함수

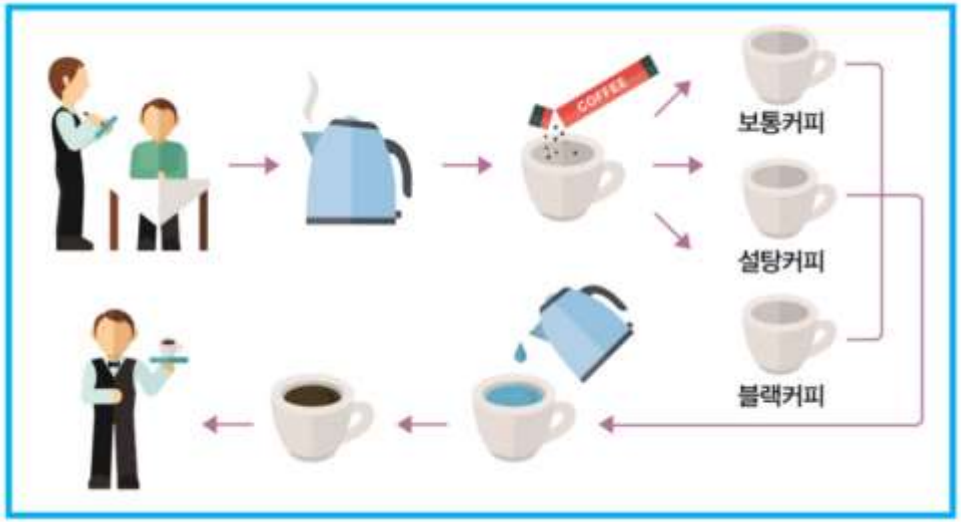
김지성 강사

함수(function)

✓ 함수는 독립적으로 설계된 프로그램 코드의 집합이다.

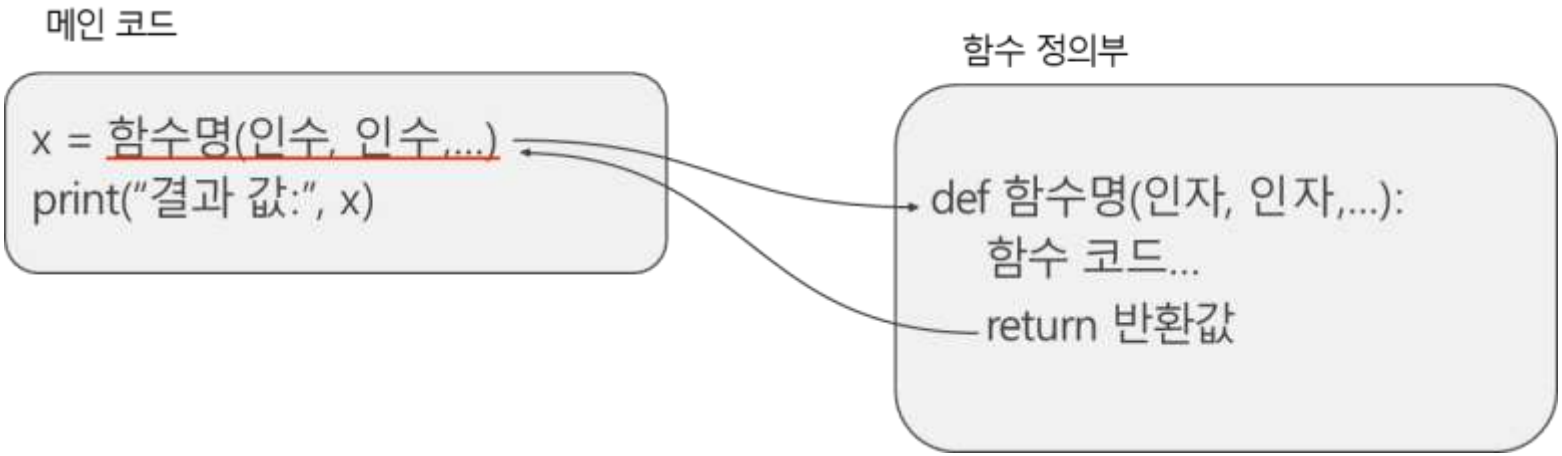
✓ 함수를 사용하면 반복적인 코드의 양을 줄이고 유지보수성을 높여준다.

✓ 사람이 직접 물어보고 커피를 타는 것 보다는 자판기의 인력버튼을 통해서 커피를 마시는게 이득!



