Functions

국민대 소프트웨어학부

수업목표

- What is a function
- Calling a function
- Passing arguments to a function
- Returning a value
- Variable scope
- Naming variables
- Recursive Functions
- lambda function

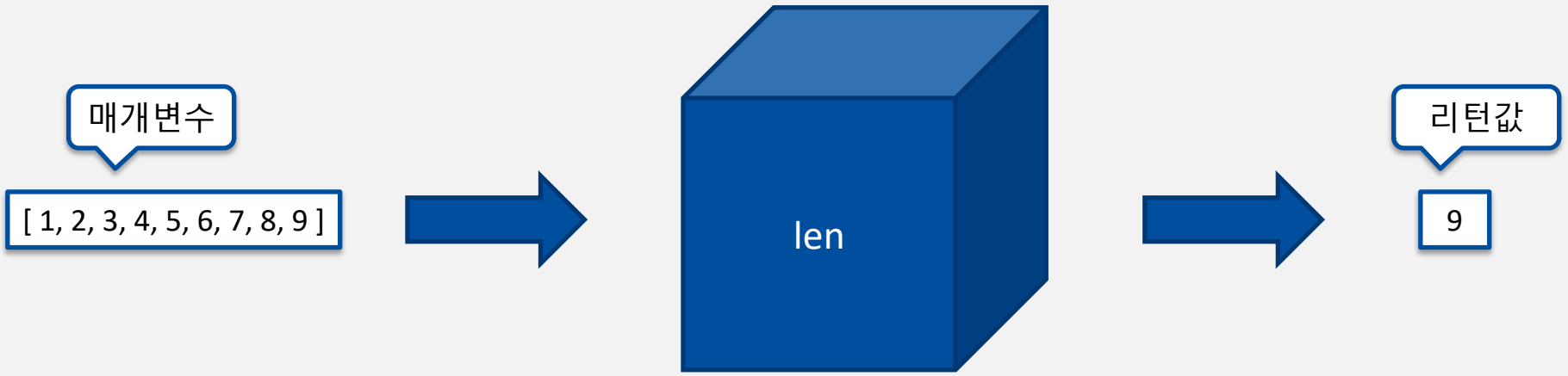
What is a function

- 프로그램
  - 작성하기 쉽고 관리하기 편리한 작은 조각으로 구성
  - Divide-and-Conquer
  - Function, Object, Module

- 함수 (function)
  - 특정 작업을 수행하는 명령어들의 모음
  - def 키워드 사용하여 생성
  - 호출(call)하여 사용
  - 예) print, input, len, int, float, str, max, min

What is a function

- 함수 (function)
  - 특정 작업을 수행하는 명령어들의 모음
  - def 키워드 사용하여 생성
  - 호출(call)하여 사용
  - 예) print, input, len, int, float, str, max, min



What is a function

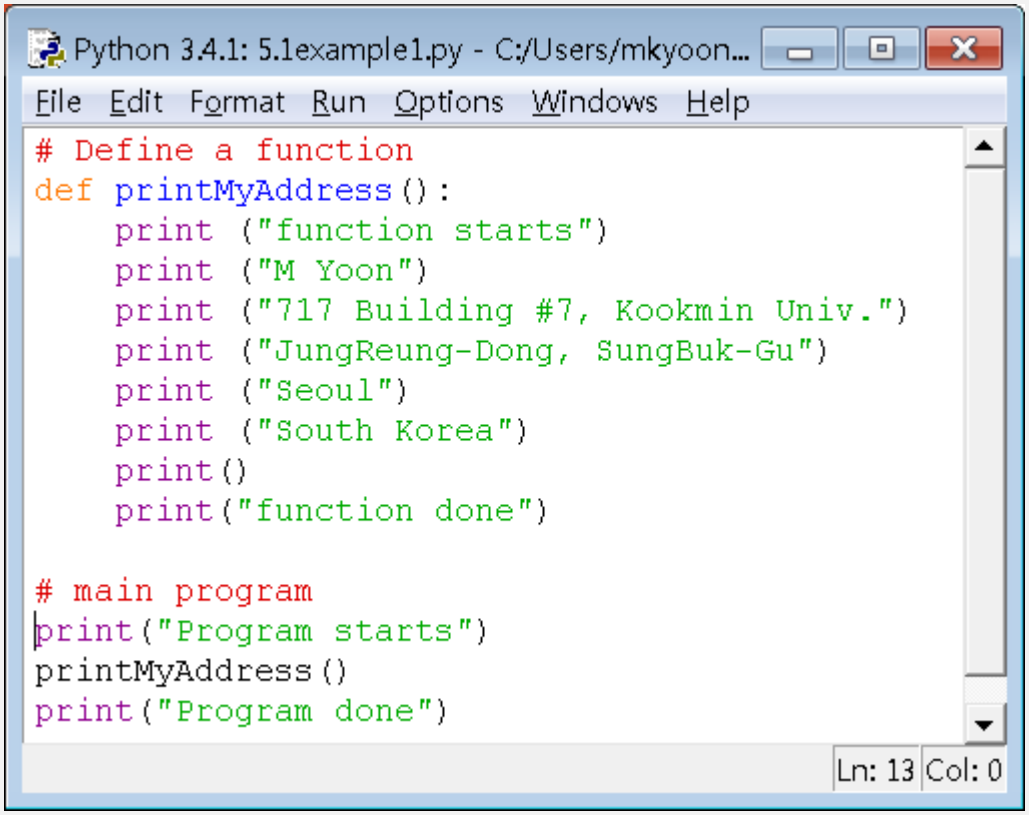
- 함수 정의
  - def 함수명():
  - def 키워드로 정의할 수 있다.
  - <매개 변수>는 없을 수도 있다.
  - 리턴 값은 없을 수도 있다.

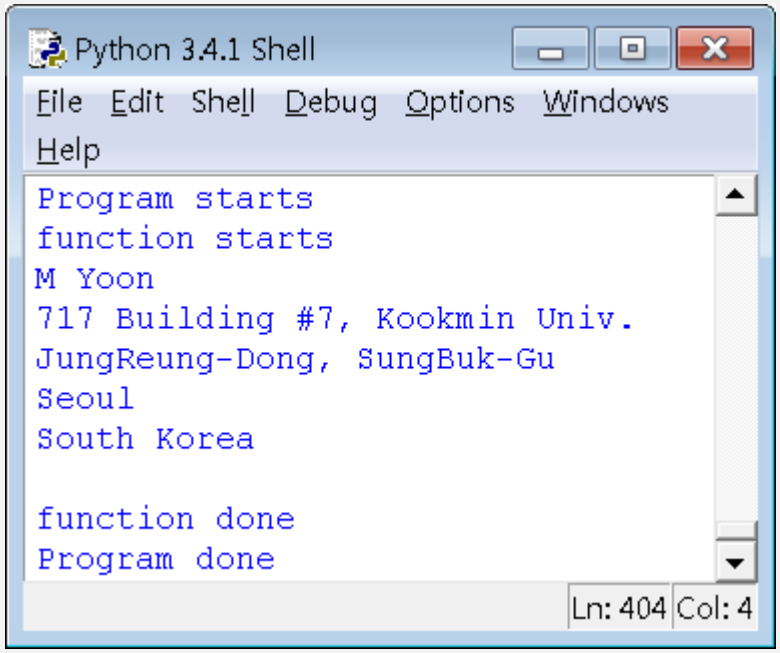
```
def <함수 이름>(<매개 변수>, <매개 변수>, ...) :
    <코드>
```

```
# 함수 선언
def print_my_info() :
    print("My name is Han.")
    print("I am 22 years old.")
    print("Nice to meet you.")

# 함수 호출
print_my_info()
```

What is a function

- 함수 정의





What is a function

- 함수 정의

```
# 함수 선언
def get_sum(start, end) :
    ret = 0
    for num in range(start, end + 1) :
        ret += num
    return ret

a = int(input("A >"))
b = int(input("B >"))

print("{} 부터 {} 까지 합은 {} 입니다.".format(a, b, get_sum(a, b)))
```

What is a function

- 가변 매개변수 함수
  - 파이썬에서는 매개변수를 원하는 만큼 받을 수 있는 함수를 만들 수 있다.
  - 가변 매개변수는 하나만 사용 할 수 있으며, 가변 매개변수 뒤에는 일반 매개변수가 올 수 없다.

```
def <함수 이름>(<매개 변수>, <매개 변수>, … , *<가변 매개변수>) :
    <코드>
```

```
# 가변 매개변수 함수를 정의 할때는 *를 가변 매개변수 이름 앞에 붙인다.
def get_sum(*args) :
    ret = 0
    for each in args :
        ret += each
    return ret

print(get_sum(1, 2, 3, 4, 5, 6, 7, 8, 9))
```

What is a function

- 기본 매개변수
  - 인자가 전달되지 않을 경우 값을 정의 하는 매개변수
  - 기본 매개변수 뒤에는 일반 매개변수가 올 수 없다.

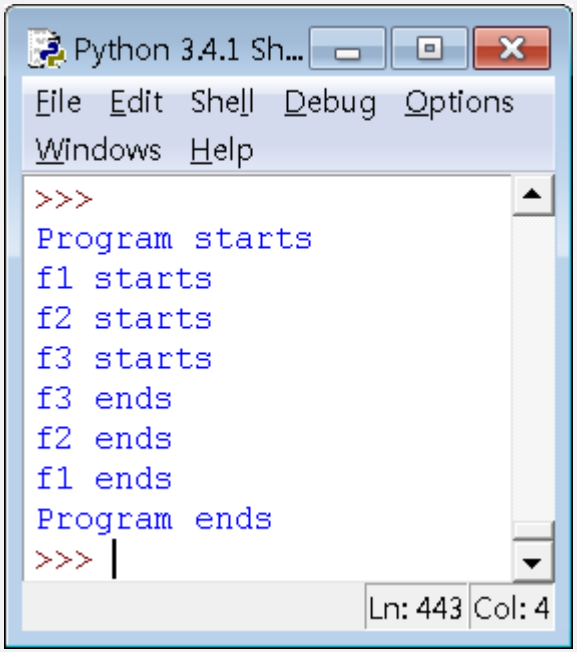
```
# 기본 매개 변수
def print_my_info( n = 1 ) :
    for i in range(n) :
        print("My name is Kim.")


print_my_info()
print("--구분선--")
print_my_info(3)
```

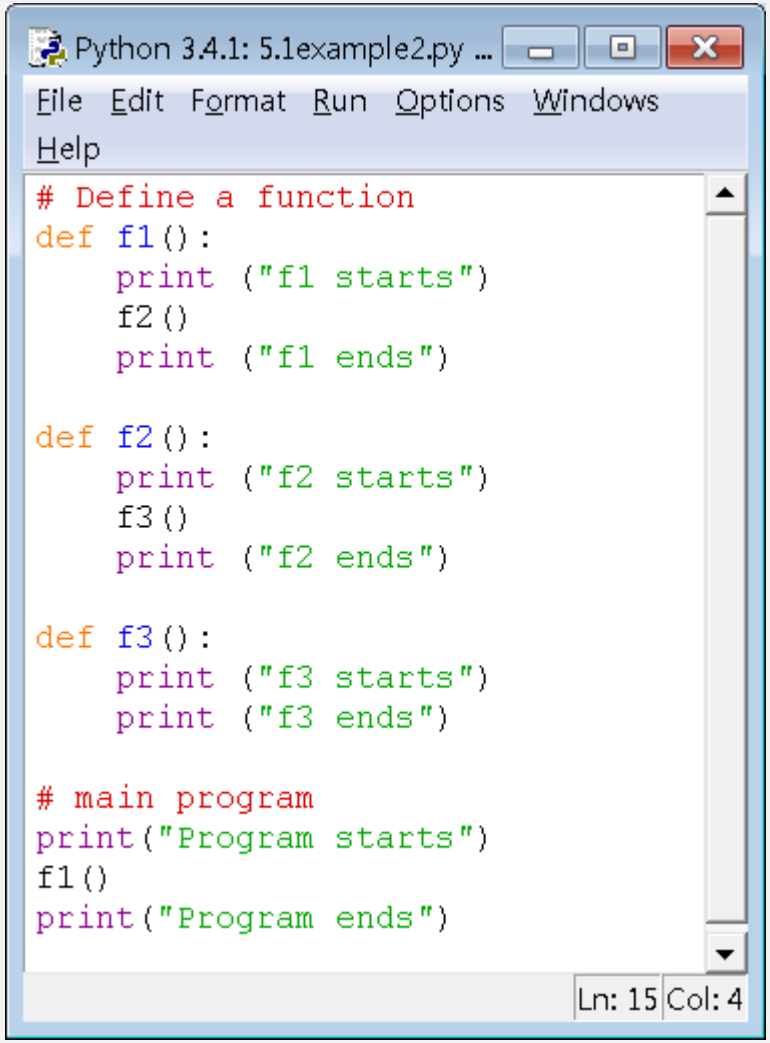
What is a function

- 함수 사용
  - 반복 호출 용이
  - 중복 코드 제거
    - 에러 발생 가능성 낮춤
  - 인자(argument) 를 함수에 전달 가능
    - 매개변수(parameter)
  - 함수 결과값 리턴 가능

