

# 파이썬의 조립 블록 구성 요소

## Assembly Block Components



서지혜 교수

서울과학기술대학교

[jihae@seoultech.ac.kr](mailto:jihae@seoultech.ac.kr)

# 학습 목차

☑ 함수의 기초 개념

☑ 함수의 종류

☑ 함수의 특성

- ▶ 값 반환 함수와 보이드 함수
- ▶ 전달인자와 매개변수
- ▶ 범위와 가시성

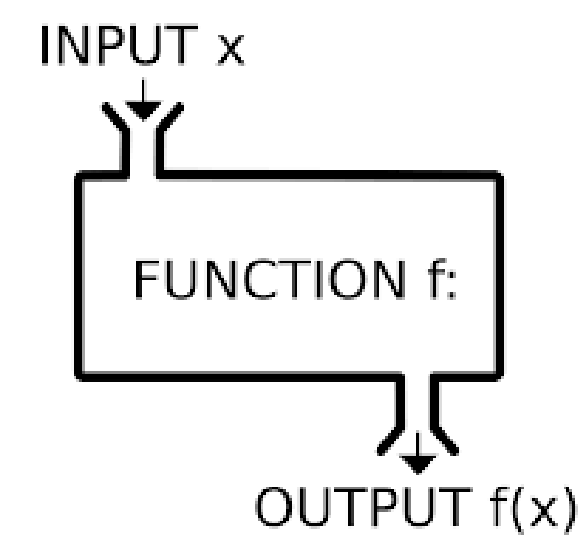
☑ 모듈과 패키지

☑ 파이썬 표준 라이브러리

# 함수의 기초 개념

---

## Function Primer



- 함수(function)란?
  - 함수란 재사용이 가능한 일련의 명령문 묶음
    - 특정 명령을 수행하는 코드를 매번 다시 작성하지 않고 필요할 때마다 호출하여 간편하게 사용할 수 있는 형태로 만든 것
    - 특정 작업을 함수로 정의함으로써 언제든지 불러와서 사용할 수 있기 때문에 코드의 재활용성을 높일 수 있다
  - 프로시저(procedure), 메소드(method), 서브루틴(subroutine) 등의 이름으로 불리기도 한다
- 사용자 함수(custom function)란?
  - 우리가 원하는 기능을 가진 함수는 대부분의 경우 누군가에 의해 이미 개발되지만, 구현하려는 기능이 기존 함수에 없으면 직접 함수를 작성해서 사용할 수 있다
  - 이렇게 직접 작성한 함수를 '사용자 함수(custom function)'라 부른다
- 함수의 구성 요소
  - 함수 이름 + 일련의 명령문 묶음

함수를 사용하면 복잡한 프로그램을 모듈화해서 몇 가지 간단한 단계의 연속으로 표현할 수 있다

- 함수를 사용할 때 필요한 정보
  - 함수 이름
  - 함수를 실행할 때 필요한 전달인자 개수와 전달인자 각각의 자료형
    - 전달인자란 함수를 호출할 때 전달하는 값
    - 전달인자는 필요 없을 수도 있다
  - 함수가 반환하는 값의 자료형(return data type)
    - 함수는 값을 반환하지 않을 수도 있다
  
- 예) 문자열 길이를 구하기 위해서는 다음 정보가 필요
  - 함수 이름 : `len()`
  - `len()` 함수의 실행에 필요한 전달인자 개수는 1개이고 전달인자의 자료형은 문자열
  - `len()` 함수가 반환하는 값의 자료형 : 정수

```
x = len('파이썬')
print(x)
```

