재귀와 반복 - 셈

Recursion and Iteration - Programming on Numbers

실습 1. 최대공약수 구하기

정의

정수 n의 **약수**divisor는 n에서 나누어서 나머지 없이 떨어지는 양수를 말한다. 예를 들어, 54의 약수는 1, 2, 3, 6, 9, 18, 27, 54이고, 24의 약수는 1, 2, 3, 4, 6, 8, 12, 24 이다. 두 정수 m과 n의 **공약수**common divisor는 m의 약수와 n의 약수 중에서 공통이 되는 수를 말한다. 예를 들어, 54와 24의 공약수는 1, 2, 3, 6 이다. 두 정수 m과 n의 **최대공약수**greatest common divisor는 두 수의 **공약수**common divisor들 중에서 가장 큰 수를 말한다. 따라서, 54와 24의 최대공약수는 6이다.

0이 아닌 n과 0의 최대공약수는 |n|이다. (|n|은 n의 절대값) 왜냐하면 0의 약수는 모든 양수가 되므로, n과 0의 공약수는 n의 약수와 같아지고, 따라서 n과 0의 최대공약수는 n과 0의 공약수 중에서 가장 큰 수인 |n|이 된다.

0과 0의 최대공약수는 따져보면 무한대로 큰 수가 되겠지만, 편의상 보통 0으로 한다.

문제

표준라이브러리(Python 3.5 이상)의 math 모듈에 최대공약수를 구하는 gcd 함수가 있다. 이를 사용하여 최대공약수를 구하면 다음과 같은 결과가 나온다.

```
>>> import math
>>> math.gcd(192,-72)
24
>>> math.gcd(18,57)
3
>>> math.gcd(-18,-57)
3
>>> math.gcd(0,24)
24
>>> math.gcd(-24,0)
24
>>> math.gcd(0,0)
0
```

이제 표준라이브러리의 gcd 함수와 똑 같이 작동하는 함수를 직접 만들어보자.

유클리드 알고리즘

유클리드 알고리즘은 나눗셈을 이용한 알고리즘으로서, 두 수의 최대공약수는 두 수의 차이로도 나누어 짐을 이용하여 다음과 같은 등식 을 이용하여 계산한다.

$$\gcd(a, 0) = a$$
$$\gcd(a, b) = \gcd(b, a \bmod b)$$

여기서 mod는 나머지 연산자로 Python의 % 연산자와 같다.

48과 18의 최대공약수는 48을 18로 나눈 나머지가 12이므로, 18과 12의 최대공약수와 같다. 18과 12의 최대공약수는 18을 12로 나눈 나머지는 6이므로, 12과 6의 최대공약수와 같다. 12와 6의 최대공약수는 12을 6으로 나눈 나머지는 0이므로, 6이다.

이를 재귀함수로 작성하면 다음과 같다.

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

gcd(18,48) 호출을 실행추적 하면서 이 함수의 의미를 이해해보자.

```
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
```

유클리드 알고리즘은 음수 인수에 대해서도 잘 작동하지만, 약수, 공약수, 최대공약수는 모두 음수로 얘기하지 않으므로 계산 결과값이 음수인 경우는 부호를 떼고 절대값을 내주어야 한다. 정수의 절대값을 내주는 라이브러리 붙박이 함수는 abs이며 이를 이용하면 된다.

```
>>> abs(-3)
3
>>> abs(4)
4
>>> abs(0)
0
```

인수가 음수라도 최대공약수 결과가 항상 양수가 나오도록 위의 gcd 프로그램을 수정하면 다음과 같다.

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

실습문제 1-1

이 꼬리재귀 함수를 참고하여 아래 틀에 맞추어 while 문으로 변환해보자.

```
1 def gcd(m,n):
2     while n != 0:
3
4
5     return abs(m)
```

이분 알고리즘

뺄셈과 반으로 나누기만 이용한 알고리즘으로서, 유크리드 알고리즘보다 좀 더 빨리 계산한다고 알려져 있다.

$$gcd(m,n) = \begin{cases} n & (m = 0) \\ m & (n = 0) \\ 2 * gcd(m/2, n/2) & (even(m)\ and\ even(n)) \\ gcd(m/2, n) & (even(m)\ and\ odd(n)) \\ gcd(m, n/2) & (odd(m)\ and\ even(n)) \\ gcd(m, (n-m)/2) & (odd(m)\ and\ odd(n)\ and\ m <= n) \\ gcd(n, (m-n)/2) & (odd(m)\ and\ odd(n)\ and\ m > n) \end{cases}$$

이 정의를 재귀함수로 작성하면 다음과 같다.

```
1  def gcd(m,n):
2      if not (m == 0 or n = 0):
3          if m % 2 == 0 and n % 2 == 0:
4              return 2 * gcd(m//2,n//2)
5          elif m % 2 == 0 and n % 2 == 1:
6              return gcd(m//2,n)
7          elif m % 2 == 1 and n % 2 == 0:
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 abs(n)
16         else:
17             return abs(m)
```

gcd(18,48) 호출을 실행추적 해보자.

```
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 * abs(3)
=> 2 * 3
=> 6
```

이 재귀 함수는 꼬리재귀 함수가 아니다. m과 n이 모두 짝수이면 모두 반으로 나누어서 재귀호출하여 얻은 결과 값에 2를 곱해야 하기 때문이다.

