파이썬 기초: 함수와 클래스

#함수 #사용자 정의 함수 #전역변수 지역변수 #클래스

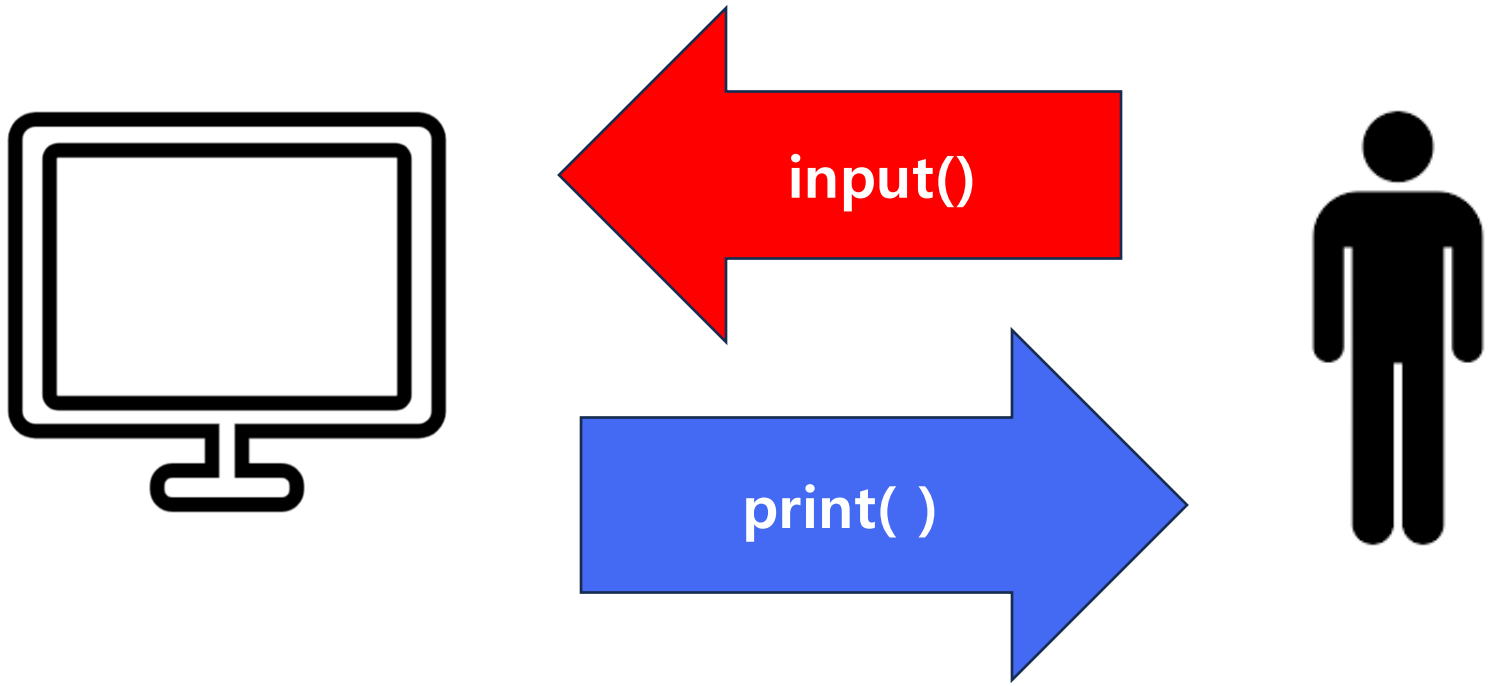
목차

함수(function)란?

함수란?

프로그래밍의 기본적인 틀

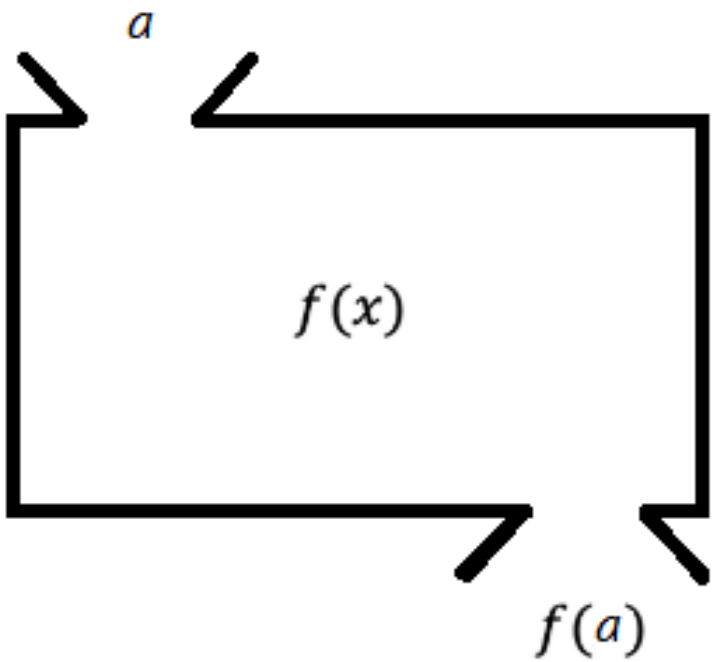
입력(input)을 받고, 그 입력을 바탕으로 내부적인 기능을 거쳐 Output을 내는 것



함수란?

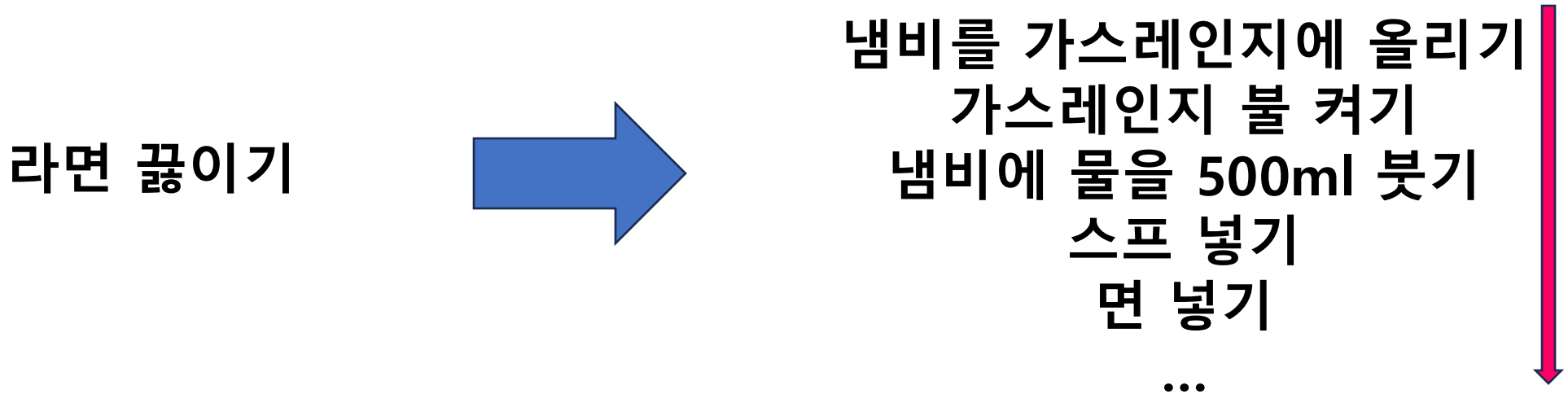
함수의 틀도 동일하다

함수는 입력을 받아서, 어떤 일을 수행하고, 결과를 돌려주는 것



그래서 함수란?

특정 하나의 기능을 수행하는 명령들의 모임

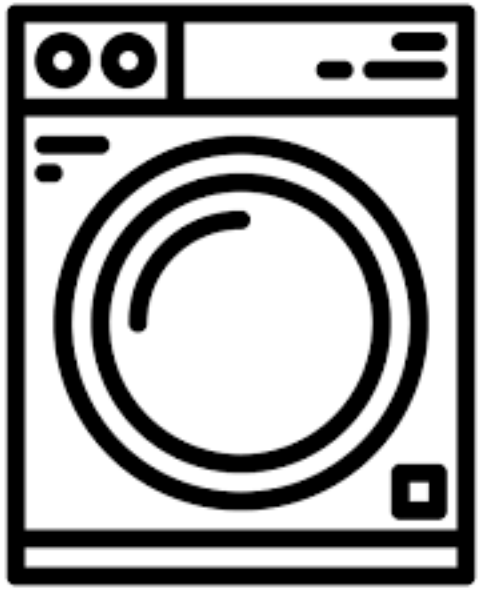


하나의 기능

기능을 수행하기 위한 명령들의 모임

함수(function)? 기능!

