

作业

1、打印图形

```
1 上三角
2
3      1
4     2 1
5    3 2 1
6   4 3 2 1
7  5 4 3 2 1
8 6 5 4 3 2 1
9 7 6 5 4 3 2 1
10 8 7 6 5 4 3 2 1
11 9 8 7 6 5 4 3 2 1
12 10 9 8 7 6 5 4 3 2 1
13 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？

```
1 def print_triangle(n):
2     tail = " ".join(map(str, range(n, 0, -1)))
3     width = len(tail) # 利用最后一行确定打印宽度
4     # 从第一行打印到倒数第二行，最后补上tail
5     for i in range(1, n):
6         line = " ".join(map(str, range(i, 0, -1)))
7         print("{:>{}}".format(line, width))
8     print(tail)
9
10 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？

```
1 def print_triangle(n):
2     tail = " ".join(map(str, range(n, 0, -1)))
3     width = len(tail)
4     step = 2
5     start = -1
6     points = {10**i for i in range(1, 3)} #set
7     for i in range(1, n+1):
8         line = tail[start:]
9         print("{:>{}}".format(line, width))
10        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
11            step += 1
12            start = start - step
13
14 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。

```

1  下三角
2  12 11 10 9 8 7 6 5 4 3 2 1
3    11 10 9 8 7 6 5 4 3 2 1
4      10 9 8 7 6 5 4 3 2 1
5        9 8 7 6 5 4 3 2 1
6          8 7 6 5 4 3 2 1
7            7 6 5 4 3 2 1
8              6 5 4 3 2 1
9                5 4 3 2 1
10               4 3 2 1
11              3 2 1
12             2 1
13            1

```

很显然，计算第一行，然后切片，右对齐

```

1  def print_triangle(n):
2      tail = " ".join(map(str, range(n, 0, -1)))
3      width = len(tail)
4      step = len(str(n)) + 1 # *
5      start = 0
6      points = {10**i for i in range(1, 3)} #set
7      for i in range(n, 0, -1):
8          line = tail[start:]
9          print("{:>{}}".format(line, width))
10         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
11             step -= 1 # *
12             start = start + step
13
14  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

```

1  def print_triangle(n): # 下三角
2      tail = " ".join(map(str, range(n, 0, -1)))
3      print(tail)
4      width = len(tail)
5      for i, c in enumerate(tail):
6          if c == ' ':
7              print("{:>{}}".format(tail[i+1:], width))
8
9  print_triangle(12)

```

```

1  def print_triangle(n): # 上三角
2      tail = " ".join(map(str, range(n, 0, -1)))
3      width = len(tail)
4      for i in range(1, width+1): # 用enumerate不合适
5          if tail[-i] == ' ':
6              print("{:>{}}".format(tail[-i+1:], width))
7      print(tail)
8  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？

```
1 # 使用内建函数完成，谁好
2 def get_values(a, b, *args): # 最大值、最小值
3     src = (a, b, *args)
4     return min(src), max(src)
5
6 def get_values(a, b, *args): # 排序
7     m, _, n = sorted((a, b, *args))
8     return m, n
```

```
1 # 自己实现
2 def get_values(a, b, *args):
3     m, n = (b, a) if a > b else (a, b)
4     for i in args:
5         if i > n:
6             n = i
7         elif i < m:
8             m = i
9     return m, n
10
11 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, /)`

```
1 def mymap(func, iterable, /):
2     for x in iterable:
3         yield func(x)
4
5 f = mymap(lambda x: x**2, range(5))
6 for i in f:
7     print(i)
```