

함수(Function)

By 윤명근 / 박수현

수업목표

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

함수는 왜 필요한가?

비슷한 코드인데 하나로 합칠 수 있을까?



```
sum = 0;
for i in range(1, 11)
    sum += i;
```

```
sum = 0;
for i in range(1, 21)
    sum += i;
```

get_sum(1, 10)

get_sum(1, 20)

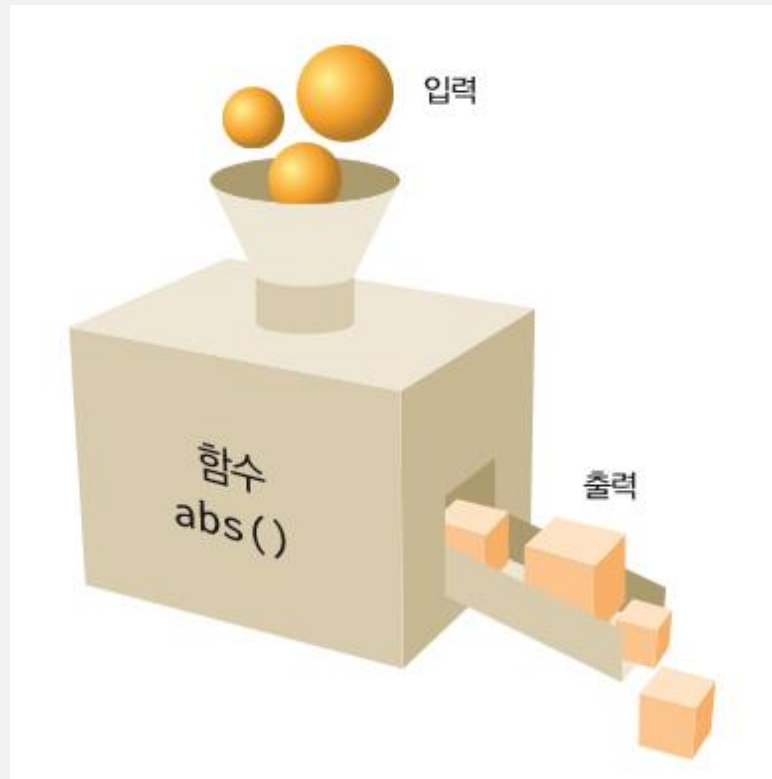
```
def get_sum(start, end)
    sum = 0;
    for i in range(start, end+1)
        sum += i;
    return sum
```

함수를 사용하면 됩니다.



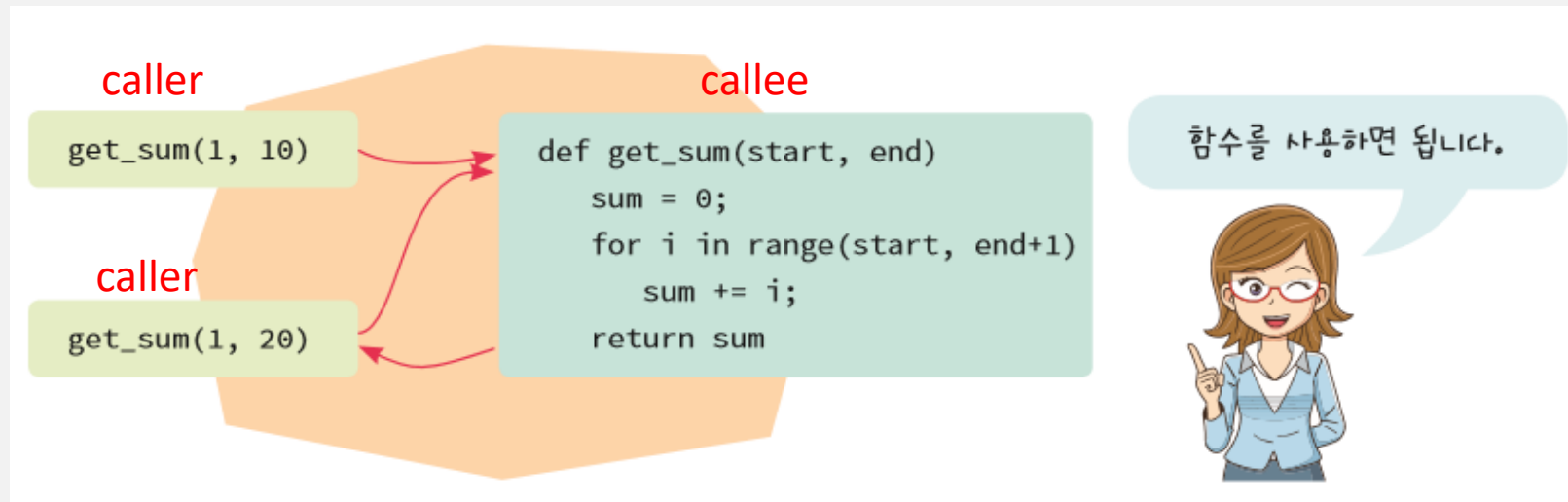
What is a function ? (1/2)

- 특정 작업을 수행하는 명령어들의 모음에 이름을 붙인 것
- 함수는 작업에 필요한 데이터(arguments)를 전달받을 수 있으며, 작업이 완료된 후에는 작업의 결과를 호출자(caller)에게 반환할 수 있음



What is a function ? (2/2)

- 어떤 일을 수행하는 code 덩어리
- 더 큰 program을 제작하는 데 사용할 수 있는 작은 조각
- 레고 블록을 이용해 무엇인가를 만드는 개념
- def keyword 사용하여 함수를 생성 (define)
- 함수의 이름을 사용해 함수를 호출(call, invocation)하여 사용



함수의 예

- `print()`
- `input()`
- `abs()`, ...
- 함수 안의 명령어들을 실행하려면 함수를 **호출(call)**하면 됨

```
>>> value = abs(-100)
>>> value
100
```

함수 사용의 장점

- Program 안에서 **중복된 코드 제거**
- 복잡한 programming 작업을 더 간단한 작업들로 분해가능 (**refinement**)
- 함수는 한번 만들어지면 다른 program에서도 **재사용(reuse)**될 수 있음
- 함수를 사용하면 **가독성(可讀性 : readability)**이 증대되고
- **유지 관리**도 쉬워짐

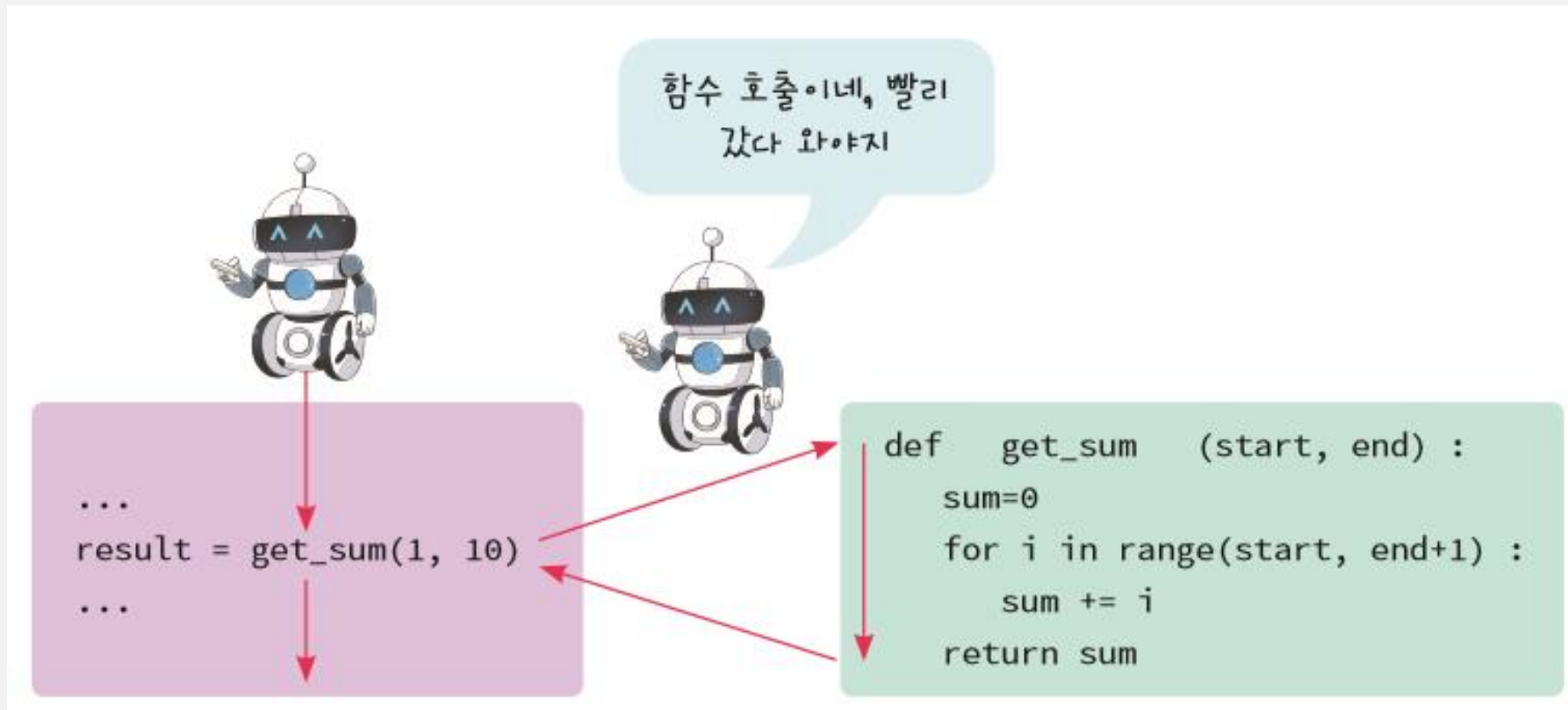
함수의 이름 (naming convention)

- 함수의 목적(주요 행위)을 설명하는 **동사** 또는 **동사+명사**를 사용
 - 함수의 이름만 보아도 어떤 역할을 하는 지 직관적으로 이해할 수 있게 naming (可讀性 증대)

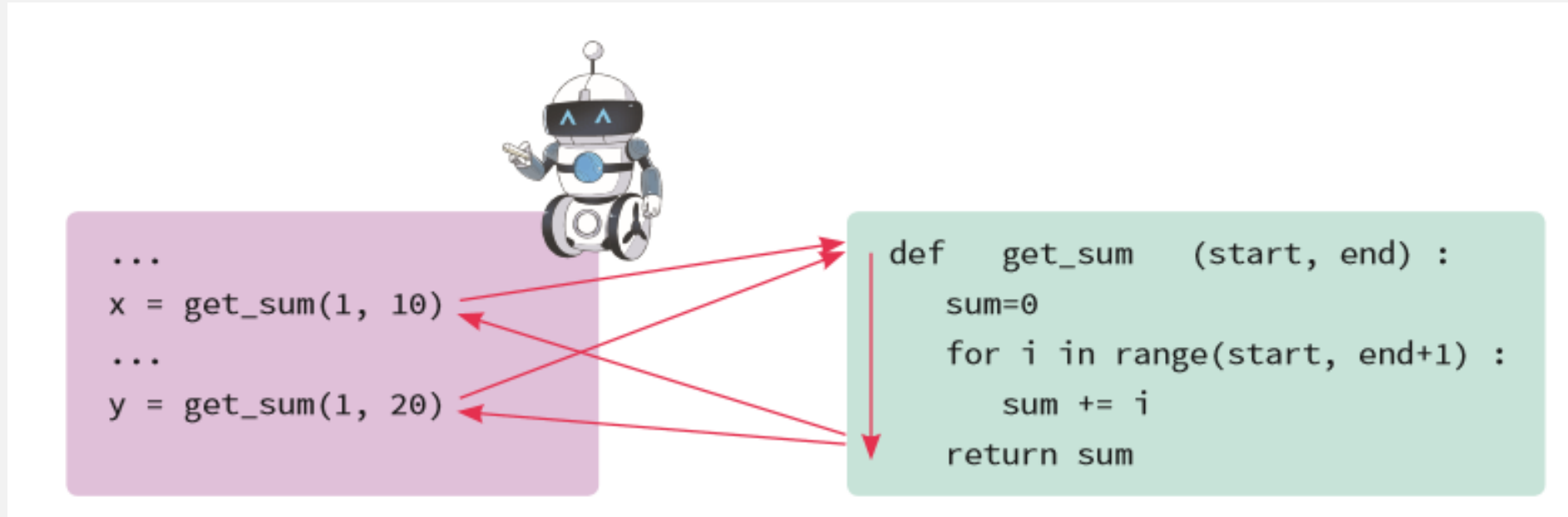
<code>square(side)</code>	// 정수를 제공하는 함수. // 정사각형 면적 구하기
<code>compute_average(list)</code>	// 평균을 구하는 함수
<code>set_cursor_type(c)</code>	// 커서의 타입을 설정하는 함수
<code>paint_the_wall(color)</code>	// 벽에 색칠하기

함수 호출

- 함수를 사용하려면 함수를 **호출(call, invocation)**하여야 함



함수는 여러 번 호출할 수 있음 – 재사용(reuse)



예제

```
def get_sum(start, end) :  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum  
  
print( get_sum(1, 10))  
print( get_sum(1, 20))
```

```
55  
210
```

```
get_sum.py
File Edit Format Run Options Window Help
1 # 함수선언
2 def get_sum(start, end) :
3     print("g-1) In get_sum({}, {})".format(start, end))
4
5     sum = 0
6     for num in range(start, end + 1) :
7         sum = sum + num
8
9     print("g-2) sum = {}".format(sum))
10    return sum
11
12 #main
13 print("\n1) main starts")
14
15 begin_val = int(input(">> 시작값을 입력하시오 : "))
16 end_val = int(input(">> 종료값을 입력하시오 : "))
17
18 print("\n2) 시작값 = ", begin_val)
19 print("    종료값 = ", end_val)
20
21 print("\n3) {} 부터 {} 까지 합은 {} 입니다."
22       .format(begin_val, end_val, get_sum(begin_val, end_val)))
23
```

1) main starts

>> 시작값을 입력하시오 : 1

>> 종료값을 입력하시오 : 100

2) 시작값 = 1

종료값 = 100

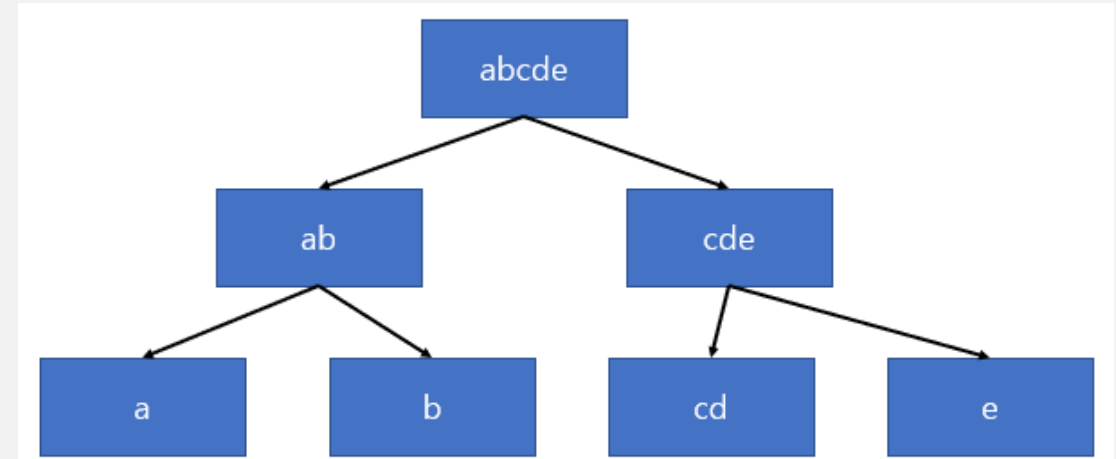
g-1) In get_sum(1, 100)

g-2) sum = 5050

3) 1 부터 100 까지 합은 5050 입니다.