Calling a function

- 함수 호출(call)
  - 호출자(caller)
  - 피호출자(callee)



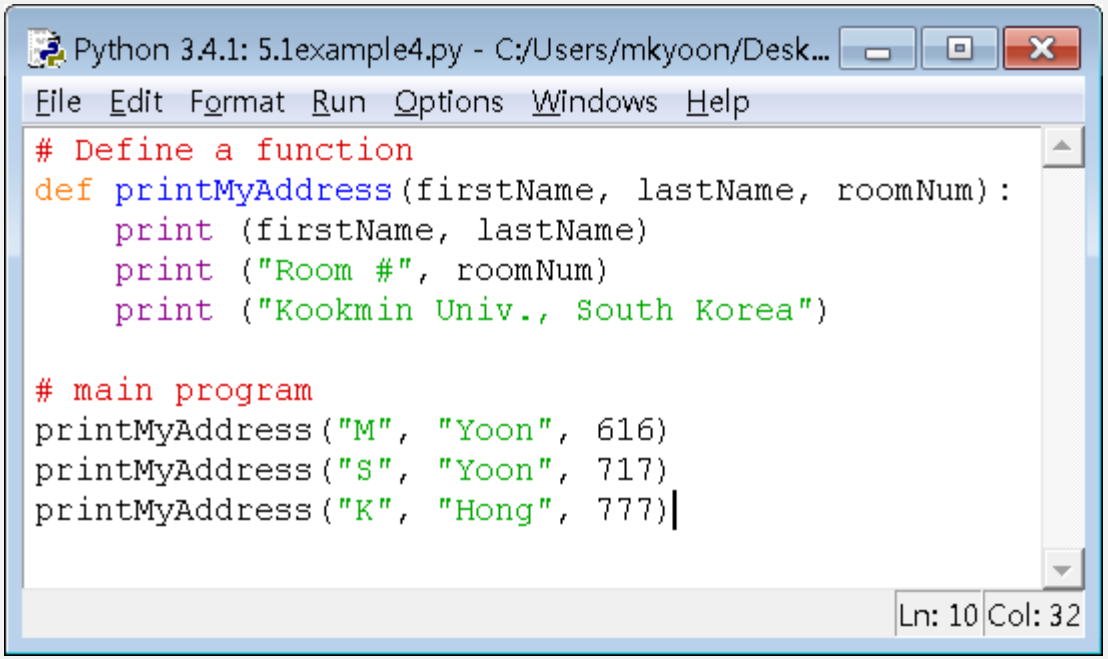


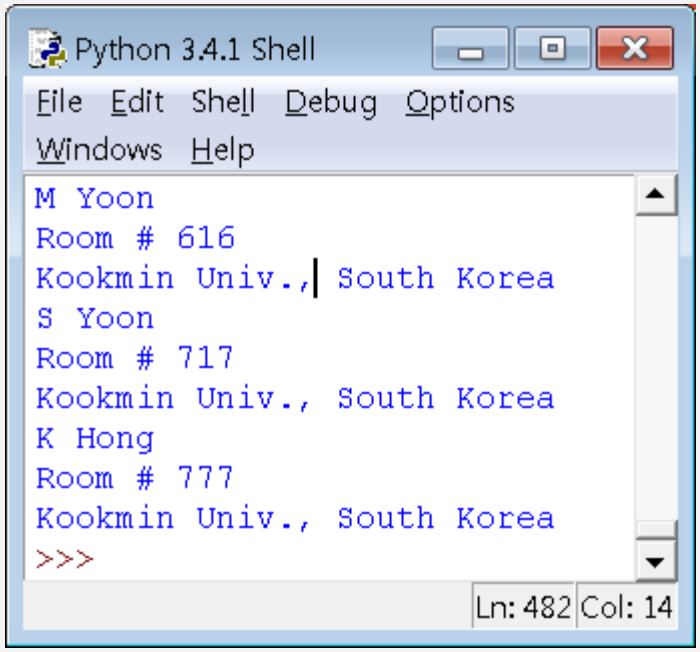
Calling a function

- 함수는 호출될 때, 독자적 메모리 공간이 할당됨
  - 스택(Stack)에서 할당
  - 함수 내부에서 정의된 변수(local variable)는 이 메모리 공간을 사용함
    - 인자(매개변수) 포함
- 함수가 종료될 때, 할당되었던 메모리 공간은 소멸됨
  - 따라서 이 메모리 공간을 사용하고 있던 모든 변수 및 인자도 동시에 소멸됨

Passing arguments to a function

- 인자(argument) 넘기기
  - 매개변수(parameter)





Passing arguments to a function

- 인자(argument) 넘기기
  - 함수 내에서 변경된 인자는 함수가 끝나고 반영되지 않음
  - 단, mutable 데이터는 예외
    - Mutable data: list, dictionary, set

Python 3.4.1: 5.1example5.py - C:/Users/mkyoo...

File Edit Format Run Options Windows Help

```
# Define a function
def printPerson(name, num):
    name = "S Lee"
    num=618
    print("In the func.:", name, num)

# main program
pName = "M Yoon"
pNum = 616
print("Before the func.:", pName, pNum)
printPerson(pName, pNum)
print("After the func.:", pName, pNum)
```

Ln: 5 Col: 41

Python 3.4.1 Shell

File Edit Shell Debug Options Windows Help

```
>>>
Before the func.: M Yoon 616
In the func.: S Lee 618
After the func.: M Yoon 616
>>>
```

Ln: 551 Col: 4

Passing arguments to a function

- 인자(argument) 넘기기
  - 함수 내에서 변경된 인자는 함수가 끝나고 반영되지 않음
  - 단, mutable 데이터는 예외
    - Mutable data: list, dictionary, set

Python 3.4.1: 5.1example6.py - C...

File Edit Format Run Options Windows Help

```
# Define a function
def mutableArgument(names):
    names.append("Bob")
    names.sort()
    print("In the func.:", names)

# main program
pNames = ["Dave", "Alice"]
print("Before the func.:", pNames)
mutableArgument(pNames)
print("After the func.:", pNames)
```

Ln: 3 Col: 21

Python 3.4.1 Shell

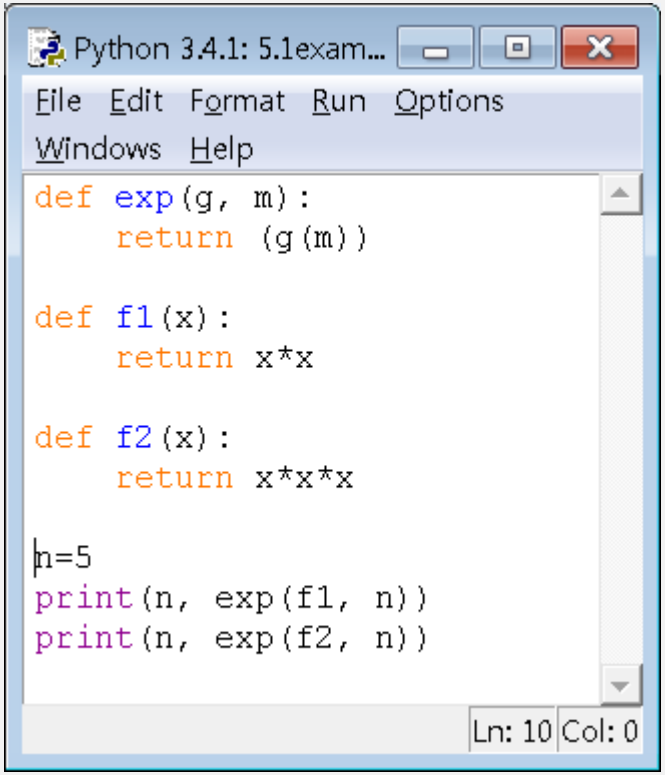
File Edit Shell Debug Options Windows Help

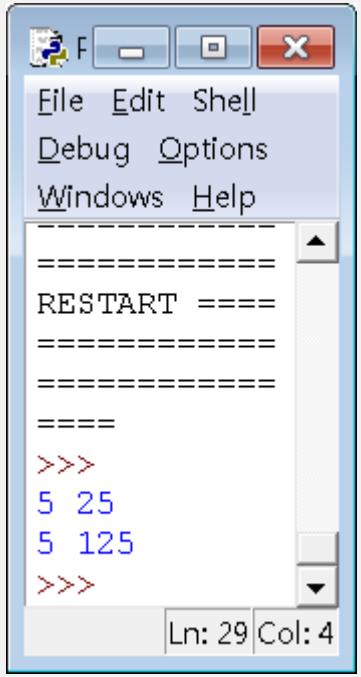
```
==========
>>>
Before the func.: ['Dave', 'Alice']
In the func.: ['Alice', 'Bob', 'Dave']
After the func.: ['Alice', 'Bob', 'Dave']
>>>
```

Ln: 574 Col: 4

Passing arguments to a function

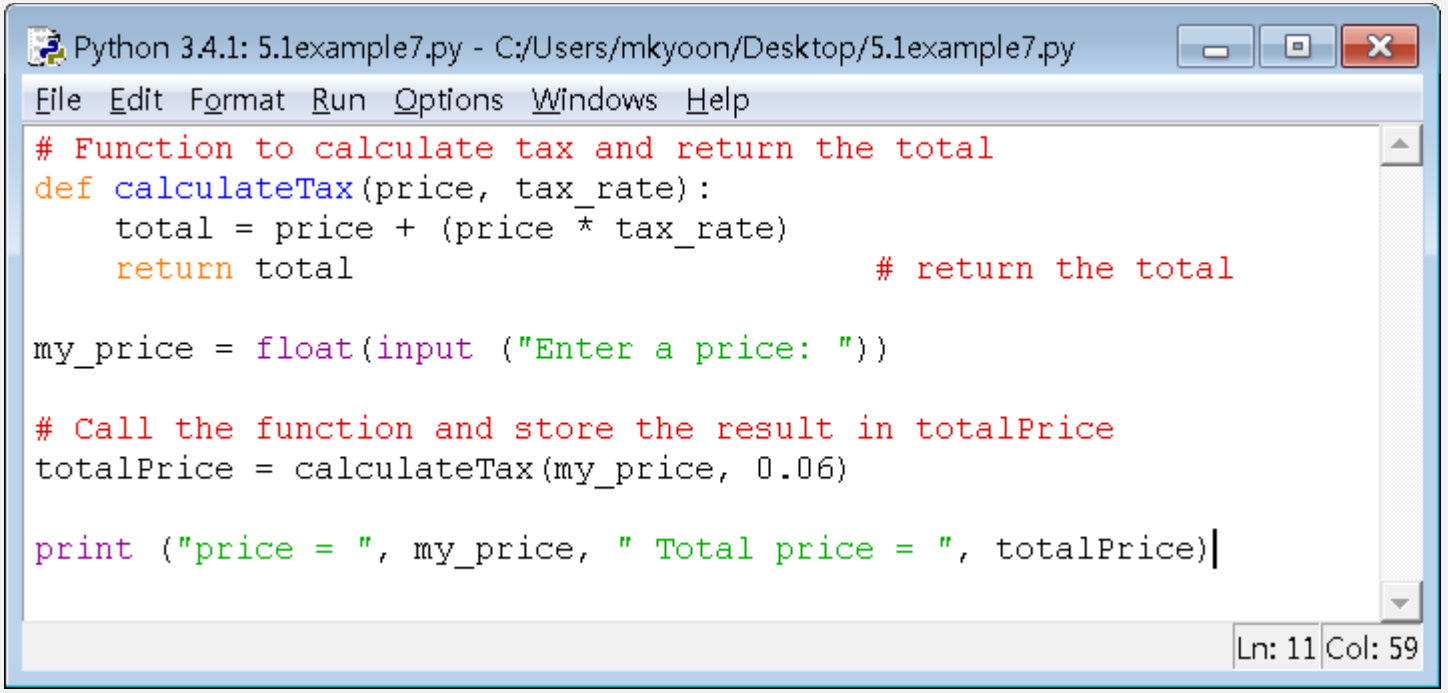
- 함수도 함수의 인자가 될 수 있음





Returning a value

- 함수 결과값 리턴
  - return (결과값)



