人工智能基础与进阶

Python拓展实践

上海交通大学

目录 content

实验环境

Jupyter Notebook

- 地址：http://121.5.222.84:8888/
- 界面

第一节 二维数组

二维数组

二维数组是什么？

list中的二维数组是什么？

numpy中的二维数组是什么？

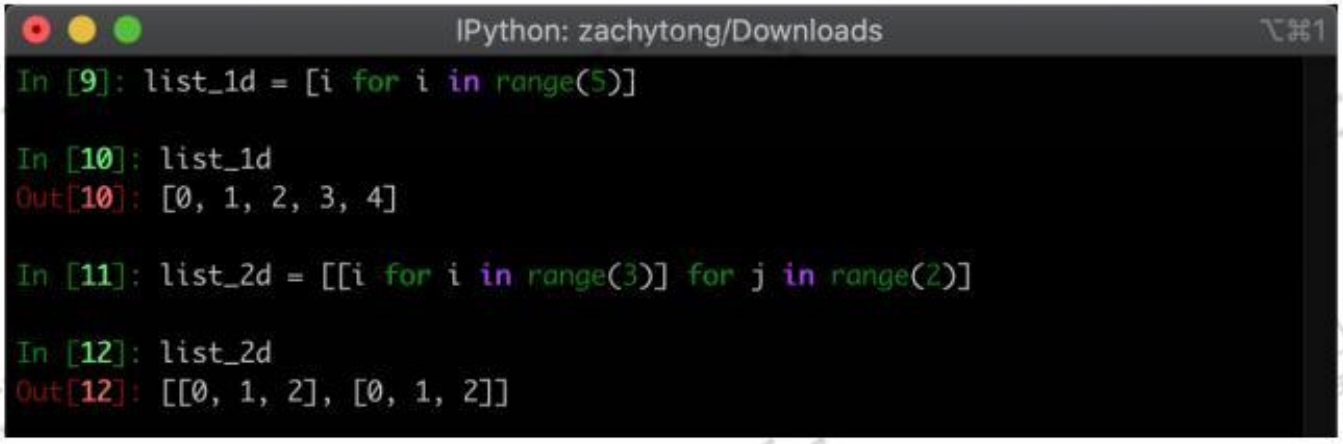
为什么在有了list二维数组之后，我们还需要numpy二维数组？

二维数组

list一维数组：list_1d = [0, 1, 2, 3, 4]（数组内的元素是整数int）

list二维数组：list_2d = [[0, 1, 2], （亦可写作[[0, 1, 2], [0, 1, 2]]）
[0, 1, 2]]

(外层数组对应的元素是数组list，内部数组对应的元素是整数int）



二维数组

二维数组索引与切片(index and slicing)：

IPython: zachyton

```
In [25]: len(list_2d)
Out[25]: 2

In [26]: list_2d[0]
Out[26]: [0, 1, 2]

In [27]: list_2d[0][2]
Out[27]: 2

In [28]: list_2d[0][:1]
Out[28]: [0]

In [29]: list_2d[0][:2]
Out[29]: [0, 1]

In [30]: list_2d[:][0:1]
Out[30]: [[0, 1, 2]]
```

```
In [103]: list_1d[::2]
Out[103]: [1, 's']

In [104]: list_1d[1::2]
Out[104]: [0, [1]]

In [105]: list_1d[-1:]
Out[105]: [[1]]

In [106]: list_1d[-1]
Out[106]: [1]

In [107]: list_1d[:-1]
Out[107]: [1, 0, 's']

In [108]: list_1d[-3:-1]
Out[108]: [0, 's']

In [109]: list_1d[0:-1]
Out[109]: [1, 0, 's']
```

二维数组

NumPy*二维数组（ndarray）
导入库：import numpy as np
array_2d：ndarray对象

```
In [38]: array_2d = np.array([[0, 1, 2], [0, 1, 2]])

In [39]: array_2d
Out[39]:
array([[0, 1, 2],
       [0, 1, 2]])

In [40]: print(array_2d)
[[0 1 2]
 [0 1 2]]
```

```
In [47]: type(array_2d)
Out[47]: numpy.ndarray

In [48]: type(list_2d)
Out[48]: list
```

*: NumPy 全称为 Numerical Python，是 Python 的一个以矩阵为主的用于科学计算的基础软件包。

二维数组

NumPy二维数组索引与切片：

IPython: zachytong/Downloads ⌥⌘1

```
In [41]: len(array_2d)
Out[41]: 2

In [42]: array_2d.shape
Out[42]: (2, 3)

In [43]: array_2d[0]
Out[43]: array([0, 1, 2])

In [44]: array_2d[0,1]
Out[44]: 1

In [45]: array_2d[:1,0]
Out[45]: array([0])

In [46]: array_2d[:1,:2]
Out[46]: array([[0, 1]])
```

