lab1_???_17307110448

March 18, 2020

1 Lab 1 – The Basics of Python and Pytorch

17307110448

1.1 Write a Python function to sum all the numbers in a list.

In [1]:
```
def sum_all_nums(ls):
    s = 0
    for ele in ls:
        s = s + ele
    return s
```

In [2]:
```
ls = [1,2,3]
print(sum_all_nums(ls))
```

```
6
```

1.2 Write a Python function that takes a list and returns a new list with unique elements of the first list.

In [3]:
```
def unique_elements(nums):
    H = {}
    i = 0
    while i < len(nums) :
        if nums[i] in H.keys():
            nums.pop(i)
            continue
        else:
            H[nums[i]] = 1
            i = i + 1
    return ls
```

In [4]:
```
ls = [1, 2, 3, 3, 3, 3, 4, 5]
result = unique_elements(ls)
result
```

Out[4]:
```
[1, 2, 3, 4, 5]
```

1.3 Write a Python function that checks whether a passed string is palindrome or not. A palindrome is a word, phrase, or sequence that reads the same backward as forward, e.g., madam or nurses run.

```
In [5]: def palindrome(string):
            string = string.replace(' ', '')
            string = string.replace(',', '')
            string = string.replace('.', '')
            string2 = string[::-1]
            if string == string2:
                return True
            return False
```

```
In [6]: palindrome('madam')
```

```
Out[6]: True
```

```
In [7]: palindrome('nurses run')
```

```
Out[7]: True
```

```
In [8]: palindrome('you can, nac uoy.')
```

```
Out[8]: True
```

```
In [9]: palindrome('madlam')
```

```
Out[9]: False
```

1.4 Write a NumPy program to find the real and imaginary parts of an array of complex numbers

```
In [10]: import numpy as np
         a = np.array( [1.00000000+0.j, 0.70710678+0.70710678j])
```

```
In [11]: def complex_find(a):
             b = np.empty(shape = [0,2])
             for i,j in enumerate(a):
                 b = np.append(b, [[j.real, j.imag]], axis = 0)
             return b

         b = complex_find(a)
```

```
In [12]: b
```

```
Out[12]: array([[1.        , 0.        ],
                [0.70710678, 0.70710678]])
```

1.5 Write a Python program to add two binary numbers.

```
In [13]: b1 = '11'
         b2 = '1'
```

```
In [14]: def add_binary(b1, b2):
             a1 = 0
             a2 = 0
             l1 = list(b1)
             l1.reverse()
             l2 = list(b2)
             l2.reverse()
             for i in range(len(l1)):
                 a1 = a1 + int(l1[i])* 2**i
             for i in range(len(l2)):
                 a2 = a2 + int(l2[i])* 2**i

             s = int(a1 + a2)
             r = []
             while s != 0:
                 temp = s % 2
                 r.append(str(temp))
                 s = int(s/2)
             r.reverse()
             return ''.join(r)
```

```
In [15]: result = add_binary(b1, b2)
         result
```

```
Out[15]: '100'
```

1.6 You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

```
In [16]: # Definition for singly-linked list.
         # class ListNode(object):
         #     def __init__(self, x):
         #         self.val = x
         #         self.next = None

         class Solution(object):
             def addTwoNumbers(self, l1, l2):
                 """
                 :type l1: ListNode
                 :type l2: ListNode
                 :rtype: ListNode
                 """
```

```
        ls = 0
        flag = 0
        while l1 is not None and l2 is not None:
            ls = ls + l1.val * (10 ** flag) + l2.val * (10 ** flag)
            l1 = l1.next
            l2 = l2.next
            flag = flag + 1
        while l1 is not None:
            ls = ls + l1.val * (10 ** flag)
            l1 = l1.next
            flag = flag + 1
        while l2 is not None:
            ls = ls + l2.val * (10 ** flag)
            l2 = l2.next
            flag = flag + 1
        if ls == 0:
            return ListNode(0)
        r = []
        while ls != 0:
            r.append(ls % 10)
            ls = ls/10
        l3 = ListNode(r.pop(0))
        head = l3
        while len(r) != 0:
            l3.next = ListNode(r.pop(0))
            l3 = l3.next
        return head
```

1.7 Implement bubble sort

```
In [17]: def bubbleSort(lists):
             """
             :type lists: List[float]
             :rtype: List[float]
             """
             l = len(lists)
             for i in range(l-1):
                 for j in range(i+1, l):
                     if lists[i] > lists[j]:
                         temp = lists[i]
                         lists[i] = lists[j]
                         lists[j] = temp
             return lists
```

```
In [18]: ls=[2,4,1,6,8,3,0]
         ls=bubbleSort(ls)
         ls
```

```
Out[18]: [0, 1, 2, 3, 4, 6, 8]
```