Variable scope

- 범위(scope)에 따른 변수(variable) 구분
  - 지역(Local): 함수나 클래스 내부에서만 사용
  - 전역(Global): 프로그램 파일 안에서 사용
  - 빌트인(Built-in): 파이썬에서 특별 정의하여 사용
- 파이썬에서 변수 이름 찾는 순서
  - 지역 → 전역 →빌트인
  - 그래도 존재하지 않으면 에러 발생
- Local/global 변수 접근 에러
  - 지역 변수를 해당 함수(클래스) 외부에서 사용하려는 경우
  - 전역 변수의 값을 함수(클래스)에서 변경하려는 경우
    - 예외: global로 선언해주면 가능

Variable scope

- Local/global 변수 접근 에러
  - 함수(클래스)에서 global 변수 접근 가능

Python 3.4.1: 5.1example8.py - C:/Users/mkyoon/Desktop/5.1example8.py

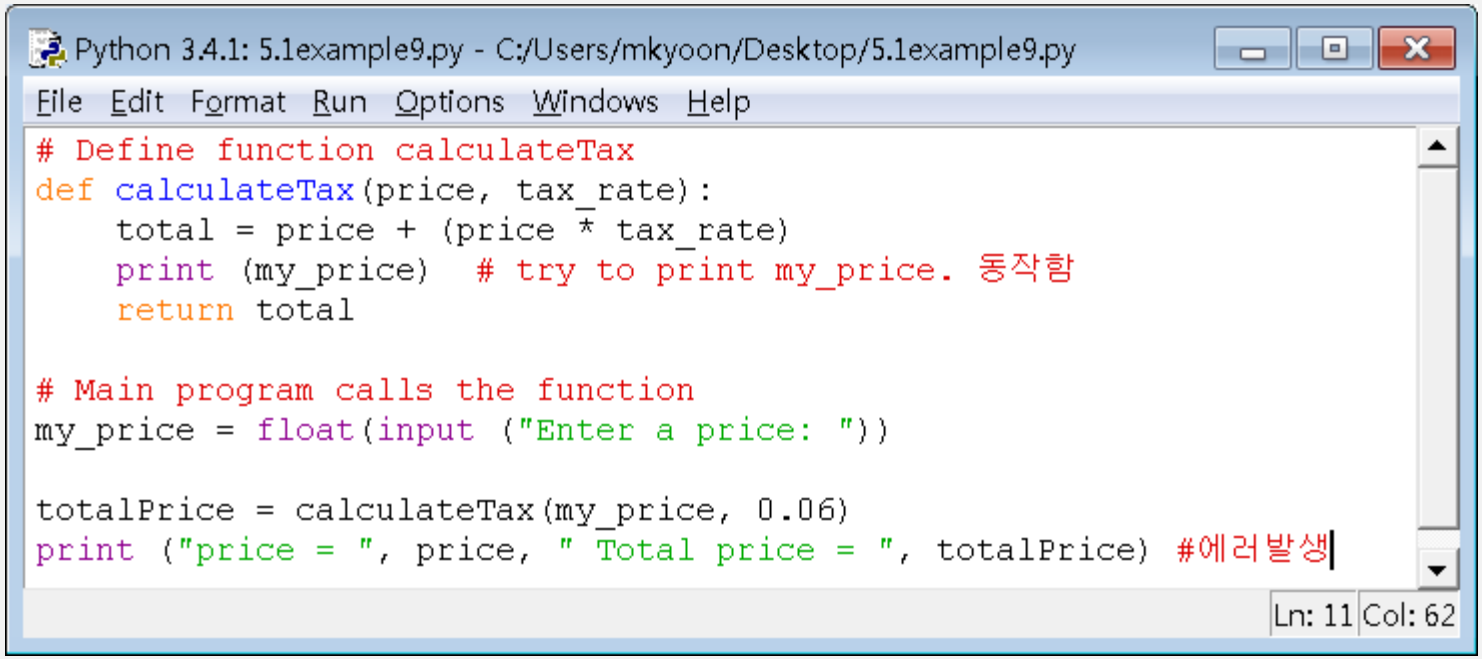
File Edit Format Run Options Windows Help

```
# Define function calculateTax
def calculateTax(price, tax_rate):
    total = price + (price * tax_rate)
    print (my_price)  # try to print my_price. 동작함
    return total

# Main program calls the function
my_price = float(input ("Enter a price: "))

totalPrice = calculateTax(my_price, 0.06)
print ("price = ", my_price, " Total price = ", totalPrice)
```

