Lab 1 – The Basics of Python and Pytorch

陈亦雄 16307110231

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

代码和输出如下:

```
def sumNums(nums):
    # method 1
    # return sum(nums)
    # method 2
    sum_ = 0
    for num in nums:
        sum_ += num
    return sum_
```

```
In [5]: sumNums([1,3,5,7,9,2,3,4,5,6])
Out[5]: 45
```

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

e.g., Input:[1, 2, 3, 3, 3, 3, 4, 5]. Output: [1, 2, 3, 4, 5].

代码和输出如下:

```
def unique(nums):
    return list(set(nums))
```

```
In [7]: unique([1,2,3,3,3,3,4,5])
Out[7]: [1, 2, 3, 4, 5]
```

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 nursesrun.

代码和输出如下:

```
def isPalindrome(S):
    return S == S[::-1]
```

```
In [9]: isPalindrome('abcdedcba')
Out[9]: True

In [10]: isPalindrome('abcdecba')
Out[10]: False
```

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

e.g.,

Input: array [1.00000000+0.j, 0.70710678+0.70710678j]

Output: array [[1, 0], [0.70710678, 0.70710678]]

代码和输出如下:

```python
import numpy as np

arr = np.array([1.00000000 + 0.j, 0.70710678 + 0.70710678j, 0 + 1.0j])
def findParts(arr):
    real = np.real(arr)
    imag = np.imag(arr)
    return np.matrix([real, imag]).T
print(findParts(arr))
```

```
In [14]: runfile('D:/学习/python_codes/temp.py', wdir='D:/学习/python_codes')
[[1.         0.        ]
 [0.70710678 0.70710678]
 [0.         1.        ]]
```

5. Write a Python program to add two binary numbers.

e.g.,

Input : ('11', '1')

Output : 100

代码和输出如下:

```python
def addBinary(nums):
    return bin(int(nums[0], 2) + int(nums[1], 2))[2:]
```

```
In [16]: addBinary(('11', '1'))
Out[16]: '100'
```

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.

e.g.,

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Explanation: 342 + 465 = 807.

代码由于太长，详见附件“链表加法.py”；结果如下:

```
In [32]: runfile('D:/学习/python_codes/temp.py', wdir='D:/学习/python_codes')
Input: 1->2 , 9->9->9
Output: 0->2->0->1
```

```
In [33]: runfile('D:/学习/python_codes/temp.py', wdir='D:/学习/python_codes')
Input: 2->4->3 , 5->6->4
Output: 7->0->8
```

7. Implement bubble sort

代码由于太长，详见附件“冒泡排序.py”；代码运行结果如下:

```
In [37]: runfile('D:/学习/python_codes/Data-structure/排序算法/冒泡排序.py',
wdir='D:/学习/python_codes/Data-structure/排序算法')
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
BubbleSort consumes 0.753032922744751 seconds to sort 10 lists with length 1000
 average 0.0753032922744751 seconds
```

8. Implement merge sort

代码由于太长，详见附件“归并排序.py”；代码运行结果如下：

```
In [35]: runfile('D:/学习/python_codes/Data-structure/排序算法/归并排序.py',
wdir='D:/学习/python_codes/Data-structure/排序算法')
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
MergeSort consumes 6.798655986785889 seconds to sort 1000 lists with length 1000
 Average 0.006798655986785889 seconds
```

9. Implement quick sort

代码由于太长，详见附件“快速排序.py“；代码运行结果如下：

```
In [38]: runfile('D:/学习/python_codes/Data-structure/排序算法/快速排序.py',
wdir='D:/学习/python_codes/Data-structure/排序算法')
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
QuickSort consumes 2.3402719497680664 seconds to sort 1000 lists with length 1000
 average 0.0023402719497680664 seconds
```