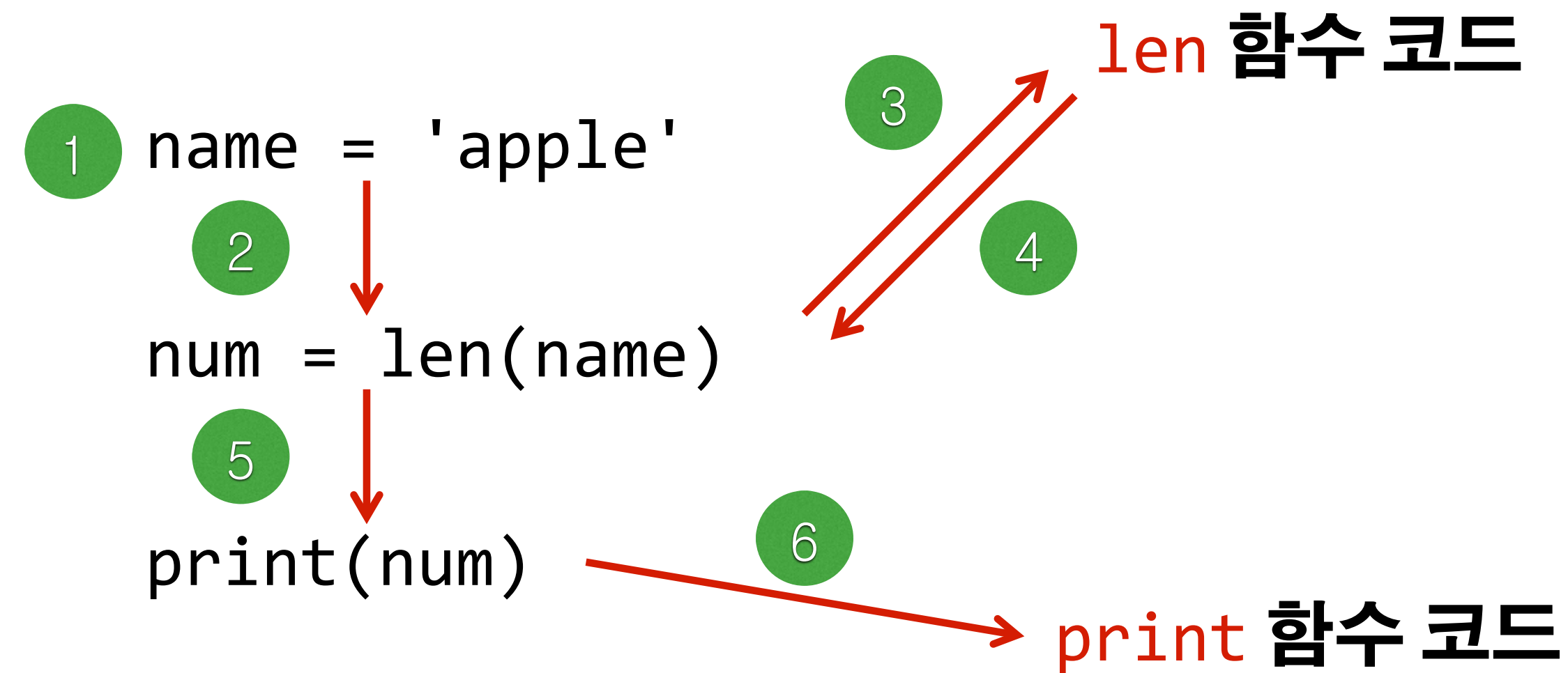
3

```
함수이름([ 전달인자-1, ..., 전달인자-N ])
변수 = 함수이름([ 전달인자-1, ..., 전달인자-N ])
```

- 함수 호출(function call)이란?
  - 함수를 불러와서 실행하는 것
- 함수를 호출하면
  - 인터프리터가 해당 함수 코드로 건너가서 명령어들을 실행
  - 해당 함수에 있는 명령어들의 실행이 끝나면 함수를 호출한 코드로 다시 돌아간다
    - 이를 '**함수 반환**(function return)'이라 부른다
- 함수 호출에 필요한 것
  - 함수 이름
  - 전달인자(arguments)
    - 전달인자는 필요 없을 수도 있다
- 함수
  - 전달인자를 받아와서 특정한 작업을 실행한 후
  - 반환할 값값이 있으면 값값을 반환하고 종료하든지, 반환하지 않아도 되면 반환하지 않고 종료한 후
  - 호출한 코드로 실행 권한을 넘긴다

# 예시 : 함수 호출

- 문자열 길이 구하기 코드



- 프로그램 실행 순서

- 문자열 'apple'을 변수 `name`에 할당
- 변수 `num`에 `len(name)`의 결과값을 받기 위해 `len` 함수를 호출
- `len` 함수를 호출하면서 이 함수에 전달인자 `name`을 넘겨준다
- 결과값으로 정수 5를 반환 → 이 값을 변수 `num`에 할당
- `num` 값을 출력하기 위해 `print` 함수를 호출
- `print` 함수를 호출하면서 전달인자 `num`을 넘겨주고 `print` 함수는 정수 5를 출력



중첩 함수 호출(nested function call)이란? 함수를 호출할 때 전달인자로 다른 함수를 사용하는 것

**함수1(함수2(...(함수N([전달인자]))))**

- 함수N에서 반환한 값을 함수N-1에 전달인자로 넘겨주고,
- 함수N-1에서 반환한 값 함수N-2에 전달인자로 넘겨주고, ...
- 최종적으로 가장 바깥에 있는 함수1이 함수2로부터 반환받은 값을 처리

```
price = eval(input('제품의 가격을 입력하세요: '))
print('총 금액이 ' + str(round(price * 1.1)) + '입니다.')
```

제품의 가격을 입력하세요: 100  
총 금액이 110입니다.



