UNIT 33

클로저 사용하기

33.1 변수의 사용 범위 알아보기

변수의 사용 범위 알아보기

global_variable.py

```
x = 10          # 전역 변수
def foo():
    print(x)    # 전역 변수 출력

foo()
print(x)        # 전역 변수 출력
```

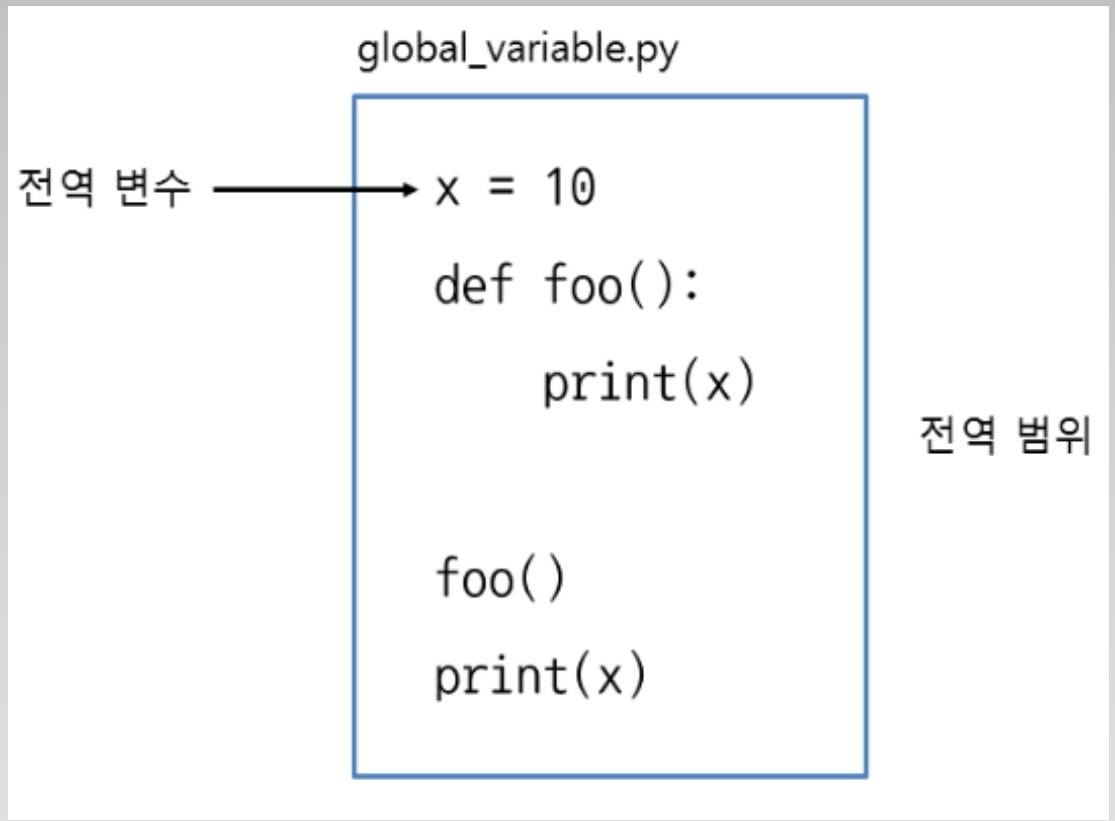
실행 결과

```
10
10
```

- 함수를 포함하여 스크립트 전체에서 접근할 수 있는 변수를 전역 변수(global variable)라고 부름
- 전역 변수에 접근할 수 있는 범위를 전역 범위(global scope)라고 함

33.1 변수의 사용 범위 알아보기

▼ 그림 전역 변수와 전역 범위



33.1 변수의 사용 범위 알아보기

변수의 사용 범위 알아보기

local_variable.py

```
def foo():
    x = 10      # foo의 지역 변수
    print(x)    # foo의 지역 변수 출력

foo()
print(x)        # 에러. foo의 지역 변수는 출력할 수 없음
```

