>>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>>

프로그래밍의 정석

파이썬

도경구 지음

생능출판

CHAPTER 4

재귀와 반복 : 자연수 계산

자연수
Natural Number

$$N = \{0, 1, 2, 3, \ldots\}$$

자연수
Natural Number

수학적 귀납歸納,인덕

Mathematical Induction

(1)	기초 Basis	0은 자연수 이다.
(2)	귀납 Induction	n이 자연수이면, n+1도 자연수이다.
(3)	그 외에 다른 자연수는 없다.	

자연수는 무한이 많이 있지만
이 귀납 정의를 이용하면
아무리 큰 자연수라도
유한한 절차를 거쳐서
자연수임을 확인할 수 있다.

CHAPTER 4

재귀와 반복 : 자연수 계산

자연수 수열의 합

덧셈만으로 0부터 n까지 누적 합을 계산하여 내주는 함수 sigma(n)을 만들자.

0 + 1 + 2 + 3 + … + n

자연수 수열의 합

덧셈만으로 0부터 n까지 누적 합을 계산하여 내주는 함수 sigma(n)을 만들자.

0 + 1 + 2 + 3 + … + n

귀납

(1)	기초 Basis	0은 자연수 이다.
(2)	인덕 Induction	n이 자연수이면, n+1도 자연수이다.

자연수 수열의 합

덧셈만으로 0부터 n까지 누적 합을 계산하여 내주는 함수 sigma(n)을 만들자.

0 + 1 + 2 + 3 + … + n

귀납

(1)	기초 Basis	0은 자연수 이다.
(2)	인덕 Induction	n이 자연수이면, n+1도 자연수이다.

$$\mathrm{sigma}(0) = 0 \quad [base]$$

자연수 수열의 합

덧셈만으로 0부터 n까지 누적 합을 계산하여 내주는 함수 sigma(n)을 만들자.

0 + 1 + 2 + 3 + … + n

귀납

(1)	기초 Basis	0은 자연수 이다.
(2)	인덕 Induction	n이 자연수이면, n+1도 자연수이다.

$$\text{sigma}(0) = 0 \quad [base]$$

$$\text{sigma}(n+1) = \text{sigma}(n) + (n+1) \quad [induction]$$

자연수 수열의 합

덧셈만으로 0부터 n까지 누적 합을 계산하여 내주는 함수 sigma(n)을 만들자.

0 + 1 + 2 + 3 + … + n

귀납

(1)	기초 Basis	0은 자연수 이다.
(2)	인덕 Induction	n이 자연수이면, n+1도 자연수이다.

$$\text{sigma}(0) = 0 \quad [base]$$

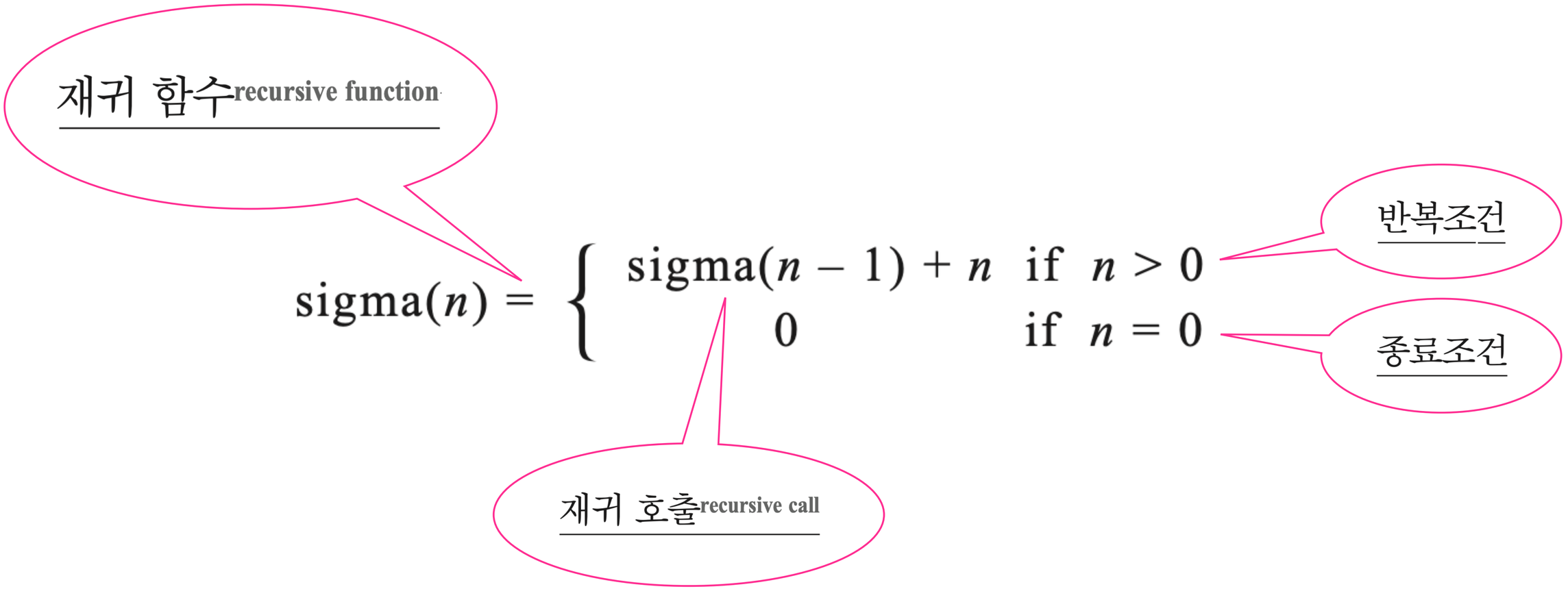
$$\text{sigma}(n+1) = \text{sigma}(n) + (n+1) \quad [induction]$$

재귀식 recursive equation

순진무구 알고리즘

$$\text{sigma}(n) = \begin{cases} \text{sigma}(n-1) + n & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

순진무구 알고리즘


재귀 함수recursive function
$\text{sigma}(n) = \begin{cases} \text{sigma}(n-1) + n & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$
반복조건
종료조건
재귀 호출recursive call

$$\text{sigma}(n) = \begin{cases} \text{sigma}(n-1) + n & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

다시쓰기 rewriting

$$\begin{aligned}
\text{sigma}(5) &= \text{sigma}(4) + 5 && [1] \\
&= (\text{sigma}(3) + 4) + 5 && [2] \\
&= ((\text{sigma}(2) + 3) + 4) + 51 && [3] \\
&= (((\text{sigma}(1) + 2) + 3) + 4) + 5 && [4] \\
&= ((((\text{sigma}(0) + 1) + 2) + 3) + 4) + 5 && [5] \\
&= ((((0 + 1) + 2) + 3) + 4) + 5 && [6] \\
&= (((1 + 2) + 3) + 4) + 5 \\
&= ((3 + 3) + 4) + 5 \\
&= (6 + 4) + 5 \\
&= 10 + 5 \\
&= 15
\end{aligned}$$

$$\text{sigma}(n) = \begin{cases} \text{sigma}(n-1) + n & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

code : 4-1.py

```
1  def sigma(n):
2      if n > 0:
3          return sigma(n-1) + n
4      else:
5          return 0
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=> ((((0 + 1) + 2) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=> ((((0 + 1) + 2) + 3) + 4) + 5
=> (((1 + 2) + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=> ((((0 + 1) + 2) + 3) + 4) + 5
=> (((1 + 2) + 3) + 4) + 5
=> ((3 + 3) + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=> ((((0 + 1) + 2) + 3) + 4) + 5
=> (((1 + 2) + 3) + 4) + 5
=> ((3 + 3) + 4) + 5
=> (6 + 4) + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=> ((((0 + 1) + 2) + 3) + 4) + 5
=> (((1 + 2) + 3) + 4) + 5
=> ((3 + 3) + 4) + 5
=> (6 + 4) + 5
=> 10 + 5
=>
```

```
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=> ((((0 + 1) + 2) + 3) + 4) + 5
=> (((1 + 2) + 3) + 4) + 5
=> ((3 + 3) + 4) + 5
=> (6 + 4) + 5
=> 10 + 5
=> 15
```

```python
def sigma(n):
    if n > 0:
        return sigma(n-1) + n
    else:
        return 0
```