함수(function)

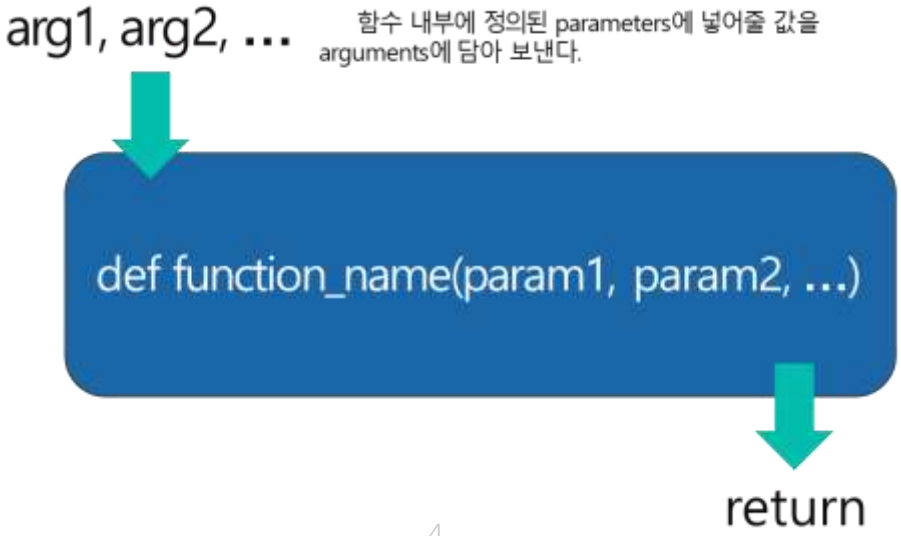
- ✓ 함수가 호출되면 함수 정의부로 가서 함수 호출부의 인수 값을 함수 정의부 인자 값에 순서대로 대입 후 함수 정의부 내부 코드를 순차적으로 실행.
- ✓ 해당 함수에 return 값이 있는 경우, 함수 호출부를 return 값으로 반환한다.



함수(function)

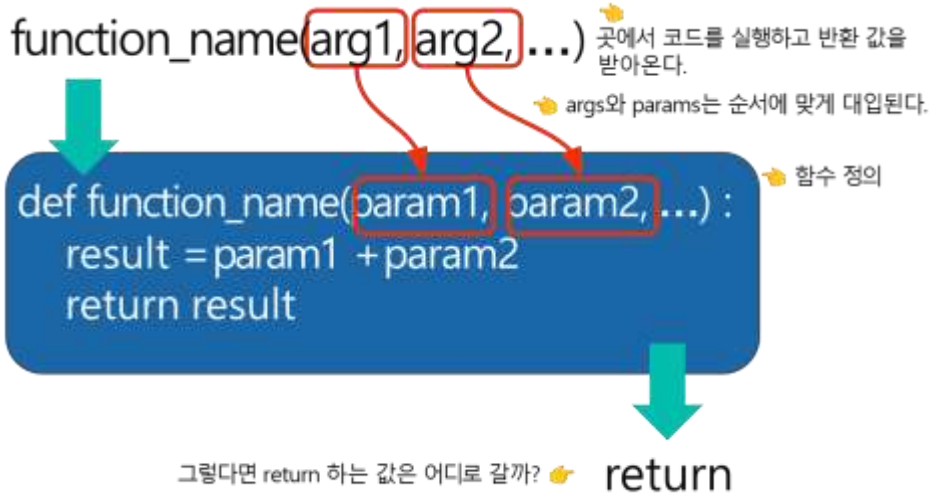
✓ 함수는 다음과 같이 정의한다.

```
def 함수이름(파라미터1,파라미터2…):
    코드 작성
    return 반환값
```



함수(function)

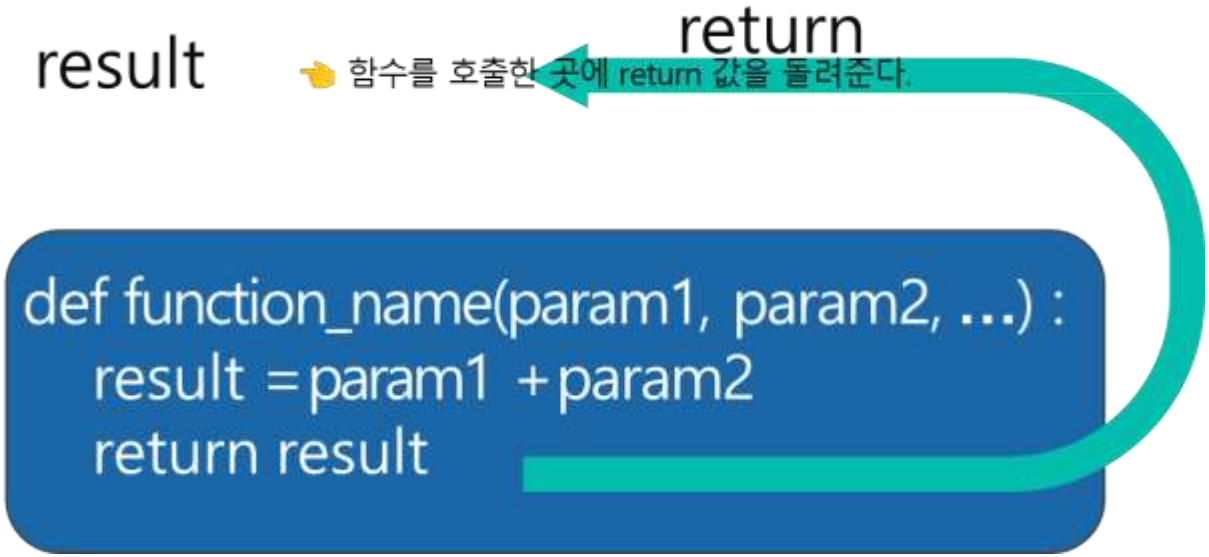
- ✓ 함수의 param1, param2에 해당되는 부분은 함수 외부에서 받아오는 변수, 값에 해당된다.
- ✓ 이렇게 받아온 파라미터(parameter)들은 함수 내부에서 따로 선언하지 않아도 자유롭게 사용이 가능하고 받아왔던 변수의 값 그 자체가 들어가있다.



