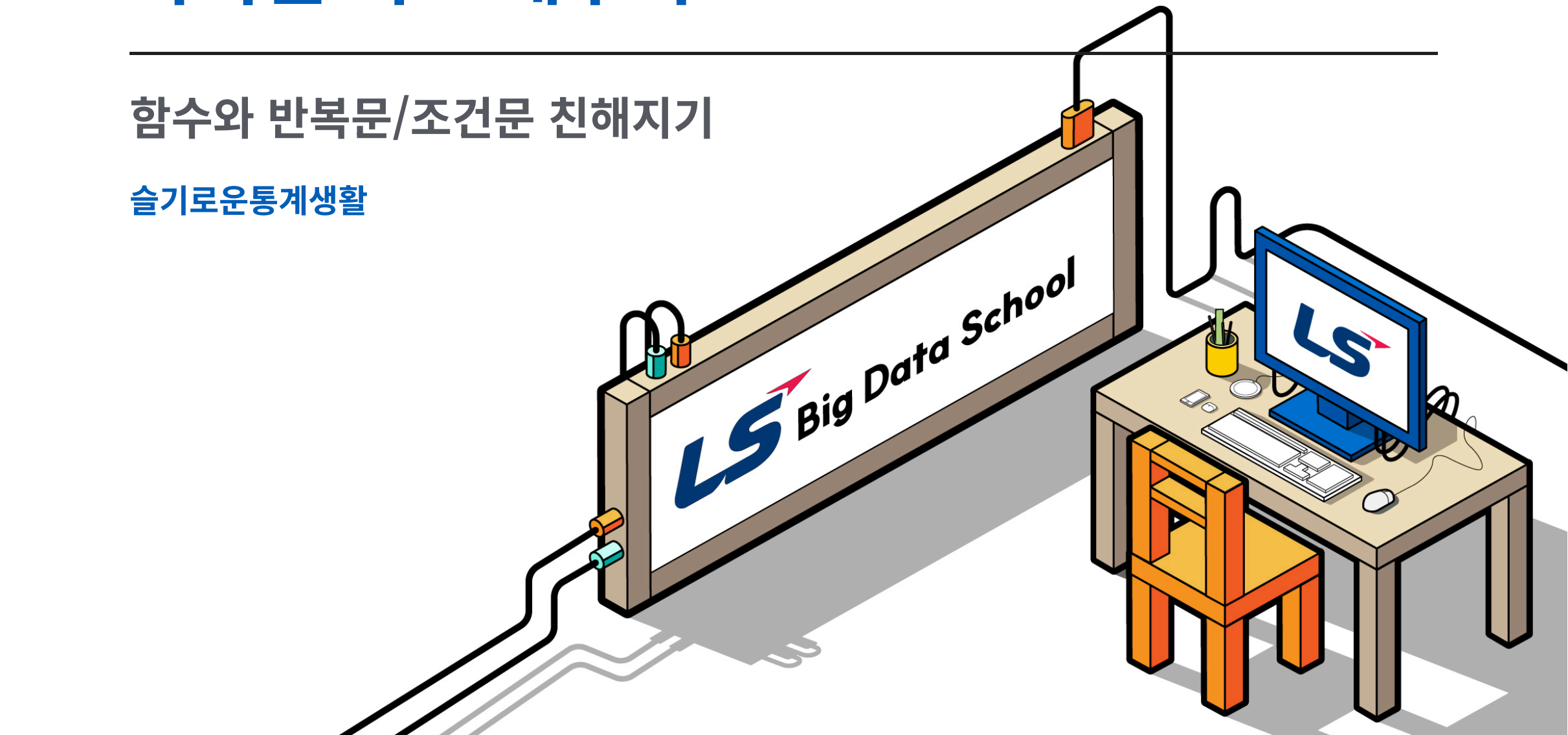
파이썬 기초 배우기

함수와 반복문/조건문 친해지기

슬기로운통계생활



코스 훑어보기.

파이썬의 함수에 대해 학습합니다.