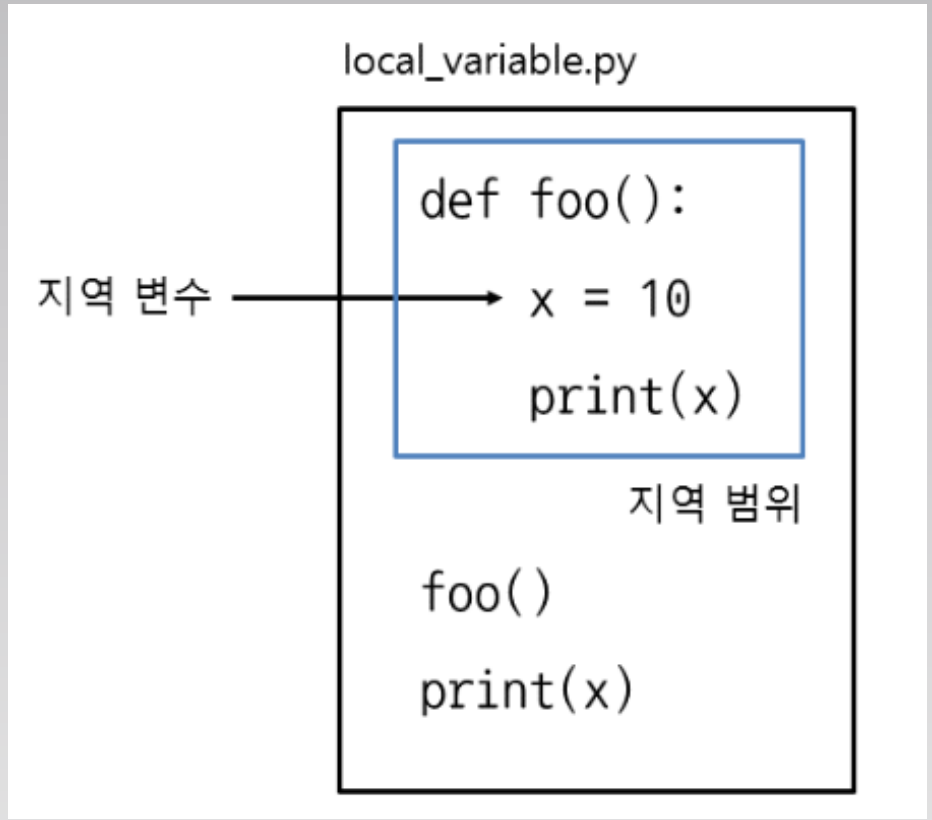
실행 결과

```
10
Traceback (most recent call last):
  File "C:\project\local_variable.py", line 6, in <module>
    print(x)        # 에러. foo의 지역 변수는 출력할 수 없음
NameError: name 'x' is not defined
```

- 실행을 해보면 x가 정의되지 않았다는 에러가 발생하는것은 변수 x는 함수 foo 안에서 만들었기 때문에 foo의 지역 변수(local variable)임
- 지역 변수는 변수를 만든 함수 안에서만 접근할 수 있고, 함수 바깥에서는 접근할 수 없음
- 지역 변수를 접근할 수 있는 범위를 지역 범위(local scope)라고 함

33.1 변수의 사용 범위 알아보기

▼ 그림 지역 변수와 지역 범위



33.1 변수의 사용 범위 알아보기

함수 안에서 전역 변수 변경하기

global_local_variable.py

```
x = 10          # 전역 변수
def foo():
    x = 20      # x는 foo의 지역 변수
    print(x)    # foo의 지역 변수 출력

foo()
print(x)        # 전역 변수 출력
```

실행 결과

```
20
10
```

- 전역 변수 x가 있고, foo에서 지역 변수 x를 새로 만들게 됨
- 이 둘은 이름만 같을 뿐 서로 다른 변수임

33.1 변수의 사용 범위 알아보기

함수 안에서 전역 변수 변경하기

- 함수 안에서 전역 변수의 값을 변경하려면 global 키워드를 사용해야 함

• global 전역변수

function_global_keyword.py

```
x = 10          # 전역 변수
def foo():
    global x    # 전역 변수 x를 사용하겠다고 설정
    x = 20      # x는 전역 변수
    print(x)    # 전역 변수 출력

foo()
print(x)        # 전역 변수 출력
```

실행 결과

```
20
20
```

33.1 변수의 사용 범위 알아보기

함수 안에서 전역 변수 변경하기

- 만약 전역 변수가 없을 때 함수 안에서 global을 사용하면 해당 변수는 전역 변수가 됨

```
# 전역 변수 x가 없는 상태
def foo():
    global x    # x를 전역 변수로 만듦
    x = 20      # x는 전역 변수
    print(x)    # 전역 변수 출력

foo()
print(x)        # 전역 변수 출력
```

33.2 함수 안에서 함수 만들기

함수 안에서 함수 만들기

- 다음과 같이 def로 함수를 만들고 그 안에서 다시 def로 함수를 만들면 됨

```
def 함수이름1():
    코드
    def 함수이름2():
        코드
```