Ln: 11 Col: 59

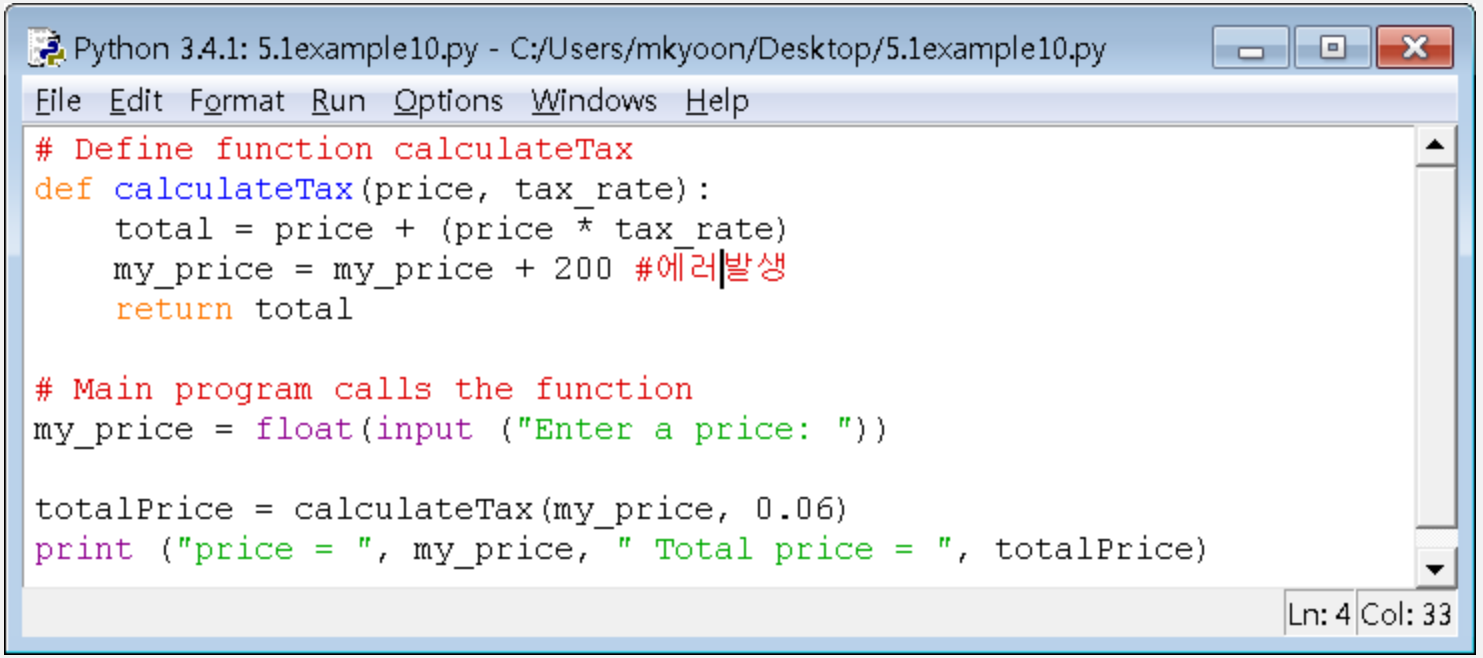
Variable scope

- Local/global 변수 접근 에러
  - 외부에서 함수(클래스)의 지역 변수 접근 에러



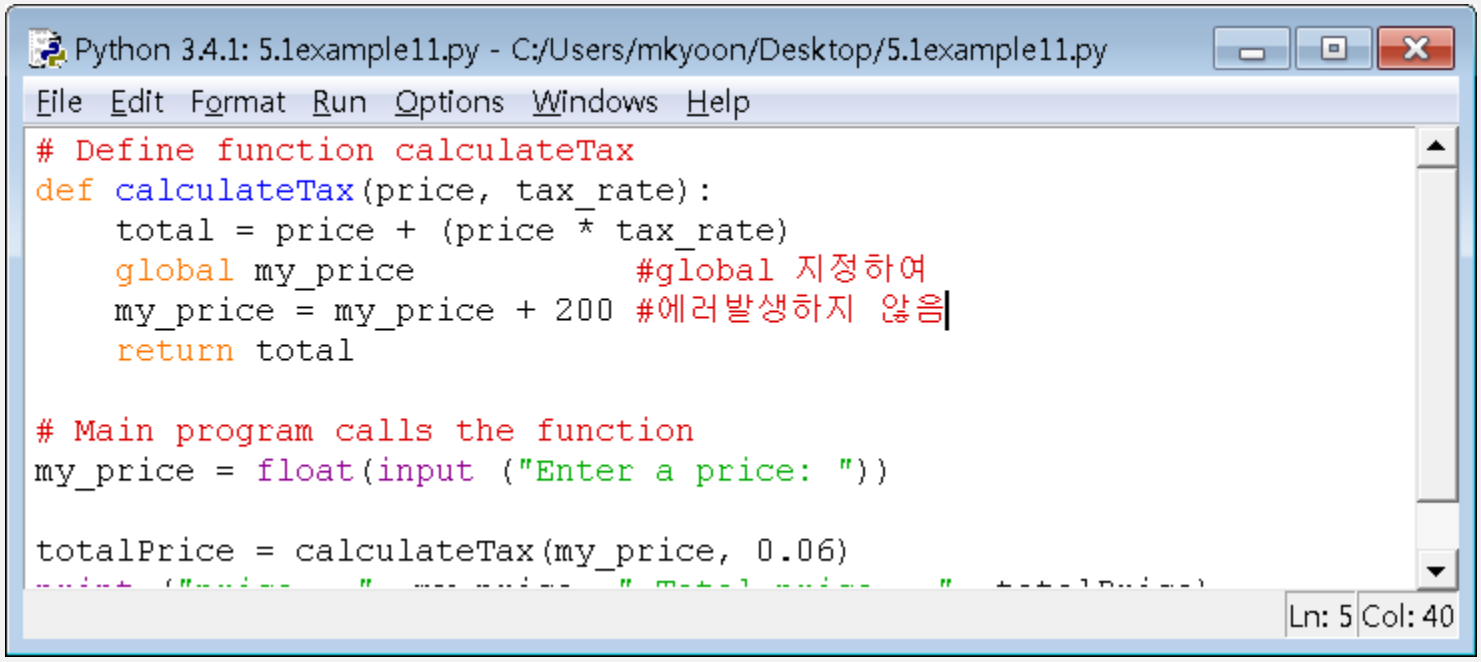
Variable scope

- Local/global 변수 접근 에러
  - 함수(클래스)에서 'global' 표시 없이 전역 변수 값을 변경하려 하여 에러 발생



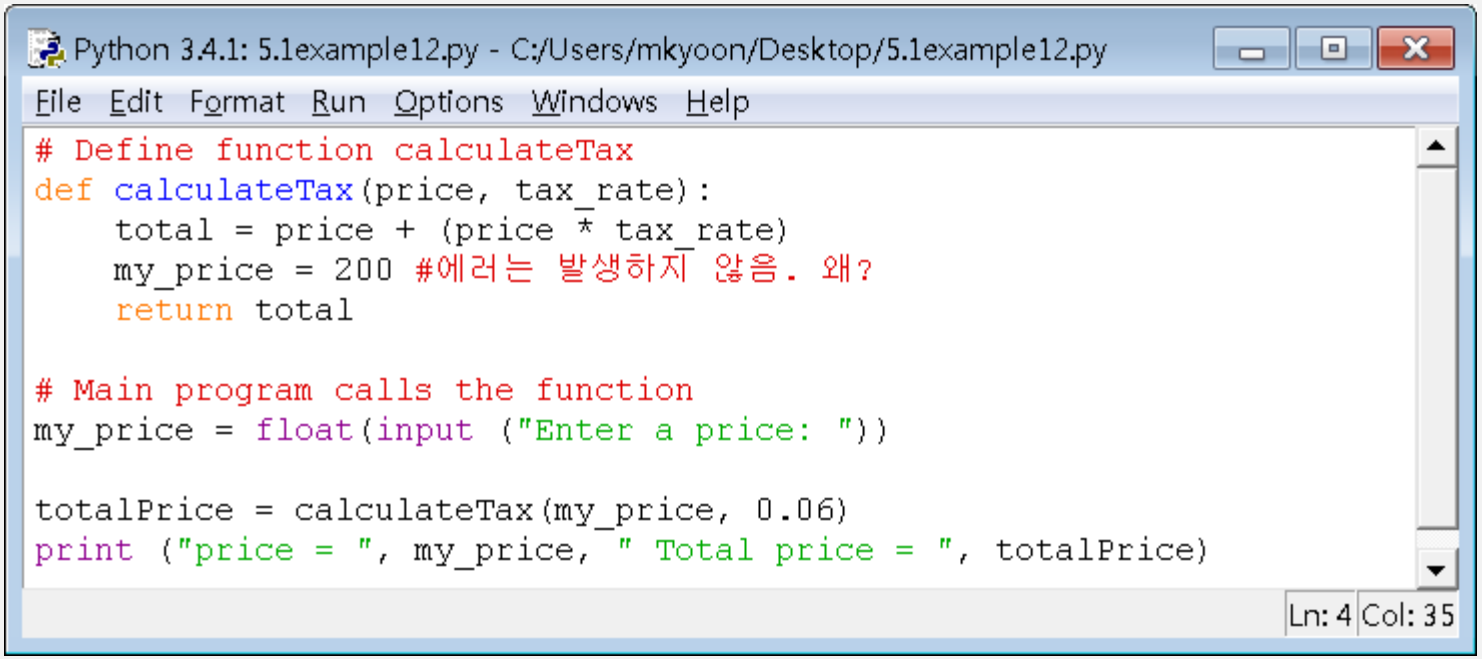
Variable scope

- Local/global 변수 접근 에러
  - 함수(클래스)에서 'global' 표시 하면 전역 변수 값을 변경 가능함
    - 하지만, 전역 변수를 함수(클래스)에서 변경하는 것은 좋지 않은 프로그래밍 습관임



Naming variables

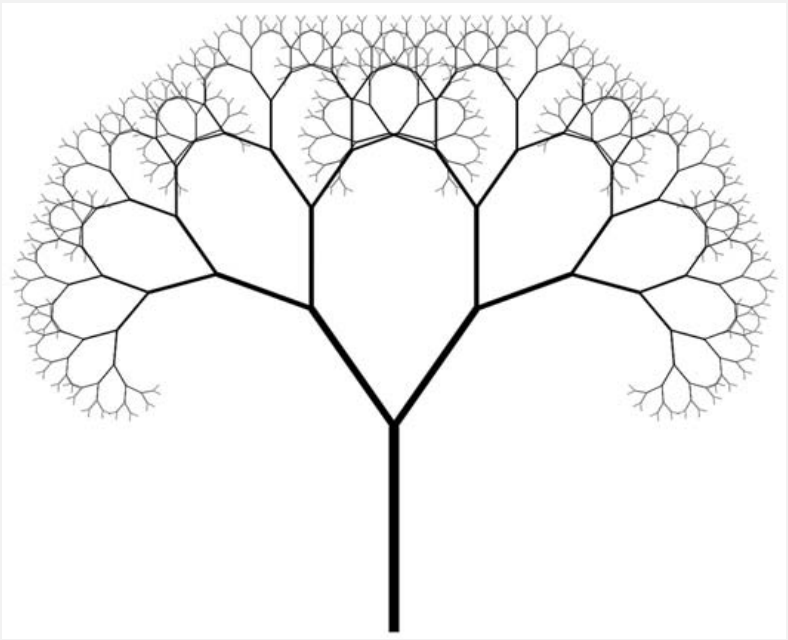
- 동일한 이름의 전역변수와 지역변수 사용은 좋지 않은 습관
  - 프로그래밍 오류 발생의 원인
  - 아래 코드는 동작하나 좋지 않은 사례



Recursive Functions

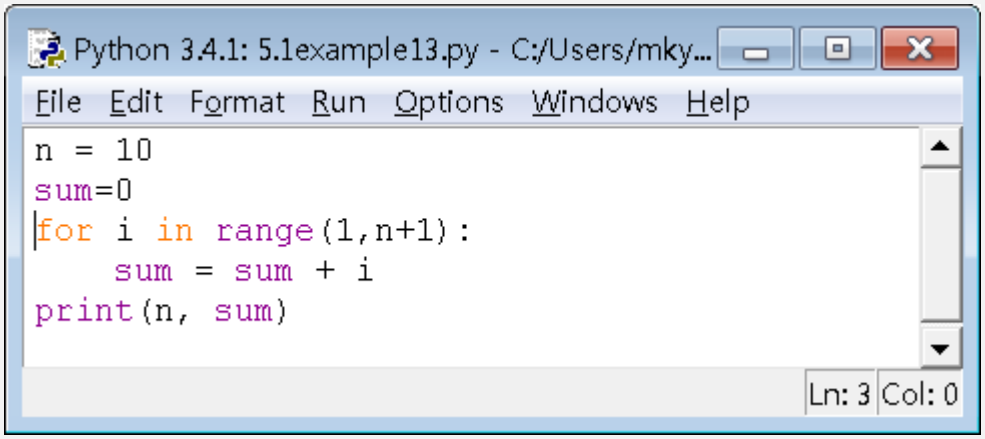
- 재귀함수(recursive function)
  - 자기 자신을 재호출하는 함수
  - 동일한 문제 해결 방식을 반복하는 상황에서 활용

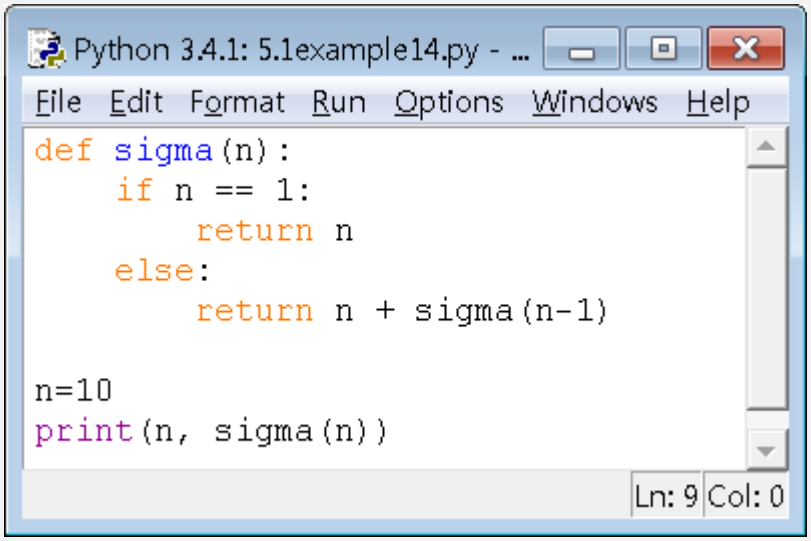
- Computer science 분야에서 흔하게 발생
  - 1부터 n까지 합산
    - 1부터 (n-1)까지 합산 + n
  - 피보나치 수열
    - f(n) = f(n-1) + f(n-2), f(1)=f(2)=1
  - 프랙탈(fractal)



Recursive Functions

- 재귀함수(recursive function)
  - 반복 패턴 + 종료 조건
  - 반복 함수 호출로 속도는 느림
    - 스택으로 부터 메모리 할당 및 해제 반복
  - Ex) 1부터 n까지 합산
    - 1부터 (n-1)까지 합산 + n





Recursive Functions

- 재귀함수(recursive function)

```
# 팩토리얼
def facto( n ) :
    if n == 0 :
        return 1
    else :
        return n * facto(n-1)

n = int(input("N >"))
print("{}! = {}".format(n, facto(n)))
```

재귀의
종료 조건

Recursive Functions

- 실습
  - 유클리드 알고리즘을 재귀함수로 구현하여 최대공약수를 구해보자.

```
# 유클리드 알고리즘을 이용한 최대공약수 구하기
def gcd( a , b ) :
    if b == 0 :
        return a
    else :
        return gcd(b, a % b)

a = int(input("a >"))
b = int(input("b >"))

print("{} 와 {}의 최대공약수는 {}입니다".format(a, b, gcd(a, b)))
```

Recursive Functions

- 실습
  - 유클리드 알고리즘을 재귀함수로 구현하여 최대공약수를 구해보자.

Recursive Functions

- 실습
  - n번째 피보나치 수를 구해보자.
  - 피보나치 수는 아래의 점화식으로 정의되는 수열이다.

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & otherwise \end{cases}$$

Recursive Functions

- 실습
  - n번째 피보나치 수를 구해보자.
  - 피보나치 수는 아래의 점화식으로 정의되는 수열이다.

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & otherwise \end{cases}$$

```
# 피보나치 수열을 구하는 함수
def fibo ( n ) :
    if n == 0 :
        return 0
    elif n == 1 :
        return 1
    else :
        return fibo(n-1) + fibo(n-2)

n = int(input("n >"))
print("{}번째 피보나치 수 : {}".format(n, fibo(n)))
```

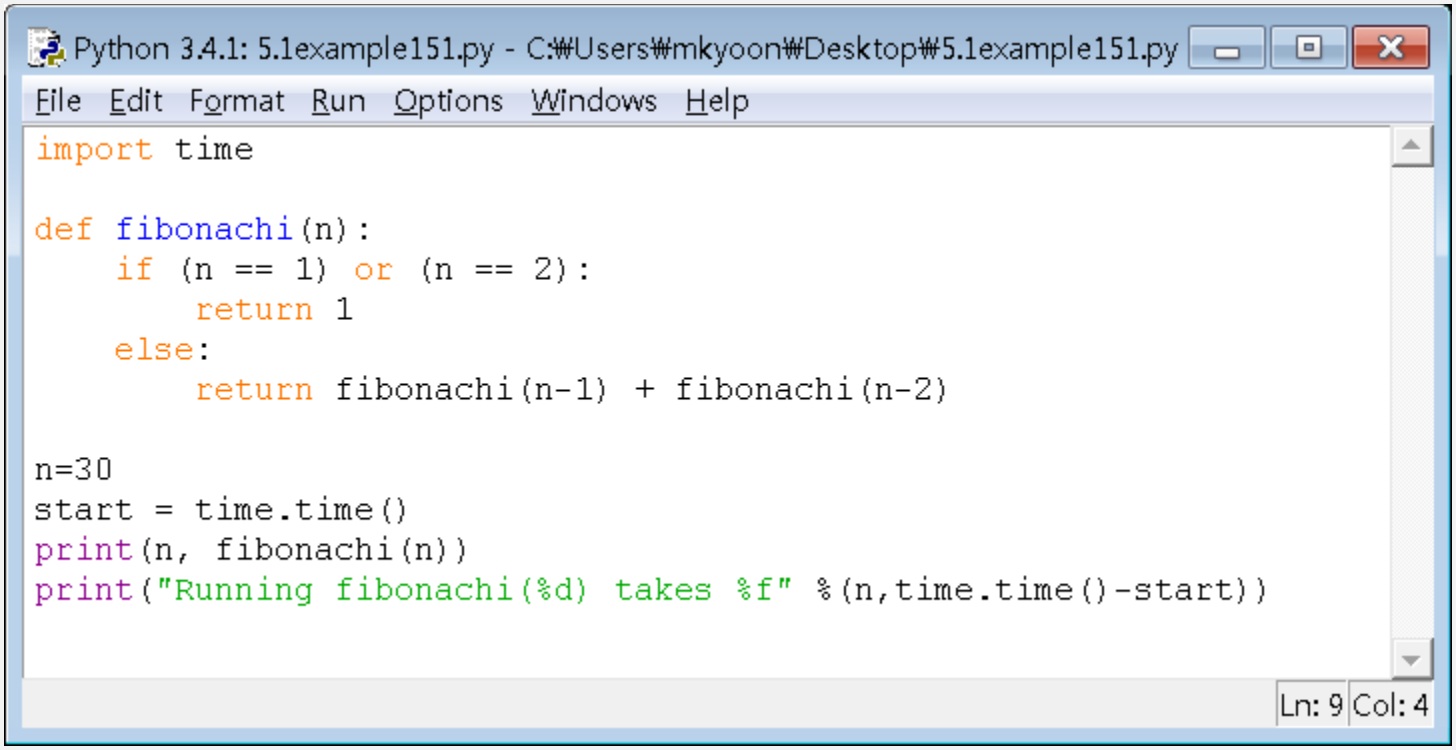
Recursive Functions

- 실습
  - n번째 피보나치 수와 피보나치 수열의 연산 횟수를 구해보자.

```
# 피보나치 수열의 연산 횟수를 구하는 함수
counter = 0

# 피보나치 수열을 구하는 함수
def fibo ( n ) :
    global counter
    counter += 1
    if n == 0 :
        return 0
    elif n == 1 :
        return 1
    else :
        return fibo(n-1) + fibo(n-2)

n = int(input("n >"))
print("{}번째 피보나치 수 : {}".format(n, fibo(n)))
print("총 {}번 연산을 하였습니다.".format(counter))
```