사용자 정의 함수와 조건문, 반복문

Python 코드의 블록 구조

Python은 C, C++, Perl과 같은 블록 구조 (block-structured) 프로그래밍 언어 중 하나입니다. 그러나 Python은 중괄호 `{}` 대신 들여쓰기를 사용하여 블록을 구분합니다. 이러한 언어들의 코드는 다음과 같은 특징들을 가지고 있습니다.

- 들여쓰기를 사용하여 블록을 구분
- 블록 안에 여러 개의 구문들 (statements) 이 존재
- 구문들은 줄 바꿈 혹은 세미 콜론 `;` 으로 구분

다음의 Python 코드는 총 4개의 구문이 2줄에 걸쳐서 작성되었다고 볼 수 있습니다.

```
a = 3; b = 2
print(a); print(b)  # 2개 구문
```

```
## 3
## 2
```

사용자 함수 정의

Python 함수는 `def` 키워드를 사용하여 정의합니다. 함수에 필요한 내용은 입력값이 무엇인지 알려주는 `input`과 함수가 어떻게 작동하는지를 알려주는 `body` 부분으로 나눌 수 있습니다.

- 간단한 함수 구조

```
def function_name(input):
    result = input + 1
    return result
```

위 코드는 Python에서 함수의 기본 구조를 보여주고 있습니다. 아래는 함수의 각 부분에 대한 설명입니다.

- `function_name`: 함수의 이름
- `input`: 함수의 입력값
- `result`: 함수의 반환 값
- `return`: 함수의 결과를 반환하는 키워드

사용자 함수 정의

함수를 정의 한 후, 함수를 호출하여 결과를 확인할 수 있습니다.

```
function_name(2)
```

```
## 3
```

함수 기본 입력값 설정

내가 만든 함수의 기본 입력값을 설정하여 사용하는 방법에 대하여 알아봅시다.

입력값 설정에 따른 함수 결과

함수의 입력값에 기본값이 설정이 안 된 경우를 생각해봅시다. 이 경우, 함수를 실행할 때 필요한 입력값을 넣지 않고 실행한다면 함수가 제대로 작동을 하지 않을 것 입니다.

```
def g(x):
    result = x + 1
    return result

g()  # Error 발생
```

```
## TypeError: g() missing 1 required positional argument: 'x'
```

위의 코드에서는 함수 `g()`의 입력값으로 설정된 `x`의 값을 받아야 하는데, 아무런 값을 입력하지 않아서 에러가 발생하고 있습니다.

입력값 설정에 따른 함수 결과

```
def g(x=3):
    result = x + 1
    return result

g()
```

```
## 4
```

위에 정의 된 함수 `g()`는 입력값으로 아무것도 넣지 않아도, 기본으로 `x`에 3이 입력되어 정상적으로 작동하는 것을 확인 할 수 있습니다.

함수의 내용 출력 (Printing)

Python에서 정의된 함수의 내용을 확인하는 방법은 매우 간단합니다. 함수 이름을 입력하고 실행하면 함수 객체에 대한 정보가 출력됩니다.

```
print(g)  # 함수 객체 정보 출력
```

```
## <function g at 0x129cd79c0>
```

`inspect` 모듈을 사용하여 함수의 소스 코드를 확인할 수 있습니다.

```
import inspect
print(inspect.getsource(g))
```

Python 스러운 코딩 스타일은 무엇일까?

각 언어에는 그 언어의 사용자들이 많이 사용하는 코딩 스타일이 존재합니다. 많은 스타일이 있지만 PEP 8 스타일 가이드를 많이 사용합니다.

결과값 반환 함수 `return()`

Python의 함수는 `return` 문을 사용하여 결과값을 **명시적으로 반환**해야 합니다. `return` 문을 생략하면 함수는 `None`을 반환합니다. 따라서 함수를 정의할 때, `return` 문을 사용하는 것이 일반적입니다. 조기 반환(early return)을 할 때도 `return` 문을 사용합니다.

```
# 좋은 예
def find_abs(x):
    if x > 0:
        return x
    return -1 * x
```

```
# 좋은 예 2
def add_two(x, y):
    return x + y
```

함수 이름 정할 때 유의사항

PEP 8 스타일 가이드는 함수와 변수 이름을 작성할 때 다음과 같은 규칙을 권장합니다:

- 함수 이름과 변수 이름은 소문자와 밑줄 _을 사용하여 작성합니다 (snake_case).
- 클래스 이름은 각 단어의 첫 글자를 대문자로 작성합니다 (CamelCase).

