

데이터 과학을 위한 파이썬 프로그래밍

2판



Chapter 05

함수



목차

1. 함수 기초
2. 함수 심화
3. 함수의 인수
4. 좋은 코드를 작성하는 방법

학습목표

- 함수를 선언하는 방법, 함수의 실행 순서, 함수의 4가지 형태에 대해 알아본다.
- 함수를 호출하는 방식과 변수의 사용 범위에 대해 학습하고, 재귀 함수에 대해 이해한다.
- 함수의 인수인 키워드 인수, 디폴트 인수, 가변 인수, 키워드 가변 인수에 대해 알아본다.
- 좋은 코드의 의미를 이해하고, 코딩 규칙과 함수 개발 가이드라인에 대해 학습한다.

01

함수 기초

1. 함수의 개념과 장점

- 함수(function): 어떤 일을 수행하는 코드의 덩어리, 또는 코드의 묶음
- 함수의 장점
 - ① 필요할 때마다 호출 가능
 - ② 논리적인 단위로 분할 가능
 - ③ 코드의 캡슐화

2. 함수의 선언

■ 파이썬에서 함수를 선언하는 방법

```
def 함수 이름 (매개변수 #1 ...):  
    명령문 1  
    명령문 2  
    return <반환값>
```

- ① **def**: 'definition'의 줄임말로 함수의 정의를 시작한다는 의미
- ② **함수 이름**: 파이썬에서는 규칙
 - 소문자 입력
 - 띄어쓰기를 할 경우에는 _ 기호 사용
 - 작업을 나타내기 위해 동사와 명사를 함께 사용하는 경우 가 많음
 - 외부에 공개하는 함수일 경우 줄임말을 사용하지 않고 짧고 명료한 이름으로 정함

2. 함수의 선언

- ③ 매개변수(parameter): 매개변수는 함수에서 입력값으로 사용하는 변수를 의미하며, 1개 이상의 값을 적을 수 있음.
- ④ 명령문: 명령문은 반드시 들여쓰기한 후 코드를 입력해야 함.

- 간단한 함수 선언의 작성 예시

```
def calculate_rectangle_area(x, y)
    return x * y
```

- 함수 이름: calculate_rectangle_area
- 매개변수: x와 y
- return: 값을 반환한다는 뜻으로, x와 y를 곱한 값을 반환하는 함수

2. 함수의 선언

여기서 잠깐! 반환

약간 어렵게 느껴질 수 있는 부분이 바로 '반환'이라는 개념이다. 이는 수학에서 함수와 같은 개념이라고 생각하면 된다. 예를 들어, 수학에서 $f(x) = x + 1$ 이라고 했을 때 $f(1)$ 의 값은 얼마일까? 중학교 정도의 수학을 이해하고 있다면 $f(1) = 2$ 라는 것을 알 수 있다. 즉, 함수 $f(x)$ 에서 x 에 1이 들어가면 2가 반환되는 것이다. 파이썬의 함수도 같은 개념이다. 수학에서 x 에 해당하는 것이 매개변수, 즉 입력값이고, $x + 1$ 의 계산 과정이 함수 안의 코드이며, 그 결과가 반환되는 값이다.

3. 함수의 실행 순서

[코드 5-1]

```
1 def calculate_rectangle_area(x, y):  
2     return x * y  
3  
4 rectangle_x = 10  
5 rectangle_y = 20  
6 print("사각형 x의 길이:", rectangle_x)  
7 print("사각형 y의 길이:", rectangle_y)  
8  
9 # 넓이를 구하는 함수 호출  
10 print("사각형의 넓이:", calculate_rectangle_area(rectangle_x, rectangle_y))
```

