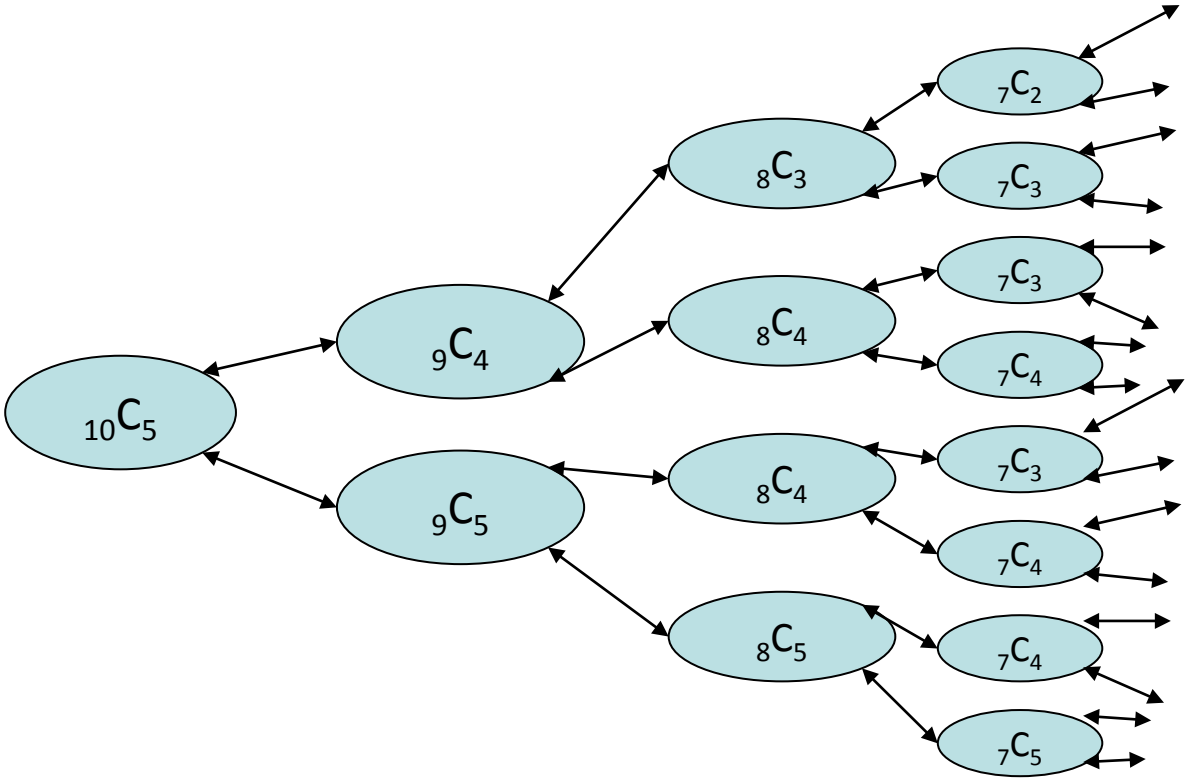
関数から計算へ

練習4.2

- n 個から k 個を選ぶ組み合わせ数 ${}_nC_k$ を求める combination(n,k) を定義せよ。（combination.rb）

```
def combination(n,k)
    if k==0
        1
    else
        if k==n
            1
        else
            combination(n-1,k-1)+combination(n-1,k)
        end
    end
end
```

combination.rb



同じ計算が繰り返し行われてしまう。

1 と + のみで値を組み上げるので、
$_nC_k$ の計算をするのに、$_nC_k$ に比例した時間がかかってしまう。あたりまえ！

配列と繰り返しを使った組み合わせ数の計算

$n \setminus k$	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

表 4.1: 組み合わせ数 ${}_nC_k$ の値

```
load ("./make2d.rb")

def combination_loop(n,k)
    c = make2d(n+1,n+1)
    for i in 0..n
        c[i ][0] = 1
        for j in 1..(i-1)
            c[i][j] = c[i-1][j-1] + c[i-1][j]
        end
        c[i][i] = 1
    end
    c[n][k]
end
```

combination_loop.rb

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2							
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1						
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2					
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1						
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3					
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3				
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