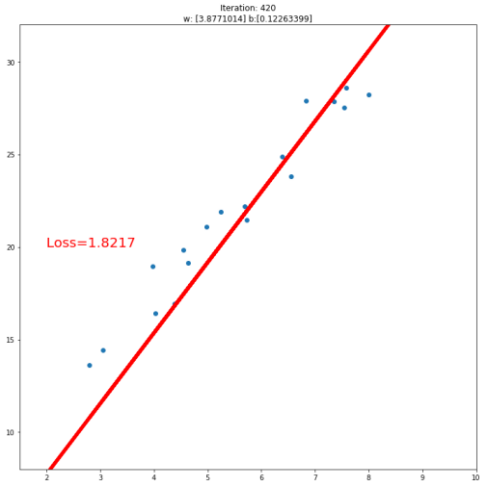
10. Implement shell sort

代码由于太长，详见附件“希尔排序.py“；代码运行结果如下：

```
In [51]: runfile('D:/学习/python_codes/Data-structure/排序算法/希尔排序.py',
wdir='D:/学习/python_codes/Data-structure/排序算法')
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
BubbleSort consumes 0.9315104484558105 seconds to sort 100 lists with length 1000
 average 0.009315104484558105 seconds
```

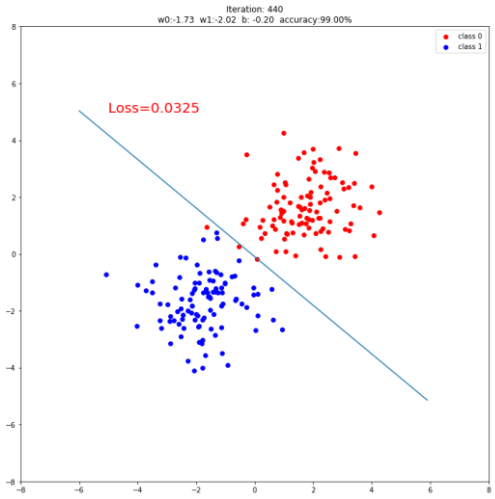
11. Implement linear regression model and use autograd to optimize it by Pytorch.

代码由于太长，详见附件“pytorch 线性回归.py“；代码运行结果如下：



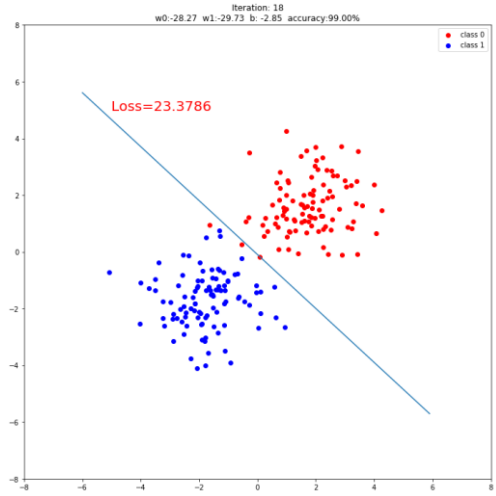
12. Implement logistic regression model and use autograd to optimize it by Pytorch.