What is a function ?

- Program
 - 작성하기 쉽고 관리하기 편리한 작은 조각으로 구성
 - Divide-and-conquer
- Program을 divide하는 2가지 방안
 - Module
 - program의 여러 부분들을 담은 각기 분리된 file
 - Object
 - Information hiding / encapsulation
- * Function
 - 어떤 일을 수행하는 코드 덩어리
- 함수 정의
 - `def 함수명(argument_1, .., argument_n):`



<http://blog.naver.com/PostView.nhn?blogId=qbxlvnf11&logNo=221222565505&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

```

1 # Define functions
2 # 학기초 학생들이 해야할 일들
3
4 # 수강신청
5 Capacity = 200 #기숙사 정원
6
7 def regist_class(subject1, subject2):
8     print("\nr-1) {}, {} 수강신청".format(subject1, subject2))
9     print("r-2) 실제 이 부분에서는 수강신청업무를 진행합니다.")
10    return("OK")
11
12 #기숙사 신청
13 def domitary_apply(s_id, s_name) :
14     print("d-1) {}, {} 기숙사 신청".format(s_id, s_name))
15     print("d-2) 실제 이 부분에서는 기숙사신청 절차를 실행합니다. ")
16     current_number_of_applicants = 199
17
18     if Capacity > current_number_of_applicants :
19         msg = "성공적으로 기숙사 신청을 완료했습니다."
20     else:
21         msg = "기숙사 정원을 넘어셨습니다."
22
23     return(msg)
24
25 # 신청내역 확인
26 def check_up(subject1, subject2, student_id, student_name):
27     print("c-1) 학생의 수강신청 내역, 기숙사 신청정보 출력")
28     print("c-2) 과목 1 :", subject1)
29     print("      과목 2 :", subject2)
30     print("c-3) 기숙사 신청학생 학번 :", student_id)
31     print("      기숙사 신청학생 이름 :", student_name)
32
33 # 기숙사비 정산
34 def how_much(num_of_month) :
35     print ("h-1) 20xx.3 ~ 6 기숙사비 정산")
36     room = 300000 * num_of_month # room charge
37     meal = 5000 * 2 * 30 * num_of_month
38     sum = room + meal
39
40     print("h-2) room charge =", room)
41     print("      식대 =", meal)
42     print("      총 기숙사비 =", sum)
43     return sum
44
45     print("h-3) end function") # 이 문장은 실행되지 않습니다.
46

```

```

47 # main
48 print("1) main")
49 print("2) 수강신청")
50
51 sub1 = "Python"
52 sub2 = "Java"
53 result_of_regist = regist_class(sub1, sub2)
54
55 if result_of_regist == "OK" :
56     print("3) 성공적으로 수강신청을 완료했습니다.")
57 else:
58     print("3) 학부사무실로 연락하기 바랍니다.")
59
60 print("\n4) 기숙사 신청")
61 stu_id = "20501234"
62 name = "김국민"
63 res_of_dorm_apply = domitary_apply(stu_id, name)
64 print("5)", res_of_dorm_apply)
65
66 print("\n6) 전체 신청상황을 출력합니다.")
67 check_up(sub1, sub2, stu_id, name)
68
69 print("\n7) 기숙사비 정산")
70 num_of_month = 4
71 total = how_much(num_of_month)
72
73 print("\n8) 기숙사비 :", total)
74

```

1) main
2) 수강신청

r-1) Python, Java 수강신청
r-2) 실제 이 부분에서는 수강신청업무를 진행합니다.
3) 성공적으로 수강신청을 완료했습니다.

4) 기숙사 신청
d-1) 20501234, 김국민 기숙사 신청
d-2) 실제 이 부분에서는 기숙사신청 절차를 실행합니다.
5) 성공적으로 기숙사 신청을 완료했습니다.

6) 전체 신청상황을 출력합니다.
c-1) 학생의 수강신청 내역, 기숙사 신청정보 출력
c-2) 과목 1 : Python
 과목 2 : Java
c-3) 기숙사 신청학생 학번 : 20501234
 기숙사 신청학생 이름 : 김국민

7) 기숙사비 정산
h-1) 20xx.3 ~ 6 기숙사비 정산
h-2) room charge = 1200000
 식대 = 1200000
 총 기숙사비 = 2400000

8) 기숙사비 : 2400000

What is a function ? - argument

- 가변 매개변수 함수 (1/2)
 - Python에서는 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>

```
variable_parameter.py
File Edit Format Run Options Window Help
1 def get_sum(*args) :
2     print("g-1) type of args :", type(args))
3     print("g-2) args =", args)
4
5     ret = 0
6     for each in args :
7         ret += each
8
9     print("g-3) sum =", ret)
10    return ret
11
12 #main
13 print("1) main")
14 print("Wn2) 1 ~ 10 까지의 합:", get_sum(1, 2, 3, 4, 5, 6, 7, 8, 9,10))
15 print("Wn3) 10, 20.5, 30.1 합:", get_sum(10, 20.5, 30.1))
16 print("Wn4) 10.1, 20.4, 30.6 합 :", get_sum(10.1, 20.4, 30.6))
17 print("5) End..")
18
```

```
1) main
g-1) type of args : <class 'tuple'>
g-2) args = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
g-3) sum = 55

2) 1 ~ 10 까지의 합: 55
g-1) type of args : <class 'tuple'>
g-2) args = (10, 20.5, 30.1)
g-3) sum = 60.6

3) 10, 20.5, 30.1 합: 60.6
g-1) type of args : <class 'tuple'>
g-2) args = (10.1, 20.4, 30.6)
g-3) sum = 61.1

4) 10.1, 20.4, 30.6 합 : 61.1
5) End..
```

What is a function ? - argument

- 가변 매개변수 함수 (2/2)

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

```
variable_parameter_2.py
File Edit Format Run Options Window Help
1 def get_sum(p1, p2, *args) :
2
3     print("Wng-1)", p1, p2, args)
4     ret = p1 + p2
5
6     for each in args :
7         ret += each
8
9     print("g-2) sum =", ret)
10    return ret
11
12 def main() :
13     print("m-1) 1 ~ 5 까지의 합 :", get_sum(1, 2, 3, 4, 5))
14     print("m-2) 10, 20.5 합 :", get_sum(10, 20.5))
15     print("m-3) 1.1, 2.1, 3.1, 4.1 합 :", get_sum(1.1, 2.1, 3.1, 4.1))
16
17 # main
18 print("Wn1) main")
19 main()
20
21 print("Wn2) End..")
22
```

1) main

g-1) 1 2 (3, 4, 5)

g-2) sum = 15

m-1) 1 ~ 5 까지의 합 : 15

g-1) 10 20.5 ()

g-2) sum = 30.5

m-2) 10, 20.5 합 : 30.5

g-1) 1.1 2.1 (3.1, 4.1)

g-2) sum = 10.4

m-3) 1.1, 2.1, 3.1, 4.1 합 : 10.4

2) End..

What is a function ? - argument

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

```
basic_argument.py
File Edit Format Run Options Window Help
1 # 기본 매개 변수
2 def print_my_info( n = 1 ) :
3     print("p-1) n = ", n)
4
5     for i in range(n) :
6         print(">> ", i, "번째")
7
8     print("p-2) function end")
9
10 # main
11 print("\n1) call print_my_info()")
12 print_my_info()
13
14 print("\n2) call print_my_info(3)")
15 print_my_info(3)
16
17 print("\n3) end")
18
```

```
1) call print_my_info()
p-1) n = 1
>> 0 번째
p-2) function end

2) call print_my_info(3)
p-1) n = 3
>> 0 번째
>> 1 번째
>> 2 번째
p-2) function end

3) end
```

What is a function ?

- 함수 사용
 - 인자(argument) 를 함수에 전달 가능
 - 매개변수(parameter)
 - 함수 결과값 리턴 가능

Calling a function

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

```

1 # Define a function
2 def f1():
3     print ("f1 starts.")
4     f2()
5     print ("f1 ends.")
6
7 def f2():
8     print ("f2 starts.")
9     f3()
10    print ("f2 ends.")
11
12 def f3():
13     print ("f3 starts.\n")
14     print ("f3 ends.")
15
16 def main():
17     print("main starts.")
18     f1()
19     print("main ends.\n")
20
21 # main program
22 print("1) Program start.\n")
23 main()
24 print("2) Program ends.")
25

```

1) Program start.

main starts.
f1 starts.
f2 starts.
f3 starts.

f3 ends.
f2 ends.
f1 ends.
main ends.

2) Program ends.

Calling a function

- 함수는 호출될 때, 독자적 memory 공간이 할당됨
 - **Stack**에서 할당
 - 함수 내부에서 정의된 변수(local variable)는 이 memory 공간을 사용함
 - 인자(매개변수 : parameter) 포함
- 함수가 **종료**될 때, 할당되었던 memory 공간은 **소멸**됨
 - 따라서 이 memory 공간을 사용하고 있던 모든 변수 및 인자도 동시에 소멸됨

함수정의

def FtoC(input_fahrenheit):

print("f-1) 입력된 화씨온도 =", input_fahrenheit)

temp_c = (5.0 * (input_fahrenheit - 32.0)) / 9.0;

print("f-2) 섭씨온도 =", temp_c)

print("f-3) id(input_fahrenheit) = ", id(input_fahrenheit))

print("f-4) id(fahrenheit) = ", id(fahrenheit))

print("f-5) id(temp_c) = ", id(temp_c))

return temp_c;

#main

print("1) main")

fahrenheit = float(input("f-1) 화씨온도를 입력하시오: "))

print("2) fahrenheit = ", fahrenheit)

print("3) id(fahrenheit) = ", id(fahrenheit))

FtoC() 함수를 호출한다.

print("f-4) 변경된 화씨온도 :", FtoC(fahrenheit))

#error

print("f-5) id(temp_c) = ", id(temp_c))

1) main

>> 화씨온도를 입력하시오: 32

2) fahrenheit = 32.0

3) id(fahrenheit) = 64958128

f-1) 입력된 화씨온도 = 32.0

f-2) 섭씨온도 = 0.0

f-3) id(input_fahrenheit) = 64958128

f-4) id(fahrenheit) = 64958128

f-5) id(temp_c) = 64200176

4) 변경된 화씨온도 : 0.0

Traceback (most recent call last):

File "C:\과소사\과소사-강의예제\ftoc.py", line 27, in <module>

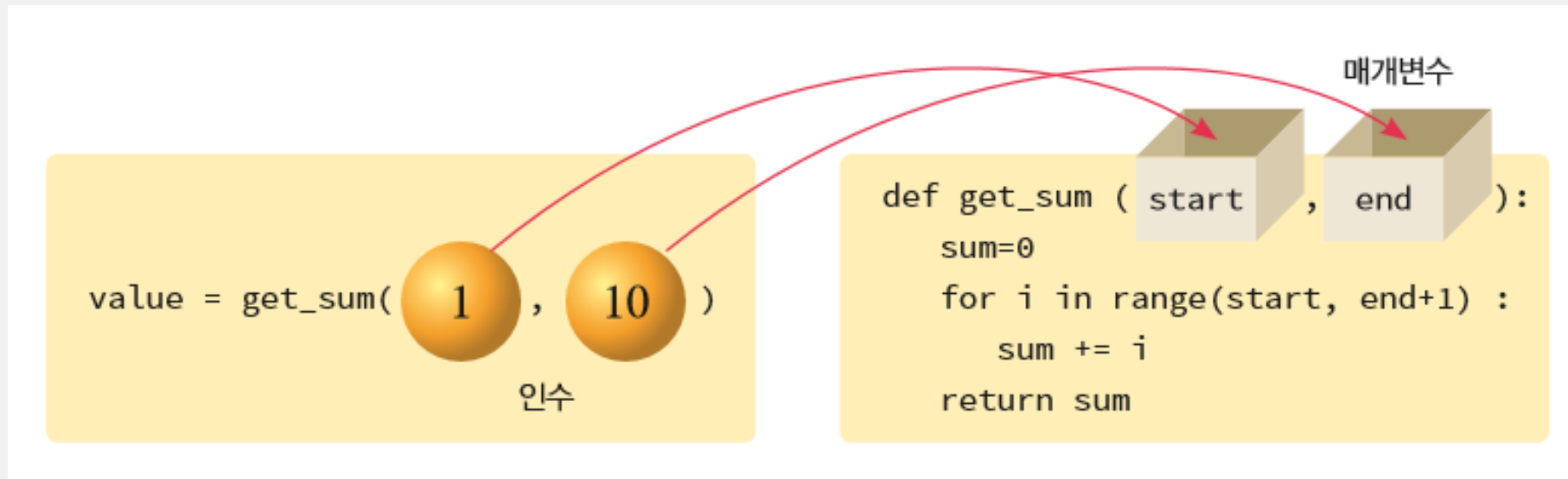
print("f-5) id(temp_c) = ", id(temp_c))