		전달인자	
		전달인자 (X)	전달인자 (O)
반환값	반환값 (X)	함수이름()	함수이름(전달인자 <sub>1</sub> , ..., 전달인자 <sub>N</sub> )
	반환값 (O)	변수 = 함수이름()	변수 = 함수이름(전달인자 <sub>1</sub> , ..., 전달인자 <sub>N</sub> )

- 전달인자(arguments)
  - 대부분의 파이썬 함수는 값을 전달인자로 받아서 작업을 수행
  - 전달인자(arguments)는 소괄호(`( )`) 안에 할당
    - 어떤 전달인자는 선택 사항이며 함수 정의에서 대괄호(`[]`)로 표시
    - 어떤 전달인자는 반드시 값을 전달해야 하며 그렇지 않으면 오류가 발생
  
- 예) `range([시작번호, ] 끝번호[, 폭])`
  - `시작번호` : 순서 열의 시작 번호
  - `끝번호` : 순서 열의 최댓값(이 값은 포함되지 않는다)
  - `폭` : 순서 열에서 서로 붙어 있는 번호 간의 간격

# 예시 : 함수 호출과 전달인자

```
# 필수 전달인자 한 개를 사용한다.
for i in range(10):
    print(i, end='/')
```

```
# 전달인자 두 개를 사용한다.
for i in range(1, 10):
    print(i, end='/')
```

```
# 전달인자가 세 개를 사용한다.
for i in range(1, 10, 2):
    print(i, end='/')
```

```
# 폭이 음수다.
for i in range(5, 1, -1):
    print(i, end='/')
```

# 매개변수 vs. 전달인자

**매개변수**(parameter) : 함수를 실행할 때 함수 내부에서 사용하는 식별자

**전달인자**(argument) : 함수를 호출할 때 넣어주는 값

```
def say_anything(anything):  
    print('안녕', anything)
```

```
say_anything('파이썬')
```

안녕 파이썬

# 예시 : 사용자 함수

```
def hi():
    print('안녕하세요 교수님!')

def john():
    print('제 이름은 John입니다.')

def emily():
    print('제 이름은 Emily입니다.')

def amy():
    print('제 이름은 Amy입니다.')

hi()
emily()
john()
amy()
```



# 매개변수를 사용하여 함수 일반화하기

```
def hi():
    print('안녕하세요 교수님!')

def john():
    print('제 이름은 John입니다.')

def emily():
    print('제 이름은 Emily입니다.')

def amy():
    print('제 이름은 Amy입니다.')

hi()
emily()
john()
amy()
```



안녕하세요 교수님!  
 제 이름은 Emily입니다.  
 제 이름은 John입니다.  
 제 이름은 Amy입니다.

john(), emily(), amy()  
 함수를  
 하나로 합칠 수 있는 방법이  
 있을까?





# 매개변수를 사용하여 함수 일반화하기

## 함수 일반화

- `john`, `emily`, `amy` 함수를 매개변수를 사용해 `person` 함수로 일반화한다
- 이제 사용자는 매개변수 값을 다르게 함으로써 하나의 함수만으로도 다양한 결과값을 얻을 수 있다
- 예를 들어, **‘제 이름은 홍길동입니다.’** 이라는 결과 값을 얻고 싶다고 했을 때 이전 코드에서는 새로운 함수를 작성해야만 했지만 매개변수를 통해 일반화된 함수를 사용하면 단순히 함수를 호출할 때 `person('홍길동')`으로 매개변수 값, 즉 전달인자만 변경해 주면 되기 때문에 매우 간편하다

```
def hi():
    print('안녕하세요 교수님!')

def person(name):
    print(f'제 이름은 {name}입니다.')
```

```
hi()
person('Emily')
person('John')
person('Amy')
```



```
안녕하세요 교수님!
제 이름은 Emily입니다.
제 이름은 John입니다.
제 이름은 Amy입니다.
```