[실행결과]

```
사각형 x의 길이: 10  
사각형 y의 길이: 20  
사각형의 넓이: 200
```

4. 프로그래밍의 함수와 수학의 함수

- 프로그래밍에서의 함수와 수학에서의 함수는 매우 비슷
- 간단히 $f(x) = x + 1$ 을 코드로 나타낸다면 [그림 5-1]과 같은 형태임

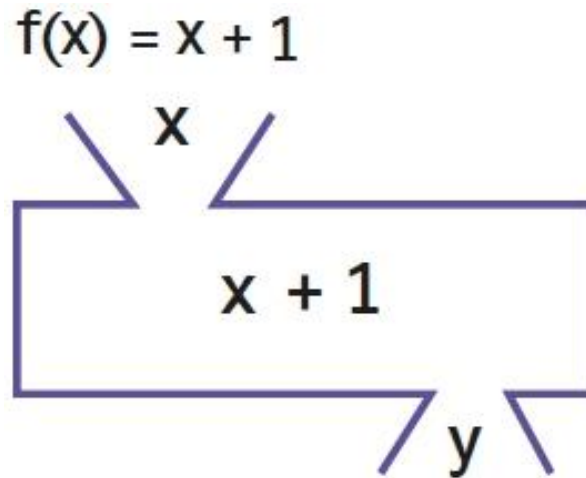


그림 5-1 수학에서 함수의 형태

4. 프로그래밍의 함수와 수학의 함수

- 실제로 다음과 같은 문제가 있다면 프로그래밍에서는 어떻게 표현할 수 있을까?

$f(x) = 2x + 7$, $g(x) = x^2$ 이고 $x = 2$ 일 때

$f(x) + g(x) + f(g(x)) + g(f(x))$ 의 값은?

$f(2) = 11$, $g(2) = 4$, $f(g(x)) = 15$, $g(f(x)) = 121$

$\therefore 11 + 4 + 15 + 121 = 151$

4. 프로그래밍의 함수와 수학의 함수

- 함수에 해당하는 $f(x)$ 와 $g(x)$ 의 내용을 코드로 작성하기


[코드 5-2]

```
1 def f(x):  
2     return 2 * x + 7  
3 def g(x):  
4     return x ** 2  
5 x = 2  
6 print(f(x) + g(x) + f(g(x)) + g(f(x)))
```

[실행결과]

151

4. 프로그래밍의 함수와 수학의 함수

여기서  잠깐! 매개변수와 인수

함수의 입력값은 매개변수로 구분되어 사용한다. 함수와 관련해 몇 가지 용어에 대한 이해가 필요한데, 대표적인 것이 매개변수(parameter)와 인수(argument)이다. 이 둘의 차이를 이해하기 위해 [코드 5-3]을 보자.

코드 5-3

parameter.py

```
1 def f(x):  
2     return 2 * x + 7  
3  
4 print(f(2))
```

11

위 코드에서 'def f(x):'의 x를 매개변수라고 한다. 일반적으로 함수의 입력값에 대한 정의를 함수 사용에 있어 인터페이스를 정의한다고 한다. 매개변수는 함수의 인터페이스 정의에 있어 어떤 변수를 사용하는지 정의하는 것이다. 즉, 위 함수에서는 x가 해당 함수의 매개변수이다. 그에 반해 인수는 실제 매개변수에 대입되는 값을 뜻한다. 매개변수가 설계도라면 인수는 그 설계도로 지은 건물 같은 것이다. 위 코드에서는 f(2)에서 2가 인수에 해당한다. 실제로 매개변수와 인수는 구분 없이 사용하는 경우가 많으며, 모두 함수의 입력값으로 부르기도 한다. 하지만 정확한 의미를 파악하고 사용하는 것을 권한다.

5. 함수의 형태

- 매개변수와 반환값(return value)의 유무에 따라 함수를 형태로 구분함

표 5-1 함수의 형태

매개변수 유무 반환값 유무	매개변수 없음	매개변수 있음
반환값 없음	함수 내부 명령문만 수행	매개변수를 사용하여 명령문만 수행
반환값 있음	매개변수 없이 명령문을 수행한 후 결과값 반환	매개변수를 사용하여 명령문을 수행한 후 결과값 반환

5. 함수의 형태

[코드 5-4]

```
1 def a_rectangle_area():           # 매개변수 × , 반환값 ×
2     print(5 * 7)
3 def b_rectangle_area(x, y):       # 매개변수 ○ , 반환값 ×
4     print(x * y)
5 def c_rectangle_area():           # 매개변수 × , 반환값 ○
6     return(5 * 7)
7 def d_rectangle_area(x , y):      # 매개변수 ○ , 반환값 ○
8     return(x * y)
9
10 a_rectangle_area()
11 b_rectangle_area(5, 7)
12 print(c_rectangle_area())
13 print(d_rectangle_area(5, 7))
```

[실행결과]

```
35
35
35
35
```


02

함수 심화

1. 함수의 호출 방식

- 함수에서는 변수를 어떻게 호출할까?