계산 비용 분석

- 시간
- 공간

```
   sigma(5)
=> if 5 > 0: return sigma(5-1) + 5 else: return 0
=> sigma(4) + 5
=> (if 4 > 0: return sigma(4-1) + 4 else: return 0) + 5
=> (sigma(3) + 4) + 5
=> ((if 3 > 0: return sigma(3-1) + 3 else: return 0) + 4) + 5
=> ((sigma(2) + 3) + 4) + 5
=> (((if 2 > 0: return sigma(2-1) + 2 else: return 0) + 3) + 4) + 5
=> (((sigma(1) + 2) + 3) + 4) + 5
=> ((((if 1 > 0: return sigma(1-1) + 1 else: return 0) + 2) + 3) + 4) + 5
=> ((((sigma(0) + 1) + 2) + 3) + 4) + 5
=> (((((if 0 > 0: return sigma(0-1) + 0 else: return 0) + 1) + 2) + 3) + 4) + 5
=> ((((0 + 1) + 2) + 3) + 4) + 5
=> (((1 + 2) + 3) + 4) + 5
=> ((3 + 3) + 4) + 5
=> (6 + 4) + 5
=> 10 + 5
=> 15
```

재귀 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **sigma 함수의 계산 복잡도**
  - 시간
    - 덧셈의 횟수 (= 재귀 함수를 호출하는 횟수)에 비례
    - 인수가 n 일 때 덧셈을 총 n번(= 재귀 호출을 총 n번) 하므로 계산시간은 n에 비례
  - 공간
    - 재귀 함수를 호출하는 횟수에 비례 (답을 구해온 뒤에 더해야 할 수를 기억해둘 공간 필요)
    - 인수가 n 일 때 재귀 호출을 총 n번 하므로 필요 공간은 n에 비례

꼬리재귀 함수

Tail Recursive Function

재귀

code : 4-1.py

```
1  def sigma(n):
2      if n > 0:
3          return sigma(n-1) + n
4      else:
5          return 0
```

꼬리 재귀

code : 4-2.py

```
1  def sigma(n):
2      return loop(n, 0)
3
4  def loop(n, total):
5      if n > 0:
6          return loop(n-1, n+total)
7      else:
8          return total
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=> if 2 > 0: return loop(2-1,2+12) else: return 12
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=> if 2 > 0: return loop(2-1,2+12) else: return 12
=> loop(1,14)
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=> if 2 > 0: return loop(2-1,2+12) else: return 12
=> loop(1,14)
=> if 1 > 0: return loop(1-1,1+14) else: return 14
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=> if 2 > 0: return loop(2-1,2+12) else: return 12
=> loop(1,14)
=> if 1 > 0: return loop(1-1,1+14) else: return 14
=> loop(0,15)
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=> if 2 > 0: return loop(2-1,2+12) else: return 12
=> loop(1,14)
=> if 1 > 0: return loop(1-1,1+14) else: return 14
=> loop(0,15)
=> if 0 > 0: return loop(0-1,0+15) else: return 15
=>
```

```
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=> if 2 > 0: return loop(2-1,2+12) else: return 12
=> loop(1,14)
=> if 1 > 0: return loop(1-1,1+14) else: return 14
=> loop(0,15)
=> if 0 > 0: return loop(0-1,0+15) else: return 15
=> 15
```

```python
def sigma(n):
    return loop(n,0)

def loop(n,total):
    if n > 0:
        return loop(n-1,n+total)
    else:
        return total
```

```
   sigma(5)
=> loop(5,0)
=> if 5 > 0: return loop(5-1,5+0) else: return 0
=> loop(4,5)
=> if 4 > 0: return loop(4-1,4+5) else: return 5
=> loop(3,9)
=> if 3 > 0: return loop(3-1,3+9) else: return 9
=> loop(2,12)
=> if 2 > 0: return loop(2-1,2+12) else: return 12
=> loop(1,14)
=> if 1 > 0: return loop(1-1,1+14) else: return 14
=> loop(0,15)
=> if 0 > 0: return loop(0-1,0+15) else: return 15
=> 15
```

계산 비용 분석

- 시간
- 공간

꼬리재귀 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **꼬리재귀 sigma 함수의 계산 복잡도**
  - 시간
    - 덧셈의 횟수 (= 재귀 함수를 호출하는 횟수)에 비례
    - 인수가 n 일 때 덧셈을 총 n번(= 재귀 호출을 총 n번) 하므로 계산시간은 n에 비례
  - 공간
    - 인수의 크기와 상관없이 일정

code : 4-2.py

```
1  def sigma(n):
2      return loop(n, 0)
3
4  def loop(n, total):
5      if n > 0:
6          return loop(n-1, n+total)
7      else:
8          return total
```

보조 함수

지역화

code : 4-3.py

```
1  def sigma(n):
2      def loop(n,total):
3          if n > 0:
4              return loop(n-1,n+total)
5          else:
6              return total
7      return loop(n,0)
```

지역 함수

local function

캡슐화

Encapsulation

재귀

code : 4-1.py

```
1  def sigma(n):
2      if n > 0:
3          return sigma(n-1) + n
4      else:
5          return 0
```

하향식
Top-down

꼬리 재귀

code : 4-3.py

```
1  def sigma(n):
2      def loop(n,total):
3          if n > 0:
4              return loop(n-1,n+total)
5          else:
6              return total
7      return loop(n,0)
```

상향식
Bottom-up

꼬리 재귀

code : 4-3.py

```
1  def sigma(n):
2      def loop(n,total):
3          if n > 0:
4              return loop(n-1,n+total)
5          else:
6              return total
7      return loop(n,0)
```

while 루프

code : 4-4.py

```
1  def sigma(n):
2      n, total = n, 0
3      while n > 0:
4          n, total = n-1, n+total
5      return total
```

꼬리 재귀

code : 4-3.py

```
1  def sigma(n):
2      def loop(n,total):
3          if n > 0:
4              return loop(n-1,n+total)
5          else:
6              return total
7      return loop(n,0)
```

while 루프

code : 4-5.py

```
1  def sigma(n):
2      total = 0
3      while n > 0:
4          n, total = n-1, n+total
5      return total
```

계산 비용 분석

- 시간
- 공간

while 루프 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **while 루프로 작성한 sigma의 계산 복잡도**
  - 시간
    - 덧셈의 횟수 (= 루프를 반복하는 횟수)에 비례
    - 인수가 n 일 때 덧셈을 총 n번(= 루프 반복을 총 n번) 하므로 계산시간은 n에 비례
  - 공간
    - 인수의 크기와 상관없이 일정

지정문의 실행 순서

code : 4-5.py

```
1  def sigma(n):
2      total = 0
3      while n > 0:
4          n, total = n-1, n+total
5      return total
```

```
n = n - 1
total = n + total
```

```
total = n + total
n = n - 1
```

정리

- 하향식으로 작동하는 재귀 함수는 실행 논리를 직관적으로 표현할 수 있어서 코딩하기 쉬운 반면, 공간 효율이 떨어질 수 있다.
- 상향식으로 작동하는 꼬리재귀 함수나 `while` 루프 버전로 변환하면 공간을 절약할 수 있다.
- 재귀 함수, 꼬리재귀 함수, while 문은 모두 구조적으로 연관이 있어서 상호 기계적으로 변환할 수 있다.
- 상대적으로 사고하기 쉬운 하향식으로 재귀 함수를 먼저 작성하고, 꼬리재귀와 `while` 루프로 차례로 변환하여 함수를 다듬는 것은 권장할만한 코딩 습관이다.