二维数组

list数组和NumPy数组索引差异：

```
In [134]: array_2d[0,1]
Out[134]: 1

In [135]: list_2d
Out[135]: [[0, 1, 2], [0, 1, 2]]

In [136]: list_2d[0,1]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-136-caeb5c027a08> in <module>
----> 1 list_2d[0,1]

TypeError: list indices must be integers or slices, not tuple

In [137]: array_2d
Out[137]:
array([[0, 1, 2],
       [0, 1, 2]])

In [138]: array_2d[0,1]
Out[138]: 1
```

二维数组

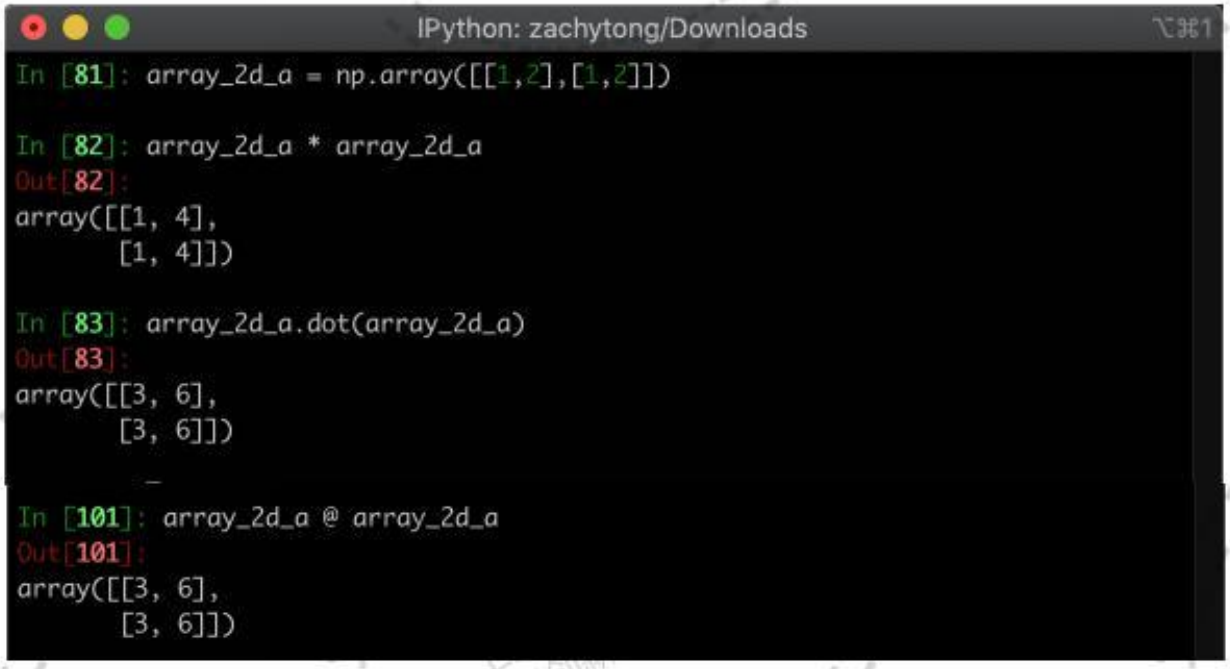
NumPy二维数组布尔索引：

```
In [137]: array_2d
Out[137]:
array([[0, 1, 2],
       [0, 1, 2]])

In [138]: array_2d[0,1]
Out[138]: 1

In [139]: array_2d
Out[139]:
array([[0, 1, 2],
       [0, 1, 2]])

In [140]: mask = array_2d > 1

In [141]: mask
Out[141]:
array([[False, False,  True],
       [False, False,  True]])

In [142]: array_2d[mask]
Out[142]: array([2, 2])
```

```
In [144]: list_2d > 1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-144-b00cebfdf616> in <module>
----> 1 list_2d > 1

TypeError: '>' not supported between instances of 'list' and 'int'

In [145]: list_2d[mask]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-145-d82bb59dd36c> in <module>
----> 1 list_2d[mask]

TypeError: only integer scalar arrays can be converted to a scalar index

In [146]: list_2d[mask.tolist()]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-146-15d3dbfbea63> in <module>
----> 1 list_2d[mask.tolist()]

TypeError: list indices must be integers or slices, not list
```

二维数组

NumPy二维数组运算：elementwise product, matrix product



二维数组

list数组和NumPy数组互相转化：

```
In [91]: array_2d_list = list(array_2d)

In [92]: array_2d_list
Out[92]: [array([0, 1, 2]), array([0, 1, 2])]

In [93]: array_2d_list = array_2d.tolist()

In [94]: array_2d_list
Out[94]: [[0, 1, 2], [0, 1, 2]]
```

IPython: zachytong/Downloads ⌥⌘1

```
In [95]: list_2d_array = np.array(list_2d)

In [96]: list_2d_array
Out[96]:
array([[0, 1, 2],
       [0, 1, 2]])

In [97]: type(list_2d_array)
Out[97]: numpy.ndarray
```