함수(function)

✓ 함수의 반환값은 return 부분을 의미한다.

✓ 함수는 변수, 함수등을 return할 수 있는데 이를 main 부분에서 받아와서 활용할 수 있다.



함수(function)

✓ 함수의 정의

```
def hello(): # 함수 정의부만 존재
    print("hello")
hello()
```

```
def PrintVar(x,y): # 함수의 파라미터 존재
    print(f"x={x},y={y}")
x=1
y=2
PrintVar(x,y)
```

```
def add(x,y): # 함수의 파라미터, 반환값이 존재 (반환값은 여러 개도 가능하다.)
    return x+y
x=1
y=2
add_x_y = add(x,y)
print(add_x_y)
```

함수(function) - 위치-키워드 인자

✓ Parameters, Arguments는 혼용되어 이야기하지만 2개는 엄밀히 다른 개념입니다.

- ○ Parameters : 함수 정의부에 나오는 이름
- ○ Arguments : 함수를 호출할 때 실제로 전달하는 값

따라서 Parameters는 어떤 종류의 Arguments를 받을지 정의합니다.

함수 정의 👉

함수 정의절에 위치한 Parameters

```
def greeting(name, age):
    print(f"{name} 씨 안녕하세요. 약 {age * 365.25} 일 되었습니다.")
```

함수 호출 👉

함수 호출부에 위치한 Arguments

```
greeting("파이썬", 32)
```

```
파이썬 씨 안녕하세요. 약 11688.0 일 되었습니다.
```

함수(function) - 위치-키워드 인자

- ✓ 위치 인수(positional arguments) 또는 키워드 인수(keyword arguments)를 받을 수 있는 인자(parameter)를 위치-키워드 인자(positional-or-keyword parameters)라고 한다.
- ✓ 아래 예제에서는 위치-키워드 인자가 위치 인수를 전달 받아 처리하고 있음.

함수 정의 👉

```
def greeting(name, age):
    print(f"{name} 씨 안녕하세요. 약 {age * 365.25} 일 되었습니다.")
```

함수 호출 👉

```
greeting("파이썬", 32)
```

```
파이썬 씨 안녕하세요. 약 11688.0 일 되었습니다.
```

함수(function) - 위치-키워드 인자

✓ 아래 예제에서 함수 정의절에 보면 위치-키워드 인자(positional-or-keyword parameters)가 기본 값을 갖지 않는다. 따라서 인수없이 함수를 호출하면 TypeError가 발생한다.

함수 정의 👉

```
def greeting(name, age):
    print(f"{name} 씨 안녕하세요. 약 {age * 365.25} 일 되었습니다.")
```

기본 값(default)가 설정되어 있지 않네요!

함수 호출 👉

```
greeting()
```

인수(arguments) 없이 호출하고 있습니다!

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-86-3ffcd7d44b66> in <module>
----> 1 greeting()

TypeError: greeting() missing 2 required positional arguments: 'name' and 'age'
```

기본 값이 없는 위치-키워드 인자(positional-or-keyword parameter)는 인수(arguments) 전달이 필수입니다.

함수(function) - 위치-키워드 인자

✓ 아래 예제에서는 키워드 인수만으로 호출하고 있다. 키워드 인수 간에 순서는 상관 없다.

함수 정의 👉

```
def greeting(name, age):
    print(f"{name} 씨 안녕하세요. 약 {age * 365.25} 일 되었습니다.")
```

함수 호출 👉

