

# 4. Recursion

2024학년도 가을학기

정보컴퓨터공학부 황원주 교수



**부산대학교**  
PUSAN NATIONAL UNIVERSITY

# 강의내용

## 1. 재귀 (순환, Recursion)

- 재귀
- 파이썬 메모리 관리 (Memory management)
- 공간복잡도
- 반복 함수와 재귀함수의 비교
- 시간복잡도
- 5 simple steps for solving any recursive problems

# 재귀 (순환, Recursion)

# 재귀

- 함수/메소드의 실행 과정 중 스스로를 호출하는 것
- 모든 재귀적 (recursive) 알고리즘은 반복적 (iterative) 알고리즘으로 구현하는 것이 가능하고 그 역도 마찬가지이다.
- 재귀 (vs 반복) 장점
  - 프로그램(알고리즘)의 **가독성**을 높일 수 있음
  - **간결한** 코드 작성이 가능함
- 재귀 (vs 반복) 단점
  - 반드시 **시간복잡도**를 줄인다고 할 수 없음
  - 호출스택을 사용하기 때문에 **공간복잡도** 측면에서 비효율적임

} 반복적으로  
구현된 코드는  
때로는 훨씬 더  
복잡하다!

- 재귀로 구현된 함수는 두 부분으로 구성:
  - Base case: 스스로를 더 이상 호출하지 않기 하는 부분 (종료조건, 바닥조건)
  - Recursive case: 스스로를 호출하는 부분
- Recursive case는 반드시 base case를 향해 다가가야 함: 무한 호출을 방지하기 위해 recursive case에서는 선언한 변수 또는 수식의 값이 호출이 일어날 때마다 감소되어 최종적으로 if-문의 조건식에서 base case를 실행하도록 제어해야 함 (무한 호출이 발생하면 실행 시 RecurrsionError 출력)

# 파이썬 메모리 관리

## 메모리

### stack 영역

- 지역변수와 매개변수를 저장하는 공간이며 메소드의 호출과 함께 할당되며 메소드의 수행이 종료되면 소멸함

메소드3

메소드2

메소드1

메소드 호출 시 할당받고,  
수행이 종료되면 없어짐

### heap 영역

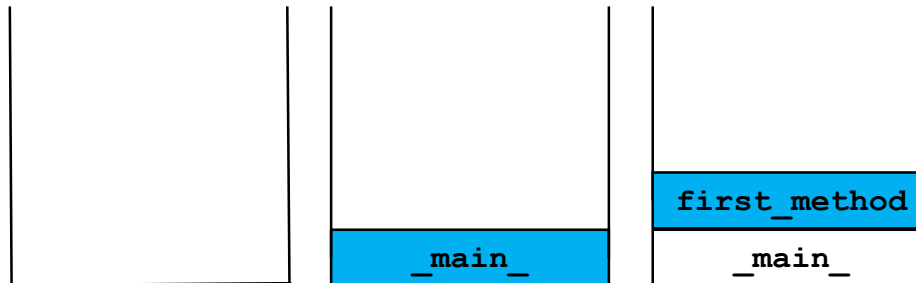
- 파이썬은 모든 게 객체이므로 값을 heap 영역에 저장하고 stack 영역에서 참조함

객체  
리스트, 튜플  
딕셔너리

레퍼런스 카운트가 0인 객체들은 파이썬 가상머신이 garbage collector로 사용하지 않는 메모리를 자동으로 찾아내어 운영체제에 반환함

# 공간복잡도

- 호출스택(call stack): 메서드의 작업공간. 메서드가 호출되면 메서드 수행에 필요한 메모리공간을 할당 받고 메서드가 종료되면 사용하던 메모리를 반환한다
- 호출스택의 특징
  - 메서드가 호출되면 수행에 필요한 메모리를 스택영역에 할당 받는다
  - 맨 처음 호출스택은 `__main__` 라는 스택 프레임(stack frame)을 가진다
  - 호출스택의 제일 위에 있는 메서드가 현재 실행중인 메서드다
  - 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드다
  - 메서드가 수행을 마치면 사용했던 메모리를 반환한다