반복적으로 사용하는 여러 줄짜리 코드에 이름을 붙여서 저장
그리고 필요할 때마다 꺼내는 도구



“세탁하기”

- 물 채우기
- 세제 투입
- 회전
- 헹굼
- 탈수

❖ **반복되는 작업을 줄이고 재사용성 증가**
❖ **코드를 잘게 나눠서 정리**
❖ **유지보수와 수정 그리고 가독성을 위함**

세탁기 버튼 하나를 통해 위 작업을 수행하며, 주기적으로 사용함

함수의 종류

함수의 종류

내장 함수

Built-in

파이썬 개발자들이 이미 만들어 둔 함수들
기본적으로 탑재가 되어있으므로 사용만 하면 됨

사용자 정의

함수

User-defined

사용자가 직접 함수를 정의하는 방식

다양한 내장 함수(Built-in)

abs( )	all( )	any( )	ascii( )	bin( )
bool( )	breakpoint( )	bytearray( )	callable( )	chr( )
classmethod( )	compile( )	complex( )	delattr( )	dict( )
dir( )	divmod( )	enumerate( )	eval( )	exec( )
filter( )	float( )	format( )	frozenset( )	getattr( )
globals( )	hasattr( )	hash( )	help( )	id( )
input( )	int( )	isinstance( )	issubclass( )	iter( )
len( )	list( )	locals( )	max( )	map( )
memoryview( )	min( )	next( )	object( )	oct( )
ord( )	open( )	pow( )	print( )	repr( )
reversed( )	round( )	set( )	setattr( )	slice( )
sorted( )	staticmethod( )	str( )	sum( )	super( )
tuple( )	type( )	vars( )	zip( )	__import__( )

내장 함수(Built-in)

다양한 내장 함수(Built-in) 이전에 봤던 함수들?

len()

시퀀스 자료형의 길이
를 구하는 함수

input()

사용자로부터 입력을
받는 함수

type()

자료형의 타입을
구하는 함수

print()

값을 출력하는 함수

다양한 내장 함수(Built-in) - 집계 함수

max()

시퀀스 자료의 최댓값을
구하는 함수

```
print(max((1, 2, 3, 4, 5)) # 5
print(max([20, 40, 10, 50, 30]) # 50
```

min()

시퀀스 자료의 최솟값을
구하는 함수

```
print(min((1, 2, 3, 4, 5)) # 1
print(min([20, 40, 10, 50, 30]) # 10
```

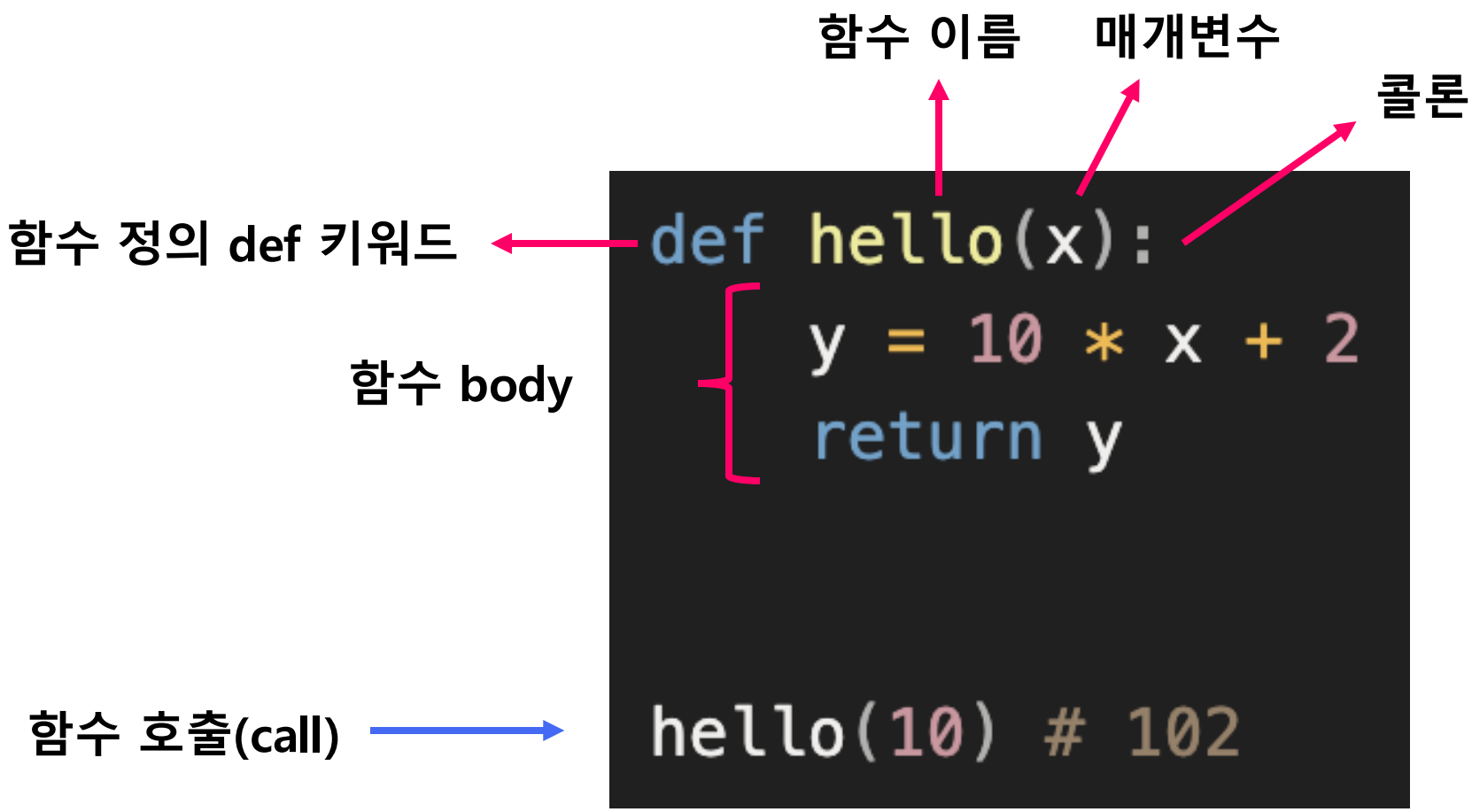
sum()

(숫자)시퀀스 자료의 합을
구하는 함수

```
print(sum((1, 2, 3, 4, 5)) # 15
print(sum([20, 40, 10, 50, 30]) # 150
```

사용자 정의 함수(User-Defined)

사용자가 여러 명령들을 묶어 직접 정의한 함수



사용자 정의 함수(User-Defined)

사용자 정의 함수 만들기 (1): 정의하기

def 키워드를 사용하여 '함수를 정의하겠다'라고 표현

함수이름은 함수를 구분 짓는 이름이며, 함수를 호출할 때 이름으로 호출

함수이름은 관례에 맞게끔 지어주면 된다.

- 함수는 어떠한 동작(Action)이므로 동사 형태
- snake_case를 활용

ex) generate_num_list, calculate_average

```
def hello(x):
    y = 10 * x + 2
    return y


hello(10) # 102
```