- 간단하게 함수 안에서 문자열을 출력하는 함수를 만들고 호출해봄

function_in_function.py

```
def print_hello():
    hello = 'Hello, world!'
    def print_message():
        print(hello)
    print_message()

print_hello()
```

실행 결과

```
Hello, world!
```

33.2 함수 안에서 함수 만들기

지역 변수의 범위

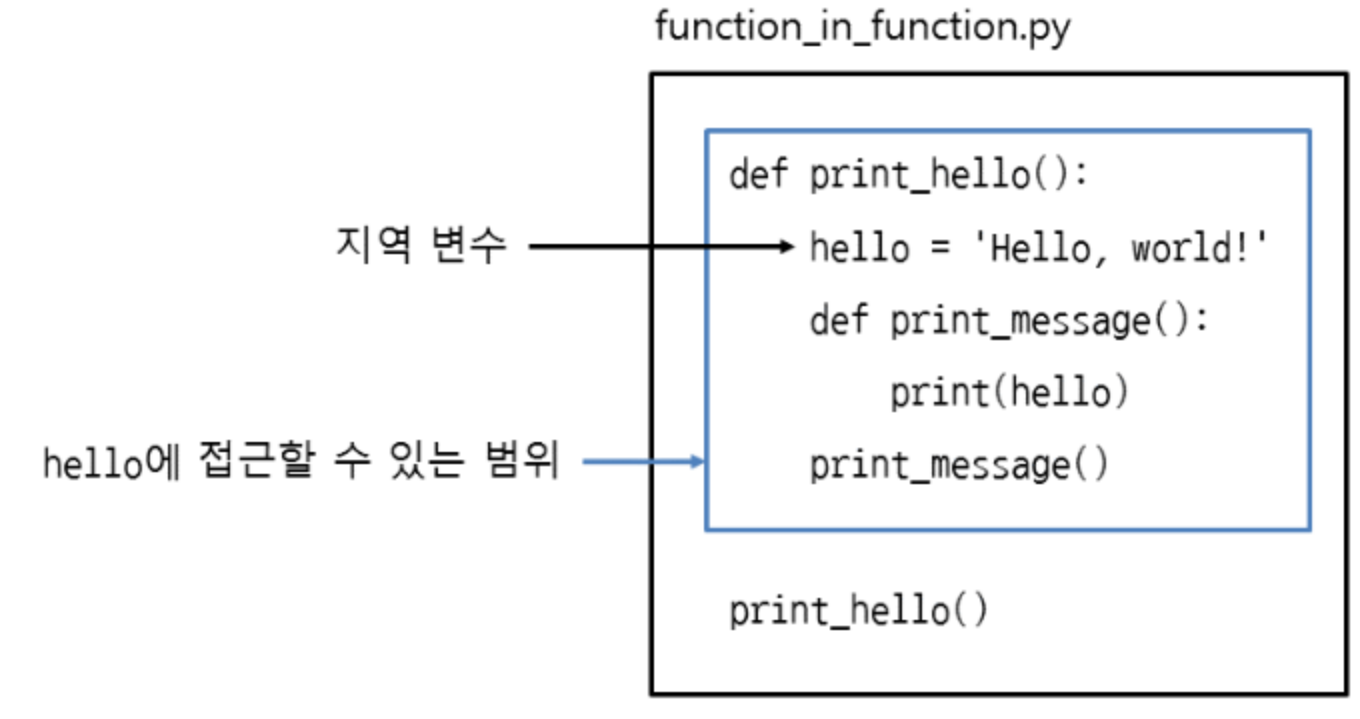
- 안쪽 함수 print_message에서는 바깥쪽 함수 print_hello의 지역 변수 hello를 사용할 수 있음

```
def print_hello():
    hello = 'Hello, world!'
    def print_message():
        print(hello)    # 바깥쪽 함수의 지역 변수를 사용
```

- 즉, 바깥쪽 함수의 지역 변수는 그 안에 속한 모든 함수에서 접근할 수 있음

33.2 함수 안에서 함수 만들기

▼ 그림 함수의 지역 변수에 접근할 수 있는 범위



33.2 함수 안에서 함수 만들기

지역 변수 변경하기

- 다음과 같이 안쪽 함수 B에서 바깥쪽 함수 A의 지역 변수 x를 변경해보자

function_local_error.py

```
def A():
    x = 10          # A의 지역 변수 x
    def B():
        x = 20      # x에 20 할당

    B()
    print(x)        # A의 지역 변수 x 출력

A()
```