```
greeting(age=32, name="파이썬")
```

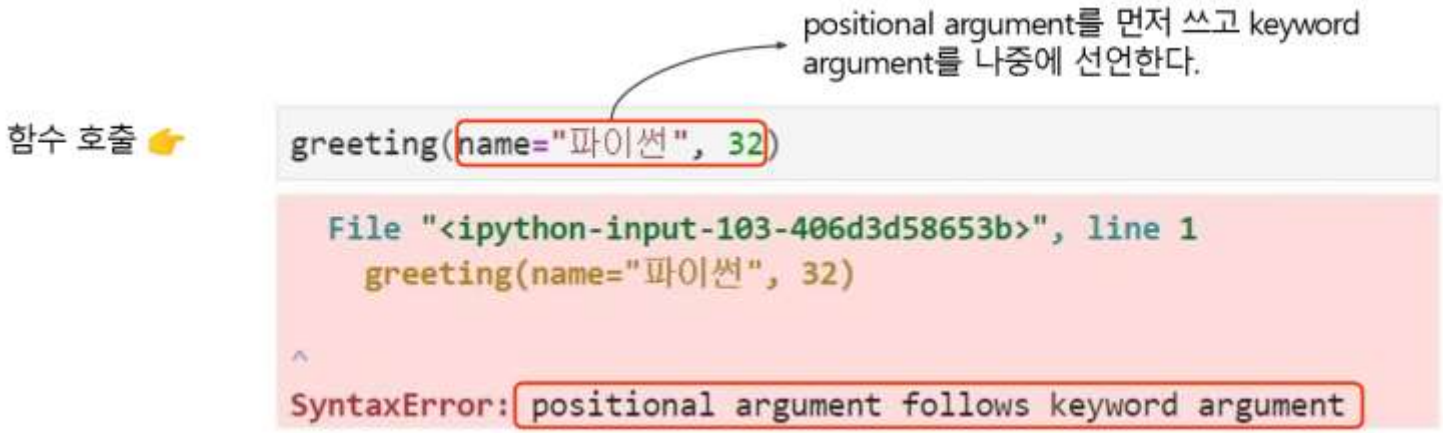
```
파이썬 씨 안녕하세요. 약 11688.0 일 되었습니다.
```

키워드 인수(keyword arguments)를 활용해서 순서를 상관않고 호출하고 있습니다!

함수(function) - 위치-키워드 인자

✓ 함수를 호출할 때 소괄호() 안에 위치 인수(positional arguments)를 먼저 작성하고 키워드 인수(keyword arguments)를 나중에 작성한다.

✓ 위치 인수는 순서에 영향을 받지만 키워드 인수는 순서에 상관없이 작성 가능하다.



함수(function) - 위치-키워드 인자

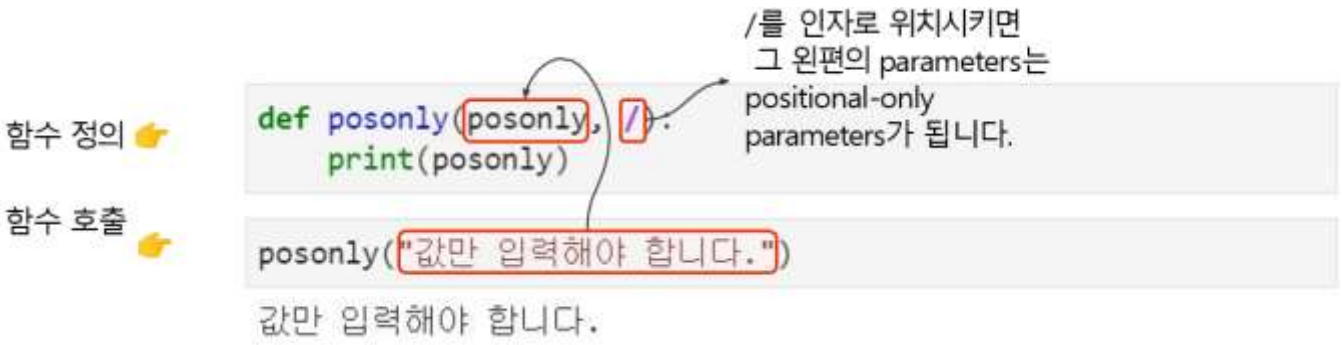
✓ 위치-키워드 인자 정리

함수(function) - 위치-키워드 인자

✓ 기본 값을 갖는 default parameter는 기본 기본 값을 갖지않는 non-default parameter 보다 뒤에 작성해야 한다.

함수(function) - 위치 전용 인자

- ✓ 위치 전용 인자(positional only parameter)는 오직 위치 인수(positional arguments)만 값의 전달이 가능하다.
- ✓ 위치 전용 인자를 선언하는 방법은 슬래시(/)를 인자 값으로 넣고 좌측 부분에 위치 전용 인자를 선언하면 된다.



함수(function) - 위치 전용 인자

✓ 위치 전용 인자(positional only parameter)에 키워드 인수(keyword arguments)를 전달하면 TypeError 오류가 발생된다.

positional-only parameters

함수 정의 👉

```
def posonly(posonly, /):
    print(posonly)
```

함수 호출 👉

```
posonly(posonly="키워드 인수로 전달하면 오류 발생!")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-94-52380c6bc56f> in <module>
----> 1 posonly(posonly="키워드 인수로 전달하면 오류 발생!")

TypeError: posonly() got some positional-only arguments passed as keyword arguments: 'posonly'
```

함수(function) - 키워드 전용 인자

✓ 키워드 전용 인자(keyword only parameter)도 기본값을 가질 수 있다.

✓ 이때 기본값을 갖는 인자에 상응하는 인수(arguments)는 생략할 수 있다.

함수 정의 👉

```
def keyonly(*, keyonly="default"):
    print(keyonly)
```

keyword-only parameters 도 기본 값 설정 가능하다.

함수 호출 👉