파이썬과 재귀의 궁합

가우스 알고리즘

$$\mathrm{sigma}(n) = 1 + 2 + \cdots + (n - 1) + n$$

가우스 알고리즘

$$\text{sigma}(n) = 1 + 2 + \cdots + (n-1) + n$$

$$\text{sigma}(n) = n + (n-1) + \cdots + 2 + 1$$ 역순으로 나열

가우스 알고리즘

$$\text{sigma}(n) = 1 + 2 + \cdots + (n-1) + n$$

$$\text{sigma}(n) = n + (n-1) + \cdots + 2 + 1$$ 역순으로 나열

두 등식의 각 항의 아래와 위를 합함

$$\text{sigma}(n) + \text{sigma}(n) = (n+1) + (n+1) + \cdots + (n+1) + (n+1)$$

가우스 알고리즘

$$\text{sigma}(n) = 1 + 2 + \cdots + (n-1) + n$$

$$\text{sigma}(n) = n + (n-1) + \cdots + 2 + 1$$ 역순으로 나열

두 등식의 각 항의 아래와 위를 합함

$$\text{sigma}(n) + \text{sigma}(n) = (n+1) + (n+1) + \cdots + (n+1) + (n+1)$$

정리

$$2 \times \text{sigma}(n) = n \times (n+1)$$

가우스 알고리즘

$$\text{sigma}(n) = 1 + 2 + \cdots + (n-1) + n$$

$$\text{sigma}(n) = n + (n-1) + \cdots + 2 + 1$$ 역순으로 나열

↓ 두 등식의 각 항의 아래와 위를 합함

$$\text{sigma}(n) + \text{sigma}(n) = (n+1) + (n+1) + \cdots + (n+1) + (n+1)$$

↓ 정리

$$2 \times \text{sigma}(n) = n \times (n+1)$$

↓ 양변을 2로 나눔

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

가우스 알고리즘

$$\text{sigma}(n) = 1 + 2 + \cdots + (n-1) + n$$

$$\text{sigma}(n) = n + (n-1) + \cdots + 2 + 1$$ 역순으로 나열

두 등식의 각 항의 아래와 위를 합함

$$\text{sigma}(n) + \text{sigma}(n) = (n+1) + (n+1) + \cdots + (n+1) + (n+1)$$

정리

$$2 \times \text{sigma}(n) = n \times (n+1)$$

양변을 2로 나눔

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

code : 4-9.py

```
1 def sigma(n):
2     return n * (n + 1) // 2
```

수학적 귀납법 검증

- 자연수 n을 중심으로 펼치는 주장 또는 명제 P(n)은 모든 n에 대해서 성립하는가?

- 수학적 귀납법 증명
  - 기초 단계 base step:

    P(0)은 참이다.
  - 귀납 단계 induction step:

    임의의 자연수 n에 대해서, P(n)이 참이면, P(n+1)도 참이다.

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n + 1)}{2}$$

증명(수학적 귀납법)

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n + 1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0 + 1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0+1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

- 귀납단계 : 임의의 자연수 n에 대해서 n까지의 누적 합은 다음과 같다고 가정하자.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

이를 귀납가정induction hypothesis이라고 한다. 이제 $n+1$까지의 누적 합이 다음과 같은지 보이면 된다.

$$\text{sigma}(n+1) = \frac{(n+1) \times ((n+1)+1)}{2}$$

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0+1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

- 귀납단계 : 임의의 자연수 n에 대해서 n까지의 누적 합은 다음과 같다고 가정하자.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

이를 귀납가정induction hypothesis이라고 한다. 이제 $n+1$까지의 누적 합이 다음과 같은지 보이면 된다.

$$\text{sigma}(n+1) = \frac{(n+1) \times ((n+1)+1)}{2}$$

이를 귀납가정을 활용하여 다음과 같이 유도하여 확인할 수 있다.

$$\text{sigma}(n+1) = \text{sigma}(n) + (n+1)$$

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0+1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

- 귀납단계 : 임의의 자연수 n에 대해서 n까지의 누적 합은 다음과 같다고 가정하자.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

이를 귀납가정induction hypothesis이라고 한다. 이제 $n+1$까지의 누적 합이 다음과 같은지 보이면 된다.

$$\text{sigma}(n+1) = \frac{(n+1) \times ((n+1)+1)}{2}$$

이를 귀납가정을 활용하여 다음과 같이 유도하여 확인할 수 있다.

$$\begin{aligned}\text{sigma}(n+1) &= \text{sigma}(n) + (n+1) \\ &= \frac{n \times (n+1)}{2} + (n+1) \qquad \text{by induction hypothesis}\end{aligned}$$

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0+1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

- 귀납단계 : 임의의 자연수 n에 대해서 n까지의 누적 합은 다음과 같다고 가정하자.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

이를 귀납가정induction hypothesis이라고 한다. 이제 $n+1$까지의 누적 합이 다음과 같은지 보이면 된다.

$$\text{sigma}(n+1) = \frac{(n+1) \times ((n+1)+1)}{2}$$

이를 귀납가정을 활용하여 다음과 같이 유도하여 확인할 수 있다.

$$\begin{aligned}
\text{sigma}(n+1) &= \text{sigma}(n) + (n+1) \\
&= \frac{n \times (n+1)}{2} + (n+1) \qquad \text{by induction hypothesis} \\
&= \frac{n \times (n+1) + 2 \times (n+1)}{2}
\end{aligned}$$

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0+1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

- 귀납단계 : 임의의 자연수 n에 대해서 n까지의 누적 합은 다음과 같다고 가정하자.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

이를 귀납가정induction hypothesis이라고 한다. 이제 $n+1$까지의 누적 합이 다음과 같은지 보이면 된다.

$$\text{sigma}(n+1) = \frac{(n+1) \times ((n+1)+1)}{2}$$

이를 귀납가정을 활용하여 다음과 같이 유도하여 확인할 수 있다.

$$\begin{aligned}
\text{sigma}(n+1) &= \text{sigma}(n) + (n+1) \\
&= \frac{n \times (n+1)}{2} + (n+1) \qquad \text{by induction hypothesis} \\
&= \frac{n \times (n+1) + 2 \times (n+1)}{2} \\
&= \frac{(n+1) \times (n+2)}{2}
\end{aligned}$$

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0+1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

- 귀납단계 : 임의의 자연수 n에 대해서 n까지의 누적 합은 다음과 같다고 가정하자.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

이를 귀납가정induction hypothesis이라고 한다. 이제 $n+1$까지의 누적 합이 다음과 같은지 보이면 된다.

$$\text{sigma}(n+1) = \frac{(n+1) \times ((n+1)+1)}{2}$$

이를 귀납가정을 활용하여 다음과 같이 유도하여 확인할 수 있다.

$$\begin{aligned}
\text{sigma}(n+1) &= \text{sigma}(n) + (n+1) \\
&= \frac{n \times (n+1)}{2} + (n+1) \qquad \text{by induction hypothesis} \\
&= \frac{n \times (n+1) + 2 \times (n+1)}{2} \\
&= \frac{(n+1) \times (n+2)}{2} \\
&= \frac{(n+1) \times ((n+1)+1)}{2}
\end{aligned}$$

가우스 알고리즘의 검증

정리 4.1.1 임의의 자연수 n까지의 누적 합 sigma(n)은 다음 식으로 계산한다.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

증명(수학적 귀납법)

- 기초단계 : 0까지의 누적 합은 0이어야 한다. 그런데 0을 대입해보면 다음과 같이 0이다.

$$\text{sigma}(0) = \frac{0 \times (0+1)}{2} = 0$$

따라서 n이 0일 때 위 명제는 참이다.