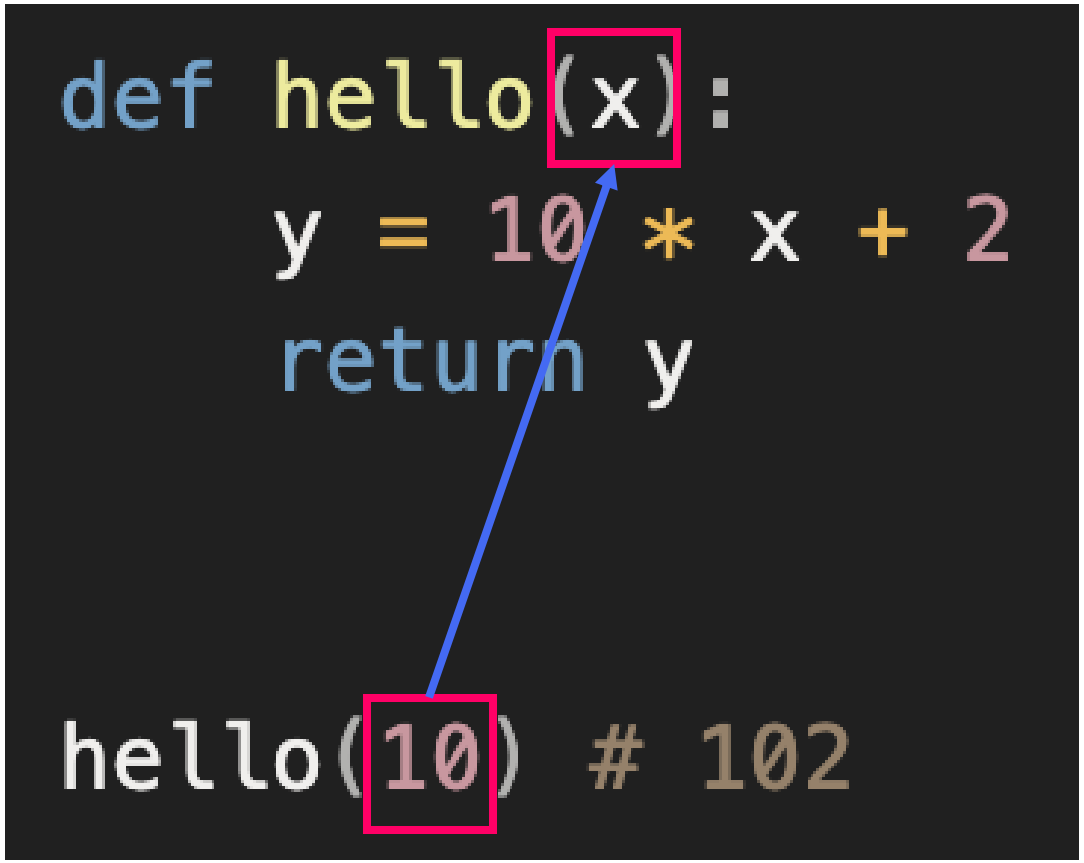
사용자 정의 함수(User-Defined)

사용자 정의 함수 만들기 (2): 함수의 입력; 매개변수

매개변수(parameter): 함수 호출 시, 함수의 외부에서 내부로 값을 전달할 때 받는 변수

인자(argument): 함수 호출 시, 함수에 전달하는 값

매개변수 이름과 개수는 자유롭게 정의 가능



사용자 정의 함수 만들기 (3): 함수의 명령 Body 정의

이 함수에서 동작할 실질적인 명령들(로직)을 작성한다.

마찬가지로 함수의 body부분은 들여쓰기를 통해 명령들을 종속 시킨다.

```
def hello(x):
    y = 10 * x + 2
    return y


hello(10) # 102
```

사용자 정의 함수 만들기 (4): 함수의 반환

최종적으로 명령들을 다 수행한 후,
결과 값을 return 키워드를 사용하여 반환.

반환된 값은 함수를 호출한 쪽에 전달된다.

```
def hello(x):
    y = 10 * x + 2
    return y


hello(10) # 102
```

사용자 정의 함수 만들기 (5): 함수 호출(function call)

함수를 정의만 한 것이지, 코드가 바로 실행되는 것이 아니다.

함수를 호출(call)을 해야 함수 내부 코드가 동작하며 결과를 받을 수 있다.

호출은 함수이름(인자) 형태로 호출.

함수의 인자(argument)에 함수로 전달할 값을 매개변수(parameter) 정의에 맞게 넣으면 된다.

```
def hello(x):
    y = 10 * x + 2
    return y


hello(10) # 102
```

사용자 정의 함수(User-Defined)

사용자 정의 함수 만들기: Example(사칙연산 계산기 함수 만들기)

```
def add(num1, num2):
    return num1 + num2

def sub(num1, num2):
    return num1 - num2

def mul(num1, num2):
    return num1 * num2

def div(num1, num2):
    return num1 / num2

print(add(3, 5))
print(sub(10, 7))
print(mul(4, 6))
print(div(15, 5))
```

```
8
3
24
3.0
```

사용자 정의 함수(User-Defined)

반환 여부에 따른 차이?

함수 내부에 일어난 일은 외부에서 모르기에 반환을 통해서 전달

```
def formula(a, b):
    c = (a**2) + (b**2)
    print("return: ", c)

formula(2, 4)

result = formula(2, 4)
print("result: ", result)
```

```
return:  20
return:  20
result:  None
```

```
def formula(a, b):
    c = (a**2) + (b**2)
    return c

formula(2, 4)

result = formula(2, 4)
print("result: ", result)
```

```
return:  20
```

사용자 정의 함수(User-Defined)

함수 매개변수의 확장: 디폴트 매개변수(Default Parameter)

매개변수에 디폴트 값을 명시하는 것으로, 매칭되는 인자없이 호출 시 디폴트 값을 사용

```
def def_para(country='Korea'):
    print('I am from', country)


print(def_para('India'))
print(def_para('Brazil'))
print(def_para())
```

```
I am from India
I am from Brazil
I am from Korea
```

```
def def_score(name, kor, eng, mat=50):
    print(name, kor, eng, mat)

def_score('John', 90, 80, 70)
def_score('Rosa', 85, 95)
```

디폴트 매개변수는 매개변수들 중에서 마지막에

```
John 90 80 70
Rosa 85 95 50
```

함수 매개변수의 확장: 가변 매개변수(Variable Argument)

여러 개의 값을 한 번에 받을 수 있는 매개변수 (*args)

- 매개변수 이름 앞에 *을 붙여주면 가변 매개변수가 된다.
- 전달 받은 매개변수는 Tuple 형태로 받은 값들을 묶는다.

```
kor, eng, mat, sci = 98, 77, 85, 90

def max_score(*args):
    return max(args)

print(max_score(kor, eng, mat))
print(max_score(eng, mat, sci))
```

```
98
90
```

사용자 정의 함수(User-Defined)

함수 매개변수의 확장: 매개변수 이름으로 인자 전달(Keyword Argument)

함수 호출 시, 각 인자에 매개변수의 이름을 명시해서 전달할 수 있다.
해당 매개변수에 전달하겠다고 구체적으로 명시하므로 더 정확한 전달 가능

```
def sum_score(kor, eng, math):
    return kor + eng + math

print(sum_score(kor=70, eng=80, math=90))
```

```
240
```

주의할 점!: 키워드 인자가 나오면, 그 뒤의 모든 인자는 반드시 키워드 형태여야함

```
print(sum_score(70, eng=80, 90))
```

사용자 정의 함수(User-Defined)

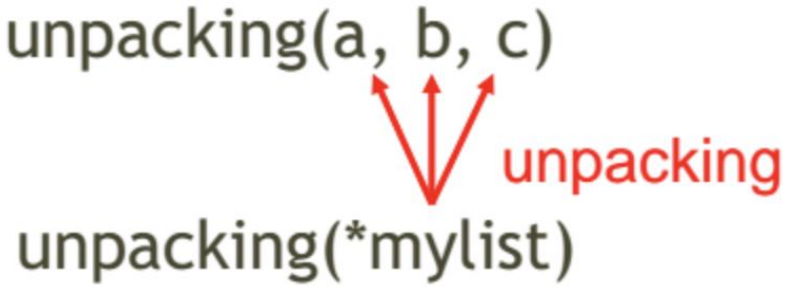
함수와 Unpacking/Packing 활용

함수의 인자로 리스트(또는 투플)에 *을 붙여주면 Unpacking을 수행.

```
def unpacking(a, b, c):
    print(a)
    print(b)
    print(c)

mylist = [10, 20, 30]
unpacking(*mylist)
```

```
10
20
30
```



함수와 Unpacking/Packing 활용

함수의 반환 부분에 여러 개의 값을 반환할 수 있다. (투플 형태로 반환)

```
def three(x):
    return x, x**2, x**3

a, b, c = three(3)
print(a, b, c)
```

```
3 9 27
```

투플 형태로 반환 후, Unpacking을 통해 각 변수에 할당

지역변수와 전역변수

지역변수와 전역변수

왜 result가 출력되지 않을까?

함수 내부에서 일어난 일은 함수 밖에 영향을 끼치지 않는다

```
def func1(n1, n2):
    result = n1 + n2
    return result

var1 = 15
var2 = 20
print(result)
answer = func1(var1, var2)
```

```
NameError: name 'result' is not defined
```

오류 발생!

