파이썬프로그래밍 2분반 ## <12주차 과제> ---

정보보안공학과 ### 202121556 ### 곽지현

2023-05-26

In [7]:
```
# 1. 재귀함수의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

def recursion(parameter_x): # recursion 함수 생성
    print("[recursion] parameter_x :", parameter_x) # parameter_x의 값을 출력
    if parameter_x > 10: # parameter_x의 값이 10보다 크다면
        return parameter_x # parameter_x를 리턴
    else: # parameter_x의 값이 10보다 크지 않다면
        return recursion(parameter_x + 1) # recursion(parameter_x + 1) 함수를 리턴

print(recursion(1)) # recursion 함수 호출
```

```
[recursion] parameter_x : 1
[recursion] parameter_x : 2
[recursion] parameter_x : 3
[recursion] parameter_x : 4
[recursion] parameter_x : 5
[recursion] parameter_x : 6
[recursion] parameter_x : 7
[recursion] parameter_x : 8
[recursion] parameter_x : 9
[recursion] parameter_x : 10
[recursion] parameter_x : 11
11
```

In [3]:
```
# 2. 재귀함수를 이용한 팩토리얼 구하기의 이해를 위해 다음 코드를 실행하여 결과를 확인해

def factorial(n): # factorial 함수 생성
    if n == 0: # n의 값이 0과 같다면
        return 1 # 1을 리턴
    else: # n의 값이 0이 아니라면
        return n * factorial(n - 1) # n * factorial(n - 1)을 리턴

# 함수 호출
print("3!", factorial(3))
print("5!", factorial(5))
```

```
3! 6
5! 120
```

In [12]:
```
# 3. 재귀함수를 이용한 피보나치 수열 구하기의 이해를 위해 다음 코드를 실행하여 결과를 확

call_counter = 0 # call_counter 변수 선언

def fibonacci(n): # fibonacci 함수 생성
    global call_counter # call_counter 전역변수 선언
```

```
    call_counter += 1 # call_counter에 1을 증가하여 대입

    if n == 1: # n의 값이 1과 같다면
        return 1 # 1을 리턴
    if n == 2: # n의 값이 2와 같다면
        return 1 # 1을 리턴
    else: # n의 값이 1도 2도 아니라면
        return fibonacci(n - 1) + fibonacci(n - 2) # fibonacci(n - 1) + fibonacci(n -

import time # 시간과 관련된 기능을 가져옴
start = time.time() # start_time에 현재 시,분,초를 초단위로 바꾼값을 대입

# fibonacci함수 호출, call_counter의 값을 출력
call_counter = 0; print("fibonacci(14) : {}, 함수 길이 : {}" .format(fibonacci(14), cal
call_counter = 0; print("fibonacci(30) : {}, 함수 길이 : {}" .format(fibonacci(30), cal
call_counter = 0; print("fibonacci(35) : {}, 함수 길이 : {}" .format(fibonacci(35), cal

end = time.time() # end_time에 현재 시,분,초를 초단위로 바꾼값을 대입
print("전체 실행시간", end-start) # end - start의 값을 출력
```

```
fibonacci(14) : 377, 함수 길이 : 753
fibonacci(30) : 832040, 함수 길이 : 1664079
fibonacci(35) : 9227465, 함수 길이 : 18454929
전체 실행시간 6.266535043716431
```

In [13]:

```
# 4. 재귀함수 메모화의 이해를 위해 다음 코드를 실행하여 결과를 확인해보세요.

memo = {
    1: 1,
    2: 1
} # 딕셔너리 선언
call_counter = 0 # call_counter 변수 선언

def fibonacci_memo(n): # fibonacci_memo 함수 생성
    global call_counter # call_counter 전역변수 선언
    call_counter += 1 # call_counter에 1을 증가하여 대입

    if n in memo: # n의 값이 memo에 있다면
        return memo[n] # memo된 값을 리턴
    else: # n의 값이 memo에 없다면
        output = fibonacci_memo(n - 1) + fibonacci_memo(n - 2)
        # fibonacci_memo(n - 1) + fibonacci_memo(n - 2)의 값을 output에 대입
        memo[n] = output # memo[n]에 output의 값을 대입
        return output # output을 리턴

import time # 시간과 관련된 기능을 가져옴
start = time.time() # start_time에 현재 시,분,초를 초단위로 바꾼값을 대입

# fibonacci함수 호출, call_counter의 값을 출력
call_counter = 0; print("fibonacci_memo(14) : {}, 함수 길이 : {}" .format(fibonacci_mem
call_counter = 0; print("fibonacci_memo(30) : {}, 함수 길이 : {}" .format(fibonacci_mem
call_counter = 0; print("fibonacci_memo(35) : {}, 함수 길이 : {}" .format(fibonacci_mem

end = time.time() # end_time에 현재 시,분,초를 초단위로 바꾼값을 대입
print("전체 실행시간", end-start) # end - start의 값을 출력
```

```
fibonacci_memo(14) : 377, 함수 길이 : 25
fibonacci_memo(30) : 832040, 함수 길이 : 33
fibonacci_memo(35) : 9227465, 함수 길이 : 11
전체 실행시간 0.0
```

In [5]:
```
# 5. 0 ~ n 까지의 합계를 구하는 프로그램입니다. 다음과 같은 결과가 나오도록 빈칸을 채워

def sum(n): # sum 함수 생성
    if n == 0: # n의 값이 0과 같다면
        return 0 # 0을 리턴
    return n + sum(n-1) # n의 값이 00이 아니라면 n + sum(n-1)을 리턴

print("0~5까지의 합계 :", sum(5)) # sum(5) 함수 호출
print("0~10까지의 합계 :", sum(10)) # sum(10) 함수 호출
```

0~5까지의 합계 : 15
0~10까지의 합계 : 55

In [17]:
```
# 6. 리스트의 인덱스 0부터 특정 인덱스 -1 까지 합을 구하는 프로그램입니다. 다음과 같은

def list_sum(list_parameter, end_index): # list_sum 함수 생성
    if not list_parameter or end_index == 0:
    # 리스트가 선언되어 있지 않거나 end_index의 값이 00이라면
        return 0 # 0을 리턴
    return list_parameter[end_index - 1] + list_sum(list_parameter, end_index - 1)
    # list_parameter[end_index - 1] + list_sum(list_parameter, end_index - 1)을 리턴

list1 = [1, 3, 5, 7, 9] # 리스트 선언

print("list[0~1]의 합 : ", list_sum(list1, 2)) # list_sum(list1, 2) 함수 호출
print("list[0~2]의 합 : ", list_sum(list1, 3)) # list_sum(list1, 3) 함수 호출
print("list[0~3]의 합 : ", list_sum(list1, 4)) # list_sum(list1, 4) 함수 호출
```

list[0~1]의 합 : 4
list[0~2]의 합 : 9
list[0~3]의 합 : 16

In [8]:
```
# 7. 특정 수의 제곱수를 구하는 프로그램입니다. 다음과 같은 결과가 나오도록 빈칸을 채워

def power(base, n): # power 함수 생성
    if n == 0: # n의 값이 0과 같다면
        return 1 # 1을 리턴
    return base * power(base, n - 1) # base * power(base, n - 1)을 리턴

print("2의 10제곱 : ", power(2,10)) # power(2,10) 함수 호출
print("2의 11제곱 : ", power(2, 11)) # power(2,11) 함수 호출
```

2의 10제곱 : 1024
2의 11제곱 : 2048