1.8 Implement merge sort

In [19]:
```
def mergeSort(lists):
    """
    :type lists: List[float]
    :rtype: List[float]
    """

    def merge(lists,p,q,r):
        n_1 = q-p+1
        n_2 = r-q
        L = []
        R = []
        for i in range(0,n_1):
            L.append(lists[p+i-1])
        for j in range(0,n_2):
            R.append(lists[q+j])
        inf = float("inf")
        L.append(inf)
        R.append(inf)
        i = 0
        j = 0
        for k in range(p-1,r):
            if  L[i] <= R[j]:
                lists[k] = L[i]
                i = i+1
            else:
                lists[k] = R[j]
                j = j+1
        return lists

    def merge_Sort(lists,p,r):
        if p<r:
            q = int((p+r)/2)
            merge_Sort(lists,p,q)
            merge_Sort(lists,q+1,r)
            merge(lists,p,q,r)
        return lists

    A = lists[:]
    n = len(A)
    p = 1
    r = n
    B = merge_Sort(A,p,r)
    return B
```

In [20]:
```
ls=[2,4,1,6,8,3,0]
ls=mergeSort(ls)
ls
```

```
Out[20]: [0, 1, 2, 3, 4, 6, 8]
```

1.9 Implement quick sort

```
In [21]: def quick(A, p, r):
             if p < r:
                 q = partition(A, p, r)
                 quick(A, p, q-1)
                 quick(A, q+1, r)

         def partition(A, p, r):
             x = A[r]
             i = p-1
             for j in range(p, r):
                 if A[j] <= x:
                     i = i+1
                     A[i], A[j] = A[j], A[i]


             A[i+1], A[r] = A[r], A[i+1]
             return i+1

         def quickSort(A):
             """
             :type lists: List[float]
             :rtype: List[float]
             """
             n = len(A)
             p = 0
             r = n-1
             quick(A, p, r)

In [22]: ls=[2,4,1,6,8,3,0]
         quickSort(ls)
         ls

Out[22]: [0, 1, 2, 3, 4, 6, 8]
```

1.10 Implement shell sort

```
In [23]: def shellSort(A):
             l = len(A)
             key = round(l/2)
             while key > 0:
                 for i in range(key, l):
                     temp = A[i]
                     j = i
                     while j >= key and A[j-key] > temp:
```

```
                A[j] = A[j-key]
                j = j-key
            A[j] = temp
        key = round(key/2)
    return A
```

In [24]:
```
ls=[2,4,1,6,8,3,0]
ls=shellSort(ls)
ls
```

Out[24]:
```
[0, 1, 2, 3, 4, 6, 8]
```

1.11 Implement linear regression model and use autograd to optimize it by Pytorch.

In [25]:
```
import torch
import torch.nn as nn
import torch.optim as optim
```

In [26]:
```
SEED = 1234
torch.manual_seed(SEED)
x = torch.rand(256,1)
y = torch.rand(256,1)
```

In [27]:
```
model = nn.Linear(1,1)
loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr = 0.01)
epochs = 300
```

In [28]:
```
for i in range(epochs):
    inputs = x
    labels = y
    outputs = model(inputs)
    loss = loss_fn(outputs, labels)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if i % 50 == 0:
        print(i, loss.item())

print(i, loss.item())
```

```
0 0.3487284481525421
50 0.11589482426643372
100 0.09488625824451447
150 0.0905403345823288
200 0.08780194073915482
250 0.08547420799732208
299 0.08346918225288391
```

```
In [29]: model.state_dict().keys()
```

```
Out[29]: odict_keys(['weight', 'bias'])
```

```
In [30]: model.weight
```

```
Out[30]: Parameter containing:
         tensor([[0.4247]], requires_grad=True)
```

```
In [31]: model.bias
```

```
Out[31]: Parameter containing:
         tensor([0.2978], requires_grad=True)
```

1.12 Implement logistic regression model and use autograd to optimize it by Pytorch.

```
In [32]: import torch
         import torch.nn as nn
         import torch.optim as optim
         import torch.nn.functional as F
```

```
In [33]: class logistic(nn.Module):
             def __init__(self):
                 super(logistic, self).__init__()
                 self.fc = nn.Linear(2,1)

             def forward(self, x):
                 score = self.fc(x)
                 return F.sigmoid(score)
```

```
In [34]: SEED = 1234
         torch.manual_seed(SEED)
         x = torch.rand(256,2).type(torch.FloatTensor)
         y1 = torch.zeros(128,1)
         y2 = torch.ones(128,1)
         y = torch.cat((y1,y2),0)
```

```
In [35]: model = logistic()
         loss_fn = nn.BCEWithLogitsLoss()
         optimizer = torch.optim.Adam(model.parameters())
         epochs = 300
```

```
In [36]: for i in range(epochs):
             outputs = model(x)
             y = y.type_as(outputs)
             loss = loss_fn(outputs, y)
             optimizer.zero_grad()
             loss.backward()
             optimizer.step()
```

```
        if i % 50 == 0:
            print(i, loss.item())

    print(i, loss.item())
```

```
E:\anaconda3\lib\site-packages\torch\nn\functional.py:1351: UserWarning: nn.functional.sigmoid i
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