실습문제 1-2

위 재귀 함수를 다음과 같이 꼬리재귀 함수로 변환할 수 있다. 밑줄 친 부분을 채워서 코드를 완성해보자.

```
1  def gcd(m,n):
2      def loop(m,n,k):
3          if not (m == 0 or n = 0):
4              if m % 2 == 0 and n % 2 == 0:
5                  return loop(m//2,n//2,_____)
6              elif m % 2 == 0 and n % 2 == 1:
7                  return loop(m//2,n,k)
8              elif m % 2 == 1 and n % 2 == 0:
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 abs(_______)
17             else: # n == 0
18                 return abs(_______)
19     return loop(m,n,1)
```

꼬리재귀 함수로 gcd(18,48) 호출을 실행추적하면 다음과 같이 방식으로 재귀호출이 이루어져야 한다.

```
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)
=> abs(2 * 3)
=> abs(6)
=> 6
```

실습문제 1-3

위 꼬리재귀 함수를 while 문을 사용하여 다음 틀에 맞추어 재작성하자.

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

실습 2. 곱셈 함수 만들기

단순 무식한 곱셈 함수

곱셈연산자가 없다면 덧셈과 뺄셈 연산자만 가지고 다음과 같이 곱셈 함수를 구현할 수 있다. 여기서 m과 n 값은 항상 자연수라고 가정한다. 즉, 음수 곱셈은 고려하지 않는다.

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

이 재귀함수는 덧셈하는 횟수가 n에 비례하고, 공간 사용량도 n에 비례한다.

mult(3,6) 호출을 실행추적하면 다음과 같다.

```
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
```

실습문제 2-1

위의 재귀함수를 공간 사용량이 일정한 꼬리재귀 형태로 함수를 다음 틀에 맞추어 재작성하자.

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

`mult(3,6)` 호출을 실행추적하면 다음과 같이 재귀호출 해야 한다.

```
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
```

실습문제 2-2

작성한 꼬리재귀 함수의 패턴을 참조하여 `while` 문을 사용하여 곱셈함수를 재작성 하시오.

```
1  def mult(m,n):
2      ans =
3      while         :
4
5
6
7      return
```

다음 실습문제에서 사용할 보조 함수

다음은 각각 인수의 2배를 내주는 `double` 함수와 인수의 반을 내주는 `halve` 함수이다.

```
1  def double(n):
2      return n * 2
3
4  def halve(n):
5      return n // 2
```

빠른 곱셈 함수

실습문제 2-3

위의 두 함수를 이용하여 곱셈함수의 덧셈을 하는 회수가 log n에 비례하는 곱셈함수 fastmult를 재귀함수로 다음 틀에 맞추어 작성하시오.

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

알고리즘 : 두번째 인수 n이 짝수인 경우 fastmult(double(m),halve(n))을 재귀호출하고, n이 홀수인 경우 m + fastmult(m,n-1)을 재귀호출 하도록 작성한다.

fastmult(3,6) 호출을 실행추적하면 다음과 같다.

```
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
```

실습문제 2-4

작성한 fastmult 함수를 공간사용량이 일정한 꼬리재귀 형태로 다음 틀에 맞추어 변환하시오.

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

mult(3,6) 호출을 실행추적하면 다음과 같아야 한다.

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

실습문제 2-5

작성한 꼬리재귀 함수를 다음 틀에 맞추어 while 문으로 재작성 하시오.

```
1  def mult(m,n):
2      ans =
3      while n > 0:
4
5
6
7      return ans
```

러시아 농부의 곱셈함수

러시아 농부들은 **두 수**를 **곱할**때 구구단 없이 **덧셈**, **두배하기**(double**함수**), **반나누기**(halve함수)만 가지고 곱셈을 계산하는 영특한 방법을 사용했다. 곱셈 방법은 다음과 같다.

- 곱할 두 수를 나란히 적는다.
- 첫째 수는 두 배를 하고, 둘째 수는 반으로 나누되 나머지는 버린다.
- 이 과정을 둘째 수가 1이 될때까지 계속한다.
- 둘째 수가 짝수인 줄은 모두 지운다.
- 남은 줄의 첫째 수를 모두 더한 값이 답니다.

예를 들어, 57 x 86 은 다음과 같이 계산한다.

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

실습문제 2-6

이를 구현하는 mult 함수를 다음 틀에 맞추어 재귀함수로 작성하시오

```
1  def mult(m,n):
2      def loop(m,n):
3          if n > 1:
4
5
6          else: # n == 1
7
8      if n > 0:
9          return loop(m,n)
10     else:
11         return 0
```

mult(57,86) 호출을 실행추적하면 다음과 같아야 한다.

```
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
```

실습문제 2-7

위 함수를 꼬리 재귀함수로 변환하시오. 이제는 코드 틀을 제공하지 않습니다.

`mult(57,86)` 호출을 실행추적하면 다음과 같아야 한다.

```
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
```

실습문제 2-8

구현한 꼬리재귀 함수를 `while` 루프로 변환하시오