- ☑ 매개변수를 사용하면 굳이 새로운 함수를 생성할 필요 없이 단순히 매개변수 값(즉, 전달인자)을 변경하는 것만으로도 원하는 결과값을 얻을 수 있다



```

def hi():
    2 print('안녕하세요 교수님!')

4 def person(name):
    5 print(f'제 이름은 {name}입니다.')

1 hi()
3 person('Emily')
6 person('John')
7 person('Amy')
  
```



안녕하세요 교수님!  
 제 이름은 Emily입니다.  
 제 이름은 John입니다.  
 제 이름은 Amy입니다.

- 1 hi 함수 호출
- 2 '안녕하세요 교수님!' 출력
- 3 person 함수 호출, 전달인자로 'Emily' 사용
- 4 넘겨받은 전달인자('Emily')를 name이라는 매개변수에 할당
- 5 print 함수를 사용해서 매개변수 name을 출력
- 6 person 함수 호출, 전달인자로 'John' 전달
- 7 person 함수 호출, 전달인자로 'Amy' 전달

4 5  
4 5

과정 반복  
 과정 반복

# 예시 : 사용자 함수 정의

```
# 매개변수와 반환 값이 없는 함수 정의
def my_function():
    print('I love Python')
```

→ # 함수 호출  
my\_function()

```
# 매개변수와 반환 값이 있는 함수 정의
def my_function2(language):
    return 'I love ' + language
```

# 함수에 전달인자를 사용함으로써, 함수의 재사용성을 높일 수 있다.

```
x = my_function2('Python')
print(x)

y = my_function2('Java')
print(y)

z = my_function2('C++')
print(z)
```

→

→

→

# 예시 : 사용자 함수 정의

```
# if-else문을 사용하는 함수 정의
def my_function3(language):
    if language in ['Python', 'Java', 'Ruby']:
        return 'I love ' + language
    elif language in ['C++', 'C']:
        return 'I like ' + language
    else:
        return 'I do not know ' + language
```

```
x = my_function3('Ruby')
print(x)

y = my_function3('C++')
print(y)

z = my_function3('Scala')
print(z)
```

```
# for문을 사용하는 함수 정의
def my_function(languages):
    for language in languages:
        if language in ['Python', 'Java', 'Ruby']:
            print('I love ' + language)
        elif language in ['C++', 'C']:
            print('I like ' + language)
        else:
            print('I do not know ' + language)
```

```
my_function(['Ruby', 'Python', 'Fortran', 'C'])
```


```
my_function(('Java', 'C++', 'C#'))
```

- 자기 자신을 호출하는 함수
  - e.g. 계승 함수(factorial function)
    - 음이 아닌 정수  $n$ 의 계승은  $n!$ 로 표현하며 계승은 1부터  $n$ 까지의 정수를 모두 곱한 것을 의미
      - e.g.,  $5! = 5 * 4 * 3 * 2 * 1 = 120$
    - $0!$ 의 값은 1