```
0 0.723075270652771
50 0.7202138900756836
100 0.7175729274749756
150 0.7151684761047363
200 0.7130001187324524
250 0.7110595703125
299 0.7093658447265625
```

1.13 Implement linear SVM model for binary classification task and use autograd to optimize it by Pytorch.

In [37]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
```

In [38]:
```
class SVM(nn.Module):
    def __init__(self):
        super(SVM, self).__init__()
        self.fc = nn.Linear(2,1)

    def forward(self, x):
        score = self.fc(x)
        return score
```

In [39]:
```
SEED = 1234
torch.manual_seed(SEED)
x = torch.rand(256,2).type(torch.FloatTensor)
y1 = torch.zeros(128,1)
y2 = torch.ones(128,1)
y = torch.cat((y1,y2),0)
```

In [40]:
```
model = SVM()
optimizer = torch.optim.Adam(model.parameters())
epochs = 300
```

In [41]:
```
for i in range(epochs):
    outputs = model(x)
    outputs = F.sigmoid(outputs)
```

```
        y = y.type_as(outputs)
        loss = torch.mean(torch.clamp(1 - outputs.t() * y, min=0))  # hinge loss
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if i % 50 == 0:
            print(i, loss.item())

    print(i, loss.item())
```

```
0 0.750423789024353
```

```
E:\anaconda3\lib\site-packages\torch\nn\functional.py:1351: UserWarning: nn.functional.sigmoid i
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

```
50 0.7379366159439087
100 0.7256088256835938
150 0.7135442495346069
200 0.7018307447433472
250 0.6905401945114136
299 0.6799389123916626
```

1.14 Add a Frobenius norm penalty for the weight w in your SVM model by two different ways..

l2 penalty

```
In [42]: model = SVM()
         optimizer = torch.optim.Adam(model.parameters())
         epochs = 300
```

```
In [43]: for i in range(epochs):
             outputs = model(x)
             outputs = F.sigmoid(outputs)
             y = y.type_as(outputs)
             loss = torch.mean(torch.clamp(1 - outputs.t() * y, min=0))  # hinge loss
             loss += 0.01 * torch.mean(model.fc.weight ** 2)  # l2 penalty
             optimizer.zero_grad()
             loss.backward()
             optimizer.step()
             if i % 50 == 0:
                 print(i, loss.item())

         print(i, loss.item())
```

```
0 0.6865286231040955
```

```
E:\anaconda3\lib\site-packages\torch\nn\functional.py:1351: UserWarning: nn.functional.sigmoid i
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

```
50 0.6754183769226074
100 0.6649170517921448
150 0.6550771594047546
200 0.645911693572998
250 0.6374163031578064
299 0.6297248601913452
```

l1 penalty

```
In [44]: model = SVM()
         optimizer = torch.optim.Adam(model.parameters())
         epochs = 300

In [45]: for i in range(epochs):
             outputs = model(x)
             outputs = F.sigmoid(outputs)
             y = y.type_as(outputs)
             loss = torch.mean(torch.clamp(1 - outputs.t() * y, min=0))  # hinge loss
             loss += 0.01 * torch.mean(abs(model.fc.weight))  # l1 penalty
             optimizer.zero_grad()
             loss.backward()
             optimizer.step()
             if i % 50 == 0:
                 print(i, loss.item())

         print(i, loss.item())
```

```
0 0.7936167120933533
50 0.7812798023223877
```

```
E:\anaconda3\lib\site-packages\torch\nn\functional.py:1351: UserWarning: nn.functional.sigmoid i
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
```

```
100 0.7687838673591614
150 0.7562322616577148
200 0.7437394261360168
250 0.7314152121543884
299 0.7195972800254822
```

1.15 Learn how to use linear regression, logistic regression, and SVM by scikit-learn.

1.15.1 linear regression

```
In [46]: from sklearn import linear_model

         reg = linear_model.LinearRegression()
         reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

Out[46]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [47]: reg.coef_

Out[47]: array([0.5, 0.5])
```