[코드 5-5]

```
1 def f(x):  
2     y = x  
3     x = 5  
4     return y * y  
5  
6 x = 3  
7 print(f(x))  
8 print(x)
```

[실행결과]

```
9  
3
```

1. 함수의 호출 방식

- 3행과 6행에서 함수 $f(x)$ 의 x 에 5와 3이 입력됨.
- 함수 안에서의 x 와 함수 밖에서의 x 는 같은 변수일까, 아니면 다른 변수일까?
 - ☞ if 문을 사용할 때 키워드 `is`가 변수들의 메모리 주소를 비교함. 즉, 함수 밖에 있는 변수 x 의 메모리 주소와 함수 안에 있는 변수 x 의 메모리 주소가 같은지, 다른지 확인해야 함.

표 5-2 함수가 변수를 호출하는 방식

종류	설명
값에 의한 호출 (call by value)	<ul style="list-style-type: none">• 함수에 인수를 넘길 때 값만 넘김• 함수 내부의 인수값 변경 시 호출된 변수에 영향을 주지 않음
참조 호출 (call by reference)	<ul style="list-style-type: none">• 함수에 인수를 넘길 때 메모리 주소를 넘김• 함수 내부의 인수값 변경 시 호출된 변수값도 변경됨

1. 함수의 호출 방식

- **메모리 주소:** 변수가 저장되는 공간으로 그 공간 자체에 새로운 값을 할당하면 그 공간을 가리키고 있는 다른 변수에도 영향을 줌
- [코드 5-5]에서 만약 참조 호출로 적용된다면 맨 마지막에 있는 x의 값은 5로 변환되어야 하지만 파이썬은 객체의 주소가 함수로 넘어간다는 뜻의 객체 호출(call by object reference) 방식을 사용함.

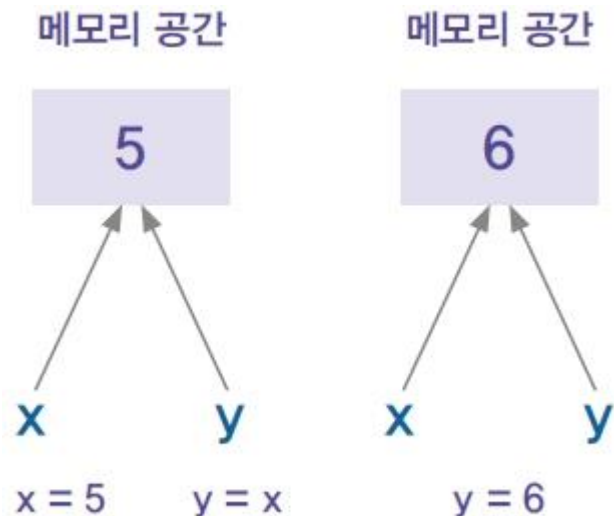


그림 5-2 파이썬에서 변수를 호출하는 방식

1. 함수의 호출 방식

[코드 5-6]

```
1 def spam(eggs):  
2     eggs.append(1)  
3     eggs = [2, 3]  
4  
5 ham = [0]  
6 spam(ham)  
7 print(ham)
```

기존 객체의 주소값에 [1] 추가
새로운 객체 생성

[실행결과]

[0, 1]

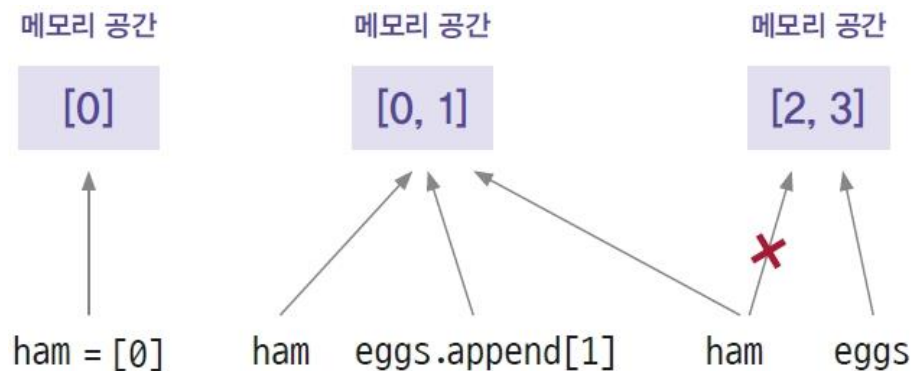


그림 5-3 객체 호출 방식

2. 변수의 사용 범위

- 변수의 사용 범위(scoping rule): 변수가 코드에서 사용되는 범위
- 변수의 사용 범위를 결정할 때 고려해야 할 두 가지 변수
 - ① 지역 변수(local variable): 함수 내부에서만 사용
 - ② 전역 변수(global variable): 프로그램 전체에서 사용