二维数组

NumPy相关函数:

1. numpy.range(), numpy.random.randint()

2. numpy.reshape(), numpy.flatten()

3. numpy.argmin(), numpy.min(), numpy.where()

4. numpy.exp(), numpy.sqrt(), numpy.sum()

二维数组

二维数组是什么？

list中的二维数组如何生成？

NumPy中的二维数组如何生成？

为什么在有了list二维数组之后，我们还需要numpy二维数组？

第二节 迭代器与生成器

迭代器与生成器

迭代器（iterator）

迭代是Python最强大的功能之一，是访问集合元素的一种方式。

迭代器是一个可以记住遍历的位置的对象。迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能向前不会后退。迭代器有两个基本的方法：iter() 和 next()。

字符串，列表或元组对象是可迭代的(iterable)，都可用于创建迭代器。

迭代器与生成器

生成器（generator）

在 Python 中，使用了 yield 的函数被称为生成器（generator）。

跟普通函数不同的是，生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个迭代器。

在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。

调用一个生成器函数，返回的是一个迭代器对象。

迭代器与生成器

斐波那契数列求和

方法1:

```
def fib(n):# 普通版本
    a,b = 1,1
    for i in range(n-1):
        a,b = b,a+b
    return a
```

方法2:

```
def fib(n):# 递归版本
    if n==1 or n==2:
        return 1
    return fib(n-1)+fib(n-2)
```

方法3:

```
def fib(n): # 生成器版本
    a, b, counter = 0, 1, 0
    while True:
        if (counter > n):
            return
        yield a
        a, b = b, a + b
        counter += 1
```

迭代器与生成器

为什么要实用迭代器和生成器？

1. 减小运行时内存占用

2. 实时/无限数据流

第三节 函数参数

函数参数

不可变类型与可变类型参数传递

```
In [164]: def change(a):
     ...:     print('id: ',id(a))
     ...:     a = 10
     ...:     print('id: ', id(a))
     ...:

In [165]: a = 1

In [166]: id(a)
Out[166]: 4395174240

In [167]: change(a)
id:  4395174240
id:  4395174528

In [168]: id(a)
Out[168]: 4395174240
```

```
In [169]: def change2(b):
     ...:     print('id: ', id(b))
     ...:     b[0] = 11
     ...:     print('id: ', id(b))
     ...:

In [170]: b = [0, 1, 2, 3]

In [171]: id(b)
Out[171]: 140273879018368

In [172]: id(b[0])
Out[172]: 4395174208

In [173]: change2(b)
id:  140273879018368
id:  140273879018368

In [174]: id(b[0])
Out[174]: 4395174560
```

函数参数

必须参数、关键字参数、默认参数（元组形式）、不定长参数（字典形式）

```
def func(arg1, arg2 = default, *argv, **kwargs):
    para1 = argv[-1]
    para2 = kwargs['demo']

    ans = process(arg1, arg2, para1, para2)

    return ans
```

第四节 规范与Tips

Python Enhancement Proposal#8 命名规范

Naming: PEP 8 suggests unique styles of naming for different parts in the language. This makes it easy to distinguish which type corresponds to each name when reading code.

- Functions, variables, and attributes should be in `lowercase_underscore` format.
- Protected instance attributes should be in `_leading_underscore` format.
- Private instance attributes should be in `__double_leading_underscore` format.
- Classes and exceptions should be in `CapitalizedWord` format.
- Module-level constants should be in `ALL_CAPS` format.
- Instance methods in classes should use `self` as the name of the first parameter (which refers to the object).
- Class methods should use `cls` as the name of the first parameter (which refers to the class).

规范与Tips

⚠ 注意索引与切片

a = ['a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , 'h']

a[:]
a[:5]
a[:-1]
a[4:]
a[-3:]
a[2:5]
a[2:-1]
a[-3:-1]

规范与标准

⚠ 注意索引与切片

a = ['a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , 'h']

```
a[:]      # ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
a[:5]     # ['a', 'b', 'c', 'd', 'e']
a[:-1]    # ['a', 'b', 'c', 'd', 'e', 'f', 'g']
a[4:]     #                      ['e', 'f', 'g', 'h']
a[-3:]    #                           ['f', 'g', 'h']
a[2:5]    #           ['c', 'd', 'e']
a[2:-1]   #           ['c', 'd', 'e', 'f', 'g']
a[-3:-1]  #                           ['f', 'g']
```

规范与Tips

使用生成器！

- 当输入很大时，列表list操作可能会由于占用内存过大导致问题
- 生成器通过每次迭代只输出一个避免内存占用问题
- 生成器可以通过for语句互相组合
- 生成器表达式组合时执行较快

