# 习题

## 求100内的素数

```
n = 100

# 基本做法
# 一个数能被从2开始到自己的平发根的正整数整数整除，就是合数
count = 0
primenumber = []
for x in range(2, n):
    for i in range(2, int(x ** 0.5) + 1):
        if x % i == 0:
            break
    else:
        #primenumber.append(x)
        count += 1

#print(primenumber)
print(count)
print('-'*30)

# 改进1
# 存储质数
# 合数一定可以分解为几个质数的乘积，2是质数
# 质数一定不能整除1和本身之内的整数
count = 0
primenumber = []
for x in range(2, n):
    for i in primenumber:
        if x % i == 0:
            break
    else:
        primenumber.append(x)
        count += 1

#print(primenumber)
print(count)
print('-'*30)


# 改进2
# 使用列表存储已有的质数，同时缩小范围
count = 0
primenumber = []
flag = False
for x in range(2, n):
    for i in primenumber:
        if x % i == 0 :
```

```
            flag = True
            break
        if i > x ** 0.5:
            flag = False
            break
    if not flag:
        primenumber.append(x)
        count += 1

#print(primenumber)
print(count)
print('-'*30)
```

打印出上例i和x的开方的关系

```
开方数    当前素数表
1.73     [2]
2.24     [2, 3]
2.65     [2, 3, 5]
3.00     [2, 3, 5, 7]
3.32     [2, 3, 5, 7]
3.61     [2, 3, 5, 7, 11]
3.87     [2, 3, 5, 7, 11, 13]
4.12     [2, 3, 5, 7, 11, 13]
4.36     [2, 3, 5, 7, 11, 13, 17]
4.58     [2, 3, 5, 7, 11, 13, 17, 19]
4.80     [2, 3, 5, 7, 11, 13, 17, 19]
5.00     [2, 3, 5, 7, 11, 13, 17, 19, 23]
5.20     [2, 3, 5, 7, 11, 13, 17, 19, 23]
5.39     [2, 3, 5, 7, 11, 13, 17, 19, 23]
5.57     [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
5.74     [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
5.92     [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
6.08     [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
6.24     [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
6.40     [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```



可以看出，和x的开方值比较可以大大减少

比较一下上面算法对效率的提升

```
import datetime

upper_limit = 10000

# 基本做法
# 一个数能被从2开始到自己的平发根的正整数整数整除，就是合数
start = datetime.datetime.now()
count = 1
```

```python
for x in range(3, upper_limit, 2): # 舍弃掉所有偶数
    if x > 10 and x % 5 == 0:
        continue # 所有大于10的质数中，个位数只有1,3,7,9
    for i in range(3, int(x ** 0.5) + 1, 2): # 奇数除以偶数才可能整除
        if x % i == 0:
            break
    else:
        count += 1

delta = (datetime.datetime.now() - start).total_seconds()
print(delta)
#print(primenumber)
print(count)
print('-'*30)


# 改进2
# 使用列表存储已有的质数，同时缩小范围
start = datetime.datetime.now()
count = 0
primenumber = []
flag = False
for x in range(2, upper_limit):
    for i in primenumber:
        if x % i == 0 :
            flag = True
            break
        if i > x ** 0.5:
            flag = False
            break
    if not flag:
        primenumber.append(x)
        count += 1

delta = (datetime.datetime.now() - start).total_seconds()
print(delta)
#print(primenumber)
print(count)
print('-'*30)
```

结果是增加了质数列表反而慢了，为什么？

修改算法如下

```python
# 需要从一下几个地方改进
# 1 忽略偶数
# 2 if i > x ** 0.5 在i循环中只需计算一次
# 使用列表存储已有的质数，同时缩小范围
start = datetime.datetime.now()
count = 1
primenumber = []
flag = False
for x in range(3, upper_limit, 2):
```