```
keyonly()
```

```
default
```

default 값이 있는 parameter에 대해서는 argument를 생략해도 정상 동작한다.

함수(function) - 키워드 전용 인자

- ✓ 키워드 전용 인자(keyword only parameter)는 오직 키워드 인수(keyword arguments)만 값의 전달이 가능하다.
- ✓ 키워드 전용 인자를 선언하는 방법은 애스터리스크(*)를 인자 값으로 넣고 우측 부분에 키워드 전용 인자를 선언하면 된다.

함수(function) - 키워드 전용 인자

✓ 키워드 전용 인자(keyword only parameter)에 위치 인수(positional arguments)를 전달하면 TypeError 오류가 발생됩니다.

keyword-only parameters

함수 정의 👉

```
def keyonly(*, keyonly):
    print(keyonly)
```

함수 호출 👉

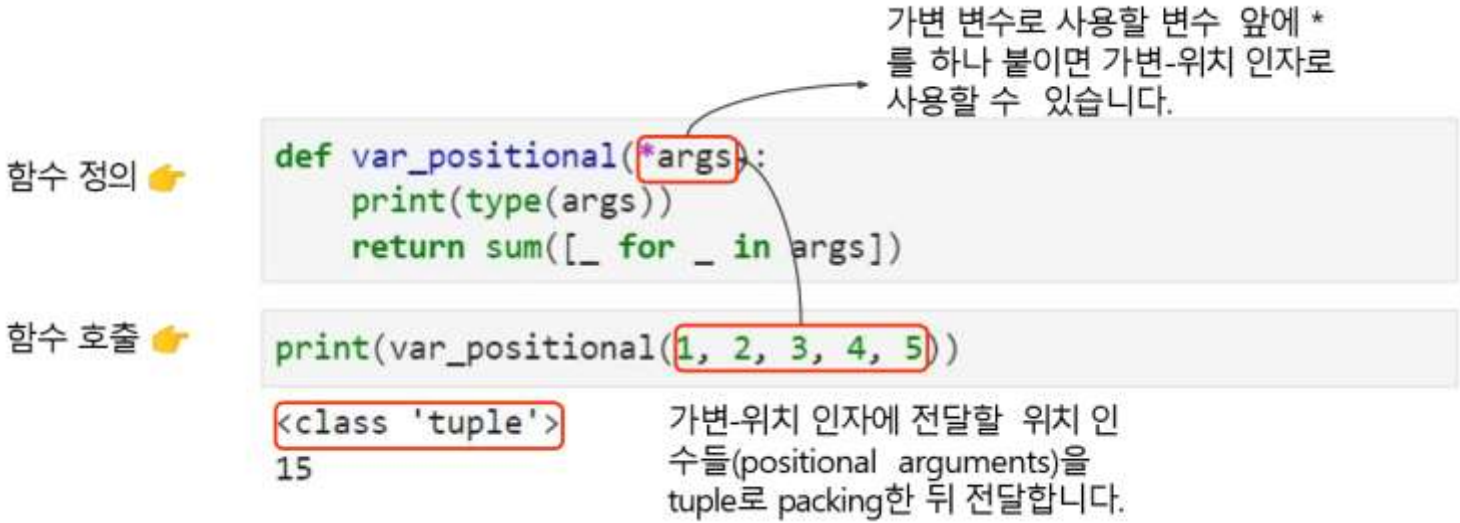
```
keyonly("키워드로만 입력해야 합니다.")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-117-25a227aedaa7> in <module>
----> 1 keyonly("키워드로만 입력해야 합니다.")