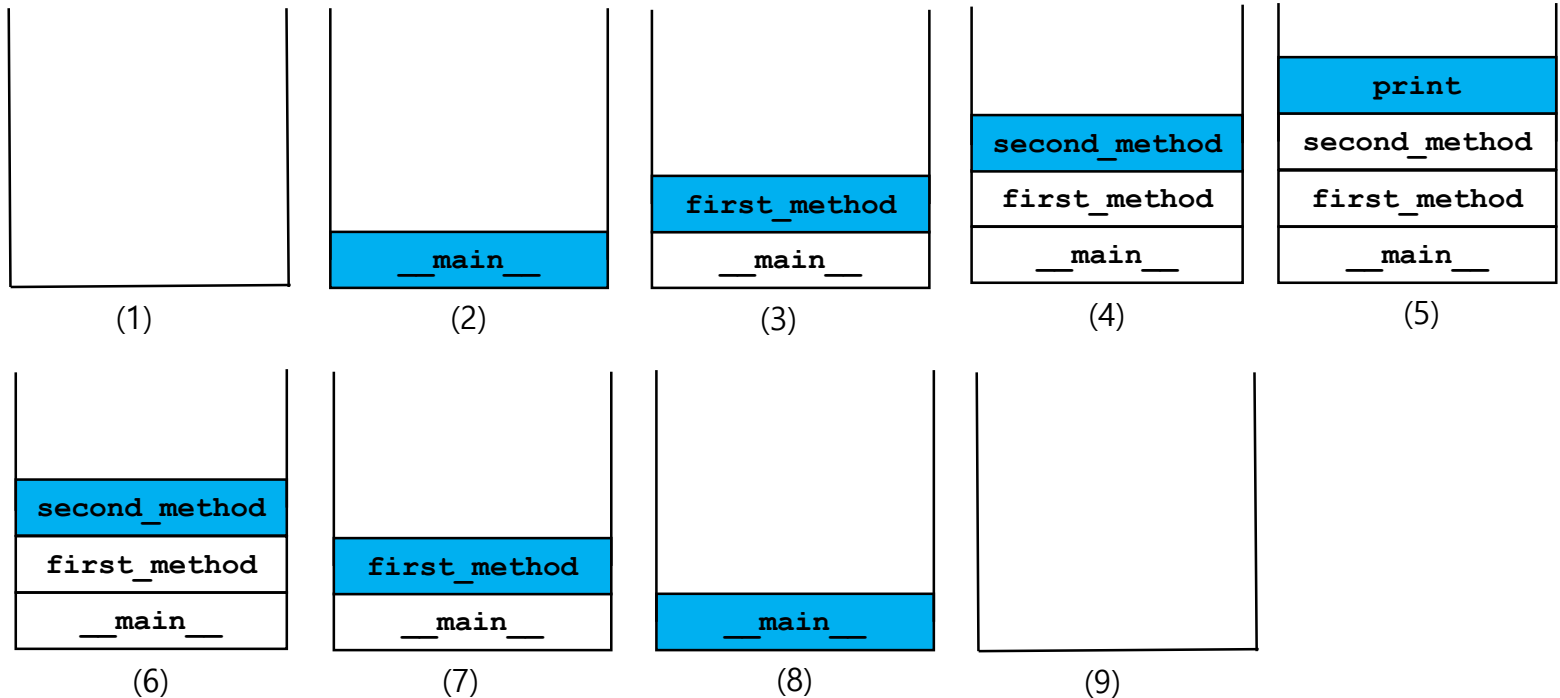


# 호출 스택의 변화

```
def go():  
    first_method()  
  
def first_method():  
    second_method()  
  
def second_method():  
    print("second_method()")
```

go()

실행 중인 메서드  
(맨 위의 스택 프레임 하나만 실행)





# 반복함수와 재귀함수의 비교

(예) factorial(3)을 반복함수와 재귀함수로 구현

## 반복함수

```
def factorial(n):  
    product=1  
    for i in range(n):  
        product=product*(i+1)  
    return product  
  
print(factorial(3))
```

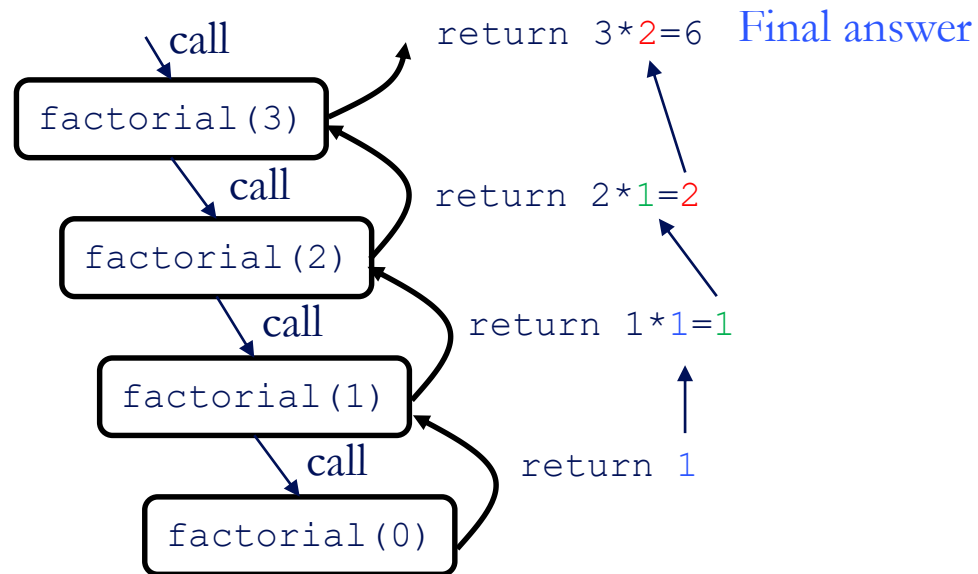
i	product * (i+1)	product
0	1 * (0+1)	1
1	1 * (1+1)	2
2	2 * (2+1)	6