실습

- 피보나치 수열의 처리 시간을 구하시오

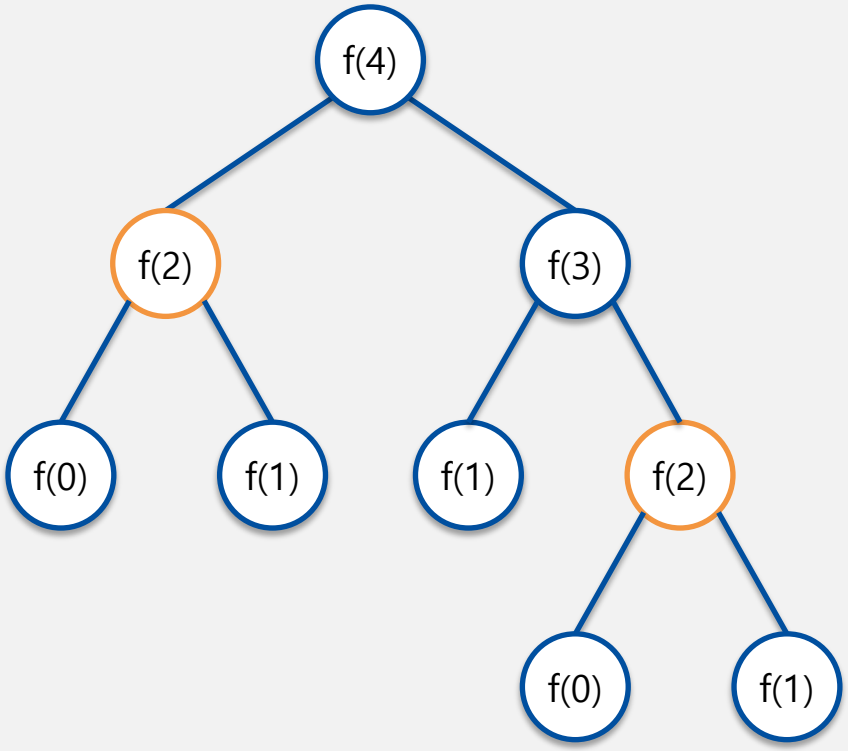


실습

- 피보나치 수열의 처리 시간이 오래 걸리는 이유는 무엇일까?
- 피보나치 수열을 재귀함수가 아닌 일반함수(for loop 사용)로 구현하고 처리 시간을 재귀함수였을 때와 비교하시오

실습

- 메모이제이션 (memoization)
  - 컴퓨터 프로그램이 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술



실습

- 메모이제이션 (memoization)
  - 컴퓨터 프로그램이 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 메모리에 저장함으로써 동일한 계산의 반복 수행을 제거하여 프로그램 실행 속도를 빠르게 하는 기술

실습

- 메모이제이션 (memoization)

```
# 피보나치 수열의 연산 횟수를 구하는 함수
counter = 0
# 피보나치 수열의 결과를 저장할 사전
memo = {}
# 피보나치 수열을 구하는 함수
def fibo ( n ) :
    global counter
    counter += 1
    if n in memo :
        return memo[n]
    if n == 0 :
        memo[n] = 0
    elif n == 1 :
        memo[n] = 1
    else :
        memo[n] = fibo(n-1) + fibo(n-2)
    return memo[n]

n = int(input("n >"))
print("{}번째 피보나치 수 : {}".format(n, fibo(n)))
print("총 {}번 연산을 하였습니다.".format(counter))
```

lambda function

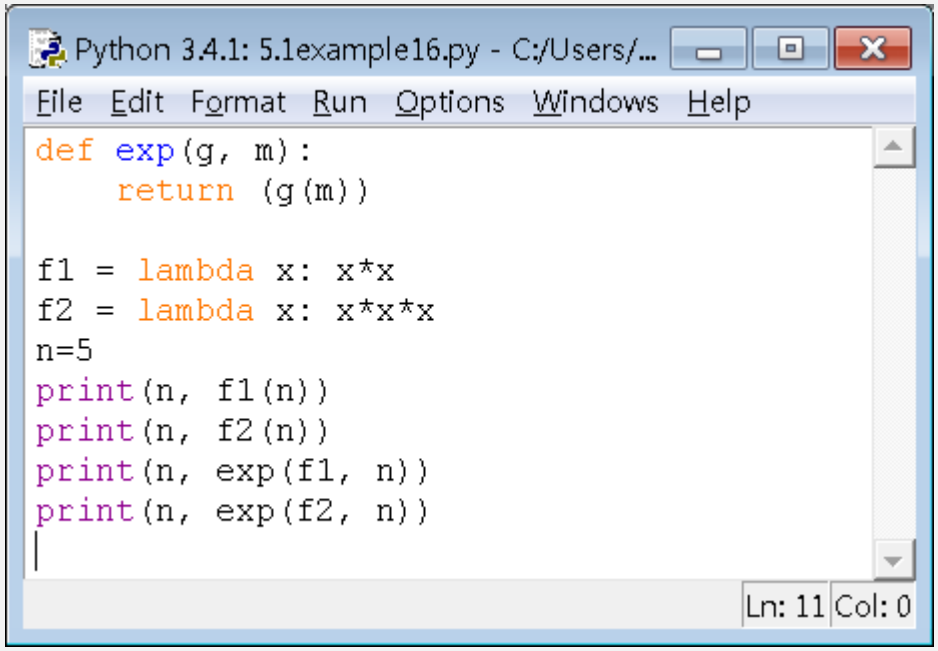
- 람다 함수 (lambda function)
  - 이름 없는 함수
  - 한 줄 함수
  - 주로 함수를 함수 인자로 넘길때 사용

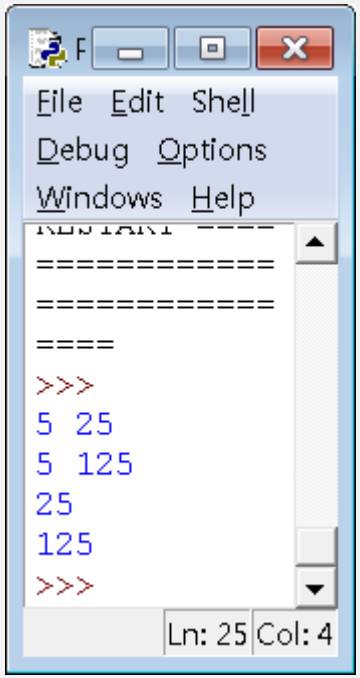
- lambda 인수: 리턴값

```
get_sum = lambda x, y : x + y

print(get_sum(1, 2))
```

lambda function

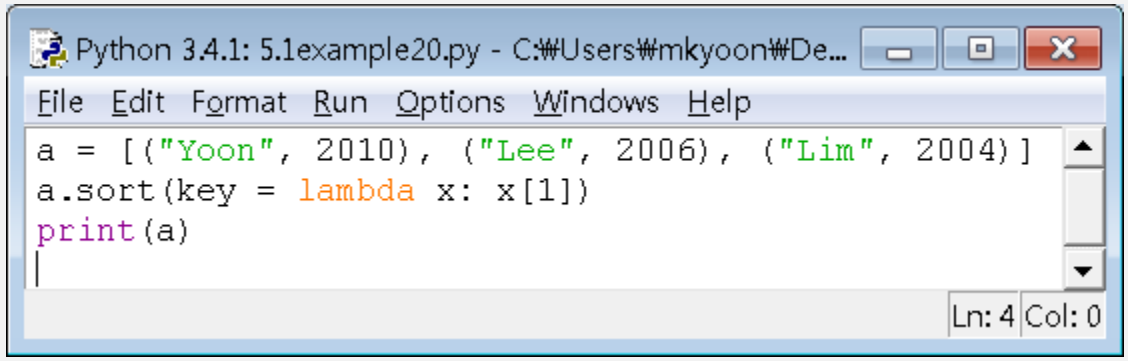
- 람다 함수 (lambda function)
  - 이름 없는 함수
  - 한 줄 함수
  - 주로 함수를 함수 인자로 넘길때 사용

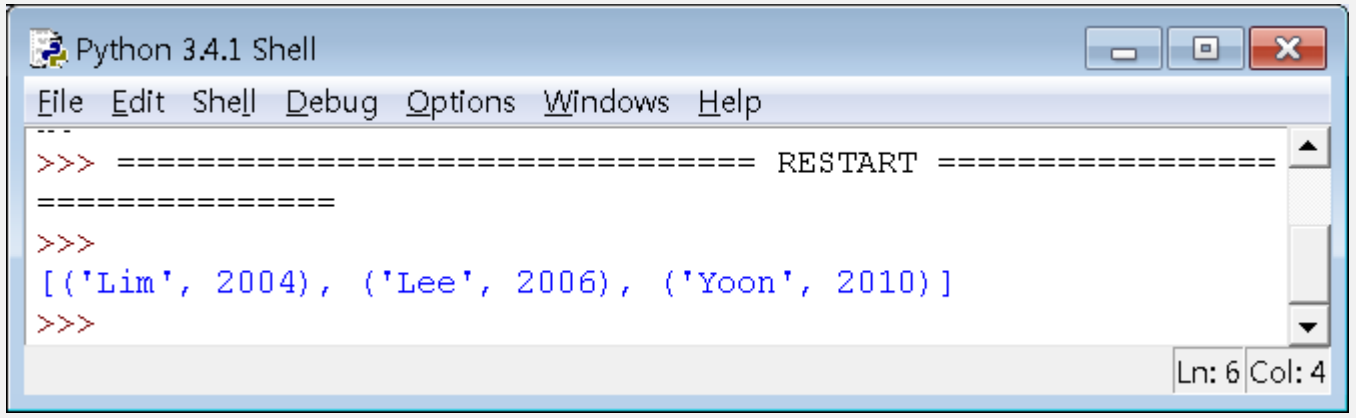




lambda function

- 람다 함수 (lambda function)
  - 예: List 정렬
    - Lambda 함수 정의: 튜플 (tuple) 두 번째 값으로 정렬





내장 함수

- 파이썬에서 기본적으로 제공하는 함수

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
asci()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

https://docs.python.org/3/library/functions.html

내장 함수

- abs(x)
  - 절대값을 반환 하는 함수
  - 복소수라면 제곱을 한 다음, 루트를 한 값을 리턴 한다.

```
# abs 는 절대 값을 반환 한다.
print(abs(-3))
# 복소수 x + yj 인경우 sqrt(x*y + y*y)의 값을 반환 한다.
print(abs(4 + 3j))
```

