



CHAPTER 10



파이썬의 조립 블록 구성 요소

Assembly Block Components



박진수 교수

서울대학교·경영대학

jinsoo@snu.ac.kr



학습 목차

함수의 기초 개념

함수의 종류

함수의 특성

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

함수 문서화

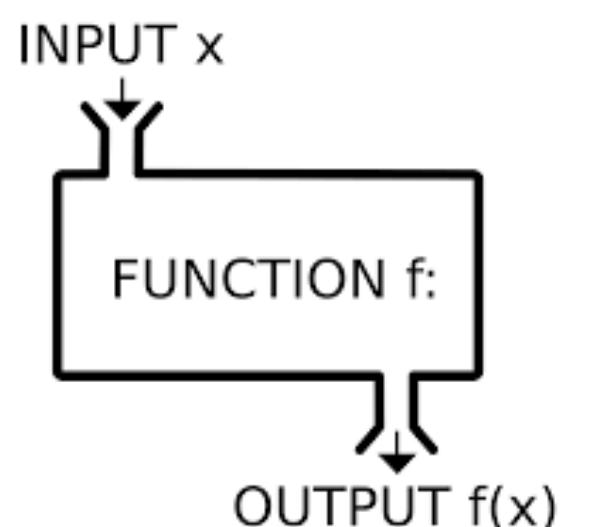
모듈과 패키지

모듈과 패키지 불러오기

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 언제든 불러와서 사용할 수 있기 때문에 코드의 재활용성을 높일 수 있다
- 프로시저(procedure), 서브루틴(subroutine), 메소드(method), 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 **1개**이고 전달인자의 자료형은 **문자열**
 - ✿ **len** 함수가 반환하는 값의 자료형 : **정수**

```
x = len('파이썬')
print(x)
```

3



함수 호출

함수 호출(function call)이란?

- 함수를 불러와서 실행하는 것

함수이름([전달인자-1, ..., 전달인자- N])

변수 = 함수이름([전달인자-1, ..., 전달인자- N])

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

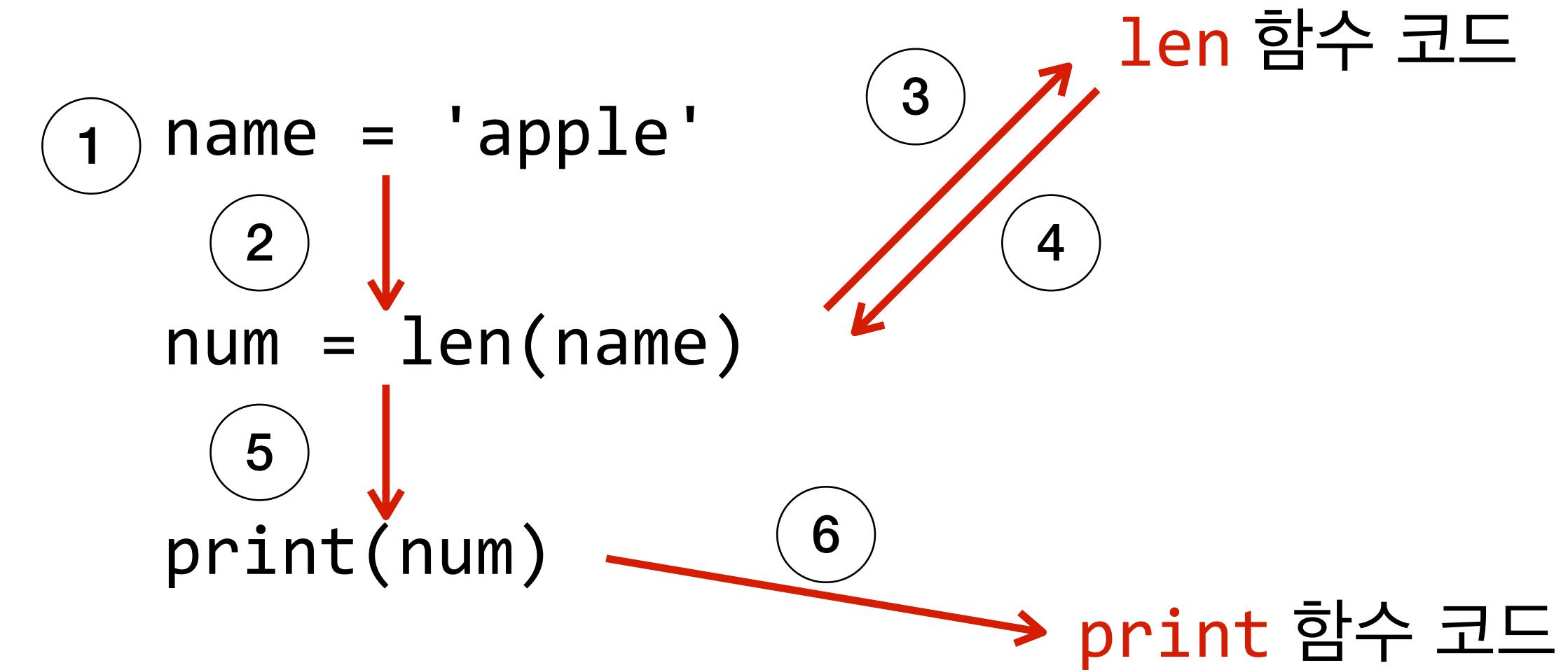
함수

- 전달인자를 받아와서 특정한 작업을 실행한 후
- 반환할 결괏값이 있으면 결괏값을 반환하고 종료하든지, 반환하지 않아도 되면 반환하지 않고 종료한 후
- 호출한 코드로 실행 권한을 넘긴다



예시 : 함수 호출

- 문자열 길이 구하기 코드



- 프로그램 실행 순서

- (1) 문자열 'apple'을 변수 name에 할당
- (2) 변수 num에 len(name)의 결괏값을 받기 위해 len 함수를 호출
- (3) len 함수를 호출하면서 이 함수에 전달인자 name을 넘겨준다
- (4) 결괏값으로 정수 5를 반환 -> 이 값을 변수 num에 할당
- (5) num 값을 출력하기 위해 print 함수를 호출
- (6) print 함수를 호출하면서 전달인자 num을 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 값을 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.



함수 호출 형식

		전달인자	
		전달인자 (X)	전달인자 (O)
반환값	반환값 (X)	함수이름()	함수이름(전달인자1, ..., 전달인자N)
	반환값 (O)	변수 = 함수이름()	변수 = 함수이름(전달인자1, ..., 전달인자N)



● 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='/' )
```

● 내장 함수(built-in functions)란?

- 파이썬은 기본적으로 다양하고 강력한 표준 내장 함수를 제공

- 항상 다음을 기억하자 : “Do not reinvent the wheel!”

- 흔히 사용하는 내장 함수

 - print

 - len

 - range

- 데이터 분석에 흔히 사용하는 내장 함수와 클래스

 - type

 - range

 - enumerate

 - len

 - max, min

 - abs

 - int, float, str

 - list, tuple, dict, set

 - sorted

 - open

```
# 흔히 사용하는 내장함수 활용 예시
```

```
print('How are you?')
```

```
# print
```

```
L = [1, 2, 3, 4, 5]
```

```
len(L)
```

```
# len
```

```
for i in range(0, len(L), 2): # range
    print(L[i])
```



사용자 함수(custom function)란? 사용자(프로그래머)가 직접 만든 함수

```
def 함수이름([매개변수, ...]):  
    함수-명령문  
    [return 객체]
```

- ❶ **함수이름**은 유효한 식별자여야 한다(4장의 102p 식별자 구성 규칙 참고)
- ❷ 함수에 포함할 수 있는 **매개변수**의 개수는 제한이 없다
 - ❸ 즉, 0개부터 **n**개의 **매개변수**를 선언할 수 있다
 - ❹ 따라서, **매개변수**는 선택 사항
- ❺ 함수 정의 부분(첫 번째 줄)의 마지막에 쌍점(:)이 온다
- ❻ **함수-명령문** 블록은 들여쓰기로 구분
- ❼ **return** 문은 선택 사항
 - ❽ **함수-명령문** 블록을 실행한 후 결괏값을
 - * **return** 문을 사용해 **객체**를 반환할 수도 있고,
 - * 반환 값이 없으면 **return** 문을 생략할 수 있다 -> **None** 을 반환



매개변수 vs. 전달인자

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

전달인자(argument) : 함수를 호출할 때 넣어주는 값

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

```
sayAnything('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

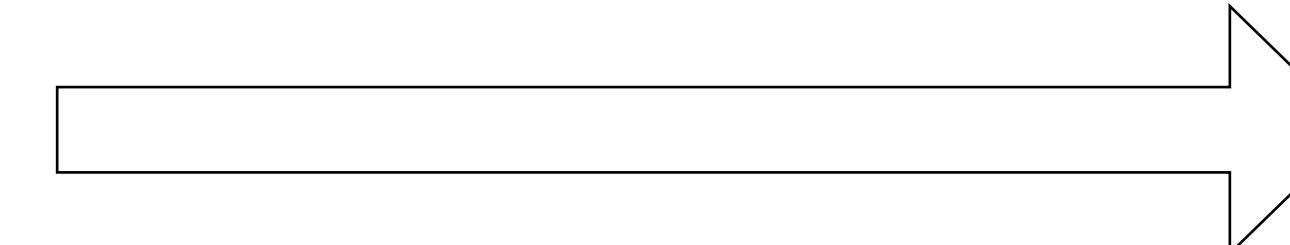
```
def hi():
    print('안녕하세요 교수님!')

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
없을까?





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 변경해주면 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



코드 실행 순서

```
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, 4 5 과정 반복
- 7 person 함수 호출, 전달인자로 'Amy' 전달, 4 5 과정 반복

예시 : 사용자 함수 정의

```
# 매개변수와 반환 값이 없는 함수 정의  
def my_function1():  
    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_function4(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

◎ 예) 계승 함수(factorial function)

▶ 음이 아닌 정수 n 의 계승은 $n!$ 로 표현하며 계승은 1부터 n 까지의 정수를 모두 곱한 것을 의미

○ 예) $5! = 5 * 4 * 3 * 2 * 1 = 120$

▶ $0!$ 의 값은 1

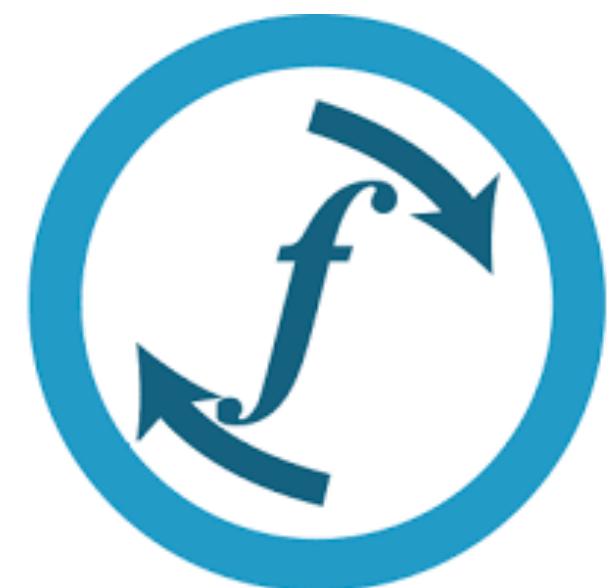
```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

```
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

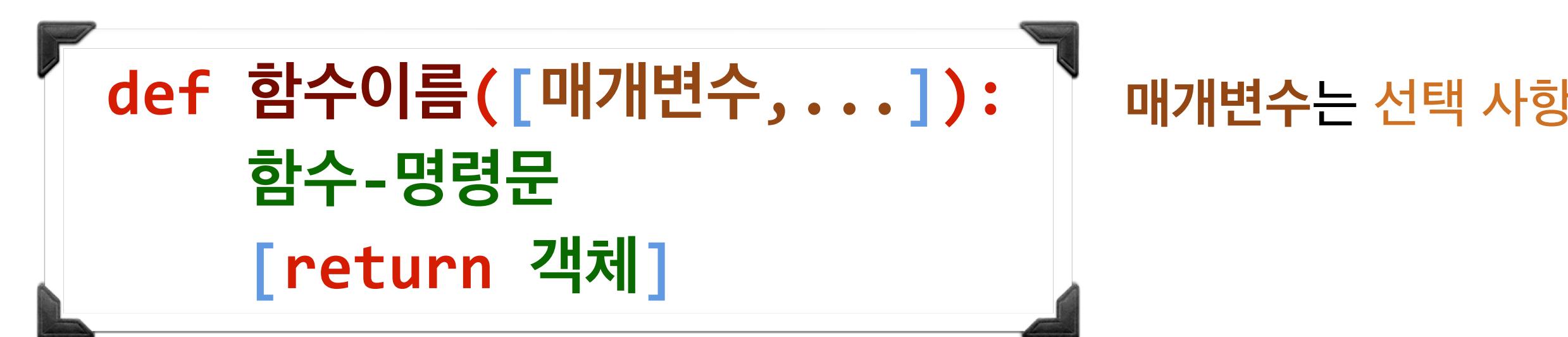
● 메소드(method)

- 클래스 내부에서 구현한 함수
- 특정 자료형, 즉, 클래스와 관련된 함수이기 때문에 해당 클래스의 인스턴스(또는 해당 클래스)와 함께 사용해야 한다
 - 예) 문자열 `.format()`



(사용자 정의) 전역 함수(global function)란?

- 가장 흔히 사용하는 일반 함수
- 같은 모듈, 즉 같은 파이썬 파일(.py) 안 어디서든 사용할 수 있는 함수
- 다른 모듈에서도 사용 할 수 있는데, 불러오기(**import**)한 후 사용하면 된다



print나 **len** 등은 파이썬 내장 전역 함수(built-in global function)

- 파이썬이 자체적으로 제공하는 내장 함수는 같은 파일 안에 없어도 불러오기(**import**)를 하지 않고 파이썬 프로그램 어디서나 사용할 수 있다



예시 : 전역 함수

매개변수(X) 반환값(X)

```
def say_hello():
    print('안녕~~~!')
```

```
say_hello()
```

안녕~~~!

매개변수(O) 반환값(O)

```
def rectangle_area(width, height):
    return width * height
```

```
x = rectangle_area(2, 5)
print(x)
```

```
y = rectangle_area(7.5, 10)
print(y)
```

10

75.0

매개변수(O) 반환값(X)

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

```
say_anything('파이썬')
```

안녕 파이썬



지역 함수(local function)란?

- 어떤 함수 안에 정의한 함수로서 내재 함수라고도 한다
- 지역 함수를 정의한 함수 안에서만 사용 가능
- 지역 함수를 정의한 함수 외 다른 함수나 프로그램에서 호출하여 사용할 수 없다
- 주로 특정 함수 안에서 필요한 기능을 여러 번 불러 사용해야 하는 경우 등 도우미 함수로 가끔 사용

```
전역 함수 → def display_stand(product):  
지역 함수 →     def display_num():  
                   import random  
                   return random.randint(1, 99)  
               print(f'{product} : {display_num()}번 진열대에 배치하세요. ')
```

```
display_stand('연필') ← 전역 함수 호출
```

```
display_stand('지우개') ← 전역 함수 호출
```

```
display_num() ← 지역 함수 호출
```

메소드(method)란?

- 클래스 내부에서 구현한 함수
- 클래스와 관련있는 함수이기 때문에 해당 클래스와 함께 사용해야 한다
- 대표적인 예로는 문자열과 함께 이미 자주 사용한 문자열.**format** 메소드가 있다

```
class myclass:  
    def say_anything(self, anything):  
        print('안녕', anything)  
  
myinstance = myclass()  
myinstance.say_anything('파이썬')
```

- 모든 메소드는 함수지만 모든 함수가 메소드인 것은 아니다
- 메소드는 클래스에 속해 있는 함수만을 지칭한다
 - ▶ 함수는 함수 자체만을 호출하여 사용할 수 있지만
 - ▶ 메소드를 사용하려면 항상 클래스(또는 클래스의 인스턴스)를 명시해줘야 한다



람다 함수(lambda function)란?

- 함수를 사용하는 시점에 표현식 형태로 정의하여 바로 사용하는 함수
- 일반적인 함수보다 구현할 수 있는 기능이 제한적이지만 다른 장점이 있다

lambda [매개변수]: 표현식

- 매개변수를 설정하면 일반적으로 위치 매개변수 형태며 한 개 이상 사용 가능
 - 매개변수는 선택 사항
- 표현식은 계산한 값을 반환
- 람다 함수의 표현식은 몇 가지 제약이 있다
 - 표현식에 `return` 문 또는 `yield` 문을 사용할 수 없다
 - 조건문이나 순환문을 사용할 수 없다
 - * 단, 간편 조건문은 사용할 수 있다
 - 표현식이 튜플이면 표현식을 반드시 괄호 안에 넣어야 한다
- 람다 함수는 함수 이름이 없기 때문에 '익명 함수(anonymous function)'라고도 한다



예시 : 람다 함수

```
def sayAnything(anything):  
    print('안녕', anything)  
sayAnything('파이썬')
```

안녕 파이썬

sayAnything 함수를
람다 함수로 재정의해보자

```
talk = lambda anything: print('안녕', anything)  
talk('파이썬')  
talk('초콜릿')
```

익명 함수인 람수 함수를 변수에 할당하는 이유는 뭘까?

