

## 프로그래밍의 정석 : 파이썬 (도경구, 생능출판사, 2020)

### 초판 1쇄 (2020년 12월 29일 발행) 오타 목록

- 의미 전달에 큰 영향을 주지 않는 사소한 오타 제외

- 35쪽 가운데

수는 컴퓨터로 표현이 가능한 범위 내에서만 다룰 수 있다. 정수는 정해진 최소 값과 최대 값 사이의 수만 표현할 수 있고, 실수는 부동 소수점 표현 방식에 따른 범위 내에서만 표현할 수 있다. 게다가 실수는 근사치로밖에 표현할 수 없는 경우가 대부분이다. 왜 그런지 뒤에서 자세히 알아본다.

수는 셀 수 있는 만큼만 프로그램으로 처리 가능하다. 정수는 셀 수 있으므로 가용 메모리 한도 안에서 아무리 큰 수라도 파이썬 프로그램으로 모두 처리 가능하지만, 실수는 셀 수 없을 만큼 많으므로 근사치로 처리할 수 밖에 없는 경우가 대부분이다. 왜 그런지 이 장의 뒤에서 자세히 알아본다.

- 84쪽 code : 2-18.py

line 6 : 69 => 67

- 117쪽 code : 3-13.py

code : 3-13.py

```

1 print("Score Average Calculator")
2 number = input("How many classes? ")
3 total = 0
4 count = 0
5 while None: # Write your loop condition
6     pass # Get and accumulate scores
7
8
9 if count > 0:
10     pass # Compute and print the average
11 else:
12     print(count, "Your average score = 0.0")

```

제거

- 131쪽 아래쪽 각주(footnote)

8 전 세계 대부분의 나라에서 사용하는 그레고리력에는 0년이 없다. 기원전 1년의 다음 해는 기원후 1년이 된다. 하지만 날짜와 시간의 데이터 교환을 위한 국제 표준 ISO 8601은 0년, -1년, -2년이 있다. 각각 기원후 1년, 기원후 2년, 기원후 3년에 해당한다. 여기에서는 국제 표준을 따른다고 가정하자.

기원전

- 176쪽 위쪽 실행 추적 사례

$\Rightarrow$  loop(2,7,1)

```

power(2,7)
=> 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

```

- 181쪽 아래쪽

### 4.3.2 나눠 풀기 알고리즘

다음 알고리즘은 뺄셈과 2로 곱하기, 2로 나누기 연산만 이용한 알고리즘으로, 유클리드 알고리즘보다 좀 더 빨리 계산한다고 알려져 있다. 특이한 알고리즘이다.

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

서로 바꾸기

- 184쪽 위쪽 코드 code : 4-24.py

```

line 14 : elif => else
line 17 : elif => else

```

• 205쪽 위쪽 식 수정

$$(1 + 2 + \dots + \frac{n}{2}) + (\frac{n}{2} + 1 + \frac{n}{2} + 2 + \dots + \frac{n}{2}) + \frac{n}{2}$$

괄호 이동

후반부에  $\frac{n}{2}$ 이  $\frac{n}{2}$ 개 있으므로 따로 뒤로 모아서 다음과 같이 변형해도 결과는 같다.

$$(1 + 2 + \dots + \frac{n}{2}) + (1 + 2 + \dots + \frac{n}{2}) + \frac{n}{2} \times \frac{n}{2}$$

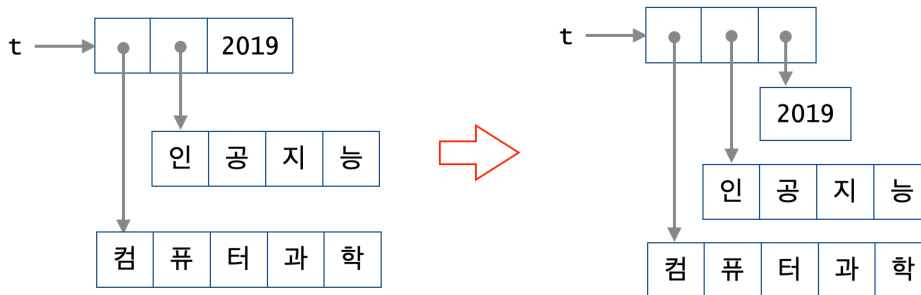
괄호 이동

그런데 첫째 항과 둘째 항이 같으므로 다음과 같이 다시 쓸 수 있다.