위의 규칙을 사용해서 함수 이름을 작성하는 예를 살펴보겠습니다.

```python
# 좋은 예
def calculate_sum(a, b):
    return a + b
```

```python
# 나쁜 예
def CalculateSum(a, b):
    return a + b
```

코스 훑어보기.

파이썬의 조건문에 대해 학습합니다.

`if` and `else` 구문

특정 조건을 만족하는 경우와 만족하지 않는 경우에 다른 값을 부여하는 `if ... else ...` 구문의 사용법을 알아봅시다.

- `if ... else ...` 구문

```
if condition:
    statement_1
else:
    statement_2
```

위 코드는 `if-else` 조건문의 기본적인 형태입니다. 조건식의 결과에 따라 실행할 코드를 선택합니다. 주어진 코드에서 `condition`은 논리적인 조건식으로, 이 조건식이 참(true)일 경우 `statement_1`을 실행하고, 그렇지 않은 경우 `statement_2`를 실행합니다.

`if else` 구문 예제

```
x = 3
if x > 4:
    y = 1
else:
    y = 2
print(y)
```

```
## 2
```

위 코드에서는 변수 `x`에 3이 할당되어 있으므로, `if` 구문의 조건 `x > 4`는 거짓(false)입니다. 따라서 `else` 블록이 실행되어 변수 `y`에 값 `2`가 할당됩니다. 따라서 `y`의 최종 값은 `2`가 됩니다.

`if else` 구문 예제

위의 코드를 `A if 조건 else B` 구문을 사용하면 훨씬 깔끔하게 나타낼 수 있습니다.

```
y = 1 if x > 4 else 2
print(y)
```

```
## 2
```

`if else` 구문이 유용한 이유는 리스트 컴프리헨션을 통해 벡터화 연산이 가능하기 때문입니다.

`if else` 구문 예제

예를 들어, 아래 코드에서는 리스트 `x`의 각 원소가 `0`보다 큰지 여부를 확인하고, 조건에 따라 "양수" 또는 "음수"를 반환합니다.

```
x = [1, -2, 3, -4, 5]
result = ["양수" if value > 0 else "음수" for value in x]
print(result)
```

```
## ['양수', '음수', '양수', '음수', '양수']
```

`if else` 구문을 사용하면 조건에 따라 리스트의 각 원소를 쉽게 처리할 수 있으므로 데이터 분석 등에서 많이 활용됩니다.

조건 3개 이상의 경우 `elif()`

Python에서는 조건이 3개 이상으로 되는 경우 `elif` 구문을 사용합니다. `elif` 구문을 사용하면 여러 조건을 연속적으로 검사할 수 있습니다.

첫 번째 조건이 참이면 해당 블록이 실행되고, 그렇지 않으면 다음 조건을 검사합니다. 이렇게 모든 조건을 검사하고 마지막으로 else 블록이 실행됩니다.

`elif` 구문 예제

```
x = 0
if x > 0:
    result = "양수"
elif x == 0:
    result = "0"
else:
    result = "음수"
print(result)
```

```
## 0
```

이 코드는 변수 `x`의 값에 따라 "양수", "0", "음수" 중 하나의 값을 `result` 변수에 할당하여 출력합니다.

코스 훑어보기.

파이썬의 반복문에 대해 학습합니다.

반복되는 작업을 쉽게! 반복문

루프에 목 매지 마세요.

다른 언어들은 루프를 상당히 중요하게 가르칩니다. 왜냐하면 반복문 말고는 반복 작업을 처리할 수 있는 구문이 없기 때문이죠. 하지만 Python에서는 리스트 컴프리헨션(list comprehensions), 제너레이터(generator), 그리고 벡터화 연산을 지원하는 `NumPy`와 같은 패키지를 사용하면 반복 작업을 보다 간결하게 처리할 수 있습니다. 이러한 특성 때문에 "Python 코드는 간결하고, 고급지다."라는 소리를 듣는 이유가 됩니다.