```
x = print  
type(x)  
x(f'2 * 3 = {2 * 3}')
```



예시 : 람다 함수

```
c = lambda i: '' if i == 1 else 's'  
for i in range(10):  
    print(f'{i} file{c(i)} processed.')
```



람다 함수를 전달인자로 사용

람다 함수는 **sorted** 함수나 **L.sort** 메소드 등에서 매개변수 **key**의 전달인자로 사용하듯이 주로 함수의 전달인자로 사용

```
pair = [(2, 'a', 'Pineapple'), (1, 'b', 'Orange'), (1, 'a', 'Banana'), (1, 'b', 'grape'), (2, 'a', 'apple')]
```

```
pair.sort()
print(pair)
```

```
pair.sort(reverse=True)
print(pair)
```

```
pair.sort(key=lambda e: (e[1], e[2]))
print(pair)
```

```
pair.sort(key=lambda e: e[1:3], reverse=True) # 위와 같은 표현식을 분할 연산자를 사용해 내림차순으로 정렬
print(pair)
```

```
pair.sort(key=lambda e: (e[1], e[2].lower())) # 대소문자 구분없이 소문자로 정렬
print(pair)
```

함수의 특성

Function Properties





값 반환 함수와 보이드 함수



Value-Returning Functions vs. Void Functions





- 함수의 명령문을 실행한 후 함수를 호출한 코드에 결괏값을 **return**문을 통해 돌려준다

값 반환 함수(value-returning function)란?

- 따라서, 호출한 코드는 함수가 반환하는 결괏값을 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 결괏값을 변수에 저장(할당)하지 않으면 어떻게 될까?
len('파이썬')



예시 : 값 반환 함수

```
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 결괏값을 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 결괏값을 받아보자
x = print('안녕 파이썬')
```

```
print(x)
```

```
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)
```



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)
```



값 반환 외 return 문의 용도

함수 빠져나가기

return 문은 값을 반환하는 역할 외 특정 조건을 만족하거나 만족하지 않으면 즉시 함수를 빠져나갈 때도 사용할 수 있다

```
def positive_product(i, j):
    """전달인자 둘 중 하나라도 음수면 함수를 빠져나간다."""
    if i < 0 or j < 0:
        return
    print(f'{i} x {j} = {i * j}')

positive_product(-5, 7)

positive_product(5, -7)

positive_product(5, 7)
```



매개변수와 전달인자



Parameters and Arguments





● 매개변수(parameter)란?

- ▣ 함수를 정의할 때 괄호 안에 선언한 식별자(변수)며,
- ▣ 전달인자를 함수 내부로 가져와 처리할 목적으로 사용

● 매개변수 구분

▣ 필수 매개변수

- ▣ 기본값이 설정되지 않은 매개변수
- ▣ 기본값이 미리 설정되어 있지 않기 때문에 함수를 실행(호출)할 때 반드시 전달인자를 지정해야 하며, 지정하지 않으면 오류가 발생
- ▣ 예) `myfunction(Length, weight)`

▣ 선택 매개변수

- ▣ 기본값이 미리 설정되어 있는 매개변수
- ▣ 기본값이 미리 설정되어 있기 때문에 함수를 실행(호출)할 때 전달인자를 지정하지 않으면 해당 함수가 기본값으로 명령문을 실행
- ▣ 따라서, 기본값이 아닌 다른 값을 사용하고자 할 때만 지정하면 된다
- ▣ 예) `myfunction(copies=1, color=None)`



함수를 정의할 때 매개변수는 크게 세 가지 종류로 나눌 수 있다

매개변수가 없다

- 매개변수가 0개(즉, 전달인자 없이 함수의 기능을 실행)
- 예) `def myfunction():`

위치 매개변수(positional parameter)

- 쉼표로 구분하는 한 개 이상의 식별자로 정의하는 매개변수
- 예) `def myfunction(Length, weight):`

키워드 매개변수(keyword parameter)

- 쉼표로 구분하는 한 개 이상의 식별자=기본값 형태의 쌍으로 정의하는 매개변수
- 예) `def myfunction(copies=1, color=None):`

앞서 전달인자를 설명할 때 위치 전달인자가 키워드 전달인자보다 항상 먼저 와야 한다고 했다

따라서 함수를 정의할 때도 모든 위치 매개변수(필수 매개변수)를 키워드 매개변수(선택 매개변수)보다 먼저 선언해야 한다

`def ok(x, y, z=1): (O)`

`def bad(x, y=1, z): (X)`



● 전달인자(argument)란?

- 함수(또는 메소드)를 호출할 때 함수(또는 메소드)의 이름 바로 다음에 오는 소괄호(()) 안에 할당해서 함수 안으로 전달하는 값
- 대부분의 파이썬 함수(또는 메소드)는 값을 전달인자로 받아서 작업을 수행

● 전달인자 구분

● 필수 전달인자

- 반드시 값을 전달해야 하며 그렇지 않으면 오류가 난다

● 선택 전달인자

- 선택 사항이라 값을 전달하지 않아도 되며 함수(메소드) 정의 문서에서 일반적으로 대괄호([])로 표시
- 값을 전달하지 않으면 기본값을 적용

`range([시작,] 끝[, 폭])`

- 시작 : 순서 열의 시작 번호(선택 전달인자 : 값을 전달하지 않으면 기본값인 0이 주어진다)
- 끝 : 순서 열의 최대 값으로 반드시 값을 전달해야 한다(필수 전달인자 : 값을 전달하지 않으면 TypeError 가 난다)
- 폭 : 순서 열에서 서로 붙어 있는 번호 간의 간격(선택 전달인자 : 값을 전달하지 않으면 기본값인 1이 주어진다)



예시 : 전달인자

range(끝)

끝은 필수 전달인자

```
range()
```

float([x])

x는 선택 전달인자

```
d1 = float()  
print(d1)
```

```
d2 = float('12')  
print(d2)
```



전달인자는 형식에 따라 '위치 전달인자'와 '키워드 전달인자' 두 종류로 구분

위치 전달인자(positional argument)

- 함수의 괄호 안에 쉼표로 구분하는 한 개 또는 여러 개의 값을 말한다
- 예) `print_setup('A4', 1, False)`

키워드 전달인자(keyword argument)

- 앞에 키워드 식별자가 오는 전달인자를 말한다
- 예) `print_setup(paper='A4', copies=1, color=False)`



필요한 전달인자보다 적거나 많은 전달인자를 사용하면 `TypeError`가 발생

`len(s)`

`len()`

```
a = [1, 2, 3]
b = [4, 5, 6]
len(a, b)
```



- 위치 전달인자가 키워드 전달인자보다 항상 먼저 와야한다

주의 (2)

- 그렇지 않으면 **SyntaxError**가 발생
- 따라서 함수를 정의할 때도 모든 위치 매개변수(필수 매개변수)가 어떤 키워드 매개변수(선택 매개변수)보다도 앞에 선언해야 한다

```
def ok(x, y, z=1): (O)
```

```
def bad(x, y=1, z): (X)
```

```
L = ['c', 'b', 'd', 'a']
```

```
sorted(L, reverse=True)
```

```
sorted(reverse=True, L)
```



예시 : 위치 전달인자

```
def string_format(text, length=10, fills=' '):
    if len(text) < length:
        text = text + fills * (length - len(text))
    return text
s = '파이썬과 빅데이터 분석'                      # s의 길이는 12
string_format(s)                                    # 필수 전달인자만 사용
string_format(s, 15)                                # length에 15
string_format(s, 5, '*')                            # length에 5, fills에 '*'(s보다 length가 짧다)
string_format(s, 15, '!')                            # length에 15, fills에 '!'
string_format(s, 25, '~')                           # length에 25, fills에 '~'
string_format(s, '~')                              # 두 번째 위치 전달인자로 문자열이 왔다
string_format(s, length=25, '~') # 키워드 전달인자가 위치 전달인자 앞에 있다
```



예시 : 키워드 전달인자

키워드 전달인자를 사용하면 전달인자의 순서는 중요하지 않다

```
def string_format(text, length=10, fills=' '):
    if len(text) < length:
        text = text + fills * (length - len(text))
    return text
s = '파이썬과 빅데이터 분석'
# s의 길이는 12
```

string_format(s, fills='\$', length=15)	# fills와 length의 순서를 바꿔 값을 전달
string_format(text=s)	# 키워드 전달인자를 필수 매개변수에도 사용 가능(나머지는 기본값)
string_format(fills='@', text=s)	# length는 기본값 10 사용
string_format(length=1, text=s)	# fills는 기본값 ' ' 사용
string_format(fills='%!', text=s, length=20)	# 모든 값에 키워드 전달인자를 사용
string_format(s, length=15, '\$')	# 키워드 전달인자가 위치 전달인자 앞에 있다
string_format(s, length=15, fills='\$')	# 키워드 전달인자가 위치 전달인자 앞에 있다
string_format(text=s, 15, '\$')	



* 연산자와 전달인자의 위치

함수 정의 부분에 * 를 선언하면 전달인자들의 위치를 지정하는 역할을 한다

- * 뒤에는 위치 전달인자가 올 수 없고 키워드 전달인자만 올 수 있다

자주 사용하는 **sorted()** 함수를 살펴보자 **sorted(iterable, *, key=None, reverse=False)**

```
L = ['a', 'd', 'C', 'E', 'b']
sorted(L)
sorted(L, key=None, reverse=False) # 키워드 전달인자로 기본값을 전달
sorted(L, None, False)           # 위치 전달인자로 기본값을 전달
```



예시 : * 연산자와 전달인자의 위치

```
def product_in_units(i, j, k, *, units='square meters'):
    return f'{i * j * k} {units}'  
  
product_in_units(2, 3, 4)
product_in_units(2, 3, 4, units='square inches')
product_in_units(2, 3, 4, 'square inches')
```



예시 : * 연산자와 전달인자의 위치

*가 매개변수 처음에 위치한다면 어떻게 될까?

```
def print_setup(*, paper='A4', copies=1, color=False):
    print(f"프린터 설정: paper='{paper}', copies={copies}, color={color}")

print_setup()
print_setup(paper='Letter', color=True)
print_setup(copies=5, paper='Letter', color=True)
print_setup('Letter', 5, True)
print_setup('Letter')
```



지금까지의 예는 함수를 호출할 때 전달인자의 개수가 정해져 있었다

그런데 우리가 잘 알고 있는 **print** 함수는 불특정 다수의 전달인자를 입력해도 처리가 가능

```
print()  
print(1)  
print('a', 'b', 'c')  
print(1, 2, 3, 'a', '가')
```

```
1  
a b c  
1 2 3 a 가
```



print 함수처럼 임의의 전달인자를 처리할 수 있는 함수는 어떻게 만들 수 있을까?



먼저 **print** 함수의 정의를 살펴보자

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

이 함수 정의에서 첫 번째 매개변수를 보면 *objects* 앞에 * 연산자가 있음을 알 수 있다



시퀀스형 패킹 연산자 *

함수를 정의할 때 * 연산자를 매개변수 앞에 두면 시퀀스형 패킹 연산자다

- * 연산자를 사용하면 콤마로 구분하는 튜플 형태의 불특정 다수 위치 전달인자를 하나의 필수 매개변수로 패킹해서 받아 처리하는 함수를 만들 수 있다
 - 따라서, 시퀀스형 패킹 연산자 * 는 몇 개의 위치 전달인자를 넘겨 받을지 모르는 함수를 정의할 때 유용하다 **Very flexible!!!**
- ☞ * 연산자를 변수 이름 바로 앞에 붙여 사용하면 시퀀스형 객체의 패킹 연산자라고 이미 6장(튜플)에서 다루었다

```
def product(*args):
    """임의의 숫자를 전달받은 후 모두 곱한 결괏값을 반환하는 함수"""
    result = 1
    for arg in args:
        result *= arg
    return result

x = product(15)
print(x)
x = product(2, 3, 4)
print(x)
x = product(5, 6, 7, 8)
print(x)
```

```
def sum_of_powers(*args, power=1):
    """임의의 숫자를 전달받은 후 그 숫자를 지정한 수의 거듭제곱으로
    계산한 값들을 모두 합해서 반환하는 함수"""
    result = 0
    for arg in args:
        result += arg ** power
    return result

x = sum_of_powers(1, 3, 5)
print(x)
x = sum_of_powers(2, 3, power=2)
print(x)
x = sum_of_powers(1, 3, 5, 7, power=3)
print(x)
```



시퀀스형 언패킹 연산자 *

함수의 전달인자 앞에 *를 붙이면 시퀀스형 언패킹 연산자가 된다

- 이를 '위치 전달인자 언패킹(positional argument unpacking)'이라고 한다
- * 패킹은 리스트로 할당을 하지만, * 언패킹은 튜플 형태로 할당되었기 때문에 '가변길이 전달인자 튜플(variable-length argument tuples)'이라고도 한다

특징은 다음과 같다

- 전달인자가 두 개 이상이면 * 연산자의 위치는 어디든 상관없다
- 단, * 연산자가 앞에 붙어 있는 전달인자가 언패킹되었을 때 총 객체 개수는 반드시 해당 함수의 전달인자 개수와 일치해야 한다

```
def add5(i, j, k, l, m):
    """5개의 전달인자를 받아 모두 합한 결괏값을 반환하는 함수"""
    return i + j + k + l + m

x = add5(1, 2, 3, 4, 5)          # 위치 전달인자가 다섯 개
print(x)
```

```
args = [1, 2, 3, 4, 5]
x = add5(*args)                  # args의 다섯 개 모두 언패킹
print(x)
x = add5(1, *args[:3], 1)        # arg의 첫 세 개만 언패킹
print(x)
x = add5(*args[-2:], 1, 2, 3)   # arg의 마지막 두 개만 언패킹
print(x)
```



매핑형 패킹 연산자 **

함수를 정의할 때 ****** 연산자를 매개변수 앞에 두면 **매핑형 패킹 연산자**다

- ****** 연산자를 사용하면 콤마로 구분하는 튜플 형태의 불특정 다수 키워드 전달인자를 하나의 선택 매개변수로 패킹해서 받아 처리하는 함수를 만들 수 있다
- 따라서, 매핑형 패킹 연산자 ******는 몇 개의 키워드 전달인자를 넘겨 받을지 모르는 함수를 정의할 때 유용하다

Very flexible!!!

```
def add_fruit_detail(fruit_name, date_produced, **kwargs):
    """위치 전달인자 두 개와 0개 이상의 불특정 다수의 매개변수를 키워드 전달인자로 받아 처리하는 함수"""
    print('과일종류 =', fruit_name)
    print('생산일자 =', date_produced)
    for key in sorted(kwargs): # 키워드 전달인자의 키 값으로 오름차순 정렬해서 출력
        print(f'{key} = {kwargs[key]}')

add_fruit_detail('블루베리', '9일')
add_fruit_detail('바나나', '11일', 원산지='필리핀')
add_fruit_detail('오렌지', '11일', 유통기간='60일', 원산지='제주도', 수량=55)
```



매핑형 언패킹 연산자 **

함수의 전달인자 앞에 ******를 붙이면 **매핑형 언패킹 연산자**가 된다

앞서 정의한 `print_setup` 함수를 다시 사용하자

```
def print_setup(*, paper='A4', copies=1, color=False):
    print(f"프린터 설정: paper='{paper}', copies={copies}, color={color}")
```