- 귀납단계 : 임의의 자연수 n에 대해서 n까지의 누적 합은 다음과 같다고 가정하자.

$$\text{sigma}(n) = \frac{n \times (n+1)}{2}$$

이를 귀납가정induction hypothesis이라고 한다. 이제 $n+1$까지의 누적 합이 다음과 같은지 보이면 된다.

$$\text{sigma}(n+1) = \frac{(n+1) \times ((n+1)+1)}{2}$$

이를 귀납가정을 활용하여 다음과 같이 유도하여 확인할 수 있다.

$$\begin{aligned}
\text{sigma}(n+1) &= \text{sigma}(n) + (n+1) \\
&= \frac{n \times (n+1)}{2} + (n+1) \qquad \text{by induction hypothesis} \\
&= \frac{n \times (n+1) + 2 \times (n+1)}{2} \\
&= \frac{(n+1) \times (n+2)}{2} \\
&= \frac{(n+1) \times ((n+1)+1)}{2}
\end{aligned}$$

기초 단계와 귀납 단계가 모두 참이므로, 수학적 귀납법에 의해서 이 명제는 참이다.

실습 4.1 구간 수열의 합

자연수 m과 n을 인수로 받아 m부터 n까지의 누적 합 sumrange(m, n)은 다음과 같이 정의한다.

$$\text{sumrange}(m, n) = m + (m + 1) + \cdots + n$$

$m > n$인 경우의 합은 0으로 하기로 하고, 재귀식으로 작성하면 다음과 같다.

$$\text{sumrange}(m, n) = \begin{cases} m + \text{sumrange}(m + 1, n) & \text{if } m \leq n \\ 0 & \text{if } m > n \end{cases}$$

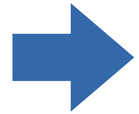
code : 4-10.py, 4-11.py, 4-12.py
일반재귀, 꼬리재귀, while루프

CHAPTER 4

재귀와 반복 : 자연수 계산

거듭제곱 b^n 계산하기

```
b**n
```



```
pow(b,n)
```

거듭제곱 b^n 계산하기

b**n

pow(b,n)

b의 n 거듭제곱인 b^n을 계산하여 내주는 함수
power(b,n)을 별도로 만들어보자.
b는 정수, n은 자연수로 제한하고,
음수 인수는 모두 0으로 취급하기로 한다.

순진무구 알고리즘

$$b^n = \begin{cases} b \times b^{n-1} & \text{if } n > 0 \\ 1 & \text{if } n \leq 0 \end{cases}$$

순진무구 알고리즘

$$b^n = \begin{cases} b \times b^{n-1} & \text{if } n > 0 \\ 1 & \text{if } n \leq 0 \end{cases}$$

code : 4-13.py

```
1  def power(b,n):
2      if n > 0:
3          return b * power(b,n-1)
4      else:
5          return 1
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=> 2 * 2 * 2 * 2 * 2 * if 0 > 0: return 2 * power(2,0-1) else: return 1
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=> 2 * 2 * 2 * 2 * 2 * if 0 > 0: return 2 * power(2,0-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * 1
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=> 2 * 2 * 2 * 2 * 2 * if 0 > 0: return 2 * power(2,0-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * 1
=> 2 * 2 * 2 * 2 * 2
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=> 2 * 2 * 2 * 2 * 2 * if 0 > 0: return 2 * power(2,0-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * 1
=> 2 * 2 * 2 * 2 * 2
=> 2 * 2 * 2 * 4
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=> 2 * 2 * 2 * 2 * 2 * if 0 > 0: return 2 * power(2,0-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * 1
=> 2 * 2 * 2 * 2 * 2
=> 2 * 2 * 2 * 4
=> 2 * 2 * 8
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=> 2 * 2 * 2 * 2 * 2 * if 0 > 0: return 2 * power(2,0-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * 1
=> 2 * 2 * 2 * 2 * 2
=> 2 * 2 * 2 * 4
=> 2 * 2 * 8
=> 2 * 16
=>
```

```
def power(b,n):
    if n > 0:
        return b * power(b,n-1)
    else:
        return 1
```

계산 비용 분석

- 시간
- 공간

```
   power(2,5)
=> if 5 > 0: return 2 * power(2,5-1) else: return 1
=> 2 * power(2,4)
=> 2 * if 4 > 0: return 2 * power(2,4-1) else: return 1
=> 2 * 2 * power(2,3)
=> 2 * 2 * if 3 > 0: return 2 * power(2,3-1) else: return 1
=> 2 * 2 * 2 * power(2,2)
=> 2 * 2 * 2 * if 2 > 0: return 2 * power(2,2-1) else: return 1
=> 2 * 2 * 2 * 2 * power(2,1)
=> 2 * 2 * 2 * 2 * if 1 > 0: return 2 * power(2,1-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * power(2,0)
=> 2 * 2 * 2 * 2 * 2 * if 0 > 0: return 2 * power(2,0-1) else: return 1
=> 2 * 2 * 2 * 2 * 2 * 1
=> 2 * 2 * 2 * 2 * 2
=> 2 * 2 * 2 * 4
=> 2 * 2 * 8
=> 2 * 16
=> 32
```

재귀 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **power 함수의 계산 복잡도**
  - 시간
    - 곱셈의 횟수 (= 재귀 함수를 호출하는 횟수)에 비례
    - 인수가 n 일 때 곱셈을 총 n번(= 재귀 호출을 총 n번) 하므로 계산시간은 n에 비례
  - 공간
    - 재귀 함수를 호출하는 횟수에 비례 (답을 구해온 뒤에 곱해야 할 수를 기억해둘 공간 필요)
    - 인수가 n 일 때 재귀 호출을 총 n번 하므로 필요 공간은 n에 비례

재귀

code : 4-13.py

```
1  def power(b,n):
2      if n > 0:
3          return b * power(b,n-1)
4      else:
5          return 1
```

재귀

code : 4-13.py

```
1  def power(b,n):
2      if n > 0:
3          return b * power(b,n-1)
4      else:
5          return 1
```

꼬리 재귀

code : 4-14.py

```
1  def power(b,n):
2      def loop(b,n,prod):
3          if n > 0:
4              return loop(b,n-1,b*prod)
5          else:
6              return prod
7      return loop(b,n,1)
```

재귀

code : 4-13.py

```
1  def power(b,n):
2      if n > 0:
3          return b * power(b,n-1)
4      else:
5          return 1
```

꼬리 재귀

code : 4-15.py

```
1  def power(b,n):
2      def loop(n,prod):
3          if n > 0:
4              return loop(n-1,b*prod)
5          else:
6              return prod
7      return loop(n,1)
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=> loop(2,8)
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=> loop(2,8)
=> if 2 > 0: return loop(2-1,2*8) else: return 8
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=> loop(2,8)
=> if 2 > 0: return loop(2-1,2*8) else: return 8
=> loop(1,16)
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=> loop(2,8)
=> if 2 > 0: return loop(2-1,2*8) else: return 8
=> loop(1,16)
=> if 1 > 0: return loop(1-1,2*16) else: return 16
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=> loop(2,8)
=> if 2 > 0: return loop(2-1,2*8) else: return 8
=> loop(1,16)
=> if 1 > 0: return loop(1-1,2*16) else: return 16
=> loop(0,32)
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=> loop(2,8)
=> if 2 > 0: return loop(2-1,2*8) else: return 8
=> loop(1,16)
=> if 1 > 0: return loop(1-1,2*16) else: return 16
=> loop(0,32)
=> if 0 > 0: return loop(0-1,2*32) else: return 32
=>
```

```
def power(b,n):
    def loop(n,prod):
        if n > 0:
            return loop(n-1,b*prod)
        else:
            return prod
    return loop(n,1)
```

```
   power(2,5)
=> loop(5,1)
=> if 5 > 0: return loop(5-1,2*1) else: return 1
=> loop(4,2)
=> if 4 > 0: return loop(4-1,2*2) else: return 2
=> loop(3,4)
=> if 3 > 0: return loop(3-1,2*4) else: return 4
=> loop(2,8)
=> if 2 > 0: return loop(2-1,2*8) else: return 8
=> loop(1,16)
=> if 1 > 0: return loop(1-1,2*16) else: return 16
=> loop(0,32)
=> if 0 > 0: return loop(0-1,2*32) else: return 32
=> 32
```