代码由于太长，详见附件“pytorch 逻辑回归.py“；代码运行结果如下：



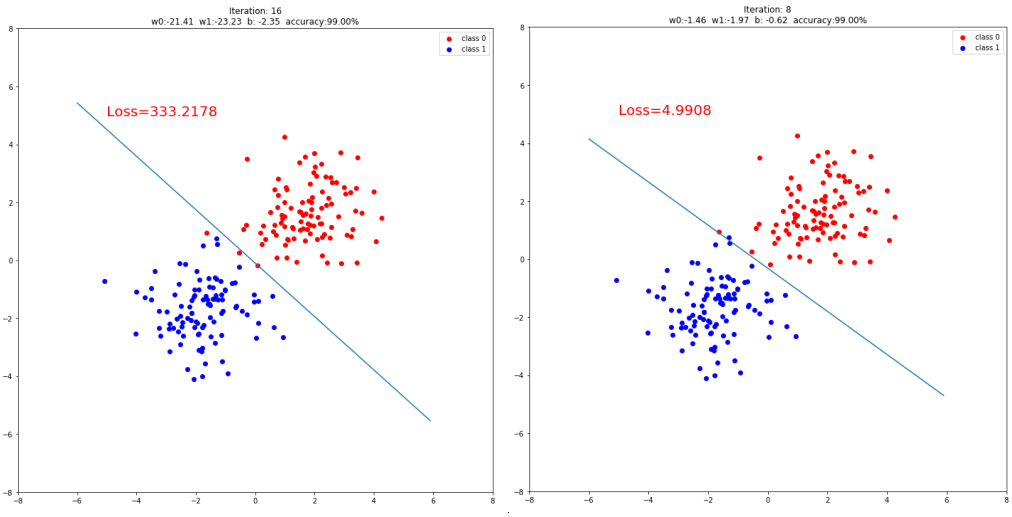
13. Implement linear SVM model for binary classification task and use autograd to optimize it by Pytorch. Hint: you may use the loss of $\Sigma \max[0, 1-y(wx+b)]$

代码由于太长，本题和 14 题详见附件“pytorch 支持向量机.py“；代码运行结果如下：



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

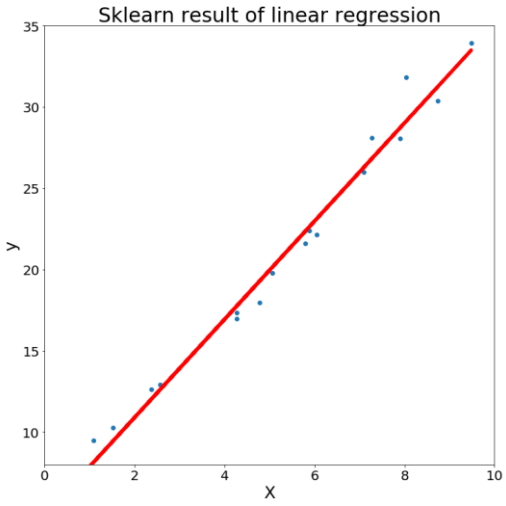
以下两图分别对应在损失函数上添加范数、在优化器上添加权重衰减。可见，两种方式的收敛效率有轻微区别，优化器更加有效，因为它添加的是所有权重的范数，而在损失函数上仅仅添加了 w 的范数。通过正则化我们可以使得模型收敛更快。详情请看“pytorch 支持向量机.py”。



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

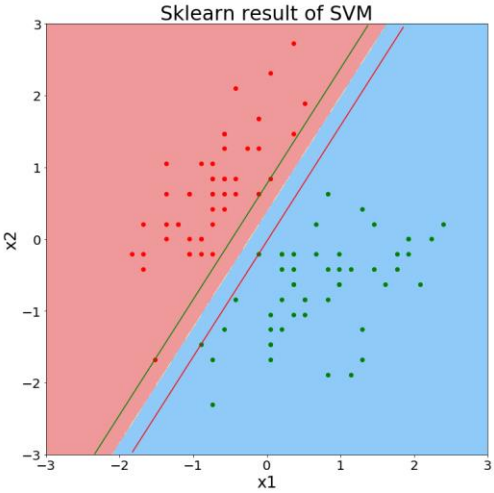
三个算法分别输出如下，代码详情请看“sklearn 线性回归.py”、“sklearn 逻辑回归.py”、“sklearn 支持向量机.py”。