## Recursion trace

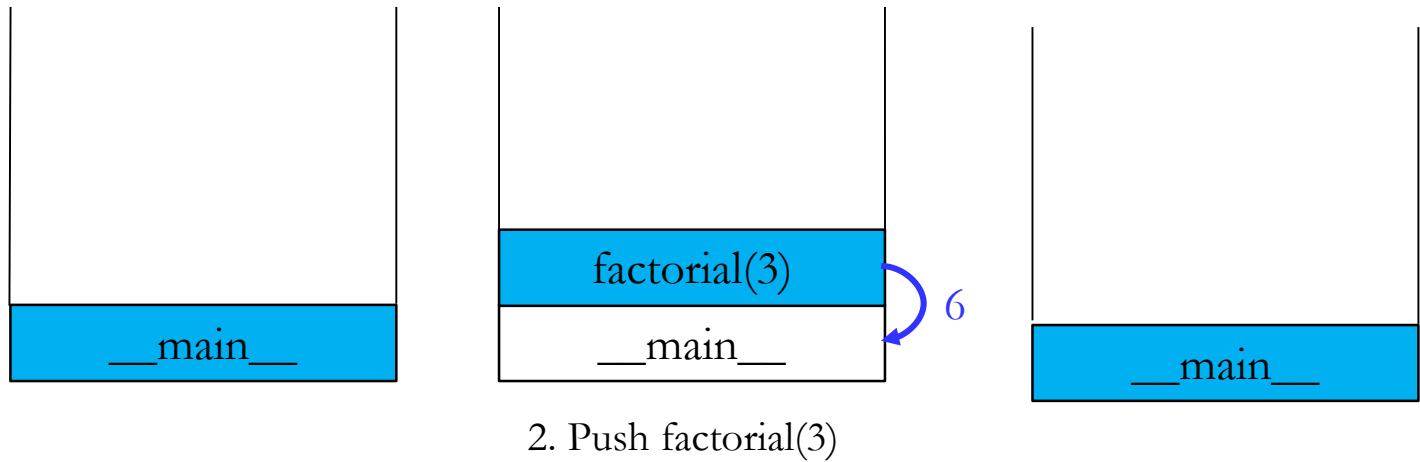
- A box for each recursive call
- An arrow from each caller to callee
- An arrow from each callee to caller showing return value

## 재귀함수

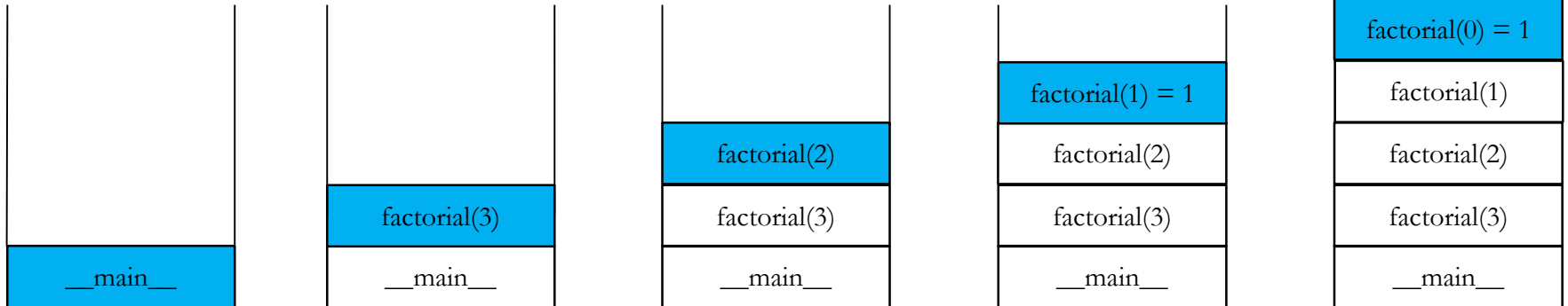
```
def factorial(n):  
    if n == 0: #base case  
        return 1  
    else: #recursive case  
        return n*factorial(n-1)  
  
print(factorial(3))
```