계산 비용 분석

- 시간
- 공간

꼬리재귀 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **꼬리재귀 power 함수의 계산 복잡도**
  - 시간
    - 곱셈의 횟수 (= 재귀 함수를 호출하는 횟수)에 비례
    - 인수가 n 일 때 곱셈을 총 n번(= 재귀 호출을 총 n번) 하므로 계산시간은 n에 비례
  - 공간
    - 인수의 크기와 상관없이 일정

꼬리 재귀

code : 4-15.py

```
1  def power(b,n):
2      def loop(n,prod):
3          if n > 0:
4              return loop(n-1,b*prod)
5          else:
6              return prod
7      return loop(n,1)
```

while 루프

code : 4-16.py

```
1  def power(b,n):
2      prod = 1
3      while n > 0:
4          prod = b * prod
5          n = n - 1
6      return prod
```

계산 비용 분석

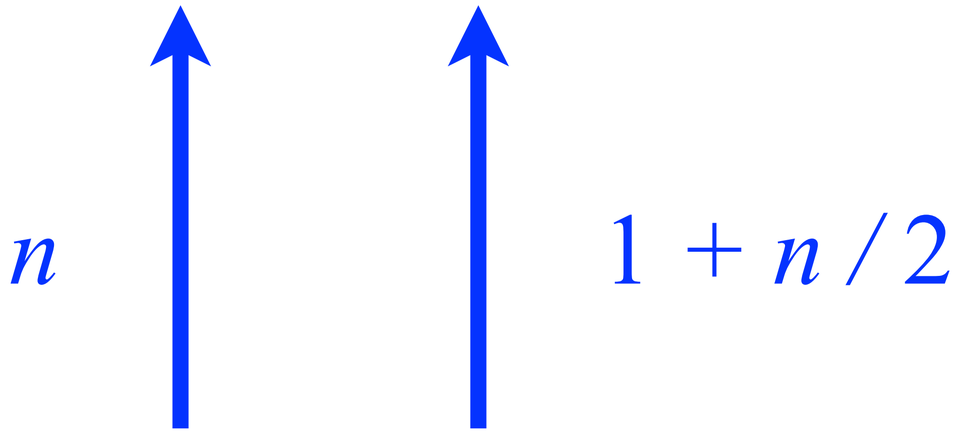
- 시간
- 공간

while 루프 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **while 루프로 작성한 power의 계산 복잡도**
  - 시간
    - 곱셈의 횟수 (= 루프를 반복하는 횟수)에 비례
    - 인수가 n 일 때 곱셈을 총 n번(= 루프 반복을 총 n번) 하므로 계산시간은 n에 비례
  - 공간
    - 인수의 크기와 상관없이 일정

계산 시간 비용 절약?

$$b^n = (b^2)^{\frac{n}{2}} \qquad \text{if } n > 0 \; and \; even(n)$$



나눠 풀기 알고리즘

Divide-and-Conquer

$$b^n = \begin{cases} (b \times b)^{\frac{n}{2}} & \text{if } n > 0 \text{ and } n \bmod 2 = 0 \\ b \times b^{n-1} & \text{if } n > 0 \text{ and } n \bmod 2 \neq 0 \\ 1 & \text{if } n \leq 0 \end{cases}$$

나눠 풀기 알고리즘

Divide-and-Conquer

$$b^n = \begin{cases} (b \times b)^{\frac{n}{2}} & \text{if } n > 0 \text{ and } n \bmod 2 = 0 \\ b \times b^{n-1} & \text{if } n > 0 \text{ and } n \bmod 2 \neq 0 \\ 1 & \text{if } n \leq 0 \end{cases}$$

code : 4-17.py

```
1  def power(b,n):
2      if n > 0:
3          if n % 2 == 0:
4              return power(b*b,n//2)
5          else:
6              return b * power(b,n-1)
7      else:
8          return 1
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=> 2 * 4 * power(4*4,2//2)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=> 2 * 4 * power(4*4,2//2)
=> 2 * 4 * power(16,1)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=> 2 * 4 * power(4*4,2//2)
=> 2 * 4 * power(16,1)
=> 2 * 4 * 16 * power(16,0)
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=> 2 * 4 * power(4*4,2//2)
=> 2 * 4 * power(16,1)
=> 2 * 4 * 16 * power(16,0)
=> 2 * 4 * 16 * 1
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=> 2 * 4 * power(4*4,2//2)
=> 2 * 4 * power(16,1)
=> 2 * 4 * 16 * power(16,0)
=> 2 * 4 * 16 * 1
=> 2 * 4 * 16
=>
```

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=> 2 * 4 * power(4*4,2//2)
=> 2 * 4 * power(16,1)
=> 2 * 4 * 16 * power(16,0)
=> 2 * 4 * 16 * 1
=> 2 * 4 * 16
=> 2 * 64
=>
```

```python
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

```
   power(2,7)
=> 2 * power(2,6)
=> 2 * power(2*2,6//2)
=> 2 * power(4,3)
=> 2 * 4 * power(4,2)
=> 2 * 4 * power(4*4,2//2)
=> 2 * 4 * power(16,1)
=> 2 * 4 * 16 * power(16,0)
=> 2 * 4 * 16 * 1
=> 2 * 4 * 16
=> 2 * 64
=> 128
```