规范与Tips

使用zip组合多个可迭代对象

找到最长的名字

```
names = [ ‘Cecilia’ , ‘Lise’ , ‘Marie’ ]
letters = [len(n) for n in names]
```

方法1:

```
longest_name = None
max_letters = 0
for i in range(len(names)):
    count = letters[i]
    if count > max_letters:
        longest_name = names[i]
        max_letters = count
```

方法2:

```
for i, name in enumerate(names):
    count = letters[i]
    if count > max_letters:
        longest_name = name
        max_letters = count
```

方法3:

```
for name, count in zip(names, letters):
    if count > max_letters:
        longest_name = name
        max_letters = count
```

第五节 动手试试⌨

动手试试⌨

1. 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出 和为目标值 target 的那 两个 整数，并返回它们的数组下标。假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

示例 1：

```
输入：nums = [2,7,11,15], target = 9
输出：[0,1]
解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。
```

示例 2：

```
输入：nums = [3,2,4], target = 6
输出：[1,2]
```

示例 3：

```
输入：nums = [3,3], target = 6
输出：[0,1]
```

动手试试

2. 给你一个整数 x ，如果 x 是一个回文整数，返回 true ；否则，返回 false 。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如，121 是回文，而 123 不是。

示例 1：

```
输入：x = 121
输出：true
```

示例 2：

```
输入：x = -121
输出：false
解释：从左向右读，为 -121 。 从右向左读，为 121- 。因此它不是一个回文数。
```

示例 3：

```
输入：x = 10
输出：false
解释：从右向左读，为 01 。因此它不是一个回文数。
```

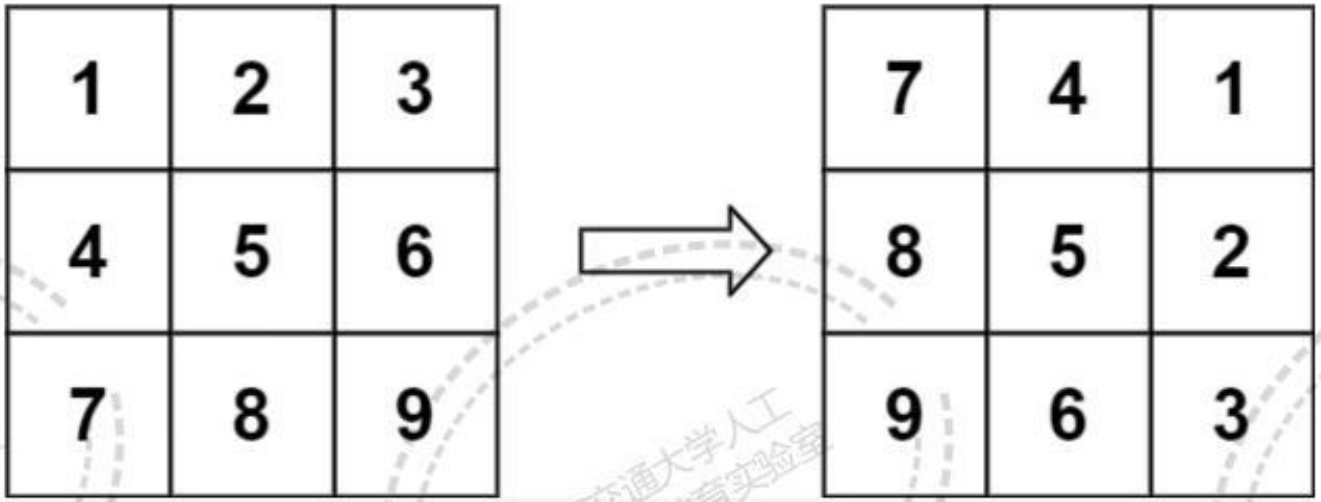
示例 4：

```
输入：x = -101
输出：false
```

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要使用另一个矩阵来旋转图像。

示例 1：




```
输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]
输出：[[7,4,1],[8,5,2],[9,6,3]]
```

分组事宜

共7组。

请各位同学自行和周围同学协商组队，之后每个小组尽量坐在一起，方便进行学习、讨论和实验等任务。

下节课

参考以下教程安装GraphViz库：


谢谢聆听
THANKS FOR YOUR ATTENTION

动手试试⌨

1. 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那 两个 整数，并返回它们的数组下标。假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

示例 1：

```
输入：nums = [2,7,11,15], target = 9
输出：[0,1]
解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。
```

示例 2：

```
输入：nums = [3,2,4], target = 6
输出：[1,2]
```

示例 3：

```
输入：nums = [3,3], target = 6
输出：[0,1]
```

动手试试

1. 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那 两个 整数，并返回它们的数组下标。假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

已知条件

① 输入：int list nums，int target

② 输出：int list [index_1, index_2]

③ 每种输入只对应一个答案: nums中两元素组合中，只有一对索引 index_i, index_j 满足 nums[index+i] +nums[index_j] == target