TypeError: keyonly() takes 0 positional arguments but 1 was given
```

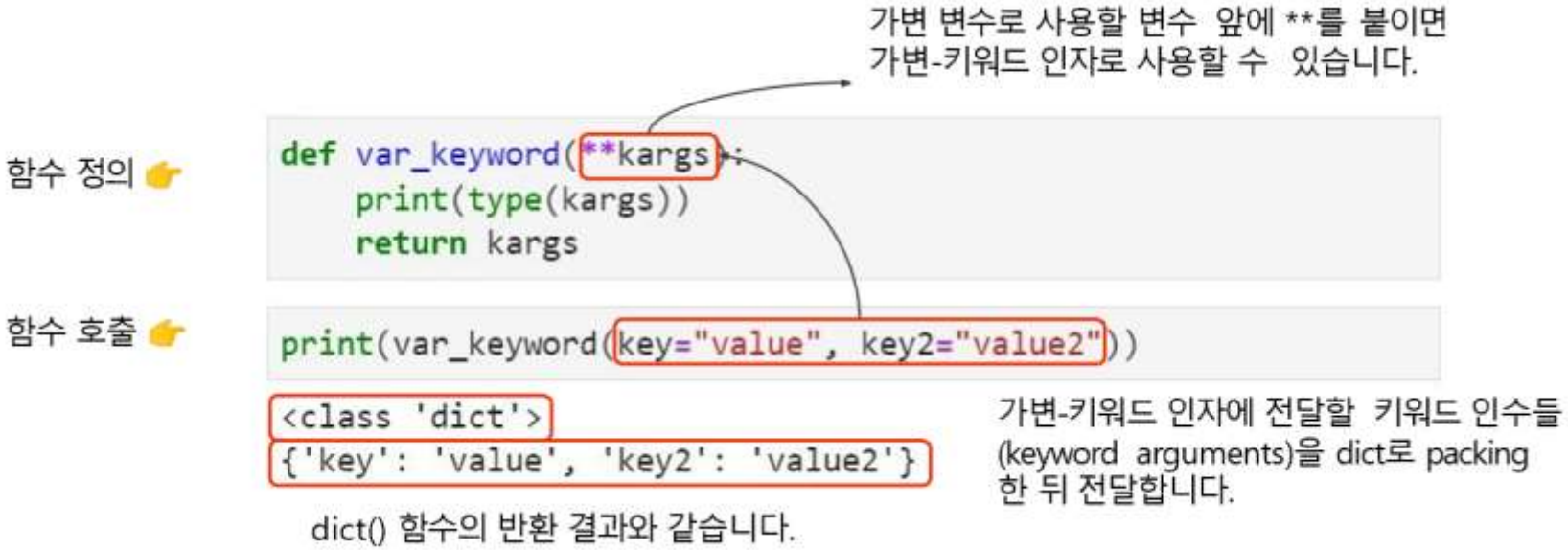
함수(function) - 가변-위치 인자

- ✓ 가변-위치 인자(var-positional parameters)는 명시된 인자 외에 추가적으로 위치 인수를 개수에 상관없이 유연하게 전달 받을 수 있다.
- ✓ 가변 변수로 사용할 변수명 앞에 *를 앞에 하나 붙여 표기.



함수(function) - 가변-키워드 인자

- ✓ 가변-키워드 인자(var-keyword parameters)는 명시된 인자 외에 추가적으로 키워드 인수를 개수에 상관없이 유연하게 전달 받을 수 있다.
- ✓ 가변 변수로 사용할 변수명 앞에 *를 앞에 두개 붙여 표기.



함수(function) - 연습 문제

✓ 각 입력받은 정수를 10씩 더해서 반환하는 함수를 만들어보세요.

○ 함수명: plus_ten

```
실행결과
plus_ten(70)
반환 값: 80
```

함수(function) - 연습 문제

✓ 화씨 온도를 섭씨 온도로 변경하는 함수를 만들어보세요. 화씨 온도를 섭씨 온도로 변경하는 공식은 아래와 같습니다.

- ℃ = (℉ − 32) × 5/9
- 함수명: to_celsius
- 인자: fahrenheit
- 반환: 섭씨 온도

```
실행결과

to_celsius(70)

반환 값: 21.11111111111111
```

함수(function) - 연습 문제

✓ 자연수를 인수로 전달하면 짝수일 때 “짝수”라는 문자열을, 홀수일 때 “홀수”라는 문자열을 반환하는 함수를 만들어보세요.

- ○ 자연수에 대해서만 “짝수”, “홀수” 반환합니다.
- ○ 자연수가 아닌 값이 들어왔을 때 반환값 없이 함수를 즉시 종료합니다.

```
print(odd_even.__doc__)
print(odd_even(10))
print(odd_even(9))
print(odd_even("가"))
```

```
number가 짝수면 "짝수", 홀수면 "홀수" 반환
짝수
홀수
None
```

함수(function) - 연습 문제

✓ 연도를 나타내는 숫자와 월을 나타내는 숫자를 인수 값으로 각각 전달하면 그 달의 날짜 개수를 반환하는 함수 days 를 만들어보세요.

○ 윤년, 평년도 계산해야 합니다.

실행결과	실행결과	실행결과	실행결과
days(1900, 11)	day(2004, 10)	days(1900, 2)	days(2000, 2)
반환 값: 30	반환 값: 31	반환 값: 28	반환 값: 29

함수(function) - 연습 문제

✓ 아래 소스 코드를 완성하여 몫과 나머지를 구하는 함수를 완성해보세요.

소스 코드:

```
x = 10

y = 3
get_quotiont_remainder() 함수를 정의하세요.
quitient, remainder = get_quotient_remainder(x, y)

print(f'몫: {quitient}, 나머지: {remainder }')
```

실행 결과:

```
몫: 3, 나머지: 1
```

변수의 범위

김지성 강사

전역 변수(Global Variable)

- ✓ 우리가 함수를 배우기 전까진 제일 바깥 영역인 전역 범위(global scope)에서 변수를 선언하고 활용했다.
- ✓ 전역 범위에서 선언했기 때문에 그 변수를 스크립트 전체에서 접근할 수 있었는데, 그 변수를 전역 변수(global variable)라고 부른다.

```
global_variable = "This is global world."

print(f"global_variable in global scope=> {global_variable}")
print(hex(id(global_variable)))

def local_world():
    print(f"global_variable in local_world=> {global_variable}")
    print(hex(id(global_variable)))

local_world()
```

👉전역 변수 선언

👉함수 내부(local scope)에서 전역 변수에 대한 조회가 가능하다.