1.15.2 logistic regression

```
In [48]: from sklearn.datasets import load_iris
         from sklearn import linear_model
         X, y = load_iris(return_X_y=True)
         clf = linear_model.LogisticRegression(random_state=0).fit(X, y)

In [49]: clf.predict(X[:2, :])

Out[49]: array([0, 0])

In [50]: clf.predict_proba(X[:2, :])

Out[50]: array([[8.79681649e-01, 1.20307538e-01, 1.08131372e-05],
                [7.99706325e-01, 2.00263292e-01, 3.03825365e-05]])

In [51]: clf.score(X, y)

Out[51]: 0.96
```

1.15.3 SVM

```
In [52]: from sklearn import svm
         X = [[0, 0], [1, 1]]
         y = [0, 1]
         clf = svm.SVC()
         clf.fit(X, y)

Out[52]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

In [53]: clf.predict([[2., 2.]])

Out[53]: array([1])

In [54]: clf.support_vectors_

Out[54]: array([[0., 0.],
                [1., 1.]])
```

1.16 Download CIFAR-10 dataset and visualize some of its images.

```
In [55]: import torch
         import torch.nn as nn
         import torch.nn.functional as F
         import torch.optim as optim
         from torchvision import datasets, transforms
```

```
In [56]: CIFAR_data = datasets.CIFAR10("./CIFAR_data", train = True, download = True)
```

```
Files already downloaded and verified
```

```
In [57]: from torchvision.transforms import ToPILImage
         show = ToPILImage()
         (data, label) = CIFAR_data[66]
```

```
In [58]: data
```

```
Out[58]:
```

1.17 Write a dataset class for loading CIFAR-10. Make sure it could be transferred to Pytorch Dataloader.

```
In [59]: CIFAR_data = datasets.CIFAR10("./CIFAR_data", train = True, download = True)
```

```
Files already downloaded and verified
```

1.18 Read and learn how to use torchvision.transforms to transform images.

```
In [60]: CIFAR_data = datasets.CIFAR10("./CIFAR_data", train = True, download = True,
                                       transform = transforms.Compose([
                                           transforms.ToTensor()
                                       ]))
```

```
Files already downloaded and verified
```

1.19 Run one epoch for loading CIFAR-10 with Pytorch Dataloader and test the loading time of different batch_size(1, 4, 64, 1024), different num_workers (0,1,4,16), and whether use pin_memory or not.

```
In [61]: import numpy as np
         data = [d[0].data.cpu().numpy() for d in CIFAR_data]

In [62]: np.mean(data)

Out[62]: 0.4733649

In [63]: np.std(data)

Out[63]: 0.25156906

In [64]: import torch.utils.data as tud
         import time
         device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
         Batch_Size = [1, 4, 64, 1024]
         Num_Workers =  [0,1,4,16]
         Pin_Memory = [True, False]

         for batch_size in Batch_Size:
             for num_workers in Num_Workers:
                 for pin_memory in Pin_Memory:
                     start = time.clock()
                     train_dataloader = tud.DataLoader(datasets.CIFAR10("./CIFAR_data", train =
                                                                  download = False,
                                                                  transform = transforms.Com
                                                                      transforms.ToTensor(),
                                                                      transforms.Normalize((
                                                          batch_size = batch_size,
                                                          shuffle = True,
                                                          pin_memory = pin_memory,
                                                          num_workers = num_workers)
                     print('batch_size:{},num_workers:{},pin_memory:{},time:{}'.format(batch_siz
```

```
batch_size:1,num_workers:0,pin_memory:True,time:1.0054397
batch_size:1,num_workers:0,pin_memory:False,time:1.3142901999999999
batch_size:1,num_workers:1,pin_memory:True,time:1.3239940999999997
batch_size:1,num_workers:1,pin_memory:False,time:1.1534934999999997
batch_size:1,num_workers:4,pin_memory:True,time:1.0954878
batch_size:1,num_workers:4,pin_memory:False,time:1.1439372000000008
batch_size:1,num_workers:16,pin_memory:True,time:0.9447430000000008
batch_size:1,num_workers:16,pin_memory:False,time:0.9421460999999995
batch_size:4,num_workers:0,pin_memory:True,time:1.1984519999999996
batch_size:4,num_workers:0,pin_memory:False,time:1.026310800000001
batch_size:4,num_workers:1,pin_memory:True,time:0.9586273000000016
batch_size:4,num_workers:1,pin_memory:False,time:1.1462135
```

```
batch_size:4,num_workers:4,pin_memory:True,time:1.1403365
batch_size:4,num_workers:4,pin_memory:False,time:0.9754456999999999
batch_size:4,num_workers:16,pin_memory:True,time:1.0511990999999998
batch_size:4,num_workers:16,pin_memory:False,time:1.2255828999999991
batch_size:64,num_workers:0,pin_memory:True,time:1.0240116000000015
batch_size:64,num_workers:0,pin_memory:False,time:0.936417500000001
batch_size:64,num_workers:1,pin_memory:True,time:1.1592266999999978
batch_size:64,num_workers:1,pin_memory:False,time:0.9515725999999987
batch_size:64,num_workers:4,pin_memory:True,time:0.9400054000000004
batch_size:64,num_workers:4,pin_memory:False,time:0.9634847999999998
batch_size:64,num_workers:16,pin_memory:True,time:0.9426193000000005
batch_size:64,num_workers:16,pin_memory:False,time:1.1429928999999994
batch_size:1024,num_workers:0,pin_memory:True,time:0.9498835999999997
batch_size:1024,num_workers:0,pin_memory:False,time:1.2590731999999996
batch_size:1024,num_workers:1,pin_memory:True,time:0.940724300000003
batch_size:1024,num_workers:1,pin_memory:False,time:0.9466465
batch_size:1024,num_workers:4,pin_memory:True,time:0.9693368000000007
batch_size:1024,num_workers:4,pin_memory:False,time:0.9867589999999993
batch_size:1024,num_workers:16,pin_memory:True,time:0.9718236999999981
batch_size:1024,num_workers:16,pin_memory:False,time:0.9590048999999965
```