내장 함수

- max()
  - 인자 값 중 최대값을 반환 한다.
- min()
  - 인자 값 중 최소값을 반환 한다.

```
numbers = [91, 7, 1, 18, 29, 66, 89, 41, 96, 32]

print("MAX :", max(numbers))
print("MIN :", min(numbers))
```

내장 함수

- float()
  - 문자열 혹은 정수를 실수로 바꾼다.

```
str_pi = "3.141592653589793"
str_e = "2.718281828459045"

print(str_pi + str_e)

float_pi = float(str_pi)
float_e = float(str_e)

print(float_pi + float_e)
```

내장 함수

- int()
  - 문자열 혹은 실수를 정수로 바꾼다.

```
str_han_birthday = "970203"
str_heo_birthday = "960913"

print(str_han_birthday + str_heo_birthday)

int_han_birthday = int(str_han_birthday)
int_heo_birthday = int(str_heo_birthday)

print(int_han_birthday + int_heo_birthday)
```

내장 함수

- enumerate(iterable, start = 0)
  - 시퀀스 객체를 입력 받아, enumerate 객체로 반환한다.
  - enumerate 객체는 첫 번째 요소로 번호, 두 번째 요소로 번호에 해당되는 값을 가진다.

```
members = [ "김성규", "장동우", "남우현", "이성열", "엘", "이성종" ]

for number, name in enumerate(members, start=1) :
    print("{} 번째 멤버 : {}".format(number, name))
```

내장 함수

- sum()
  - 리스트 혹은 튜플의 합을 반환하는 함수

```
number_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(sum(number_list))
```

내장 함수

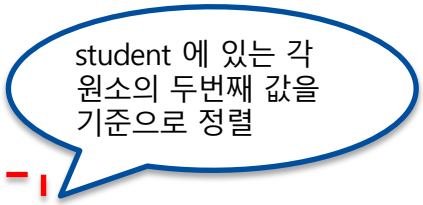
- sorted(iterable, [key], [reverse])
  - iterable 객체 안에 들어 있는 항목들로부터 정렬된 리스트를 생성하여 반환
  - key : 정렬의 기준이 되는 값
  - reverse : 정렬 결과를 뒤집을지 결정

```
number_list = [91, 7, 1, 18, 29, 66, 89, 41, 96, 32]

sorted_number_list = sorted(number_list)

print("number_list :", number_list)
print("sorted_number_list :", sorted_number_list)
```

내장 함수

- sorted(iterable, [key], [reverse])

```
student = [
    ("허준녕", 20153253, 4.2),
    ("김영재", 20153180, 3.7),
    ("한채연", 20153250, 4.5),
]

print("Before sorted :", student)
sort_by_id = sorted(student, key = lambda x : x[1])
print("Sort by id :", sort_by_id)
sort_by_grade = sorted(student, key = lambda x : x[2], reverse=True)
print("Sort by grade :", sort_by_grade)
```



내장 함수

- sorted(iterable, [key], [reverse])

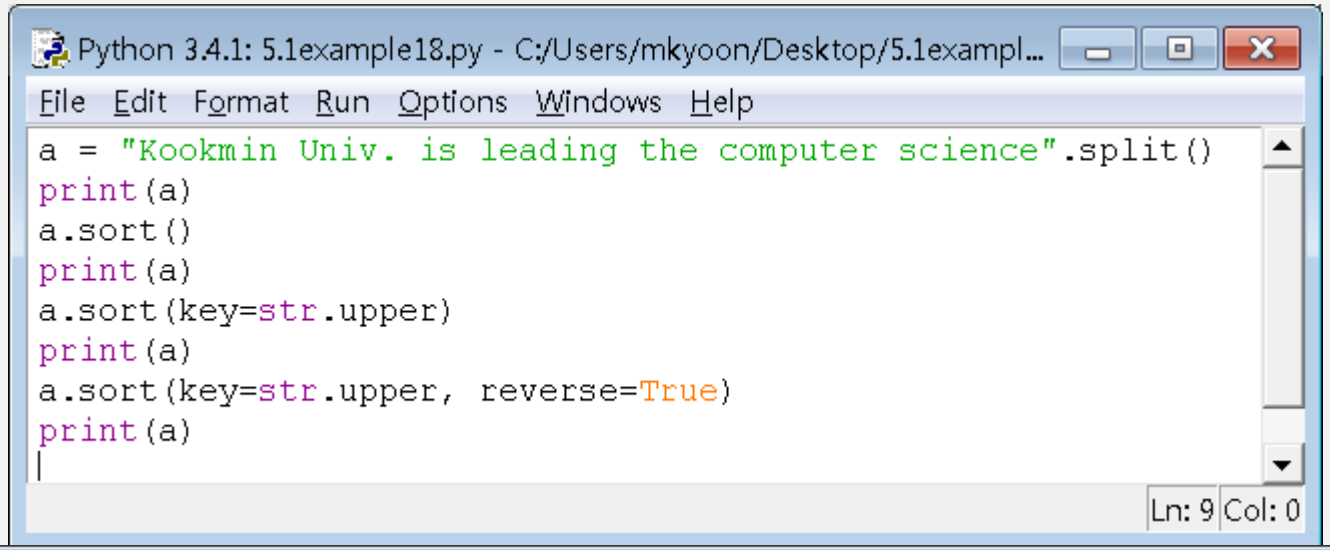
```
# 사전을 정의 합니다.
student = {
    20153180 : 3.7,
    20153250 : 4.5,
    20153253 : 4.2
}

# 학점을 기준으로 정렬해서 출력 합니다.
for key, value in sorted(student.items(), key=lambda x:x[1], reverse=True) :
    print(key, ":", value)
```

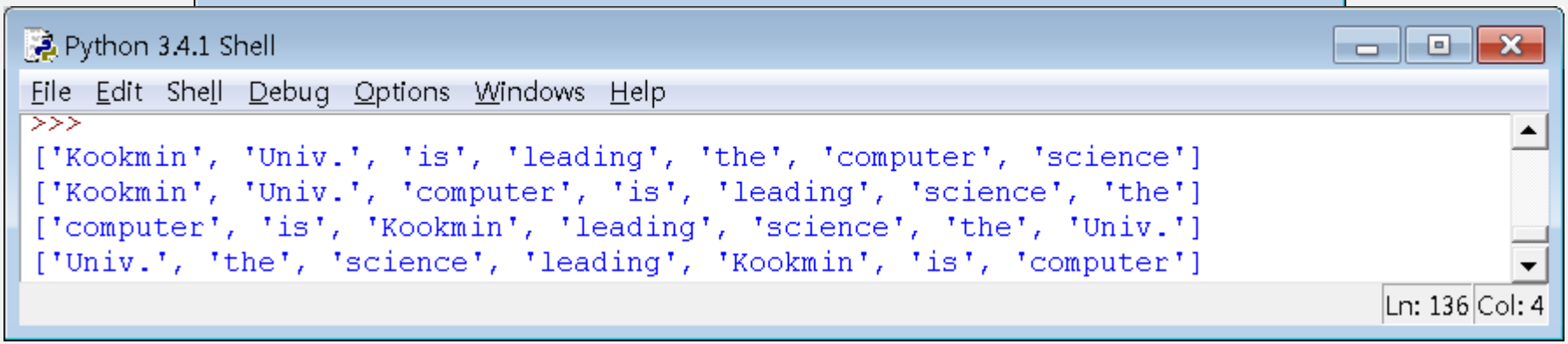
내장 함수

- sorted vs sort
- sorted(iterable, [key], [reverse])
  - 파이썬 내장 함수
  - 입력값을 정렬하고 결과를 리스트로 리턴함
- list.sort()
  - 리스트 자료형의 함수
  - 리스트 자체를 정렬함. 결과 리턴 없음.

내장 함수

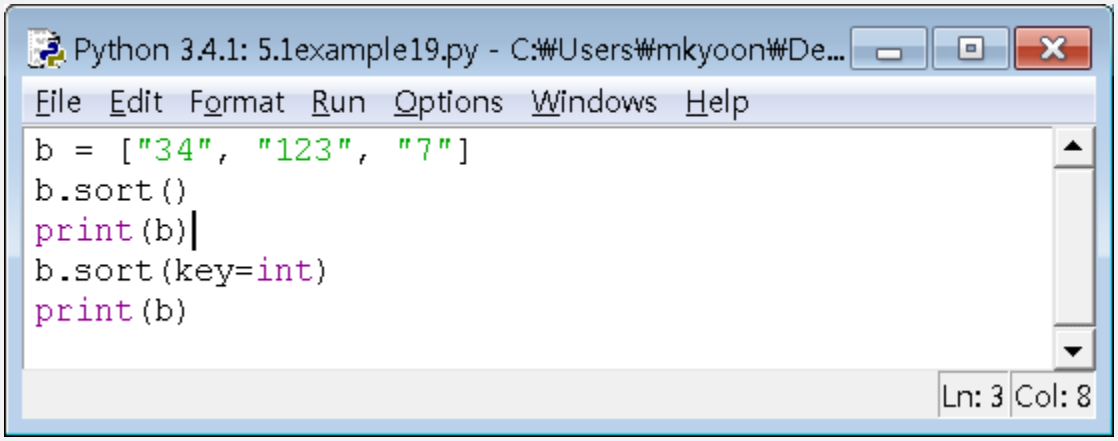
- list.sort()
  - 리스트 자료형의 함수
  - 예) key=str.upper 사용으로 대소문자 구분 없이 정렬

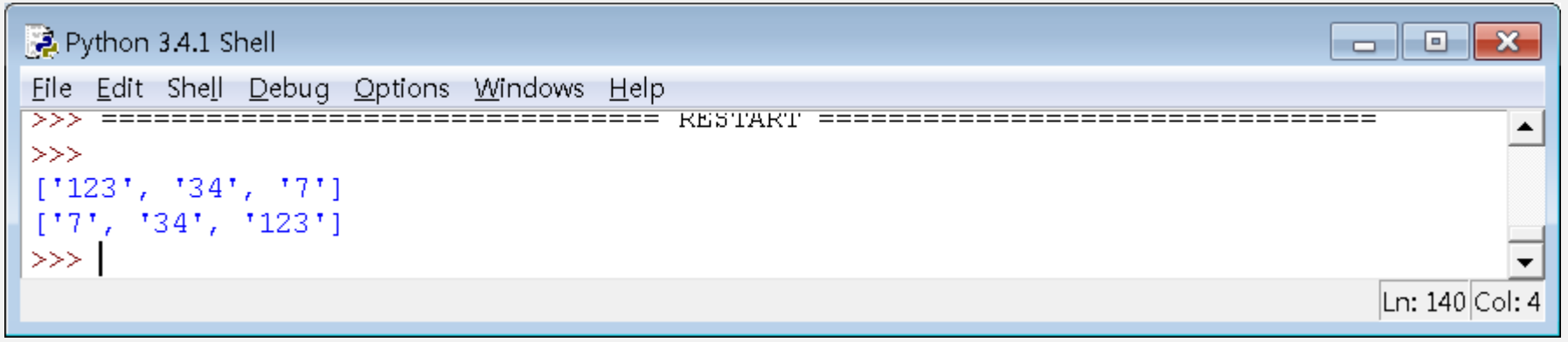




내장 함수

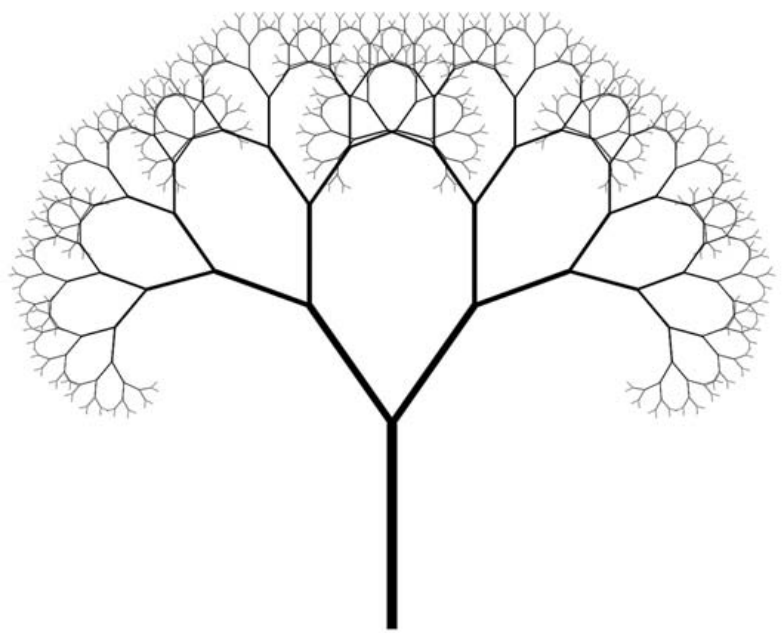
- list.sort()
  - 리스트 자료형의 함수
  - 예) int() 함수 사용으로 스트링을 정수로 변경하여 정렬