****** 를 매핑형 언패킹 연산자로 사용하려면, 딕셔너리 형태의 객체를 참조하는 변수를 하나의 전달인자로 받아 언패킹해서 사용할 수 있다

```
opt1 = dict(copies=35, color=True, paper='Letter')
print_setup(**opt1)
```

일부 전달인자의 값만 지정하고 나머지는 기본값을 사용하면 변경할 값만 딕셔너리로 만들어 전달할 수도 있다

```
opt2 = {'copies': 5}
print_setup(**opt2)
```



패킹 연산자로 함수 정의

* 와 ** 연산자를 매개변수로 함께 사용하여 임의의 위치 전달인자와 임의의 키워드 전달인자를 받아서 처리하는 함수도 만들 수 있다

```
def print_args(*args, **kwargs):
    """먼저 위치 전달인자를 순서대로 출력하고,
    그 다음에 키워드 전달인자를 순서대로 출력하는 함수다."""
    for i, arg in enumerate(args):
        print(f'위치 전달인자 {i} = {arg}')
    for key in kwargs:
        print(f'키워드 전달인자 {key} = {kwargs[key]}')


print_args('라이언', -7, 2.5, age=19, email='ryan@korea.kr')
print_args('포도', 유통기한='30일', 원산지='김해', 수량=100)
```



print() 함수 : revisited

- 작성 방법

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

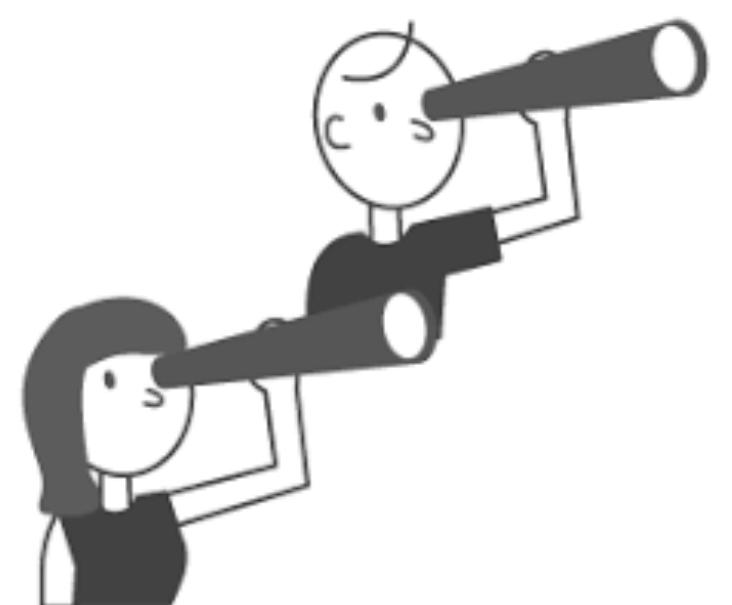
- 다수의 위치 전달인자(**objects*)를 받을 수 있고 4개의 키워드 전달인자를 받을 수 있음
 - 각 키워드 전달인자는 기본값이 있음
- sep** 매개변수의 기본값은 공백(' ')
 - 두 개 이상의 위치 전달인자가 주어질 경우 각각의 전달인자는 **sep** 값으로 나뉘어 출력됨
 - 위치 전달인자가 하나만 주어졌을 경우 **sep** 매개변수는 아무런 역할도 하지 않음
- end** 매개변수의 기본값은 새줄바꿈(newline)
 - 위치 전달인자들이 출력되고 난 후 마지막에 **end** 매개변수의 값이 출력됨
- file** 매개변수의 기본값은 표준 출력 스트림(standard output stream)
 - 표준 출력 스트림은 주로 콘솔(console)을 사용
- flush** 매개변수는 파이썬 3.3부터 사용 가능
 - 값이 **True**이면 **file** 스트림이 강제적으로 내보내짐(flush)



범위와 가시성



Scope and Visibility



범위(scope)란?

- 변수나 함수의 가시성을 나타낸다
- 범위를 벗어난 변수나 함수는 다른 코드에서 사용할 수 없기 때문에 호출할 수도 없다

- 모든 변수와 함수는 만들어진 위치에 따라 범위가 정해진다
- 프로그램 최상위 레벨에서 변수를 정의하면 모든 함수에서 접근할 수 있는데, 이런 변수를 **전역 변수**라고 한다
- 반면에 함수 안에 정의한 변수인 **지역 변수**는 그 함수 외 다른 함수에서 사용할 수 없도록 범위가 제한되어 있다

전역 변수(global variable)

- 다른 함수를 포함해서 프로그램 전체에서 사용 가능

```
x = '나는 전역 변수 x다.'      # 전역 변수를 선언
def myfn1():
    y = '나는 지역 변수 y다.'  # 지역 변수를 선언
    print(y)                  # 지역 변수를 출력 <- in scope
    print(x)                  # 전역 변수를 출력 <- in scope
```

지역 변수(local variable)

- 해당 함수에서만 사용 가능하며 타 함수에서는 사용할 수 없다
- 지역 변수란 특정 함수 내에서 생성한 변수를 지칭한다

```
myfn1()
print(x)                      # 전역 변수에 접근
print(y)                      # 지역 변수에 접근
```

가시성(visibility)란? 변수나 함수를 프로그램의 어느 부분에서 볼 수 있는지, 즉 변수나 함수에 합법적으로 접근할 수 있는 프로그램 영역

- 가시성은 어떤 변수나 함수를 호출해서 사용할 수 있는지 여부를 결정한다
- 가시성이 없으면 그 변수나 함수를 사용할 수 없다

범위와 가시성은 주로 일치하지만,

- 같은 범위 안에 같은 이름의 변수나 함수가 함께 있다면 이 중 하나를 일시적으로 볼 수 없는 상황이 존재할 수도 있다
- 즉, 같은 범위에 있는 변수나 함수라도 가시성이 없을 수 있다
- 예를 들어, 같은 이름의 전역 변수와 지역 변수가 같은 범위(e.g., 함수 안)에 있을 때, 전역 변수의 가시성은 해당 범위에서 일시적으로 사라진다

```
x = '나는 전역 변수 x다.' # 전역 변수 선언
def myfn2():
    x = 'Hi Python'      # 같은 이름의 지역 변수 선언
    print(x)

myfn2()
print(x)
```



만약 함수 안에서 전역 변수를 직접 조작해
이 함수의 범위 바깥에도 영향을 주려면
어떻게 해야 할까?



전역 변수에 접근하기

```
x = '나는 전역 변수 x다.'  
def myfn3():  
    x = 'Hi Python'  
    print(x)  
  
myfn3()  
print(x)
```

```
x = '나는 전역 변수 x다.'  
def myfn3():  
    global x  
    x = 'Hi Python'  
    print(x)  
  
myfn3()  
print(x)
```



전역 변수의 값이 **가변형**(mutable)(예, 리스트, 딕셔너리 등)이면 **global**을 선언하지 않고도 변경 가능



nonlocal 문

상위 변수에 접근하기

nonlocal 문을 사용하면 상위 단계의 변수에 접근 가능

```
x = '나는 전역 변수 x다.'
def myfn4():
    x = '나는 지역 변수 x다.'
    def localfn():
        nonlocal x
        x = '코딩을 즐기자.'
        print('2:', x)
    print('1:', x)
    localfn()
    print('3:', x)

myfn4()
print(x)
```

함수 문서화

Docstrings & Naming Convention



- 명명규칙(naming convention)이란?

- 좋은 함수 이름과 매개변수 이름 짓는 법
- 이름이 기능에 미치는 영향은 없으나 명확한 명명체계를 가지고 있으면 이름만으로도 쉽게 변수, 상수, 함수, 클래스 등 구분이 가능

- 이름 짓는 규칙을 하나 정한 후 일관되게 적용

- **상수** 이름은 전체를 영문 대문자 : **UPPERCASE**

- PI = 3.14
 - INTEREST_RATE = 1.618

- **클래스** 및 **예외** 이름의 각 단어의 첫 글자만 영문 대문자 : **TitleCase**

- class MyClass(s): pass

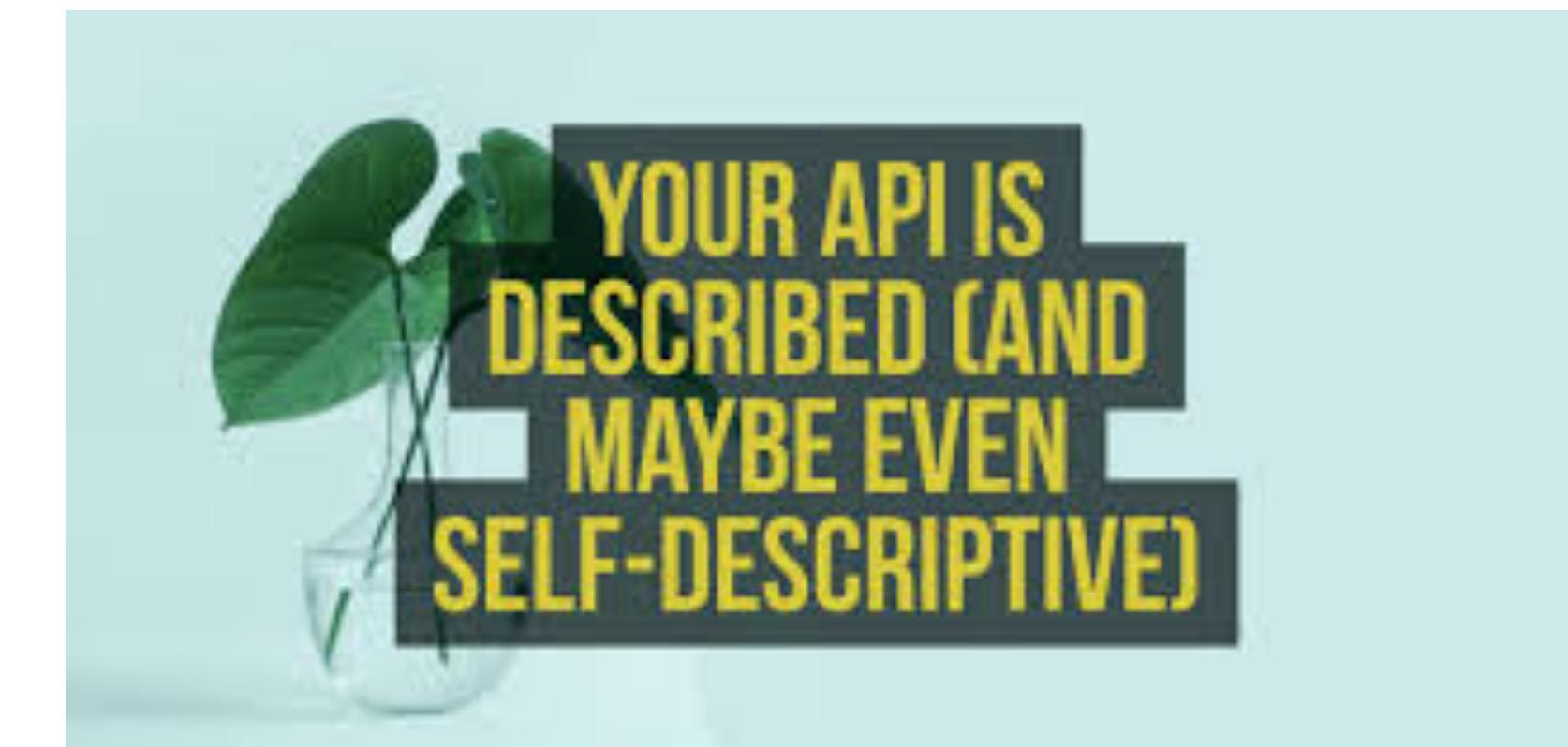
- 이외 나머지는 소문자 또는 밑줄로 구분한 소문자 사용 : **lowercase** 또는 **lower_case**

- 자기 기술적(self-descriptive)

- 함수 이름만으로 해당 함수의 역할을 파악할 수 있어야 한다
 - 어떤 기능을 수행하는지, 결과 값으로 무엇을 반환하는지 등

- 약어 사용 지양

- 보편적으로 사용하는 약어를 제외하고 가급적이면 약어는 사용하지 말 것





설명문자열(docstrings)이란?

- 함수가 하는 기능을 요약해서 문서 형태로 설명한 것
- 설명문자열은 함수 뿐만 아니라 모듈, 패키지, 클래스에서도 같은 방식으로 작성할 수 있다
- 설명문자열을 사용해서 문서를 작성하면 소스 코드를 보지 않고도 그 문서를 확인할 수 있다
 - 주석은 소스 코드를 봐야만 내용을 알 수 있지만, 설명문자열로 작성하면 소스 코드를 보지 않아도 확인할 수 있다
 - 예를 들어, `sorted` 함수의 기능을 알고 싶으면 소스 코드를 보지 않고도 모든 함수(사용자 함수 포함)에 기본적으로 포함되어 있는 특별한 속성인 `__doc__`이나 `help` 함수를 통해 확인할 수 있다

```
print(sorted.__doc__)
```

Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.

```
help(sorted)
```

Help on built-in function sorted in module builtins:

```
sorted(iterable, /, *, key=None, reverse=False)
```

Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.



작성 방법

- 주석과는 달리 # 기호를 사용하지 않고,
- 삼중 따옴표(triple quotes)로 함수나 메소드 이름 바로 아래 첫 부분에 작성
- 즉, def 명령문 바로 아래 줄부터 작성

```
def string_format(text, length=10, fills=' '):
    """Returns text or a truncated copy with the indicator added.

    Args:
        text (str): any string
        length (int): maximum length of the text with indicator
        fills (str): indicator added at the end of shortened text

    Returns:
        str: truncated text with the indicator
    """
    if len(text) < length:
        text = text + fills * (length - len(text))
    return text
```



마찬가지로 `__doc__` 이나 `help` 함수를 통해 설명문자열의 내용을 확인할 수 있다

```
print(string_format.__doc__)
```

Returns text or a truncated copy with the indicator added.

Args:

- text (str): any string
- length (int): maximum length of the text with indicator
- fills (str): indicator added at the end of shortened text

Returns:

- str: truncated text with the indicator

```
help(string_format)
```

Help on function `string_format` in module `__main__`:

```
string_format(text, length=10, fills=' ')
```

Returns text or a truncated copy with the indicator added.