Pascal の三角形

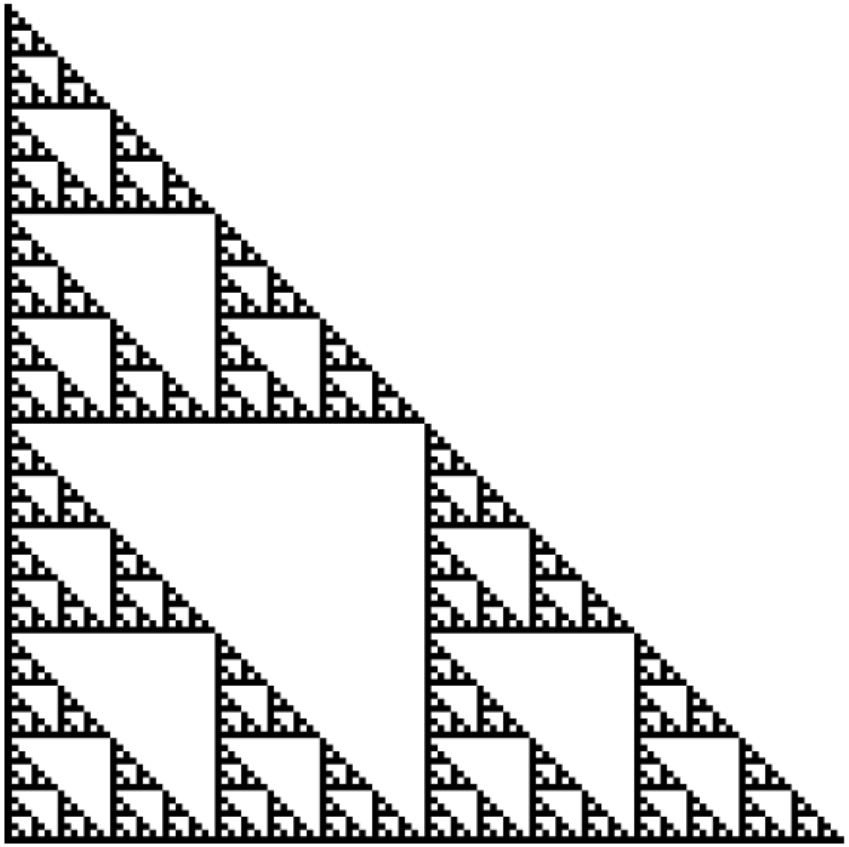
n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

Sierpinski の三角形

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	0	1				
3	1	1	1	1			
4	1	0	0	0	1		
5	1	1	0	0	1	1	
6	1	0	1	0	1	0	1

練習4.15

- Sierpinski の三角形
  - 大きさ n*n で、i 行 j 列目が ${}_iC_j$ を 2 で割った余りになっているような配列を作る関数 sierpinski(n) を定義せよ。（見易さのために白黒を逆にしている。）
  - sierpinski.rb に。