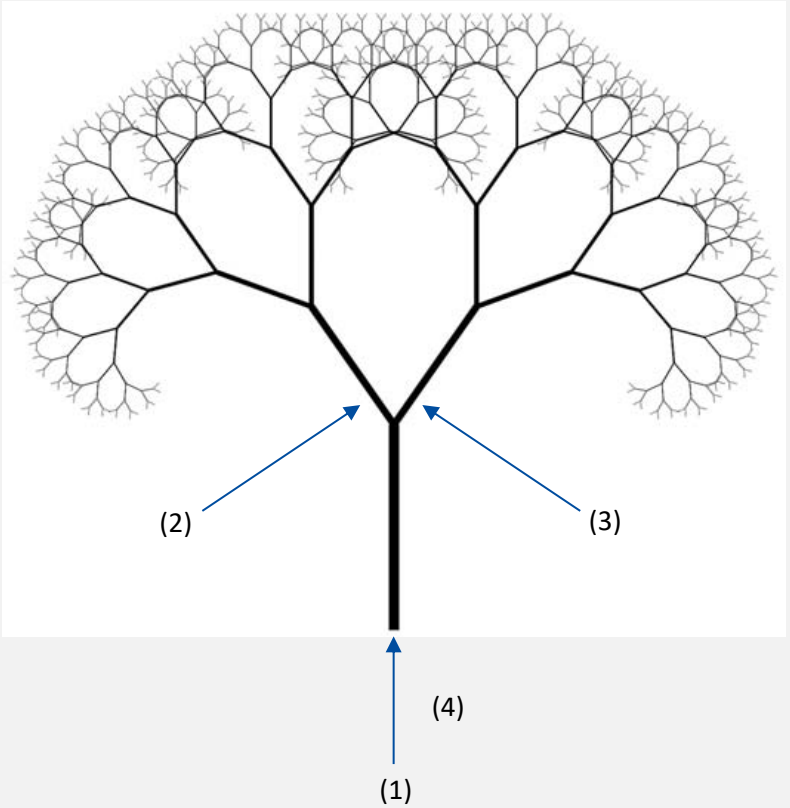
실습

- 재귀적 나무 그리기
  - 터틀 그래픽을 이용해서 다음 그림과 같은 재귀적 구조의 나무를 그리시오



실습

- 재귀적 나무 그리기
  - 재귀적 구조 찾기
    - 나무는 가운데 직선, 왼쪽 나무, 오른쪽 나무로 구성된다!
      - (1) 시작점에서 직선을 그린다
      - (2) 왼쪽 나무를 그린다
      - (3) 오른쪽 나무를 그린다
      - (4) 시작점으로 돌아온다



실습

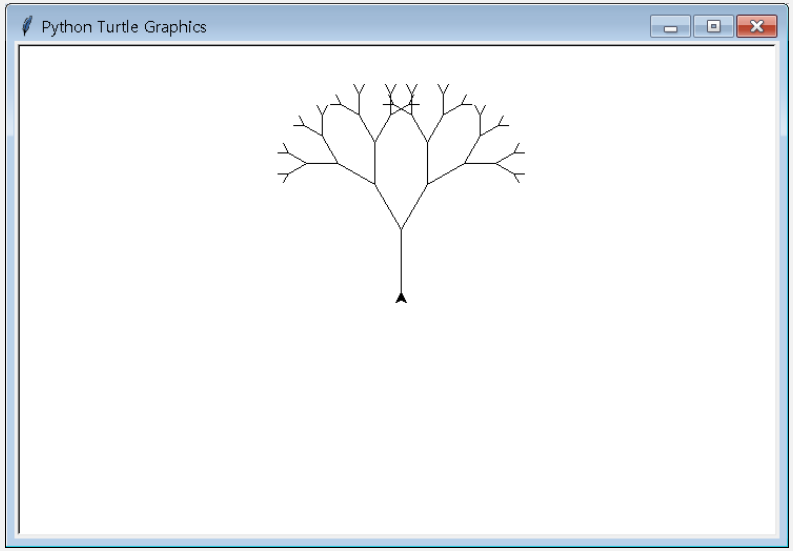
- 재귀적 나무 그리기
  - 재귀적 구조 찾기

Python 3.4.1: 5.1example152.py - C:/Users/...

File Edit Format Run Options Windows Help

```
import turtle

s = turtle.Screen()
t = turtle.Turtle()
angle = 30

def drawTree(t, lineLength):
    if (lineLength >0):
        t.forward(lineLength)
        t.left(angle)
        drawTree(t,lineLength-10)
        t.right(angle)
        t.right(angle)
        drawTree(t,lineLength-10)
        t.left(angle)
        t.backward(lineLength)

if __name__ == "__main__":
    lineLength=60
    t.left(90)
    drawTree(t, lineLength)
```

Ln: 22 Col: 0



실습

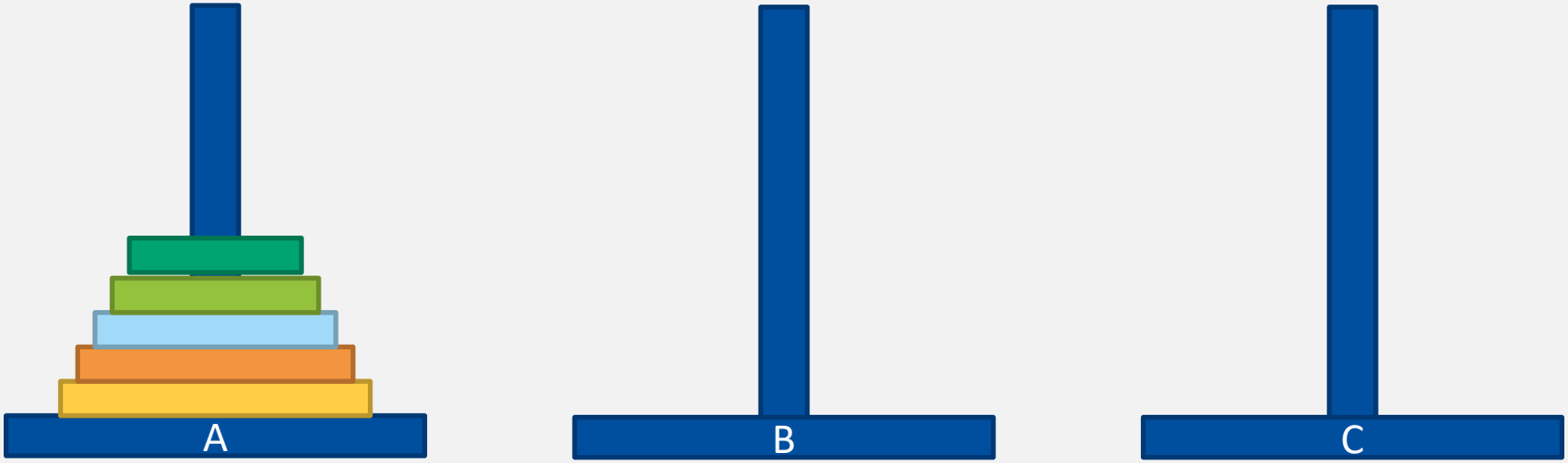
- 하노이탑
  - 하노이탑을 재귀함수로 구현하시오.
  - 원반의 개수가 n개일 때, 몇 번 원반을 옮겨야 하는가?
  - 원반의 개수가 n개일 때, 어떻게 원반을 옮겨야 하는가?

실습

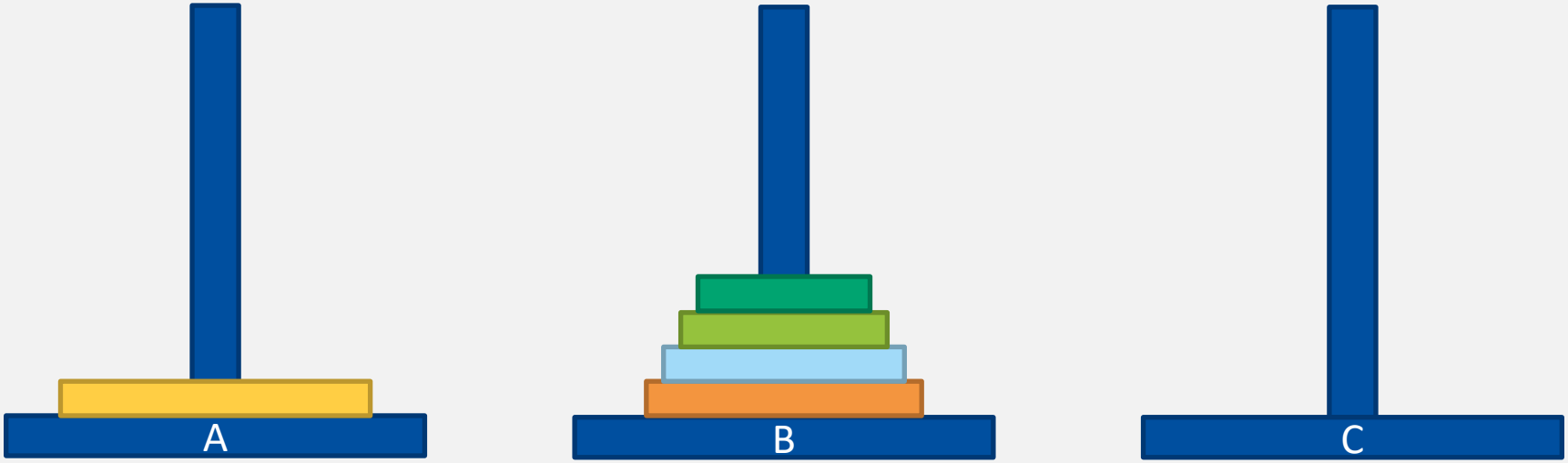
- 3 개의 장대가 있고 첫 번째 장대에는 반경이 서로 다른 n개의 원판이 쌓여 있다. 각 원판은 반경이 큰 순서대로 쌓여 있다. 이제 수도승들이 다음 규칙에 따라 첫 번째 장대에서 세 번째 장대로 옮기려 한다. 이 작업을 수행하는데 필요한 이동순서를 출력하는 프로그램을 작성하라

1. 한 번에 한 개의 원판만을 다른 탑으로 옮길 수 있다.
2. 쌓아 놓은 원판은 항상 위의 것이 아래의 것보다 작아야 한다.(중간 과정 역시 그래야함)