```python
    edge = x ** 0.5
    for i in primenumber:
        if x % i == 0 :
            flag = True
            break
        if i > edge:
            flag = False
            break
    if not flag:
        primenumber.append(x)
        count += 1

delta = (datetime.datetime.now() - start).total_seconds()
print(delta)
#print(primenumber)
print(count)
print('-'*30)
```

这回测试，速度第一了。也就是增加了列表，记录了质数，控制了边界后，使用质数来取模比使用奇数计算更少。空间换时间。

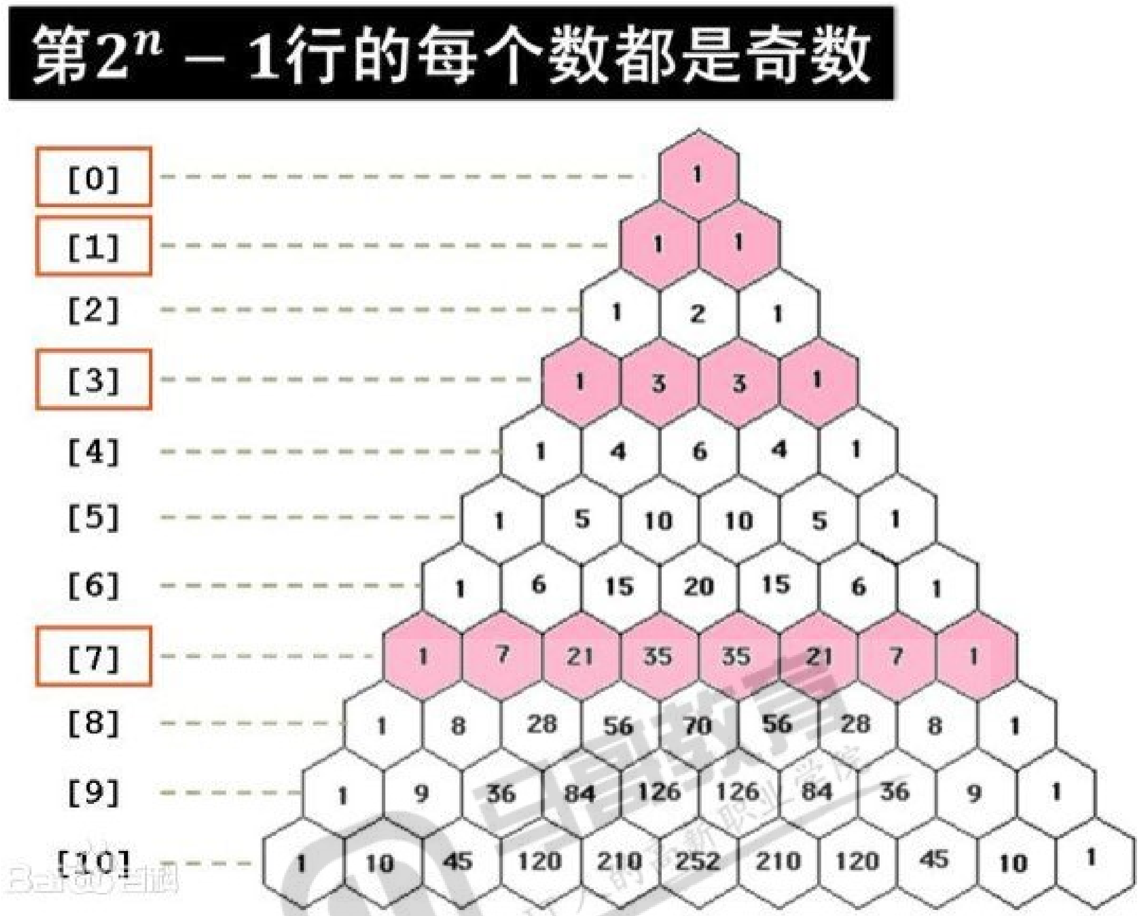
素数性质

> 大于3的素数只有6N-1和6N+1两种形式，如果6N-1和6N+1都是素数成为孪生素数

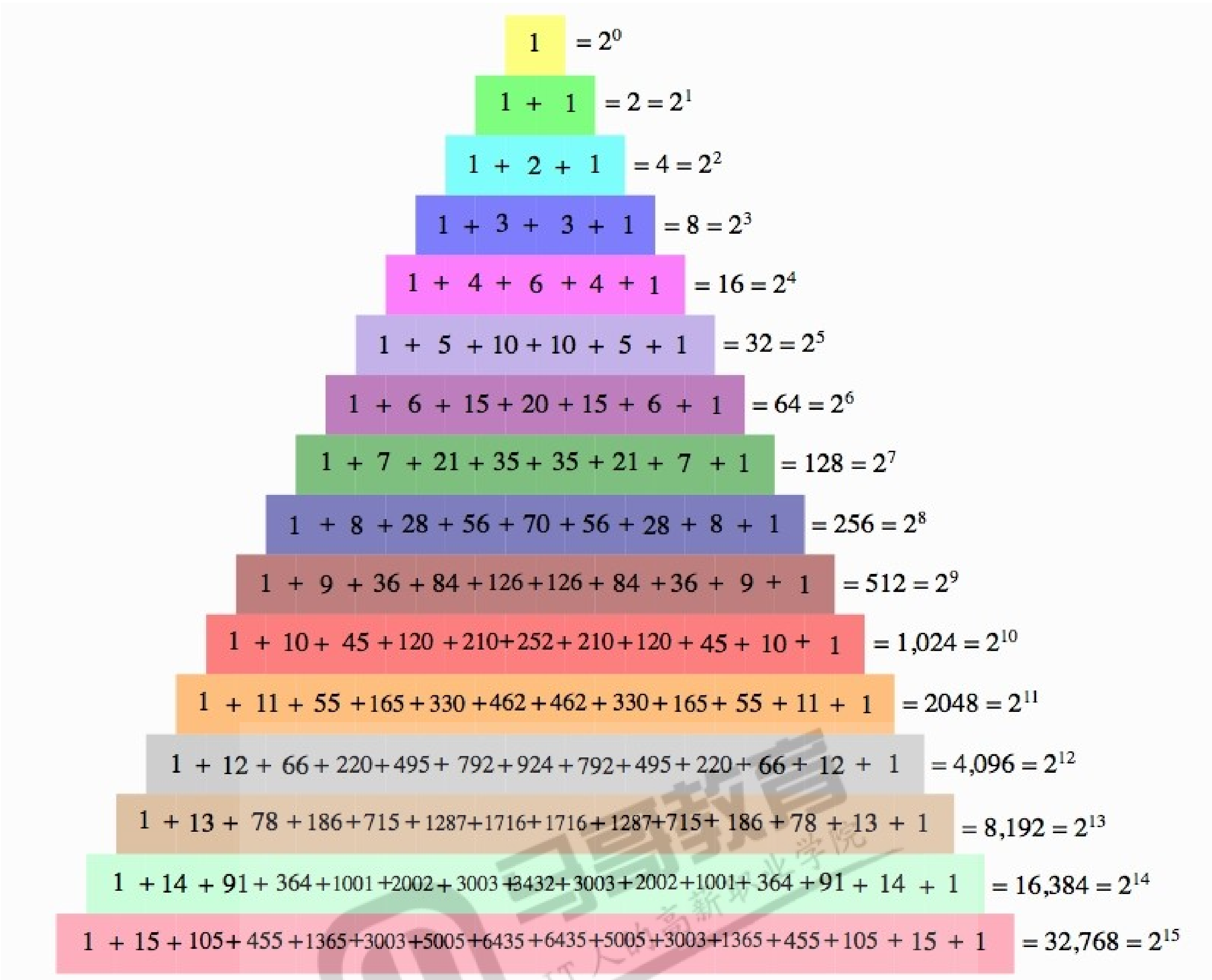
```python
# 大于3的素数只有6N-1和6N+1两种形式，如果6N-1和6N+1都是素数成为孪生素数
start = datetime.datetime.now()
count = 2 # 2,3
step = 2
x = 5 # 6的倍数前后都是奇数
while x < upper_limit:
    for i in range(3, int(x ** 0.5) + 1, 2):
        if x % i == 0:
            break
    else:
        count += 1