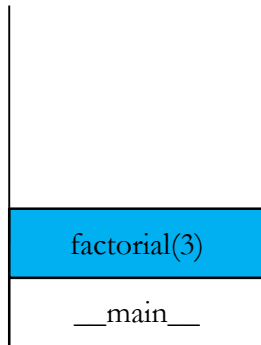
- 반복함수일 때 호출스택의 변화



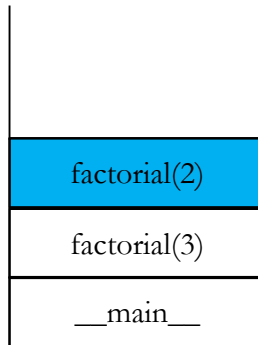
## • 재귀함수일 때 호출스택의 변화



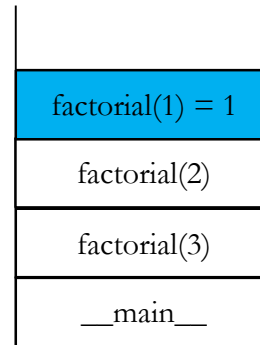
2. Push  
factorial(3)



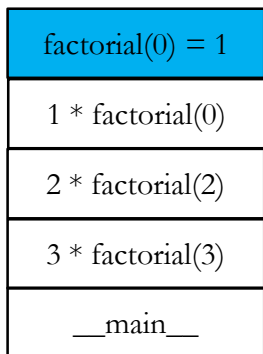
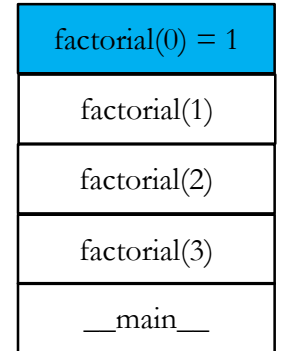
3. Push  
factorial(2)



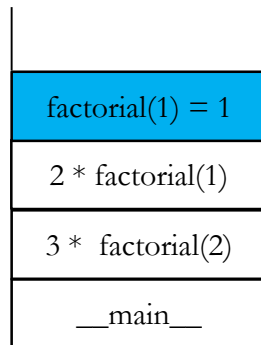
4. Push  
factorial(1)



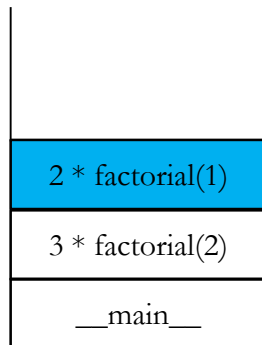
5. Push  
factorial(0)



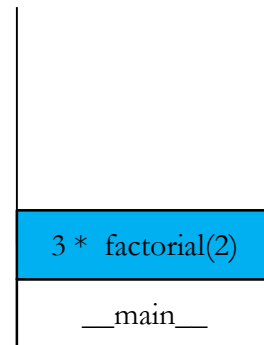
6. Pop factorial(0)  
return 1



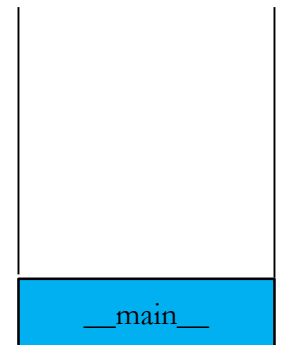
7. Pop factorial(1)  
return 1



8. Pop factorial(2)  
return 2



9. Pop factorial(3)  
return 6



# 시간복잡도

- 예 :  $1 + 2 + \dots + n$ 을 구하는  $\text{sum}(n)$ 을 재귀함수로 정의해보자

$$\text{sum}(n) = \underbrace{1+2+\dots+(n-1)}_{\text{sum}(n-1)} + n$$

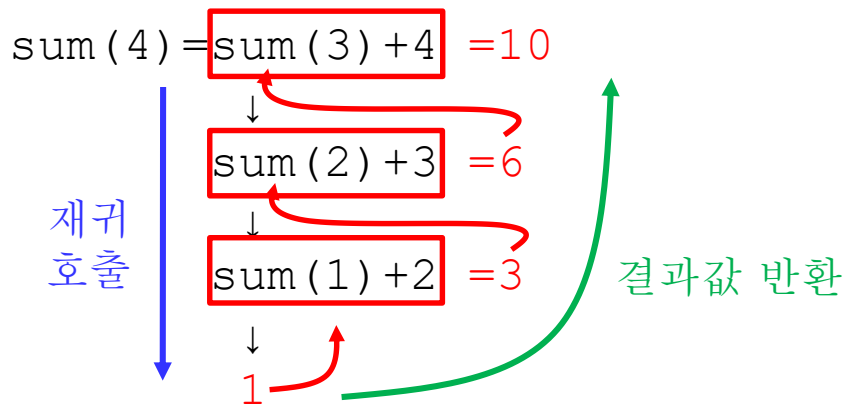
수행시간 :

$\text{sum}(n)$ 의 수행시간을  $T(n)$ 이라고 하면,

```
def sum(n):  
    if n==1: return 1  
    return sum(n-1) + n
```

$$\begin{aligned} T(n) &= T(n-1) + c \quad (\text{점화식}) \\ &= (T(n-2) + c) + c \\ &= T(n-2) + 2c \\ &\vdots \\ &= T(n - (n-1)) + (n-1)c \\ &= \underbrace{T(1)}_{=c} + (n-1)c \end{aligned}$$

$$\begin{aligned} &= cn \\ &= O(n) \quad (\leftarrow \text{Big-Oh 표기법의 정의}) \end{aligned}$$