截距为： 4.830793926528338
斜率为： 3.023247894945036



测试准确率： 0.995

w: [4.03240364 -2.50700346]
b: 0.9273331213417593



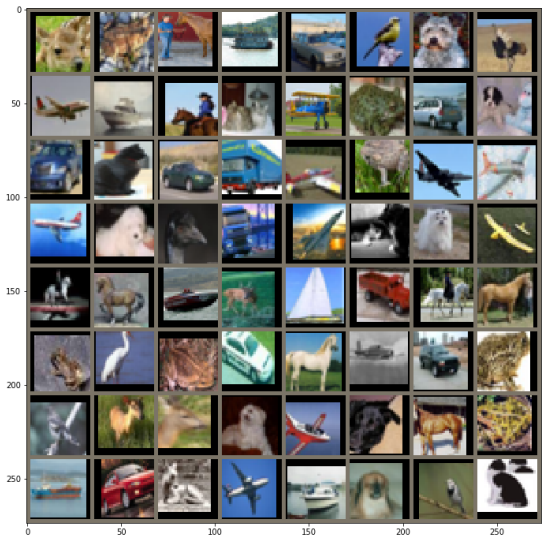
16. Download CIFAR-10 dataset and visualize some of its images.

可视化结果如下，代码详见“CIFAR-10 可视化.py”，运行代码并使用 tensorboard 打开 runs 文件夹下的可视化结果。

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

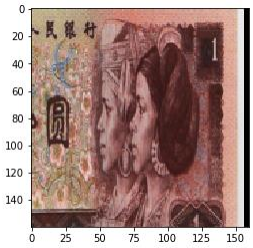
使用自己写的 dataset 类读取的数据可视化如下，详见“CIFAR-10_Dataset.py”。

```
一个batch数据的尺寸： torch.Size([64, 3, 32, 32])
一个batch标签的尺寸： torch.Size([64])
```

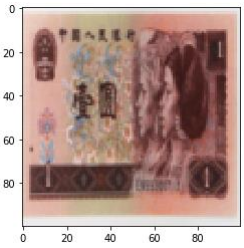


18. Read and learn how to use torchvision.transforms to transform images.

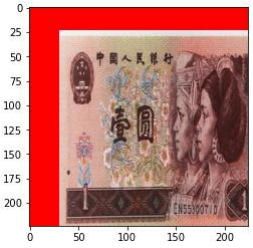
代码详见“transforms_methods_1.py”、“transforms_methods_2.py”，程序注释可以取消，来测试各种图像变换算法。几种典型的 transforms 方法的输出效果如下：



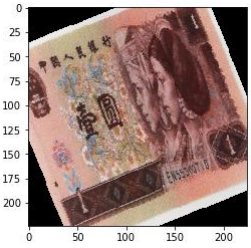
裁剪



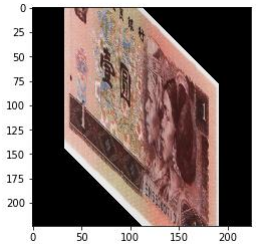
缩放



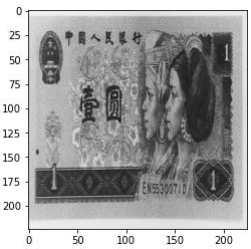
平移



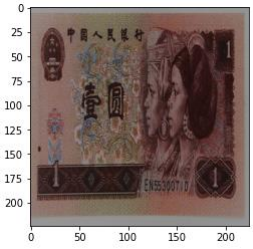
旋转



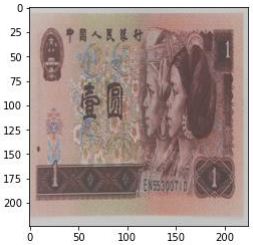
错切



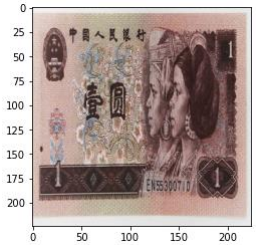
转换为灰度图



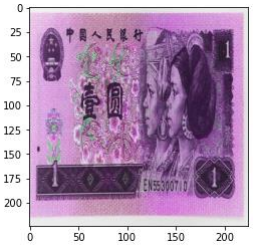
亮度变换



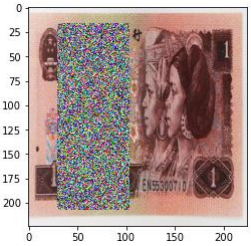
对比度变换



饱和度变换



色相变换



随机遮挡

水平翻转

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.