NameError: name 'temp_c' is not defined

~~~

# Passing arguments to a function

- **Actual argument**
  - 인수(argument)는 호출 program(caller)에 의하여 함수에 실제로 전달되는 값들
- **Formal parameter**
  - 매개 변수(parameter)는 callee에서 이 값을 전달받는 변수



# Passing arguments to a function

- 인자(argument) 넘기기
  - Actual argument : arguments in caller, ex) pName, pNum
  - Formal parameter : arguments in callee, ex) name, num

```
5.1example5.py
File Edit Format Run Options Window Help
1 # Define a function
2 def printPerson(name, num):
3
4     print("\np-1) formal parameters : {}, {}".format(pName, pNum))
5     name = "Soo"
6     num = 1774
7
8     print("\np-2) In the func.: {}, {}".format(name, num))
9
10
11 # main program
12 print("\n1) main")
13
14 pName = "Park"
15 pNum = 4559
16
17 print("\n2) Before call, actual arguments : {}, {}".format(pName, pNum))
18 printPerson(pName, pNum)
19
20 print("\n3) After call, actual arguments :", pName, pNum)
21 #print("\n3) After call, name, num :", name, num)
22
```

1) main  
2) Before call, actual arguments : Park. 4559  
  
p-1) formal parameters : Park, 4559  
p-2) In the func.: Soo. 1774  
  
3) After call, actual arguments : Park 4559

# Passing arguments to a function

- 함수를 호출할 때, 변수를 전달하는 2가지 방법
  - 값에 의한 호출(**call-by-value**)
    - Callee의 formal parameter 값의 변경이 caller의 actual argument값에 반영되지 않음
  - 참조에 의한 호출(**call-by-reference**)
    - Callee의 formal parameter 값의 변경이 caller의 actual argument값에 반영 됨
    - mutable 데이터 : list, dictionary, set



```
def modify1(m, nai):
    print("r-1) m =", m, " ,id(m) =", id(m))
    print("r-2) nai =", nai, " ,id(nai) =", id(nai))
    m += " To You !!"
    nai = nai + 2

    print("r-3) new message =", m, " ,id(m) =", id(m))
    print("r-4) new age =", nai, " ,id(nai) =", id(nai))

#main
msg = "Happy Birthday"
age = 20

print("1) msg =", msg, " ,id(msg) =", id(msg))
print("2) age =", age, " ,id(age) =", id(age))
print()

modify1(msg, age)
print("3) msg =", msg, " ,id(msg) =", id(msg))
print("4) age =", age, " ,id(age) =", id(age))
```

## • Call by value

- Caller의 actual argument address와 callee의 formal parameter address가 서로 다름

1) msg = Happy Birthday ,id(msg) = 54810408  
2) age = 20 ,id(age) = 1475074528

r-1) m = Happy Birthday ,id(m) = 54810408  
r-2) nai = 20 ,id(nai) = 1475074528

r-3) new message = Happy Birthday To You !! ,id(m) = 50062688  
r-4) new age = 22 id(nai) = 1475074560

3) msg = Happy Birthday ,id(msg) = 54810408  
4) age = 20 ,id(age) = 1475074528

```

1 def modify1(m, nai):
2     print("m-1) m =", m, ", id(m) =", id(m))
3     print("m-2) nai =", nai, ", id(nai) =", id(nai))
4     m[0] = " Happy Birthday To You !!"
5     nai[0] = 22
6
7     print("m-3) new message =", m, ", id(m) =", id(m))
8     print("m-4) new age =", nai, ", id(nai) =", id(nai))
9
10 #main
11 msg = ["Happy Birthday"]
12 age = [20]
13
14 print("1) msg =", msg, ", id(msg) =", id(msg))
15 print("2) age =", age, ", id(age) =", id(age))
16 print()
17
18 modify1(msg, age)
19 print("m-3) msg =", msg, ", id(msg) =", id(msg))
20 print("4) age =", age, ", id(age) =", id(age))
21

```

- Call by reference
  - Caller의 actual argument address와 callee의 formal parameter address가 서로 같음

```

1) msg = ['Happy Birthday'] , id(msg) = 52826408
2) age = [20] , id(age) = 17420424

```

```

m-1) m = ['Happy Birthday'] , id(m) = 52826408
m-2) nai = [20] , id(nai) = 17420424

```

```

m-3) new message = [' Happy Birthday To You !!'] , id(m) = 52826408
m-4) new age = [22] , id(nai) = 17420424


```

```

3) msg = [' Happy Birthday To You !!'] , id(msg) = 52826408
4) age = [22] , id(age) = 17420424

```

# Call by reference 예제 2)

 call\_by\_ref\_2.py

File Edit Format Run Options Window Help

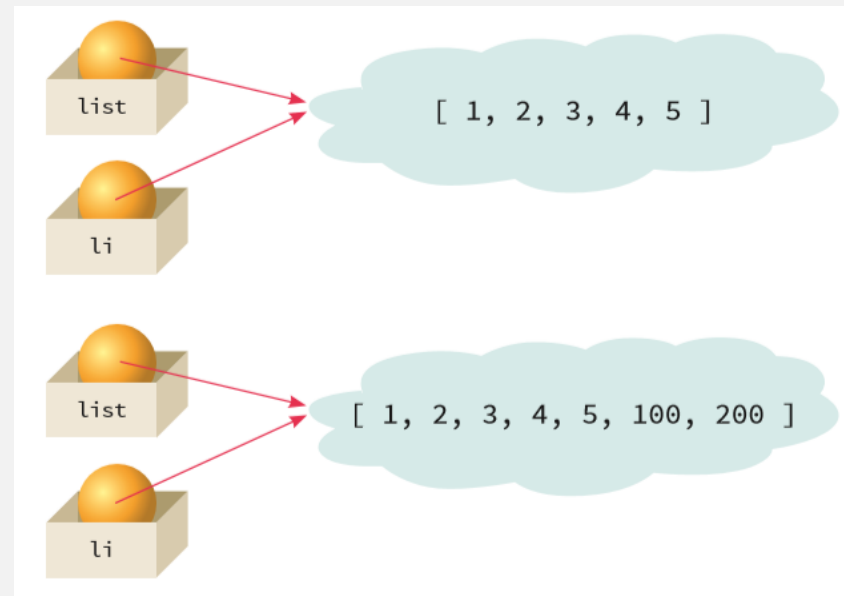
```
def modify2(li):  
    print("r-1) li = ", li)  
    li += [100,200]  
    print("r-2) li = ", li)  
  
#main  
list_val = [1,2,3,4,5]  
print("n1) list = ", list_val)  
  
modify2(list_val)  
print("n2) list = ", list_val)
```

1) list = [1, 2, 3, 4, 5]

r-1) li = [1, 2, 3, 4, 5]

r-2) li = [1, 2, 3, 4, 5, 100, 200]

2) list = [1, 2, 3, 4, 5, 100, 200]



# Passing arguments to a function

5.1passing-function.py

File Edit Format Run Options Window Help

```
def exp(g,m):  
    print("r-1) In exp. ", "return g(", m, ") --> f1 or f2(", m, ")")  
    return(g(m))  
  
def f1(x):  
    print("r-2) In f1(", x, ")")  
    print("r-3) x*x = ", x*x)  
    return x*x  
  
def f2(x):  
    print("r-4) In f2(", x, ")")  
    print("r-5) x*x*x = ", x*x*x)  
    return x*x*x  
  
# main  
n = 5  
print("1) In main, n =", n, "\n")  
  
print("2) call exp(f1, ", n, ")")  
print("r-3) 결과 :", n, ", ", exp(f1, n))  
  
print("r-4) call exp(f2, ", n, ")")  
print("r-5) 결과 ", n, ", ", exp(f2, n))
```

- “함수” 자체도 함수의 인자가 될 수 있음

1) In main, n = 5

2) call exp(f1, 5 )

r-1) In exp. return g( 5 ) --> f1 or f2( 5 )

r-2) In f1( 5 )

r-3) x\*x = 25

3) 결과 : 5 , 25

4) call exp(f2, 5 )

r-1) In exp. return g( 5 ) --> f1 or f2( 5 )

r-4) In f2( 5 )

r-5) x\*x\*x = 125

5) 결과 5 , 125

# Returning a value

- 반환값(return value)
  - Caller로 반환되는 Callee에서 실행한 결과값

```
value = get_sum( 1 , 10 )
```

```
def get_sum (    start , end    ):  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum
```

# Returning a value

- 함수 결과값 리턴
  - return (결과값)

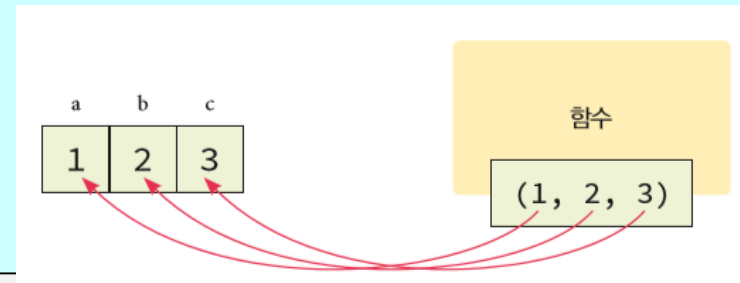
```
5.1example7.py
File Edit Format Run Options Window Help
1 # Function to calculate tax and return the total
2 def calculateTax(price, tax_rate):
3     total = price + (price * tax_rate)
4     return total # return the total
5
6
7 # main
8 my_price = float(input ("1) Enter a price: "))
9
10 # call the function and store the result in totalPrice
11 totalPrice = calculateTax(my_price, 0.06)
12 print ("2) Price = ", my_price)
13 print ("3) Total price = ", totalPrice)
14
```

```
1) Enter a price: 100
2) Price = 100.0
3) Total price = 106.0
```

# 여러 개의 값 반환하기

```
def sub():  
    return 1, 2, 3
```

```
a, b, c = sub()  
print(a, b, c)
```



1 2 3

# Default argument

- Python에서는 함수의 매개변수가 기본값을 가질 수 있음
  - 이것을 **디폴트 인수(default argument)**라고 함

```
def greet(name, msg="별일 없죠?):  
    print("안녕 ", name + ', ' + msg)  
  
greet("영희/")
```

```
안녕 영희, 별일 없죠?
```



# Keyword argument

- 인수들이 위치가 아니고 keyword에 의하여 함수로 전달되는 방식

```
keyword_argument.py
File Edit Format Run Options Window Help
def calc(x, y, z):
    print("c-1) In calc(), x,y,z =", x,y,z)
    return x+y+z

#main
value = calc(10, 20, 30)
print("1) value = ", value)

value = calc(x=10, y=20, z=30)
print("2) value = ", value)

value = calc(z=30, y=20, x=10)
print("3) value = ", value)

value = calc(y=100, z=200, x=300)
print("4) value = ", value)
```

```
c-1) In calc(), x,y,z = 10 20 30
1) value = 60

c-1) In calc(), x,y,z = 10 20 30
2) value = 60

c-1) In calc(), x,y,z = 10 20 30
3) value = 60

c-1) In calc(), x,y,z = 300 100 200
4) value = 600
```

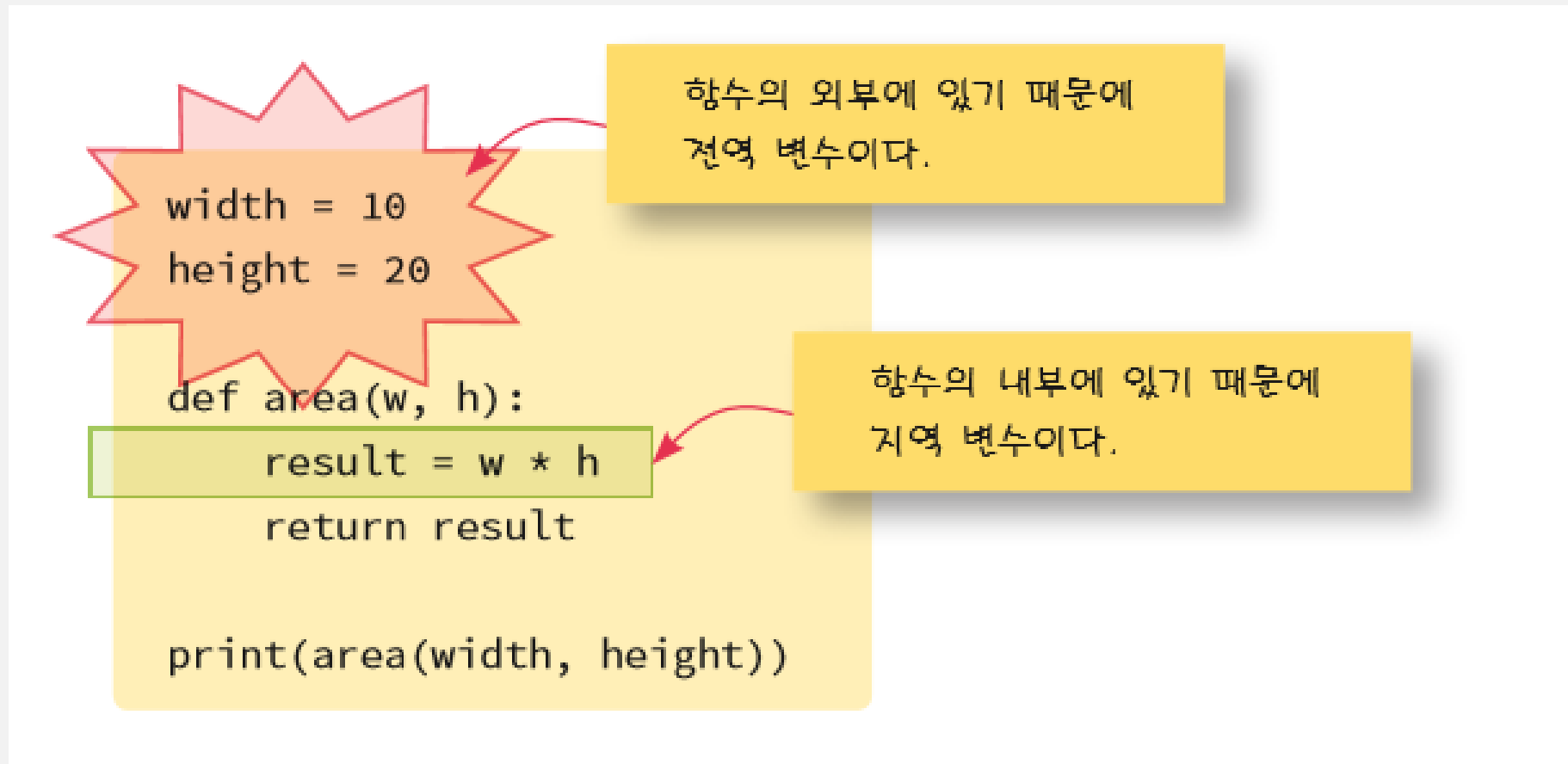
# Variable scope

- 범위(scope)에 따른 변수(variable) 구분
  - 지역(Local) 변수
    - o 함수나 Class 내부에서만 사용
    - o Formal parameter는 지역변수
  - 전역(Global) 변수: program 파일 안에서 사용.
  - 빌트인(Built-in) 변수: Python에서 특별 정의하여 사용
- Python에서 변수 이름 찾는 순서
  - 지역 → 전역 → 빌트인
  - 그래도 존재하지 않으면 error 발생

# Variable scope

- Python이 전역변수를 다루는 방식은 타 언어와 **아주 다른** 접근 방식을 활용
  - 다른 설정이 없으면 함수 안에서 선언된 변수는 무조건 지역변수
  - 전역변수는 함수 내에서 **“read-only”**로 사용
- 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 variable)



# Local variable

- 변수 `s`는 함수 `sub()` 안에서만 사용
- 함수의 호출이 종료되면 `s`는 사라짐
- 지역 변수를 해당 함수(클래스) 외부에서 사용하려는 경우 error 발생