    # 下一个x是谁
    x += step
    step = 4 if step == 2 else 2

delta = (datetime.datetime.now() - start).total_seconds()
print(delta)
#print(primenumber)
print(count)
print('-'*30)
```

用了这个性质并没有提升效率，原因还是在于使用列表可以减少计算。

## 计算杨辉三角前6行



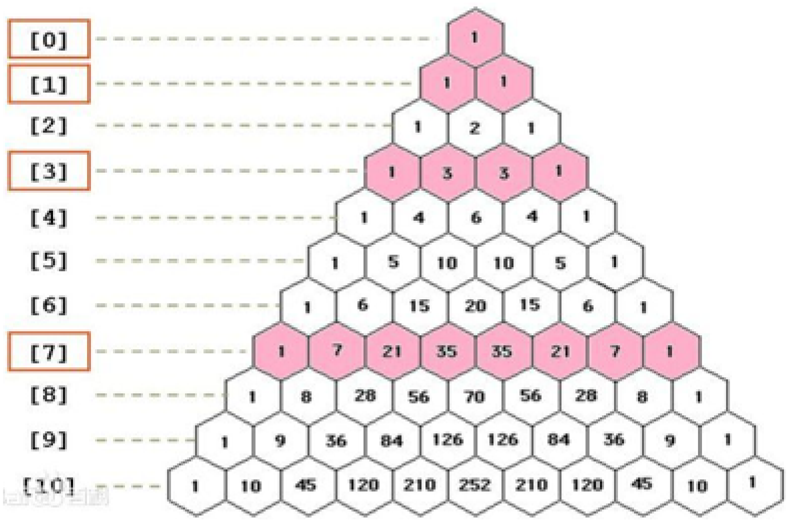

第n行有n项，n是正整数




第n行数字之和为2**(n-1)

## 杨辉三角的基本实现（方法1）

下一行依赖上一行所有元素，是上一行所有元素的两两相加的和，再在两头各加1




预先构建前两行，从而推导出后面的所有行