Args:

- text (str): any string
- length (int): maximum length of the text with indicator
- fills (str): indicator added at the end of shortened text

Returns:

- str: truncated text with the indicator



PEP 8 : 파이썬 코드 작성 스타일 가이드

- <https://www.python.org/dev/peps/pep-0008/>
- 들여쓰기는 공백문자 4개
 - 들여쓰기는 탭(tab) 대신 공백문자 4개(4-space indentation) 사용을 권장
- 한 줄은 최대 79자
 - 한 줄은 최대 79자를 넘지 않도록 함
 - 사용자가 한 눈에 보기 용이함
- 빈 줄 넣기
 - 클래스와 함수, 그리고 함수 안의 큰 코드 뭉치들끼리는 빈 줄을 넣어 구분함
- 줄마다 주석 달기
 - 가급적이면 줄마다 주석을 달아 코드에 관한 설명을 넣을 것
- 설명문자열(docstrings) 사용하기



PEP 8 : 파이썬 코드 작성 스타일 가이드

● 공백 넣기

- 연산자 앞뒤와 쉼표 뒤에는 공백을 넣을 것
- 괄호 안의 처음과 마지막에는 공백을 넣지 않고 괄호와 바로 붙여서 작성함
- 예) `a = f(1, 2) + g(3, 4)`

● 함수과 클래스 이름

- 함수와 클래스 이름은 일관되게 정함(앞 ‘명명규칙’ 슬라이드 참고)

● 인코딩

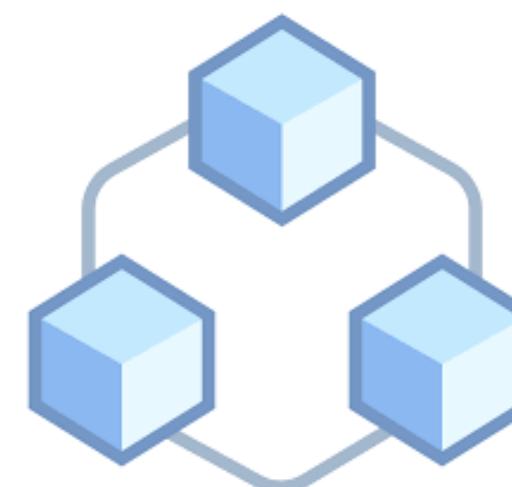
- 필요 이상으로 복잡하고 화려한 인코딩 방식을 사용하지 말 것
- UTF-8이 가장 좋은 인코딩 방식

● 식별자

- 식별자로는 ASCII 문자만 사용
- 즉, 영문자만 사용하기를 권장

모듈과 패키지

Modules and Packages

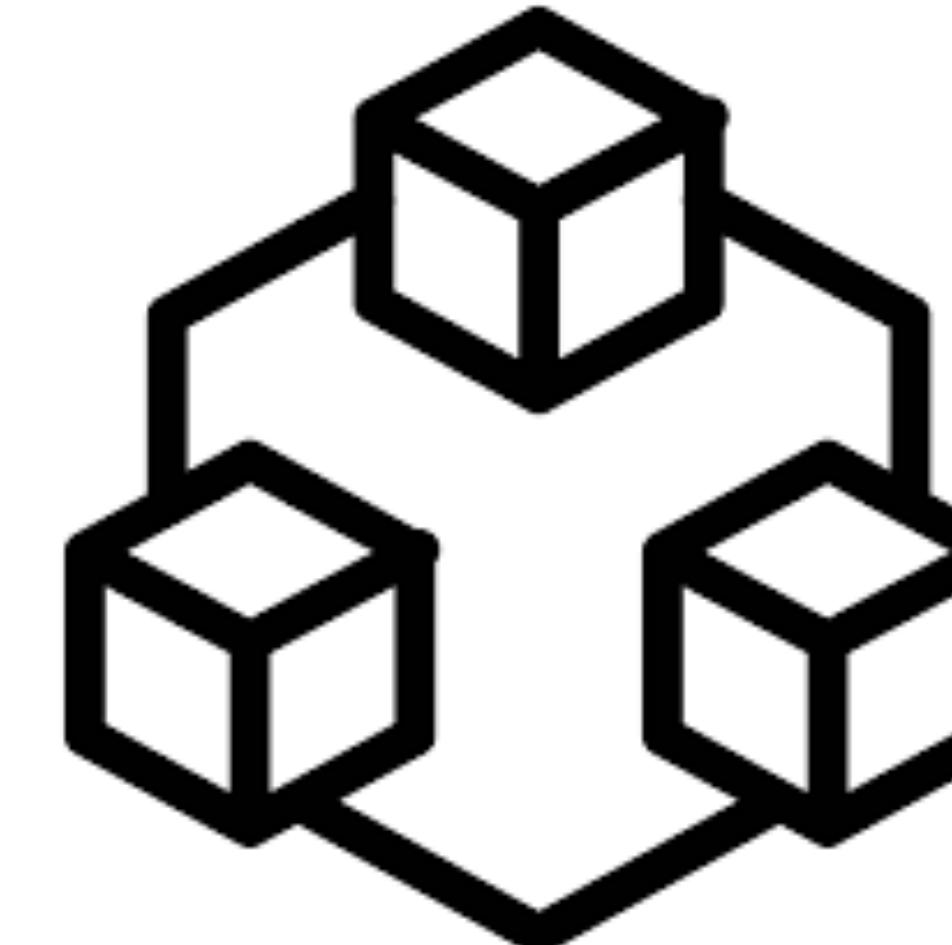




모듈화의 개념

● 모듈화(modularization)란?

- 하나의 프로그램을 여러 개의 작은 단위로 나누는 것을 뜻한다
- **모듈**(modules) : 하나의 프로그램을 구성하는 단위
 - 하나의 프로그램을 여러 개의 작은 단위로 쪼개거나
 - 이를 분할정복(divide and conquer)이라고 한다
 - 자주 사용하는 코드를 뽑아 하나의 프로그램으로 저장한 후 다른 여러 프로그램에서 불러와 사용
 - 이 경우 모듈은 주로 여러 함수들의 집합
 - 예) `import numpy` # numpy 모듈을 불러옴
- ...





모듈화의 장점

● 코드 중복 방지

- 불필요한 코드의 중복을 막고 코드를 단순화
- 코드를 한번만 작성하면 이후 여러 프로그램에서 필요할 때마다 마음껏 가져다 쓸 수 있다
- 코드의 수정과 업데이트가 매우 쉽고 빠르다

● 추상화

- 추상화가 가능
 - **추상화(abstraction)** : 불필요한 세부 내용은 무시하고 중요한 핵심 내용에만 초점을 맞추는 것
 - 예) '밥을 하다'는 추상화된 작업이라 할 수 있으며, '쌀통에서 쌀을 꺼내다', '물을 붓고 쌀을 씻다', '밥 물을 맞춘다', '전기 밥솥에 넣고 취사 버튼을 누른다' 등의 세부 과정을 굳이 알 필요가 없다
- 모든 컴퓨터 프로그램은 일정 수준의 추상화가 되어 있다

● 재사용

- 재사용(reuse)이 가능해 같은 프로그램 내에서 여러 번 사용할 수 있다
- 불러오기(**import**)를 통해 다른 프로그램에서도 사용 가능
- 하나의 프로그램을 개발할 때 각자 맡은 모듈만 작성하면 되므로 여러 명이 동시에 하나의 프로그램을 개발할 수 있다



● 모듈(module)이란?

- 서로 연관된 함수나 클래스를 담고 있는 `.py`라는 확장자를 가진 파이썬 파일
- 그러나...
 - 프로그램 역시 `py`라는 확장자를 파이썬 파일에 담겨져 있는 형태

● 모듈과 프로그램의 가장 중요한 차이점

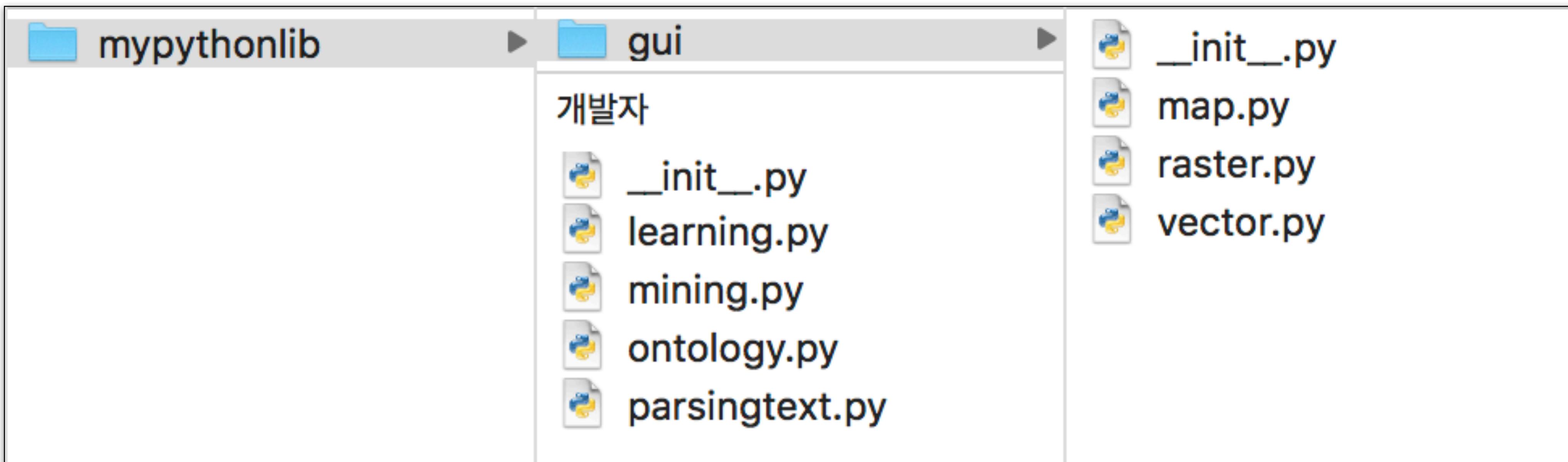
- **프로그램** : 어떤 문제를 해결하기 위해 파이썬 인터프리터 모드에서 특정 작업을 실행하는 파일
- **모듈** : 프로그램이 특정 작업을 실행할 때 필요한 함수나 클래스 등을 모아 놓은 파일
 - 따라서 프로그램이 모듈을 사용하려면 먼저 `import`나 `from ... import` 문으로 불러와야 한다

● 모듈에 접근하기

- 모듈 이름과 함수 이름 또는 속성(변수) 이름을 점(.)으로 구분해서 호출(실행)한다
 - 예) `math.pi`, `math.factorial(x)`
- 이처럼 점 연산자(.)로 접근하는 방법을 '점 표기법(dot notation)'이라 한다

- 패키지(package)란?

- 모듈들을 모아놓은 **디렉토리**(폴더)이며 다음과 같은 아이템을 담고 있다
 - 해당 디렉토리(폴더)에 **`__init__.py`** 파일(파일에 내용이 비어 있어도 됨)이 있다
 - **`__init__.py`** 파일의 내용은 비어 있어도 상관없다
 - 위의 두 조건을 만족하면 디렉토리 이름이 패키지 이름이 된다
 - directory(folder) name = package name





정리 : 모듈 vs. 패키지 vs. 표준 라이버러리

● 모듈(module)이란?

- 서로 연관된 함수들을 담고 있는 파이썬 파일
- 간단하게 생각하면 .py 파일이 모듈
 - 파이썬 프로그램 파일을 인터프리터 모드에서 실행하거나 파이썬 프로그램을 대화형 모드에서 실행할 때 불러오기(**import**)를 해서 사용할 수 있는 또 다른 파이썬 프로그램 파일(.py)

● 패키지(package)란?

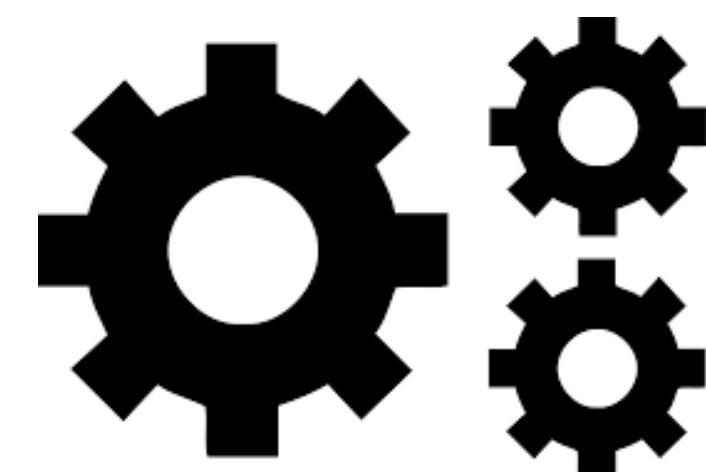
- 관련 모듈들을 체계적으로 모아놓은 디렉토리(폴더)
- 패키지에 속해 있는 모듈에 접근하려면 도트 연산자(.)를 통해 접근해야한다
 - 예) 패키지이름.모듈이름

● 표준 라이버러리(library)

- 내장 함수(built-in functions)와 마찬가지로 파이썬은 기본적으로 유용한 모듈과 패키지를 표준 라이버러리를 통해 제공한다
- 아나콘다(Anaconda)의 경우에는 데이터 분석에 유용한 대부분의 패키지를 기본적으로 제공하고 있다
 - 예) NumPy, Pandas, Scikit-learn, BeautifulSoup, ...

모듈과 패키지 불러오기

Importing Modules and Packages





불러오기 예시

```
import math # 모듈 불러오기  
x = math.exp(10)  
print(x)
```

```
import math as m # 식별자(가명)로 모듈 불러오기  
x = m.exp(10)  
print(x)
```

```
from math import exp # 모듈로부터 함수 불러오기  
x = exp(10)  
print(x)
```

```
from math import * # 모듈의 모든 클래스와 함수 불러오기  
x = exp(10)  
print(x)
```

22026.465794806718
22026.465794806718
22026.465794806718
22026.465794806718



import 문 : 단일 모듈 불러오기

- **X**는 모듈 이름 또는 패키지 안의 모듈 이름이다
 - 예) 모듈 이름 : `import collections`
 - 예) 패키지 안의 모듈 이름(패키지이름.모듈이름) : `import os.path`
 - 패키지 이름만 사용할 수도 있지만 별다른 쓰임새가 없다
- **import** 문을 사용하면 항상 패키지 또는 모듈 이름과 함께 함수나 클래스 이름 전부를 사용하기 때문에 이름이 길어져 번거로울 수 있지만 잠재적인 명칭 오류를 피할 수 있다
 - 명칭 오류(name conflict)란 같은 이름의 모듈이나 함수들 중복 사용해 발생하는 오류를 말한다
- **as** 식별자는 선택 사항
 - 불러오는 **X**에 **식별자**(변수)를 지정해서 별칭으로 사용할 수 있다
 - 이때 **식별자**로 기존의 변수나 모듈 또는 패키지 이름을 사용하면 명칭 오류가 발생할 수 있으니 주의해야 한다
 - 이론상으로는 명칭 오류가 발생할 수 있지만 실제로는 **as** 문을 통해 명칭 오류를 피할 수 있다

```
import math  
math.pi  
math.factorial(5)
```

3.141592653589793

120

```
import math as m  
m.pi  
m.factorial(5)
```

3.141592653589793

120



import 문 : 복수의 모듈 불러오기

- 여러 개의 모듈 이름 또는 패키지 안의 모듈 이름을 한 줄에 작성해서 한꺼번에 불러오기 할 때 사용하는 방법
- X는 모듈 이름 또는 패키지 안의 모듈 이름이다
 - 패키지 이름만 사용할 수도 있지만 별다른 쓰임새가 없다
- 여러 줄로 작성해야 할 때는 '\' (역슬래시)를 사용할 수 있지만 권장하지 않는다
- import 문을 사용하면 항상 패키지 또는 모듈 이름과 함께 함수나 클래스 이름 전부를 사용하기 때문에 잠재적인 명칭 오류를 피할 수 있다

```
import X1, X2, ..., Xn
```

```
import math, random, datetime, os, os.path # 여러 개의 모듈을 한꺼번에 불러온다
math.sqrt(2) # 2의 제곱근을 구한다
random.randint(1, 99) # 1~99 사이 임의의 정수를 생성한다
datetime.datetime.now().month # 현재 몇 월인지 알아본다
os.environ['LOGNAME'] # 현재 로그인 사용자의 ID를 알아본다
os.path.exists('고향의봄.txt') # 현재 폴더에 지정한 파일이 있는지 확인한다
```

```
1.4142135623730951
```

```
18
```

```
8
```

```
'jinsoopark'
```

```
True
```



from ... import 문 : 단일 모듈/함수(클래스) 불러오기

- **X**는 모듈 이름 또는 패키지 이름이거나 패키지 안의 모듈 이름이다
- **Y**는 모듈 이름 또는 모듈 안 함수 이름(괄호는 생략) 또는 클래스 이름(괄호는 생략)이다
- **from ... import** 문은 점 연산자(.)로 패키지와 모듈 이름을 불러오지 않고 직접 함수 이름 등을 불러 사용하기 때문에 명칭 오류가 생길 수 있다
- 예를 들어, numpy 모듈의 sum 함수를 **from numpy import sum**으로 불러왔다면, **numpy.sum**이 아니라 **sum**으로 사용하기 때문에, 코드에서 파이썬 표준 라이브러리의 **sum** 함수를 호출한 것인지 numpy 모듈의 **sum** 함수를 호출한 것인지 구분하기 어려워진다
- **as** 식별자는 선택 사항
 - 불러오는 모듈이나 함수에 **식별자**(변수)를 지정해서 별칭으로 사용할 수 있다
 - **식별자**로 기존 모듈이나 함수 이름을 사용하면 명칭 오류가 나니 주의해야 한다

from X import Y [as 식별자]

```
from math import pi  
pi
```

3.141592653589793

```
from math import factorial  
factorial(5)
```

120

```
from math import pi as mp  
mp
```

3.141592653589793

```
from math import factorial as mf  
mf(5)
```

120



from ... import 문 : 복수의 모듈/함수(클래스) 불러오기

- 1 `from X import Y1, Y2, ..., Yn`
- 2 `from X import (Y1, Y2, Y3, Y4, Y5, ..., Yn)`
- 3 `from X import *`