# Simple steps for solving any recursive problems

[출처] <https://www.youtube.com/watch?v=ngCos392W4w&t=1085s>

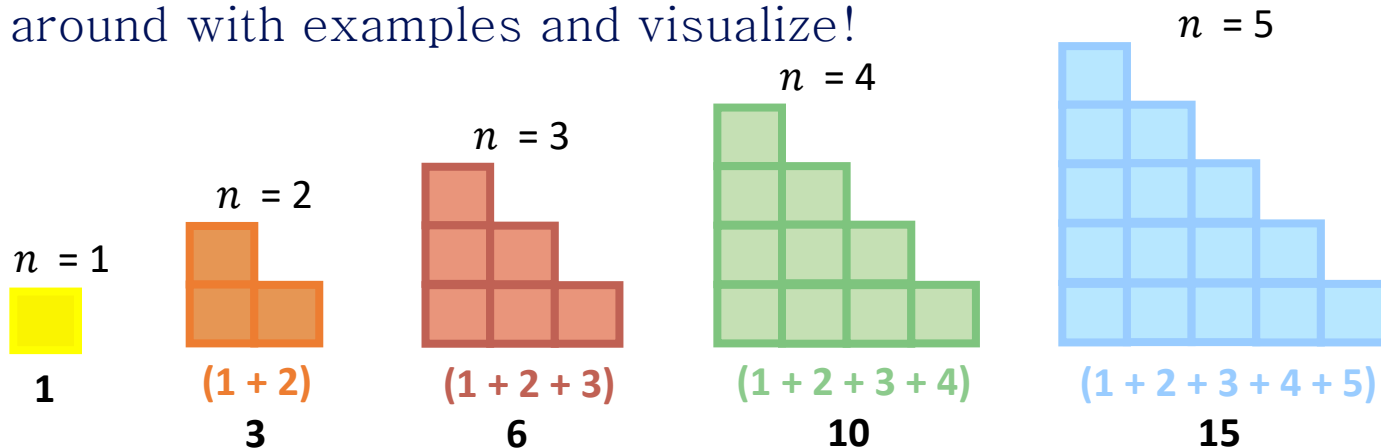
- 예:  $1+2+\dots+n$ 을 구하는  $\text{sum}(n)$ 을 재귀함수로 정의해보자.

- 1. What is the simplest possible input?

  - $\text{sum}(n)=0$

 Base Case

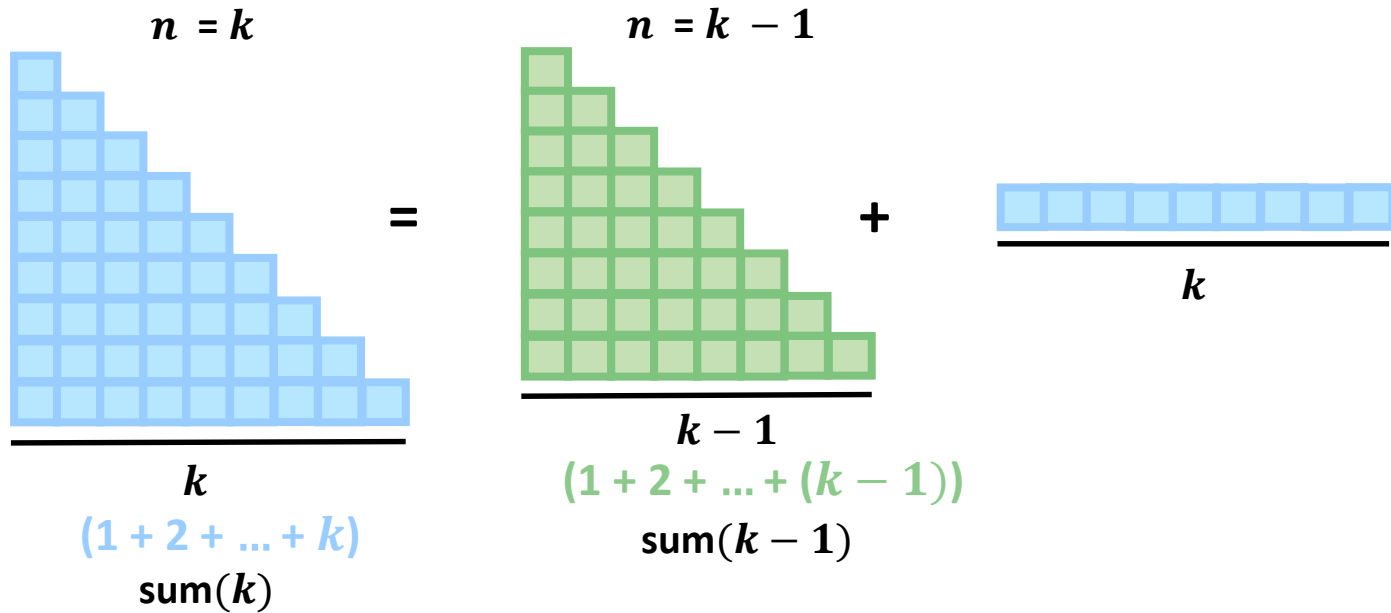
- 2. Play around with examples and visualize!