1.20 Calculate the mean and std of CIFAR-10' training set within each RGB channel.

```
In [65]: import numpy as np
         data = [d[0][0].data.cpu().numpy() for d in CIFAR_data]

In [66]: np.mean(data)

Out[66]: 0.49139968

In [67]: np.std(data)

Out[67]: 0.24703233

In [68]: import numpy as np
         data = [d[0][1].data.cpu().numpy() for d in CIFAR_data]

In [69]: np.mean(data)

Out[69]: 0.48215827

In [70]: np.std(data)

Out[70]: 0.24348505

In [71]: import numpy as np
         data = [d[0][2].data.cpu().numpy() for d in CIFAR_data]

In [72]: np.mean(data)

Out[72]: 0.44653124

In [73]: np.std(data)

Out[73]: 0.26158768
```

1.21 Image to character painting

In [74]:
```
from PIL import Image
char = list('1 ')
img = Image.open(r'1.png')
img = img.convert("RGB")

fp = open('character painting.txt', 'w')
width, height = img.size

for i in range(1, width):
    for j in range(1, height):
        R, G, B= img.getpixel((i, j))
        gray = 0.2126 * R + 0.7152 * G + 0.0722 * B
        unit = 256 / len(char)

        fp.write(char[int(gray//unit)])
    fp.write('\n')
fp.close()
```

In [75]:
```
img = Image.open('3.jpg')
width, height = img.size
```

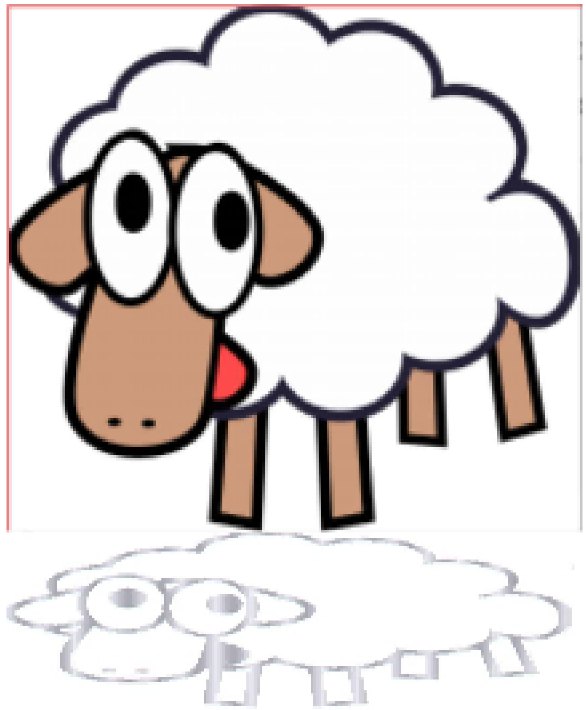
In [76]:
```
width
```

Out[76]: 1080

In [77]:
```
height
```

Out[77]: 1391

In [78]:
```
out = img.resize((216, 280))
display(out)
```



1.22 Numpy exercises

Consider a random 10x2 matrix representing cartesian coordinates, convert them to polar coordinates

```
In [79]: import numpy as np

In [80]: n = np.arange(0, 20, 1)# start at 0 count up by 2, stop before 30
         n = n.reshape(10, 2) # reshape array to be 3x5

In [81]: n

Out[81]: array([[ 0,  1],
                [ 2,  3],
                [ 4,  5],
                [ 6,  7],
                [ 8,  9],
                [10, 11],
                [12, 13],
                [14, 15],
                [16, 17],
                [18, 19]])

In [82]: import math
         def R(x,y):
```

```
    r = np.sqrt(x*x+y*y)
    return r

def theta(x,y):
    q = np.arctan(y/x) * 180 / math.pi
    return q

a = []
for x,y in zip(n[:,0],n[:,1]):
    a.append(R(x,y))
    a.append(theta(x,y))

a = np.array(a)
a = a.reshape(10, 2)
```

```
E:\anaconda3\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: divide by zero encounter
  import sys
```