④ 数组同一个元素在答案里不能重复出现: index_i != index_j

动手试试⌨

1. 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那 两个 整数，并返回它们的数组下标。假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

思路一：双层循环

```
for i in range(len(nums)-1):
    for j in range(i, len(nums)):
        if nums[i] + nums[j] == target:
            return [i, j]
```

动手试试

1. 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那 两个 整数，并返回它们的数组下标。假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

思路二：给nums中所有两元素组合的和排序，并找到target在其中的对应位置

```
nums_copy = nums.copy()
nums.sort()
temp_i, temp_j = 0, len(nums)-1
while True:
    if nums[temp_i] + nums[temp_j] < target:
        temp_i += 1
    elif nums[temp_i] + nums[temp_j] > target:
        temp_j -= 1
    else:
        break
i = nums_copy.index(nums[temp_i])
if nums[temp_j] != nums[temp_i]:
    j = nums_copy.index(nums[temp_j])
else:
    j = nums_copy[i+1:].index(nums[temp_j]) + i + 1
return [i, j]
```

动手试试

1. 给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那 两个 整数，并返回它们的数组下标。假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

思路三：运用字典数据结构

```
maps = {}
        for i, num in enumerate(nums):
        maps[num] = i
for i, num in enumerate(nums):
        j = maps.get(target - num)
        if j is not None and j != i:
                return [i, j]
```

动手试试

2. 给你一个整数 x ，如果 x 是一个回文整数，返回 true ；否则，返回 false 。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如，121 是回文，而 123 不是。

示例 1：

```
输入：x = 121
输出：true
```

示例 2：

```
输入：x = -121
输出：false
解释：从左向右读，为 -121 。 从右向左读，为 121- 。因此它不是一个回文数。
```

示例 3：

```
输入：x = 10
输出：false
解释：从右向左读，为 01 。因此它不是一个回文数。
```

示例 4：

```
输入：x = -101
输出：false
```

动手试试⌨

2. 给你一个整数 x ，如果 x 是一个回文整数，返回 true ；否则，返回 false 。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如，121 是回文，而 123 不是。

已知条件
- ① 输入：int x
- ② 输出：Bool true/false
- ③ x需要满足数字前、后半部分对称相等
- ④ 负号参与倒序操作，负数肯定不是回文数

动手试试

2. 给你一个整数 x ，如果 x 是一个回文整数，返回 true ；否则，返回 false 。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如，121 是回文，而 123 不是。

思路一：计算出x对应的倒叙数x_inverse，再判断两者是否相等

```
if x < 0:
    return False
else:
    quo = x
    res, x_inverse = 0, 0
    while quo != 0:
        res = quo % 10
        x_inverse = x_inverse*10 + res
        quo = quo / 10
    return x == x_inverse
```

动手试试

2. 给你一个整数 x ，如果 x 是一个回文整数，返回 true ；否则，返回 false 。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如，121 是回文，而 123 不是。

思路二：转换成str类型，遍历前半部分判断是否与后半部分对应相等

```
x_str = str(x)
half_lenth = len(x_str)//2 if len(x_str)%2 != 0 else len(x_str)//2 + 1
for i in range(half_lenth):
    if x_str[i] != x_str[len(x_str)-1-i]:
            return False
return True
```

动手试试⌨

2. 给你一个整数 x ，如果 x 是一个回文整数，返回 true ；否则，返回 false 。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如，121 是回文，而 123 不是。

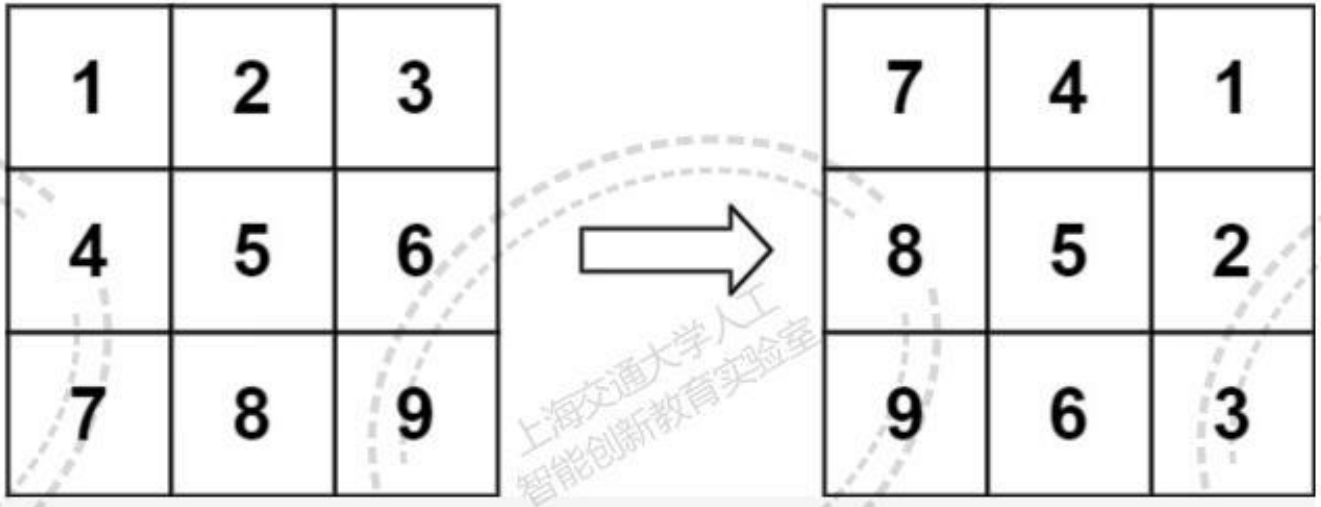
思路三：转换成str类型完成倒序操作，再进行判断

```
return x == str(x)[::-1]
```