$$2 \times (1 + 2 + \dots + \frac{n}{2}) + \frac{n}{2} \times \frac{n}{2}$$

괄호 이동

• 216쪽 위쪽 그림 수정



• 225쪽 위쪽 알고리즘 수정

리스트 xs를 정렬하려면,

**(반복조건)** xs != []

- xs에서 가장 작은 원소를 찾아서 smallest로 지정하고,
- xs에서 smallest를 제거하고,
- xs를 재귀로 정렬하고, → smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
- smallest와 정렬된 xs를 나란히 붙여서 리턴한다.

**(종료조건)** xs == []

- 정렬할 원소가 없으므로 []를 그대로 리턴한다.

• 244쪽 위쪽 알고리즘

리스트  $xs$ 를 퀵정렬하려면,

(반복조건)  $len(xs) > 1$

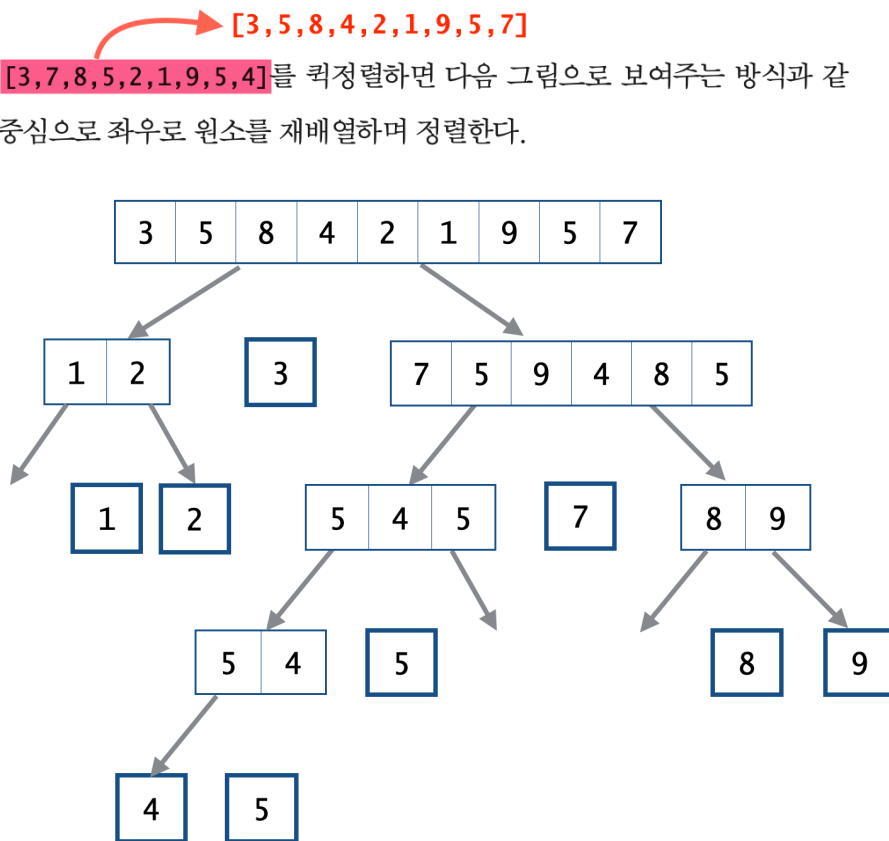
- 기준으로 사용할 피벗 원소  $pivot$ 을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다.<sup>12</sup>
- $pivot$ 을 기준으로 작은 원소는 왼쪽 리스트  $ls$ 로, 큰 원소는 오른쪽 리스트  $rs$ 로 옮긴다.
- 왼쪽 리스트  $ls$ 와 오른쪽 리스트  $rs$ 를 각각 재귀로 정렬하고,  $ls$ 와  $pivot$ 과  $rs$ 를 나란히 붙여서 리턴한다.