In [83]:
```
a
```

Out[83]:
```
array([[ 1.        , 90.        ],
       [ 3.60555128, 56.30993247],
       [ 6.40312424, 51.34019175],
       [ 9.21954446, 49.39870535],
       [12.04159458, 48.36646066],
       [14.86606875, 47.72631099],
       [17.69180601, 47.29061004],
       [20.51828453, 46.97493401],
       [23.34523506, 46.73570459],
       [26.17250466, 46.5481577 ]])
```

Create a 2D array subclass such that Z[i, j] == Z[j, i].

In [84]:
```
X = np.random.rand(4**2).reshape(4, 4)
X = np.triu(X)
X += X.T - np.diag(X.diagonal())
```

In [85]:
```
X
```

Out[85]:
```
array([[0.7515699 , 0.98450985, 0.29782246, 0.68924979],
       [0.98450985, 0.08249835, 0.88963963, 0.0752501 ],
       [0.29782246, 0.88963963, 0.55514237, 0.47631188],
       [0.68924979, 0.0752501 , 0.47631188, 0.93007048]])
```

Consider 2 sets of points P0, P1 describing lines (2d) and a set of points P, how to compute distance from each point j (P[j]) to each line i (P0[i],P1[i])?

In [86]:
```
n = np.arange(0, 40, 2)# start at 0 count up by 2, stop before 30
n = n.reshape(10, 2) # reshape array to be 3x5
```

```
In [87]: n

Out[87]: array([[ 0,  2],
                [ 4,  6],
                [ 8, 10],
                [12, 14],
                [16, 18],
                [20, 22],
                [24, 26],
                [28, 30],
                [32, 34],
                [36, 38]])
```

```
In [88]: k =(n[0][1]-n[1][1]) /(n[0][0]-n[1][0]) * 1.0
         b = n[0][1] - k * n[0][0]
         k,b

Out[88]: (1.0, 2.0)
```

```
In [89]: p = np.arange(0,2,1)
         distance = round(abs(k * p[0] - p[1]+ b) / np.sqrt(k ** 2 + 1), 2)
         distance

Out[89]: 0.71
```

1.23 Bilinear Interpolation

```
In [90]: import math
         def BilinearInterpolation(A, P):
             row_low = math.floor(P[0])-1
             row_high = math.ceil(P[0])-1
             col_low = math.floor(P[1])-1
             col_high = math.ceil(P[1])-1
             low_value = (A[row_high][col_low] - A[row_low][col_low]) * (P[0] -  math.floor(P[0]
             high_value = (A[row_high][col_high] - A[row_low][col_high]) * (P[0] -  math.floor(P
             value = (high_value - low_value) * (P[1] -  math.floor(P[1])) + low_value
             return value
```

```
In [91]: A =((110, 120, 130),(210, 220, 230),(310, 320, 330))
```

```
In [92]: BilinearInterpolation(A, (1, 1))

Out[92]: 110
```

```
In [93]: BilinearInterpolation(A, (2.5, 2.5))

Out[93]: 275.0
```

1.24 Cartesian product

In [94]:
```
def combine(l1, l2):
          r = []
          a = len(l1)
          b = len(l2)
          if a == 0:
              return l2
          if b == 0:
              return l1

          for i in range(a):
              for j in range(b):
                  m = l1[i] + l2[j]
                  r.append(m)
          return r

def ls_reshape(ls):
    a = []
    for ele in ls:
        a.append([ele])
    return a
```

In [95]:
```
a = [1, 2, 3]
b = [4, 5]
c = [6, 7]

a = ls_reshape(a)
b = ls_reshape(b)
c = ls_reshape(c)

start = [a, b, c]
result = []

for ele in start:
    result = combine(result, ele)

result
```

Out[95]:
```
[[1, 4, 6],
 [1, 4, 7],
 [1, 5, 6],
 [1, 5, 7],
 [2, 4, 6],
 [2, 4, 7],
 [2, 5, 6],
 [2, 5, 7],
 [3, 4, 6],
 [3, 4, 7],
```

```
       [3, 5, 6],
       [3, 5, 7]]
```

