作业

1、打印图形

```
上三角
                          1
                        2 1
                      3 2 1
                    4 3 2 1
                  5 4 3 2 1
                6 5 4 3 2 1
              7 6 5 4 3 2 1
            8 7 6 5 4 3 2 1
          9 8 7 6 5 4 3 2 1
       10 9 8 7 6 5 4 3 2 1
    11 10 9 8 7 6 5 4 3 2 1
12 11 10 9 8 7 6 5 4 3 2 1
```

可以右对齐打印，但是，你不能事先确定出打印宽度，因为直到最后一行打印才知道，怎么办？

能否先得出最后一行？

```python
def print_triangle(n):
    tail = " ".join(map(str, range(n, 0, -1)))
    width = len(tail) # 利用最后一行确定打印宽度
    # 从第一行打印到倒数第二行，最后补上tail
    for i in range(1, n):
        line = " ".join(map(str, range(i, 0, -1)))
        print("{:>{}}".format(line, width))
    print(tail)

print_triangle(12)
```

看似不错，但是上面的代码频繁的拼接每一行字符串，非常消耗时间。

能否不用拼接每一行？

```python
def print_triangle(n):
    tail = " ".join(map(str, range(n, 0, -1)))
    width = len(tail)
    step = 2
    start = -1
    points = {10**i for i in range(1, 3)} #set
    for i in range(1, n+1):
        line = tail[start:]
        print("{:>{}}".format(line, width))
        if i + 1 in points: # 先判断要不要换步长，只要碰到10的幂就步长加1
            step += 1
        start = start - step

print_triangle(12)
```

以上2种算法，去掉print打印语句测试，第2种切片省去了拼接字符串的过程，性能非常高。

```
下三角
12 11 10 9 8 7 6 5 4 3 2 1
   11 10 9 8 7 6 5 4 3 2 1
      10 9 8 7 6 5 4 3 2 1
         9 8 7 6 5 4 3 2 1
           8 7 6 5 4 3 2 1
             7 6 5 4 3 2 1
               6 5 4 3 2 1
                 5 4 3 2 1
                   4 3 2 1
                     3 2 1
                       2 1
                         1
```

很显然，计算第一行，然后切片，右对齐

```python
def print_triangle(n):
    tail = " ".join(map(str, range(n, 0, -1)))
    width = len(tail)
    step = len(str(n)) + 1 # *
    start = 0
    points = {10**i for i in range(1, 3)} #set
    for i in range(n, 0, -1):
        line = tail[start:]
        print("{:>{}}".format(line, width))
        if i + 1 in points: # 先判断要不要换步长，只要碰到10的幂就步长减1
            step -= 1 # *
        start = start + step

print_triangle(12)
```

以上所有代码，计算步长是不完美的，能否不用步长?

以字符串本身字符迭代，碰到空格就截取。这个思路更加简单

```python
def print_triangle(n): # 下三角
    tail = " ".join(map(str, range(n, 0, -1)))
    print(tail)
    width = len(tail)
    for i, c in enumerate(tail):
        if c == ' ':
            print("{:>{}}".format(tail[i+1:], width))

print_triangle(12)
```

```python
def print_triangle(n): # 上三角
    tail = " ".join(map(str, range(n, 0, -1)))
    width = len(tail)
    for i in range(1, width+1): # 用enumerate不合适
        if tail[-i] == ' ':
            print("{:>{}}".format(tail[-i+1:], width))
    print(tail)
print_triangle(12)
```

2、比较函数

编写一个函数，能够接受**至少**2个参数，返回最小值和最大值

下面几种实现，哪一种好些?

```python
# 使用内建函数完成，谁好
def get_values(a, b, *args): # 最大值、最小值
    src = (a, b, *args)
    return min(src), max(src)

def get_values(a, b, *args): # 排序
    m, *_, n = sorted((a, b, *args))
    return m, n
```

```python
# 自己实现
def get_values(a, b, *args):
    m, n = (b, a) if a > b else (a, b)
    for i in args:
        if i > n:
            n = i
        elif i < m:
            m = i
    return m, n

print(get_values(*range(10)))
```

3、编写mymap函数

编写一个函数，能够实现内建函数map的功能，函数签名 `def mymap(func, iterable, /)`

```python
def mymap(func, iterable, /):
    for x in iterable:
        yield func(x)

f = mymap(lambda x: x**2, range(5))
for i in f:
    print(i)
```