재귀 용법(recursive call, 재귀 호출)

고급 정렬 알고리즘에서 재귀 용법을 사용한다.

1. 재귀 용법(recursive cll, 재귀 호출)

- 함수 안에서 동일한 함수를 호출하는 형태
- 여러 알고리즘 작성시 사용되므로, 익숙해져야 함

---

2. 재귀 용법 이해

- 예제를 풀어보며, 재귀 용법을 이해해보기

예제

- 팩토리얼을 구하는 알고리즘을 Recursive Call 을 활용해서 알고리즘 작성하기

예제 - 분석하기

- 간단한 경우부터 생각해보기
  - 2! = 1 x 2
  - 3! = 1 x 2 x 3 =2! x 3
  - 4! = 1 x 2 x 3 x 4 = 3! x 4
- 규칙이 보임 : n! = (n-1)! x n
  1. 함수를 하나 만든다.
  2. 함수(n)은 n>1이면 return n x 함수(n-1)
  3. 함수(n)은 n=1이면 return n
- 검증(코드로 검증하지 않고, 직접 간단한 경우부터 대입해서 검증해야 함)
  1. 먼저 2! 부터

- 함수(2) 이면, 2 > 1 이므로 2 X 함수(1)
- 함수(1) 은 1 이므로, return 2 X 1 = 2 맞다!

2. 먼저 3! 부터
   - 함수(3) 이면, 3 > 1 이므로 3 X 함수(2)
   - 함수(2) 는 결국 1번에 의해 2! 이므로, return 2 X 1 = 2
   - 3 X 함수(2) = 3 X 2 = 3 X 2 X 1 = 6 맞다!
3. 먼저 4! 부터
   - 함수(4) 이면, 4 > 1 이므로 4 X 함수(3)
   - 함수(3) 은 결국 2번에 의해 3 X 2 X 1 = 6
   - 4 X 함수(3) = 4 X 6 = 24 맞다!

예제 - 코드 레벨로 적어보기

```
def factorial(num):
  if num>1:
    return num * factorial(num-1)
  else:
    return num
```

```
for num in range(10):
  print(factorial(num))
>>
0
1
2
6
24
120
720
5040
40320
362880
```

예제 - 시간 복잡도와 공간 복잡도

- factorial(n)은 n-1번의 factorial()함수를 호출해서, 곱셈을 함
  - 일종의 n-1번 반복문을 호출한 것과 동일

    - factorial()함수를 호출할 때마다, 지역변수 n이 생성됨
- 시간 복잡도/공간 복잡도는 O(n-1)이므로 결국, 둘 다 O(n)

---

3. 재귀 호출의 일반적인 형태

```
# 일반적인 형태1
def function(입력):
    if 입력 > 일정값: # 입력이 일정 값 이상이면
        return function(입력 - 1) # 입력보다 작은 값
    else:
        return 일정값, 입력값, 또는 특정값 # 재귀 호출 종료
```

```
# 일반적인 형태2
def function(입력):
    if 입력 <= 일정값:               # 입력이 일정 값보다 작으면
        return 일정값, 입력값, 또는 특정값              # 재귀 호출 종료
    function(입력보다 작은 값)
    return 결과값
```

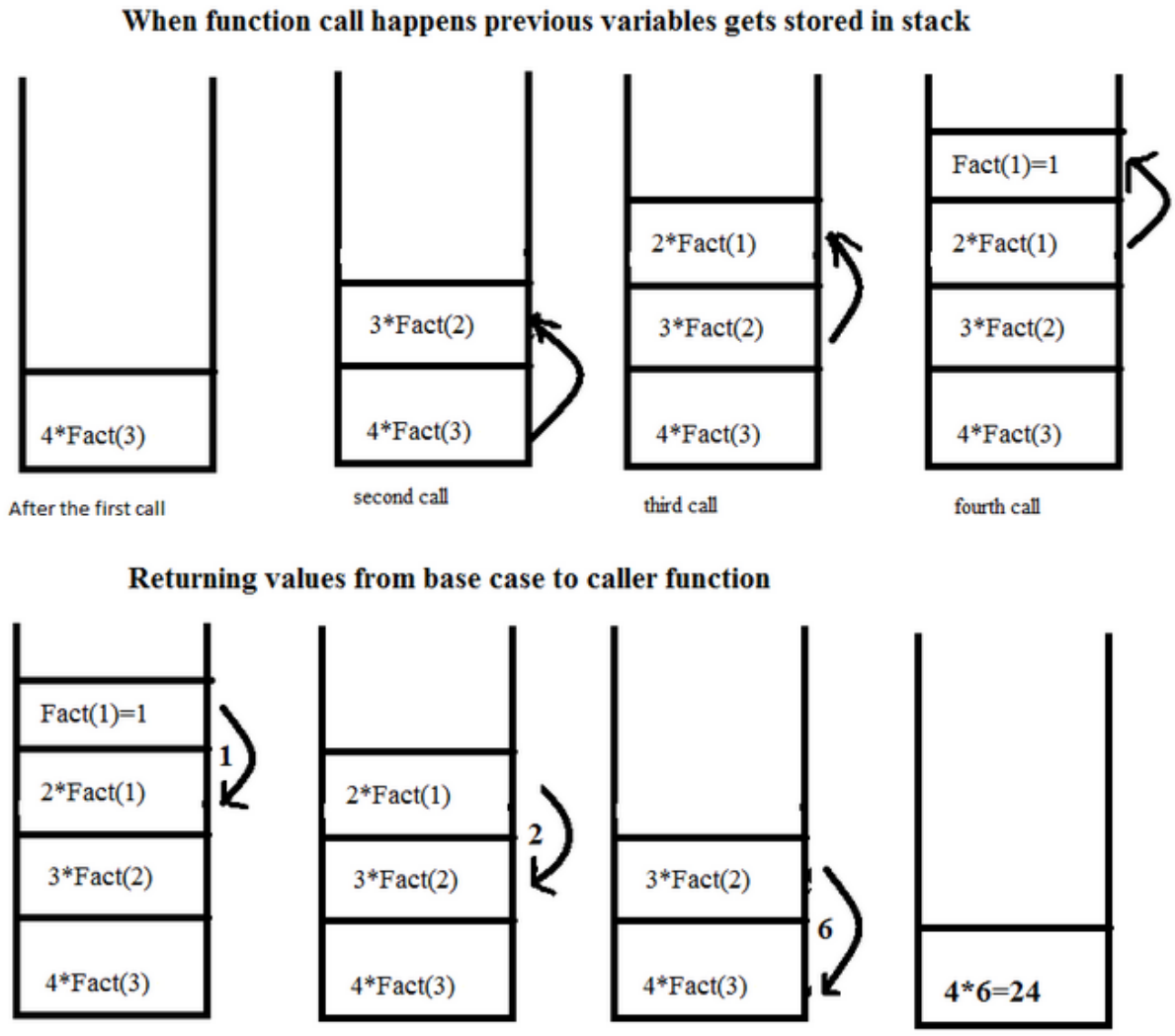
```
def factorial(num):
    if num <= 1:
        return num

    return num * factorial(num - 1)
```

```
for num in range(10):
    print (factorial(num))
>>0
1
2
6
24
120
720
5040
40320
362880
```

