파이썬 프로그래밍

7. 함수2

00

Contents

❖ 수업 목표

- SWAP 함수를 작성할 수 있다.
- 전역 변수와 지역변수를 구분하여 사용할 수 있다.
- global 키워드의 의미를 이해할 수 있다.
- 디폴트 매개변수를 사용할 수 있다.

❖ 세부 목표

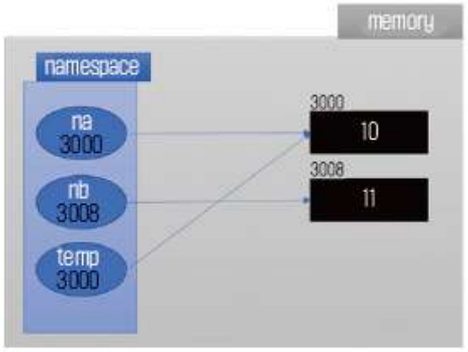
00

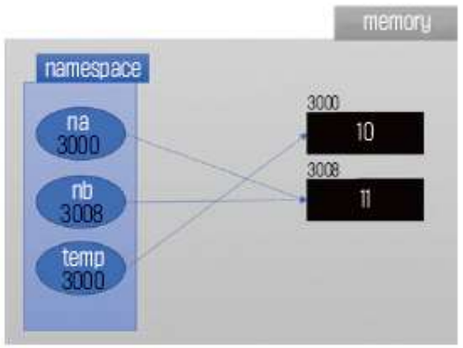
1. SWAP 기능 구현

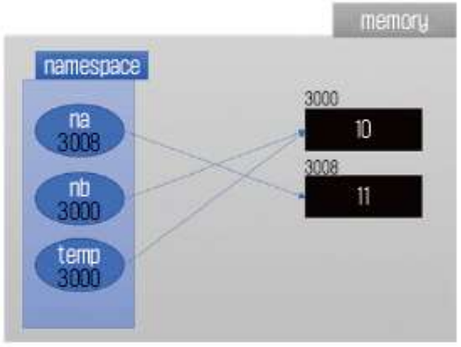
❖ SWAP 기능

- **SWAP(스왑)**
  - 두 데이터 서로 맞바꾸기

```
na=10
nb=11
print("na 값은", na, "nb 값은", nb)
temp=na
①
na=nb
②
nb=temp
③
print("na 값은", na, "nb 값은", nb)
```







〈화면 출력〉

```
na 값은  10 nb 값은  11
na 값은  11 nb 값은  10
```

00

1. SWAP 기능 구현

❖ SWAP 기능

- 파이썬은 l_value를 여러 개 사용할 수 있어 데이터를 서로 맞바꾸기(SWAP)위해 temp 변수가 필요 없음
  - a, b = b, a # a와 b를 교환
- 여기서는 일반적인 알고리즘을 이해하기 위해 l_value를 하나만 사용

00

1. SWAP 기능 구현

❖ SWAP 함수 생성

- 두 수를 swap 하는 기능을 함수로 정의 및 호출
- 왼쪽 예시 코드 동작 시, 화면 출력 내용 예측하기

```
def funca(pa, pb):
    temp=pa
    pa=pb
    pb=temp

na=10
nb=11
print("na 값은 ", na, "nb 값은 ", nb)
funca(na, nb)
print("na 값은 ", na, "nb 값은 ", nb)
```

〈화면 출력〉

00

1. SWAP 기능 구현

❖ SWAP 함수 생성

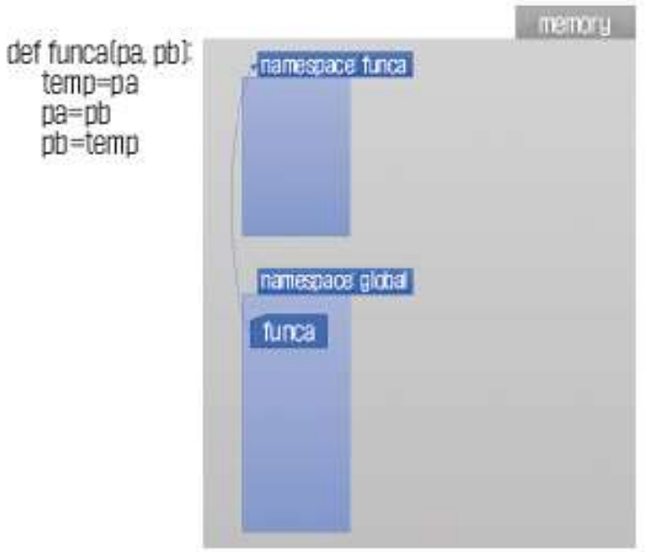
- 코딩하여 예측한 결과 확인하기

```
def funca(pa, pb):
    temp=pa
    pa=pb
    pb=temp
```

메모리 설명

[Step 1] funca 함수가 정의되어 있는 것을 확인하고 namespace: funca 공간을 확보한다.