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

```
factorial(5)
```

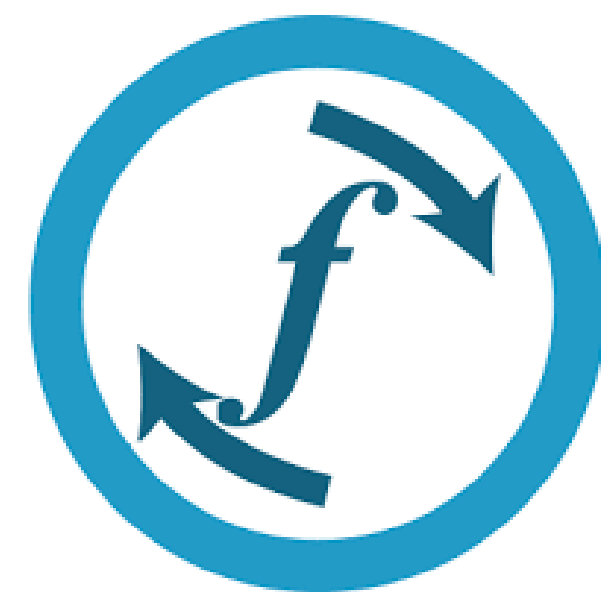
```
120
```



# 함수의 종류

---

Function Types



# 함수의 4가지 종류

- **전역 함수**(global function)
  - 같은 모듈(즉, 같은 `.py` 파일)내 어디서든 사용 가능
  - 다른 모듈에서도 사용 가능
    - 즉, 다른 파일에서도 불러올 수 있다(`import`, `from ... import ...`)
- **지역 함수**(local function)
  - 특정 함수 안에 정의한 함수(**내재함수**라고도 부른다)
  - 지역 함수를 정의한 함수 내에서만 사용 가능
  - 다른 곳에서는 사용하지 않는 간단한 도우미 함수가 필요할 때 유용
- **람다 함수**(lambda function)
  - 사용할 시점에 **표현식** 형태로 정의하여 바로 사용하는 함수
  - 일반적인 함수보다 구현할 수 있는 기능이 제한적
- **메소드**(method)
  - 클래스 내부에서 구현한 함수
  - 특정 자료형, 즉, 클래스와 관련된 함수이기 때문에 해당 클래스의 인스턴스(또는 해당 클래스)와 함께 사용해야 한다
    - 예) `문자열.format()`

# 함수의 특성

---

Function Properties





# 값 반환 함수와 보이드 함수

Value-Returning Functions vs. Void Functions



- 함수의 명령문을 실행한 후 함수를 호출한 코드에 결과값을 **return**문을 통해 돌려준다
- 값 반환 함수**(value-returning function)란?
  - 따라서, 호출한 코드는 함수가 반환하는 결과값을 변수에 저장(할당)할 수가 있다
  - 예) **input**, **len** 함수 등

**변수 = 함수이름([전달인자-1, ..., 전달인자-N])**

```
x = len('파이썬')
print(x)
```

```
# 값 반환 함수의 결과값을 변수에 저장(할당)하지 않으면 어떻게 될까?
len('파이썬')
```

대화형 모드에서 값 반환 함수를 실행할 때  
 함수가 반환한 값을 변수로 받지 않으면  
 파이썬 셸로 반환(출력)한다

```
def value_returning_product(i, j):
    """값 반환 함수는 return문을 사용해 값을 반환한다."""
    return i * j
```

```
x = value_returning_product(3, 5)
print(x)
```

```
value_returning_product(3, 5)
```

- 함수의 명령문을 실행한 후 함수를 호출한 코드에 결과값을 반환하지 않고 종료
- 보이드 함수**(void function)란? 즉, **return** 문이 없어 **None** 을 반환
- 예) **print** 함수 등

**함수이름**([전달인자-1, ..., 전달인자-N])

```
print('안녕 파이썬')
```

```
# 값 반환 함수를 호출한 것처럼 보이드 함수를 호출한 후 변수로 결과값을 받아보자
x = print('안녕 파이썬')
```

```
print(x)
```

**x** 에는 아무 값도 없는데 왜 그럴까? **print** 함수는 값을 반환하지 않는 보이드 함수이기 때문

```
x # x 값을 대표 형식으로 확인한다
```

# 예시 : 보이드 함수

```
def void_product(i, j):
    """보이드 함수는 return문이 없다."""
    print(f'{i} x {j} = {i * j}')
```

```
void_product(3, 5)
```

```
x = void_product(3, 5)
```

```
print(x)
```

```
x
```

# 두 개 이상의 값을 반환

다음 함수는 **return** 문이 두 개

```
def value_returning_fn(i, j):
    """return문이 두 개(이상) 있다."""
    return i - j
    return i * j

# 3과 5를 전달인자로 해서 이 함수를 호출
x = value_returning_fn(3, 5)

# 결과값이 두 개인가? 아니면 첫 번째 return문인가? 두 번째 return문인가?
print(x)
```

☑ 함수는 항상 단일한 값만 반환

return 문을 한 번 사용하면서 두 개 이상의 값을 반환할 수 있을까?