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

示例 1：



输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]
输出：[[7,4,1],[8,5,2],[9,6,3]]

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

1	2	3	1
1	2	3	4
1	2	3	4
4	2	3	4

1	1	3	1
1	2	3	2
3	2	3	4
4	2	4	4

1	1	1	1
2	2	3	2
3	2	3	3
4	4	4	4

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

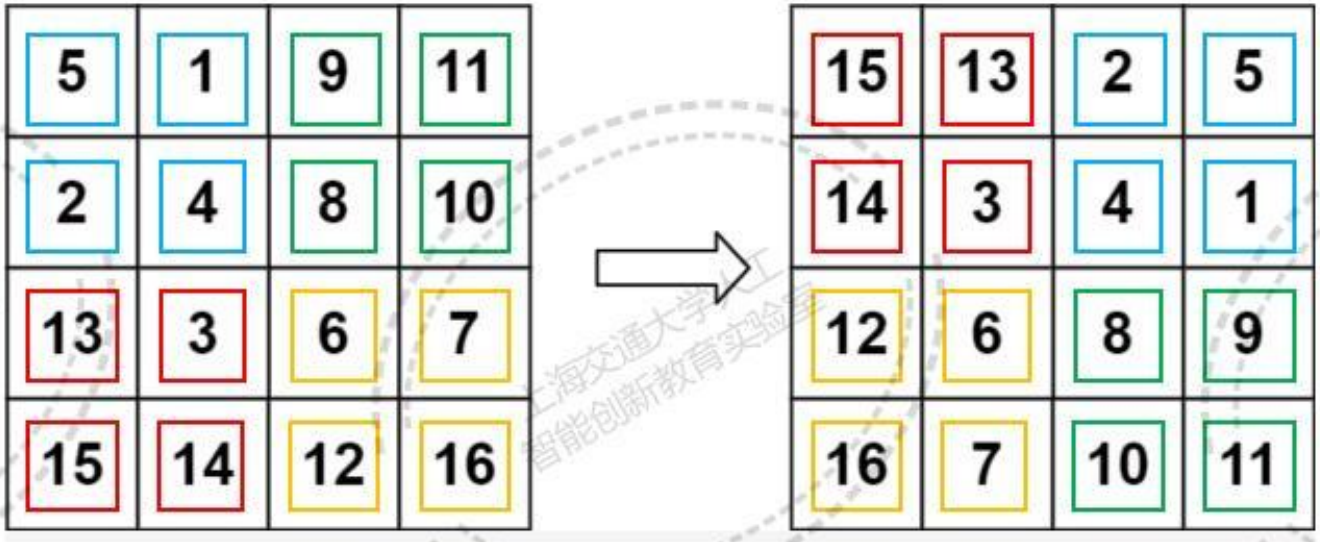
```
import numpy as np
n = int(input("The size of the sqare:"))
arr = []
mask = np.zeros([n,n],dtype = int)
line = []
temp = []
size = n
for i in range(n):
    for j in range(n):
        print("Input number (%d,%d)"%(i+1,j+1),end=":")
        num = input()
        num = int(num)
        line.append(num)
    arr.append(line)
    line = []
```

```
i=0
j=0
offset=0
while(True):
    arr[j][n-1-i],arr[n-1-i][n-1-j],arr[n-1-j][i],arr[i][j] = arr[i][j],arr[j][n-1- i],arr[n - 1 - i][n - 1 - j],arr[n - 1 - j][i]
    mask[i][j] = 1
    mask[j][n - 1 - i] = 0
    mask[n - 1 - i][n - 1 - j] = 0
    mask[n - 1 - j][i] = 0
    j+=1
    if(j-offset==size-1):
        size-=2
        i+=1
        offset+=1
        j=offset
    if(size<=1):
        break
print(arr)
```