꼭 알아야하는 반복문 2개

그럼에도 불구하고, 알고 있어야하는 반복문 2가지를 공부해보겠습니다.

- `for`
- `while`

입니다. 이러한 반복문은 타언어를 공부할 때에도 도움이 되고, 의미가 명확하기 때문에 코드의 가독성이 좋아진다는 이점이 있습니다. 필요한 경우 사용하면 좋은 경우가 종종 있기 때문에 알아둘 필요가 있습니다.

`for` 반복문(loop) 구문

`for` 반복문은 주어진 객체 안의 원소들에 접근하여 특정 작업을 반복하는 코드를 작성하고 싶을 때 사용합니다. `for` 반복문은 초기값, 조건식, 반복 후 실행될 코드로 구성되어 있습니다. `for` 반복문의 구문은 다음과 같습니다.

```
for 변수 in 범위:
    반복 실행할 코드
```

for 반복문 예시

```
for i in range(1, 4):
    print(f"Here is {i}")
```

```
## Here is 1
## Here is 2
## Here is 3
```

위 코드에서 `for` 반복문은 변수 `i`가 1부터 3까지 반복하도록 지정되어 있습니다. `print(f"Here is {i}")`는 "Here is"와 변수 `i`를 결합하여 출력하는 코드입니다.

for 반복문 예시

Python에서는 다음과 같이 리스트 컴프리헨션으로 같은 작업을 수행할 수 있습니다.

```
print([f"Here is {i}" for i in range(1, 4)])
```

```
## ['Here is 1', 'Here is 2', 'Here is 3']
```

그럼에도 불구하고, 직관적인 `for` 반복문은 이해가 쉽고, 가독성이 좋아서 자주 등장합니다.

참고

Python에서 `f`는 `f-string` 또는 포맷 문자열(format string)을 의미합니다. 이는 Python 3.6부터 도입된 기능으로, 문자열 내에서 변수를 직접 삽입할 수 있게 해주는 강력하고 간편한 방법입니다. `f-string`을 사용하면, 중괄호 `{}` 안에 변수나 표현식을 넣어 해당 값이 문자열 내에 삽입되도록 할 수 있습니다.

예를 들어:

```
name = "John"
age = 30
greeting = f"Hello, my name is {name} and I am {age} years old."
print(greeting)
```

```
## Hello, my name is John and I am 30 years old.
```

참고

각각 대응하는 내용을 출력하고 싶은 경우, 다음과 같이 `zip()`을 사용해서 `for` 문을 돌려줍니다.

```
names = ["John", "Alice"]
ages = [25, 30]

# 각 이름과 나이에 대해 별도로 인사말 생성
greetings = [f"Hello, my name is {name} and I am {age} years old." for name, age in zip(name

# 인사말 출력
for greeting in greetings:
    print(greeting)
```

```
## Hello, my name is John and I am 25 years old.
## Hello, my name is Alice and I am 30 years old.
```

`zip()` 함수는 여러 개의 이터러블(예: 리스트, 튜플)을 병렬적으로 묶어서 각 이터러블에서 동일한 인덱스에 있는 요소들을 하나의 튜플로 묶어주는 함수입니다. 이렇게 묶인 튜플들은 반복 가능한 객체로 반환됩니다.