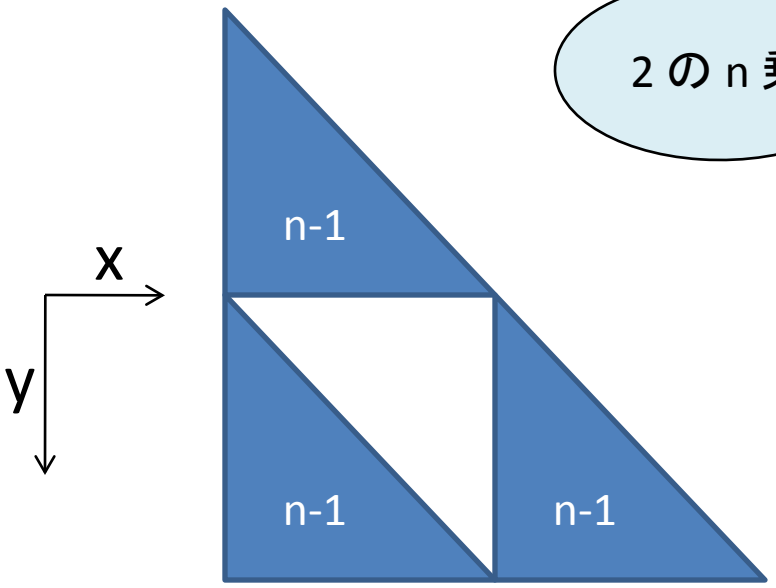


```
def sierpinski(n)
    c = make2d(n,n)
    for i in 0..(n-1)
        c[i][0] = 1
        for j in 1..(i-1)
            c[i][j] = (c[i-1][j-1] + c[i-1][j])%2
        end
        c[i][i] = 1
    end
    c
end
```

Sierpinski の三角形の別の作り方

```
def sier(n)
    a = make2d(2**n, 2**n)
    subsier(a, n, 0, 0)
    a
end
```

2 の n 乗

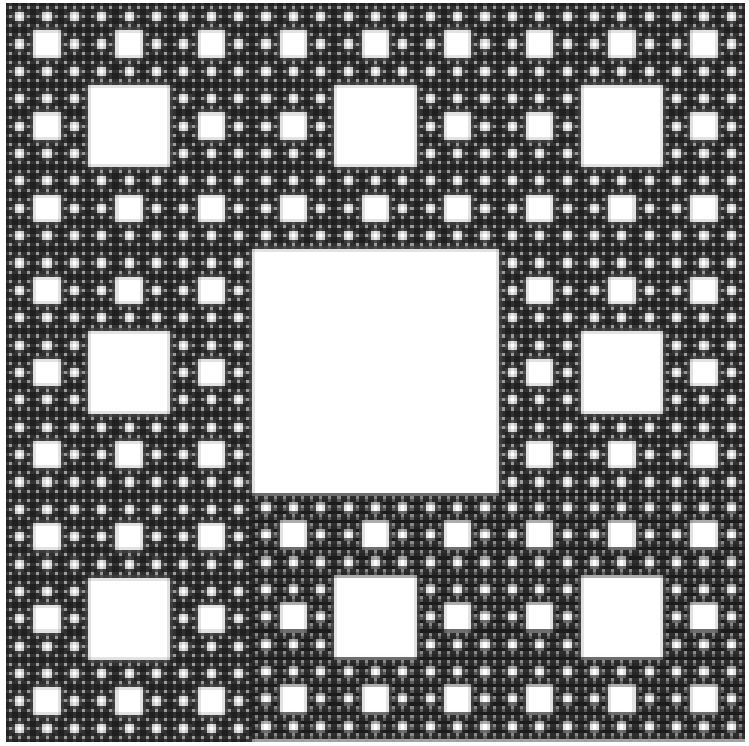



```
def subsier(a, n, x, y)
    if n==0
        a[y][x] = 1
    else
        m = 2**(n-1)
        subsier(a, n-1, x, y)
        subsier(a, n-1, x, y+m)
        subsier(a, n-1, x+m, y+m)
    end
end
```

局所変数
m の設定

練習

- Sierpinski のカーペット
  - n 次のカーペットは、縦横が 3**n で、n-1 次ののカーペットを 8 枚敷き詰めて作られる。真ん中は空いている。0 次のカーペットは、縦横 1 の黒い正方形とする。（実際のプログラムでは、白黒が反転する。）



Hilbert曲線

4.4　定義のまとめ

条件を満たすまでの繰り返し: 式 が成り立つ間 命令$_1$ から 命令$_n$ を毎回実行する**繰り返し**は次のように書く。

```
while 式
    命令1
    ⋮
    命令n
end
```

条件を満たす値を探す繰り返し

```
load ("./divisible.rb")


def gd_loop(k,n)
    while !divisible(k,n)
        n = n-1
    end
    n
end
```

gd_loop.rb