动手试试⌨

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

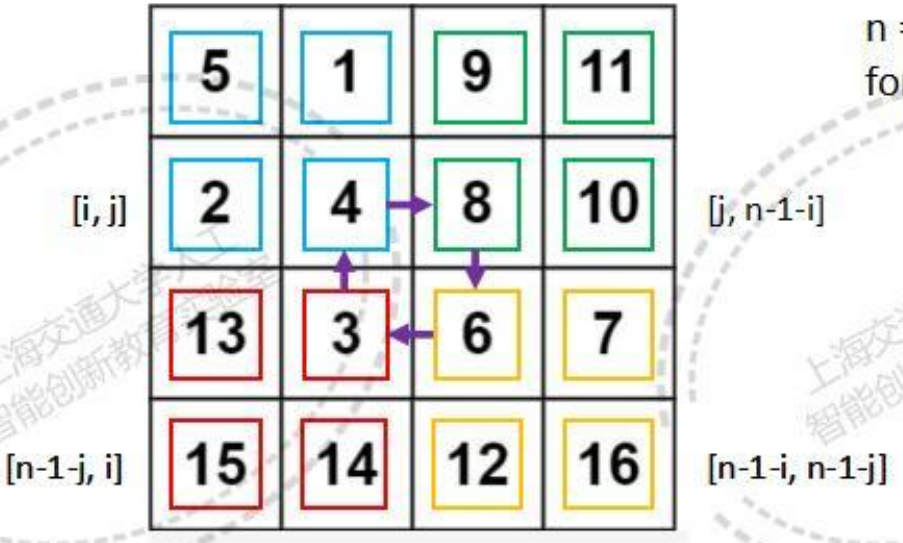
思路一：找出顺时针旋转90度对应坐标变换关系，遍历完成变换



动手试试⌨

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

思路一：找出顺时针旋转90度对应坐标变换关系，遍历完成变换

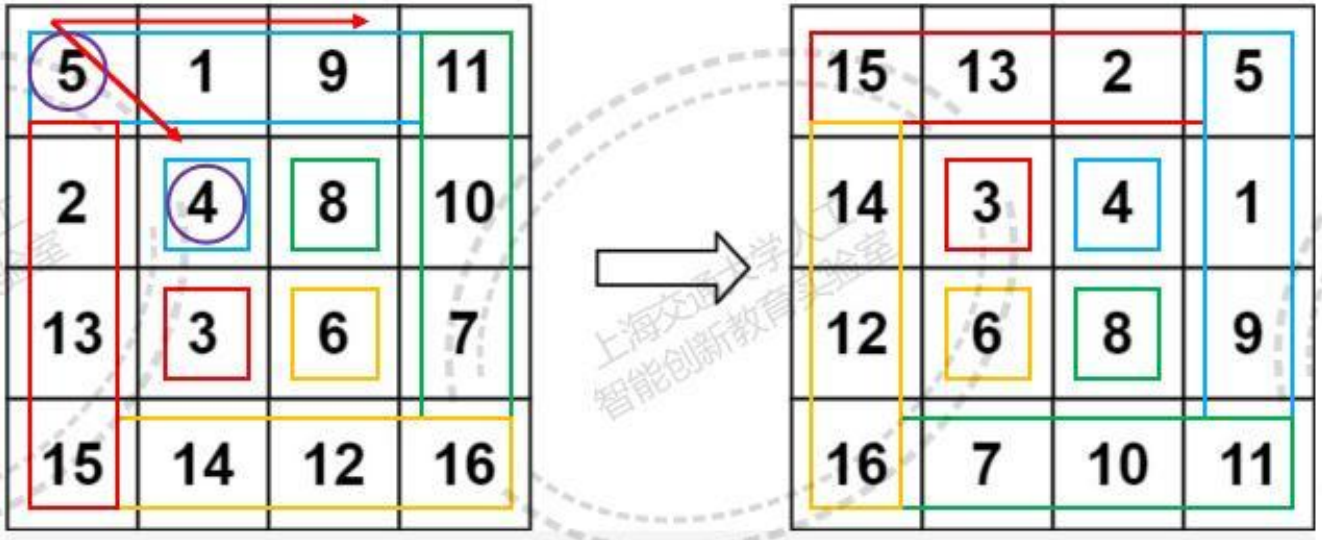



```
n = len(matrix)
for i in range(n // 2 + n % 2): # row
        for j in range(n // 2): # column
                temp = matrix[n - 1 - j][i]
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - 1 - j]
                matrix[n - 1 - i][n - 1 - j] = matrix[j][n - 1 - i]
                matrix[j][n - 1 - i] = matrix[i][j]
                matrix[i][j] = temp
```