- 3. Relate hard cases to simpler cases

  - Can you relate  $\text{sum}(4)$  and  $\text{sum}(5)$ ?

- 4. Generalize the pattern



- 5. Write code by combining recursive pattern with the base case

$$\text{sum}(n) = \begin{cases} 0 & \text{if } n = 0 \\ \text{sum}(n - 1) + n & \end{cases}$$

# How to debug recursive functions

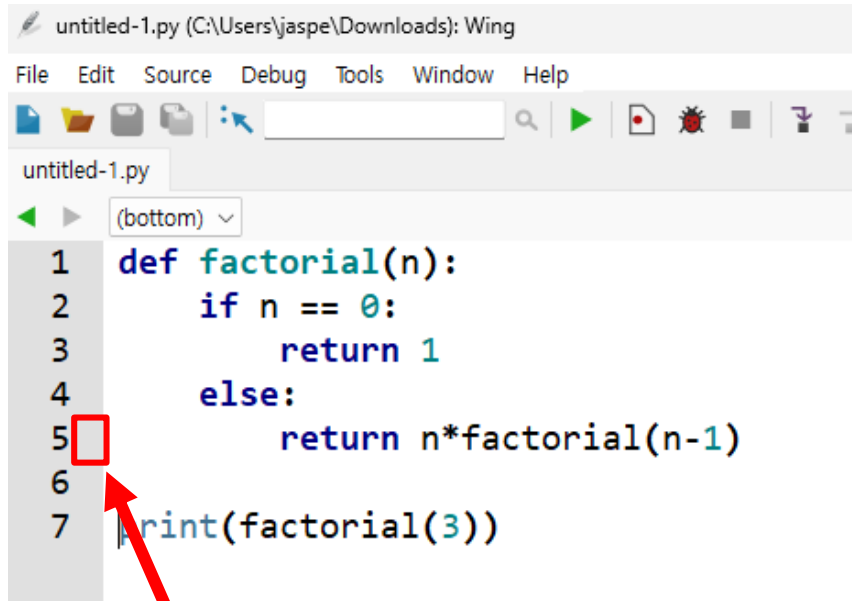
[출처] <https://www.youtube.com/watch?v=Btoobiaf5Gw>

The screenshot displays the Wing IDE interface with four main panes:

- Code (코드):** The top-left pane shows the source code of a recursive factorial function. Line 5, `return n*factorial(n-1)`, is highlighted in red, indicating the current execution point.
- Call Stack:** The top-right pane shows the sequence of function calls. It lists the current call and its recursive predecessors, all pointing to the same line in the `factorial` function.
- Stack Data:** The bottom-left pane shows the current state of the stack frame. It lists local variables (like `n`) and global variables (like `__doc__`, `__file__`, etc.).
- Debug Console (Debug 입출력):** The bottom-right pane shows the output of the debug session, including I/O and the Python Shell.

# How to debug recursive functions

[출처] <https://www.youtube.com/watch?v=Btoobiaf5Gw>

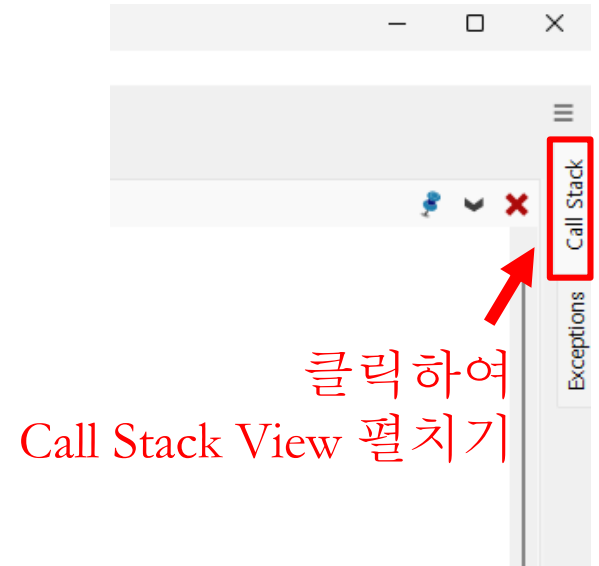


The screenshot shows the Wing IDE interface. The menu bar includes File, Edit, Source, Debug, Tools, Window, and Help. The toolbar contains icons for file operations, search, and execution. The editor window displays a Python file named 'untitled-1.py' with the following code:

```
1 def factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n*factorial(n-1)  
6  
7 print(factorial(3))
```

A red square highlights the line number '5' in the left margin, and a red arrow points to it from the text '클릭하여 BreakPoint 생성'.