- X 는 모듈 이름 또는 패키지 이름이거나 패키지 안의 모듈 이름이다
- Yn 는 모듈 이름 또는 모듈 안 함수 이름(괄호는 생략) 또는 클래스 이름(괄호는 생략)이다
- ① 여러 개의 모듈이나 함수 또는 클래스를 한 줄에 작성해서 한꺼번에 불러올 때 사용하는 방법
- ②는 ①과 같지만 여러 줄에 걸쳐 다수의 모듈을 불러올 때 사용하는 방법
- ③ 모듈 안의 모든 함수를 불러올 때 사용하는 방법
 - 주로 GUI 라이브러리처럼 패키지 안에 불러올 객체가 많을 때 사용할 수 있다
 - 하지만 명칭 오류가 발생하지 않도록 주의해야 한다.
 - 사용자가 만든 패키지나 모듈이 있고, 패키지 폴더에 저장한 `__init__.py` 파일에 모듈의 객체 이름 목록을 담은 `__all__` 전역 변수를 지정했다면 `__all__`에 있는 객체들만 불러온다
 - 단, `private` 접근 제어자 객체는 제외되기 때문에 다른 프로그램에서 불러 사용하기 원치 않는 변수나 함수 이름은 밑줄(`_`)로 시작하면 된다

1 `from math import factorial, isnan, exp, log, sqrt`

2 `from math import (factorial, isnan, exp, log, sqrt, sin, tan, cos, degrees, radians)`

3 `from math import *`



예시 : from ... import 문

```
from math import factorial, isnan, exp, log, sqrt
```

```
factorial(5) # math.factorial 함수는 계승 함수
```

```
120
```

```
isnan(1) # math.isnan 함수는 숫자인지 아닌지 확인
```

```
False
```

```
exp(2) # math.exp 함수는 지수 함수( $e ** 2$ )
```

```
7.38905609893065
```

```
log(2) # math.log 함수는 로그 함수
```

```
0.6931471805599453
```

```
sqrt(2) # math.sqrt 함수는 제곱근 함수
```

```
1.4142135623730951
```

파이썬 표준 라이브러리

The Python Standard Library





파이썬 표준 라이브러리

- 지난 수년간 다양하고 완성도 높은 모듈들이 개발되었으며 파이썬 표준 라이브러리는 이러한 모듈들을 포함하고 있다 ...
- 그리고 파이썬 배포용 패키지의 용량이 커지다보니 몇몇 뛰어난(그러나 포괄적(generic)이지 못한) 모듈들은 파이썬 표준 라이브러리에서 제외 되기도 했다 ...
- 하지만, 해당 모듈들은 개인 개발자 혹은 단체 등 제3자(써드파티, third party)에 의해서 개별적으로 유지보수가 되고 있으며 ...
- 이는 ... 써드파티가 만들어 배포하는 모듈들도 파이썬 표준 라이브러리만큼이나 유용하고 뛰어나다는 의미이다 ...
- 따라서 ... 가급적이면 ... 직접 코드를 작성하기 전에 ...
- 기존에 존재하는 모듈이 있는지 확인해보는 습관을 가지는 것이 좋다 ...
- **Do NOT Reinvent the Wheel!!!**
- 이어지는 내용에서 보편적으로 많이 사용되는 패키지와 모듈에 대한 간략한 정보를 다루고자 한다



파이썬이 제공하는 유용한 패키지/모듈

● 파이썬 3.x 표준 라이버러리

- <https://docs.python.org/3/library/index.html>
- 흔히 사용하는 파이썬 표준 라이버러리(패키지/모듈)
 - ✿ `datetime, time, math, collections, random, os, urllib, itertools`

● 아나콘다

- <https://www.anaconda.com>
- 아나콘다(Avacaconda)에 포함되어 있는 라이버러리 중 데이터 분석에 매우 유용한 패키지/모듈
 - ✿ `matplotlib, nltk, networkx, numpy, pandas, scikit-learn, scipy, xlrd`



random.random()

균일 분포(uniform distribution)에 기반하여 0에서 1 사이(1은 포함하지 않는다) [0.0, 1.0) 실수 중 하나를 무작위로 선택해 반환

```
import random  
random.random()
```

0.8949032467617718

0에서 1사이의 임의의 실수 5개를 만들려면 어떻게 하면 될까?

```
import random  
rand_numbers = []  
for _ in range(5):  
    rand_numbers.append(random.random())  
else:  
    print(rand_numbers)
```



random 모듈

random 모듈은 다양한 통계 분포에 기반한 무작위 추출법으로 의사난수(pseudo-random number)를 생성하는 함수들을 제공

- 의사난수란 알고리즘으로 생성하는 난수이기 때문에 진정한 난수는 아니라 유사난수라고도 한다
- 의사난수로 생성하는 값은 규칙적(deterministic)이긴 하지만 아주 긴 주기를 갖고 생성한 숫자 배열이라 일반적으로 통계나 모델링에서 많이 사용
- 재현 가능한 결과를 얻으려면 random.seed 함수를 사용하면 된다

random.seed()

seed의 전달인자로 같은 정수를 사용하면 매번 같은 결과를 가져온다

```
print('1:', random.random())
print('2:', random.random())
random.seed(15)
print('3:', random.random())
random.seed(15)
print('4:', random.random())
print('5:', random.random())
random.seed(15)
print('6:', random.random())
```

1: 0.18854877350939192

2: 0.07113772341293145



random 모듈

random.randint(시작, 끝)

- 시작과 끝 사이(끝 포함) (시작, 끝) 난수 값 정수를 반환
- randrange(시작, 끝 + 1)와 같다

random.randrange(끝)

random.randrange(시작, 끝[, 폭])

- range 클래스로부터 받은 정수 값 중 무작위로 하나를 선택해서 반환
- choice(range())와 같지만 실제로 range 객체를 생성하지는 않는다
- range 클래스와 마찬가지로 끝은 포함하지 않는다

```
random.randint(1, 99)
```

```
random.randrange(1, 100)
```

random.shuffle(x[, random])

- 리스트 x의 객체들을 무작위로 섞는다
- 선택 매개변수 random의 기본값은 random 함수

```
L = list(range(30))
random.shuffle(L)
print(L)
```



random 모듈

random.choice(*seq*)

- 시퀀스형(*seq*, sequence data type)으로부터 받은 객체 중 무작위로 하나를 선택해서 반환
- 만약 *seq*가 빈(empty) 자료형이면 IndexError가 발생

```
random.choice(range(1, 100))
```

```
t = 1, 2, 3, 4, 5, 6, 7, 8, 9  
random.choice(t)
```

```
import string  
random.choice(string.ascii_letters)
```

random.sample(*population*, *k*)

- 모집단 *population*(리스트, 튜플, 세트 등)에서 *k* 개를 중복 없이 (즉, 교체하지 않고) 무작위로 선택해서 리스트로 반환

```
lotto = random.sample(range(1, 46), 6)  
print(lotto)
```

```
random.sample(range(1, 10), 6)
```



choice vs. sample

- 모집단에서 중복 허용하면서 샘플을 무작위로 선택하려면 `choice` 함수를 샘플 크기만큼 호출
- 모집단에서 중복 없이 샘플을 무작위로 선택하려면 `sample` 함수를 사용

```
[random.choice(range(5)) for _ in range(4)]
```

```
random.sample(range(5), 4)
```



collections 모듈의 Counter 클래스

collections.Counter([순회형])

- 같은 값을 가진 객체가 몇 개인지 확인할 때 사용
- 순회형 자료의 객체를 키로, 그들의 개수를 맵핑값으로 담고 있다
- 딕셔너리의 하위 클래스

```
from collections import Counter

L = ['a', 'a', 'b', 'c', 'c', 'c', 'd', 'd', 'd', 'e', 'e', 'f', 'f', 'f', 'f']
counts = Counter(L)

print(counts)
counts.most_common(3)
```



itertools 모듈

itertools 모듈 효율적인 순환(반복) 작업을 하도록 하는 순회자를 생성하는 함수들을 제공하는 모듈

itertools.product(*순회형, repeat=1)

- 불특정 다수의 순회형 값의 데카르트 곱(Cartesian product)을 생성하는 순회자(튜플)를 반환
- 이는 전달받은 모든 순회형을 중첩 **for** 문에서 사용하는 것과 유사
- 전달받은 순회형 자료 자체의 곱을 계산하려면 매개변수 **repeat**에 반복할 횟수를 설정
예) **product(x, repeat=3)** 은 **product(x, x, x)**와 같다

```
import itertools
x = itertools.product('123', repeat=2) # product('123', '123')와 같다
print(x)
for i in x:
    print(i)
```



itertools 모듈

itertools.permutations(순회형, r=None)

- 전달받은 순회형의 모든 값 중 연속적인 **r** 길이의 순열 순회자(튜플)를 반환
- **r**-length tuples, all possible orderings, no repeated elements

```
x = itertools.permutations('123', 2)
print(x)
for i in x: # 3P2 = 3 x 2 = 6
    print(i)
```



itertools 모듈

- 전달받은 순회형의 모든 값 중 r 길이의 중복 없는 조합 순회자(튜플)를 반환
- `itertools.combinations(순회형, r)`
- 조합 튜플은 입력한 순회형 값의 순서에 따라 구성
- r -length tuples, in sorted order, no repeated elements

```
x = itertools.combinations('123', 2)
print(x)
for i in x: # 3C2 = 3! / (2! x (3 - 2)! = 6 / (2 x 1) = 3
    print(i)
```



itertools.combinations_with_replacement(순회형, r)

- 전달받은 순회형의 모든 값 중 **r** 길이의 중복 없는 조합 순회자(튜플)를 반환
- 개별 값을 두 번 이상 반복할 수 있다
- 조합 튜플은 입력한 순회형 값의 순서에 따라 구성
- **r-length tuples, in sorted order, with repeated elements**

```
x = itertools.combinations_with_replacement('123', 2)
print(x)
for i in x:
    print(i)
```



예시 : **itertools** 모듈

```
import itertools

# 항목들의 곱집합(cartesian product)
colors = ['green', 'blue', 'red']
instruments = ['drum', 'guitar']
for i in itertools.product(colors, instruments):
    print(i)
```



예시 : itertools 모듈

```
for i in itertools.product('xyz', repeat=2): # 항목들의 곱집합(cartesian product)
    print(i)
```

```
for i in itertools.permutations('xyz', 2): # 항목들의 순열(permuation)
    print(i)
```

```
for i in itertools.combinations('xyz', 2): # 항목들의 조합(combination)
    print(i)
```



모듈	설명
string	<ul style="list-style-type: none">• <code>string.ascii_letters</code>와 <code>string.hexdigits</code>은 어떤 문자가 ASCII 문자인지 16진수인지 구분할 때 유용하게 사용할 수 있는 상수• <code>string.Formatter</code> 클래스는 하위 클래스를 만들어 사용자 정의 문자열 서식을 설정할 수 있다
textwrap	문자열의 줄 길이를 설정하거나 들여쓰기를 설정할 때 유용
struct	<ul style="list-style-type: none">• 숫자, 불린, 문자열의 이진 값을 사용하여 바이트(byte) 객체로 패킹 또는 언패킹하는 함수를 제공• 데이터를 C로 쓰여진 라이브러리로 보내거나 받는 경우 이러한 데이터를 처리할 때 유용
difflib	문자열과 같은 시퀀스형 데이터를 비교하는 클래스와 메소드를 제공하고, 결과 값을 표준 'diff' 형식 또는 HTML 형식으로 출력
re	파이썬의 가장 강력한 문자열 처리 모듈
io.StringIO	<ul style="list-style-type: none">• 메모리 내(in-memory) 텍스트 파일처럼 동작하는 문자열 형태의 객체를 제공하는 클래스• 실제 파일을 생성하지 않고 문자열을 파일처럼 사용할 수 있다



커맨드 라인(command-line)

모듈	설명
fileinput	<ul style="list-style-type: none">• <code>input</code> 함수는 콘솔로부터 반복적으로 입력 값을 가져오거나 파일로부터 원하는 모든 줄을 하나의 연속된 줄처럼 가져온다• 이 모듈은 <code>filename</code> 함수와 <code>lineno</code> 함수를 사용하여 현재 파일명과 줄 번호를 확인할 수 있고 일부 압축 파일도 처리할 수 있다
getopt	커맨드 라인 옵션을 처리할 수 있으며 사용하기 편리해 개발자들에게 있기가 많아 표준 라이브러리에 포함된 지 오래 됐다
optparse	라이브러리에 새로 추가된 더 강력한 커맨드 라인 옵션을 처리하는 함수



모듈/패키지	설명
math	표준 수학 함수를 제공
cmath	복소수 수학 함수를 제공
random	난수 생성과 관련된 다양한 함수를 제공
numbers	<ul style="list-style-type: none">➊ 숫자형 추상 클래스를 담고 있음➋ 추상 클래스 : 직접 사용할 수는 없고 상속을 통해서만 사용이 가능한 클래스➌ x라는 이름의 객체가 있을때 <code>isinstance(x, numbers.Number)</code> 함수를 사용해서 x가 숫자인지를 확인할 수 있고, <code>isinstance(x, numbers.Rational)</code> 또는 <code>isinstance(x, numbers.Integral)</code> 함수를 사용해서 x가 어떤 종류의 숫자인지를 구체적으로 확인할 때 유용
numpy	매우 효율적인 n -차원 배열과 기본 선형 대수학 함수, 푸리에(Fourier) 변환 함수를 제공할 뿐만 아니라 C, C++, Fortran 코드와 통합할 수 있는 도구를 제공하는 써드파티 패키지 <ul style="list-style-type: none">➌ www.scipy.org에서 다운받을 수 있음
scipy	Numpy를 통합하고 통계 계산, 신호와 이미지 처리, 유전자 알고리즘을 포함해서 그 이 외에도 다양한 기능을 포함한 모듈들을 제공하는 써드파티 패키지 <ul style="list-style-type: none">➌ www.scipy.org에서 다운받을 수 있다



모듈	설명
calendar datetime	그레고리력(Gregorian calendar)에 기준한 시간과 날짜를 처리하는 다양한 함수와 클래스를 제공
time	<ul style="list-style-type: none">⌚ 타임스탬프를 처리할 때 사용<ul style="list-style-type: none">⌚ 타임스탬프는 시간을 표시하는 숫자(초)라 보면 됨⌚ 유닉스 기준 1970-01-01T00:00:00에 시작된 총 시간(초)의 합⌚ 협정세계시(UTC, Universal Time Universal Time Coordinated) 또는 서머타임제 등의 현지 시각을 반영한 장치의 타임스탬프를 확인할 때 사용⌚ 날짜와 시간을 생성하거나 날짜/시간 문자열 서식을 다양한 방법으로 설정할 수 있으며 날짜와 시간을 포함한 문자열을 분석할 수도 있음



calendar 모듈

calendar.calendar(year, w=2, l=1, c=6, m=3)

- 전달받은 년도(**year**) 달력을
- 3 컬럼(**m**) 형식의
- 멀티라인(multi-line)
- 문자열로 반환
- 기본값으로
- 날짜 컬럼의 크기(**w**)는 2
- 1주당 1줄(**l**)
- 달간 간격(**c**)은 6

<pre>import calendar print(calendar.calendar(2025, m=4, c=3))</pre>
2025
January
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
February
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
March
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
April
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
May
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
June
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
July
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
August
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
September
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
October
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
November
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
December
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

calendar.prcal(year, w=0, l=0, c=6, m=3)

calendar 함수와 같은 결과를 보여 주지만 멀티라인 문자열을 반환하지 않고 대신 달력을 출력

<pre>import calendar calendar.prcal(2025, m=4, c=3)</pre>
2025
January
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
February
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
March
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
April
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
May
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
June
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
July
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
August
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
September
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
October
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
November
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
December
Mo Tu We Th Fr Sa Su 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31



calendar 모듈

calendar.month(*year*, *month*, *w*=0, *l*=0)

- 전달받은 년도(*year*)의 월(*month*)을 멀티라인(multi-line) 문자열로 반환

```
import calendar
print(calendar.month(2025, 7))
```

July 2025						
Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

calendar.pmonth(*year*, *month*, *w*=0, *l*=0)

- month* 함수와 같은 결과를 보여 주지만 멀티라인 문자열을 반환하지 않고 대신 달력을 출력

```
import calendar
calendar.pmonth(2025, 7)
```

July 2025						
Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			



calendar.weekday(*year*, *month*, *day*)