1.25 Extracting a subpart of an array

```
In [96]: import numpy as np
         def extract(Z, shape, fill, position):
             P  = np.array(list(position)).astype(int)
             Z_temp = np.array(list(Z.shape)).astype(int)

             Result = np.ones(shape, dtype = Z.dtype) * fill
             R_temp = np.array(list(Result.shape)).astype(int)

             Result_start = np.zeros((len(shape),)).astype(int)
             Result_stop  = np.array(list(shape)).astype(int)

             Z_start = (P - R_temp // 2)
             Z_stop  = (P + R_temp // 2) + R_temp % 2

             Result_start = (Result_start - np.minimum(Z_start,0)).tolist()
             Z_start = (np.maximum(Z_start,0)).tolist()

             Result_stop = np.maximum(Result_start, (Result_stop - np.maximum(Z_stop-Z_temp,0)))
             Z_stop = (np.minimum(Z_stop,Z_temp)).tolist()

             r = [slice(start,stop) for start,stop in zip(Result_start, Result_stop)]
             z = [slice(start,stop) for start,stop in zip(Z_start, Z_stop)]

             Result[r] = Z[z]
             return Result
```

```
In [97]: Z =  np.random.randint(0, 10, (5, 5))
         shape = (4,4)
         fill  = 0
         position = (1,1)
         Z
```

```
Out[97]: array([[8, 2, 8, 6, 6],
                [2, 0, 0, 0, 7],
                [3, 3, 1, 8, 1],
                [2, 8, 9, 8, 1],
                [0, 5, 9, 2, 3]])
```

```
In [98]: extract(Z, shape, fill, position)
```

```
E:\anaconda3\lib\site-packages\ipykernel_launcher.py:24: FutureWarning: Using a non-tuple sequen
```

```
Out[98]: array([[0, 0, 0, 0],
                [0, 8, 2, 8],
```

```
              [0, 2, 0, 0],
              [0, 3, 3, 1]])
```

1.26 Matrix operations

Please implement following matrix (just 2D) operations without numpy

- add

```
In [99]: def add(a, b):
             m = len(a)
             n = len(a[0])
             result = []
             for i in range(m):
                 r = []
                 for j in range(n):
                     r.append(a[i][j] + b[i][j])
                 result.append(r)
             return result
```

- subtract

```
In [100]: def subtract(a, b):
              m = len(a)
              n = len(a[0])
              result = []
              for i in range(m):
                  r = []
                  for j in range(n):
                      r.append(a[i][j] - b[i][j])
                  result.append(r)
              return result
```

- scalar multiply

```
In [101]: def scalar_multiply(a, x):
              m = len(a)
              n = len(a[0])
              result = []
              for i in range(m):
                  r = []
                  for j in range(n):
                      r.append(a[i][j] * x)
                  result.append(r)
              return result
```

- multiply

```
In [102]: def multiply(a, b):
              m = len(a)
              p = len(a[0])
              n = len(b[0])
              result =  [ [0] * n for i in range(m)]
              for i in range(m):
                  for j in range(n):
                      s = 0
                      for k in range(p):
                          s = s + a[i][k] * b[k][j]
                      result[i][j] = s
              return result
```

- identity

```
In [103]: def identity(x):
              result = []
              for i in range(x):
                  r = []
                  for j in range(x):
                      if i == j:
                          r.append(1)
                      else:
                          r.append(0)
                  result.append(r)
              return result
```

- transpose

```
In [104]: def transpose(a):
              m = len(a)
              n = len(a[0])
              for i in range(m):
                  for j in range(i+1, n):
                      temp = a[i][j]
                      a[i][j] = a[j][i]
                      a[j][i] = temp
              return a
```