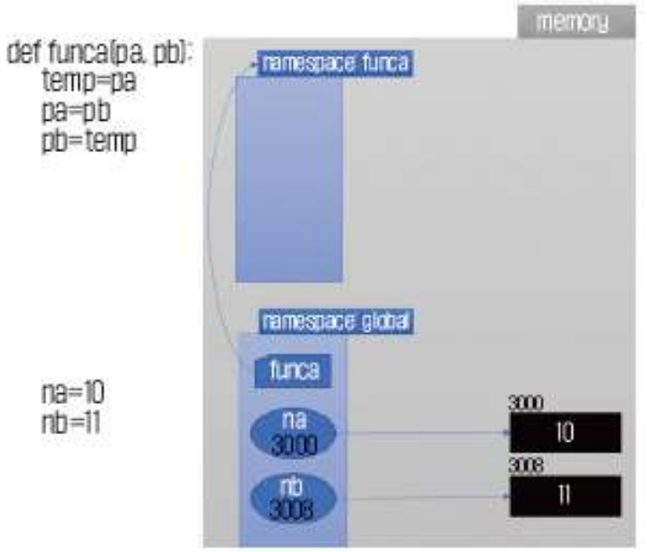
[Step 2] namespace:global 영역을 확보하고 funca 함수가 정의되어 있다는 기록을 해 놓는다.




```
na=10
nb=11
print("na 값은", na, "nb 값은 ", nb)
```

메모리 설명

[Step 3] namespace: global 영역에 주소 data를 넣을 수 있는 기억공간을 확보하고 변수명을 na로 붙이고 정수 데이터 10이 저장된 주소 3000을 저장한다. 또한, 기억공간을 확보하고 변수명을 nb로 붙이고 정수 데이터 11이 저장된 주소 3008을 저장한다.



1. SWAP 기능 구현

❖ SWAP 함수 생성

▪ 코딩하여 예측한 결과 확인하기

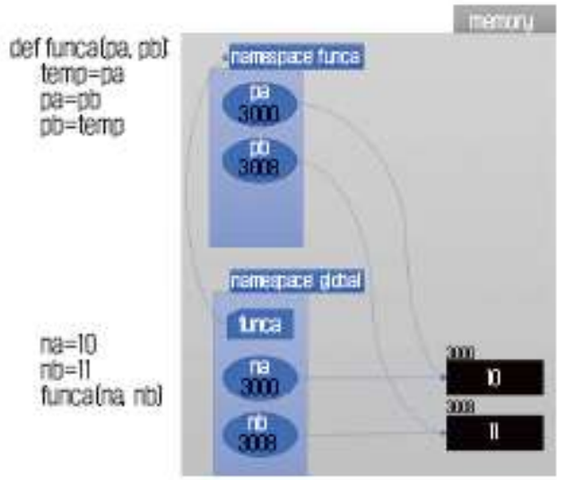
```
funca(pa, pb)
```

메모리 설명

[Step 4] funca 함수를 호출한다. 함수는 호출되면서 실제 실행된다.
namespace: funca 영역에 주소 data를 저장할 기억공간 2개를 확보하고 변수명 pa, pb라고 이름을 붙인다.

```
pa = na
pb = na
```

인수와 매개변수의 관계는 지금까지 설명했듯이 위와 같은 관계로 na 변수의 값 3000이 pa 변수에 nb 변수의 값 3008이 pb 변수에 저장된다.
그 결과 pa, na변수는 10을 가리키고 pb, nb 변수는 11을 가리킨다.