전역 변수(Global variable)와 지역 변수(Local variable)

전역 변수

전역적인 범위에서 사용 가능한 변수

```
global_var = "Hi"

def my_func(a):
    print(a)
    print(global_var)

my_func(global_var)

print(global_var)
```

```
Hi
Hi
Hi
```

지역 변수

특정 구문(for문, 함수) 안에서 정의한 변수
변수를 정의한 범위에서만 사용 가능

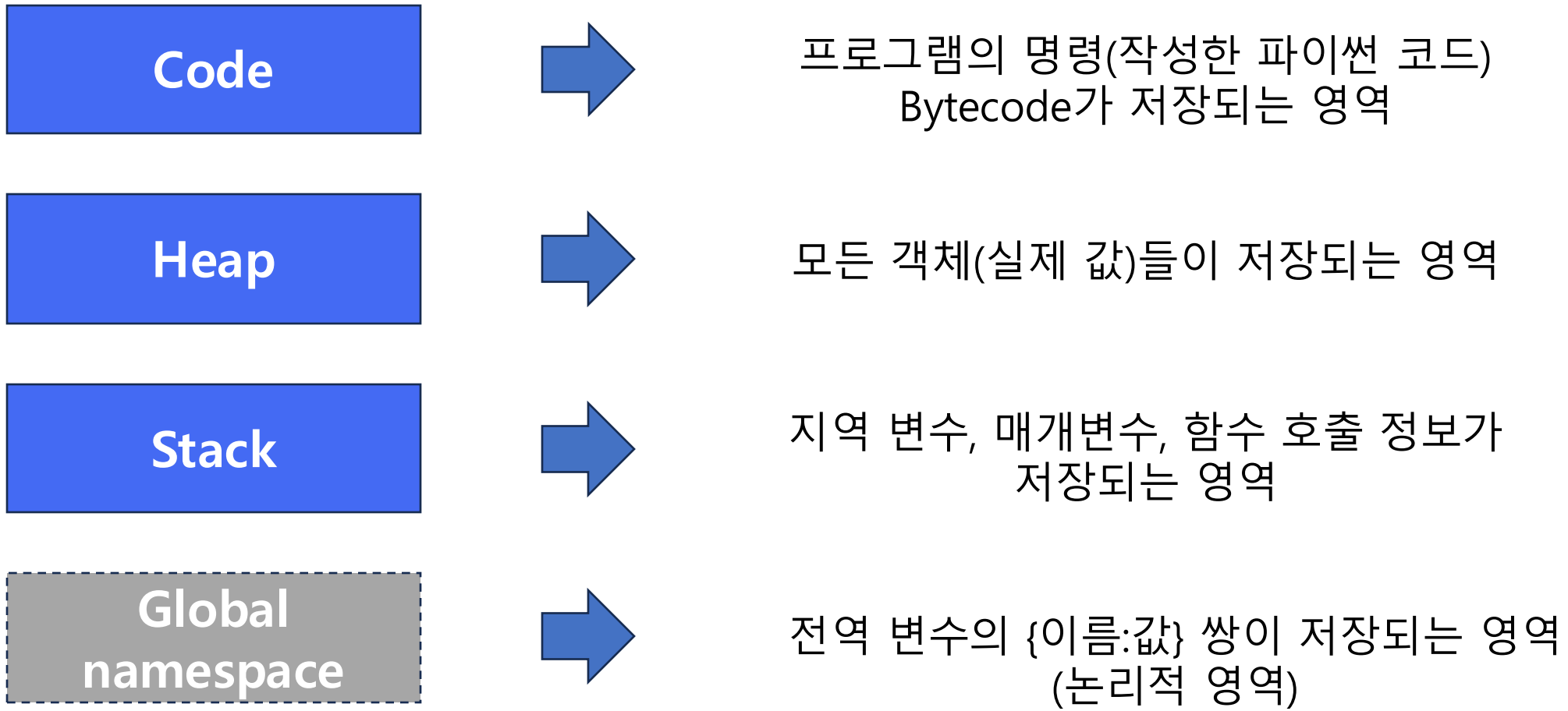
```
def my_func():
    local_var = "Hello"
    print(local_var)

my_func()

print(local_var)
```

```
Hello
NameError: name 'local_var' is not defined
```

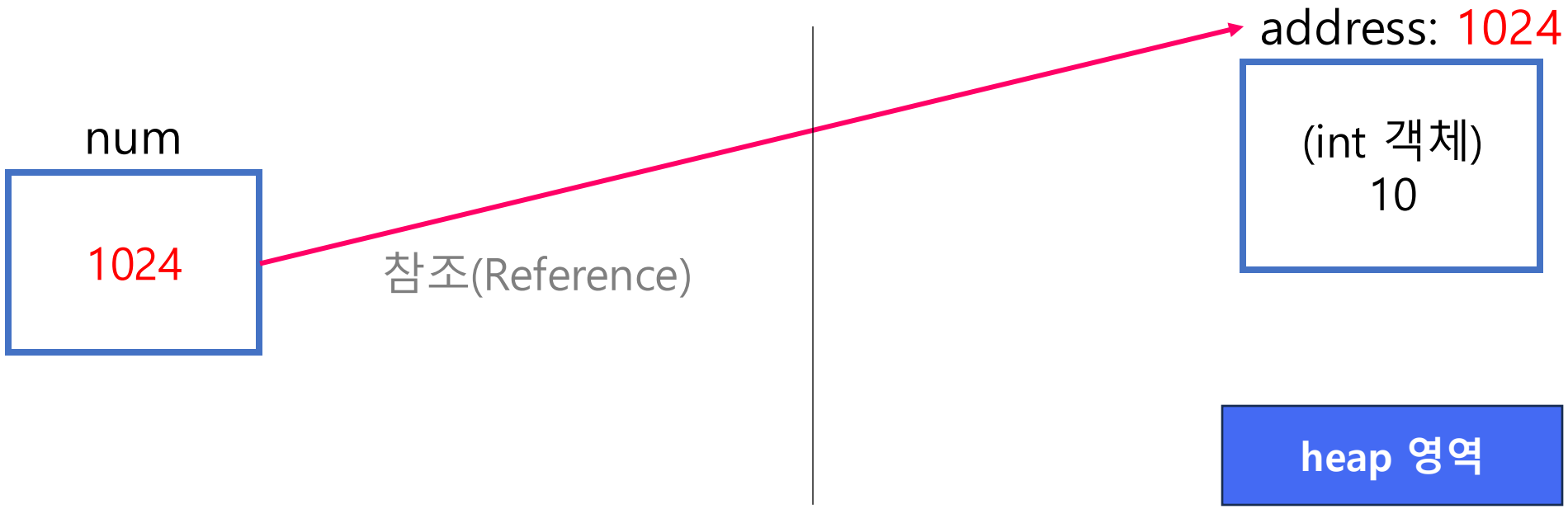
프로세스 메모리 구조 (Python)



파이썬 변수는 사실 값을 저장하는 것이 아니다?

변수에 실제 값이 저장되어 있는 것이 아니라, 값이 보관된 메모리 주소의 값을 저장하고 있다. 그리고 해당 객체를 참조(Reference)하고 있는 구조이다.