代码详见“dataloader_test.py”，输出如下图：

```
batch size: 1 num_workers: 0 pin_memory: True consume 3.41163969039917 seconds
batch size: 1 num_workers: 0 pin_memory: False consume 0.003072977066040039 seconds
batch size: 1 num_workers: 1 pin_memory: True consume 0.12059903144836426 seconds
batch size: 1 num_workers: 1 pin_memory: False consume 0.08325886726379395 seconds
batch size: 1 num_workers: 4 pin_memory: True consume 0.24973845481872559 seconds
batch size: 1 num_workers: 4 pin_memory: False consume 0.2053232192993164 seconds
batch size: 1 num_workers: 16 pin_memory: True consume 0.7383387088775635 seconds
batch size: 1 num_workers: 16 pin_memory: False consume 0.7119805812835693 seconds
batch size: 4 num_workers: 0 pin_memory: True consume 0.0045430660247802734 seconds
batch size: 4 num_workers: 0 pin_memory: False consume 0.0029144287109375 seconds
batch size: 4 num_workers: 1 pin_memory: True consume 0.11424779891967773 seconds
batch size: 4 num_workers: 1 pin_memory: False consume 0.08178925514221191 seconds
batch size: 4 num_workers: 4 pin_memory: True consume 0.22236990928649902 seconds
batch size: 4 num_workers: 4 pin_memory: False consume 0.22105145454406738 seconds
batch size: 4 num_workers: 16 pin_memory: True consume 0.7717611789703369 seconds
batch size: 4 num_workers: 16 pin_memory: False consume 0.687913179397583 seconds
batch size: 64 num_workers: 0 pin_memory: True consume 0.012583255767822266 seconds
batch size: 64 num_workers: 0 pin_memory: False consume 0.010115385055541992 seconds
batch size: 64 num_workers: 1 pin_memory: True consume 0.121368408203125 seconds
batch size: 64 num_workers: 1 pin_memory: False consume 0.10213255882263184 seconds
batch size: 64 num_workers: 4 pin_memory: True consume 0.28722596168518066 seconds
batch size: 64 num_workers: 4 pin_memory: False consume 0.2295677661895752 seconds
batch size: 64 num_workers: 16 pin_memory: True consume 0.7443833351135254 seconds
batch size: 64 num_workers: 16 pin_memory: False consume 0.7439095973968506 seconds
batch size: 1024 num_workers: 0 pin_memory: True consume 0.16042447090148926 seconds
batch size: 1024 num_workers: 0 pin_memory: False consume 0.12572813034057617 seconds
batch size: 1024 num_workers: 1 pin_memory: True consume 0.3594624996185303 seconds
batch size: 1024 num_workers: 1 pin_memory: False consume 0.3509366512298584 seconds
17050096it [00:29, 11468916.00it/s]
batch size: 1024 num_workers: 4 pin_memory: True consume 0.6385698318481445 seconds
batch size: 1024 num_workers: 4 pin_memory: False consume 0.5087635517120361 seconds
batch size: 1024 num_workers: 16 pin_memory: True consume 1.3442130088806152 seconds
batch size: 1024 num_workers: 16 pin_memory: False consume 1.2722358703613281 seconds
```

我们能够发现，随着 batch size 的增大，数据加载越来越慢（第一次加载除外，因为要进行第一次 dataloader 的一系列初始化）。num_worker 参数越大，加载耗时越长，pin_memory 参数设置为 False 时比 True 时加载更快。

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

代码详见“三通道均值标准差.py”，输出如下：

```
In [16]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/三通道均值标准差.py',
wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
训练集维度： (50000, 3072)
红色通道均值： 125.306918046875
绿色通道均值： 122.950394140625
蓝色通道均值： 113.86538318359375
红色通道标准差： 62.993219278136884
绿色通道标准差： 62.08870764001421
蓝色通道标准差： 66.70489964063091

测试集维度： (10000, 3072)
红色通道均值： 126.02464140625
绿色通道均值： 123.70850419921875
蓝色通道均值： 114.85431865234375
红色通道标准差： 62.89639134921991
绿色通道标准差： 61.93752718231365
蓝色通道标准差： 66.70605639561605
```