- inverse

```
In [105]: def inverse(a):
              n = len(a)
              new = identity(n)
              swap = []
              l1 = []
              for i in range(n):
                  swap.append(i)
                  l1.append([])
```

```
    for j in range(n):
        l1[i].append(0)

for i in range(n):
    max_row = a[i][i]
    row = i
    for j in range(i,n):
        if a[j][i] >= max_row:
            max_row = a[j][i]
            row = j
    swap[i] = row
    if row != i:
        for j in range(0,n):
            a[i][j],a[row][j] = a[row][j],a[i][j]
    for j in range(i+1,n):
        if a[j][i] != 0:
            l1[j][i] = a[j][i] / a[i][i]
            for k in range(0,n):
                a[j][k] = a[j][k] - (l1[j][i] * a[i][k])

long = len(a)-1
l2 = []
for i in range(n):
    l2.append([])
    for j in range(n):
        l2[i].append(0)

for i in range(n-1):
    for j in range(long-i):
        if a[long-i-j-1][long-i] != 0 and a[long-i][long-i] != 0:
            l2[long-i-j-1][long-i] = a[long-i-j-1][long-i] / a[long-i][long-i]
            for k in range(n):
                a[long-i-j-1][k] = a[long-i-j-1][k] - l2[long-i-j-1][long-i] * a[l

l3 = []
for i in range(n):
    l3.append(a[i][i])

for i in range(n):
    if swap[i] != i:
        new[i],new[swap[i]] = new[swap[i]],new[i]
    for j in range(i+1,n):
        for k in range(0,n):
            if l1[j][i] != 0:
                new[j][k] = new[j][k] - l1[j][i] * new[i][k]
for i in range(0,n-1):
    for j in range(0,n-i-1):
        if l2[n-1-i-j-1][n-1-i] != 0:
```

```
                for k in range(0,n):
                    new[n-1-i-j-1][k] = new[n-1-i-j-1][k] - l2[n-1-i-j-1][n-i-1] * new
        for i in range(0,n):
            for j in range(0,n):
                new[i][j] = new[i][j] / l3[i]
        return new
```

```
In [106]: matrix_a = [[12, 10], [3, 9]]
          matrix_b = [[3, 4], [7, 4]]
          matrix_c = [[11, 12, 13, 14], [21, 22, 23, 24], [31, 32, 33, 34], [41, 42, 43, 44]]
          matrix_d = [[3, 0, 2], [2, 0, -2], [0, 1, 1]]
```

```
In [107]: add(matrix_a, matrix_b)
```

```
Out[107]: [[15, 14], [10, 13]]
```

```
In [108]: subtract(matrix_a, matrix_b)
```

```
Out[108]: [[9, 6], [-4, 5]]
```

```
In [109]: scalar_multiply(matrix_b, 3)
```

```
Out[109]: [[9, 12], [21, 12]]
```

```
In [110]: multiply(matrix_a, matrix_b)
```

```
Out[110]: [[106, 88], [72, 48]]
```

```
In [111]: identity(3)
```

```
Out[111]: [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

```
In [112]: transpose(matrix_c)
```

```
Out[112]: [[11, 21, 31, 41], [12, 22, 32, 42], [13, 23, 33, 43], [14, 24, 34, 44]]
```

```
In [113]: inverse(matrix_d)
```

```
Out[113]: [[0.19999999999999998, 0.20000000000000004, 0.0],
           [-0.2, 0.30000000000000004, 1.0],
           [0.2, -0.30000000000000004, -0.0]]
```

1.27 Greatest common divisor

```
In [114]: def GCD(a,b):
              if a == 0:
                  return b
              if b == 0:
                  return a

              a = abs(a)
```

```
    b = abs(b)
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

In [115]: `GCD(3, 5)`

Out[115]: 1

In [116]: `GCD(6, 3)`

Out[116]: 3

In [117]: `GCD(-2, 6)`

Out[117]: 2

In [118]: `GCD(0, 3)`

Out[118]: 3

1.28 Find all consecutive positive number sequences whose sum is N

In [119]:
```
def sum_all(N):
    ls = [i for i in range(N)]
    end = int(N/2)+1
    result = []
    for i in range(1, end+1):
        for n in range(1, end+1):
            target = 1/2 * n * n + (i - 1/2) * n
            if abs(target - N) < 1e-6:
                result.append(ls[i:i+n])
    return result
```

In [120]: `sum_all(100)`

Out[120]: [[9, 10, 11, 12, 13, 14, 15, 16], [18, 19, 20, 21, 22]]

In [121]: `sum_all(1000)`

Out[121]:
```
[[28,
  29,
  30,
  31,
  32,
  33,
  34,
```

```
35,
  36,
  37,
  38,
  39,
  40,
  41,
  42,
  43,
  44,
  45,
  46,
  47,
  48,
  49,
  50,
  51,
  52],
 [55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70],
 [198, 199, 200, 201, 202]]
```

```
In [122]: ls
```

```
Out[122]: [0, 1, 2, 3, 4, 6, 8]
```

1.29 Password checking

```
In [123]: import re
          def step1(string):
              pattern = r'[a-z]'
              res = re.search(pattern, string)
              if res:
                  return 1
              else:
                  return 0

          def step2(string):
              pattern = r'[0-9]'
              res = re.search(pattern, string)
              if res:
                  return 1
              else:
                  return 0

          def step3(string):
              pattern = r'[A-Z]'
              res = re.search(pattern, string)
              if res:
```

```
            return 1
        else:
            return 0

    def step4(string):
        pattern = r'[$#@]'
        res = re.search(pattern, string)
        if res:
            return 1
        else:
            return 0

    def Password_checking(passwords):
        ls = passwords.split(',')
        r = []
        for ele in ls:
            if len(ele) <= 12 and len(ele)>=6:
                if step1(ele) and step2(ele) and step3(ele) and step4(ele):
                    r.append(ele)
        return ','.join(r)
```

```
In [124]: passwords = 'ABd1234@1,a F1#,2w3E*,2We3345'
```

```
In [125]: Password_checking(passwords)
```

```
Out[125]: 'ABd1234@1'
```