

7주차: 재귀함수

```
33 self.fingerprints = {}
34 self.logdups = True
35 self.debug = debug
36 self.logger = logging.getLogger(__name__)
37 if path:
38     self.file = open(os.path.join(path, 'requests.log'), 'a')
39     self.file.seek(0)
40     self.fingerprints.update({x: 0 for x in self.fingerprints})
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.get('SUPERFILTER_DEBUG')
45     return cls(debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     fp = fingerprint(request)
```

커리큘럼

1. 변수, 입출력
2. 조건문, 반복문
3. 리스트, 튜플, 세트, 딕셔너리
4. 함수
5. class
6. 알고리즘 입문, 그리디 알고리즘
7. 재귀함수
8. 탐색
9. DP(Dynamic Programming)

재귀함수

- 재귀대명사 : myself, yourself,...
- 자기 자신을 호출하는 함수
 - ✓ 자기 자신을 호출해서 불필요한 코드를 단축
 - ✓ 메모리를 많이 차지함, 상대적으로 느림

과제 2-2 (피보나치 수열)

피보나치 수열의 항을 구하는 문제. n 번째 피보나치 수열의 항을 구하시오.

$$a_n = a_{n-1} + a_{n-2} \ (n \geq 2), a_0 = 1, a_1 = 1$$

```
n=int(input())
left=1
right=1
result=0
for i in range(n+1):
    if i==0 or i==1:
        result=1
        continue
    result=left+right
    right=left
    left=result
print(result)
```

```
def fibonacci(n):
    if n==0:
        return 1
    elif n==1:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)

n=int(input())
a = fibonacci(n)
print(a)
```

(참고)꼬리 재귀

재귀함수의 직관성과 반복문의 효율성을 동시에 만족하는 방법

```
def fibonacci(n):  
    return fibonacci_Tail(n,1,1)  
def fibonacci_Tail(n,left,right):  
    if n==0:  
        return right  
    else:  
        return fibonacci_Tail(n-1,left+right,left)  
  
n=int(input())  
a = fibonacci(n)  
print(a)
```

팩토리얼

$$0! = 1, \quad n! = n \times (n - 1)!$$

```
import math

n=int(input())
a=math.factorial(n)
print(n)
```

```
def factorial(n):
    if n==0:
        return 1
    else:
        return n*factorial(n-1)

n=int(input())
a=factorial(n)
print(n)
```

분할 정복법(Divide & Conquer)

- 어떠한 문제가 있을 때, 그 문제를 나누고(Divide) 합치면서 해결하는(Conquer) 문제 해결방법 (나누어서-풀어서-합치기)
- POINT: 어떻게 나눌 것인가?
- 합병 정렬(Merge sort), 퀵 정렬(Quick sort) 등

분할 정복법 구현

- 자기 자신을 호출, 작업단위 줄임
- 간단 : 작업(Conquer)
- 복잡 : 분할(Divide)

```
def F(x):  
    if F(x)가 간단:  
        return F(x) 계산  
    else:  
        x를 x1, x2로 분할  
        F(x1)과 F(x2)를 호출  
        return F(x1), F(x2)로 표현
```


합병 정렬(Merge sort)

```
def mergesort(a):
    n=len(a)
    if n<=1:
        return a
    mid=n//2
    left=mergesort(a[:mid])
    right=mergesort(a[mid:])
    result=[]
    while len(left)>0 and len(right)>0:
        if left[0]<right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    if len(left)>0:
        result=result+left
    else:
        result=result+right
    return result
```

퀵 정렬(Quick sort)

```
def quicksort(a):  
    n=len(a)  
    if n<=1:  
        return a  
    left,mid,right=[],[],[]  
    for i in a:  
        if i<a[0]:  
            left.append(i)  
        elif i>a[0]:  
            right.append(i)  
        else:  
            mid.append(i)  
    return quicksort(left)+mid+quicksort(right)
```

```
31 def __init__(self, path):
32     self.file = None
33     self.fingerprints = set()
34     self.logdups = True
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(os.path.join(path, 'requests.log'),
39                         'a')
40         self.file.seek(0)
41         self.fingerprints.update(e.request() for e in self._requests)
```

```
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('DEBUG_FILTER_REQUESTS')
45     return cls(settings.get('LOG_DIR', settings.get('LOG_DIR', 'log')), debug)
```

```
46 def request_seen(self, request):
47     fp = self.request_fingerprint(request)
48     if fp in self.fingerprints:
49         return True
50     self.fingerprints.add(fp)
51     if self.file:
52         self.file.write(fp + os.linesep)
```

```
53 def request_fingerprint(self, request):
54     return request_fingerprint(request)
55
56
```

과제

7-1 (하노이탑)

A, B, C 막대가 있을 때, A 막대에 n 개의 하노이탑 원반이 있다. 이를 B 막대로 모두 옮기고자 할 때 아래 규칙을 만족해서 최소 횟수로 옮긴다고 한다면, 옮기는 과정을 출력하시기 바랍니다.

1. 한 번에 하나의 원반만 이동 가능하다.
2. 이동할 때 가장 위의 원반만 이동 가능하다.
3. 모든 원반의 크기가 다르고, 작은 원반 위에 큰 원반이 있으면 안된다.

입력 예시 3

출력 예시

```
Move no.1 disk : A to B
Move no.2 disk : A to C
Move no.1 disk : B to C
Move no.3 disk : A to B
Move no.1 disk : C to A
Move no.2 disk : C to B
Move no.1 disk : A to B
```

7-2

한 변의 길이가 n 인 프랙탈 출력하기 바랍니다.

9

입력 예시

```
*****
*  **  **  *
*****
***      ***
*  *      *  *
***      ***
*****
*  **  **  *
*****
```

출력 예시

7-3

$N \times N$ 행렬의 거듭제곱(A^B)을 구현하시기 바랍니다.

```
2 5
1 2
3 4
```

입력 예시1

첫번째 줄 : 행렬의 크기 N, B 제공

두번째 줄: 행렬 A

```
5 10
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
```

입력 예시2

```
69 558
337 406
```

출력 예시1

```
512 0 0 0 512
512 0 0 0 512
512 0 0 0 512
512 0 0 0 512
512 0 0 0 512
512 0 0 0 512
```

출력 예시2