num = 10

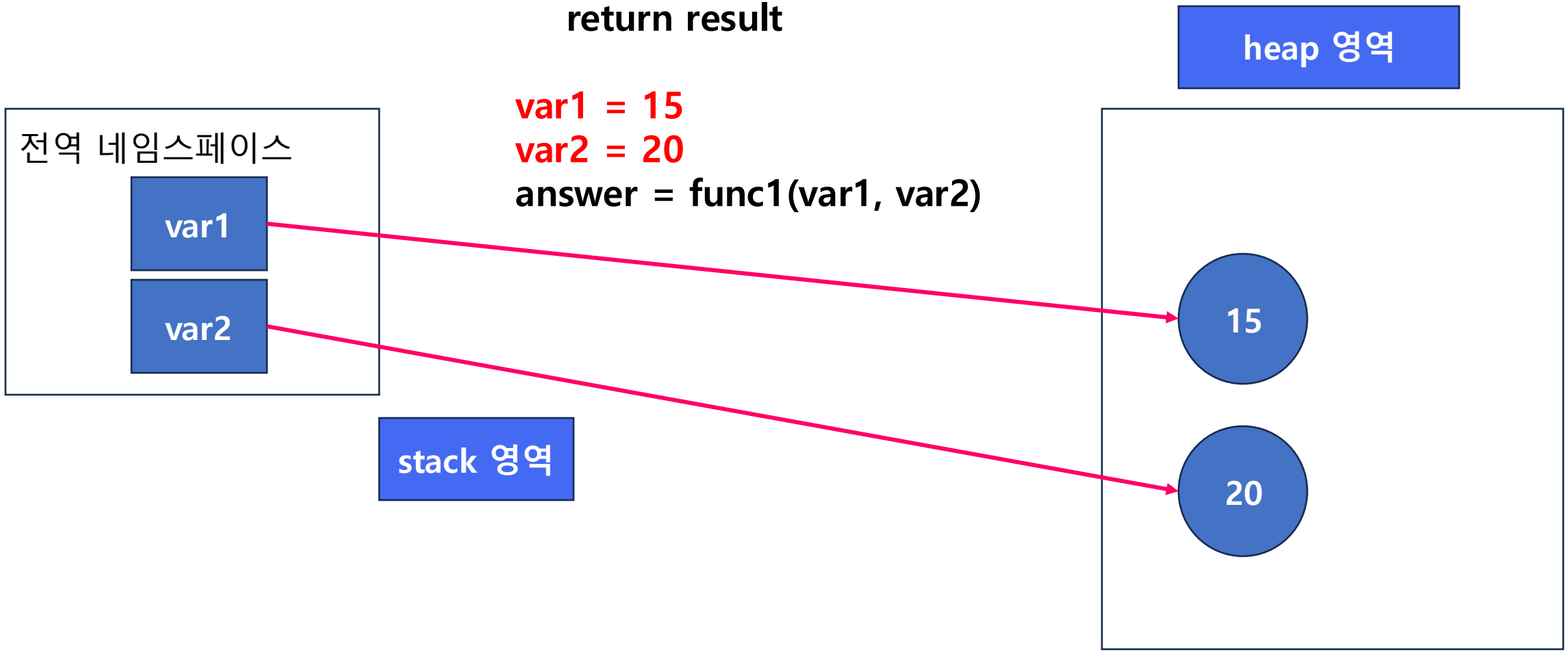


지역변수와 전역변수

과정을 살펴보자!

```
def func1(n1, n2):
    result = n1 + n2
    return result
```

```
var1 = 15
var2 = 20
answer = func1(var1, var2)
```

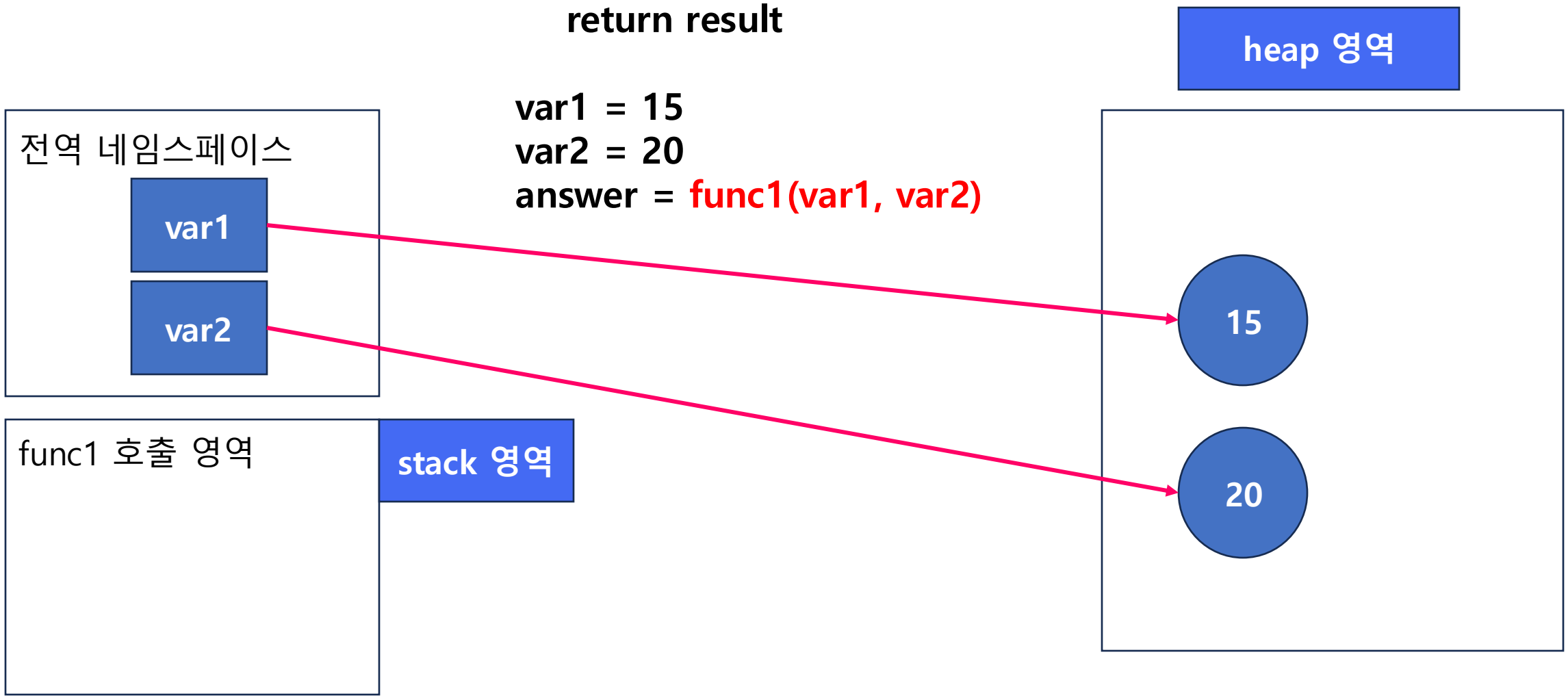


지역변수와 전역변수

과정을 살펴보자!

```
def func1(n1, n2):
    result = n1 + n2
    return result

var1 = 15
var2 = 20
answer = func1(var1, var2)
```