```
local_variable_1.py
File Edit Format Run Options Window Help
1 def sub():
2     print("s-1) In sub(), 변수 s는 local variable")
3
4     s = "바나나가 좋음!" # s : local variable
5     print("s-2) s =", s)
6
7 # main
8 print("\n1) main, call sub()")
9 sub()
10
11 print("\n2) main에서 local variable s 접근 시도 : error !")
12 print(s) # sub()의 지역변수 s 접근 시도. error !
13
```

```
1) main, call sub()
s-1) In sub(), 변수 s는 local variable
s-2) s = 바나나가 좋음!

2) main에서 local variable s 접근 시도 : error !
Traceback (most recent call last):
  File "C:\와소사\와소사-강의예제\local_variable_1.py", line 12, in <module>
    print(s) # sub()의 지역변수 s 접근 시도. error !
NameError: name 's' is not defined
```

# Global variable

- 전역변수 s를 함수 sub()에서 “read-only”로 사용

global\_variable\_1.py

File Edit Format Run Options Window Help

```
1 def sub():
2     print("\ns-1) in sub(), global variable s 접근 : OK !")
3     print("\ns-2) s = ", s)
4
5 # main
6 print("\n1) in main, global variable s 선언")
7
8 s = "사과가 좋음 !" # s : global variable
9 print("\n2) s = ", s)
10
11 print("\n3) call sub()")
12 sub()
13
14 print("\n4) End.")
15
```

1) in main, global variable s 선언  
2) s = 사과가 좋음 !  
3) call sub()

s-1) in sub(), global variable s 접근 : OK !  
s-2) s = 사과가 좋음 !

4) End.

# 지역 변수와 전역 변수

```
def sub():  
    s = "바나나가 좋음!"  
    print("s-1) s = ", s)  
  
#main  
s = "사과가 좋음!"  
sub()  
print("1) in main, s" = s)
```

전역 변수를 사용하는 것이 아님!  
지역 변수 s를 새로이 정의하는 것임

```
s-1) s = 바나나가 좋음!  
1) in main, s = 사과가 좋음!
```

- Python이 전역변수를 다루는 방식은 타 언어와 아주 다른 접근 방식을 활용
- 다른 설정이 없으면 함수 안에서 선언된 변수는 무조건 지역 변수

# 전역 변수를 함수 안에서 사용하려면

```
def sub():  
    global s  
  
    print("s-1 ) s = ", s)  
    s = "바나나가 좋음!"  
    print("s-2) s = ", s)  
  
# main  
s = "사과가 좋음!"  
sub()  
print("1) s = ", s)
```

```
s-1) s = 사과가 좋음!  
s-2) s = 바나나가 좋음!  
1) s = 바나나가 좋음!
```



# 예제

```
def sub(x, y):  
    global a  
  
    a = 7  
    x,y = y,x  
    b = 3  
    print(a, b, x, y)  
  
a,b,x,y = 1,2,3,4  
sub(x, y)  
print(a, b, x, y)
```

```
7 3 4 3  
7 2 3 4
```

# Lab: 매개변수 = 지역변수

- 다음 program의 실행 결과는 어떻게 될까?

```
# 함수가 정의된다.  
def sub( mylist ):  
    # 리스트가 함수로 전달된다.  
    mylist = [1, 2, 3, 4] # 새로운 리스트가 매개변수로 할당된다.  
    print (" 함수 내부에서의 mylist: ", mylist)  
    return  
  
# main 에서 sub() 함수를 호출한다.  
mylist = [10, 20, 30, 40];  
sub( mylist );  
print (" 함수 외부에서의 mylist: ", mylist)
```

```
함수 내부에서의 mylist: [1, 2, 3, 4]  
함수 외부에서의 mylist: [10, 20, 30, 40]
```

# Lab: 매개변수 = 지역변수

local\_global\_variable\_0.py

File Edit Format Run Options Window Help

```
def sub( mylist ):  
  
    print ("s-1) sub 함수도 전달된 mylist: ", mylist)  
    mylist = [1, 2, 3, 4]    # 새로운 리스트가 매개변수로 할당된다.  
  
    print ("s-2) sub 함수에서 mylist를 새로이 정의 후: ", mylist)  
    return
```

```
#main  
mylist = [10, 20, 30, 40];  
print ("Wn1) main에서 정의된 mylist: ", mylist)  
  
sub(mylist);  
print ("Wn2) 리턴된 후 함수 외부(main)에서의 mylist: ", mylist)
```

1) main에서 정의된 mylist: [10, 20, 30, 40]

s-1) sub 함수도 전달된 mylist: [10, 20, 30, 40]

s-2) sub 함수에서 mylist를 새로이 정의 후: [1, 2, 3, 4]

2) 리턴된 후 함수 외부(main)에서의 mylist: [10, 20, 30, 40]

```

1 def sub(mylist, mycomment):
2
3     print ("Wns-1) sub 함수도 전달된 mylist:", mylist, ", id(mylist) =", id(mylist))
4     print ("mycomment:", mycomment, ", id(mycomment) =", id(mycomment))
5
6     # Error --> UnboundLocalError: local variable 'comment' referenced before assignment
7     # global comment
8     print ("s-2) comment :", comment, ", id(comment) = ", id(comment))
9
10    mylist += ["NEW", "LIST"]
11    print ("s-3) mylist에 값 추가 후 :", mylist, ", id(mylist) =", id(mylist))
12
13    # 새로운 local 변수인 mylist를 선언하여 새로운 값을 할당
14    mylist = [1, 2, 3, 4]
15    mycomment = "Sub 입니다."
16
17    # comment = "변경된 값"
18
19    print ("Wns-4) sub 함수에서 mylist를 새로이 정의 후:", mylist, ", id(mylist) =", id(mylist))
20    print ("s-5) sub 함수에서 mycomment를 새로이 정의 후:", mycomment, ", id(comment) =", id(mycomment))
21
22    return mylist, mycomment
23
24
25 #main
26 print("1) main")
27
28 comment = "Main 입니다."
29 mylist = [10, 20, 30, 40];
30
31 print ("Wn2) main에서 정의된 mylist :", mylist, ", id(mylist) =", id(mylist))
32 print ("3) comment :", comment, ", id(comment) =", id(comment))
33
34 ret_list, ret_comment = sub(mylist, comment);
35
36 print("Wn4) 리턴된 mylist :", mylist, ", id(mylist) =", id(mylist))
37 print("5) 리턴된 값 ret_list :", ret_list, ", id(ret_list) =", id(ret_list))
38
39 print("Wn6) comment :", comment, ", id(comment) =", id(comment))
40 print("7) ret_comment :", ret_comment, ", id(ret_comment) =", id(ret_comment))
41

```

1) main

2) main에서 정의된 mylist : [10, 20, 30, 40] , id(mylist) = 62984520

3) comment : Main 입니다. , id(comment) = 65370976

s-1) sub 함수도 전달된 mylist: [10, 20, 30, 40] , id(mylist) = 62984520  
mycomment: Main 입니다. , id(mycomment) = 65370976

s-2) comment : Main 입니다. , id(comment) = 65370976

s-3) mylist에 값 추가 후 : [10, 20, 30, 40, 'NEW', 'LIST'] , id(mylist) = 62984520

s-4) sub 함수에서 mylist를 새로이 정의 후: [1, 2, 3, 4] , id(mylist) = 27644040

s-5) sub 함수에서 mycomment를 새로이 정의 후: Sub 입니다. , id(comment) = 61765968

4) 리턴된 mylist : [10, 20, 30, 40, 'NEW', 'LIST'] , id(mylist) = 62984520

5) 리턴된 값 ret\_list : [1, 2, 3, 4] , id(ret\_list) = 27644040

6) comment : Main 입니다. , id(comment) = 65370976

7) ret\_comment : Sub 입니다. , id(ret\_comment) = 61765968

## Lab: 상수

- 파이를 전역 변수로 선언하고 이것을 이용하여서 원의 면적과 원의 둘레를 계산하는 함수를 작성해보자.

*원의 반지름을 입력하시오: 10*  
*원의 면적: 314.159265358979*  
*원의 둘레: 62.8318530717958*

# Solution

 area\_of\_circle.py

File Edit Format Run Options Window Help

```
PI = 3.14159265358979 # 전역 상수
```

```
def circleArea(radius):  
    return PI*radius*radius
```

```
def circleCircumference(radius):  
    return 2*PI*radius
```

```
def main():  
    radius = float(input('원의 반지름을 입력하시오 : '))  
    print('원의 면적 :', circleArea(radius))  
    print('원의 둘레 :', circleCircumference(radius))
```

```
#main  
main()
```

```
>> 원의 반지름을 입력하시오 : 5  
원의 면적 : 78.53981633974475  
원의 둘레 : 31.4159265358979  
^^^
```

# Variable scope

- Local/global 변수 접근 예러
  - 함수(클래스)에서 global 변수 (**my\_price**) 접근 가능

```
5.1example8.py
File Edit Format Run Options Window Help
def calculateTax(price, tax_rate):
    total = price + (price * tax_rate)
    print ("c-1) my_price =", my_price) # try to print my_price. 동작함
    return total

# main
my_price = float(input (">> Enter a price : "))

totalPrice = calculateTax(my_price, 0.06) >> Enter a price : 100
print ("1) price =", my_price)
print ("2) Total price =", totalPrice) c-1) my_price = 100.0
```

```
1) price = 100.0
2) Total price = 106.0
```

# Variable scope

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

```
5.1example9.py
File Edit Format Run Options Window Help
1 # Define function calculateTax
2 def calculateTax(price, tax_rate):
3     total = price + (price * tax_rate)
4     print ("c-1) my_price =", my_price) # try to print my_price. 동작함
5     return total
6
7 # Main program calls the function
8 my_price = float(input ("1) >> Enter a price :"))
9
10 totalPrice = calculateTax(my_price, 0.06)
11 print ("2) price = ", price, " Total price = ", totalPrice) #에러발생
12
```

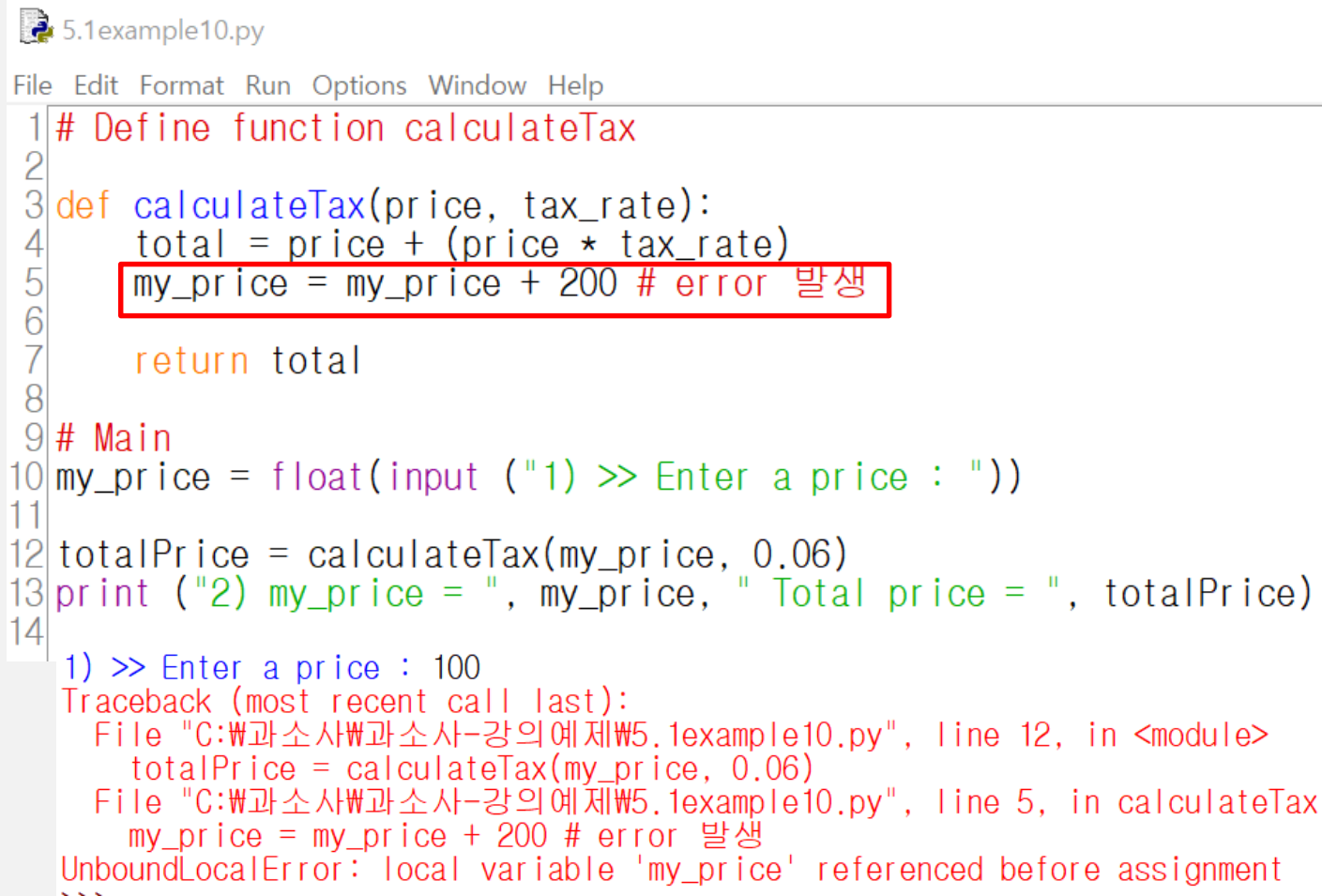
```
1) >> Enter a price :100

c-1) my_price = 100.0
Traceback (most recent call last):
  File "C:\과소사\과소사-강의예제\5.1example9.py", line 11, in <module>
    print ("2) price = ", price, " Total price = ", totalPrice) #에러발생
NameError: name 'price' is not defined
```



# Variable scope

- Local/global 변수 접근 에러
  - 함수(클래스)에서 'global' 표시 없이 전역 변수(**my\_price**) 값을 **변경**하려 하여 에러 발생



