all()

# test_08:
Xt1,Xt2,yt1,yt2 =
split_to_2sets(Xt,yt,first_set_size=0.34,shuffle=False)
assert np.equal(Xt2,np.array([[3,4],[5,6]])).all()

# test_09:
Xt1,Xt2,yt1,yt2 =
split_to_2sets(Xt,yt,first_set_size=0.7,shuffle=False)
assert np.equal(Xt1,np.array([[1,2],[3,4]])).all()

# test_10:
Xt1,Xt2,yt1,yt2 =
split_to_2sets(Xt,yt,first_set_size=0.1,shuffle=False)
assert (Xt1.size == 0) and (Xt2.size == Xt.size)
```

check function documentation

```
split_to_2sets.__doc__

'\n    Splits each of X,y input arrays to 2 separate arrays with the
same orders\n    Inputs:\n    -X: A numpy array of shape (N, D)\n    -
y: A numpy array of shape (N)\n    write about next inputs
<<<<<<<<<<<<<<<<<<<<<<<<<\n    - first_set_size: given as a percentage
of the array (0<first_set_size<1) or the absolute size of the sample
(first_set_size >=1)\n    - shuffle: default is True, if set to false,
\n    outputs:\n    X1: A numpy array of shape (N1_indices, D) which
contains the first set\n    X2: write about X2\n    write about next
outputs <<<<<<<<<<<<<<<<<<<<<<<<<\n    '
```

We can also use

```
split_to_2sets?

Signature:
split_to_2sets(
    X,
    y,
    first_set_size=0.25,
    shuffle=True,
    RandomState_seed=57,
)
Docstring:
Splits each of X,y input arrays to 2 separate arrays with the same
orders
Inputs:
-X: A numpy array of shape (N, D)
-y: A numpy array of shape (N)
write about next inputs  <<<<<<<<<<<<<<<<<<<<<<<
- first_set_size: given as a percentage of the array
(0<first_set_size<1) or the absolute size of the sample
(first_set_size >=1)
- shuffle: default is True, if set to false,
outputs:
X1: A numpy array of shape (N1_indices, D) which contains the first
set
X2: write about X2
write about next outputs <<<<<<<<<<<<<<<<<<<<<<<
File:      c:\users\admin\appdata\local\temp\
ipykernel_28388\2247409205.py
Type:      function
```

or even

```
split_to_2sets??

Signature:
split_to_2sets(
    X,
    y,
    first_set_size=0.25,
    shuffle=True,
    RandomState_seed=57,
)
Source:
def
split_to_2sets(X,y,first_set_size=0.25,shuffle=True,RandomState_seed=5
7):
    """
    Splits each of X,y input arrays to 2 separate arrays with the same
orders
    Inputs:
    -X: A numpy array of shape (N, D)
    -y: A numpy array of shape (N)
    write about next inputs  <<<<<<<<<<<<<<<<<<<<<<<
    - first_set_size: given as a percentage of the array
(0<first_set_size<1) or the absolute size of the sample
(first_set_size >=1)
    - shuffle: default is True, if set to false,
    outputs:
    X1: A numpy array of shape (N1_indices, D) which contains the
first set
    X2: write about X2
    write about next outputs <<<<<<<<<<<<<<<<<<<<<<<
    """
    #check the inputs
    assert X.shape[0]==y.shape[0]
    if  first_set_size<=0:
        raise ValueError('first_set_size should greater then zero')

    ### Start of your code ##
     #check other inputs to be valid

        # Check if random_state is set for reproducibility
    if RandomState_seed is not None:
        np.random.seed(RandomState_seed)
        # Determine the size of the first set
    if isinstance(first_set_size, float) and 0 <= first_set_size < 1:
        first_set_count = int(len(X) * first_set_size)
    elif isinstance(first_set_size, int) and first_set_size >= 1:
        first_set_count = first_set_size
    else:
        raise ValueError("first_set_size must be a float in [0, 1) or
an int >= 1.")
    ### End of your code ##
    N = X.shape[0]  #number of samples
    ### Start of your code ##
    # separate X1,X2,y1,y2
    # Generate a random permutation of indices if shuffle = True, else
keep the same order
    if shuffle:
        indices = np.random.permutation(len(X))
    else:
        indices = np.arange(len(X))


    # Split indices into two parts
    first_indices = indices[:first_set_count]
    second_indices = indices[first_set_count:]
    # Use indices to split the arrays
    X1, X2 = X[first_indices], X[second_indices]
    y1, y2 = y[first_indices], y[second_indices]
    ### End of your code ##

    return X1,X2,y1,y2
```

```
File:      c:\users\admin\appdata\local\temp\
ipykernel_28388\2247409205.py
Type:      function
```

E3.3) (1 pnt for each valid test, Max 5)

Add as much as **new** *valid tests* you can on `edge cases` (boundary conditions) or inputs/outputs