```
global_variable in global scope=> This is global world.
0x1e62dc7a260
global_variable in local_world=> This is global world.
0x1e62dc7a260
```

지역 변수(Local Variable)

✓ 그렇다면 함수 바디(local scope)에서 선언된 변수는 바깥에서 접근이 가능할까?

○ NameError가 에러가 발생됩니다. 즉 지역 변수는 변수를 만든 함수 안에서만 접근이 가능한 것을 확인할 수 있다.

지역 범위(local scope)

```
def my_little_world():
    my_variable = "This is my little world."
```

👉지역 변수 선언

```
print(my_variable)
```

👉에러 발생 지점

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-34-f7bf4eb2daee> in <module>
      2     my_variable = "This is my little world."
      3
----> 4 print(my_variable)

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

지역 범위에서 전역 변수 할당하기

- ✓ 지역 범위에서 전역 변수에 대한 접근이 가능하다고 했다. 그렇다면 지역 범위에서 전역 변수의 값을 변경하는 건 가능할까?
- ✓ 아래의 예제에서 그 시도를 했을 때 결론은 변경이 되지 않는다. 지역 범위에서 할당하고 있는 'important_…_heart' 변수는 사실 전역 변수가 아닌 지역 변수였기 때문이다.

```
important_is_an_unbroken_heart = "중요한 것은 꺾이지 않는 마음"

def trials_and_tribulations():
    important_is_an_unbroken_heart = "흔들흔들"

trials_and_tribulations()
print(important_is_an_unbroken_heart)
```

중요한 것은 꺾이지 않는 마음 👉지역 범위에서 값을 입력 시도 했지만 변경되지 않았습니다.

지역 범위에서 전역 변수 할당하기

✓ 지역 범위에서 전역 변수를 변경하려면 global 키워드를 사용해야 한다.

```
global_variable = "변경이 됩니까?"

def heartbreaker():
    global global_variable
    global_variable = "global 키워드 활용해서 변경 완료"

heartbreaker()
print(global_variable)
```

👉global 키워드를 활용하여 전역 변수를 사용하겠다고 설정

```
global 키워드 활용해서 변경 완료
```

함수 바디에 함수를 중첩하기

✓ 함수 바디 안에 def를 써서 다시 함수를 만들 수 있다.

✓ level2 함수에서 바깥쪽 level1 함수의 지역변수 message를 출력하고 있다. level1에 선언된 지역 변수는 level1 함수 바디 범위 내에서 접근 가능함을 알 수 있습니다.

```
def level1():
    message = "This is level 1."
    def level2():
        print(message)
    level2()
```

👉message 변수에 접근할 수 있는 범위

```
level1()
```

```
This is level 1.
```

함수 바디에 함수를 중첩하기

✓ 안쪽 함수에서 바깥쪽 함수에 선언된 변수에 대한 출력이 가능한 것을 확인해봤다. 변경도 가능할까?

```
def level1():
    message = "This is level 1."
    def level2():
        message = "Level2 is better than level1."
    level2()
    print(message)
```

새로운 범위의 지역 변수 message를 만든 것입니다.

바깥쪽 함수 영역의 변수 값 변경에 실패했습니다.

```
level1()
```

```
This is level 1.
```

함수 바디에 함수를 중첩하기

✓ 안쪽 함수에서 바깥쪽 함수에 선언된 변수에 대한 출력이 가능한 것을 확인해봤다. 변경도 가능할까?

```
def level1():
    message = "This is level 1."
    def level2():
        nonlocal message
        message = "Level2 is better than level1."
    level2()
    print(message)
```

nonlocal 키워드를 통해 바깥 영역의 변수를 사용하겠다고 설정

바깥쪽 함수 영역의 변수 값 변경에 성공했습니다.

```
level1()
```

```
Level2 is better than level1.
```

global, nonlocal 특징

- ✓ global 키워드는 함수의 중첩된 정도와 상관없이 전역 범위의 변수를 매칭.
- ✓ 중첩된 함수마다 같은 이름의 변수가 있다면 nonlocal 키워드는 제일 가까운 바깥 변수를 매칭.
- ✓ 가급적이면 함수마다 이름이 같은 변수를 사용하기보단 다른 변수명을 사용하는 것이 좋다.

변수 범위 정리

- ✓ 함수 안에서 선언한 변수는 함수를 호출해 실행되는 동안만 사용할 수 있다.
- ✓ 범위마다 같은 이름의 변수를 사용해도 각각 독립적으로 동작한다.
- ✓ 지역 변수(local variable)를 저장하는 이름 공간을 지역 영역(local scope)라고 한다.
- ✓ 전역 변수(global variable)를 저장하는 이름 공간을 전역 영역(global scope)라고 한다.
- ✓ 파이썬 자체에서 정의한 이름 공간을 내장 영역(built-in scope)이라고 합니다.
- ✓ 함수에서 변수를 호출하면 지역 영역 → 전역 영역 → 내장 영역 순으로 해당하는 변수를 확인한다. 이 순서를 LEGB 규칙이라고 한다.

변수 범위 정리

✓ 함수에서 변수를 호출하면 지역 영역 → 전역 영역 → 내장 영역 순으로 해당하는 변수를 확인한다. 이 순서를 LEGB 규칙이라고 한다.