```
5.1example10.py
File Edit Format Run Options Window Help
1 # Define function calculateTax
2
3 def calculateTax(price, tax_rate):
4     total = price + (price * tax_rate)
5     my_price = my_price + 200 # error 발생
6
7     return total
8
9 # Main
10 my_price = float(input ("1) >> Enter a price : "))
11
12 totalPrice = calculateTax(my_price, 0.06)
13 print ("2) my_price = ", my_price, " Total price = ", totalPrice)
14
15 1) >> Enter a price : 100
Traceback (most recent call last):
  File "C:\과소사\과소사-강의예제\5.1example10.py", line 12, in <module>
    totalPrice = calculateTax(my_price, 0.06)
  File "C:\과소사\과소사-강의예제\5.1example10.py", line 5, in calculateTax
    my_price = my_price + 200 # error 발생
UnboundLocalError: local variable 'my_price' referenced before assignment
```

# Variable scope

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

```
5.1example11.py
File Edit Format Run Options Window Help
1 # Define function calculateTax
2 def calculateTax(price, tax_rate):
3     total = price + (price * tax_rate)
4
5     global my_price          # global 지정하여
6     my_price = my_price + 200 # error 발생하지 않음
7
8     return total
9
10 # Main
11 my_price = float(input ("1) >> Enter a price : "))
12
13 totalPrice = calculateTax(my_price, 0.06)
14 print ("2) my_price = ", my_price, " Total price = ", totalPrice)
15
```

```
1) >> Enter a price : 100
2) my_price = 300.0 Total price = 106.0
```

# Naming variables

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

5.1example12.py

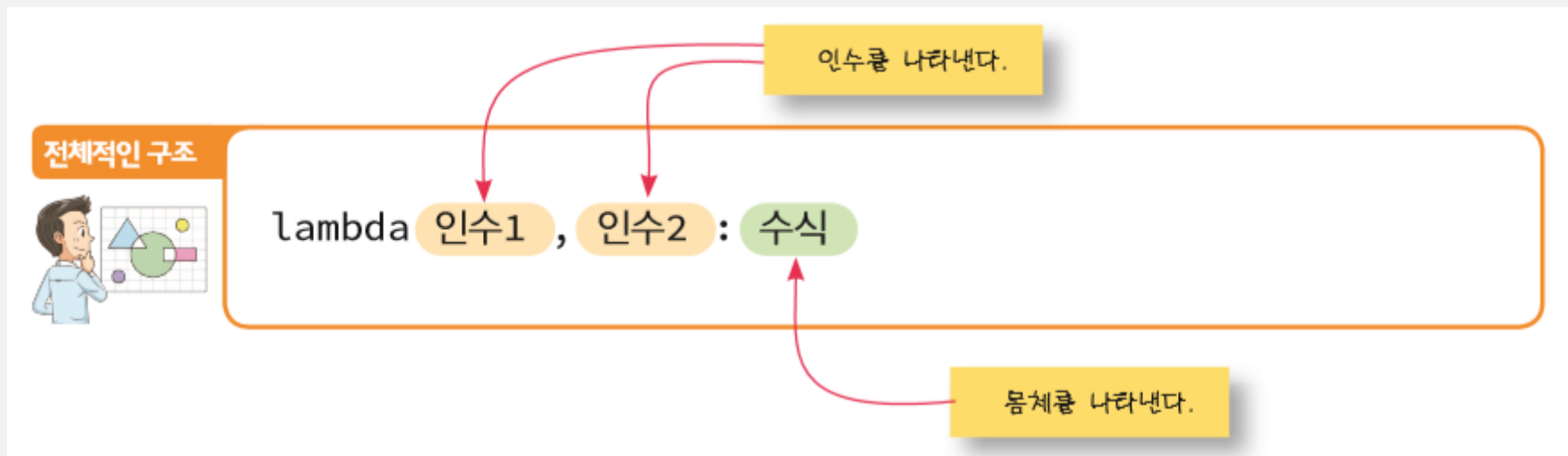
File Edit Format Run Options Window Help

```
1 # Define function calculateTax
2 def calculateTax(price, tax_rate):
3     total = price + (price * tax_rate)
4     my_price = 200 #에러는 발생하지 않음. 왜?
5
6     return total
7
8 # Main
9 my_price = float(input ("1) >> Enter a price : "))
10
11 totalPrice = calculateTax(my_price, 0.06)
12 print ("2) my_price = ", my_price, " Total price = ", totalPrice)
13
```

```
1) >> Enter a price : 100
2) my_price = 100.0 Total price = 106.0
```

# lambda function(무명 함수)

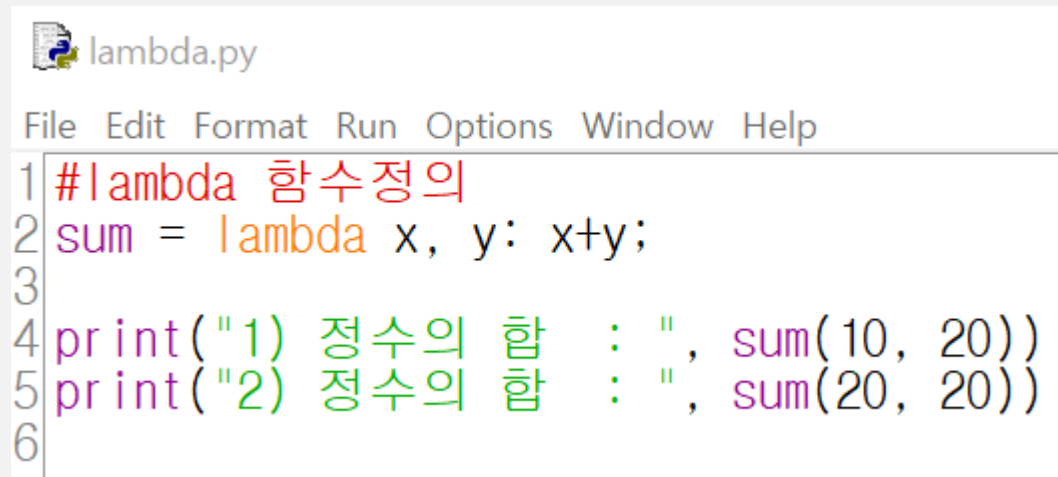
- 무명 함수는 이름은 없고 몸체만 있는 함수
  - 이름없는 함수
  - Python에서 무명 함수는 lambda keyword를 사용
  - 한 줄 함수
  - 주로 함수를 함수 인자로 넘길 때 사용
- lambda 인수: 특별한 인수는 없고 “리턴값”이 인수



# lambda function(무명 함수)

- 주요 용도
  - Code 안에서 함수를 포함하는 곳 어디서든 활용가능
  - GUI event를 처리하는 callback handler 등에서 사용
  - Jump table 등

# lambda function 예 1



```
lambda.py
File Edit Format Run Options Window Help
1 #lambda 함수정의
2 sum = lambda x, y: x+y;
3
4 print("1) 정수의 합 : ", sum(10, 20))
5 print("2) 정수의 합 : ", sum(20, 20))
6
```

```
1) 정수의 합 : 30
2) 정수의 합 : 40
//
```

# lambda function 예 2

lambda2.py

File Edit Format Run Options Window Help

```
1 # 예제 1
2 Function_List = [ lambda x : x**2,
3                   lambda y : y**3,
4                   lambda z : print("lambda) {}**4 = {}".format(z, z**4)) ]
5
6 for f in Function_List:
7     print("1) f(2) = ", f(2))
8     print("2) f(10) = ", f(10), "\n")
9
10 # 예제 2
11 min = (lambda x,y : x if x < y else y )
12
13 min_value = min(100, 200)
14 print("3) min =", min_value)
15
```

1) f(2) = 4  
2) f(10) = 100

1) f(2) = 8  
2) f(10) = 1000

lambda) 2\*\*4 = 16  
1) f(2) = None  
lambda) 10\*\*4 = 10000  
2) f(10) = None

3) min = 100

# lambda function 예3



5.1example16-1.py

File Edit Format Run Options Window Help

```
1 def exp(g, m):
2     print("\ne-1) start")
3     print("e-2) g :",g)
4     print("e-3) m =", m)
5     print("e-4) g(m) =", g(m))
6
7     return (g(m))
8
9 print("1) >")
10 f1 = lambda x: x*x
11
12 print("2) >>")
13 f2 = lambda x: x*x*x
14
15 print("3) >>>\n")
16
17 n = 2
18 print("4) f1(10) :",f1(n))
19 print("5) f2(10) :",f2(n))
20
21 print("\n6) exp(f1, 10) =", exp(f1, n))
22 print("\n7) exp(f2, 10) =", exp(f2, n))
23
```

```
1) >
2) >>
3) >>>
```

```
4) f1(10) : 4
5) f2(10) : 8
```

```
e-1) start
e-2) g : <function <lambda> at 0x03B17BB0>
e-3) m = 2
e-4) g(m) = 4
```

```
6) exp(f1, 10) = 4
```

```
e-1) start
e-2) g : <function <lambda> at 0x03B17BF8>
e-3) m = 2
e-4) g(m) = 8
```

```
7) exp(f2, 10) = 8
```

```
>>>
```



# lambda function 예4

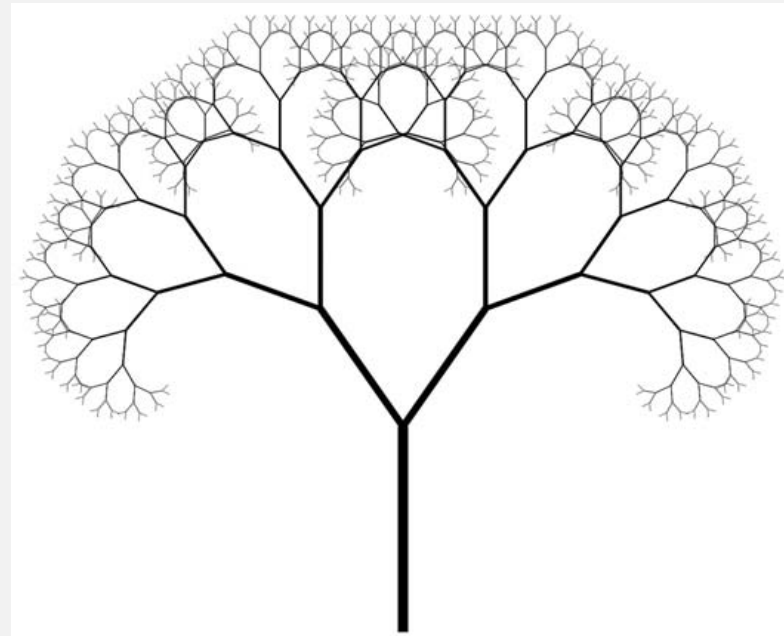
- 예: List 정렬
  - Lambda 함수 정의: Tuple 첫 번째, 두 번째 값으로 정렬

```
5.1example20.py
File Edit Format Run Options Window Help
1 #tuple list 정렬
2 tuple_list = [ ("Fa",400),("Re",200), ("Do",100), ("Mi",300) ]
3
4 print("1) Sorting 이전 :", tuple_list)
5
6 print("\n2) x[0]로 sorting")
7 tuple_list.sort(key = lambda x: x[0])
8 print("3) After sorting :", tuple_list)
9
10 print("\n4) x[1]로 sorting")
11 tuple_list.sort(key = lambda x: x[1])
12 print("5) After sorting :", tuple_list)
13
14 tuple_list.sort()
15 print("\n6) sort() :", tuple_list)
16
```

```
1) Sorting 이전 : [('Fa', 400), ('Re', 200), ('Do', 100), ('Mi', 300)]
2) x[0]로 sorting
3) After sorting : [('Do', 100), ('Fa', 400), ('Mi', 300), ('Re', 200)]
4) x[1]로 sorting
5) After sorting : [('Do', 100), ('Re', 200), ('Mi', 300), ('Fa', 400)]
6) sort() : [('Do', 100), ('Fa', 400), ('Mi', 300), ('Re', 200)]
^^^
```

# Recursive Functions

- 재귀함수(recursive function)
  - 자기 자신을 재호출하는 함수
  - 동일한 문제 해결 방식을 반복하는 상황에서 활용
- Computer science 분야에서 흔하게 발생
  - 1부터  $n$ 까지 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

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

5.1example13.py

File Edit Format Run Options Window Help

```
1 n = 100
2 sum=0
3
4 for i in range(1,n+1):
5     sum = sum + i
6
7 # main
8 print(">> n = {}, sum = {}".format(n, sum))
9
10
```

```
>> n = 100, sum = 5050
```

5.1example14.py

File Edit Format Run Options Window Help

```
1 def sigma(n):
2     if n == 1:
3         return n
4     else:
5         return n + sigma(n-1)
6
7 # main
8 n=10
9 print(">> sigma({}) = {}".format(n, sigma(n)))
10
```