2. 변수의 사용 범위

[코드 5-7]

```
1 def test(t):
2     print(x)
3     t = 20
4     print("In Function:", t)
5
6 x = 10
7 test(x)
8 print("In Main:", x)
9 print("In Main:", t)
```

[실행결과]

```
10
In function: 20
In Main: 10
Traceback (most recent call last):
  File "scoping_rule.py", line 9, in <module>
    print("In Main:", t)
NameError: name 't' is not defined
```

2. 변수의 사용 범위

[코드 5-8]

```
1 def f():  
2     s = "I love London!"  
3     print(s)  
4  
5 s = "I love Paris!"  
6 f()  
7 print(s)
```

[실행결과]

```
I love London!  
I love Paris!
```


2. 변수의 사용 범위

[코드 5-9]

```
1 def f():  
2     global s  
3     s = "I love London!"  
4     print(s)  
5  
6 s = "I love Paris!"  
7 f()  
8 print(s)
```

[실행결과]

```
I love London!  
I love London!
```

2. 변수의 사용 범위

[코드 5-10]

```
1 def calculate(x, y):  
2     total = x + y           # 새로운 값이 할당되어 함수 내부 total은 지역 변수가 됨  
3     print("In Function")  
4     print("a:",str(a),"b:",str(b),"a + b:",str(a + b),"total:",str(total))  
5     return total  
6  
7 a = 5                       # a와 b는 전역 변수  
8 b = 7  
9 total = 0                   # 전역 변수 total  
10 print("In Program - 1")  
11 print("a:", str(a), "b:", str(b), "a + b:", str(a + b))  
12  
13 sum = calculate (a, b)  
14 print("After Calculation")  
15 print("Total:", str(total), " Sum:", str(sum))  
    # 지역 변수는 전역 변수에 영향을 주지 않음
```

2. 변수의 사용 범위

[실행결과]

```
In Program - 1  
a: 5 b: 7 a + b: 12  
In Function  
a: 5 b: 7 a + b: 12 total: 12  
After Calculation  
Total : 0 Sum: 12
```

3. 재귀 함수

- 재귀 함수(recursive function): 자기 자신을 다시 호출하는 함수
 - 재귀적이라는 표현은 자신을 이용해 다른 것을 정의한다는 뜻.

$$n! = n \cdot (n-1) \cdots 2 \cdot 1 = \prod_{i=1}^n i$$
$$\begin{aligned} 1! &= 1 \\ 2! &= 2(1) = 2 \\ 3! &= 3(2)(1) = 6 \\ 4! &= 4(3)(2)(1) = 24 \\ 5! &= 5(4)(3)(2)(1) = 120 \end{aligned}$$

그림 5-4 점화식

3. 재귀 함수

[코드 5-11]

```
1 def factorial(n):  
2     if n == 1:  
3         return 1  
4     else:  
5         return n * factorial(n - 1)  
6  
7 print(factorial(int(input("Input Number for Factorial Calculation: "))))
```

[실행결과]

```
Input Number for Factorial Calculation: 5  
120
```

← 사용자 입력
← 화면 출력

3. 재귀 함수

- factorial() 함수는 n이라는 변수를 입력 매개변수로 넘겨받은 후 $n == 1$ 이 아닐 때까지 입력된 n과 n에서 1을 뺀 값을 입력 값으로 다시 factorial() 함수를 호출하여 반환된 값과 곱함.