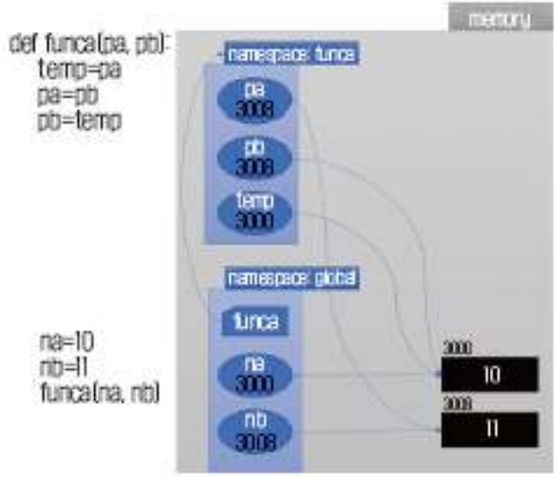



```
print("na 값은", na, "nb 값은 ", nb)
```

메모리 설명

[Step 5] funca 함수 명령어가 실행되면서 namespace: funca 영역에서 temp 변수를 추가로 생성 후 서로 맞바꾸기(SWAP)한 결과 pa에는 주소 data 3008, pb 변수에는 주소 data 3000, temp 변수에는 주소 data 3000이 저장된다. funca 영역에서 print("pa 값은", pa, "pb 값은 ", pb)를 실행하면 당연히 pa 값은 11 pb 값은 10이 출력된다.

그러나 namespace:global 영역에서 다음과 같이 실행하면 na 값은 10 nb 값은 11이 출력된다.



〈화면 출력〉

```
na 값은  10 nb 값은  11
na 값은  10 nb 값은  11
```

2. 전역 변수와 지역 변수

❖ 네임스페이스(namespace, 이름공간)

- 특정 객체를 이름에 따라 구분 가능한 범위
  - 즉, 같은 이름이 중복 사용되면 안되는 공간
- 변수 명을 정의하는 공간

- 네임스페이스 구별
  - 프로그램 전체 범위에서 사용되는 전역 이름공간
    - namespace:global
  - 함수 영역에서 사용되는 지역 이름공간
    - namespace: 함수명

2. 전역 변수

❖ 전역 변수(Global Variable)

- 프로그램 전체에서 접근 가능한 유효 범위를 갖는 변수
- 함수 외부에서 정의된 변수
- 프로그램 종료 시 같이 소멸
- 모든 함수에서 접근 가능

3. 지역 변수

❖ 지역 변수(Local Variable)

- 해당 변수가 생성된 범위(코드 블록) 내부에서만 유효
- 함수 내부에서 정의된 변수
- 함수 코드 종료 시 소멸
- 해당 함수 내부에서만 접근 가능