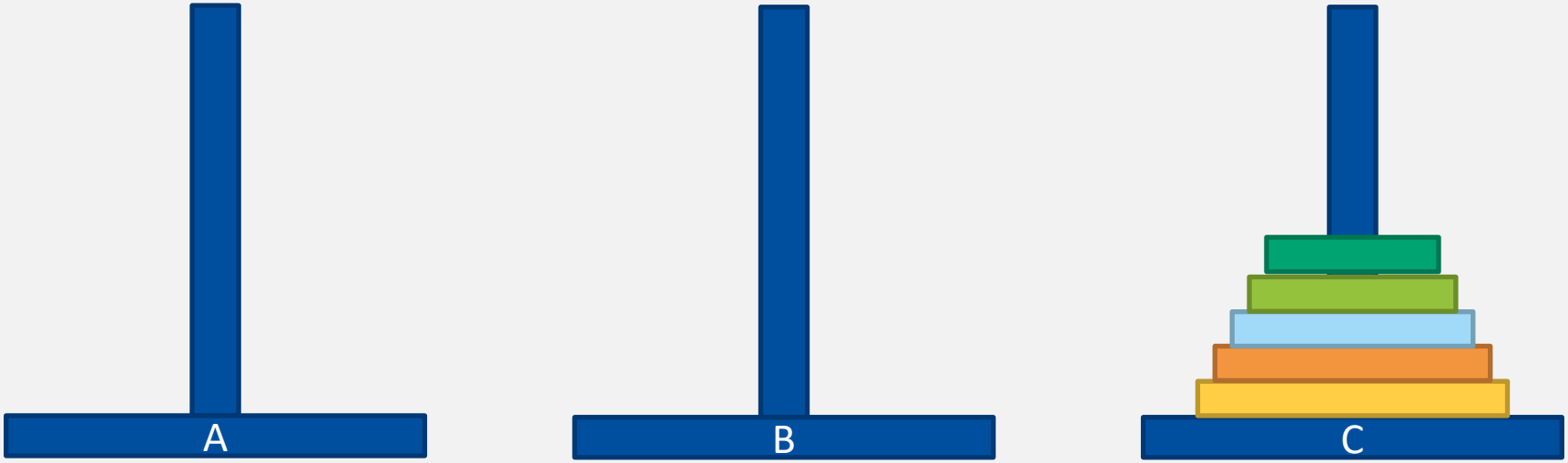
실습



실습



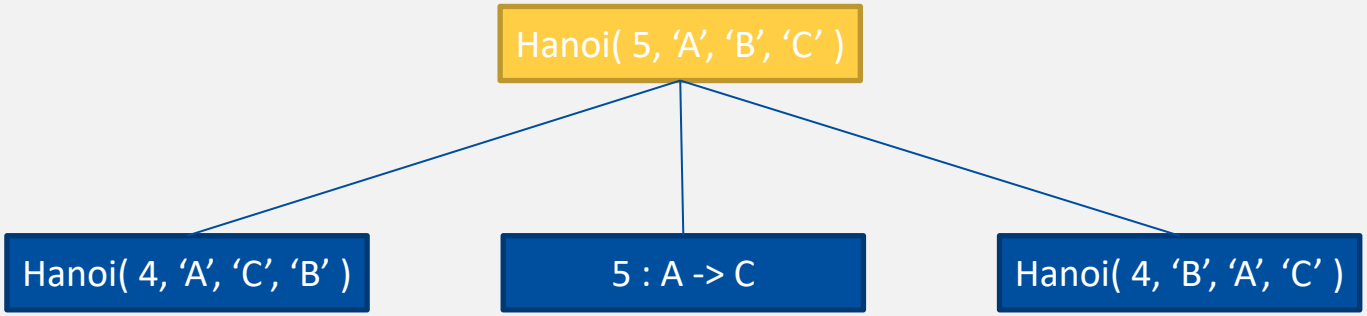
실습



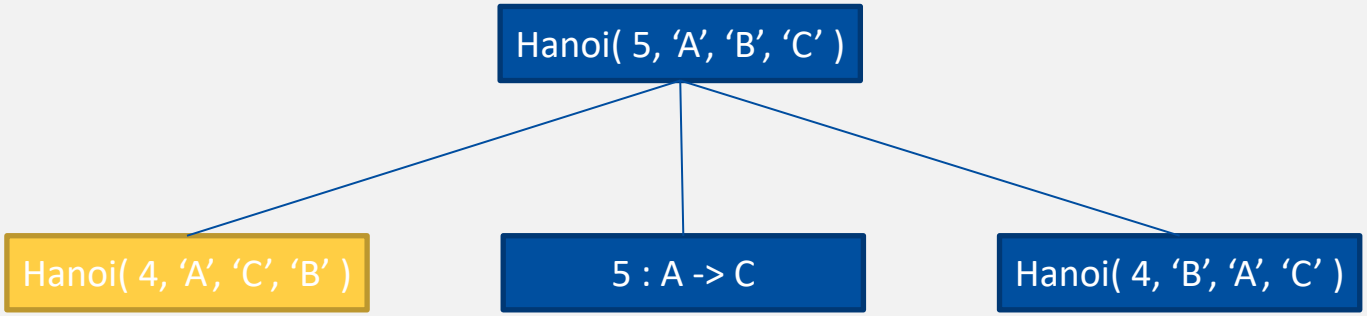
실습

- 5 개의 원판을 A->C로 옮기는 방법
  1. 초록색 ~ 주황색 원판을 A -> B 로 옮긴다.
  2. 노란색 원판을 A -> C 로 옮긴다.
  3. 다시 초록색 ~ 주황색 원판을 B -> C 로 옮긴다.
- 4개의 원판을 A->B로 옮기는 방법은?
- 4개의 원판을 B->C로 옮기는 방법은?

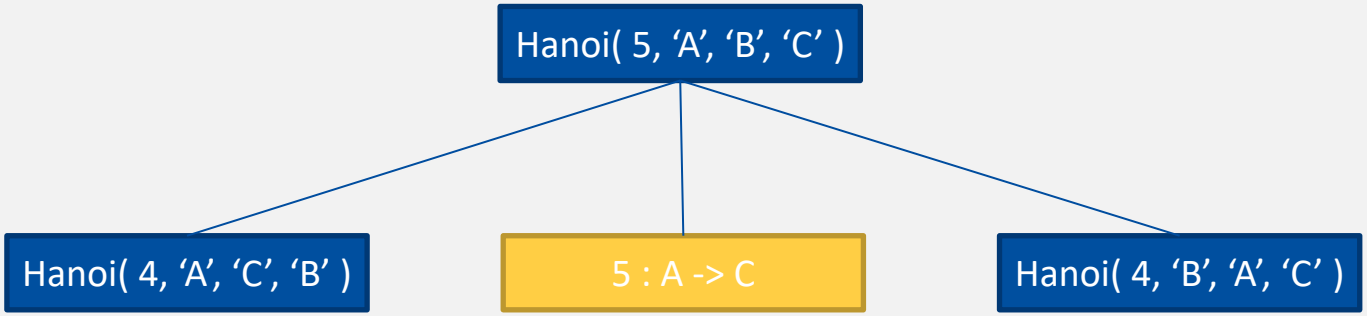
실습 – Hanoi (n, s, m, e)



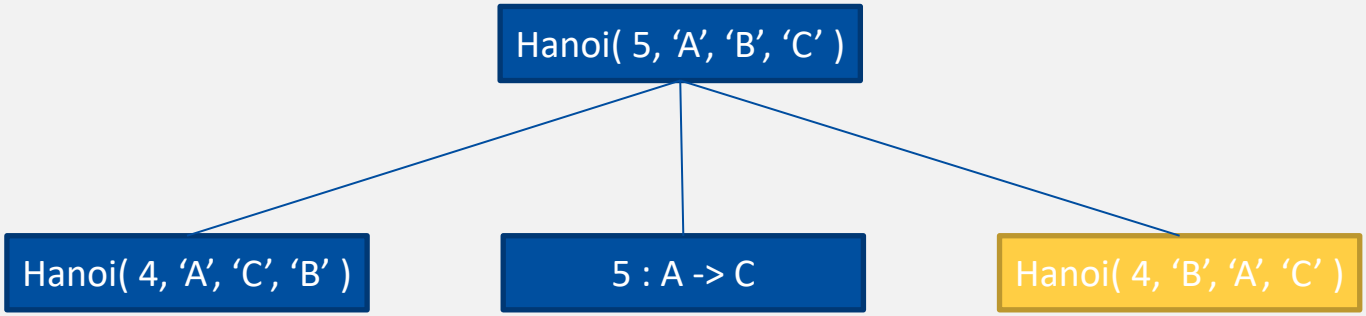
실습 – Hanoi (n, s, m, e)



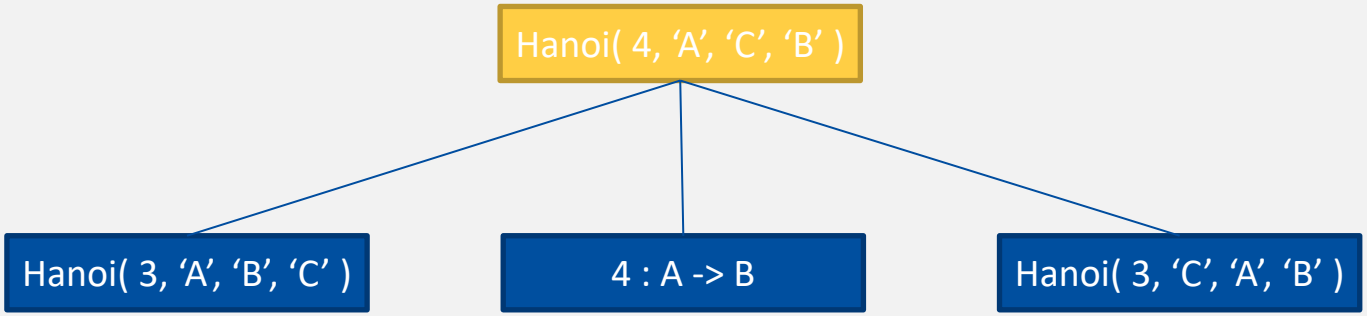
실습 – Hanoi (n, s, m, e)



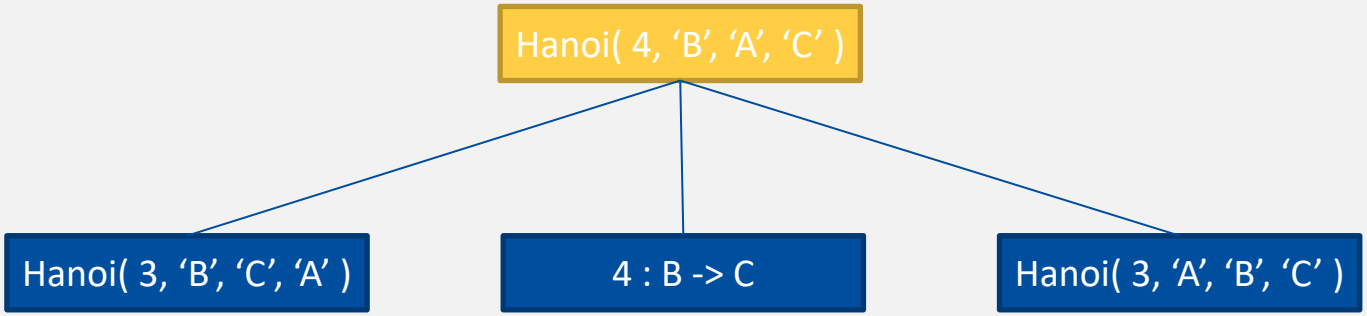
실습 – Hanoi (n, s, m, e)



실습 – Hanoi (n, s, m, e)



실습 – Hanoi (n, s, m, e)



실습 – Hanoi (n, s, m, e)

hanoi.py - C:/Users/corea/Documents/hanoi.py (3.6.1)

File Edit Format Run Options Window Help

```
def hanoi( n, start, mid, end ) :
    # 원판이 1개인 경우 바로 옮길 수 있다.
    if n == 1 :
        print("%d : %c -> %c" %(n, start, end))
    else :
        # n-1개의 원판을 start에서 mid로 먼저 옮겨 준다.
        hanoi(n-1, start, end, mid)
        # 가장 아래에 있던 n번째 원판을 start에서 end로 옮겨준다.
        print("%d : %c -> %c" %(n, start, end))
        # mid에 있는 n-1개의 원판을 end로 옮겨준다.
        hanoi(n-1, mid, start, end)

if __name__ == '__main__' :
    n = int(input("N : "))
    hanoi(n, 'A', 'B', 'C')
```

Ln: 6 Col: 36

```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/corea/Documents/hanoi.py ================
N : 5
1 : A -> C
2 : A -> B
1 : C -> B
3 : A -> C
1 : B -> A
2 : B -> C
1 : A -> C
4 : A -> B
1 : C -> B
2 : C -> A
1 : B -> A
3 : C -> B
1 : A -> C
2 : A -> B
1 : C -> B
5 : A -> C
1 : B -> A
2 : B -> C
1 : A -> C
3 : B -> A
1 : C -> B
2 : C -> A
1 : B -> A
4 : B -> C
1 : A -> C
2 : A -> B
1 : C -> B
3 : A -> C
1 : B -> A
2 : B -> C
1 : A -> C
>>>
```

숙제

- 주어진 날짜로부터 x일 후의 날이 몇 일이고 무슨 요일인지 계산해주는 프로그램을 작성하시오.
  - 예: 2015.5.1일의 1000일 후?