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

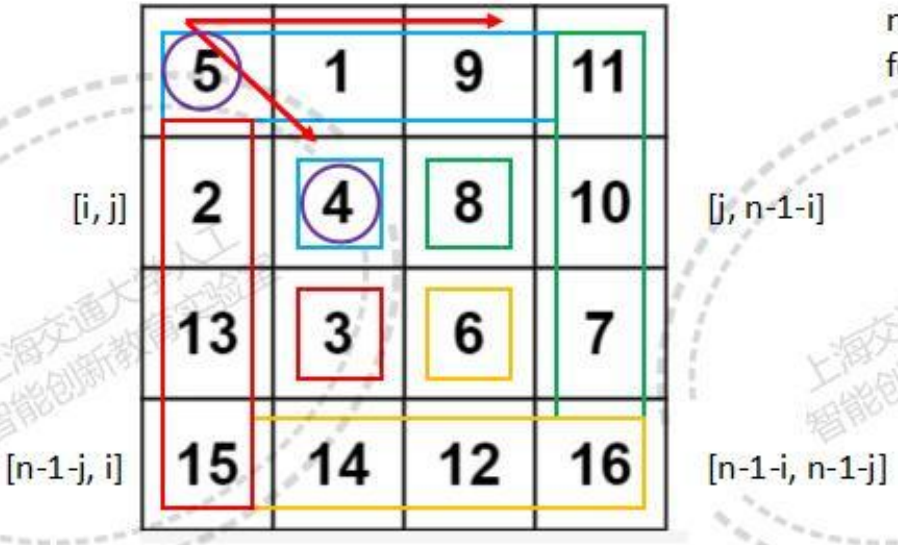
思路二：找出顺时针旋转90度对应坐标变换关系，遍历完成变换



动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

思路二：找出顺时针旋转90度对应坐标变换关系，遍历完成变换

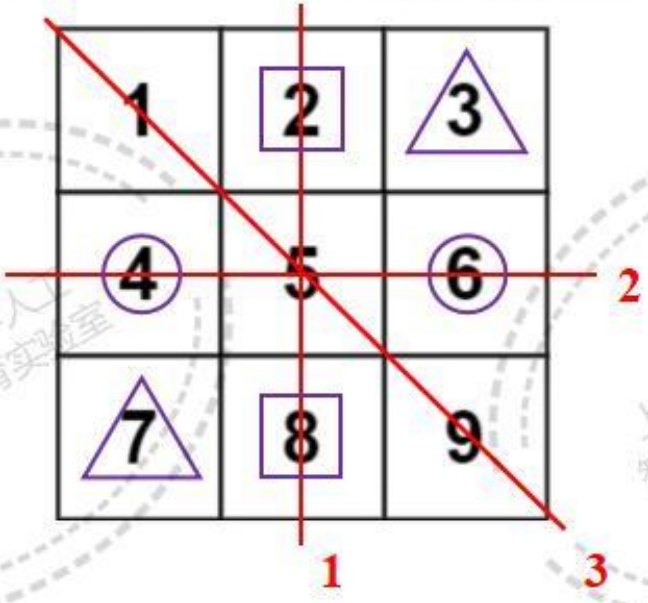



```
n = len(matrix)
for i in range(n // 2): # layer
        for j in range(i, n - i - 1): # layer size
                temp = matrix[n - 1 - j][i]
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - 1 - j]
                matrix[n - 1 - i][n - 1 - j] = matrix[j][n - 1 - i]
                matrix[j][n - 1 - i] = matrix[i][j]
                matrix[i][j] = temp
```

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

思路三：将旋转变换分解为基本的对角变换（矩阵转置）和左右对称变换组合



1: [i, j] <-> [i, n-j-1]
2: [i, j] <-> [n-i-1, j]
3: [i, j] <-> [j, i]

动手试试

3. 给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

思路三：将旋转变换分解为对角变换（矩阵转置）和左右对称变换

```
n = len(matrix)
for i in range(n):
        for j in range(i):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
for i in range(n):
        for j in range(n):
                matrix[i][j] = matrix[i][n-j-1]


    matrix[:] = zip(*matrix[::-1])
```

动手试试

总 结

1. 分析问题。确定已知与问题描述，将条件与要求转换成数学语言/编程语言。
2. 分而化之。大问题拆分为子问题，整流程拆分成子步骤。
3. 放宽条件。被问题卡住时先放宽条件解决，再思考原条件对应的难点是什么以及如何处理。
4. 工具思维。用什么样的数据结构？用什么样的操作？考虑不同工具（数据结构）的优势和劣势，从不同角度去设计解法和实现。
5. 边界条件/边角案例（corner cases）。可以将输入分为统一进行处理的/需要单独处理的，在确定循环次数、索引下标、跳出条件等时需要考虑不同输入的特点进行设置。
6. 想法实现。解决的思路、实现的过程会遇到不同的难点，先想清楚问题与解法，再动手写代码。
7. 取舍/折中（Tradeoff）。具体问题背景下，时间与空间复杂度的权衡。
8. Have fun!


谢谢聆听
THANKS FOR YOUR ATTENTION