```
# 내장 영역(built-in scope)에 있는 함수
print(len("Python"))  # 내장 함수 len 사용 (결과: 6)

# 지역 및 전역 변수와 내장 함수 이름 충돌
len = 10  # 전역 변수로 len을 선언 (내장 함수와 이름 충돌)
print(len)  # 결과: 10 (내장 함수 len을 덮어씀)

# 내장 함수를 다시 사용하려면 삭제 필요
del len
print(len("Python"))  # 결과: 6 (내장 함수 복구)
```

```
print(dir(__builtins__))  # 내장 영역에 정의된 이름 목록 출력
```

람다식(Lambda)

김지성 강사

람다 (Lambda)

- ✓ 호출될 때 값이 구해지는 하나의 표현식이다. 또한 이름이 없는 인라인 함수이다.
- ✓ 아래의 두 식은 동일한 식이다.

```
lambda [parameters]: expression
```

매개변수를 지정 — 반환값으로 사용할 식

👈 Lambda 표현식

일반 함수 정의절 👉

```
def func_name(parameter):
    return expression
```

람다 (Lambda)

✓ 람다 표현식을 바로 호출하는 방법은 다음과 같다.

1. 람다 표현식 전체를 소괄호 ()로 감싼다.
2. 뒷쪽에 작성한 소괄호 안에 인수를 넣으면 람다 표현식이 바로 호출된다.

```
(lambda x: x + 10)(10)
```

```
20
```

(lambda [parameters]: expression)(arguments)

람다 (Lambda)

✓ 람다는 기본적으로 이름없는 함수(anonymous function)이다.

✓ 람다로 만든 익명 함수를 호출하려면 변수에 할당해서 사용할 수 있다.

○ 파이썬에서 함수는 일급 객체(first-class object)이다. 특징은 다음과 같다.

1. 변수 할당 가능
2. 다른 함수의 인자로 전달 가능
3. 다른 함수의 반환값으로 사용 가능
4. 데이터 구조 안에 저장 가능

```
twice = lambda x: x*2
print(twice(10))
```

```
20
```

```
variable = lambda [parameters]: expression
variable(argument)
```

람다 (Lambda)

- ✓ 람다의 expression 부분은 변수 없이 식 한 줄로 표현 가능(인라인)해야 한다.
- ✓ 따라서 람다 표현식 안에 새 변수를 만들 수 없다. 변수가 필요한 경우는 def를 써서 함수를 정의해서 사용하는 것이 좋다.

```
(lambda x: y=10; x + y)(1)
```

```
  File "<ipython-input-1-f6633d85a5c5>", line 1
    (lambda x: y=10; x + y)(1)
                   ^
SyntaxError: invalid syntax
```

람다 (Lambda)

- ✓ if else를 한 줄로 작성할 수 있는 방법을 활용하여. lambda에 사용하면 활용도가 좋다.
- ✓ 주의할점은 if만 사용할 수 없다. 반드시 else와 같이 써야한다. 중첩해서 사용 가능하지만 중첩이 너무 많아지면 파악이 어려우니 가독성 좋게 쓰는 것이 중요하다.


```
score = 90
'A' if 90<score<=100 else 'B' if 80<score else 'C'
```

```
'B'
```

람다 (Lambda) Map() 함수와 함께 사용하기

✓ 람다와 map() 함수는 자주 함께 사용된다. 그 이유는 map() 함수의 파라미터를 보면 알 수 있다.

○ map(function, iterable)

1. function : 각 요소에 적용할 함수
2. iterable : 함수를 적용할 데이터 집합

```
def square(x):
    return x**2

numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)
print(list(squared_numbers))  # [1, 4, 9, 16, 25]
```

```
[1, 4, 9, 16, 25]
```

람다 (Lambda) Map() 함수와 함께 사용하기

✓ 람다와 map() 함수는 자주 함께 사용된다. 그 이유는 map() 함수의 파라미터를 보면 알 수 있다.

○ map(function, iterable)

1. function : 각 요소에 적용할 함수
2. iterable : 함수를 적용할 데이터 집합

```
def add(x, y):
    return x + y

numbers1 = [1, 2, 3, 4, 5]
numbers2 = [10, 20, 30, 40, 50]
added_numbers = map(add, numbers1, numbers2)
print(list(added_numbers))  # [11, 22, 33, 44, 55]
```

```
[11, 22, 33, 44, 55]
```

람다 (Lambda) Map() 함수와 함께 사용하기

✓ 람다와 map() 함수를 같이 쓸 때 다음과 같이 사용한다.

✓ 이때 필요에 따라 parameter와 iterable의 개수를 맞춰서 사용하면 된다. 어느 한쪽 iterable의 길이가 짧아도 상관없이 동작하며, 길이가 작은 쪽에 맞춰 값을 반환합니다.

map(lambda parameter1, parameter2, …: expression, iterable1, iterable2, …)

```
numbers1 = [1, 2, 3, 4, 5]
numbers2 = [10, 20, 30, 40, 50]

# lambda를 사용한 map
added_numbers = list(map(lambda x, y: x + y, numbers1, numbers2))
print(added_numbers)  # [11, 22, 33, 44, 55]
```

```
[11, 22, 33, 44, 55]
```