(종료조건)  $len(xs) \leq 1$

- 정렬할 필요가 없으므로 그대로 리턴한다.

• 244쪽 아래쪽 그림 대체

예를 들어 **[3, 7, 8, 5, 2, 1, 9, 5, 4]**를 퀵정렬하면 다음 그림으로 보여주는 방식과 같이 피벗을 중심으로 좌우로 원소를 재배열하며 정렬한다.



- 247쪽 위쪽 실행 추적  
변수이름 변경 : `ls => left, rs => right` (각각 5곳씩)

```

partition(5, [7, 2, 1, 9, 4])
=> left, right = partition(5, [2, 1, 9, 4])
    => left, right = partition(5, [1, 9, 4])
        => left, right = partition(5, [9, 4])
            => left, right = partition(5, [4])
                => left, right = partition(5, [])
                    => [], []
                => [4], []
            => [4], [9]
        => [4, 1], [9]
    => [4, 1, 2], [9]
=> [4, 1, 2], [9, 7]

```

- 247쪽 아래쪽 5-30.py 코드  
변수이름 변경 : `ls => left, rs => right` (각각 2곳씩)
- 248쪽 위쪽 5-31.py 코드  
변수이름 변경 : `ls => left, rs => right` (각각 2곳씩)
- 248쪽 아래쪽 5-32.py 코드  
변수이름 변경 : `ls => left, rs => right` (각각 2곳씩)

- 278쪽 아래쪽 수정

## 6.2 이분검색

### 6.2.1 존재 유무 확인

순차검색 알고리즘으로 리스트 `s`를 검색하는 경우, 키 `x`를 찾는 순간 바로 검색을 종료할 수 있다. 하지만 키가 맨 뒤에 있거나 아예 없는 경우에는 리스트 전체를 모두 검사해야 최종 판단을 할 수 있다. 그런데 리스트가 정렬되어 있으면, 리스트 전체를 다 검사해보지 않고도 키의 존재유무를 판단할 수 있다. 정렬된 리스트 `ss`를 반으로 나누어 가운데 원소와 키 `x`를 비교하면, 좌우 반쪽 중 어느 쪽에 검색 키가 있을지 판단할 수 있다. 리스트의 구조를 기반으로 반복조건과 종료조건으로 나누어 알고리즘을 기술하면 다음과 같다.

→ 에 원소가 있는지 여부에 따라

- 284쪽 code : 6-12.py

```

line 1 : key => x
line 5 : key => x
line 8 : key => x

```

- 285쪽 아래쪽 수정

### 6.3.2 성능 테스트

순차검색 함수와 이분검색 함수의 성능을 실행 시간을 측정하여 비교해 보자. 두 함수의 실행 속도 차이를 느껴보려면 검색 대상의 규모가 커야 한다. 임의의 대규모 리스트를 수동으로 만들기는 성가신 일이니, **자동으로** 실행 시간을 측정하는 테스트 코드를 만들어보자. 테스트 코드를 작성하는데 필요한 검색 대상 리스트를 무작위로 만드는 방법, 검색 키를 무작위로 선택하는 방법, 함수 호출의 실행 시간을 재는 방법을 먼저 차례로 살펴본 다음, 테스트 코드를 작성한다. **검색 대상 리스트를 자동으로 만들어서**

- 288쪽 code : 6-14.py : 라인 4의 문자열 일부 제거 (의미 명확성 향상)

code : 6-14.py

```

1 from random import sample, randrange
2 from time import perf_counter
3
4 print("Preparing data for seq_search. Please, wait a moment ...")
5 data = sample(range(10000000), 8000000)
6

```

제거

- 306쪽 연습 문제 6.1 - code 6-20.py

라인 10: = 를 == 로 수정

- 318쪽 위쪽 6-34.py

code : 6-34.py

```

1 def longest_streak1(s):
2     pass # write your code here.
3

```

2

- 326쪽 본문 아래쪽 (의미 명확성 향상)