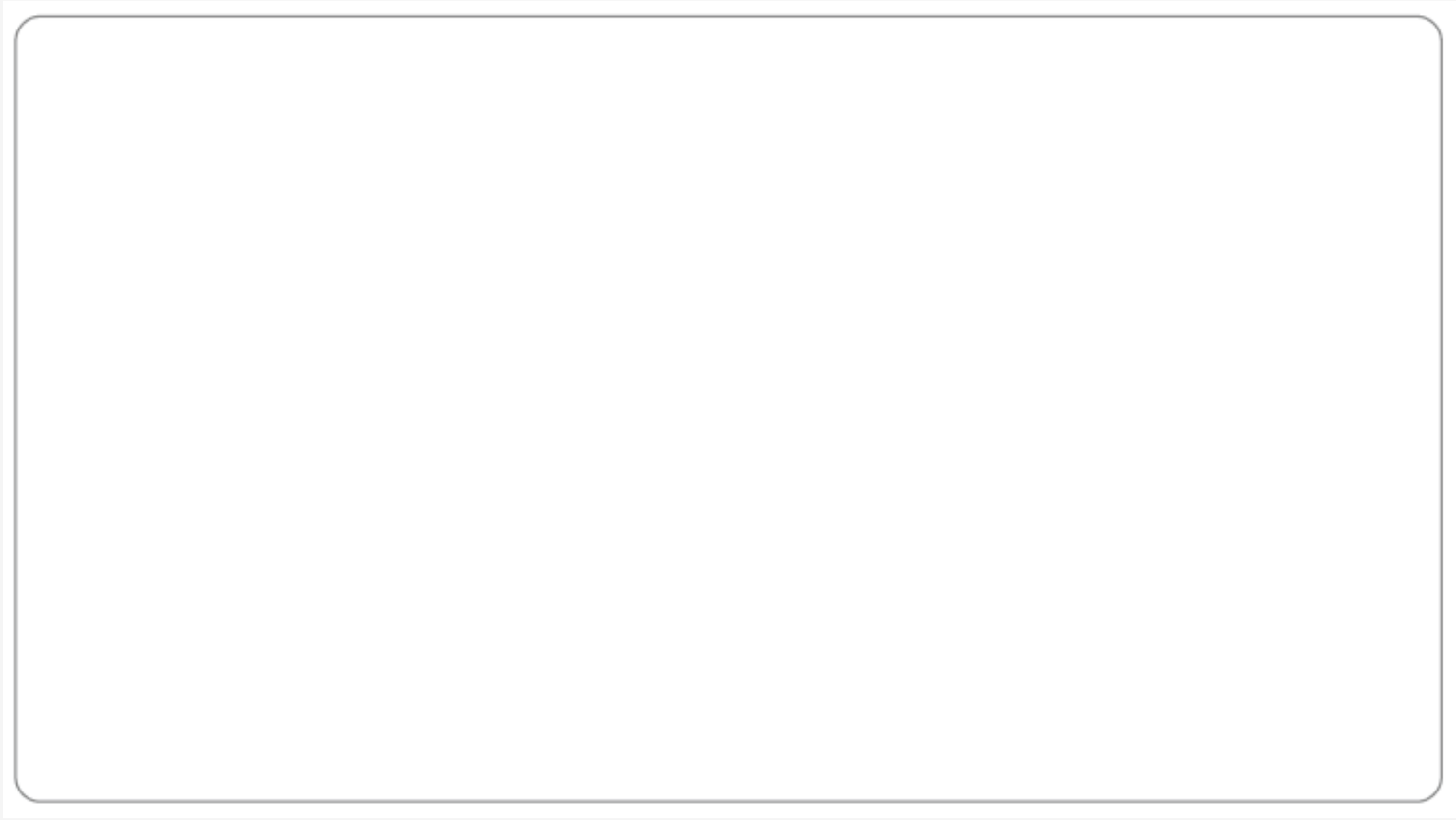
3. 전역 변수와 지역 변수

❖ SWAP 함수 생성 코드 개선

- 함수 영역에서 사용된 지역 변수의 값을 바꾼 후 return 수행
- return 값은 전역변수 재 할당
- 앞서 작성한 프로그램 내 funca 함수 동작 수행 후 변수값을 반환하여 SWAP된 결과를 화면에 출력하도록 프로그램 변경하기

3. 전역 변수와 지역 변수

- 앞서 작성한 프로그램 내 funca 함수 동작 수행 후 변수값을 반환하여 SWAP된 결과를 화면에 출력하도록 프로그램 변경하시오.



4. global 키워드

❖ global

- 변수 유효범위(변수 스코프, Variable Scope)
  - namespace에 정의되어 있는 접근 가능 범위

- 지정한 변수의 유효범위가 전역임을 선언
- 지역 변수의 생성을 차단

4. global 키워드

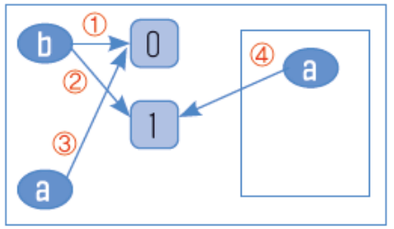
❖ 지역 변수를 전역변수로 선언

- 함수 외부에 정의된 변수 a와 함수 내 정의된 변수 a는 서로 다른 변수

```
b=0 # ---------------------> ①
print("b의 값은",b)
b=1 # ---------------------> ②
print("b의 값은",b)

def scope_test():
  a=1 # ------------------> ④
  print("scope_test() 함수 안의 a 값은", a)

a=0 # ---------------------> ③
print("scope_test() 함수 밖의 a 값은", a)
scope_test() # ------------> ④
print("socpe_test() 함수 호출 후 a 값은", a)
```



〈화면 출력〉

```
b의 값은 0
b의 값은 1
scope_test() 함수 밖의 a 값은 0
scope_test() 함수 안의 a 값은 1
scope_test() 함수 호출 후 a 값은 0
```