클릭하여 BreakPoint 생성

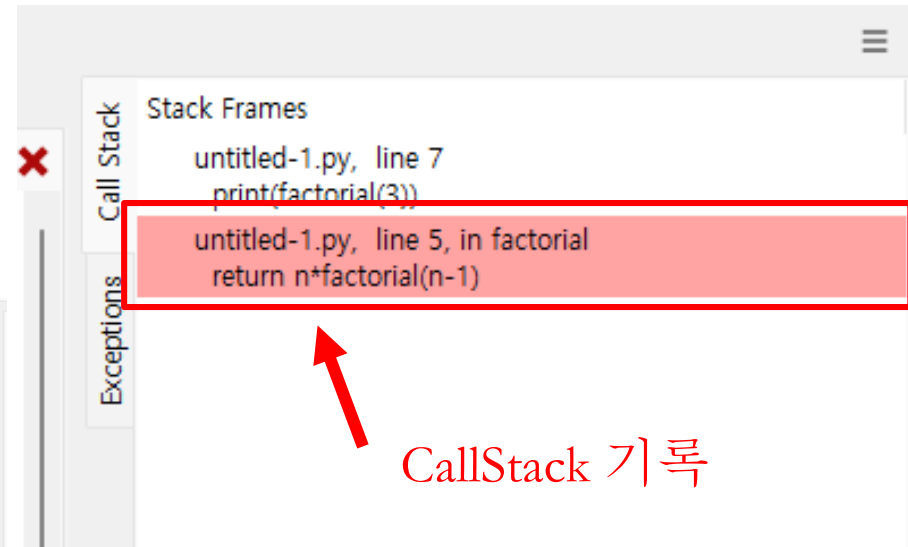
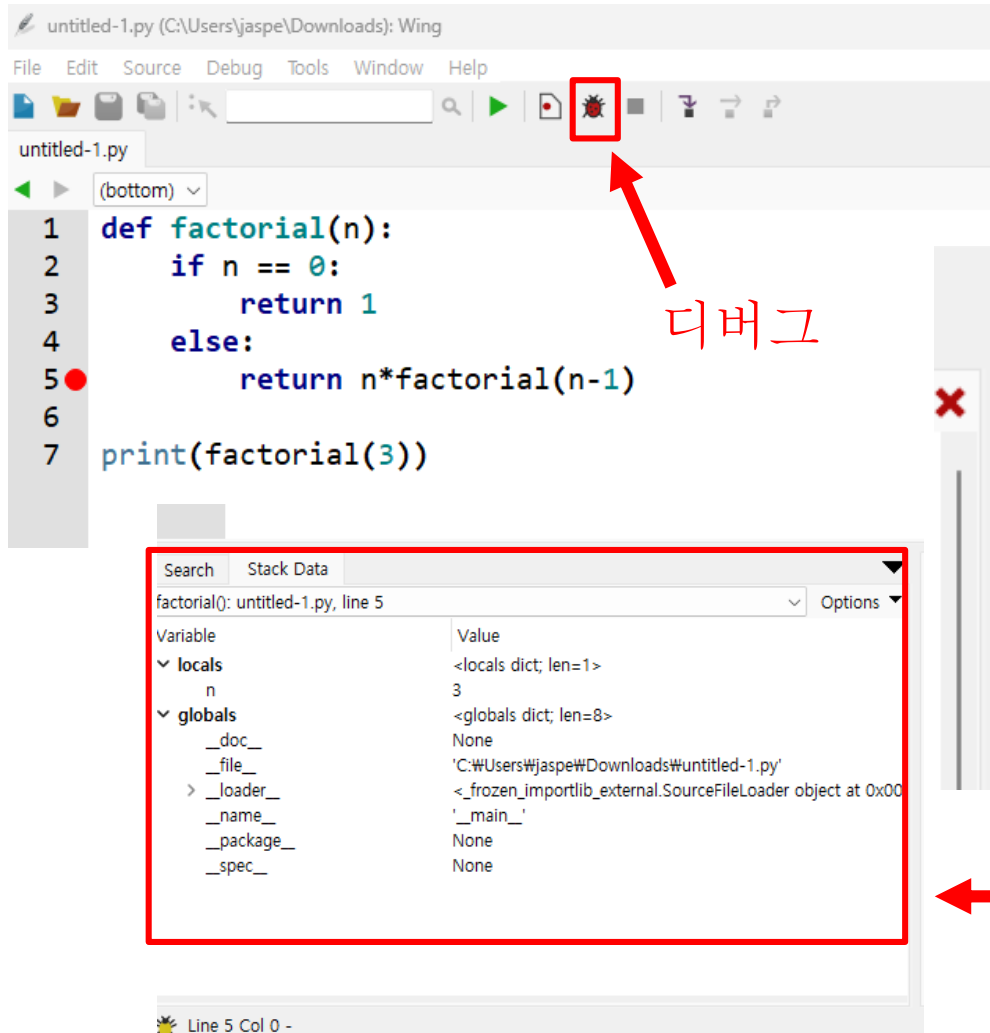