21. Image to character painting

(a) Target Converting the RGB color image to character painting with Python code

• Character painting is a combination of a series of characters. We can think of characters as relatively large pixels. A character can represent a color. The more types of characters, the more colors can be represented, and the picture will be more hierarchical sense.

(b) Requirements

• Python 3.5

• pillow 5.1.0

(c) Method

i. Use PIL (pillow) to get the input picture

ii. Use the following formula to map RGB values to gray values (note that this formula is not a real algorithm, but a simplified sRGB IEC61966-2.1 formula)

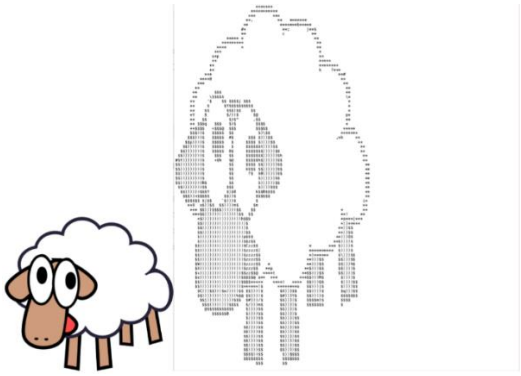
• gray = 0.2126 * r + 0.7152 * g + 0.0722 * b

iii. Create a character list (length and content are customized)

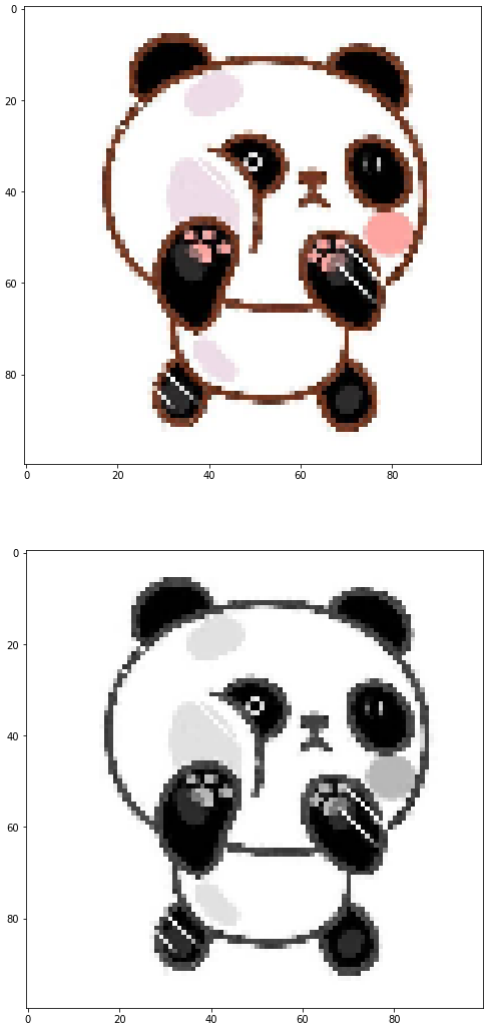
iv. Map the gray value to characters and save the result with a string (note the corresponding picture size, add line breaks)

v. Export character painting to a .txt file

(d) Result



文本输出见以下左图，具体可以看 character_painting.txt；代码输出为右侧两图，分别为降低分辨率后的 RGB 图像以及灰度图像，详情见“Image_to_character_painting.py”。



22. Numpy exercises

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

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

• 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])?