## 튜플 반환이란?

- 함수는 항상 단일 값만 반환할 수 있지만 반환 값으로 튜플을 사용하면 복수의 값을 반환하는 것과 같은 결과를 얻을 수 있다
- 파이썬이 제공하는 대표적인 내장 함수로 **divmod(x, y)**가 있다



```
i = divmod(13, 7)
print(i)
```



```
quot, rem = divmod(13, 7)
print(quot)
print(rem)
```





# 예시 : 튜플 반환

```
def value_returning_fn2(i, j):
    """전달받은 두 정수의 사칙연산 결과를 튜플로 반환하는 함수다."""
    return i + j, i - j, i * j, i / j
```

→

```
x = value_returning_fn2(3, 5)
print(x)
```

→

```
i, j, k, l = value_returning_fn2(3, 5)
print(i)
print(j)
print(k)
print(l)
```

```
def min_max(k):
    """리스트나 튜플을 전달인자로 받는 함수다."""
    return min(k), max(k)
```

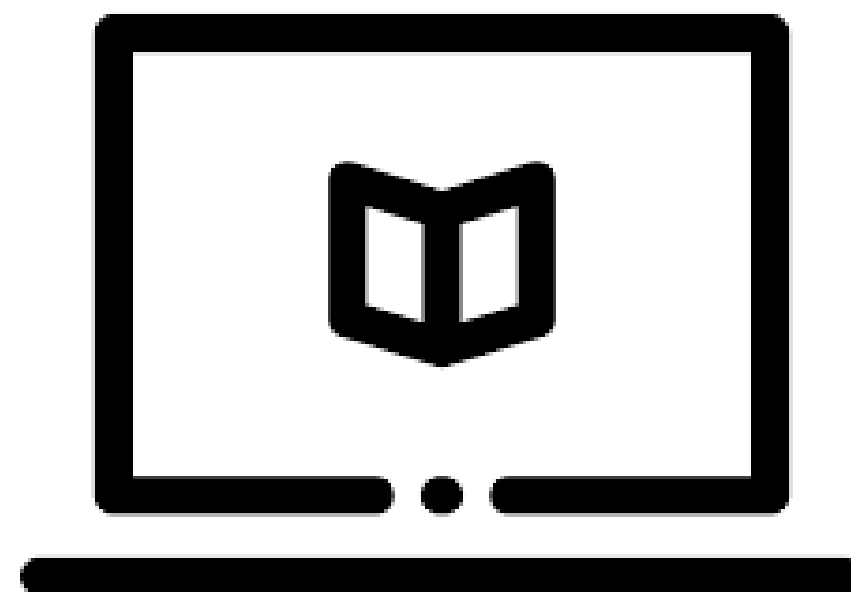
→

```
min, max = min_max((12, 45, 98, 38, 85, 7, 49))
print(min)
print(max)
```

# 파이썬의 조립 블록 구성 요소

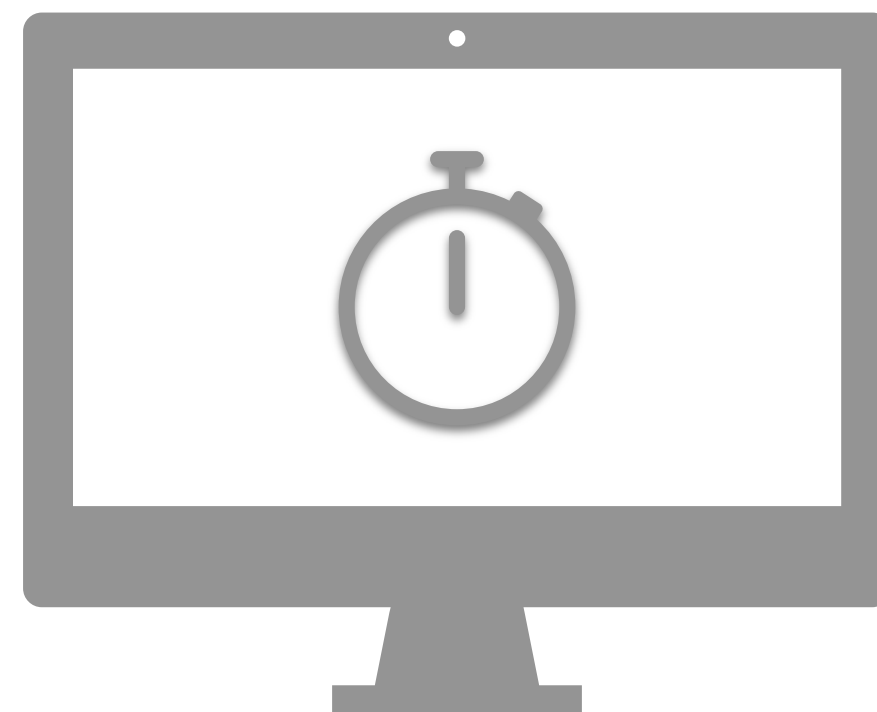
---

## Lab Exercises



# 함수의 기초 개념

---





## • 전달인자로 함수 조작하기

- 정수 1부터 100까지 포함하는 리스트를 생성한 후 출력
- 앞서 만든 리스트의 각 항목을 1씩 증가시킨 후 튜플 자료형으로 형변환 한 후 출력