- 사용자가 5를 입력했을 때의 계산

```
5 * factorial(5 - 1)
= 5 * 4 * factorial(4 - 1)
= 5 * 4 * 3 * factorial(3 - 1)
= 5 * 4 * 3 * 2 * factorial(2 - 1)
= 5 * 4 * 3 * 2 * 1
```

- [코드 5-11]은 for문이나 while문으로도 표현할 수 있음
 - 재귀 함수의 기본 구조가 종료 조건, 단계별 반환으로 구성되어 있으므로 크게 변경 없이도 사용할 수 있음.
 - 반복문에서도 종료 조건과 반복문마다 동일한 연산이 진행되기 때문.

03

함수의 인수

- **함수의 인수(argument):** 함수의 입력으로 들어가는 변수의 다양한 형태

표 5-3 파이썬에서 인수를 사용하는 방법

종류	내용
키워드 인수	함수의 인터페이스에서 지정한 변수명을 사용하여 함수의 인수를 지정하는 방법
디폴트 인수	별도의 인수값이 입력되지 않을 때 인터페이스 선언에서 지정한 초기값을 사용하는 방법
가변 인수	함수의 인터페이스에서 지정한 변수 이외의 추가 변수를 함수에 입력할 수 있도록 지원하는 방법
키워드 가변 인수	매개변수의 이름을 따로 지정하지 않고 입력하는 방법

1. 키워드 인수

- **키워드 인수(keyword arguments):** 함수에 입력되는 매개변수의 변수명을 사용하여 함수의 인수를 지정하는 방법

[코드 5-12]

```
1 def print_something(my_name, your_name):  
2     print("Hello {0}, My name is {1}".format(your_name, my_name))  
3  
4 print_something("Sungchul", "TEAMLAB")  
5 print_something(your_name = "TEAMLAB", my_name = "Sungchul")
```

[실행결과]

```
Hello TEAMLAB, My name is Sungchul  
Hello TEAMLAB, My name is Sungchul
```

2. 디폴트 인수

- **디폴트 인수(default arguments):** 매개변수에 기본값을 지정하여 사용하고, 아무런 값도 인수로 넘어가지 않을 때 지정된 기본값을 사용하는 방식

[코드 5-13]

```
1 def print_something_2(my_name, your_name = "TEAMLAB"):  
2     print("Hello {0}, My name is {1}".format(your_name, my_name))  
3  
4 print_something_2("Sungchul", "TEAMLAB")  
5 print_something_2("Sungchul")
```

[실행결과]

```
Hello TEAMLAB, My name is Sungchul  
Hello TEAMLAB, My name is Sungchul
```

3. 가변 인수

- **가변 인수(variable-length arguments):** *(asterisk라고 부름)로 표현
 - *는 파이썬에서 기본적으로 곱셈 또는 제곱 연산 외에도 변수를 묶어주는 가변 인수를 만들 때 사용함.

[코드 5-14]

```
1 def asterisk_test(a, b, *args):  
2     return a + b + sum(args)  
3  
4 print(asterisk_test(1, 2, 3, 4, 5))
```

[실행결과]

15

3. 가변 인수

[코드 5-15]

```
1 def asterisk_test(a, b, *args):  
2     print(args)  
3  
4 print(asterisk_test(1, 2, 3, 4, 5))
```

[실행결과]

```
(3, 4, 5)  
NONE
```

- **튜플(tuple) 자료형:** 괄호로 묶여 출력되는 자료형 → 리스트 자료형처럼 인덱스로 접근할 수 있는 자료형

3. 가변 인수

[코드 5-16]

```
1 def asterisk_test_2(*args):  
2     x, y, *z = args  
3     return x, y, z  
4  
5 print(asterisk_test_2(3, 4, 5))
```

[실행결과]

```
(3, 4, [5])
```

- 입력 받은 가변 인수의 개수를 정확히 안다면, `x, y, *z = args`처럼 언패킹 가능
- `*z`가 아닌 상태에서 `asterisk_test_2(3, 4, 5, 10, 20)`으로 변경하여 코드를 실행하면 언패킹의 개수가 맞지 않기 때문에 오류 발생

3. 가변 인수

- 언패킹 코드를 `x, y, *z = args`로 변경한 코드

[코드 5-17]

```
1 def asterisk_test_2(*args):  
2     x, y, *z = args  
3     return x, y, z  
4  
5 print(asterisk_test_2(3, 4, 5, 10, 20))
```