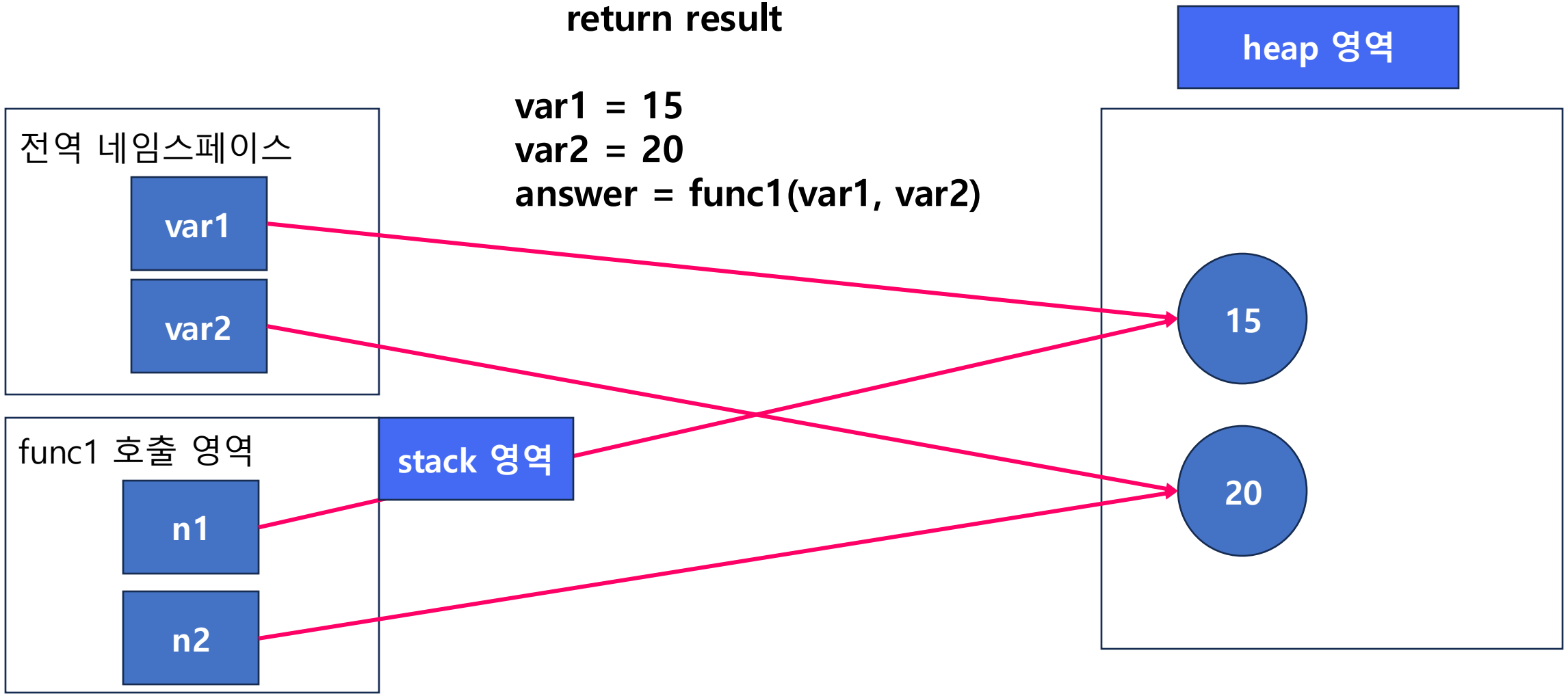


지역변수와 전역변수

과정을 살펴보자!

```
def func1(n1, n2):
    result = n1 + n2
    return result

var1 = 15
var2 = 20
answer = func1(var1, var2)
```

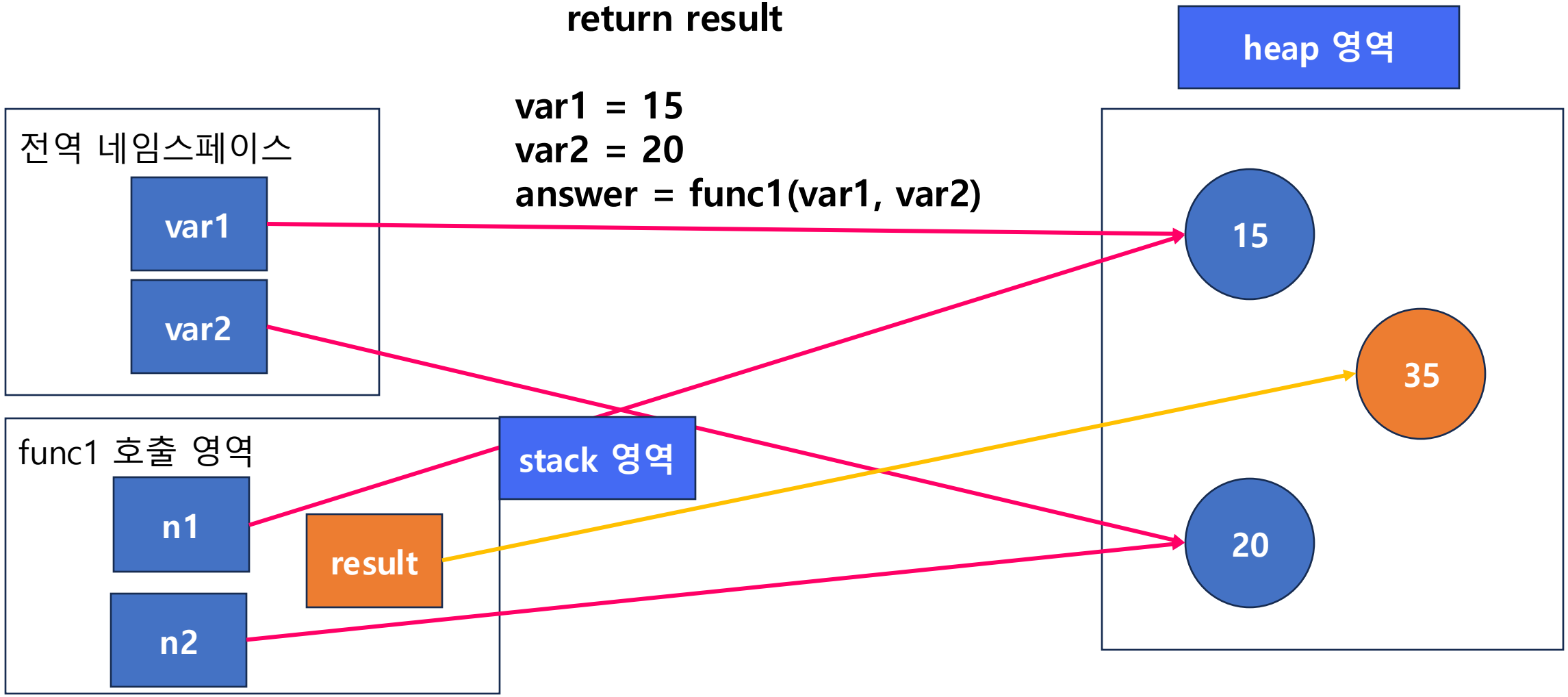


지역변수와 전역변수

과정을 살펴보자!

```
def func1(n1, n2):
    result = n1 + n2
    return result

var1 = 15
var2 = 20
answer = func1(var1, var2)
```

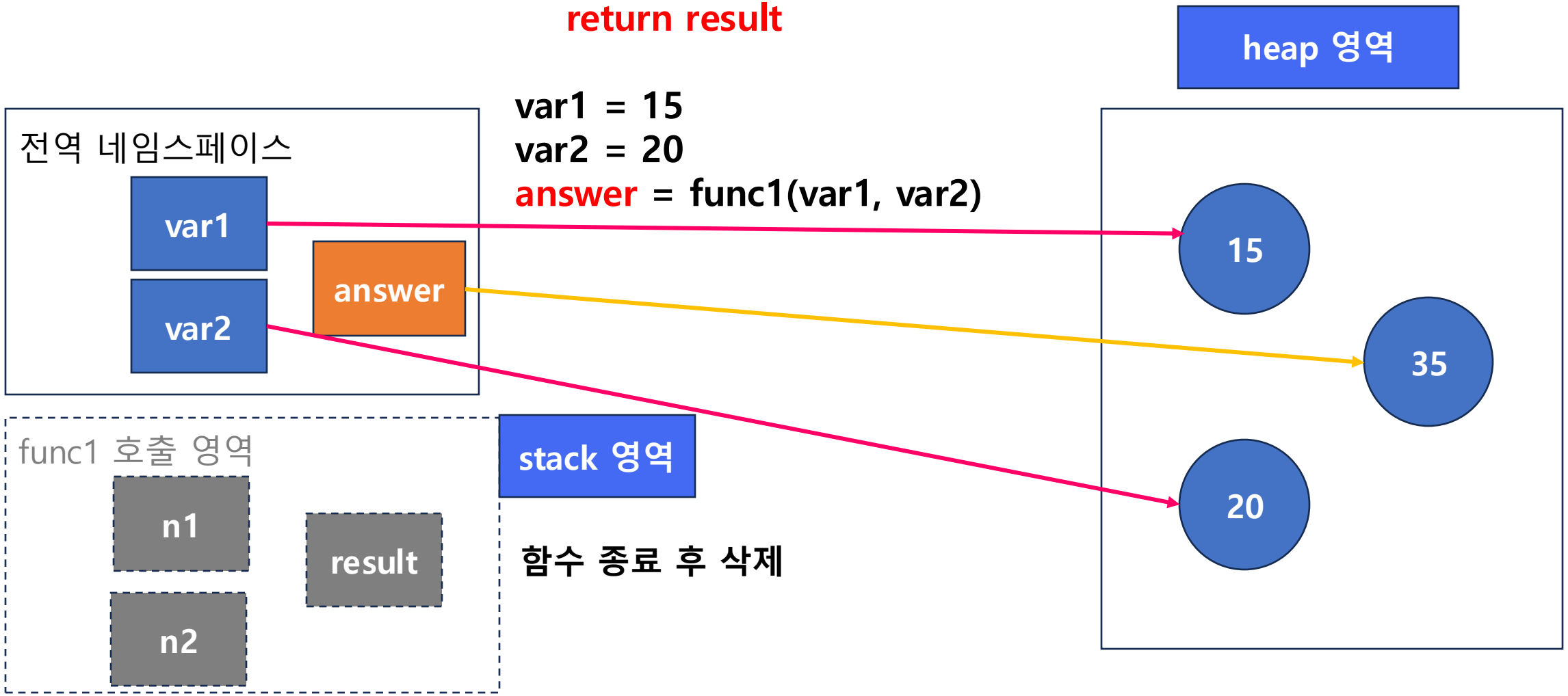


지역변수와 전역변수

과정을 살펴보자!

```
def func1(n1, n2):
    result = n1 + n2
    return result
```

```
var1 = 15
var2 = 20
answer = func1(var1, var2)
```