```
def run_tests(random_seed=59):
    # Set seed for reproducibility
    np.random.seed(random_seed)
    random_sample_num = np.random.randint(6, 11)
    random_features_num = np.random.randint(2, 5)

    # Generate random arrays
    X = np.random.rand(random_sample_num, random_features_num)
    y = np.random.randint(0, 2, random_sample_num)

    # Test with a fractional first_set_size
    first_set_fraction = 0.4
    X1, X2, y1, y2 = split_to_2sets(X, y,
first_set_size=first_set_fraction, shuffle=False)
    expected_first_set_size = int(first_set_fraction *
random_sample_num)
    assert len(X1) == expected_first_set_size
    assert len(X2) == random_sample_num - expected_first_set_size
    np.testing.assert_array_equal(y1, y[:expected_first_set_size])
    np.testing.assert_array_equal(y2, y[expected_first_set_size:])

    # Test with an absolute first_set_size
    absolute_size = 3
    first_set_size = min(absolute_size, random_sample_num)
    X1, X2, y1, y2 = split_to_2sets(X, y,
first_set_size=absolute_size, shuffle=False)
    assert len(X1) == first_set_size
    assert len(X2) == random_sample_num - first_set_size
    np.testing.assert_array_equal(y1, y[:first_set_size])
    np.testing.assert_array_equal(y2, y[first_set_size:])

    # Test with shuffle enabled
    X1, X2, y1, y2 = split_to_2sets(X, y,
first_set_size=absolute_size, shuffle=True, RandomState_seed=42)
    assert len(X1) == first_set_size
    assert len(X2) == random_sample_num - first_set_size

    # Test with an invalid first_set_size
    try:
        split_to_2sets(X, y, first_set_size=0)
    except ValueError as e:
        assert str(e) == 'first_set_size should greater then zero'

    # Test with first_set_size greater than the length of X
    X1, X2, y1, y2 = split_to_2sets(X, y, first_set_size=15,
shuffle=False)
    assert len(X1) == random_sample_num
    assert len(X2) == 0

    # Test with empty arrays
    X_empty = np.array([]).reshape(0, random_features_num)
    y_empty = np.array([])
    X1, X2, y1, y2 = split_to_2sets(X_empty, y_empty,
first_set_size=0.5, shuffle=False)
    assert len(X1) == 0
    assert len(X2) == 0
    split_to_2sets??
    print("All tests passed, hope i pass the test :')")

run_tests()
```

```
All tests passed, hope i pass the test :')
```

```
Signature:
split_to_2sets(
    X,
    y,
    first_set_size=0.25,
    shuffle=True,
    RandomState_seed=57,
)
Source:
def
split_to_2sets(X,y,first_set_size=0.25,shuffle=True,RandomState_seed=5
7):
    """
    Splits each of X,y input arrays to 2 separate arrays with the same
orders
    Inputs:
    - X: A numpy array of shape (N, D)
    - y: A numpy array of shape (N)
    write about next inputs  <<<<<<<<<<<<<<<<<<<<<<<
    - first_set_size: given as a percentage of the array
(0<first_set_size<1) or the absolute size of the sample
(first_set_size >=1)
    - shuffle: default is True, if set to false,
    - RandomState_seed: a variable to determine the seed for the
random number generation for reproducibility purposes if needed
    outputs:
    X1: A numpy array of shape (N1_indices, D) which contains the
```

```
first set
    X2: write about X2
    write about next outputs <<<<<<<<<<<<<<<<<<<<<<<
    """
    #check the inputs
    assert X.shape[0]==y.shape[0]
    if  first_set_size<=0:
        raise ValueError('first_set_size should greater then zero')

    ### Start of your code ##
     #check other inputs to be valid

        # Check if random_state is set for reproducibility
    if RandomState_seed is not None:
        np.random.seed(RandomState_seed)
        # Determine the size of the first set
    if isinstance(first_set_size, float) and 0 <= first_set_size < 1:
        first_set_count = int(len(X) * first_set_size)
    elif isinstance(first_set_size, int) and first_set_size >= 1:
        first_set_count = first_set_size
    else:
        raise ValueError("first_set_size must be a float in [0, 1) or
an int >= 1.")
    ### End of your code ##
    N = X.shape[0]  #number of samples
    ### Start of your code ##
    # separate X1,X2,y1,y2
    # Generate a random permutation of indices if shuffle = True, else
keep the same order
    if shuffle:
        indices = np.random.permutation(len(X))
    else:
        indices = np.arange(len(X))
    # Split indices into two parts
    first_indices = indices[:first_set_count]
    second_indices = indices[first_set_count:]
    # Use indices to split the arrays
    X1, X2 = X[first_indices], X[second_indices]
    y1, y2 = y[first_indices], y[second_indices]
    ### End of your code ##

    return X1,X2,y1,y2
File:      c:\users\admin\appdata\local\temp\
ipykernel_28388\4189449194.py
Type:      function
```