재귀 호출은 스택의 전형적인 예

- 함수는 내부적 스택처럼 관리된다.



4. 재귀 용법을 활용한 프로그래밍 연습

1. 다음 함수를 재귀 함수를 활용해서 완성해서 1부터 num까지의 곱이 출력되게 만들어라!

```
def muliple(data):
    if data <= 1:
        return data

return ------------------------multiple(10)
```

```
def multiple(num):
  val = 1
```

```
  for i in range(1,num+1):
    val = val * i
  return val
```

```
def multiple(num):
  if num<=1:
    return num
  return num * multiple(num-1)

multiple(10)
>>3628800
```

2. 숫자가 들어 있는 리스트가 주어졌을 때, 리스트의 합을 리턴하는 함수를 만들어라!

```
참고: 임의 값으로 리스트 만들기 random.sample(0 ~ 99까지 중에서, 임의로 10개를 만들어서 10개 값을 가지는 리스트 변수 만들기
import random
data = random.sample(range(100), 10)
```

```
import random
data = random.sample(range(100),10)
print(data)
>>[72, 50, 8, 38, 77, 32, 90, 48, 74, 79]
```

```
def sum_list(data):
  if len(data)<=1:
    return data[0]
  return data[0] + sum_list(data[1:])

sum_list(data)
>>568
```

3. 회문(palindrome)은 순서를 거꾸로 읽어도 제대로 읽은 것과 같은 단어와 문장을 의미한다. 회문을 판별할 수 있는 함수를 리스트 슬라이싱을 활용해서 만들어라

```
def palindrome(string):
  if len(string)<=1:
    return True
```

```
  if string[0] == string[-1]:
    return palindrome(string[1:-1])
  else:
    return False
```

4. 문제

   1, 정수 n에 대해

   2. n이 홀수이면 3 X n + 1 을 하고,

   3. n이 짝수이면 n 을 2로 나눈다.

   4. 이렇게 계속 진행해서 n 이 결국 1이 될 때까지 2와 3의 과정을 반복한다.

   예를 들어 n에 3을 넣으면,

```
3
10
5
16
8
4
2
1
```

   이 된다.

   이렇게 정수 n을 입력받아, 위 알고리즘에 의해 1이 되는 과정을 모두 출력하는 함수를 작성하여라.

```
def func(n):
  print(n)
  if n== 1:
    return n
  if n%2==1:
    return func(((3*n)+1))
  else:
    return (func(int(n/2)))
```

```
func(3)
>>
3
10
5
16
8
```

```
4
2
1
```

5. 문제

프로그래밍 연습

```
문제: 정수 4를 1, 2, 3의 조합으로 나타내는 방법은 다음과 같이 총 7가지가 있음
1+1+1+1
1+1+2
1+2+1
2+1+1
2+2
1+3
3+1
정수 n이 입력으로 주어졌을 때, n을 1, 2, 3의 합으로 나타낼 수 있는 방법의 수를 구하시오
```

힌트: 정수 n을 만들 수 있는 경우의 수를 리턴하는 함수를 f(n) 이라고 하면,f(n)은 f(n-1) + f(n-2) + f(n-3) 과 동일하다는 패턴 찾기출처: ACM-ICPC > Regionals > Asia > Korea > Asia Regional - Taejon 2001

```
def func(data):
  if data==1:
    return 1
  elif data ==2:
    return 2
  elif data ==3:
    return 4
  return func(n-1) + func(n-2) + func(n-3)
```