```
>> sigma(10) = 55
```

# Recursive Functions

– 예1) 1부터 n까지 합산

$\text{sigma}(1) = 1$

$\text{sigma}(2) = \text{sigma}(1) + 2$

$\text{sigma}(3) = \text{sigma}(2) + 3$

$\text{sigma}(4) = \text{sigma}(3) + 4$

.....

$\text{sigma}(n) = \text{sigma}(n-1) + n$

→  $\text{sigma}(1) = 1$

$\text{sigma}(n) = \text{sigma}(n-1) + n$



5.1example14.py

File Edit Format Run Options Window Help

```
1 def sigma(n):
2     if n == 1:
3         return n
4     else:
5         return n + sigma(n-1)
6
7 # main
8 n=10
9 print(">> sigma({}) = {}".format(n, sigma(n)))
10
```

# Recursive Functions

## – 예) 2) factorial

facto(0) = 1

facto(1) = 1 \* facto(0)

facto(2) = 2 \* facto(1)

facto(3) = 3 \* facto(2)

.....

facto(n) = n \* facto(n-1)

→ facto(0) = 1

facto(n) = n \* facto(n-1)

*# 팩토리얼*

**def** facto( n ) :

**if** n == 0 :

**return** 1

**else** :

**return** n \* facto(n-1)

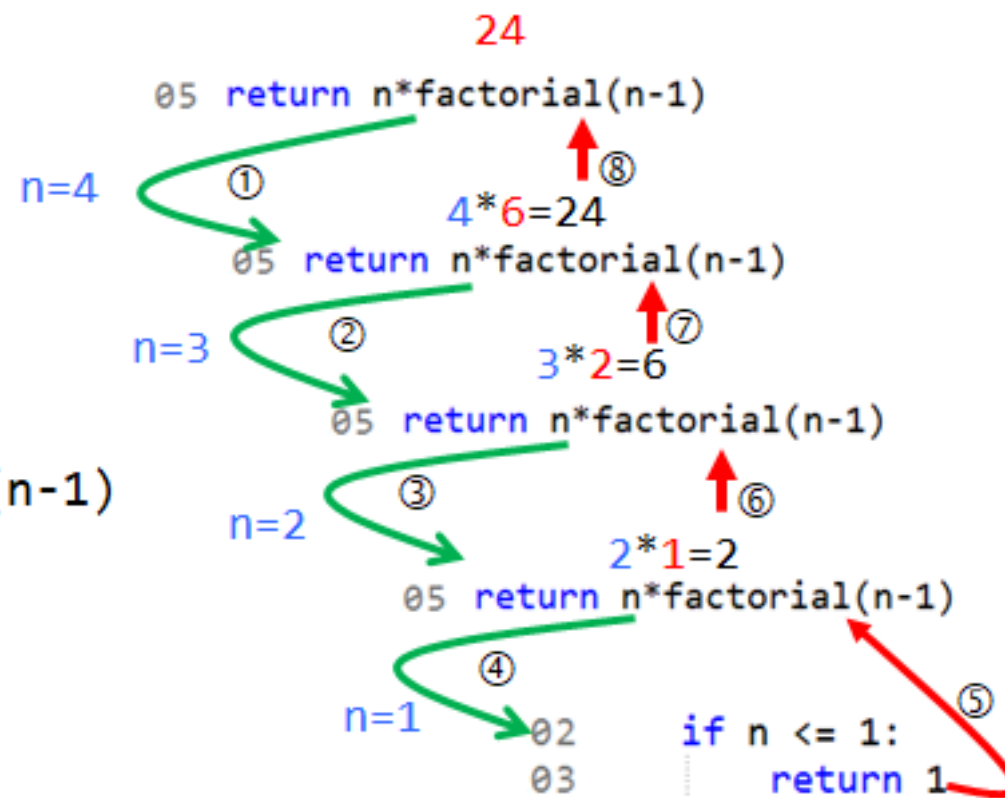
n = int(input("N >"))

print("{}!={}".format(n,facto(n)))

## 팩토리얼 계산

- Line 07에서 factorial(4)로 함수 호출
- [그림]의 번호 순서대로 수행되어 24를 출력

```
01 def factorial(n):  
02     if n <= 1:  
03         return 1  
04     else:  
05         return n*factorial(n-1)  
06  
07 print(factorial(4))
```



# 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

## – 예3) 피보나치(Fibonacci) 수열

- 0, 1, 1, 2, 3, 5, 8, 13, 21 .....

$\text{fibonacci}(0) = 0$

$\text{fibonacci}(1) = 1$

$\text{fibonacci}(2) = \text{fibonacci}(1) + \text{fibonacci}(0) = 1$

$\text{fibonacci}(3) = \text{fibonacci}(2) + \text{fibonacci}(1) = 2$

$\text{fibonacci}(4) = \text{fibonacci}(3) + \text{fibonacci}(2) = 3$

.....

$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$

→  $\text{fibonacci}(0) = 0$

$\text{fibonacci}(1) = 1$

$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$

*# 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

```
5.1example151.py
File Edit Format Run Options Window Help
import time

def fibonacci(n):
    if (n == 1) or (n == 2):
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

# main
n=30

start = time.time()

print(n, fibonacci(n))
print("1) Running fibonacci(%d) takes %f" %(n,time.time()-start))
```

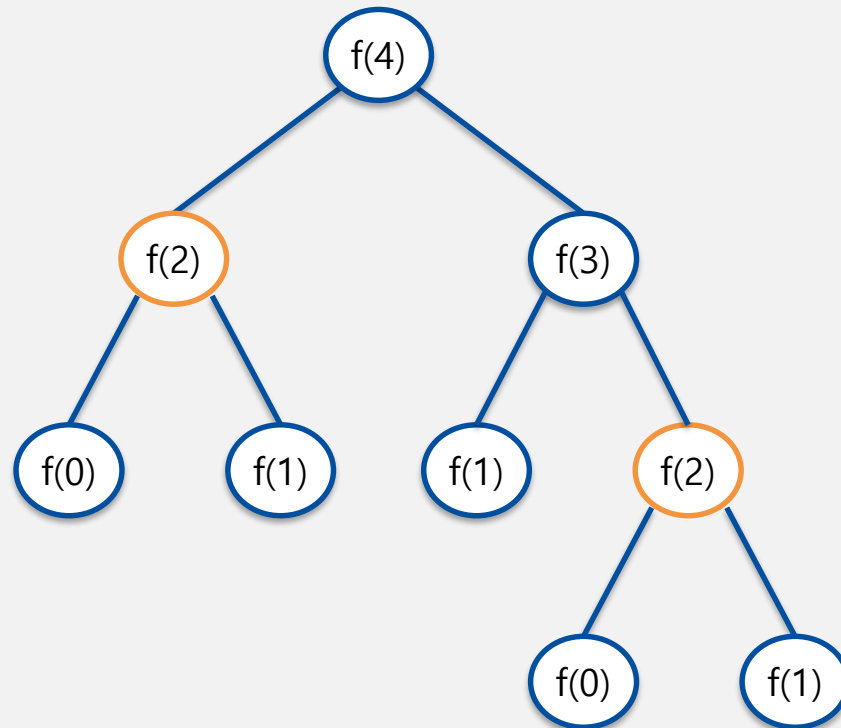
```
30 832040
1) Running fibonacci(30) takes 0.280654
^^^
```

## 숙제 – 파일명 : fibo-이름-학번-일시.py

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

- 메모이제이션 (memoization)

- 컴퓨터 program이 동일한 계산을 반복해야 할 때, 이전에 계산한 값을 memory에 저장함으로써 동일한 계산의 반복 수행을 제거하여 program 실행 속도를 빠르게 하는 기술



# 실습

- 메모이제이션 (memoization)

```
counter = 0 # 피보나치 수열의 연산 횟수
memo = {} # 피보나치 수열의 결과를 저장할 사전

# 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))
```

# 실습

memoization\_fibo.py

File Edit Format Run Options Window Help

```
1 counter = 0 # 피보나치 수열의 연산 횟수
2 memo = {}   # 피보나치 수열의 결과를 저장할 사전
3
4 # 피보나치 수열을 구하는 함수
5 def fibo (n) :
6
7     global counter
8
9     counter += 1
10
11     if n in memo :
12         print("f1) memo[{}] = {} 값 사용".format(n, memo[n]))
13         return memo[n]
14
15     if n == 0 :
16         memo[n] = 0
17         print("f2) memo[{}] = {} 저장".format(n, memo[n]))
18
19     elif n == 1 :
20         memo[n] = 1
21         print("f3) memo[{}] = {} 저장".format(n, memo[n]))
22
23     else :
24         memo[n] = fibo(n-1) + fibo(n-2)
25         print("f4) memo[{}] = {} 저장".format(n, memo[n]))
26
27     return memo[n]
28
29 # main
30 n = int(input("f1) >> n = "))
31
32 print("f2) {}번째 피보나치 수 : {}".format(n, fibo(n)))
33 print("f3) 총 {}번 연산을 하였습니다.".format(counter))
34 print("f4) memo =", memo)
35
```

```
>> n = 7
f3) memo[1] = 1 저장
f2) memo[0] = 0 저장
f4) memo[2] = 1 저장

f1) memo[1] = 1 값 사용
f4) memo[3] = 2 저장

f1) memo[2] = 1 값 사용
f4) memo[4] = 3 저장

f1) memo[3] = 2 값 사용
f4) memo[5] = 5 저장

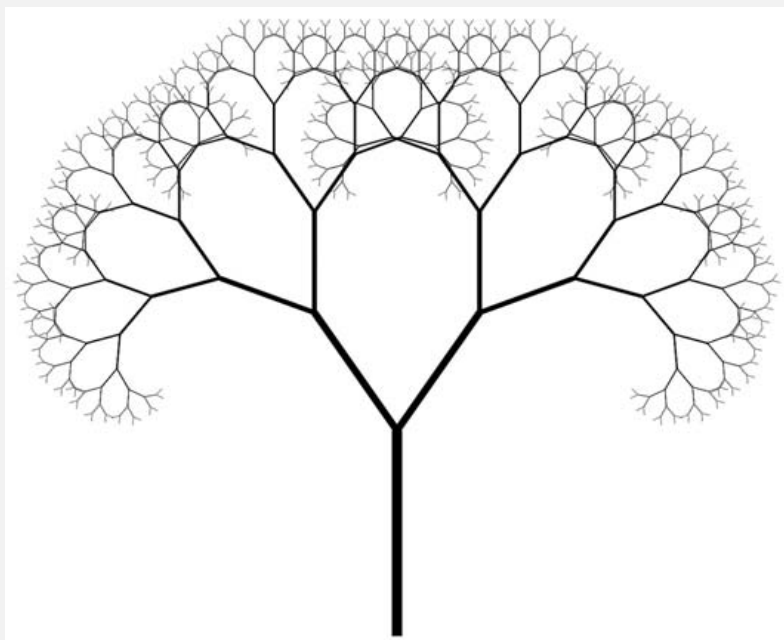
f1) memo[4] = 3 값 사용
f4) memo[6] = 8 저장

f1) memo[5] = 5 값 사용
f4) memo[7] = 13 저장

1) 7번째 피보나치 수 : 13
2) 총 13번 연산을 하였습니다.
3) memo = {1: 1, 0: 0, 2: 1, 3: 2, 4: 3, 5: 5, 6: 8, 7: 13}
```

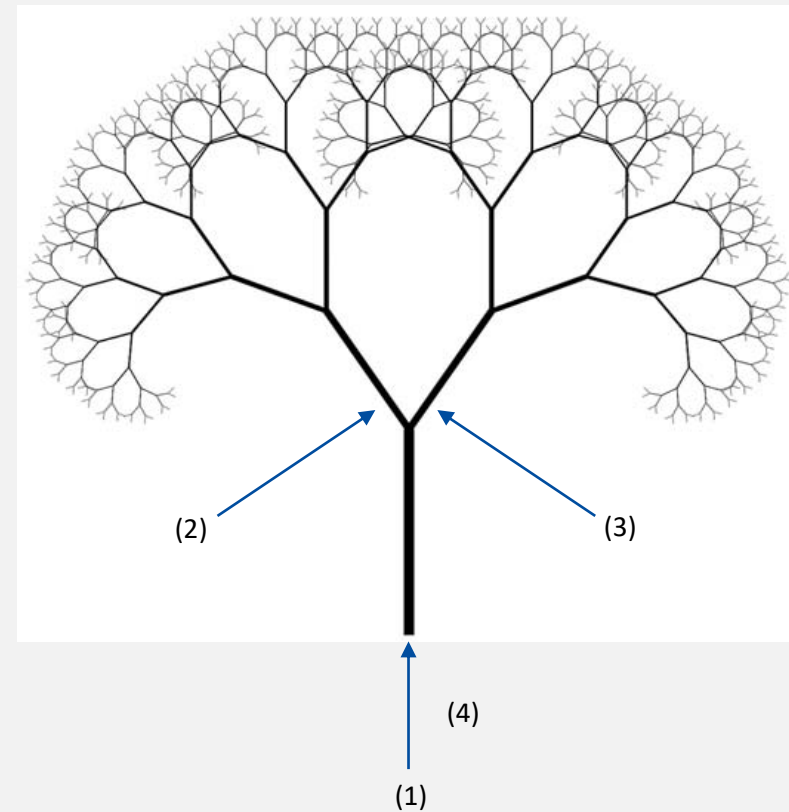
# 숙제 - 파일명 : tree-이름-학번-일시.py

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



# 숙제

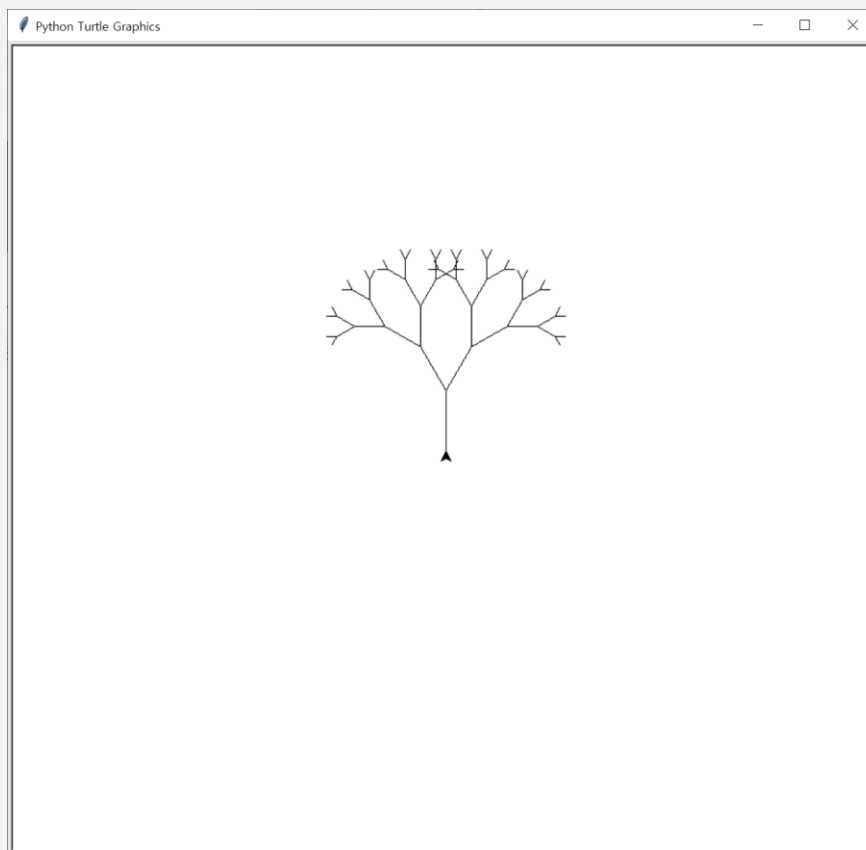
- 재귀적 나무 그리기
  - 재귀적 구조 찾기
    - 나무는 가운데 직선, 왼쪽 나무, 오른쪽 나무로 구성된다!
      - (1) 시작점에서 직선을 그린다
      - (2) 왼쪽 나무를 그린다
      - (3) 오른쪽 나무를 그린다
      - (4) 시작점으로 돌아온다





# 숙제

- 재귀적 나무 그리기
  - 재귀적 구조 찾기



5.1example152.py

File Edit Format Run Options Window Help