💡 **range** 클래스로 숫자를 생성한다

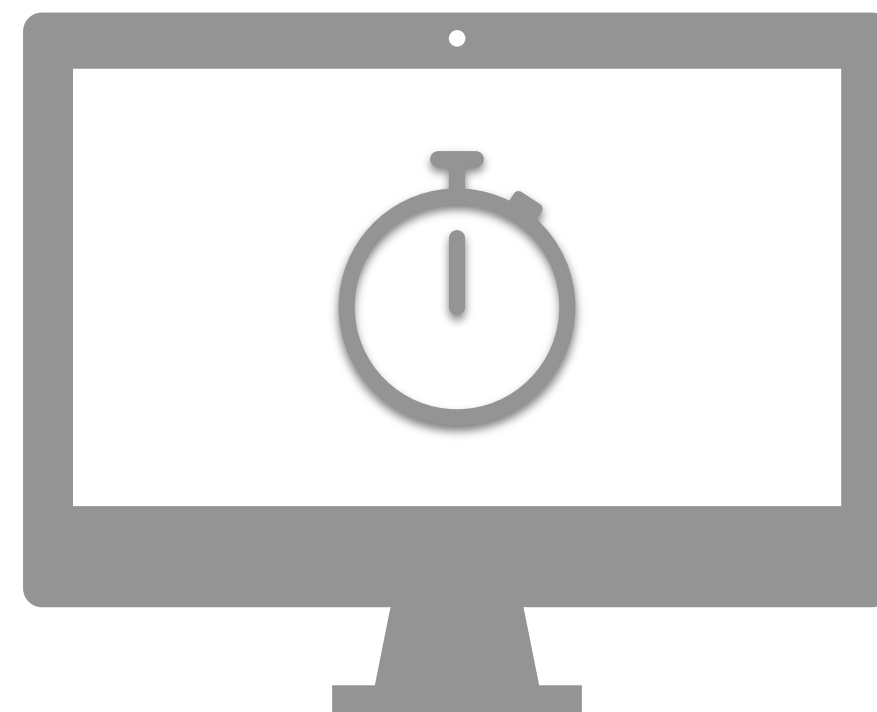
## • 실행 결과 예시

```
[1, 2, 3, 4, 5, ... (중략) ..., 96, 97, 98, 99, 100]
(2, 3, 4, 5, 6, ... (중략) ..., 97, 98, 99, 100, 101)
```

```
L = list(range(1, 101))  
print(L)  
  
for i in range(len(L)):  
    L[i] += 1  
  
t = tuple(L)  
print(t)
```

# 값 반환 함수와 보이드 함수

---



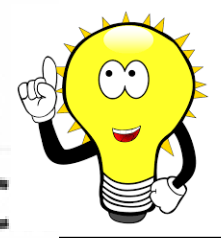


- 두 직사각형 면적의 비율을 계산하는 함수 구현하기
  - 직사각형의 가로와 세로 길이를 정수로 입력받아 직사각형 넓이를 계산한 후, 가로의 길이는 **5** 증가시키고 세로의 길이는 두 배로 확장한 직사각형의 넓이를 계산한 후에 원래 넓이에서 확장한 후의 넓이를 나눈 값을 소수점 두 자리까지 출력하는 **expand\_rectangle** 함수를 구현
  - 원래 입력한 직사각형 넓이를 확장한 직사각형 넓이로 나눈 비율을 출력하기 전에 확장한 가로와 세로 길이를 먼저 출력
  - **가로 길이 = 7, 세로 길이 = 10**을 넣고 결과를 테스트

- 실행 결과 예시

```
>>> expand_rectangle(7, 10)
Width = 12
Length = 20
Area Ratio = 0.29
```





```
def expand_rectangle(w, h):
    area_original = w * h
    w += 5
    h *= 2

    area_expanded = w * h
    ratio = area_original / area_expanded

    print('Width =', w)
    print('Length =', h)
    print('Area Ratio =', f'{ratio:.2f}')
```