00

4. global 키워드

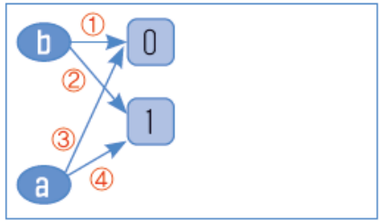
❖ 지역 변수를 전역변수로 선언

- global a 를 선언 시, 변수 a는 지역 변수가 아닌 전역 변수
  - ④ global a
    - a의 지역 변수 생성 차단
    - 전역 변수 a 사용 선언
  - ③ 전역 변수 a에 0 할당
  - ④ 함수 실행 시,
    - a는 기존 전역 변수 a를 의미

```
b=0 # ----------------------> ①
print("b의 값은",b)
b=1 # ----------------------> ②
print("b의 값은",b)

def scope_test():
  global a
  a=1 # ------------------> ④
  print("scope_test() 함수 안의 a 값은", a)

a=0 # ----------------------> ③
print("scope_test() 함수 밖의 a 값은", a)
scope_test() # -------------> ④
print("scope_test() 함수 호출 후 a 값은", a)
```



〈화면 출력〉

```
b의 값은 0
b의 값은 1
scope_test() 함수 밖의 a 값은 0
scope_test() 함수 안의 a 값은 1
scope_test() 함수 호출 후 a 값은 1
```

00

4. global 키워드

❖ 지역 변수를 전역변수로 선언

- 코드 설명
  - ① 변수 a 초기화 및 값 출력
  - ② scope_test( ) 함수 호출
  - ③ 함수 scope_test( ) 내
    지역 변수 a를 참조하고
    3을 더한 값을 출력

- 다음 코드의 에러 발생 이유?

```
b=0
print("b의 값은", b)
b=1
print("b의 값은", b)

def scope_test():
    a=a+3 # ----------------------------> ③
    print("scope_test() 함수 안의 a 값은", a)

a=0
print("scope_test() 함수 밖의 a 값은", a) # -------> ①
scope_test() # ----------------------------> ②
print("scope_test() 함수 호출 후  a 값은", a)
```

〈화면 출력〉

```
b의 값은 0
b의 값은 1
scope_test() 함수 밖의 a 값은 0
---------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Input In [3], in <cell line: 13>()
     11 a=0
     12 print("scope_test() 함수 밖의 a 값은", a)
---> 13 scope_test()
     14 print("scope_test() 함수 호출 후  a 값은", a)
Input In [3], in scope_test()
      7 def scope_test():
----> 8     a=a +3
      9     print("scope_test() 함수 안의 a 값은", a)
UnboundLocalError: local variable 'a' referenced before assignment
```

4. global 키워드

❖ 지역 변수를 전역변수로 선언

- UnboundLocalError 오류 발생
  - bound: 할당
- 할당(l_value 정의) 전 참조(r_value로 사용)됨

- 다음 두 가지 해결 방법 생각해보기
  - 1. scope_test( ) 함수 내 a 변수 값을 전역 변수 선언
  - 2. scope_test( ) 함수 내 a 변수를 지역 변수로 초기화

4. global 키워드

- 1. scope_test( ) 함수 내 a 변수 값을 전역 변수 선언

```
# a를 전역 변수로 사용하도록 변경
```

4. global 키워드

- 2. scope_test( ) 함수 내 a 변수를 지역 변수로 초기화

```
# a를 지역 변수로 사용하는 방법
```

00

5. 디폴트 매개변수

❖ 디폴트(default, 기본) 매개 변수

- 일반적으로 함수에서 매개변수와 인수는 일대일 대응 관계
- 인수와 매개변수의 개수 불일치 시, 오류 발생

- 디폴트 매개변수 사용시, 인수와 매개변수의 개수가 불일치해도 사전 정의된 기본 매개변수의 값을 사용 가능

syntax : 정확한 코딩

```
def 함수명(매개변수1=defaut값1. 매개변수2= defalut값2, ...):
```

5. 디폴트 매개변수

❖ 디폴트 매개변수 코드

- 2개 함수 정의
- persona()
- personb()

```
def persona(width, height): # ----------> ①
    print("함수디폴트값없음,width=", width,"height=", height)

def personb(width=4, height=3): # -------> ②
    print("함수디폴트값설정width=", width,"height=", height)
persona(10, 20) # ---------------------> ③
persona() # --------------------------> ④
personb() # --------------------------> ⑤
```