실행 결과

```
10
```

33.2 함수 안에서 함수 만들기

지역 변수 변경하기

- 파이썬에서는 함수에서 변수를 만들면 항상 현재 함수의 지역 변수가 됨

```
def A():
    x = 10          # A의 지역 변수 x
    def B():
        x = 20      # B의 지역 변수 x를 새로 만듦
```

33.2 함수 안에서 함수 만들기

지역 변수 변경하기

- 현재 함수의 바깥쪽에 있는 지역 변수의 값을 변경하려면 nonlocal 키워드를 사용해야 함

• nonlocal 지역변수

function_nonlocal_keyword.py

```
def A():
    x = 10          # A의 지역 변수 x
    def B():
        nonlocal x    # 현재 함수의 바깥쪽에 있는 지역 변수 사용
        x = 20        # A의 지역 변수 x에 20 할당

    B()
    print(x)      # A의 지역 변수 x 출력

A()
```

실행 결과

```
20
```

- nonlocal은 현재 함수의 지역 변수가 아니라는 뜻이며 바깥쪽 함수의 지역 변수를 사용함

33.2 함수 안에서 함수 만들기

nonlocal이 변수를 찾는 순서

- nonlocal은 현재 함수의 바깥쪽에 있는 지역 변수를 찾을 때 가장 가까운 함수부터 먼저 찾음

function_in_function_nonlocal_keyword.py

```
def A():
    x = 10
    y = 100
    def B():
        x = 20
        def C():
            nonlocal x
            nonlocal y
            x = x + 30
            y = y + 300
            print(x)
            print(y)
        C()
    B()

A()
```

실행 결과

```
50
400
```

33.2 함수 안에서 함수 만들기

global로 전역 변수 사용하기

- 함수가 몇 단계든 상관없이 global 키워드를 사용하면 무조건 전역 변수를 사용하게 됨

function_in_function_global_keyword.py

```
x = 1
def A():
    x = 10
    def B():
        x = 20
        def C():
            global x
            x = x + 30
            print(x)
        C()
    B()

A()
```

실행 결과

```
31
```

33.3 클로저 사용하기

클로저 사용하기

- 다음은 함수 바깥쪽에 있는 지역 변수 a, b를 사용하여 a * x + b를 계산하는 함수 mul_add를 만든 뒤에 함수 mul_add 자체를 반환함

closure.py

```
def calc():
    a = 3
    b = 5
    def mul_add(x):
        return a * x + b    # 함수 바깥쪽에 있는 지역 변수 a, b를 사용하여 계산
    return mul_add          # mul_add 함수를 반환

c = calc()
print(c(1), c(2), c(3), c(4), c(5))
```

실행 결과

```
8 11 14 17 20
```

33.3 클로저 사용하기

클로저 사용하기

- 그다음에 함수 mul_add에서 a와 b를 사용하여 a * x + b를 계산한 뒤 반환함

```
def calc():
    a = 3
    b = 5
    def mul_add(x):
        return a * x + b    # 함수 바깥쪽에 있는 지역 변수 a, b를 사용하여 계산
```

- 함수 mul_add를 만든 뒤에는 이 함수를 바로 호출하지 않고 return으로 함수 자체를 반환함(함수를 반환할 때는 함수 이름만 반환해야 하며 ( )(괄호)를 붙이면 안 됨)

```
    return mul_add          # mul_add 함수를 반환
```