는 상황식 문제풀이 기법인 표채워풀기<sup>13</sup>를 피보나치 수열, 조합, 1까지 줄이는 최소 스텝 문제를 풀면서 공부한다. 이어서 인수의 크기에 비하여 **호출** 횟수가 기하급수적으로 증가하는데 중복 호출이 전혀 없는 대표적인 사례인 하노이의 탑 문제도 살펴본다. 이 문제를 푸는 함수는 재귀로 쉽게 만들 수 있지만, 계산량이 너무 많아 실용적으로 사용할 수 없다. 개선의 여지가 전혀 없다고 정식으로 증명된 바는 없지만, 개선이 불가능하리라 것이 학계의 정설이다.

<sup>13</sup> 동적계획법(dynamic programming)으로 널리 알려져있지만, 용어의 순화 차원에서 의미를 정확하게 표현하는 새로운 용어를 제안한다.

- 332쪽 code : 7-2.py

line 7: 4 => 5

- 341쪽 가운데 7.2절 조합

첫 줄 :  $n$ 개의 자연수에서 => 서로 다른  $n$ 개의 원소에서

- 361쪽 아래쪽

이동 횟수는 원반이 하나 늘어날 때마다 약 두 배씩 증가하고, 계산시간도 마찬가지로 약 두 배씩 증가함을 알 수 있다. 정확히 따지면, **디스크**의 개수가  $n$ 이면, 원반을  $2^{n-1}$  회 이동해야 한다.

여기서 구한 이동 횟수 미만으로 **원반**을 이동하여 이 문제를 풀 수 있는 방법은 아직 알려진 바 없다. 중복 호출이 전혀 없으니 어찌 보면 당연하다.

- 383쪽 아래쪽 (의미 명확성 향상)

### 스도쿠 정답보드 만들기

위의 �도쿠 **보드**는 첫 가로줄을 [1, 2, 3, 4]로 고정한 다음 나머지 가로줄을 각각 다음과 같은 요령으로 배치하여 **정답보드** 하나를 만들 수 있다.

```
row0 = [1, 2, 3, 4]
row1 = row0[2:4] + row0[0:2]
row2 = [row0[1], row0[0], row0[3], row0[2]]
row3 = row2[2:4] + row2[0:2]
board = [row0, row1, row2, row3]
```

- 388쪽 code : 8-9.py : 라인 2 제거, 라인 4,5 한 줄로 수정

code : 8-9.py

```

1 def transpose(board):
2   transposed = [] ← 제거
3   size = len(board)
4   for _ in range(size):
5       transposed.append([])
6   for row in board:
7       for i in range(size):
8           pass # Write your code here.
9   return transposed

```

`transposed = [[] for _ in range(size)]`

- 392쪽 중단 실행 예시

```

>>> solution_board = [[1, 2, 3, 4], [3, 4, 1, 2], [2, 1, 4, 3], [4, 3,
2, 1]]
>>> puzzle_board = solution_board[:3] ← 제거
>>> puzzle_board
[[1, 2, 3, 4], [3, 4, 1, 2], [2, 1, 4, 3], [4, 3, 2, 1]]
>>> puzzle_board[2][3] = 0
>>> puzzle_board
[[1, 2, 3, 4], [3, 4, 1, 2], [2, 1, 4, 0], [4, 3, 2, 1]]
>>> solution_board
[[1, 2, 3, 4], [3, 4, 1, 2], [2, 1, 4, 0], [4, 3, 2, 1]]

```

- 419쪽 마지막 문장  
구성원소는 임의의 식이 될 수 있다.  
=> 구성원소 부분은 임의의 식을 넣을 수 있으나, 계산 결과가 수정가능한 값이 되는 식은 허용하지 않는다.
- 425쪽 code : 9-1.py 에서 라인 10 주석  
deak => deck
- 434쪽 블랙잭 알고리즘의 4.  
단계 5~14을 => 단계 5~14를
- 454쪽 블랙잭 알고리즘(확장)의 17.  
칩 최대 보유 멤버 5명까지 보여준다. => 칩 최대 보유 멤버를 5명까지 보여준다.