[실행결과]

```
(3, 4, [5, 10, 20])
```

- *는 기능이 다양하여 언패킹할 때도 값을 가변 인수의 형태로 받을 수 있음

4. 키워드 가변 인수

- 키워드 가변 인수(keyword variable-length arguments): 매개변수의 이름을 따로 지정하지 않고 입력하는 방법

[코드 5-18]

```
1 def kwargs_test(**kwargs):  
2     print(kwargs)  
3     print("First value is {first}".format(**kwargs))  
4     print("Second value is {second}".format(**kwargs))  
5     print("Third value is {third}".format(**kwargs))  
6  
7 kwargs_test(first = 3, second = 4, third = 5)
```

[실행결과]

```
{'first': 3, 'second': 4, 'third': 5}  
First value is 3  
Second value is 4  
Third value is 5
```

4. 키워드 가변 인수

- 딕셔너리 자료형 변수에 * 2개 붙이면 개별 변수로 풀리면서 함수에 들어갈 수 있음.
- 인터 프리터는 다음과 같이 해석함.

```
>>> kwargs = {'first': 3, 'second': 4, 'third': 5}
>>> print("Second value is {second}".format(**kwargs))
Second Value is 4
>>> print("Second value is {second}".format(first = 3,second = 4,third = 5))
Second Value is 4
```


4. 키워드 가변 인수

- 일반적으로 매개변수, 가변 인수, 키워드 가변 인수를 모두 사용하면 매개변수, 가변 인수, 키워드 가변 인수 순으로 각 값을 적당한 위치의 변수에 할당
- 다음 코드에서 3, 4는 각각 one, two에 할당되고, 나머지 5, 6, 7, 8, 9는 args에, first = 3, second = 4, third = 5는 딕셔너리형으로 kwargs에 할당됨.

```
>>> def kwargs_test(one, two, *args, **kwargs):  
...     print(one + two + sum(args))  
...     print(kwargs)  
...  
>>> kwargs_test(3, 4, 5, 6, 7, 8, 9, first = 3, second = 4, third = 5)  
42  
{'first': 3, 'second': 4, 'third': 5}
```

04

좋은 코드를 작성하는 방법

1. 좋은 코드의 의미

- 프로그래밍은 사실 팀플레이(team play)로 여러 사람과 함께 프로그래밍하는 경우가 훨씬 많으므로 좋은 프로그래밍을 위한 규칙이 있어야 함.



그림 5-5 메타 플랫폼의 사무실

1. 좋은 코드의 의미

- 여러 사람과 함께 일하기 때문에 프레젠테이션 발표를 하듯 사람들과 소통하면서 프로그래밍 해야 함

“컴퓨터가 이해할 수 있는 코드는 어느 바보나 다 짤 수 있다. 좋은 프로그래머는 사람이 이해할 수 있는 코드를 짤다.” - 마틴 파울러

- 다른 사람이 내가 작성한 코드를 쉽게 이해할 수 있도록 프로그램을 작성해야 함
- ☞ 프로그램 코드는 많은 사람이 쉽게 읽고 이해할 수 있도록 가독성이 좋아야 함

2. 코딩 규칙

- 코딩 규칙(coding convention): 프로그래밍에서 여러 사람의 이해를 돕기 위한 규칙
- 파이썬의 기본 코딩 규칙
 - 들여쓰기는 4 스페이스
 - 한 줄은 최대 79자까지
 - 불필요한 공백은 없애기

2. 코딩 규칙

- **PEP 8 (Python Enhance Proposal 8):** 파이썬 개발자들이 앞으로 필요한 파이썬의 기능이나 여러 가지 부수적인 것을 정의한 문서

- **PEP 8의 규칙**

- = 연산자는 1칸 이상 띄우지 않는다.

```
variable_example = 12      # 필요 이상으로 빈칸이 많음  
variable_example = 12      # 정상적인 띄어쓰기
```

- 주석은 항상 갱신하고 불필요한 주석은 삭제한다.
- 소문자 l, 대문자 O, 대문자 I는 사용을 금한다.

```
lI00 = "Hard to Understand"    # 변수를 구분하기 어려움
```

- 함수명은 소문자로 구성하고, 필요하면 밑줄로 구분한다

2. 코딩 규칙

여기서  잠깐! flake8 모듈

코딩을 한 후, 코딩 규칙을 제대로 지켰는지 확인하는 방법 중 하나는 flake8 모듈로 체크하는 것이다. flake8을 설치하기 위해서는 먼저 cmd 창에 다음과 같이 입력한다.

```
conda install -c anaconda flake8
```

비주얼 스튜디오 코드에서 [코드 5-19]와 같이 작성한 후, 'test_flake.py'로 저장한다.