33.3 클로저 사용하기

클로저 사용하기

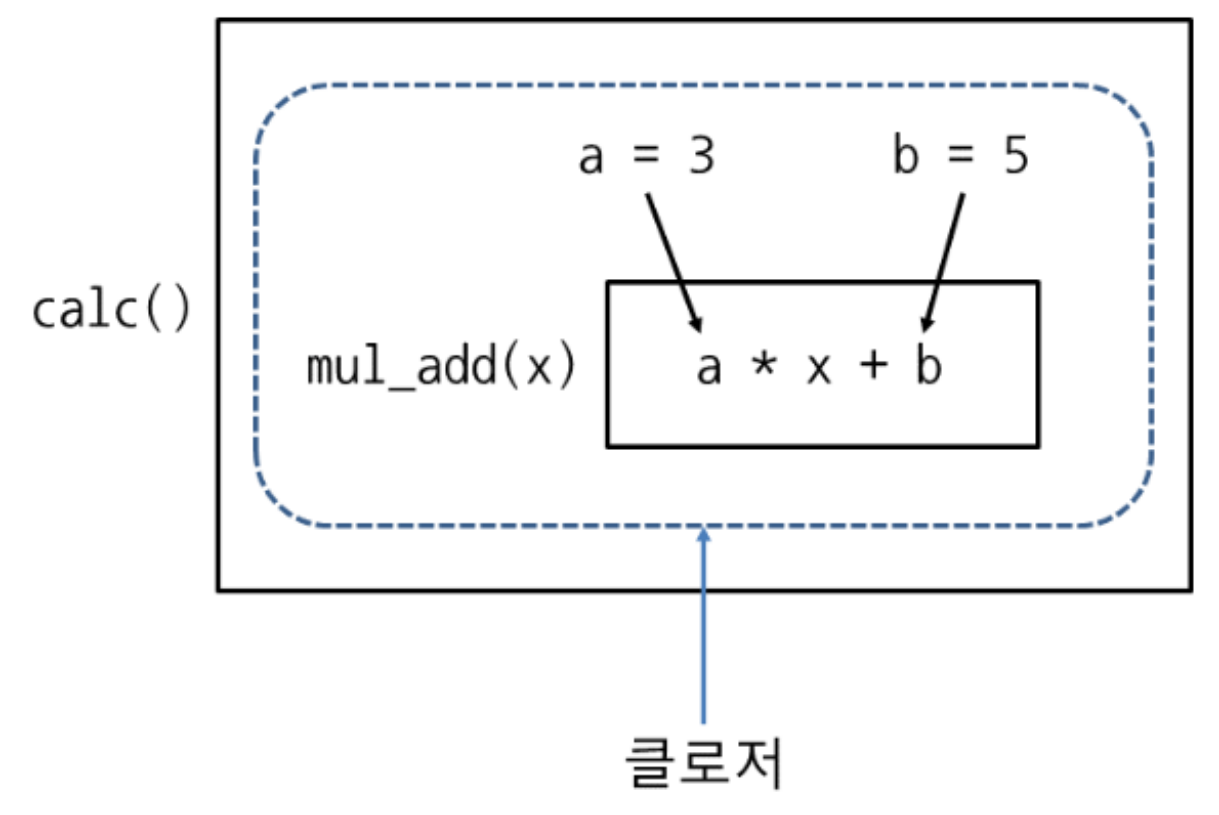
- 다음과 같이 함수 calc를 호출한 뒤 반환값을 c에 저장함
- calc에서 mul_add를 반환했으므로 c에는 함수 mul_add가 들어감
- c에 숫자를 넣어서 호출해보면 a * x + b 계산식에 따라 값이 출력됨

```
c = calc()
print(c(1), c(2), c(3), c(4), c(5))    # 8 11 14 17 20
```

- 함수를 둘러싼 환경(지역 변수, 코드 등)을 계속 유지하다가, 함수를 호출할 때 다시 꺼내서 사용하는 함수를 클로저(closure)라고 함
- 여기서는 c에 저장된 함수가 클로저임

33.3 클로저 사용하기

▼ 그림 클로저의 개념



33.3 클로저 사용하기

클로저 사용하기

- 클로저를 사용하면 프로그램의 흐름을 변수에 저장할 수 있음
- 클로저는 지역 변수와 코드를 묶어서 사용하고 싶을 때 활용함
- 클로저에 속한 지역 변수는 바깥에서 직접 접근할 수 없으므로 데이터를 숨기고 싶을 때 활용함

33.3 클로저 사용하기

lambda로 클로저 만들기

closure_lambda.py

```
def calc():
    a = 3
    b = 5
    return lambda x: a * x + b    # 람다 표현식을 반환

c = calc()
print(c(1), c(2), c(3), c(4), c(5))
```

실행 결과

```
8 11 14 17 20
```

- 람다는 이름이 없는 익명 함수를 뜻하고, 클로저는 함수를 둘러싼 환경을 유지했다가 나중에 다시 사용하는 함수를 뜻함

33.3 클로저 사용하기

클로저의 지역 변수 변경하기

- 클로저의 지역 변수를 변경하고 싶다면 nonlocal을 사용하면 됨
- 다음은 a * x + b의 결과를 함수 calc의 지역 변수 total에 누적함

closure_nonlocal.py

```
def calc():
    a = 3
    b = 5
    total = 0
    def mul_add(x):
        nonlocal total
        total = total + a * x + b
        print(total)
    return mul_add

c = calc()
c(1)
c(2)
c(3)
```

실행 결과

```
8
19
33
```