• 461쪽 표 10.1

표 10.1 대표적인 내장 예외

예외 타입	발생 상황
<del>SyntaxError</del>	<del>문법이 틀린 경우</del>
TypeError	피연산자 또는 함수 인수의 타입이 틀린 경우
ValueError	피연산자 또는 함수 인수의 값이 틀린 경우
NameError	지정한 적이 없는 모르는 이름이 나타난 경우
IndexError	없는 인덱스를 사용한 경우
KeyError	없는 키를 사용한 경우
ZeroDivisionError	0으로 나누려 하는 경우
<b>IOError</b>	없는 파일을 열려고 하는 경우, 열지 않고 파일을 읽거나 쓰려고 하는 경우 등

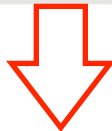
기타 내장 예외 목록<sup>23</sup>은 파이썬 표준 라이브러리 문서에서 열람할 수 있다.

→ **FileNotFoundError**

• 475쪽 셋째 아이템

- 키보드 입력이 자연수가 아닌 경우 assert 문을 사용하여 AssertionError 예외를 발생시키고, 이를 처리하여 다음과 같이 메시지를 실행창에 출력하며 재입력 받는다.

Must be a natural number.

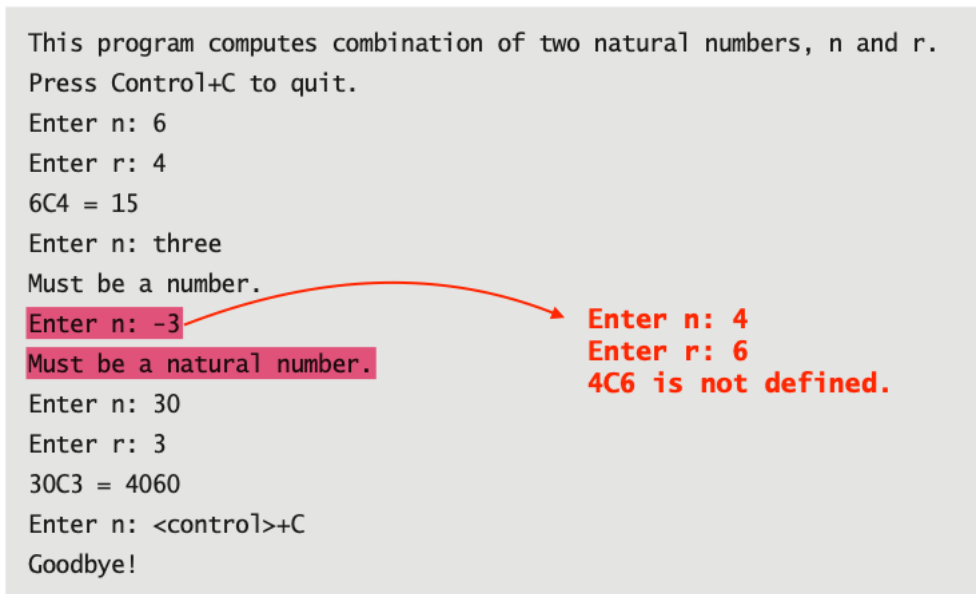


- n 값과 r 값이 0 이상이 아니거나, r 값이 n 값보다 큰 경우 조합을 정의할 수 없다. 이 경우 assert 문을 활용하여 AssertionError 예외를 발생시키고, 이를 처리하여 다음과 같은 형식으로 메시지를 출력창에 출력하고 재입력 받는다.

4C6 is not defined.

- 476쪽 실행 세션 사례 캡처

```
This program computes combination of two natural numbers, n and r.  
Press Control+C to quit.  
Enter n: 6  
Enter r: 4  
6C4 = 15  
Enter n: three  
Must be a number.  
Enter n: -3  
Must be a natural number.  
Enter n: 30  
Enter r: 3  
30C3 = 4060  
Enter n: <control>+C  
Goodbye!
```



**Enter n: 4**  
**Enter r: 6**  
**4C6 is not defined.**