代码详见“numpy_exercise”.py，程序输出如下：

```
cartesian:                    polar:
 [[-1.76884571  0.07555227]  [[1.77045849 3.09890585]
 [-1.1306297  -0.65143017]   [1.30486964 3.66430262]
 [-0.89311563 -1.27410098]   [1.55595271 4.10100754]
 [-0.06115443  0.06451384]   [0.08889264 2.32946847]
 [ 0.41011295 -0.57288249]   [0.70454736 5.33369085]
 [-0.80133362  1.31203519]   [1.53739127 2.11908728]
 [ 1.27469887 -1.2143576 ]   [1.76054583 5.52202502]
 [ 0.31371941 -1.44482142]   [1.47848868 4.92620366]
 [-0.3689613  -0.76922658]   [0.85313655 4.26515163]
 [ 0.3926161   0.05729383]] [0.39677448 0.14490557]]
```

```
Symmetric matrix X:
 [[0.503 0.668 0.034 0.456 0.156]
 [0.668 0.17  0.896 0.373 0.38 ]
 [0.034 0.896 0.583 0.668 0.178]
 [0.456 0.373 0.668 0.764 0.92 ]
 [0.156 0.38  0.178 0.92  0.192]]
```

```
由[0 2]、[2 0]两点组成的直线到点[0 0]距离为1.414
由[0 2]、[2 0]两点组成的直线到点[1 0]距离为0.707
由[0 2]、[2 0]两点组成的直线到点[0 1]距离为0.707
由[0 2]、[1 0]两点组成的直线到点[0 0]距离为0.894
由[0 2]、[1 0]两点组成的直线到点[1 0]距离为0.0
由[0 2]、[1 0]两点组成的直线到点[0 1]距离为0.447
由[0 2]、[-1  0]两点组成的直线到点[0 0]距离为0.894
由[0 2]、[-1  0]两点组成的直线到点[1 0]距离为1.789
由[0 2]、[-1  0]两点组成的直线到点[0 1]距离为0.447
```

23. Bilinear Interpolation

Please implement the bilinear interpolation algorithm using python. Check this for an introduction to bilinear interpolation.

Test samples:

A = ((110, 120, 130),
(210, 220, 230),
(310, 320, 330))

BilinearInterpolation(A, (1, 1)) == 110

BilinearInterpolation(A, (2.5, 2.5)) == 275

代码详情请看“双线性插值.py”，程序输出如下：

```
In [58]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/双线性插值.py', wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
坐标(1, 1)处插值像素为：110
坐标(2.5, 2.5)处插值像素为：275.0
坐标(1.7, 2.4)处插值像素为：194.0
```

24. Cartesian product

Given an arbitrary number of vectors, build the cartesian product (every combinations of every item).

e.g. [1, 2, 3], [4, 5], [6, 7] ==> [[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]]

代码详情请看“笛卡尔积.py”，程序输出如下：

```
In [72]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/笛卡尔积.py', wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
[[1, 2, 3], [4, 5], [6, 7]] 的笛卡尔积为：
[[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]]
```

25. Extracting a subpart of an array

Consider an arbitrary array, write a function that extract a subpart with a fixed shape and centered on a given element (pad with a fill value when necessary)

e.g.

In:

>> Z = np.random.randint(0, 10, (5, 5))

>> shape = (4, 4)

>> fill = 0

>> position = (1,1)

>> Z [[3 6 8 5 9]
[4 9 0 0 9]
[6 1 4 0 8]
[9 1 2 0 9]
[4 1 7 5 0]]

Out: [[0 0 0 0]
[0 3 6 8]
[0 4 9 0]
[0 6 1 4]]

代码详情请看“提取子阵.py”，程序输出如下：

```
In [100]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/提取子阵.py', wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
原矩阵：
[[0 8 3 6 3]
 [3 7 8 0 0]
 [8 9 3 7 2]
 [3 6 5 0 4]
 [8 6 4 1 1]]

形状(4, 3)，中心(1, 1)的子矩阵如下：
[[0 0 0]
 [0 0 8]
 [0 3 7]
 [0 8 9]]
```