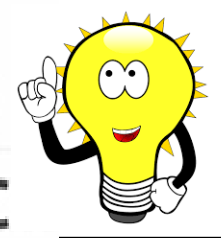


- 영어 모음 개수를 계산하는 함수 구현하기
  - 영어 단어(또는 문장) 한 개를 입력받아 대소문자 구분 없이 단어(또는 문장) 내에 속한 모음 알파벳(a, e, i, o, u)의 개수를 반환하는 **vowel** 함수를 구현

- 실행 결과 예시

```
>>> vowel('Apples')
2

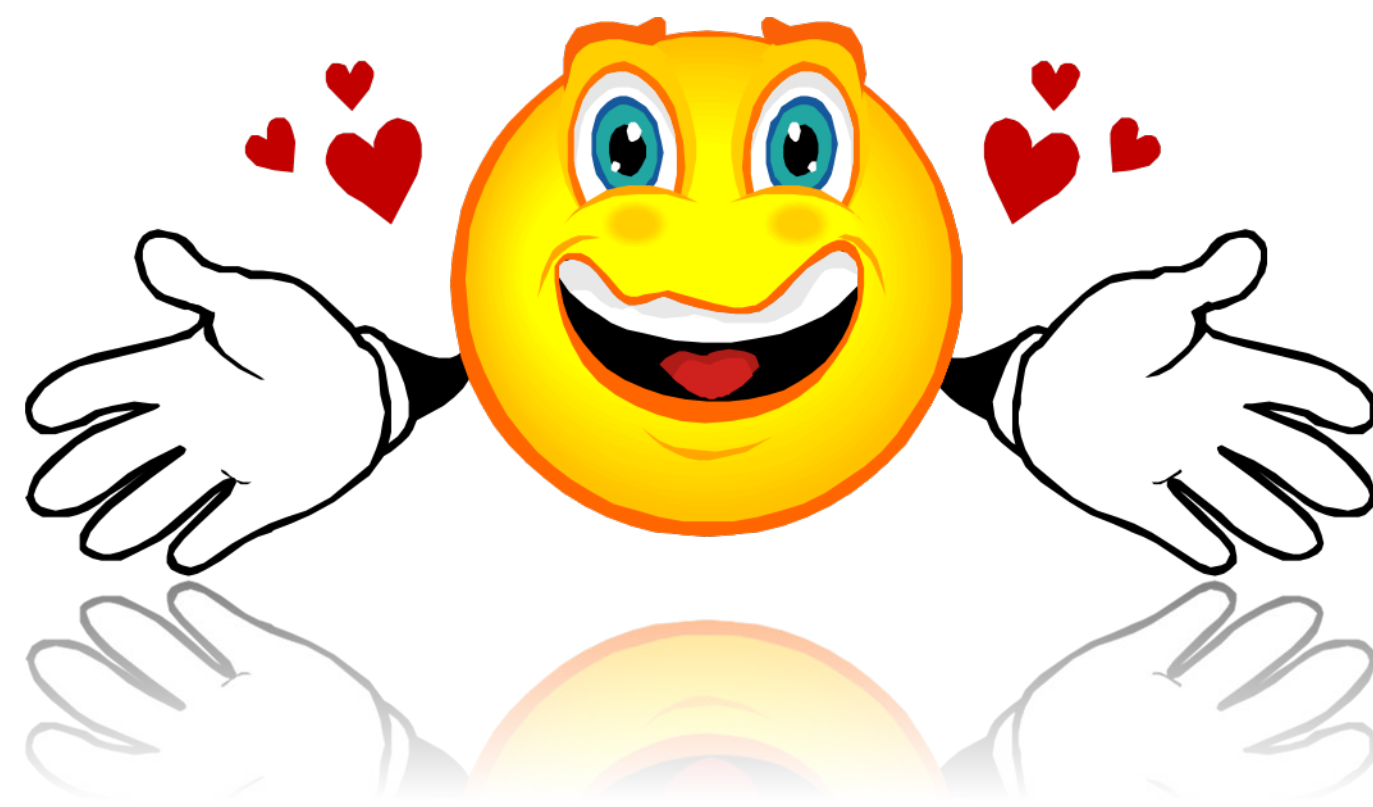
>>> vowel('I love Python')
4
```



```
def vowel(s):
    vowels = 'aeiou'
    count = 0

    for char in s:
        if char.lower() in vowels:
            count += 1

    return count
```



끝

수고 하셨습니다