```python
triangle = [[1], [1, 1]]

for i in range(2, 6):
    cur = [1]
    pre = triangle[i-1]
    for j in range(len(pre)-1):
        cur.append(pre[j] + pre[j+1])
    cur.append(1)
    triangle.append(cur)
print(triangle)
```

**变体**

从第一行开始

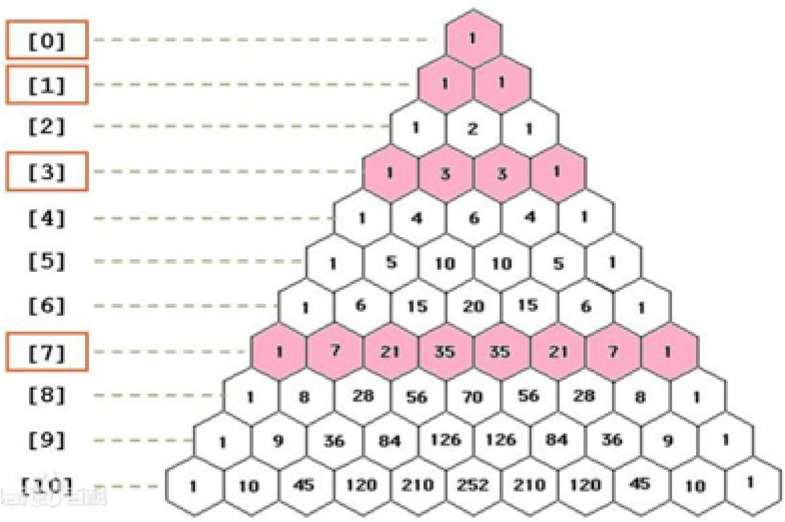
```python
triangle = []
n = 6
for i in range(n):
    cur = [1]
    triangle.append(cur)

    if i == 0:
        continue

    pre = triangle[i-1]
    for j in range(len(pre)-1):
        cur.append(pre[j] + pre[j+1])
    cur.append(1)

print(triangle)
```



## 补零（方法2）

除了第一行以外，每一行每一个元素（包括两头的1）都是由上一行的元素相加得到。如何得到两头的1呢？

目标是打印指定的行，所以算出一行就打印一行，不需要用一个大空间存储所有已经算出的行。




**while循环实现**

```python
n = 6
newline = [1] # 相当于计算好的第一行
print(newline)

for i in range(1, n):
    oldline = newline.copy() # 浅拷贝并补0
    oldline.append(0) # 尾部补0相当于两端补0
    newline.clear() # 使用append，所以要清除

    offset = 0
    while offset <= i:
        newline.append(oldline[offset-1] + oldline[offset])
        offset += 1
    print(newline)
```

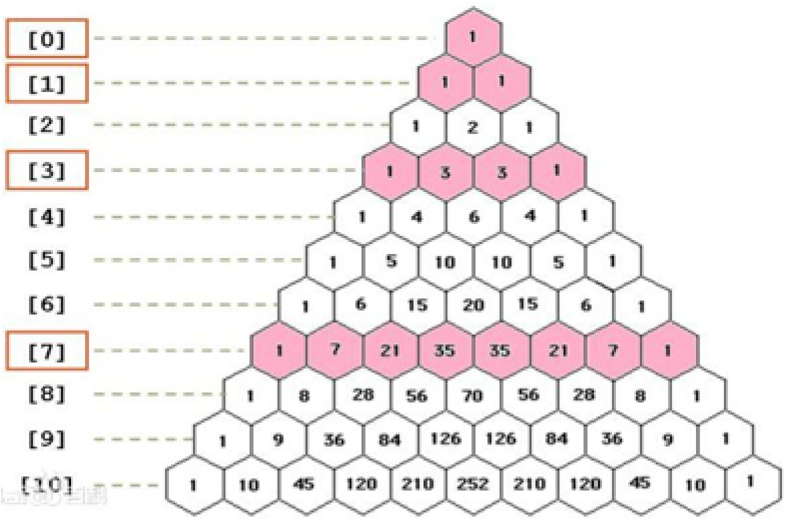
**for循环实现**