- 전달받은 년도, 월(1-12), 일(1-31), 즉 해당 날짜의 요일(0-6) 정보를 반환

- 0 : 월요일
- 1 : 화요일
- 2 : 수요일
- 3 : 목요일
- 4 : 금요일
- 5 : 토요일
- 6 : 일요일

```
import calendar  
calendar.weekday(2025, 7, 1)
```



calendar.monthrange(*year*, *month*)

전달받은 년도(*year*)와 월(*month*)의 첫 번째 날(즉, 1일)이 무슨 요일인지와 그 달이 며칠까지 있는지를 튜플 형태로 반환

```
import calendar  
calendar.monthrange(2025, 7)
```



time.sleep(secs)

- 정해진 시간만큼 초(*secs*) 단위로 현재 실행중인 스레드(thread)를 중지
- secs*는 실수로 표현하며 1이면 1초, 0.7이면 0.7초를 의미

```
import time
for i in range(10):
    print(f'{i / 2} second passed')
    time.sleep(i / 2)
else:
    print('Done!')
```



time.time()

- 1970년 1월 1일 00:00:00(협정세계시 : UTC, universal time coordinated) 기준으로 현재 시간을 초 단위 실수형으로 반환

time.localtime([secs])

- 전달받은 초(secs)를 현지 시간(년도, 월, 일, 시, 분, 초)으로 변환해서 반환
- 초(secs)가 주어지지 않으면 현재 현지 시간을 반환

```
import time  
  
time.time()  
  
time.localtime(time.time())  
  
time.localtime()
```



time 모듈

time.asctime([t])

- 튜플 자료형 또는 `time.localtime` 함수가 반환한 `struct_time`의 튜플 자료형 값(`t`)을 전달받아 사람이 이해하기 쉬운 형식으로 반환
- 시간(`t`)이 주어지지 않으면 현재 현지 시간을 반환

```
import time
```

```
timeasctime(time.localtime(time.time()))
```

time.ctime([secs])

- 1970년 1월 1일 00:00:00(협정세계시 : UTC, universal time coordinated) 기준으로 현재 시간을 초(`secs`) 단위 현지 시간으로 입력 받아 사람이 이해하기 쉬운 형식으로 반환
- 초(`secs`)가 주어지지 않으면 현재 현지 시간을 반환
- `time.ctime(secs)`과 `timeasctime(time.localtime(secs))`은 동일

```
timeasctime()
```

```
time.ctime(time.time())
```

```
time.ctime()
```



time.strftime(format[, t]) 튜플 자료형 또는 `time.localtime` 함수가 반환한 `struct_time`의 튜플 자료형 값(*t*)을 전달받아 포맷(*format*) 전달인자가 제공하는 형식으로 변환해서 문자열로 반환

```
import time
time.strftime('%A, %x => current time is %X [%I%p]', time.localtime())
```

코드	설명	예시
%c	설정된 현지(locale) 형식의 날짜와 시간	Fri Jul 1 13:55:49 1988
%x	설정된 현지(locale) 형식의 날짜	07/01/88
%X	설정된 현지(locale) 형식의 시간	13:55:49
%a	요일 줄임 표기	Thu
%A	요일	Thursday
%w	숫자 요일	[0(일),6(토)]
%b	월 줄임 표기	Apr
%B	월	April



time 모듈

코드	설명	예시
%m	숫자 월	[01,12]
%d	일(day of the month)	[01,31]
%j	누적 일(day of the year)	[001,366]
%H	시(hour) - 24시간 형식	[00,23]
%I	시(hour) - 12시간 형식	[01,12]
%P	AM / PM	
%M	분(minute)	[00,59]
%S	초(second)	[00,59]
%U	1년 중 누적 주(일요일을 한 주의 시작으로) 1년 중 첫 번째 일요일보다 앞서 있는 날들은 모두 0주로 표시	[00,53]
%W	1년 중 누적 주(월요일을 한 주의 시작으로) 1년 중 첫 번째 월요일보다 앞서 있는 날들은 모두 0주로 표시	[00,53]
%Y	년도	1988
%y	뒷 두자리 년도(앞 두자리 세기부분 제외)	[00,99]
%z	+HHMM or -HHMM 형식의 UTC/GMT 시차를 나타내는 표준시간대 오프셋은 24개로 나눠져 있다 한국: +0900	[-23:59, +23:59] 한국: +0900
%Z	표준시간대 명	대한민국 표준시: KST
%%	리터럴 '%' 문자	%



모듈	설명
bisect	<p>정렬된 리스트(sorted list)와 같이 정렬된 시퀀스형(sorted sequence)을 검색하거나 정렬 순서를 유지하면서 추가 항목을 삽입하는 기능을 제공</p> <ul style="list-style-type: none">이진검색(binary search) 알고리즘을 사용하기 때문에 처리 속도가 매우 빠르다
heapq	<p>리스트와 같은 시퀀스형을 힙(heap)으로 변환하는 함수들을 제공</p> <ul style="list-style-type: none">힙 : 첫 번째 항목이 항상 가장 작은 항목이고 항목의 삽입하거나 제거할 때 항상 그 순서를 유지하는 시퀀스형
array	<ul style="list-style-type: none">숫자 또는 문자를 매우 효율적으로 저장할 수 있는 array.array 시퀀스형을 제공생성시 정한 객체형 이외 유형의 객체는 저장할 수 없다는 점을 제외하면 리스트와 유사numpy 패키지 또한 효율적인 배열(array)을 제공
weakref	<ul style="list-style-type: none">약한 참조를 생성하는 기능을 제공(약한 참조 : 일반적인 객체참조와 비슷하지만 만약 어떤 객체에 참조되는 것이 약한 참조 뿐일 경우 해당 객체는 가비지 컬렉션 대기열에 올라간다)garbage collection : 사용하지 않는 객체를 메모리에서 삭제하는 것이 모듈을 이용하면 단순히 객체에 대한 참조를 가지고 있다는 것만으로 객체를 메모리에 남겨둠으로써 낭비되는 메모리를 최소화 할 수 있다약한 참조를 받는 객체가 아직 남아있는지 그리고 해당 객체에 대한 접근이 가능한지에 대한 확인도 가능



파일 형식, 인코딩, 데이터 지속성(persistence)

모듈	설명
bz2	.bz 파일 처리
gzip	.gz 파일 처리
tarfile	.tar, .tar.gz, .tgz, tar.bz2 파일 처리
zipfile	.zip 파일 처리
aifc	AIFF(Audio Interchange File Format) 오디오 형식 처리
wave	(압축되지 않은) .wav 파일 처리
audioop	특정 형식의 오디오 데이터를 처리
sndhdr	파일에 저장된 소리 데이터가 어떤 종류의 소리 데이터인지 그리고 샘플링 레이트(sampling rate) 등 일부 속성을 알아내는 함수를 제공
configparse	RFC 822 형식의 구성 파일을 읽고 쓰는 함수를 제공 💡 구성 파일(configuration file) : 구 윈도우 .ini 파일과 유사



모듈	설명
csv	<p>CSV(common separated value) 파일이나 이와 유사한 탭 단락(tab-delimited) 데이터 등을 읽고 쓰는데 사용하며 CSV 파일을 문자 그대로 읽히지 않도록 방지하는 특징이 있다</p> <ul style="list-style-type: none">엑셀과 같은 많은 프로그램들이 CSV 파일 형식으로 읽고 쓸 수 있다
pickle	<ul style="list-style-type: none">시퀀스형 전체를 포함해서 임의의 파이썬 객체를 디스크에 저장하거나 디스크로부터 불러오는 기능을 제공다양한 DBM 파일을 지원DBM 파일 : 딕셔너리와 비슷하지만 모든 요소들이 메모리가 아닌 디스크에 저장되고 키와 값이 바이트(bytes) 객체 또는 문자열 형태
shelve	<ul style="list-style-type: none">문자열 키와 임의의 파이썬 객체를 값으로 갖는 DBM 파일을 제공파이썬 객체와 바이트(bytes) 객체 간의 자유로운 변환을 가능케 한다



모듈	설명
shutil	파일과 디렉토리 관리에 유용한 함수를 제공 • <code>copy()</code> 함수는 파일을 복사할 때 사용 • <code>copytree()</code> 함수는 디렉토리 전체를 복사할 때 사용 • <code>move()</code> 함수는 특정 디렉토리로 이동할 때 사용 • <code>rmtree()</code> 함수는 하위 디렉토리(non-empty 디렉토리 포함)를 포함한 디렉토리 전체를 삭제할 때 사용
tempfile	임시 파일과 디렉토리 관련 함수를 제공
filecmp	파일 또는 전체 디렉토리를 비교
subprocess	다른 프로세스를 시작하고 해당 프로세스와 데이터를 주고 받은 후 결과 값을 가져오는 여러 함수를 제공
multiprocessing	다중 프로세스의 분할 실행과 누적 결과 값 처리를 위한 환경을 제공할 뿐만 아니라 멀티쓰레드(multi-threading) 방식의 대안으로 자주 사용



shutil.copy(src, dst)

- *src* 파일을 *dst* 파일로 복사
- 만약 *dst*가 파일 이름이 아니라 폴더 이름이면 *src* 파일 이름으로 *dst* 폴더에 복사하고 동일한 파일 이름이 있으면 덮어쓴다

```
import shutil  
shutil.copy('고향의봄.txt', '고향의봄.bak')
```

'고향의봄.bak'



tempfile 모듈 파일을 임시로 만들어서 사용할 때 유용한 모듈

tempfile.mkstemp(suffix=None, prefix=None, dir=None, text=False)

- 가장 안전한 방법으로 임시 파일을 생성하는 함수
- 임시 파일을 생성한 사용자 ID만이 파일 읽기와 쓰기가 가능
- 중복되지 않는 파일 이름을 무작위로 만들어 파일 생성
- 다음 페이지에서 설명하는 `TemporaryFile` 클래스와는 달리 `mkstemp` 함수 사용자가 임시 파일 사용 후 직접 삭제를 해야 한다
- 튜플(파일 열기 위한 OS 레벨 핸들, 파일이 생성된 절대경로)을 반환

```
import tempfile  
tmp = tempfile.mkstemp()  
tmp
```

```
(57, '/var/folders/64/psv8bs955pq9wmp785kp0ghh0000gn/T/tmpgea5mf1g')
```



tempfile 모듈

tempfile.TemporaryFile(*mode='w+b'*, *buffering=None*, *encoding=None*, *newline=None*, *suffix=None*, *prefix=None*, *dir=None*)

- 임시 저장 공간으로 사용할 파일 객체를 반환
- `tempfile.mkstemp` 함수와 마찬가지로 안전한 방법으로 파일을 생성
- 파일이 닫히는(`close()`) 순간 자동으로 삭제된다

```
import tempfile  
tmpfile = tempfile.TemporaryFile()
```

```
tmpfile.write(b'Hello Python')
```

12

```
tmpfile.seek(0) # tmpfile의 맨 앞으로 이동
```

0

```
tmpfile.read()
```

b'Hello Python'

```
tmpfile.close() # 생성한 임시파일이 자동으로 삭제됨
```



모듈	설명
os	<p>플랫폼에 상관없이 운영체제 기능을 이용할 수 있는 함수를 제공</p> <ul style="list-style-type: none">• <code>environ</code> : 환경변수명과 그 값을 담고 있는 맵핑 객체를 저장하고 있는 변수• <code>getcwd()</code> : 현재 작업중인 디렉토리를 반환• <code>chdir()</code> : 디렉토리 위치를 이동• <code>access()</code> : 파일의 존재 여부와 그 파일이 읽기 전용인지 쓰기 전용인지 등을 판단• <code>listdir()</code> : . 와 .. 를 제외한 파일과 디렉토리의 목록을 반환• <code>stat()</code> : 파일이나 디렉토리의 상태, 수정한 날짜, 크기 등의 속성 정보를 제공• <code>mkdir()</code> : 디렉토리를 생성• <code>makedirs()</code> : 중간(intermediate) 디렉토리를 생성• <code>rmdir()</code> : 빈(empty) 디렉토리를 제거• <code>removedirs()</code> : 빈 디렉토리만 담고 있는 디렉토리 트리를 제거• <code>remove()</code> : 파일/디렉토리를 제거• <code>rename()</code> : 파일명/디렉토리명을 수정



os.environ

현재 시스템의 환경 변수 값들을 딕셔너리 자료 형태로 반환

```
import os
os.environ

environ({'TERM_SESSION_ID': 'w0t0p0:52A23250-88D1-4B45-AD4C-DA52F54F4D32', 'SSH_AUTH_SOCK': '/private/tmp/com.apple.launchd.Kb2q9IFeKB/Listeners', 'LC_TERMINAL_VERSION': '3.4.8', 'COLORFGBG': '15;0', 'ITERM_PROFILE': 'Default', 'XPC_FLAGS': '0x0', 'LANG': 'ko_KR.UTF-8', 'PWD': '/Users/jinsoopark/Dropbox/_data/document/lecture/big-data-analytics', 'SHELL': '/bin/zsh', '__CFBundleIdentifier': 'com.googlecode.iterm2', 'TERM_PROGRAM_VERSION': '3.4.8', 'TERM_PROGRAM': 'iTerm.app', 'PATH': '/Users/jinsoopark/Dropbox/_data/venv/3.9full/bin:/Library/Frameworks/Python.framework/Versions/3.9/bin:/Library/Frameworks/Python.framework/Versions/3.8/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin', 'LC_TERMINAL': 'iTerm2', 'COLORTERM': 'truecolor', 'COMMAND_MODE': 'unix2003', 'TERM': 'xterm-color', 'HOME': '/Users/jinsoopark', 'TMPDIR': '/var/folders/7l/7zxx962d12x3744rn6scyct40000gn/T/', 'USER': 'jinsoopark', 'XPC_SERVICE_NAME': '0', 'LOGNAME': 'jinsoopark', 'ITERM_SESSION_ID': 'w0t0p0:52A23250-88D1-4B45-AD4C-DA52F54F4D32', '__CF_USER_TEXT_ENCODING': '0x1F5:0x3:0x33', 'SHLVL': '1', 'OLDPWD': '/Users/jinsoopark/Dropbox/_data/document/lecture/big-data-analytics', 'PS1': '(3.9full) %(?::%{\x1b[01;32m%}→ :%{\x1b[01;31m%}→ ) %{$fg[cyan]%'%c%{$reset_color%} $(git_prompt_info)', 'ZSH': '/Users/jinsoopark/.oh-my-zsh', 'PAGER': 'cat', 'LESS': '-R', 'LSCOLORS': 'Gxfxcxdxbxegedabagacad', 'VIRTUAL_ENV': '/Users/jinsoopark/Dropbox/_data/venv/3.9full', '_': '/Users/jinsoopark/Dropbox/_data/venv/3.9full/bin/jupyter', 'JPY_PARENT_PID': '3366', 'CLICOLOR': '1', 'GIT_PAGER': 'cat', 'MPLBACKEND': 'module://ipykernel.pylab.backend_inline'})
```

딕셔너리 자료 형태이기 때문에 키 값을 입력해 필요한 정보만 검색 가능

```
os.environ['PATH']
```

```
'/Users/jinsoopark/Dropbox/_data/venv/3.9full/bin:/Library/Frameworks/Python.framework/Versions/3.9/bin:/Library/Frameworks/Python.framework/Versions/3.8/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin'
```

```
os.environ['HOME']
```

```
'/Users/jinsoopark'
```

```
os.environ['LOGNAME']
```

```
'jinsoopark'
```



모듈	설명
os.path	<p>경로의 문자열 조작 및 파일 시스템 관련 편리한 기능들을 제공</p> <ul style="list-style-type: none">• abspath() : 불필요한 경로 분리자(path separator)와 .. 요소를 제거한 절대 경로(absolute path)를 반환• split() : 첫번째 요소는 경로를, 두번째 요소는 파일명을 담고(파일 이름이 없는 경우 두번째 요소는 공백으로 처리) 있는 2개의 요소를 가진 튜플(2-tuple)을 반환• basename() : 파일명을 담고 있다• dirname() : 경로를 담고 있다• splittext() : 파일명을 이름과 확장자, 두 부분으로 나눈다• join() : 경로 문자열들을 입력받아 플랫폼에 적합한 경로 분리자를 적용한 하나의 경로를 반환• exists(), getsize(), isfile(), isdir()
mimetypes	<ul style="list-style-type: none">• guess_type() : 주어진 파일의 MIME(Multi-Purpose Internet Mail Extensions) 유형을 추측