26. Matrix operations

Please implement following matrix (just 2D) operations without numpy: • add • subtract • scalar multiply • multiply • identity • transpose • inverse

Test samples:

In:

>> 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]]

Out:

add(matrix_a, matrix_b) == [[15, 14], [10, 13]]

subtract(matrix_a, matrix_b) == [[9, 6], [-4, 5]]

scalar_multiply(matrix_b, 3) == [[9, 12], [21, 12]]

multiply(matrix_a, matrix_b) == [[106, 88], [72, 48]]

identity(3) == [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

transpose(matrix_c) == [[11, 21, 31, 41], [12, 22, 32, 42], [13, 23, 33, 43], [14, 24, 34, 44]]

inverse(matrix_d) == [[0.2, 0.2, 0.0], [-0.2, 0.3, 1.0], [0.2, -0.3, 0.0]]

代码详情请看“矩阵操作.py”，程序输出如下：

```
In [5]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/矩阵操作.py',
wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
A: [[12, 10], [3, 9]]
B: [[3, 4], [7, 4]]
C: [[11, 12, 13, 14], [21, 22, 23, 24], [31, 32, 33, 34], [41, 42, 43, 44]]
D: [[3, 0, 2], [2, 0, -2], [0, 1, 1]]

A + B:
[[15, 14], [10, 13]]
A - B:
[[9, 6], [-4, 5]]
B x 3:
[[9, 12], [21, 12]]
A x B:
[[106, 88], [72, 48]]
对角矩阵：
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
C的转置：
[[11, 21, 31, 41], [12, 22, 32, 42], [13, 23, 33, 43], [14, 24, 34, 44]]
D的逆矩阵：
[[0.19999999999999998, 0.20000000000000004, 0.0], [-0.2, 0.30000000000000004,
1.0], [0.2, -0.30000000000000004, -0.0]]
```

27. Greatest common divisor Find the greatest common divisor(gcd) of two integers.

Test samples:

- GCD(3, 5) = 1
- GCD(6, 3) = 3
- GCD(-2, 6) = 2
- GCD(0, 3) = 3

代码详情请看“最大公因数.py”，程序输出如下：

```
In [119]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/最大公因数.py',
wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
GCD(3, 5) = 1
GCD(6, 3) = 3
GCD(-2, 6) = 2
GCD(0, 3) = 3
```

28. Find all consecutive positive number sequences whose sum is N

e.g. 18+19...+22 = 9+10+...+16 = 100

Find all consecutive positive number sequences whose sum is 1000, and report your results.

代码详情请看“连续正整数和.py”，程序输出如下：

```
In [125]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/连续正整数和.py',
wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
100可拆成以下数组和： [[18, 19, 20, 21, 22], [9, 10, 11, 12, 13, 14, 15, 16]]
1000可拆成以下数组和： [[198, 199, 200, 201, 202], [55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70], [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]]
```

29. Password checking

A website requires the users to input username and password to register. Write a program to check the validity of password input by users. Following are the criteria for checking the password:

- At least 1 letter between [a-z]
- At least 1 number between [0-9]
- At least 1 letter between [A-Z]
- At least 1 character from [$#@]
- Minimum length of transaction password: 6
- Maximum length of transaction password: 12

Your program should accept a sequence of comma separated passwords and will check them according to the above criteria. Passwords that match the criteria are to be printed, each separated by a comma.

e.g. If the following passwords are given as input to the program: ABd1234@1, a F1#, 2w3E*, 2We3345. Then, the output of the program should be: ABd1234@1.

代码详情请看“有效密码.py”，程序输出如下：

```
In [134]: runfile('D:/学习/课程/大数据/深度学习和神经网络/作业/lab1/有效密码.py',
wdir='D:/学习/课程/大数据/深度学习和神经网络/作业/lab1')
All passwords:  ABd1234@1,aF1#,2w3E*,2We3345,UvwXY123@#
Valid passwords:  ABd1234@1,UvwXY123@#
```