클릭하여  
Call Stack View 펼치기



# How to debug recursive functions

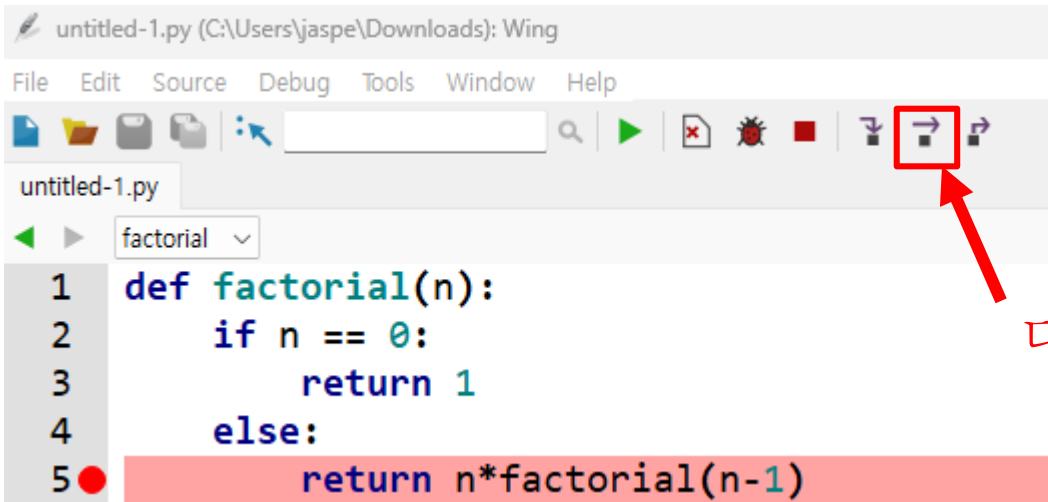
[출처] <https://www.youtube.com/watch?v=Btoobiaf5Gw>



← Stack Data

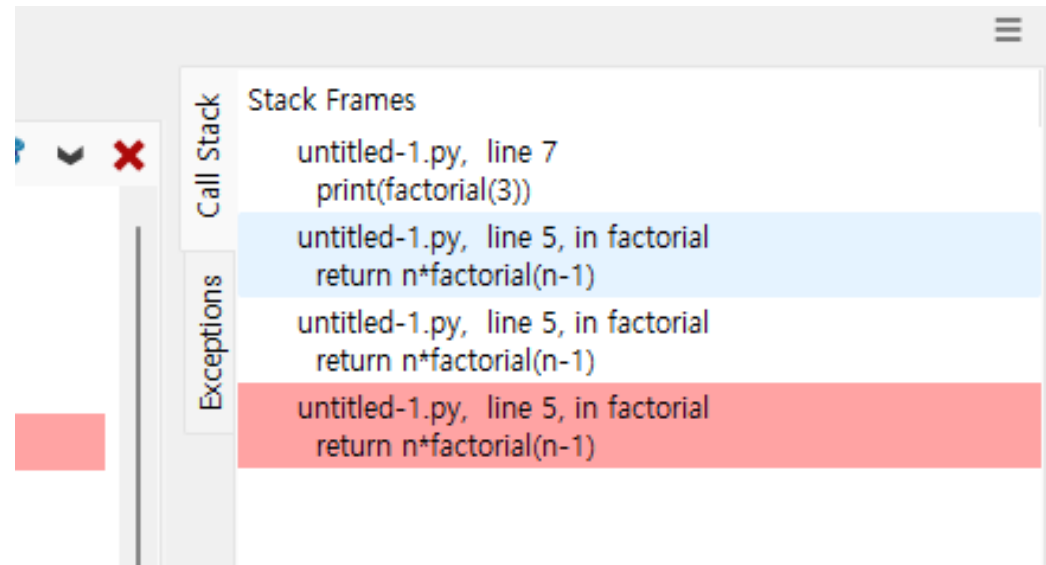
# How to debug recursive functions

[출처] <https://www.youtube.com/watch?v=Btoobiaf5Gw>



```
untitled-1.py (C:\Users\jaspe\Downloads): Wing
File Edit Source Debug Tools Window Help
[Icons] [Search] [Run] [Stop] [Break] [Step Over] [Step Into] [Step Out]
untitled-1.py
factorial
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n*factorial(n-1)
```

다음줄 실행



# 감사합니다!



부산대학교  
PUSAN NATIONAL UNIVERSITY