계산 비용 분석

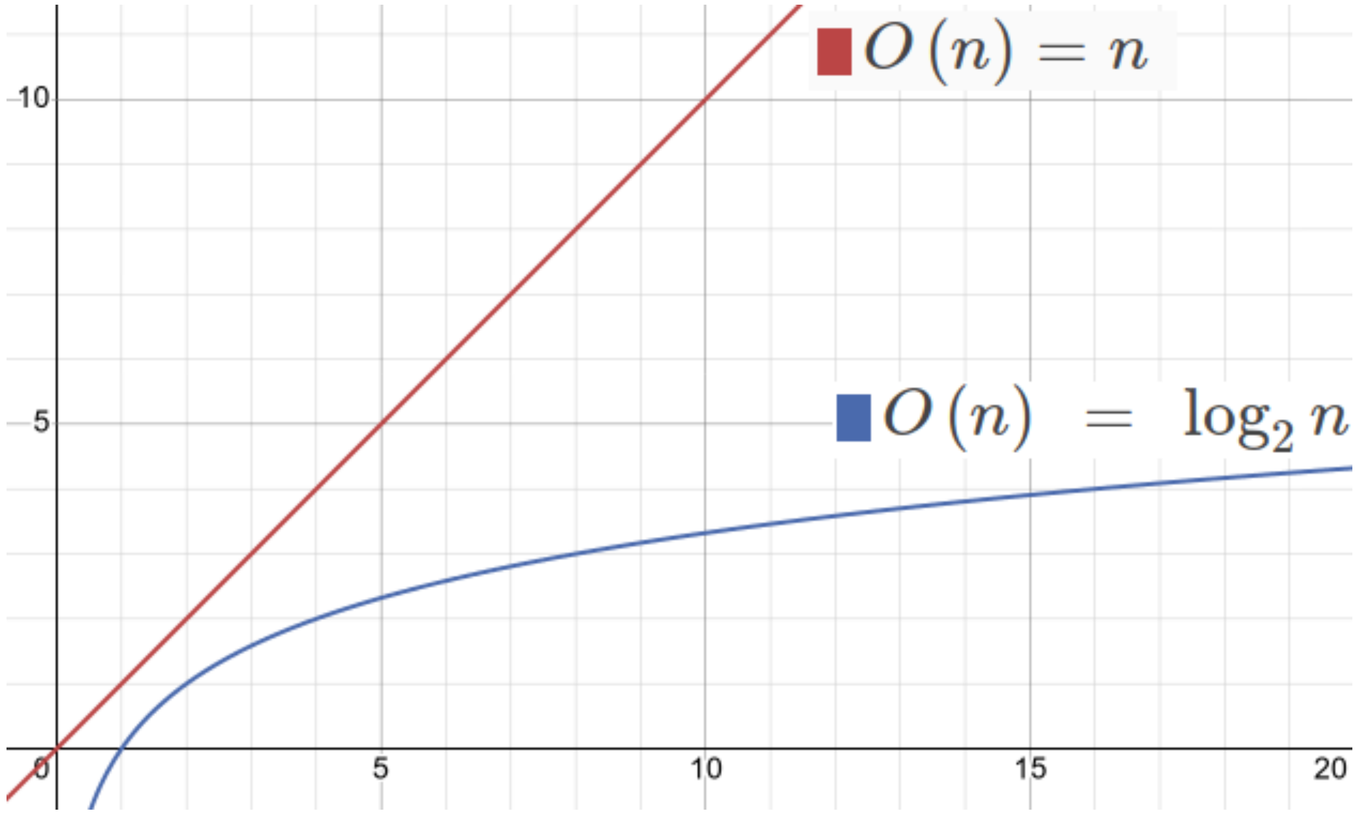
- 시간
- 공간

재귀 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **power 함수의 계산 복잡도**
  - 시간
    - 곱셈의 횟수 (= 재귀 함수를 호출하는 횟수)에 비례
    - 둘째 인수 n이 짝수 일 때 인수의 크기가 반으로 작아지므로 호출 횟수를 대략 따져보면 약 $\log_2 n$ 번이 되므로 계산시간은 $\log_2 n$에 비례함
  - 공간
    - 재귀 함수를 호출하는 횟수에 비례 (답을 구해온 뒤에 곱해야 할 수를 기억해둘 공간 필요)
    - 둘째 인수 n이 짝수 일 때 위와 마찬가지로 재귀 호출을 약 $\log_2 n$ 번 하므로 계산시간은 $\log_2 n$에 비례함

n	$\log_2 n$
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16

n vs. $\log_2 n$



power(2, 66506)

66506

33253 33252

16626

8313 8312

4156

2078

1039 1038

519 518

259 258

129 128

64

32

16

8

4

2

1 0

재귀

code : 4-17.py

```
def power(b,n):
    if n > 0:
        if n % 2 == 0:
            return power(b*b,n//2)
        else:
            return b * power(b,n-1)
    else:
        return 1
```

꼬리 재귀

code : 4-18.py

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=>
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=>
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) ==
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=>
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) ==
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=>
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=> loop(4,3-1,4*2) ==
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=> loop(4,3-1,4*2) == loop(4,2,8)
=>
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=> loop(4,3-1,4*2) == loop(4,2,8)
=> loop(4*4,2//2,8) ==
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=> loop(4,3-1,4*2) == loop(4,2,8)
=> loop(4*4,2//2,8) == loop(16,1,8)
=>
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=> loop(4,3-1,4*2) == loop(4,2,8)
=> loop(4*4,2//2,8) == loop(16,1,8)
=> loop(16,1-1,16*8) ==
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=> loop(4,3-1,4*2) == loop(4,2,8)
=> loop(4*4,2//2,8) == loop(16,1,8)
=> loop(16,1-1,16*8) == loop(16,0,128)
=>
```

```
def power(b,n):
    def loop(b,n,prod):
        if n > 0:
            if n % 2 == 0:
                return loop(b*b,n//2,prod)
            else:
                return loop(b,n-1,b*prod)
        else:
            return prod
    return loop(b,n,1)
```

```
   power(2,7)
=> loop(2,7,1)
=> loop(2,7-1,2*1) == loop(2,6,2)
=> loop(2*2,6//2,2) == loop(4,3,2)
=> loop(4,3-1,4*2) == loop(4,2,8)
=> loop(4*4,2//2,8) == loop(16,1,8)
=> loop(16,1-1,16*8) == loop(16,0,128)
=> 128
```

계산 비용 분석

- 시간
- 공간

꼬리재귀 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **꼬리재귀 power 함수의 계산 복잡도**
  - 시간
    - 곱셈의 횟수 (= 재귀 함수를 호출하는 횟수)에 비례
    - 둘째 인수 n이 짝수 일 때 인수의 크기가 반으로 작아지므로 호출 횟수를 대략 따져보면 약 $\log_2 n$ 번이 되므로 계산시간은 $\log_2 n$에 비례함
  - 공간
    - 인수의 크기와 상관없이 일정

꼬리 재귀

code : 4-18.py

```
1  def power(b,n):
2      def loop(b,n,prod):
3          if n > 0:
4              if n % 2 == 0:
5                  return loop(b*b,n//2,prod)
6              else:
7                  return loop(b,n-1,b*prod)
8          else:
9              return prod
10     return loop(b,n,1)
```

while 루프

code : 4-19.py

```
1  def power(b,n):
2      prod = 1
3      while n > 0:
4          if n % 2 == 0:
5              b = b * b
6              n = n // 2
7          else:
8              n = n - 1
9              prod = b * prod
10     return prod
```

계산 비용 분석

- 시간
- 공간

while 루프 함수의 계산 비용 분석

- **계산복잡도**computational complexity
  - 시간 : 프로그램이 얼마나 빨리 답을 계산하는가?
  - 공간 : 답을 계산하면서 얼마나 많은 공간을 사용하는가?
- **while 루프로 작성한 power의 계산 복잡도**
  - 시간
    - 곱셈의 횟수 (= 루프를 반복하는 횟수)에 비례
    - 둘째 인수 n이 짝수 일 때 인수의 크기가 반으로 작아지므로 계산시간은 $\log_2 n$에 비례함
  - 공간
    - 인수의 크기와 상관없이 일정

code : 4-19.py

```
1  def power(b,n):
2      prod = 1
3      while n > 0:
4          if n % 2 == 0:
5              b = b * b
6              n = n // 2
7          else:
8              n = n - 1
9              prod = b * prod
10     return prod
```

CHAPTER 4

재귀와 반복 : 자연수 계산

약수
Divisor

정수의 약수는
나누어서
나머지 없이 떨어지는
양수를 말한다.

예를 들어,
42의 약수는
42, 21, 14, 7, 6, 3, 2, 1

공약수
Common Divisor

두 정수의 공약수는
두 정수의 약수 중에서
공통되는 수를 말한다.

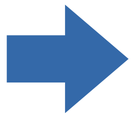
예를 들어,
54의 약수는 54, 27, 18, 9, 6, 3, 2, 1
24의 약수는 24, 12, 8, 6, 4, 3, 2, 1
54와 24의 공약수는
6, 3, 2, 1

최대공약수
Greatest Common Divisor

공약수들 중에서
가장 큰 수를
최대공약수라고 한다.

예를 들어,
54와 24의 공약수
6, 3, 2, 1
중에서
최대공약수는 6이다.