`while` & `break` 루프 구문

`while` 반복문은 조건식이 참인 동안에 반복해서 실행되는 구문입니다. `while` 반복문의 구문은 다음과 같습니다.

```
while 조건식:
    반복 실행할 코드
```

`while` 안의 조건식이 참인 동안에는 `{}` 안의 내용이 무한번 반복해서 실행됩니다.

while 반복문 예시

```
i = 0
while i <= 10:
    i += 3
    print(i)
```

```
## 3
## 6
## 9
## 12
```

위 코드에서 `while` 반복문은 `i` 값이 `10` 이하일 때까지 반복하여 실행됩니다. `i` 값은 초기값으로 `0`이 할당되어 있으며, `i` 값을 3씩 증가시키는 코드인 `i += 3`와 `print(i)`가 반복적으로 실행됩니다.

while 반복문 주의사항

`while` 반복문은 조건식이 참일 때에만 실행되므로, 조건식을 설정하는 것이 매우 중요합니다. 만약 조건식이 항상 참이라면, 무한 반복(infinite loop)에 빠져 프로그램이 종료되지 않을 수 있습니다. 따라서 `while` 반복문을 사용할 때에는 조건식을 신중하게 설정하는 것이 필요합니다.

혹은 `break` 구문을 사용하여 `while` 구문이 계속 실행되어도, 중간에 빠져나올 수 있도록 설정할 수 있습니다.

```
i = 0
while True:
    i += 3
    if i > 10:
        break
    print(i)
```

```
## 3
## 6
## 9
```

코스 훑어보기.

파이썬의 함수 환경에 대해 학습합니다.

함수와 환경

함수에는 대응하는 환경이 따로 존재

- Python 함수의 구성요소
  - **입력값 (arguments)**: 함수가 입력으로 받는 값들입니다.
  - **내용 (body)**: 함수가 실행하는 코드 블록입니다.
  - **환경 (environment)**: 함수가 정의된 시점의 변수가 저장된 공간입니다.
- 함수가 만들어 질 때 존재하는 객체들 모음
  - 함수는 정의될 때, 현재 존재하는 모든 변수를 기억합니다. 이를 **클로저**라고 부릅니다.
- Global 환경에 존재하는 변수 `y`와 `my_func()` 안에 존재하는 `y`는 **다른 환경**에 같은 이름을 가진 변수
  - 함수 내에서 정의된 변수는 함수가 실행될 때만 유효하며, 함수 외부에 있는 동일한 이름의 변수와는 독립적으로 존재합니다.

함수와 환경 예제

```
y = 2
def my_func(x):
    y = 1
    result = x + y
    return result

print(y)  # 이 코드는 전역 변수 y의 값을 출력합니다.
```

```
## 2
```

```
my_func(2)
```

```
## 3
```

- **설명**: 여기서 함수 `my_func` 내부에서 정의된 `y`는 함수 내에서만 사용되는 지역 변수입니다. 함수 밖에 있는 `y`는 전역 변수로, 함수 내의 `y`와는 전혀 다른 변수입니다.

Python 환경의 계층 구조

함수 안에 함수 있다.

- **top-level인** `Global Env.` **에 정의된** `outer_func()`
  - Python에서는 함수 내에 다른 함수를 정의할 수 있습니다. 이는 **중첩 함수**(nested function)라고 합니다.

```
def outer_func():
    def inner_func(input):
        return input + 2

    # check env of inner_func()
    print(inner_func.__code__.co_varnames)
```

- **설명**: `inner_func`는 `outer_func` 내부에 정의된 함수입니다. `inner_func`의 환경은 `outer_func` 내에서만 접근 가능합니다. `inner_func.__code__.co_varnames`는 함수 `inner_func`의 지역 변수들을 출력합니다.

함수 안에 함수 있다.

```
print(inner_func.__code__.co_varnames)
```

```
## NameError: name 'inner_func' is not defined
```

- `inner_func()`**의 환경은** `outer_func()`**에서 접근 가능**
  - 함수 `inner_func`는 `outer_func` 내에 정의되었으므로, `outer_func` 내에서 호출하거나 접근할 수 있습니다.

```
outer_func()
```

```
## ('input',)
```

- **설명**: 위의 코드는 `outer_func`를 호출하며, `inner_func`의 변수 정보를 출력합니다.

