# 고급통계프로그래밍 Solution 3


## 컴퓨터과학부 2017920054 이호준


**과제 index는 번역본 기준입니다.**

## # Ex 6.7

### Solution

```
def is_power(a, b):
    if (a == 0):
        return False
    if (a == 1):
        return True

    if ((a % b == 0) and is_power(a/b, b)):
        return True
    else:
        return False
```

- 기저 조건으로 a == 0과 a == 1을 두었다.
  - a가 0이면 a/b가 0이라는 의미로 a가 b보다 작은 상황이다. 따라서 이는 거듭제곱 관계가 성립할 수 없다.
  - a가 1이면 a%b == 0이면서 a/b == 1인 경우는 a == b인 경우이므로 거듭제곱 관계의 기저(지수가 1)을 확인할 수 있다.
- recursive하게 두 조건(`a % b == 0`, `is_power(a/b, b)`)이 참일 때 True를 반환하도록 하여 문제를 해결하였다.

### Result

```
# Test
print(is_power(4, 2))
print(is_power(1, 10))
print(is_power(3, 2))
print(is_power(16, -4))

# in Stdout:
True
True
False
True
```

# # Ex 6.8

## Solution

```
def gcd(a, b):
    if (b == 0):
        return a

    return gcd(b, a%b)
```

- 주어진 오일러 방법을 이용해 `gcd(a, 0)`일 땐 a를 반환하도록 하고, 이외엔 경우에는 `gcd(a, b) == gcd(b, r)`를 이용하기 위해 `gcd(b, r)`을 재귀호출하였다.

## Result

```
# Test
print(gcd(17, 2))
print(gcd(6, 3))
print(gcd(42, 63))

# in Stdout:
1
3
21
```

# # Ex 7.3

## Solution

```
from math import sqrt as module_sqrt

epsilon = 0.000000001

def newton_sqrt(a):
    x = a
    while True:

        y = (x + a/x) / 2
        if (abs(y - x) < epsilon):
            return y
        x = y

n = 1.0
```

```
while (n <= 9.0):

    n_sqrt = newton_sqrt(n)
    m_sqrt = module_sqrt(n)
    print('{:<5} {:<20} {:<20} {:<20}'.format(n, n_sqrt, m_sqrt, abs(n_sqrt - m_sqrt)))
    n = n + 1
```

- `epsilon` : 오차의 임계값
- `newton_sqrt()` : 뉴튼근사를 이용해 매개변수 a의 근사를 구하는 함수
  - 매개변수 : 제곱근을 구할 숫자 `a`
  - 반환값 : `a`의 제곱근
- `module_sqrt()` : `math.sqrt()`
- n = 1.0 ~ 9.0까지 순회하면서 while 반복을 진행
  - `n_sqrt` : newton_sqrt()를 이용해 구한 n의 제곱근
  - `m_sqrt` : module_sqrt()를 이용해 구한 n의 제곱근
  - `{:<x}` formatting : 왼쪽으로 align한 다음 $x$칸만큼의 공간을 할당하여 출력하는 방법
    각각 5칸, 20칸, 20칸, 20칸

### Result

```
1.0   1.0                  1.0                  0.0
2.0   1.414213562373095    1.4142135623730951   2.220446049250313e-16
3.0   1.7320508075688772   1.7320508075688772   0.0
4.0   2.0                  2.0                  0.0
5.0   2.23606797749979     2.23606797749979     0.0
6.0   2.449489742783178    2.449489742783178    0.0
7.0   2.6457513110645907   2.6457513110645907   0.0
8.0   2.82842712474619     2.8284271247461903   4.440892098500626e-16
9.0   3.0                  3.0                  0.0
```

## # Ex 7.4

### Solution

```
def eval_loop():
    stdin = input()
    while (stdin != 'done'):
        eval(stdin)
        stdin = input()
```

- `stdin` 변수에 입력을 받고, stdin이 **'done'**이 아닌 동안 while loop을 진행
- `stdin`을 `eval()`의 인자로 넣어 함수를 실행한 후, `stdin`에 다시 입력을 받음, 이후 다시 while loop의 조건식으로 돌아간다.

## Result

```
# Test
eval_loop()

# In Stdin:
print("Hello Stat!")
print(1 + 3 * 4)
print(int(input()) * 10)
15
done

# In Stdout:
Hello Stat!
13
150
```