```python
n = 6
newline = [1] # 相当于计算好的第一行
print(newline)

for i in range(1, n):
    oldline = newline.copy() # 浅拷贝并补0
    oldline.append(0) # 尾部补0相当于两端补0
    newline.clear() # 使用append，所以要清除

    for j in range(i+1):
        newline.append(oldline[j - 1] + oldline[j])
    print(newline)
```



# 对称性（方法3）

思路：
能不能一次性开辟空间，可以使用列表解析式或者循环迭代的方式。
能不能减少一半的数字计算。



```
中点的确定
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
把整个杨辉三角看成左对齐的二维矩阵。
i==2时，在第3行，中点的列索引j==1
i==3时，在第4行，无中点
i==4时，在第5行，中点的列索引j==2
得到以下规律，如果有i==2j，则有中点
```

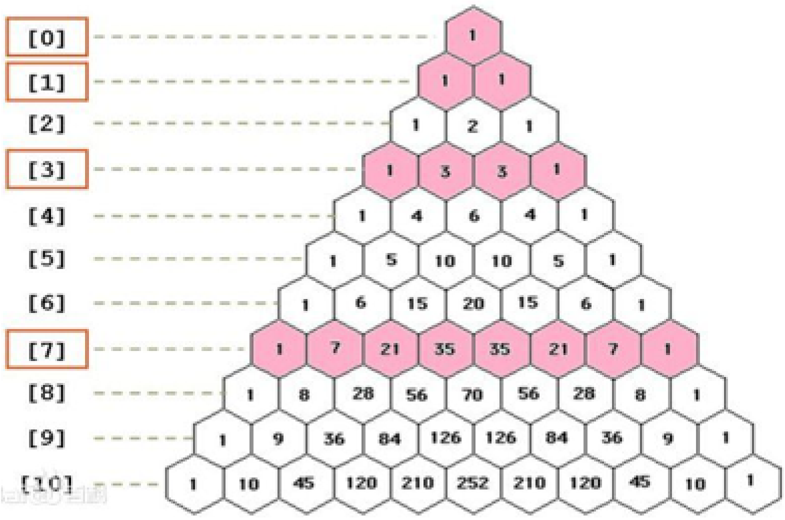
```python
triangle = []
n = 6
for i in range(n):
    row = [1] # 开始的1
    for k in range(i): # 中间填0，尾部填1
        row.append(1) if k == i-1 else row.append(0)
    triangle.append(row)
    if i == 0:
        continue
    for j in range(1,i//2+1): # i=2第三行才能进来
        #print(i, j)
        val = triangle[i - 1][j-1] + triangle[i - 1][j]
        row[j] = val
        # i为2，j为0 1 2，循环1次
        # i为3，j为0 1 2 3，循环1次
        # i为4，j为0 1 2 3 4，循环2次
        if i != 2*j: # 奇数个数的中点跳过
            row[-j-1] = val
print(triangle)
```

上面的代码看似不错，但行初始化的代码明显繁琐了，进一步简化

```python
triangle = []
n = 6
for i in range(n):
    row = [1] * (i+1) # 一次性开辟
    triangle.append(row)
    for j in range(1,i//2+1): # i=2第三行才能进来
        #print(i, j)
        val = triangle[i - 1][j-1] + triangle[i - 1][j]
        row[j] = val
        if i != 2*j: # 奇数个数的中点跳过
            row[-j-1] = val
print(triangle)
```

## 单行覆盖（方法4）

方法2每次都要清除列表，有点浪费时间。
能够用上方法3的对称性的同时，只开辟1个列表实现吗？



首先我们明确的知道所求最大行的元素个数，例如前6行的最大行元素个数为6个。
下一行等于首元素不变，覆盖中间元素。

```
n = 6
row = [1] * n # 一次性开辟足够的空间

for i in range(n):
    offset = n - i
    z = 1 # 因为会有覆盖影响计算，所以引入一个临时变量
    for j in range(1, i//2+1): # 对称性
        val = z + row[j]
        z = row[j]
        row[j] = val
        if i != 2*j:
            row[-j-offset] = val
    print(row[:i+1])
```