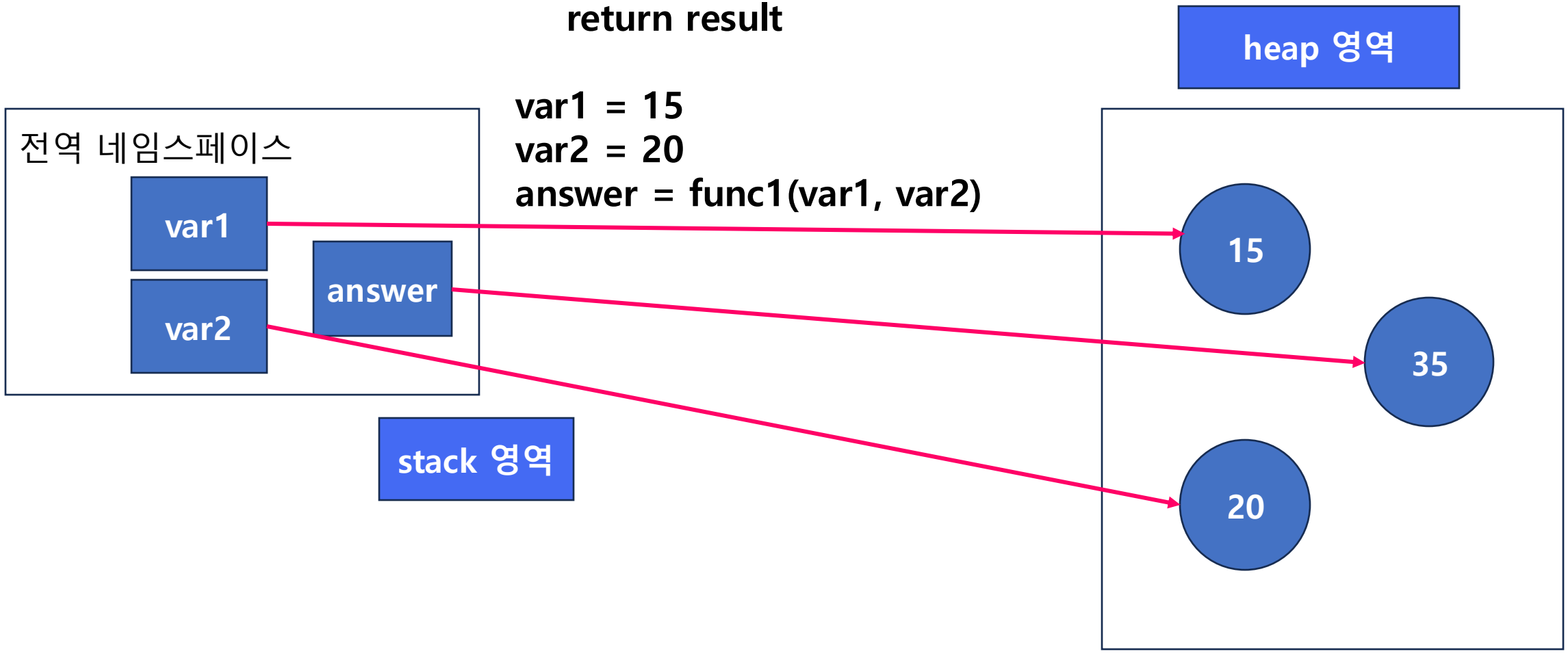


지역변수와 전역변수

과정을 살펴보자!

```
def func1(n1, n2):
    result = n1 + n2
    return result

var1 = 15
var2 = 20
answer = func1(var1, var2)
```



전역 변수(Global variable)

전역 변수는 Global Namespace 영역에 {이름: 객체} 딕셔너리 형태로 저장되어 있음
Stack 영역에 Global Namespace에 대한 포인터(참조)가 있어서 함수 내부에서 전역 변수 참조

```
global_var = 10

def my_func():
    print(global_var)
    global_var += 10
    print(global_var)

my_func()
```

단, 읽기만 가능

수정하려면 변수 앞에
global 키워드를 붙여야함

```
UnboundLocalError: cannot access local variable 'global_var' where it is not asso
ciated with a value
```

```
global_var = 10

def my_func():
    global global_var
    print(global_var)
    global_var += 10
    print(global_var)

my_func()
```

```
10
20
```

클래스(Class)와 객체(Object)

객체지향 프로그래밍(Object-Oriented Programming)

객체를 중심으로 프로그래밍하는 패러다임

객체지향 프로그래밍에서는 모든 것을 객체(Object)로 간주한다.

현실 세계에 실존하는 사람, 사물, 사건 등의 모든 것을 프로그래밍 상에서 정의/표현하기 위함이다.

반례

Q. 이전까지 배운 내용으로만

사람 자체를 파이썬에서 정의하려면 어떻게?
한 반의 40명의 학생들을 모두 정의하려면 어떻게?

클래스(Class)

클래스(Class)란?

객체들의 유사한 특징을 지닌 속성과 동작을 묶어놓은 집합

Example: “사람”

Q. 사람의 공통 속성?

Q. 사람의 공통 동작?

실제 사람들은 여러 속성과 동작을 갖지만, 서로 다른 속성을 가짐

하지만, 그들 모두 “사람”이라는 클래스에 속함

클래스(Class)란?

클래스는 속성(Attribute), 동작(Behavior)을 갖고 있는 일종의 **자료형**이다.

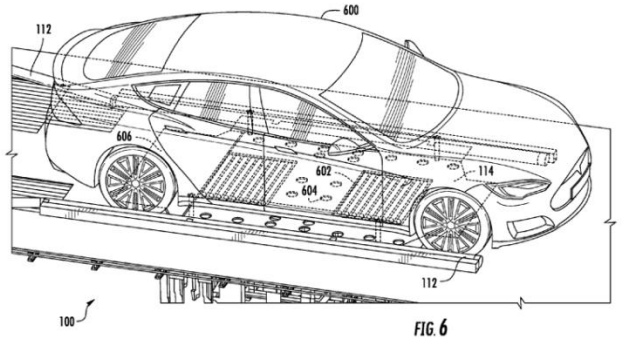


FIG. 6

클래스는 **설계도**와 같은 역할

이 **설계도**를 바탕으로 찍어낸 것이 구체화된 **객체**

클래스 ↔ 객체

설계도 — 설계도를 바탕으로 만들어진 실제 상품

**프로그래밍 상에서 실제 메모리 상에서 생성된 객체를 인스턴스(instance)라고 함*

클래스(Class)

클래스 예시 1: 붕어빵 클래스

붕어빵

속성:
- 속 재료
- 속의 양

동작:
- 속 채우기
- 붕어빵 꺼내기
- 속 양 조절하기

클래스(= 설계도)

객체화

속 재료 : **팥**
속의 양 : 보통

속 재료 : **슈크림**
속의 양 : 많이

속 재료 : **고구마**
속의 양 : 적게

설계도를 바탕으로 찍혀서 나온 실제 제품들
(= 객체 또는 인스턴스)

클래스(Class)

클래스 예시 2: 자동차 클래스

자동차

속성:
- 색상
- 연비
- 모델명

동작:
- 시동 걸기
- 가속하기
- 멈추기
- 기어 변경하기

클래스(= 설계도)