모듈	설명
socket	<ul style="list-style-type: none">• 네트워크의 가장 밑단(lowest level)에서 필요로 하는 핵심적인 네트워크 기능을 제공• 소켓(socket) 생성, DNS(Domain Name System) 검색, IP 주소 처리 등
ssl	암호화되고 인증된 소켓을 제공
socketserver	<ul style="list-style-type: none">• TCP와 UDP(User Datagram Protocol) 서버를 제공• 이 서버들은 요청을 직접 처리하거나 별도의 프로세스 또는 쓰레드를 생성하여 요청을 처리할 수 있다
asyncore	비동기식 클라이언트와 서버 소켓 관리 기능을 제공
asynchat	asyncore 기반 위에 작성된 모듈
http.server	CGI(Common Gateway Interface) 스크립트를 실행할 수 있는 요청 처리기(request handler)를 지원하는 HTTP서버를 제공
http.client	HTTP 요청에 대한 클라이언트 접속을 제공
http.cookiejar	쿠키를 관리하는 여러 함수를 제공
urllib	http.client보다 고수준의 쉽고 간편한 URL 접속을 제공하는 패키지
cgi cgitb	CGI 스크립트와 관련된 기능을 제공



모듈	설명
<code>html.parser</code>	HTML과 XHTML 문서를 분석
<code>xmlrpclib.client</code> <code>xmlrpclib.server</code>	XML-RPC(Remote Procedure Call) 관련 기능을 제공
<code>ftplib</code>	FTP(File Transfer Protocol) 관련 기능을 제공
<code>nntplib</code>	NNTP(Network News Transfer Protocol) 관련 기능을 제공
<code>telnetlib</code>	TELNET 관련 기능을 제공
<code>smtpd</code>	SMTP(Simple Mail Transfer Protocol) 서버를 제공
<code>smtplib</code>	SMTP용 이메일 클라이언트를 제공
<code>poplib</code>	POP3(Post Office Protocol)용 이메일 클라이언트를 제공
<code>mailbox</code>	다양한 이메일 형식을 제공
<code>email</code>	개별 이메일 메시지를 작성하고 수정하는 기능을 제공(다중 구조 메시지 포함)

- Twisted(www.twistedmatrix.com) : 포괄적인 써드파티 네트워킹 라이브러리를 제공
- Django(www.djangoproject.com), Trubogears(www.turbogears.org) : 웹 애플리케이션 개발을 위한 써드파티 웹 프로그래밍 라이브러리를 제공
- Plone(www.plone.org), Zope(www.zope.org) : 웹 프레임워크와 컨텐츠 관리 시스템을 제공



webbrowser 모듈

- 사용자에게 웹 기반 문서를 볼 수 있도록 해줌
- `webbrowser.open(url, new=0, autoraise=True)`
 - 기본 웹 브라우저를 사용해 `url`에 접속
 - `new=0` : (가능하다면) 현재 실행 중인 브라우저 창에서 `url`에 접속
 - `new=1` : (가능하다면) 새 브라우저 창을 열어 `url`에 접속
 - `new=2` : (가능하다면) 새 브라우저 탭(tab)을 열어 `url`에 접속
 - `autoraise`가 참(`True`)이면 접속하는 창이 현재 창으로 스크린에 나타난다
- `webbrowser.open_new(url)`
 - (가능하다면) 기본 웹 브라우저의 새 창을 열어 `url`에 접속
- `webbrowser.open_new_tab(url)`
 - (가능하다면) 기본 웹 브라우저의 새 탭(tab)을 열어 `url`에 접속



예시 : `webbrowser` 모듈

```
import webbrowser
url = 'http://www.python.org/'

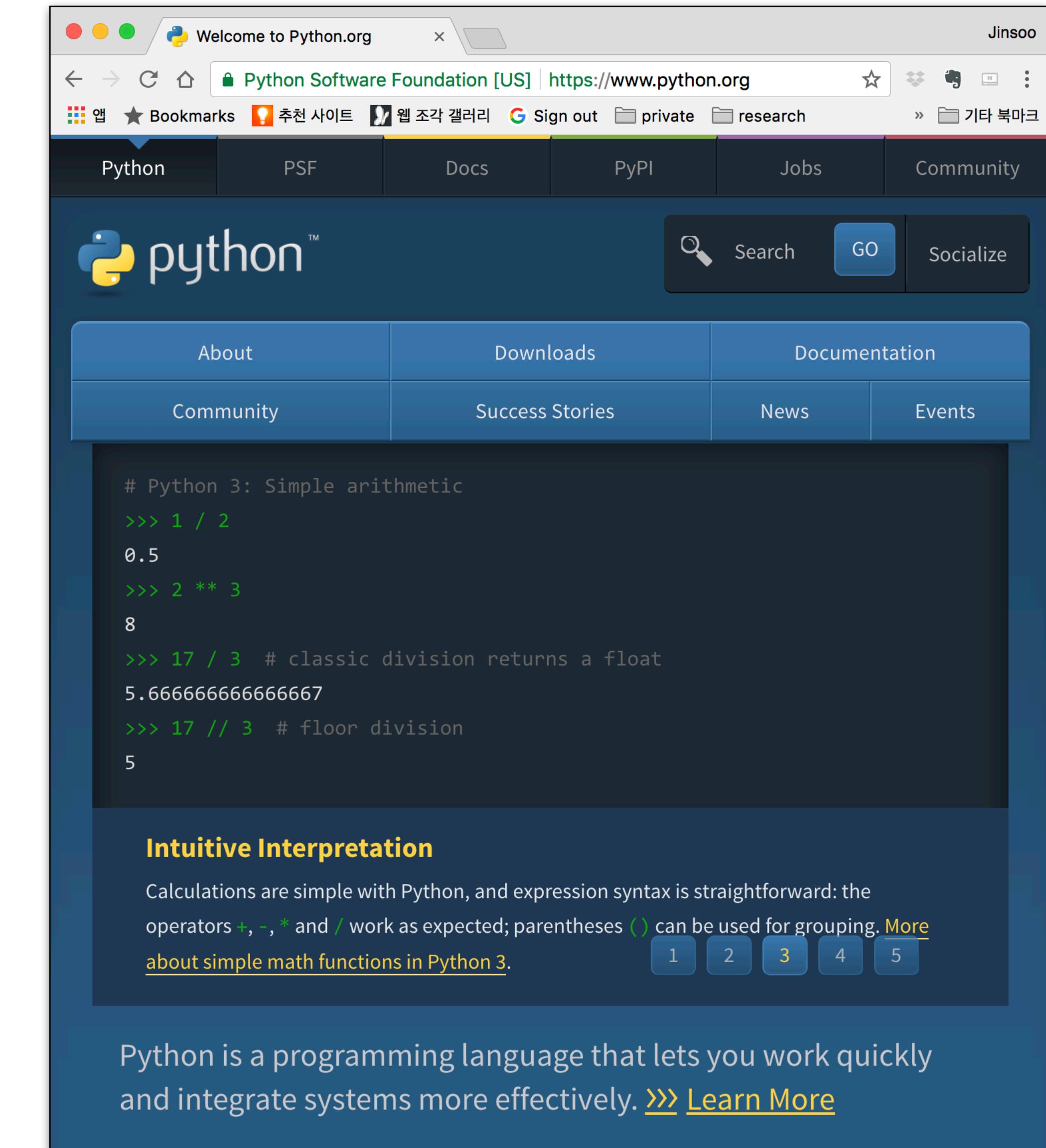
webbrowser.open(url)
True

webbrowser.open(url, new=1)
True

webbrowser.open(url, new=2)
True

webbrowser.open_new(url)
True

webbrowser.open_new_tab(url)
True
```





모듈/라이브러리	설명
<code>xml.dom</code> <code>xml.dom.minidom</code>	DOM(Document Object Model) 파서(Parser, 문장구조분석기)
<code>xml.sax</code>	SAX(Simple API for XML) 파서(Parser, 문장구조분석기)
<code>lxml</code>	트리 모듈이 제공하는 것들의 사실상 상위 집합인 인터페이스를 제공할 뿐만 아니라 Path, XSLT, 및 기타 여러 XML 관련 기술을 지원하는 많은 추가 기능을 제공하는 써드파티 라이브러리(www.codespeak.net/lxml)



표준 라이브러리에서 약 200여개의 패키지와 모듈을 사용할 수 있다

모듈/라이브러리	설명
doctest	문서화(docstrings) 문자열을 생성하고 테스트 하는 기능을 제공
unittest	유닛 테스트(unit-test) 프레임워크
py.test	코드가 문제없이 잘 작동하는지 테스트하는 써드파티 테스트 프레임워크(pytest.org)
logging	로깅(logging)을 위한 통합 인터페이스를 제공; 파일에 기록할 수도 있고 HTTP GET 또는 POST 요청이나 이메일, 소켓 등을 사용하여 기록할 수도 있다
pprint	• 시퀀스형과 같은 특정 파이썬 객체를 깔끔하게 출력('pretty printing')할 수 있는 함수 제공 • 디버깅에 유용하다
inspect	현재 사용중인 객체들의 상태를 확인
threading	쓰레드 애플리케이션 생성을 지원
queue	멀티 쓰레드에서 안전한 실행을 보장(thread-safe)하는 3종류의 큐(queue)를 제공
tkinter	Tk 그래픽 사용자 인터페이스(GUI) 라이브러리
abc	추상기본클래스(Abstract Base Class)를 생성할 때 필요한 함수를 제공하는 모듈
copy	<code>copy</code> 함수와 <code>deepcopy</code> 함수를 제공

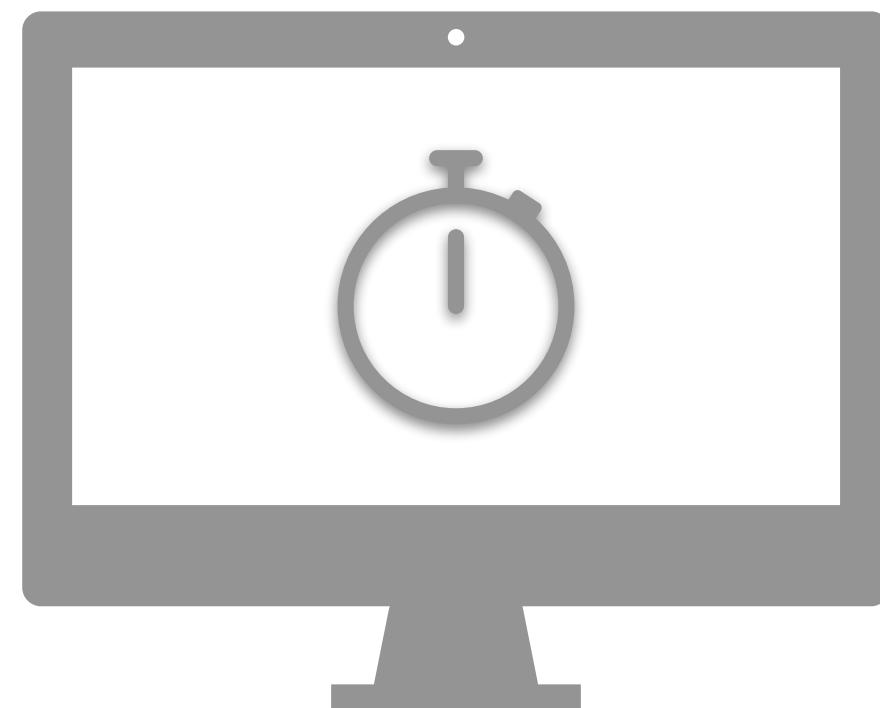
파이썬의 조립 블록 구성 요소

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)
```



● 내장 함수 다루기

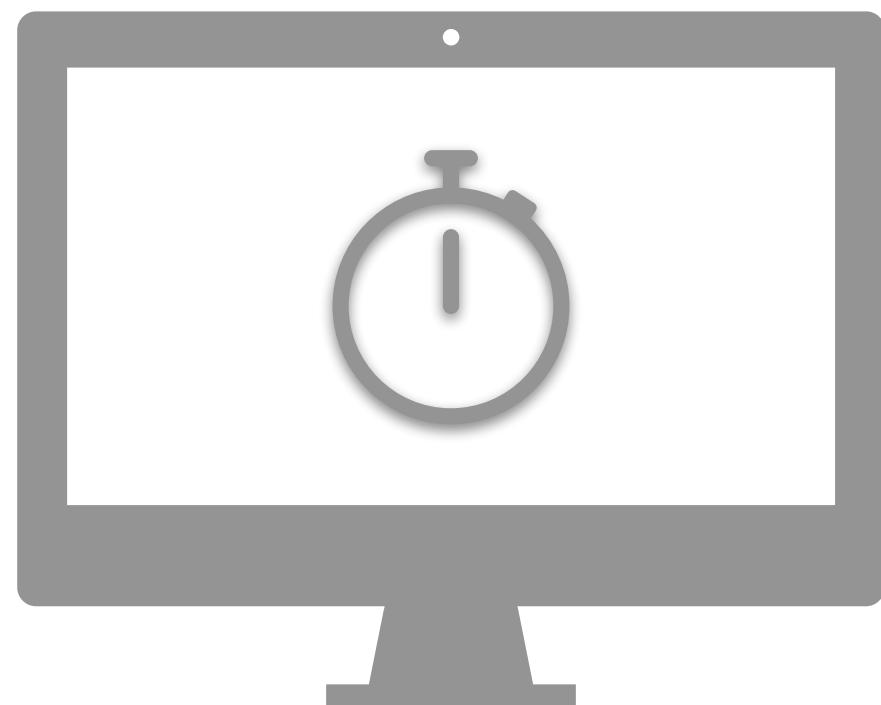
- 앞서 언급한 데이터 분석에 흔히 사용하는 내장 함수들을 활용해서 다음 작업을 실행
 - 아래 정수들을 포함하는 튜플을 생성한 후 변수에 할당하고 출력
 - 1, 3, 5, 7, 9
 - 위에서 생성한 튜플을 리스트로 형변환한 후 출력
 - 리스트에 속한 모든 객체를 문자열로 변환
 - 리스트에 'x', 'y', 'z'를 추가
 - 리스트에 속한 모든 객체와 객체의 자료형을 출력
 - 자료형은 **type** 함수를 사용해 출력

● 실행 결과 예시

```
(1, 3, 5, 7, 9)
[1, 3, 5, 7, 9]
1 <class 'str'>
3 <class 'str'>
5 <class 'str'>
7 <class 'str'>
9 <class 'str'>
x <class 'str'>
y <class 'str'>
z <class 'str'>
```



람다 함수





Lab : 일반 함수를 람다 함수로 변환



- 다음 함수를 람다 함수로 바꾸기

```
def area(b, h):
    """밑변과 높이를 전달받아서 삼각형의 넓이를 구하는 함수다.