```
1 import turtle
2
3 s = turtle.Screen()
4 t = turtle.Turtle()
5
6 angle = 30
7
8 def drawTree(t, lineLength):
9
10     if (lineLength > 0):
11
12         t.forward(lineLength)
13         t.left(angle)
14         drawTree(t, lineLength-10)
15
16         t.right(angle)
17         t.right(angle)
18         drawTree(t, lineLength-10)
19
20         t.left(angle)
21         t.backward(lineLength)
22
23
24 if __name__ == "__main__":
25     lineLength=60
26     t.left(90)
27     drawTree(t, lineLength)
28
```

# 내장 함수

- Python에서 기본적으로 제공하는 함수

|               |             |              |            |                |
|---------------|-------------|--------------|------------|----------------|
| abs()         | dict()      | help()       | min()      | setattr()      |
| all()         | dir()       | hex()        | next()     | slice()        |
| any()         | divmod()    | id()         | object()   | sorted()       |
| ascii()       | 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()      |                |

# 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  $x + yj$  인 경우  $\sqrt{x^2 + y^2}$ 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 (번호, 값) 들로 구성.

```
enumerate.py
File Edit Format Run Options Window Help
1 actors = [ "Jack Nicholson", "Morgan Freeman", "Robert De Niro",
2            "Al Pacino", "Leonardo DiCaprio", "Tom Hanks", "Russell Crowe" ]
3
4 print("\n1) actors = ", actors)
5
6 enu = enumerate(actors)
7 print("\n2) enu =", enu)
8
9 li = list(enu)
10 print("\n3) li =", li)
11
12 print()
13
14 for number, name in enumerate(actors, start = 1) :
15     print(">> {}번째 Actor : {}".format(number, name))
16
```

```
1) actors = ['Jack Nicholson', 'Morgan Freeman', 'Robert De Niro', 'Al Pacino', 'Leonardo DiC
aprio', 'Tom Hanks', 'Russell Crowe']
2) enu = <enumerate object at 0x03039B68>
3) li = [(0, 'Jack Nicholson'), (1, 'Morgan Freeman'), (2, 'Robert De Niro'), (3, 'Al Pacino'),
(4, 'Leonardo DiCaprio'), (5, 'Tom Hanks'), (6, 'Russell Crowe')]
>> 1번째 Actor : Jack Nicholson
>> 2번째 Actor : Morgan Freeman
>> 3번째 Actor : Robert De Niro
>> 4번째 Actor : Al Pacino
>> 5번째 Actor : Leonardo DiCaprio
>> 6번째 Actor : Tom Hanks
```

# 내장 함수

- `sum()`
  - 리스트 혹은 tuple의 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 = [  
    ("Park", 20503253, 4.2),  
    ("Lee", 20503180, 3.7),  
    ("Song", 20503250, 4.5) ]
```

student 에 있는 각  
원소의 첫번째 값을  
기준으로 정렬

```
print("Before sorted :", student)  
sort_by_id = sorted(student, key = lambda x : x[0])  
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 = {  
    20503180 : 3.7,  
    20503250 : 4.5,  
    20503253 : 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])
  - Python 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

5.1example18.py

File Edit Format Run Options Window Help

```
1 a = "Kookmin University is leading the computer science".split()
2
3 print("1) a :", a)
4
5 a.sort()
6 print("2) sort 후, a : ", a)
7
8 a.sort(key=str.upper)
9 print("3) upper sort 후, a :", a)
10
11 a.sort(key=str.upper, reverse=True)
12 print("4) upper reverse sort 후, a =", a)
13
```

```
1) a : ['Kookmin', 'University', 'is', 'leading', 'the', 'computer', 'science']
2) sort 후, a : ['Kookmin', 'University', 'computer', 'is', 'leading', 'science', 'the']
3) upper sort 후, a : ['computer', 'is', 'Kookmin', 'leading', 'science', 'the', 'University']
4) upper reverse sort 후, a = ['University', 'the', 'science', 'leading', 'Kookmin', 'is', 'computer']
```

# 내장 함수

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

```
5.1example19.py
File Edit Format Run Options Window Help
1 b = ["34", "123", "7"]
2
3 print("1) b :", b)
4
5 b.sort()
6 print("2) sort :", b)
7
8 b.sort(key=int)
9 print("3) int() 함수 사용으로 스트링을 정수로 변경하여 정렬 후 :", b)
10
...
```

```
1) b : ['34', '123', '7']
2) sort : ['123', '34', '7']
3) int() 함수 사용으로 스트링을 정수로 변경하여 정렬 후 : ['7', '34', '123']
```

# main

- 참고) [https://hashcode.co.kr/questions/3/if-\\_\\_name\\_\\_-\\_\\_main\\_\\_%EC%9D%80-%EC%99%9C%EC%93%B0%EB%82%98%EC%9A%94](https://hashcode.co.kr/questions/3/if-__name__-__main__%EC%9D%80-%EC%99%9C%EC%93%B0%EB%82%98%EC%9A%94)(2020/5/4 현재 )
- Script가 Python interpreter 명령어로 passing되어 실행되면( a.py같이) 다른 언어들과는 다르게 Python은 자동으로 실행되는 main 함수가 없음
- Python은 main 함수가 없는 대신 들여쓰기 하지 않은 모든 코드 (level 0 코드)를 실행
- 다만, 함수나 클래스는 정의되었지만, 실행되지는 않음
- `__name__`은 현재 모듈의 이름을 담고있는 내장 변수임
- **a.py 같이 이 모듈이 직접 실행되는 경우에만, `__name__`은 `"__main__"`으로 설정됨**



a.py

File Edit Format Run Options Window Help

```
1 def func():
2     print("5) 여기는 function func() in a.py")
3
4 print("6) top-level A.py")
5
6 if __name__ == "__main__":
7     print("7) a.py 직접 실행")
8 else:
9     print("8) a.py가 임포트되어 사용됨")
10
```

6) top-level A.py  
7) a.py 직접 실행

a.py를 실행 시



b.py

File Edit Format Run Options Window Help