객체화

색상 : 초록색
연비 : 10.1
모델명 : A

색상 : 노란색
연비 : 12.2
모델명 : B

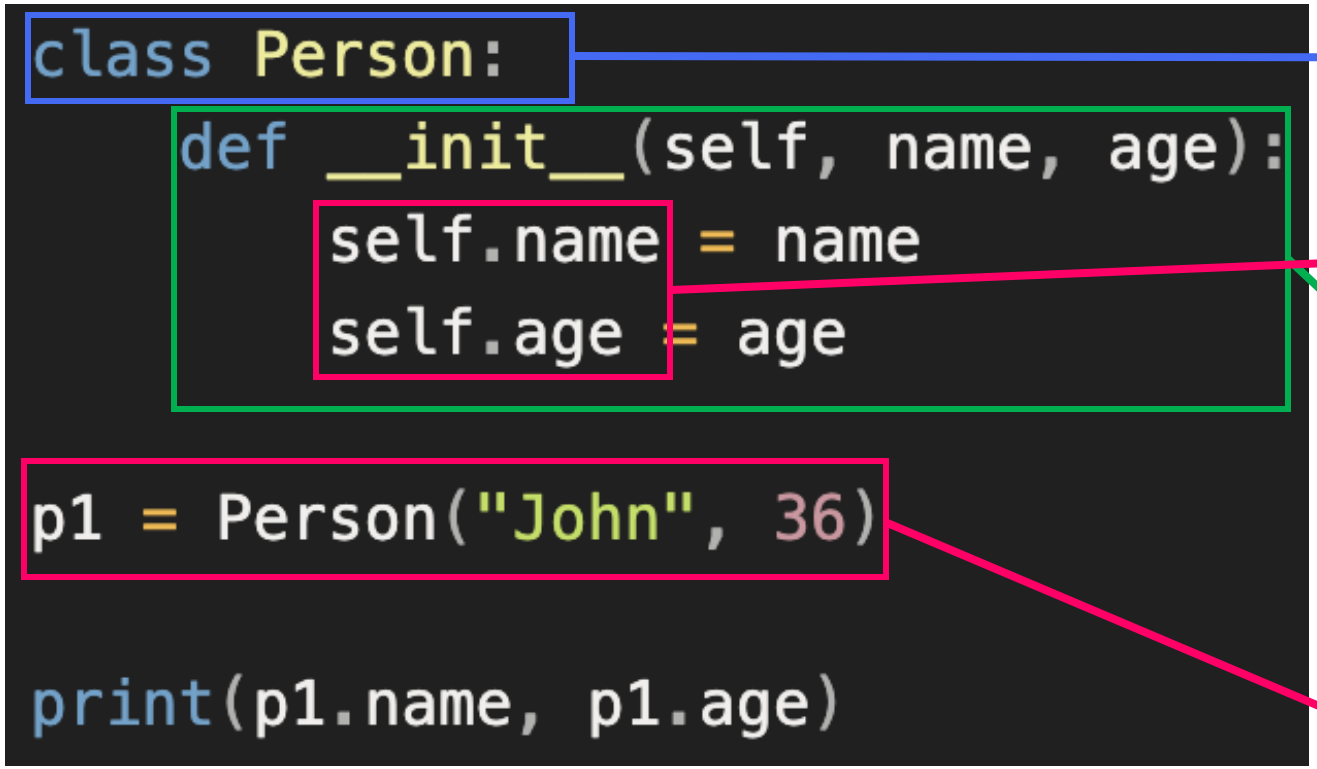
색상 : 보라색
연비 : 14.4
모델명 : C

색상 : 빨간색
연비 : 16.6
모델명 : D

설계도를 바탕으로 찍혀서 나온 실제 상품들
(= 객체 또는 인스턴스)

Python에서의 클래스(Class)

클래스(Class) 정의 및 객체 생성하기



클래스(Class) 정의하기: 클래스 이름

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name, p1.age)
```

클래스 이름
→ 설계도 이름

PascalCase로 이름을 짓는 것이 관례적

ex) Car, SmartPhone, Dog, Student

클래스(Class) 정의하기: (1) 생성자

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name, p1.age)
```

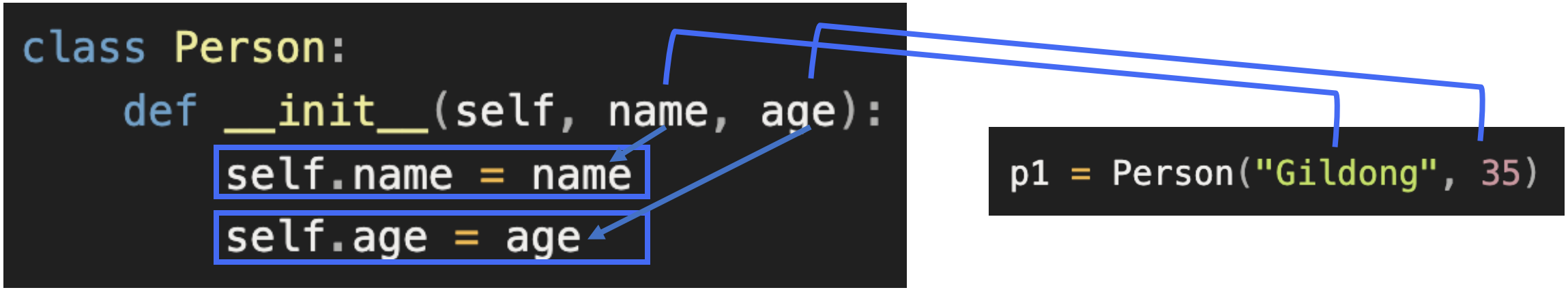
객체(인스턴스)가 생성될 때, 한 번 호출되는 함수

이름은 반드시 “__init__” 형태

호출할 때는 클래스 이름으로 호출하며, 생성자의 매개변수에 인자를 넣어준다.

생성자(Constructor)의 역할

인스턴스 생성 시, 단 한 번 호출되므로, 클래스의 속성 정의하고, 초기화하는 작업에 주로 사용



name과 age 속성 변수를 생성 후, 생성자의 매개변수로 받은 값으로 초기화

클래스(Class) 정의하기: (2) 인스턴스 변수

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name, p1.age)
```

```
John 36
```

클래스의 속성으로 정의된 변수를 의미하며 객체마다 서로 다르다

객체 생성 후, 인스턴스 멤버변수에 접근하는 방법은 객체.멤버변수 형태

p1 사람 객체의 이름은 John이며, 나이는 36이다

self 키워드

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name, p1.age)
```

객체 자기 자신을 의미하는 키워드

클래스는 설계도이기에 실제 객체를 특정하는 것이 안되기에

실제 객체가 생성되었을 때, self가 해당 객체 자기자신이 되는 것

이름이 John이며 나이가 36인 사람

그 사람 자기자신의 name 속성, age 속성, 생성자, 그 외의 메서드

클래스(Class) 정의하기: (3) 인스턴스 메서드

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def eat(self, food):
        print(self.name, "이", food, "를 먹는다.")

p1 = Person("John", 36)

print(p1.name, p1.age)
p1.eat("Banana")
```

클래스 내부에 정의된 함수를 메서드라 함.

- 특정 주체가 해당 동작을 수행 (함수 호출의 주체가 있는)

마찬가지로 객체 생성 후, **객체.메서드()** 형태로 호출

!모든 인스턴스 메서드에는 self 매개변수가 포함되어야 한다.

```
John 이 Banana 를 먹는다.
```

클래스(Class)

Example

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def walk(self):
        print(f"{self.name}이 걷습니다.")

    def eat(self, food):
        print(f"{self.name}이 {food}를 먹습니다.")

gildong = Person("Gildong", 22)
john = Person("John", 40)

gildong.walk()
john.eat("Banana")
```

사람 클래스 정의(선언부)
- name과 age 멤버 변수(속성)
- walk와 eat 멤버 메서드(동작)

사람 클래스를 바탕으로 객체 2개 생성
(= 두 명의 사람 생성)

객체 활용(멤버 변수, 메서드 호출 등)

Exercise

Car 클래스를 정의하고, 생성자로부터 두 가지의 속성을 정의하고 두 가지의 메서드를 정의. 그리고 두 객체를 생성 후, 오른쪽 아래와 같이 출력하도록 코드를 작성하시오.

- 클래스 이름: Car
- 속성: name, speed
- 메서드: get_name, get_speed
  - get_name: name 속성 값을 반환
  - get_speed: speed 속성 값을 반환

```
Audi's speed is 10
BMW's speed is 30
```

Exercise: Solution

Car 클래스를 정의하고, 생성자로부터 두 가지의 속성을 정의하고 두 가지의 메서드를 정의. 그리고 두 객체를 생성 후, 오른쪽 아래와 같이 출력하도록 코드를 작성하시오.

```
class Car:
    def __init__(self, name, speed):
        self.name = name
        self.speed = speed
    def get_name(self):
        return self.name
    def get_speed(self):
        return self.speed

car1 = Car("Audi", 10)
car2 = Car("BMW", 30)

print(f"{car1.get_name()}'s speed is {car1.get_speed()}")
print(f"{car2.get_name()}'s speed is {car2.get_speed()}")
```

감사합니다!