위 예시를 보면 3이 출력되는 것이 너무 당연하게 느껴질 것입니다. 3을 매개변수인 `a`로 전달해서 `a`에 3을 더하고 출력은 `num`을 했으니 당연히 3이 출력됩니다. 이때, 함수 밖에서 선언된 `num`은 전역변수이고, 함수 안에서 선언된 `a`는 지역변수입니다. `a`는 매개변수로서 단지 인자를 받아오는 역할을 할 뿐만 아니라 `plusThree(a)` 자체로 변수 `a`를 선언함과 동시에 3을 복사해와 할당하는 것입니다. 이렇게 함수 내에서 선언하고 사용한 변수인 `a`는 지역변수입니다. **매개변수는 함수 내에서 사용할 변수의 '선언'과 전달인자의 값을 '복사'해와 할당하는 역할을 하는 것을 잘 기억해두기를 바랍니다.**

지역 변수와 전역 변수는 아래와 같이 아주 큰 차이점이 두 가지가 존재합니다.

- 메모리에 존재하는 시간
- 변수에 접근할 수 있는 범위

먼저 '메모리에 존재하는 시간'에 대해 알아보겠습니다. 함수 내에서 선언된 지역변수는 함수 내에서 선언되는 순간 메모리에 저장되고 함수를 벗어나면(종료되면) 자동으로 소멸됩니다. 하지만 전역변수는 프로그램이 시작되어 선언되는 순간부터 메모리에 저장되고 코드 전체가 끝나 프로그램이 종료될 때까지 메모리를 차지하고 있습니다.

지역변수는 해당 함수가 종료되면 소멸되고 함수 밖에서 선언한 변수는 프로그램이 종료되어야 소멸되는 것을 잘 알아두기를 바랍니다.

위 코드에서 확인할 수 있는 사실은 전역변수 `num`을 함수 내에서 사용했다는 것입니다. 맞습니다. 전역변수는 어디서든 '참조'할 수 있습니다. 전역변수는 어디에서나 읽을 수 있지만, 함수 안에서 전역변수에 새로운 값을 '대입'하는 것은 불가능합니다.(예외적으로 `global` 키워드를 사용하면 가능함) 그리고 전역변수의 값은 함수 내에서 수정이 가능합니다. 위 코드에서 `globallist`에 `append(3)`으로 요소 3을 추가했습니다. 이는 **"전역변수 = 값"** 형식의 대입이 아니기 때문에 가능한 것입니다.

아래 지역변수와 전역변수의 접근 조건에 대해 정리했습니다.

기능	전역변수	지역변수
함수 안에서 읽기/수정	가능	가능
함수 안에서 =을 사용한 대입	불가능(`global` 사용 제외)	가능
함수 밖에서 읽기/수정	가능	불가능
함수 밖에서 =을 사용한 대입	가능	불가능

이렇게 전역변수와 지역변수의 차이점에 대해 알아가다보면 전역변수가 어디서든 읽을 수 있고 수정할 수 있기 때문에 전역변수를 많이 사용하는 것이 좋다고 생각할 수 있습니다. 하지만 전역 변수는 아래와 같이 단점을 가지고 있습니다.

전역변수 수정

그럼에도 불구하고 불가피하게 전역변수를 수정해야하는 일이 있다면 `global` 키워드를 사용하면 됩니다. `global` 키워드는 전역 변수 앞에 입력하면 어디서든 값을 대입할 수 있습니다. 아래 예시를 바로 실행해보세요.

```
num = 1 #전역변수 선언

def plusNum() :
  global num #전역변수를 함수 내에서 사용함을 선언

  num += 1

  print(num)

plusNum()

plusNum()

print(num)
```

```
프로세스가 시작되었습니다..
> 2
3
3
```