〈화면 출력〉

```
함수디폴트값없음,width=10 height=20
Traceback (most recent call last):
  File "C:/함수인수디폴트값.py", line 8, in <module>
    persona()
TypeError: persona() missing 2 required positional arguments: 'width' and
'height' # ------------------------> ⑥
>>>
```

5. 디폴트 매개변수

❖ 디폴트 매개변수 코드

- 해결 시도: 매개변수 개수를 맞추어 동일 이름의 새 함수를 정의
  - 동일한 이름의 함수를 중복 생성 불가

```
def persona(width, height): # ---------> ①
    print("함수디폴트값없음,width=", width,"height=", height)
def persona(): # -------------------> ⑦
    print("매개변수없는 함수")

def personb(width=4, height=3):
    print("함수디폴트값설정width=", width,"height=", height)

persona(10, 20)
persona() # ----------------------> ④
personb()
```

00

5. 디폴트 매개변수

❖ 디폴트 매개변수 코드

- 해결 시도: 기존 함수에 디폴트 매개변수를 사용하여 정의

```
def persona(width=11, height=21): # -------> ①
    print("함수디폴트값없음,width=", width,"height=", height)

def personb(width=4, height=3):
    print("함수디폴트값설정width=", width,"height=", height)

persona(10, 20)
persona() # ----------------------> ④
personb()
```

5. 디폴트 매개변수

❖ 타입 힌트 소개

- 함수의 매개변수와 반환값에 대해 예상되는 데이터 타입을 명시
  - Python 3.5 이상 사용 가능
  - 강제성 없음(에러 발생하지 않음)
  - 코드 가독성을 높이고 타입 검사를 수행할 수 있도록 돕는 기능

- 타입 힌트 문법
  - def 함수이름(매개변수이름: 타입, 매개변수이름: 타입) -> 반환타입:
  - # 함수 내용
  - return 값

  - 예)
    - def add(x: int, y: int) -> int:
    - return x + y
    - print(add(3, 5)) # 8
    - print(add(3.2, 5)) # 문제없이 실행되지만, 타입 힌트를 위반

6. 재귀 함수

❖ 재귀 함수 소개

- 함수가 자기 자신을 호출하는 프로그래밍 기법
  - 기저 조건(Base case): 재귀 호출을 멈추는 조건
  - 재귀 단계(Recursive step): 자기 자신을 호출하는 단계

- 타입 힌트 문법
  - def 함수이름(매개변수이름: 타입, 매개변수이름: 타입) -> 반환타입:
  - # 함수 내용
  - return 값
    - 예)

```
def count_down(n):
    if n == 0:
        print("완료!")
        return
    print(n)
    count_down(n-1)

count_down(5)
```

00

7. 연습 문제

- 다음 세 개의 프로그램의 화면 출력 결과를 작성해 보세요.

```
x=10
def fadd(num):
    b=x+num
    print("변수x값은", x)
    print("변수b값은", b)
fadd(10)
```

〈화면 출력〉

```
x=10
def fadd(num):
    x=x+num
    print("변수x값은", x)

fadd(10)
```

〈화면 출력〉

```
x=10
def fadd(num):
    global x
    x=x+num
    print("변수x값은", x)

fadd(10)
```

〈화면 출력〉

00

7. 연습 문제

- 현재 가격을 입력 받아 10% 할인된 가격을 출력하는 print_lower_price 함수를 정의 및 호출하시오.

00

7. 연습 문제

- 아래 코드의 실행 결과를 예측하시오.

```
def func1(num):
    return num + 4

a = func1(10)
b = func1(a)
c = func1(b)
print(c)
```

Notice

❖ 과제

- 1. SWAP 함수 작성하고 맞바꾼 값 출력하기
- 2. 함수 내 전역 변수 선언 코드 작성하기
- 3. 함수 내외에서 지역변수 영역 확인 코드 작성하기
- 4. global 키워드 사용 후 동일 이름 지역변수 선언 불가 확인하기
- 5. 디폴트 매개변수 사용하여 함수 작성하기

❖ 다음 수업 내용

- 자료구조와 알고리즘
  - 리스트 데이터 스왑, id함수, 최대값 찾기, 선택 정렬, 딕셔너리 매개변수