```
from math import gcd
```



최대공약수 계산 함수

두 자연수 `m`과 `n`을 인수로 받아
두 자연수의 최대공약수를 계산하는
함수 `gcd`을 만들자.

유클리드 알고리즘

Euclid

- 나눗셈을 이용한 알고리즘
- 두 수의 최대공약수는 두 수의 차이로도 나누어지는 성질을 이용

$$\gcd(m, n) = \begin{cases} \gcd(n, m \bmod n) & \text{if } n \neq 0 \\ m & \text{if } n = 0 \end{cases}$$

유클리드 알고리즘

Euclid

- 나눗셈을 이용한 알고리즘
- 두 수의 최대공약수는 두 수의 차이로도 나누어지는 성질을 이용

$$\gcd(m, n) = \begin{cases} \gcd(n, m \bmod n) & \text{if } n \neq 0 \\ m & \text{if } n = 0 \end{cases}$$

code : 4-20.py

```
1  def gcd(m,n):
2      if n != 0:
3          return gcd(n,m%n)
4      else:
5          return m
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=>
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) ==
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=>
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=> gcd(18,48%18) ==
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=> gcd(18,48%18) == gcd(18,12)
=>
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=> gcd(18,48%18) == gcd(18,12)
=> gcd(12,18%12) ==
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=> gcd(18,48%18) == gcd(18,12)
=> gcd(12,18%12) == gcd(12,6)
=>
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=> gcd(18,48%18) == gcd(18,12)
=> gcd(12,18%12) == gcd(12,6)
=> gcd(6,12%6) ==
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=> gcd(18,48%18) == gcd(18,12)
=> gcd(12,18%12) == gcd(12,6)
=> gcd(6,12%6) == gcd(6,0)
=>
```

유클리드 알고리즘

Euclid

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

```
   gcd(18,48)
=> gcd(48,18%48) == gcd(48,18)
=> gcd(18,48%18) == gcd(18,12)
=> gcd(12,18%12) == gcd(12,6)
=> gcd(6,12%6) == gcd(6,0)
=> 6
```

```
def gcd(m,n):
    if n != 0:
        return gcd(n,m%n)
    else:
        return m
```

실습 4.2 최대공약수 함수 완성

이 꼬리재귀 함수를 while 루프를 사용하는 함수로 변환해 보자.

code : 4-21.py

```
1  def gcd(m,n):
2      while n != 0:
3          pass # Fill your code in this space.
4
5      return m
```

나눠 풀기 알고리즘

Divide-and-Conquer

- 뺄셈, 2로 곱하기, 2로 나누기 연산만 이용한 알고리즘

$$
\gcd(m, n) = \begin{cases}
2 \times \gcd(\frac{m}{2}, \frac{n}{2}) & \text{if even}(m) \text{ and even}(n) \\
\gcd(\frac{m}{2}, n) & \text{if even}(m) \text{ and odd}(n) \\
\gcd(m, \frac{n}{2}) & \text{if odd}(m) \text{ and even}(n) \\
\gcd(m, \frac{n-m}{2}) & \text{if odd}(m) \text{ and odd}(n) \text{ and } m \leq n \\
\gcd(n, \frac{m-n}{2}) & \text{if odd}(m) \text{ and odd}(n) \text{ and } m > n \\
n & \text{if } m = 0 \\
m & \text{if } n = 0
\end{cases}
$$

$$\gcd(m, n) = \begin{cases} 2 \times \gcd(\frac{n}{2}, \frac{m}{2}) & \text{if even}(m) \text{ and even}(n) \\ \gcd(\frac{m}{2}, n) & \text{if even}(m) \text{ and odd}(n) \\ \gcd(m, \frac{n}{2}) & \text{if odd}(m) \text{ and even}(n) \\ \gcd(m, \frac{n-m}{2}) & \text{if odd}(m) \text{ and odd}(n) \text{ and } m \leq n \\ \gcd(n, \frac{m-n}{2}) & \text{if odd}(m) \text{ and odd}(n) \text{ and } m > n \\ n & \text{if } m = 0 \\ m & \text{if } n = 0 \end{cases}$$

```
1  def even(n):
2      return n % 2 == 0
3
4  def odd(n):
5      return n % 2 == 1
```

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
==
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
==
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
==
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
==
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
== 2 * gcd(9,3)
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
== 2 * gcd(9,3)
=> 2 * gcd(3,(9-3)//2)
==
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
== 2 * gcd(9,3)
=> 2 * gcd(3,(9-3)//2)
== 2 * gcd(3,3)
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
== 2 * gcd(9,3)
=> 2 * gcd(3,(9-3)//2)
== 2 * gcd(3,3)
=> 2 * gcd(3,(3-3)//2)
==
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
== 2 * gcd(9,3)
=> 2 * gcd(3,(9-3)//2)
== 2 * gcd(3,3)
=> 2 * gcd(3,(3-3)//2)
== 2 * gcd(3,0)
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
== 2 * gcd(9,3)
=> 2 * gcd(3,(9-3)//2)
== 2 * gcd(3,3)
=> 2 * gcd(3,(3-3)//2)
== 2 * gcd(3,0)
=> 2 * 3
=>
```

나눠 풀기 알고리즘

Divide-and-Conquer

code : 4-23.py

```
1  def gcd(m,n):
2      if not (m == 0 or n == 0):
3          if even(m) and even(n):
4              return 2 * gcd(m//2,n//2)
5          elif even(m) and odd(n):
6              return gcd(m//2,n)
7          elif odd(m) and even(n):
8              return gcd(m,n//2)
9          elif m <= n:
10             return gcd(m,(n-m)//2)
11         else:
12             return gcd(n,(m-n)//2)
13     else:
14         if m == 0:
15             return n
16         else:
17             return m
```

```
   gcd(18,48)
=> 2 * gcd(18//2,48//2)
== 2 * gcd(9,24)
=> 2 * gcd(9,24//2)
== 2 * gcd(9,12)
=> 2 * gcd(9,12//2)
== 2 * gcd(9,6)
=> 2 * gcd(9,6//2)
== 2 * gcd(9,3)
=> 2 * gcd(3,(9-3)//2)
== 2 * gcd(3,3)
=> 2 * gcd(3,(3-3)//2)
== 2 * gcd(3,0)
=> 2 * 3
=> 6
```

실습 4.3 나눠 풀기 알고리즘 꼬리재귀 버전

code : 4-24.py

```
1  def gcd(m,n):
2      def loop(m,n,k):
3          if not (m == 0 or n == 0):
4              if even(m) and even(n):
5                  return loop(m//2,n//2,_____)
6              elif even(m) and odd(n):
7                  return loop(m//2,n,k)
8              elif odd(m) and even(n):
9                  return loop(m,n//2,k)
10             elif m <= n:
11                 return loop(m,(n-m)//2,k)
12             else:
13                 return loop(n,(m-n)//2,k)
14         else:
15             if m == 0:
16                 return _______
17             else: # n == 0
18                 return _______
19     return loop(m,n,1)
```

```
   gcd(18,48)
=> loop(18,48,1)
=> loop(18//2,48//2,1*2)
== loop(9,24,2)
=> loop(9,24//2,2)
== loop(9,12,2)
=> loop(9,12//2,2)
== loop(9,6,2)
=> loop(9,6//2,2)
== loop(9,3,2)
=> loop(3,(9-3)//2,2)
== loop(3,3,2)
=> loop(3,(3-3)//2,2)
== loop(3,0,2)
=> 2 * 3
=> 6
```

실습 4.4 나눠 풀기 알고리즘 while 루프 버전

code : 4-25.py

```
1  def gcd(m,n):
2      k = 1
3      while not (m == 0 or n == 0):
4          if even(m) and even(n):
5              m, n, k = _______
6          elif even(m) and odd(n):
7              m = _______
8          elif odd(m) and even(n):
9              n = _______
10         elif m <= n:
11             n = _______
12         else:
13             m, n = _______
14     if m == 0:
15         return _______
16     else: # n == 0
17         return _______
```

CHAPTER 4

재귀와 반복 : 자연수 계산

4.1 자연수 수열의 합

4.2 거듭제곱

4.3 최대공약수

✔ 4.4 곱셈

곱셈

덧셈/뺄셈 알고리즘

$$m \times n = \begin{cases} m + m \times (n - 1) & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

곱셈

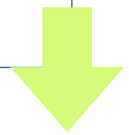
덧셈/뺄셈 알고리즘

$$m \times n = \begin{cases} m + m \times (n-1) & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

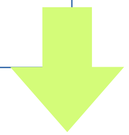
```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
    mult(3,6)
=>
```



```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
    mult(3,6)
=> 3 + mult(3,5)
=>
```



```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=>
```

```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=>
```

```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

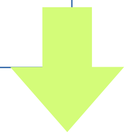
```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=>
```