Args:
    b (int | float): 밑변
    h (int | float): 높이

Returns:
    float: 삼각형의 넓이
    """
    return 0.5 * b * h

x = area(4, 5)
print(x)
```

10.0



● 람다 함수 만들기

- 임의의 두 실수를 비교하여, x 가 y 보다 크거나 같으면 x 를 출력하고, 그렇지 않으면 y 를 출력하는 함수를 **lambda**를 이용해 구현

● 실행 결과 예시

- 람다 함수를 정의해서 변수 g 로 할당한 후 g 를 실행

```
>>> g(1, 2)  
2
```



Lab : 람다 함수 (2)



● 람다 함수 만들기

- 임의의 두 정수를 입력받아 둘 다 양수라면 **True**를, 그렇지 않다면 **False**를 반환하는 함수를 **lambda**를 이용해 구현

● 실행 결과 예시

- 람다 함수를 정의해서 변수 **g**로 할당한 후 **g**를 실행

```
>>> g(0, 5)
False

>>> g(1, 3)
True
```



Lab : 람다 함수 (3)



● 람다 함수 만들기

- 정수값들로 이루어진 리스트를 입력받아, 리스트가 담고 있는 값들의 최솟값, 최댓값, 평균값을 계산하여 순서대로 튜플로 반환하는 함수를 **lambda**를 이용해 구현

● 실행 결과 예시

- 람다 함수를 정의해서 변수 *g*로 할당한 후 *g*를 실행

```
>>> g([1, 7, -5])  
(-5, 7, 1.0)
```



Lab : 람다 함수 (4)



● 람다 함수 만들기

- 알파벳으로 이루어진 임의의 문자열을 입력받아, 문자열의 길이가 홀수라면 문자열의 홀수번째 글자들만 반환하고, 문자열의 길이가 짝수라면 문자열의 짝수번째 글자들만 반환하는 함수를 **lambda**를 이용해 구현

● 실행 결과 예시

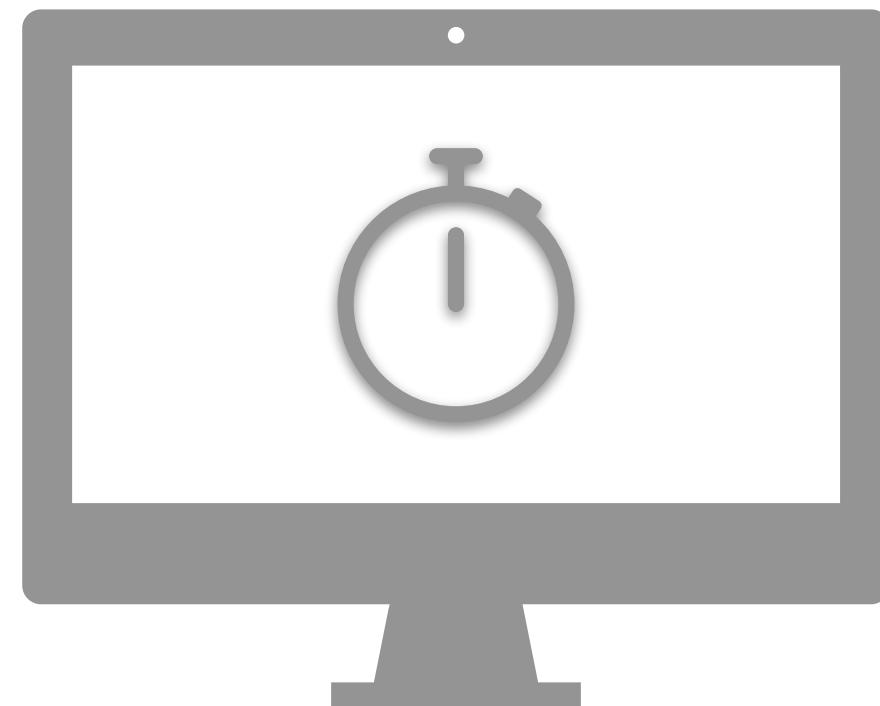
- 람다 함수를 정의해서 변수 **g**로 할당한 후 **g**를 실행

```
>>> g('I love python')
'Ilv yhn'

>>> g('I love programming')
' oepormig'
```



값 반환 함수와 보이드 함수





Lab : 출력과 값 반환을 동시에 처리하는 함수



● 출력과 값 반환을 동시에 처리하는 함수 구현하기

- 실행 시 '==== 나는 함수다 ==='라는 문장을 출력한 후, '나는 함수다!'라는 문장을 반환하는 **i_am_function** 함수를 구현

- 따라서

- ▶ `>>> x = i_am_function()`

- ▶ 처럼 실행할 때, **x** 에 '나는 함수다!'라는 문장이 할당(저장)되어야 한다

- 실행 결과 예시

```
>>> i_am_function()
==== 나는 함수다 ===
'나는 함수다!'

>>> x = i_am_function()
==== 나는 함수다 ===

>>> print(x)
나는 함수다!
```



● 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 = 10을 넣고 결과를 테스트
- 실행 결과 예시

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



● 계산기 구현하기

- 연산자와 정수 두 개를 입력받아 수치 연산을 수행한 후 결과를 반환하는 **calculator** 함수를 구현
- 함수를 정의할 때 세 개의 매개함수를 아래 순서로 정의

- operator, integer1, integer2**

- 만약 **operator**가 '**add**'이면, 두 정수에 더하기 연산을 수행 : **integer1 + integer2**
 - 만약 **operator**가 '**sub**'이면, 두 정수에 빼기 연산을 수행 : **integer1 - integer2**
 - 만약 **operator**가 '**mul**'이면, 두 정수에 곱하기 연산을 수행 : **integer1 * integer2**
 - 만약 **operator**가 '**div**'이면, 두 정수에 나누기 연산을 수행 : **integer1 / integer2**

- 실행 결과 예시**

```
>>> calculator('addition', 3, 5)
addition is not supported.
```

```
>>> calculator('add', 3, 5)
8
```

```
>>> calculator('sub', 7, 8)
-1
```

```
>>> calculator('mul', 11, 2)
22
```

```
>>> calculator('div', 7, 3)
2.3333333333333335
```

```
>>> calculator('add', '3', '5')      # 문자열 더하기
'35'
```

```
>>> calculator('add', [1, 2], [3, 4]) # 리스트 더하기
[1, 2, 3, 4]
```



Lab : 영어 모음 개수를 세는 함수



- 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
```



● 문자열을 검색하는 함수 구현하기

- 문장(혹은 단어)과 문자를 입력받은 후 입력받은 문자와 같은 문자를 문장(혹은 단어)에서 (좌에서 우로) 찾아 첫 번째 검색한 문자의 위치(인덱스)를 반환하는 **search_char** 함수를 구현
- 이 함수는 다음과 같이 두 개의 매개변수를 전달인자로 받아들인다
 - 첫 번째 매개변수 : 문장 혹은 단어
 - 두 번째 매개변수 : 검색할 문자
- 문자가 문장(혹은 단어) 내에 있으면 그 인덱스를 반환하고 발견하지 못하면 **-1**을 반환
 - 영어는 대소문자 구분해야한다

● 실행 결과 예시

```
>>> search_char('python', 'o')
4

>>> search_char('I love Python', 'o')
3

>>> search_char('python', 'P')
-1
```

```
>>> search_char('안녕 파이썬', '파이썬')
3

>>> search_char('바나나', '나')
1
```



Lab : 리스트나 튜플 내의 객체 위치 검색 함수



● 리스트(튜플) 내의 항목(객체)을 찾아 주는 함수 구현하기

- 리스트나 튜플 내에 특정 객체가 들어 있는지를 확인하는 **search** 함수를 구현
- 이 함수는 다음과 같이 두 개의 매개변수를 전달인자로 받아들인다
 - 첫 번째 매개변수 : 리스트 혹은 튜플
 - 두 번째 매개변수 : 찾고자 하는 객체
- 리스트나 튜플 내에 특정 객체가 있으면 그 인덱스를 반환하고, 들어 있지 않으면 **False**를 반환
- 영어는 대소문자 구분해야한다

● 실행 결과 예시

```
>>> mylist = [1, 2, 3]
>>> search(mylist, 2)
1
>>> search(mylist, 10)
False
```

```
>>> mytuple = ('a', 'b', 'c')
>>> search(mytuple, 'c')
2
>>> search(mytuple, '1')
False
```



Lab : 숫자만 골라 합과 평균을 구하는 함수



- 숫자만 골라 합과 평균을 구하는 함수 구현하기

- 리스트나 튜플로 구성되어 있는 데이터를 전달인자로 받아서 이 데이터가 담고 있는 객체가 숫자면 따로 추출하여 자유롭게 정한 자료형에 담은 후 이들의 합과 평균을 튜플 형태로 반환하는 **pick_numbers** 함수를 구현
- 평균은 소수점 **2**자리로 반올림한다

- 실행 결과 예시

```
>>> data = [1, 2.5, 3, '4', 5, 6, 'seven', (8, 9.9), -2, 11]
>>> pick_numbers(data)
(26.5, 3.79)
```



- 단어의 길이가 긴 순서부터 정렬하는 함수 구현하기

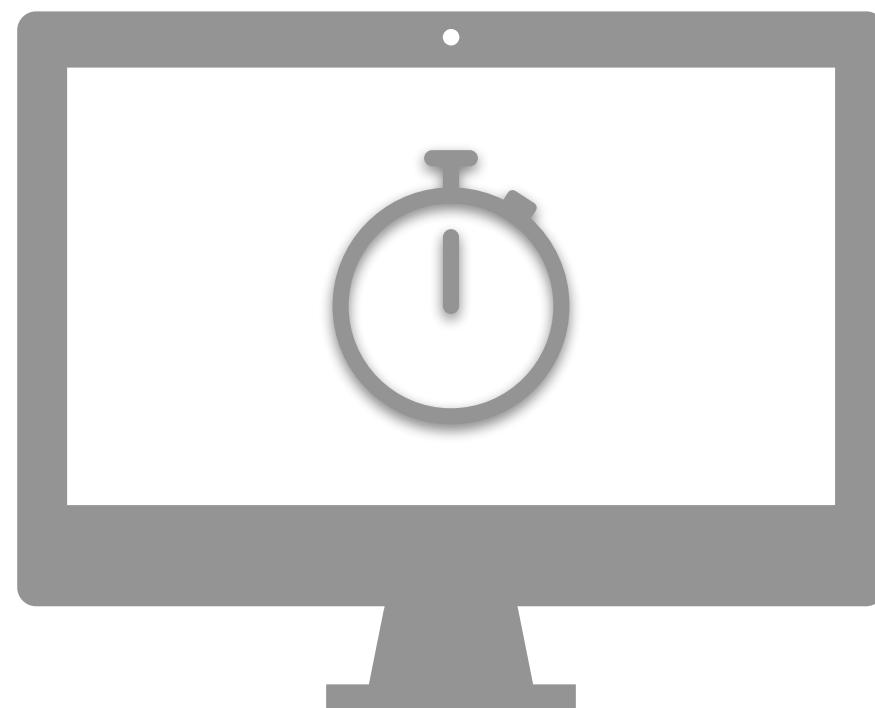
- 한 개 이상의 단어를 가진 리스트나 튜플을 전달받아서 단어의 길이가 긴 순서부터 정렬해서 리스트로 반환하는 `sort_by_word_length` 함수를 구현
- 만약 단어의 길이가 같다면 단어를 내림차순으로 정렬

- 실행 결과 예시

```
>>> colors = 'red', 'blue', 'green', 'brown', 'gray'  
>>> sort_by_word_length(colors)  
['green', 'brown', 'gray', 'blue', 'red']  
  
>>> instruments = ['피아노', '바이올린', '첼로', '기타', '드럼']  
>>> sort_by_word_length(instruments)  
['바이올린', '피아노', '첼로', '드럼', '기타']
```



매개변수와 전달인자





● 점수를 정렬하는 함수 구현하기

- 한 개 이상의 점수(0점 이상 100점 이하)를 받아 가장 높은 점수부터 가장 낮은 점수까지 차례대로 출력하는 `sort_scores` 함수를 구현

● 실행 결과 예시

```
>>> sort_scores(100, 27, 65, 88, 46, 97, 75, 53)
100
97
88
75
65
53
46
27
```



● 기술통계 함수 구현하기

- 한 개 이상의 임의의 정수나 실수를 전달받아 이들 숫자의 총 개수, 합, 평균, 최댓값, 최솟값을 튜플로 반환하는 **stat_summary** 함수를 구현
- 정수나 실수가 아닌 다른 형태의 자료형을 전달받으면, '잘못된 자료형입니다.'를 출력하면서 **None**을 반환

● 실행 결과 예시

```
>>> result = stat_summary(1, 3, 5.5, 7, 9, -2)
>>> result
(6, 23.5, 3.9166666666666665, 9, -2)

>>> result = stat_summary(1.1, '3', 12, -100)
잘못된 자료형입니다.
>>> result

>>> result = stat_summary(1, 2, 3, 4, 5, 'a')
잘못된 자료형입니다.
>>> print(result)
None
```



선택적으로 문자열을 뒤집어 출력하는 함수 구현하기

- 임의의 단어(또는 문장)을 전달받아 만약 해당 단어(또는 문장)의 길이가 홀수면 거꾸로 뒤집어 출력하는 `reverse_odd_text` 함수를 구현
- 입력한 모든 문자열을 출력한 후(이때 길이가 짝수면 그대로 출력), 거꾸로 뒤집은 횟수(즉, 홀수 길이 문자열의 수)를 반환

실행 결과 예시

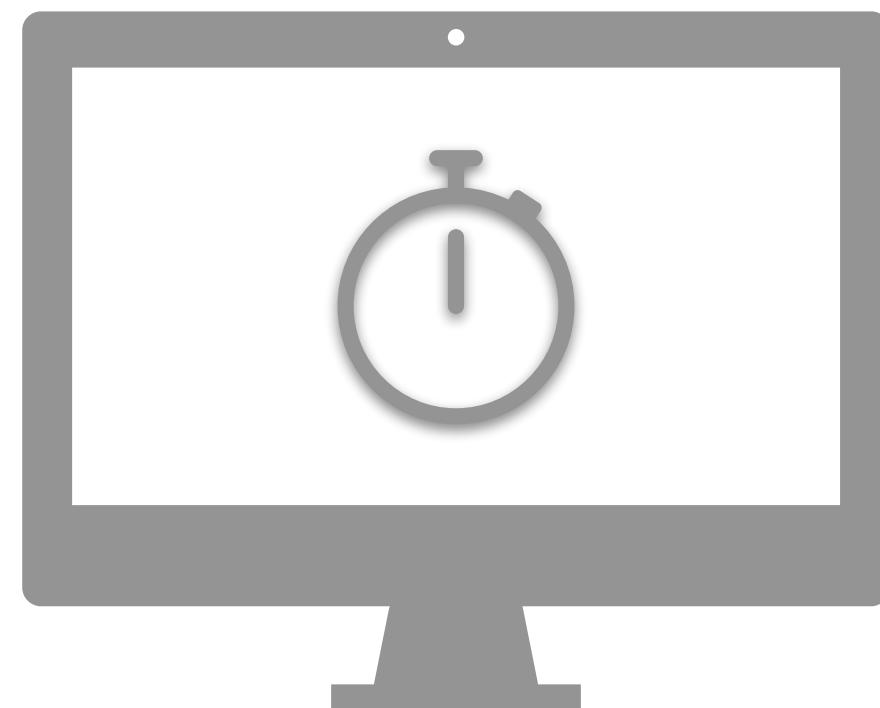
```
>>> odd_count = reverse_odd_text()
>>> odd_count
0

>>> odd_count = reverse_odd_text('red', 'blue', 'green', 'yellow', 'purple', 'black', 'white')
der
blue
neerg
yellow
purple
kcalb
etihw
>>> odd_count
4

>>> odd_count = reverse_odd_text('나는 드러머입니다', '베이스', '기타', '키보드', '보컬')
다니입머러드 는나
스이베
기타
드보키
보컬
>>> odd_count
3
```



파이썬 표준 라이브러리





● collections.Counter 클래스 활용

- ◉ `collections.Counter` 클래스를 사용하여 아래 리스트에 속한 객체(항목) 중 3회 이상(3회 포함) 중복된 값을 가진 항목을 출력

- `L = [1, 1, 7, 7, 7, 4, 4, 4, 2, 1, 5, 5, 9, 11, 3, 'a', 'x', 9, 8, 'b', 'b', 'z', 'b']`

- ◉ 참고

- <https://docs.python.org/3/library/collections.html?highlight=counter#collections.Counter>

- ◉ 실행 결과 예시

```
> python counter_more_than_twice.py
1
7
4
b
```



Lab : 비밀번호 조합



● 자물쇠의 비밀번호 조합 개수 구하기

- 두 개의 문자와 두 개의 숫자로 이루어져 있는 비밀번호 기능을 제공하는 자물쇠

- e.g., E-k-2-5

- 아래 조건을 만족하는 모든 비밀번호의 가능한 조합의 총 개수를 구한다

- 첫 번째 문자는 모음 대문자 중 하나
 - 두 번째 문자는 'a'부터 'e'까지 문자 중 하나
 - 대소문자 모두 가능
 - 세 번째 숫자는 5 미만 소수(prime number)만 가능
 - 네 번째 숫자는 6 이하 홀수

- 실행 결과 예시

```
> python lock_password.py  
?
```



● math 모듈 활용하기

- math 모듈에 있는 함수를 사용해서 *y*를 계산한 후 그 결과 값을 출력

$$y = e^{\pi} + 10$$

● 참고

- <https://docs.python.org/3/library/math.html?highlight=math#module-math>

● 실행 결과 예시

```
> python exp_pi.py  
33.14069263277926
```