`global` 키워드와 환경구조

상위 환경과의 교류

`global` 키워드를 사용하면 함수 내에서 전역 변수를 참조하거나 수정할 수 있습니다. 다음 예제를 살펴보죠.

```
y = 2
def my_func():
    global y
    y = 1

print(y)  # 전역 변수 y의 값을 출력 (2)
```

```
## 2
```

```
my_func()
print(y)  # 전역 변수 y의 값을 출력 (1)
```

```
## 1
```

여기서 `global y`는 `my_func` 함수가 전역 변수 `y`를 수정하도록 합니다. `my_func` 함수가 호출되기 전에는 `y`의 값이 2였지만, `my_func`가 호출된 후에는 1로 변경됩니다.

연습문제 1: 함수 정의와 기본값 설정

1. 다음 요구사항에 맞는 함수를 작성하세요:
    - 함수 이름: add_numbers
    - 입력값: 두 개의 숫자 a와 b (기본값 a=1, b=2)
    - 반환값: 두 숫자의 합
2. 함수를 호출하여 기본값을 사용한 결과와, a=5, b=7일 때의 결과를 각각 출력하세요.

출력 예시:

- 기본값 사용: 3
- 입력값 사용: 12

연습문제 1: 함수 정의와 기본값 설정 해답

```python
def add_numbers(a=1, b=2):
    return a + b

# 기본값 사용
print("기본값 사용:", add_numbers())
```

```
## 기본값 사용: 3
```

```python
# 입력값 사용
print("입력값 사용:", add_numbers(5, 7))
```

```
## 입력값 사용: 12
```

연습문제 2: 조건문 사용

1. 다음 요구사항에 맞는 함수를 작성하세요:
    - 함수 이름: check_sign
    - 입력값: 하나의 숫자 x
    - 출력값: x가 양수면 “양수”, 음수면 “음수”, 0이면 “0”이라는 문자열 반환
2. 숫자 10, -5, 0에 대해 함수를 호출하고 결과를 출력하세요.

출력 예시:

- 10: 양수
- -5: 음수
- 0: 0

연습문제 2: 조건문 사용 해답

```python
def check_sign(x):
    if x > 0:
        return "양수"
    elif x < 0:
        return "음수"
    else:
        return "0"
```

```python
# 함수 호출
print("10:", check_sign(10))
```

```
## 10: 양수
```

```python
print("-5:", check_sign(-5))
```

```
## -5: 음수
```

```python
print("0:", check_sign(0))
```

```
## 0: 0
```

연습문제 3: 반복문 사용

1. 1부터 10까지 숫자를 출력하는 함수를 작성하세요:
    - 함수 이름: print_numbers
    - 출력값: 1부터 10까지의 숫자를 줄바꿈하여 출력
2. 함수를 호출하여 결과를 확인하세요.

출력 예시:

```
1
2
3
...
10
```

연습문제 3: 반복문 사용 해답

```python
def print_numbers():
    for i in range(1, 11):
        print(i)

# 함수 호출
print_numbers()
```

```
## 1
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9
## 10
```

연습문제 4: 중첩 함수 사용

1. 다음 요구사항에 맞는 함수를 작성하세요:
    - 함수 이름: outer_function
    - 내부에 inner_function을 정의하고, inner_function은 입력값에 2를 더한 값을 반환
    - outer_function은 inner_function을 호출하여 결과를 반환
2. 숫자 5를 입력값으로 outer_function을 호출하고 결과를 출력하세요.

출력 예시:

- 결과: 7

연습문제 4: 중첩 함수 사용 해답

```
def outer_function(x):
    def inner_function(y):
        return y + 2
    return inner_function(x)

# 함수 호출
print("결과:", outer_function(5))
```

```
## 결과: 7
```

연습문제 5: while 반복문과 break

1. 다음 요구사항에 맞는 함수를 작성하세요:
    - 함수 이름: find_even
    - 입력값: 시작 숫자 start
    - 동작: start부터 시작하여 처음으로 나오는 짝수를 반환
    - 짝수를 찾으면 break로 반복문 종료
2. 함수에 start=3을 입력하여 호출하고 결과를 출력하세요.

출력 예시:

- 결과: 4

연습문제 5: while 반복문과 break 해답

```
def find_even(start):
    while True:
        if start % 2 == 0:
            return start
        start += 1

# 함수 호출
print("결과:", find_even(3))
```

```
## 결과: 4
```