```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=>
```

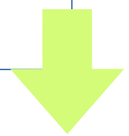
```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=>
```



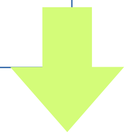
```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=> 3 + 3 + 3 + 3 + 3 + 3 + 0
=>
```



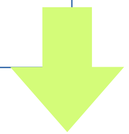
```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=> 3 + 3 + 3 + 3 + 3 + 3 + 0
=> 3 + 3 + 3 + 3 + 3 + 3
=>
```



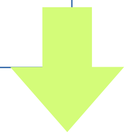
```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=> 3 + 3 + 3 + 3 + 3 + 3 + 0
=> 3 + 3 + 3 + 3 + 3 + 3
=> 3 + 3 + 3 + 3 + 6
=>
```



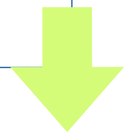
```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=> 3 + 3 + 3 + 3 + 3 + 3 + 0
=> 3 + 3 + 3 + 3 + 3 + 3
=> 3 + 3 + 3 + 3 + 6
=> 3 + 3 + 3 + 9
=>
```



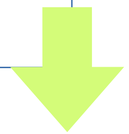
```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=> 3 + 3 + 3 + 3 + 3 + 3 + 0
=> 3 + 3 + 3 + 3 + 3 + 3
=> 3 + 3 + 3 + 3 + 6
=> 3 + 3 + 3 + 9
=> 3 + 3 + 12
=>
```



```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=> 3 + 3 + 3 + 3 + 3 + 3 + 0
=> 3 + 3 + 3 + 3 + 3 + 3
=> 3 + 3 + 3 + 3 + 6
=> 3 + 3 + 3 + 9
=> 3 + 3 + 12
=> 3 + 15
=>
```



```
1  def mult(m,n):
2      if n > 0:
3          return m + mult(m,n-1)
4      else:
5          return 0
```

```
   mult(3,6)
=> 3 + mult(3,5)
=> 3 + 3 + mult(3,4)
=> 3 + 3 + 3 + mult(3,3)
=> 3 + 3 + 3 + 3 + mult(3,2)
=> 3 + 3 + 3 + 3 + 3 + mult(3,1)
=> 3 + 3 + 3 + 3 + 3 + 3 + mult(3,0)
=> 3 + 3 + 3 + 3 + 3 + 3 + 0
=> 3 + 3 + 3 + 3 + 3 + 3
=> 3 + 3 + 3 + 3 + 6
=> 3 + 3 + 3 + 9
=> 3 + 3 + 12
=> 3 + 15
=> 18
```

실습 4.5 덧셈/뺄셈 알고리즘 : 꼬리재귀 함수 버전

code : 4-27.py

```
1  def mult(m,n):
2      def loop(n,ans):
3          if n > 0:
4              return ____
5          else:
6              return ____
7      return loop(n,____)
```

```
    mult(3,6)
=> loop(3,6,0)
=> loop(3,5,3)
=> loop(3,4,6)
=> loop(3,3,9)
=> loop(3,2,12)
=> loop(3,1,15)
=> loop(3,0,18)
=> 18
```

실습 4.6 덧셈/뺄셈 알고리즘 : while 루프 버전

code : 4-28.py

```
1  def mult(m,n):
2      ans = ___
3      while _______ :
4          pass # Fill your code in this space.
5
6
7      return ____
```

곱셈

덧셈/뺄셈/절반 알고리즘

$$m \times n = (m + m) \times (n \div 2) \qquad \text{if} \quad \text{even}(n)$$

곱셈

덧셈/뺄셈/절반 알고리즘

$$m \times n = (m + m) \times (n \div 2) \qquad \text{if } \text{even}(n)$$

$$m \times n = \begin{cases} (m + m) \times (n \div 2) & \text{if } n > 0 \text{ and } \text{even}(n) \\ m + m \times (n - 1) & \text{if } n > 0 \text{ and } \text{odd}(n) \\ 0 & \text{if } n = 0 \end{cases}$$

실습 4.7 덧셈/뺄셈/절반 알고리즘 : 재귀 함수 버전

code : 4-29.py

```
1  def fastmult(m,n):
2      if n > 0:
3          if n % 2 == 0:
4              return ___
5          else:
6              return ___
7      else:
8          return ___
```

```
   fastmult(3,6)
=> fastmult(6,3)
=> 6 + fastmult(6,2)
=> 6 + fastmult(12,1)
=> 6 + 12 + fastmult(12,0)
=> 6 + 12 + 0
=> 6 + 12
=> 18
```

실습 4.8 덧셈/뺄셈/절반 알고리즘 : 꼬리재귀 함수 버전

code : 4-30.py

```
1  def fastmult(m,n):
2      def loop(m,n,ans):
3          if n > 0:
4              pass # Fill your code in this space.
5
6          else:
7              return ___
8      return loop(m,n,__)
```

```
    fastmult(3,6)
=> loop(3,6,0)
=> loop(6,3,0)
=> loop(6,2,6)
=> loop(12,1,6)
=> loop(12,0,18)
=> 18
```

실습 4.9 덧셈/뺄셈/절반 알고리즘 : while 루프 버전

code : 4-31.py

```
1  def fastmult(m,n):
2      ans = ______
3      while n > 0:
4          pass # Fill your code in this space.
5
6
7      return ans
```

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.

57×86

- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57 × 86

57	86

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

57	86
114	43

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

57	86
114	43
228	21

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

57	86
114	43
228	21
456	10

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

57	86
114	43
228	21
456	10
912	5

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

57	86
114	43
228	21
456	10
912	5
1824	2

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

57	86
114	43
228	21
456	10
912	5
1824	2
3648	1

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

~~57~~	~~86~~
114	43
228	21
~~456~~	~~10~~
912	5
~~1824~~	~~2~~
3648	1

곱셈

러시아 농부 알고리즘

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 절반을 하여(나머지는 버림) 다음 줄에 나란히 적는다.
- 이 과정을 둘째 수가 1이 될 때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답이다.

57×86

	~~57~~	~~86~~
	114	43
	228	21
	~~456~~	~~10~~
	912	5
	~~1824~~	~~2~~
+	3648	1
	4902	

실습 4.10 러시아 농부 알고리즘 : 재귀 함수 버전

code : 4-32.py

```
1  def russian_mult(m,n):
2      def loop(m,n):
3          if n > 1:
4              pass # Fill your code in this space.
5
6          else: # n == 1
7              pass # Fill your code in this line.
8      if n > 0:
9          return loop(m,n)
10     else:
11         return 0
```

```
   russian_mult(57,86)
=> loop(57,86)
=> loop(114,43)
=> 114 + loop(228,21)
=> 114 + 228 + loop(456,10)
=> 114 + 228 + loop(912,5)
=> 114 + 228 + 912 + loop(1824,2)
=> 114 + 228 + 912 + loop(3648,1)
=> 114 + 228 + 912 + 3648
=> 114 + 228 + 4560
=> 114 + 4788
=> 4902
```

실습 4.11 러시아 농부 알고리즘 : 꼬리재귀 함수 버전

code : 4-33.py

```
1  def russian_mult(m,n):
2      def loop(m,n,a):
3          if n > 1:
4              pass # Fill your code in this space.
5
6          else: # n == 1
7              pass # Fill your code in this line.
8      if n > 0:
9          return loop(m,n,__)
10     else:
11         return 0
```

```
   russian_mult(57,86)
=> loop(57,86,0)
=> loop(114,43,0)
=> loop(228,21,114)
=> loop(456,10,342)
=> loop(912,5,342)
=> loop(1824,2,1254)
=> loop(3648,1,1254)
=> 1254 + 3648
=> 4902
```

실습 4.12 러시아 농부 알고리즘 : while 루프 버전

code : 4-34.py

```
1  def russian_mult(m,n):
2      if n > 0:
3          pass # Fill your code in this space.
4
5
6
7
8
9      else:
10         return 0
```

>>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>>

프로그래밍의 정석

파이썬

도경구 지음

생능출판

CHAPTER 4

재귀와 반복 : 자연수 계산