```
1 import a
2
3 print("1) top-level in B.py")
4 print("2) a.py에 있는 a.func() 호출 ")
5 a.func()
6
7 if __name__ == "__main__":
8     print("3) b.py가 직접 실행")
9 else:
10     print("4) b.py가 임포트되어 사용됨")
11
```

6) top-level A.py  
8) a.py가 임포트되어 사용됨  
1) top-level in B.py  
2) a.py에 있는 a.func() 호출  
5) 여기는 function func() in a.py  
3) b.py가 직접 실행

b.py를 실행 시



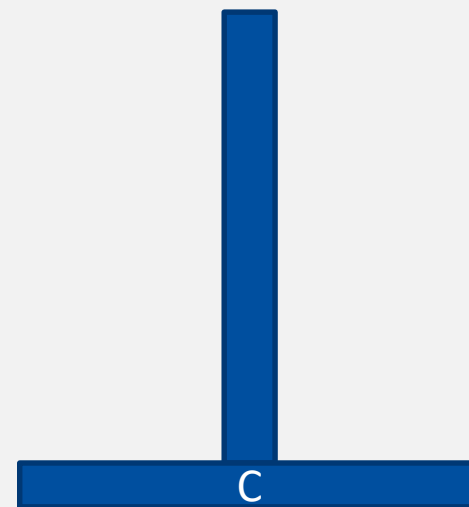
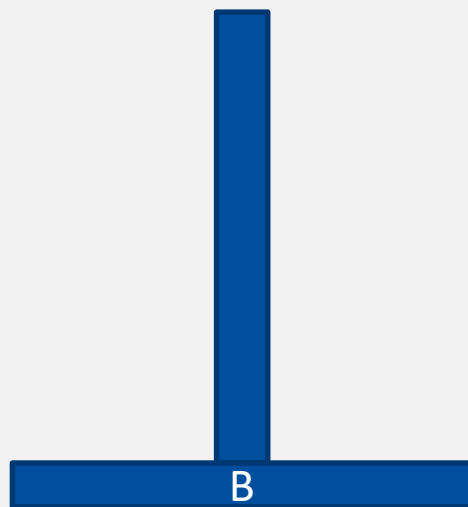
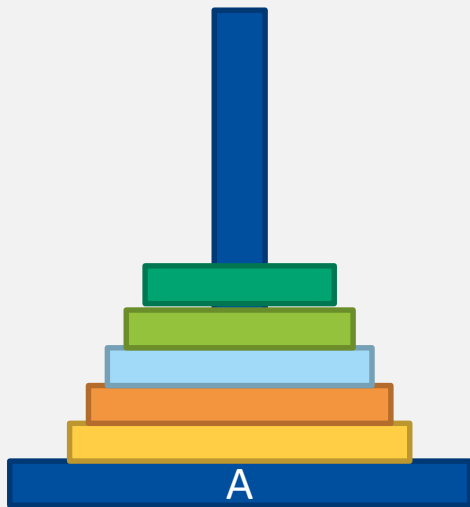
# 숙제 - 파일명 : hanoi-이름-학번-일시.py

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

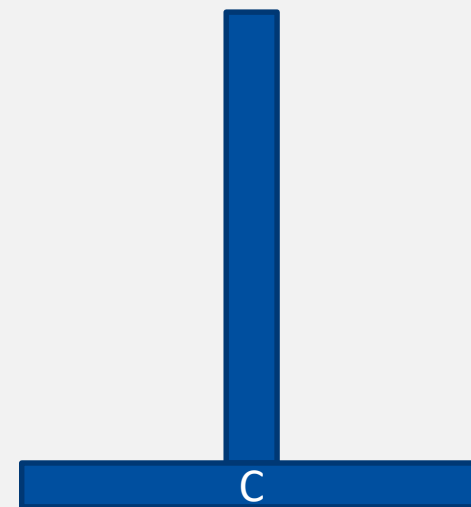
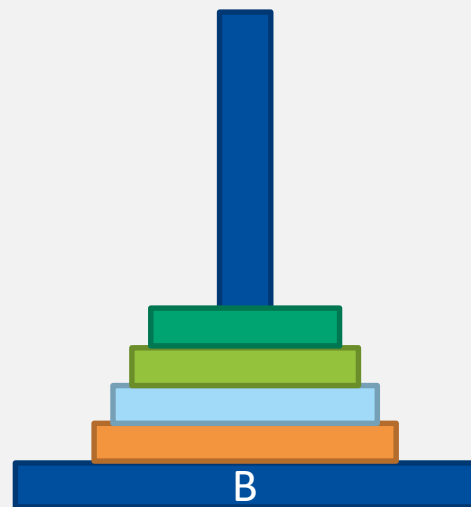
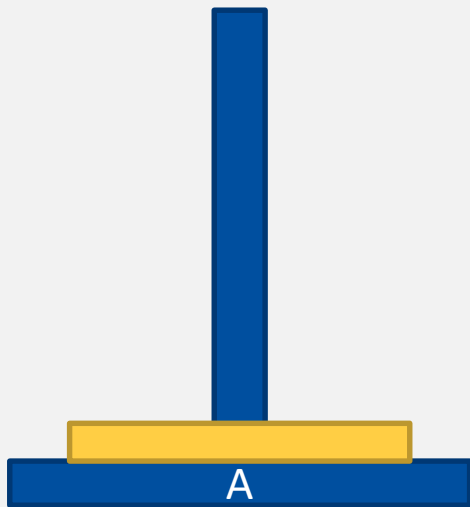
# 숙제

- 3 개의 장대가 있고 첫 번째 장대에는 반경이 서로 다른  $n$ 개의 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 program을 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)

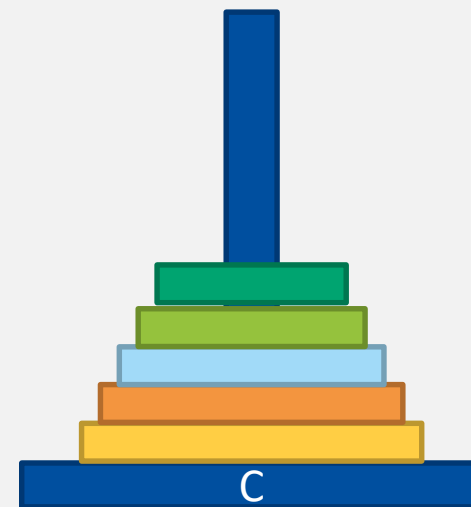
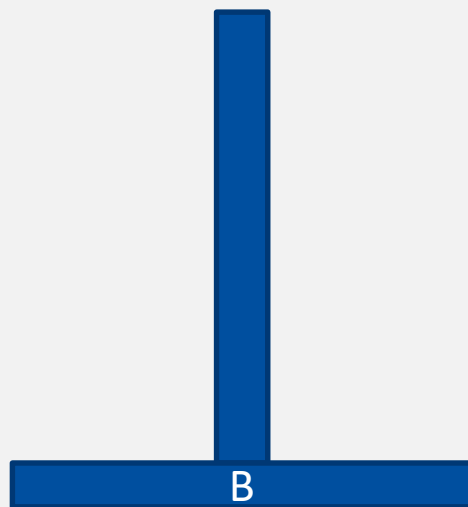
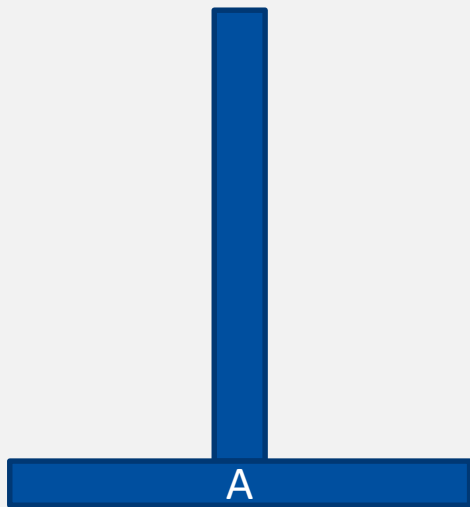
# 숙제



# 숙제



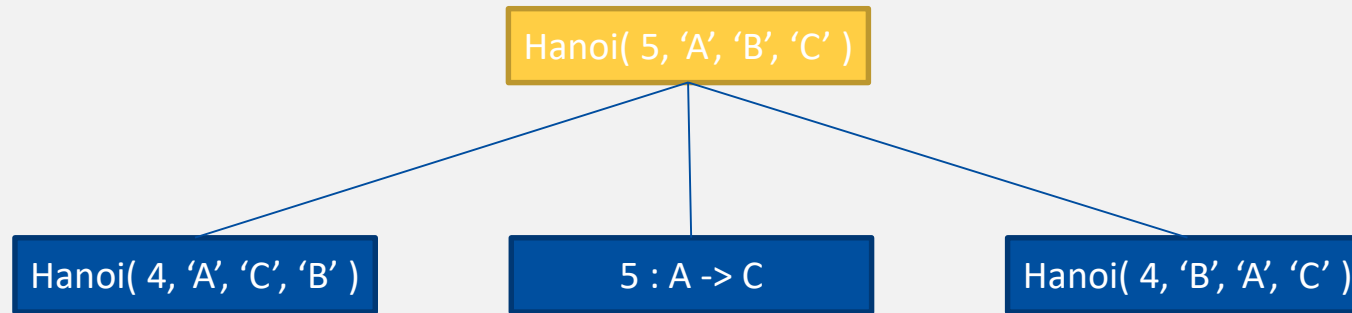
# 숙제



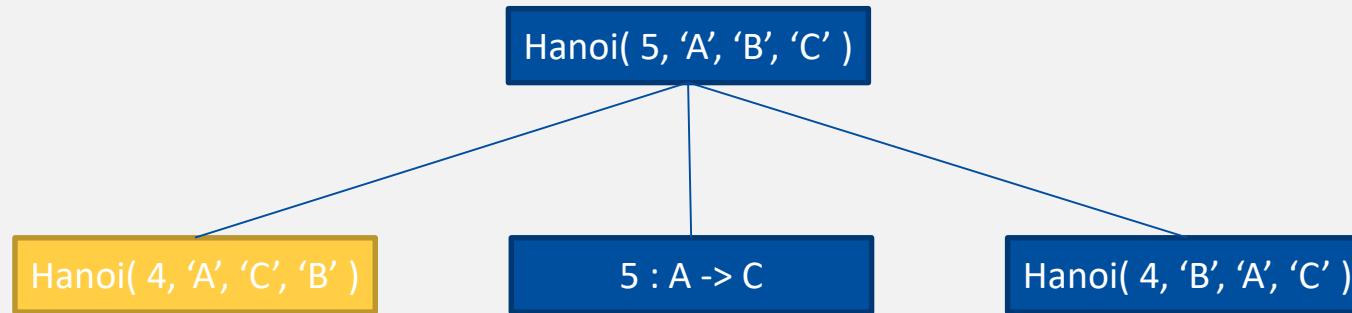
# 숙제

- 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?

# 숙제 – Hanoi ( n, s, m, e )

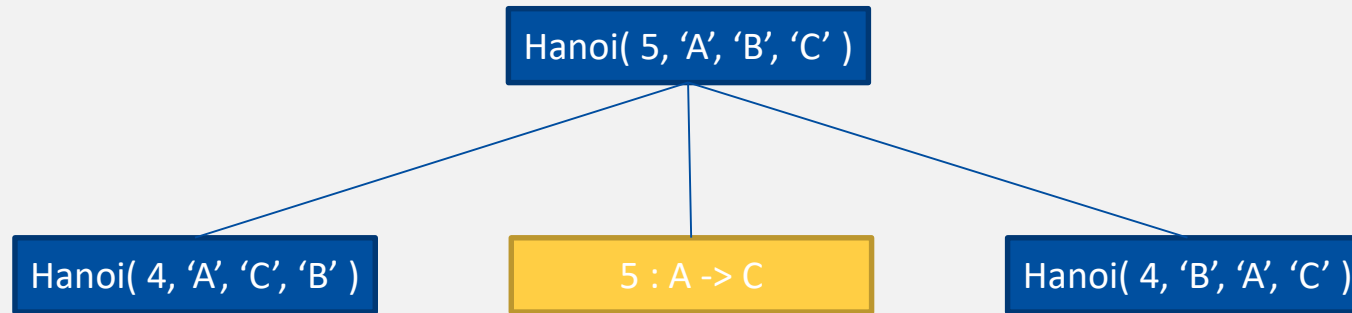


# 숙제 – Hanoi ( n, s, m, e )

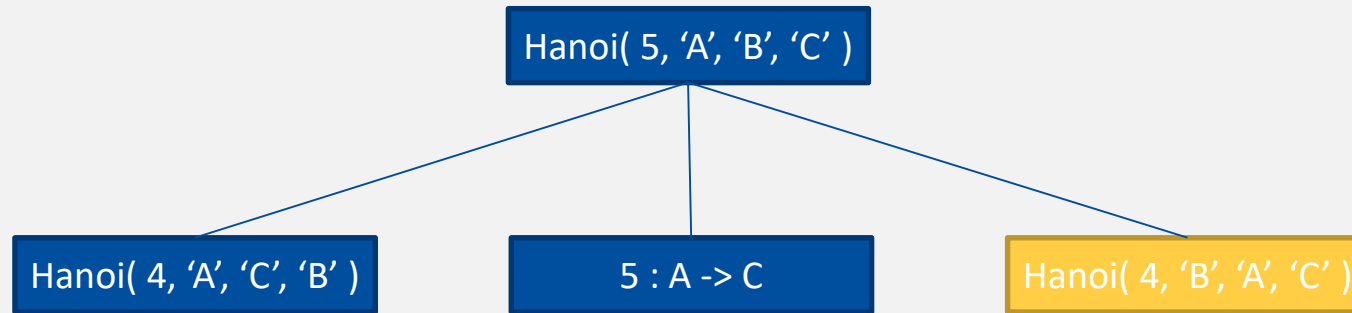




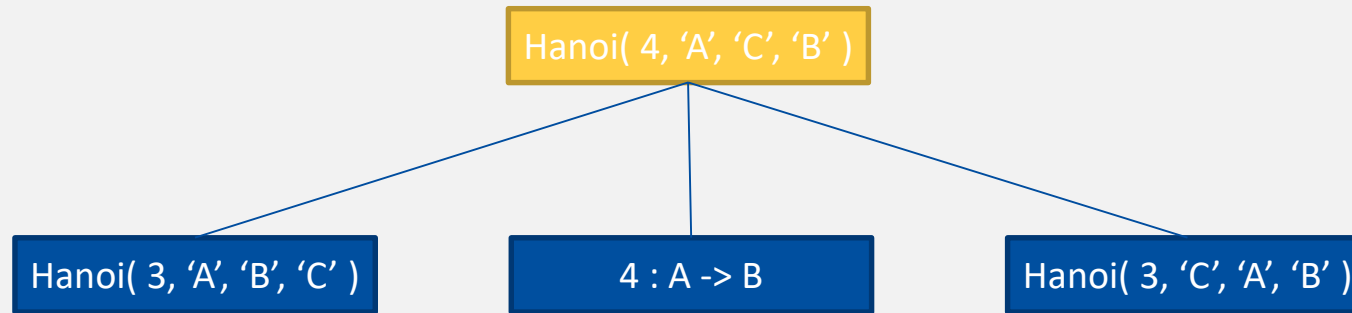
# 숙제 – Hanoi ( n, s, m, e )



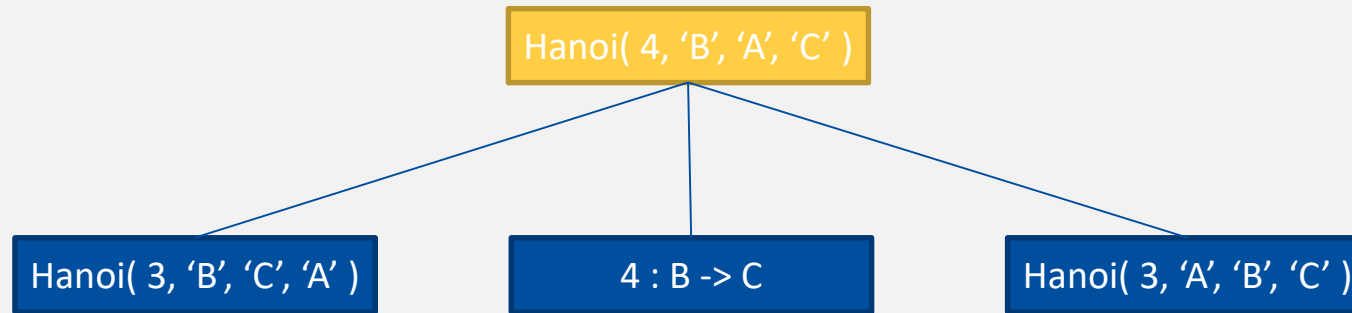
# 숙제 – Hanoi ( n, s, m, e )



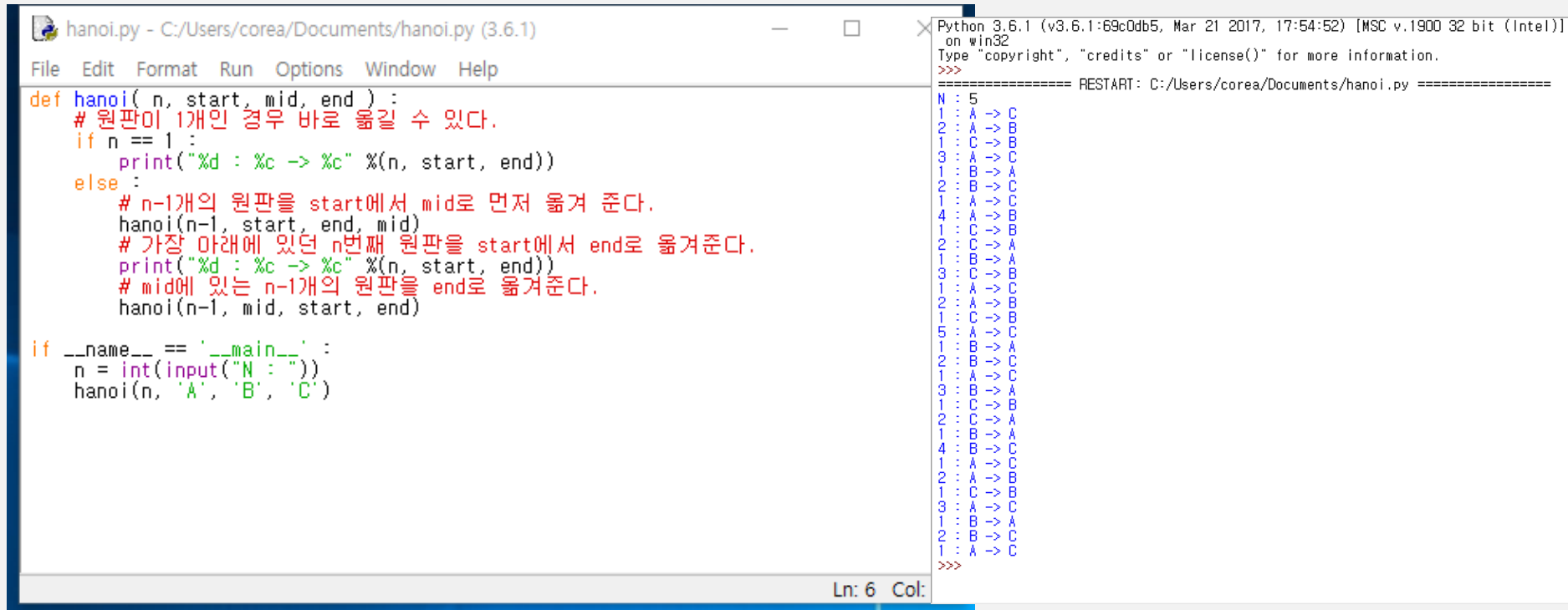
# 숙제 – Hanoi ( n, s, m, e )



# 숙제 – Hanoi ( n, s, m, e )



# 숙제 – 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')
```

```
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
>>>
```

Ln: 6 Col:

## 숙제- 파일명 : day-이름-학번-일시.py

- 주어진 날짜로부터 x일 후의 날이 몇 일이고 무슨 요일인지 계산해주는 program을 작성하시오.
  - 예: 2018.10.9 의 1000일 후?

# Homework

- 아래의 프로그램을 다음 수업 시작 전까지 ecampus로 upload

- 화일명

fibo-이름-학번.py

tree-이름-학번.py

hanoi-이름-학번.py

day-이름-학번.py

\* 파일이 여러 개일 경우 zip으로 묶어서 제출