코드 5-19

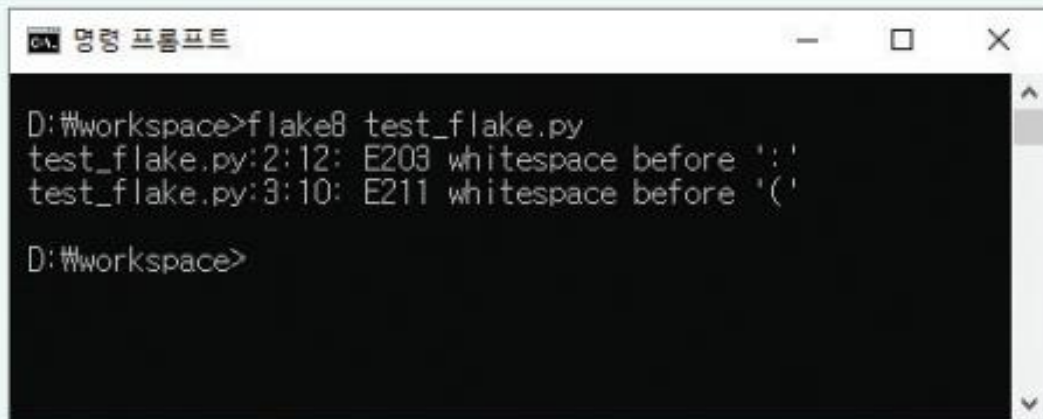
test_flake.py

```
1 lL00 = "123"
2 for i in 10 :
3     print("Hello")
```

그리고 cmd 창에 다음과 같이 입력하면, 각 코드의 수정 방법을 알려준다.

2. 코딩 규칙

```
flake8 test_flake.py
```



```
명령 프롬프트
D:\workspace>flake8 test_flake.py
test_flake.py:2:12: E203 whitespace before ':'
test_flake.py:3:10: E211 whitespace before '('
D:\workspace>
```

그림 5-6 flake8 모듈로 코드 수정 방법 확인

3. 함수 개발 가이드라인

3.1 함수 이름

- 함수 내용은 가능하면 짧게 작성할 것(줄 수를 줄일 것)
- 함수 이름에 함수의 역할과 의도를 명확히 드러낼 것

```
def print_hello_world():  
    print("Hello, World")  
def get_hello_world():  
    return "Hello, World"
```

3. 함수 개발 가이드라인

3.2 함수의 역할

- 하나의 함수에는 유사한 역할을 하는 코드만 포함시켜야 함. 즉 함수는 한 가지 역할을 명확히 해야 함.

```
def add_variables(x, y):
```

```
    return x + y
```

```
def add_variables(x, y):
```

```
    print(x, y)
```

```
    return x + y
```

3. 함수 개발 가이드라인

3.3 함수를 만들어야 하는 경우

- 공통으로 사용되는 코드를 함수로 변환

[코드 5-20]

```
1 a = 5
2 if (a > 3):
3     print("Hello World")
4     print("Hello TEAMLAB")
5 if (a > 4):
6     print("Hello World")
7     print("Hello TEAMLAB")
8 if (a > 5):
9     print("Hello World")
10    print("Hello TEAMLAB")
```

[실행결과]

```
Hello World
Hello TEAMLAB
Hello World
Hello TEAMLAB
```

3. 함수 개발 가이드라인

[코드 5-21]

```
1 def print_hello():
2     print("Hello World")
3     print("Hello TEAMLAB")
4
5 a = 5
6 if (a > 3):
7     print_hello()
8
9 if (a > 4):
10    print_hello()
11
12 if (a > 5):
13    print_hello()
```

[실행결과]

```
Hello World
Hello TEAMLAB
Hello World
Hello TEAMLAB
```

3. 함수 개발 가이드라인

3.3 함수를 만들어야 하는 경우

- 복잡한 로직이 사용되었을 때 식별 가능한 이름의 함수로 변환

[코드 5-22]

```
1 import math
2 a = 1; b = -2; c = 1
3
4 print((-b + math.sqrt(b ** 2 - (4 * a * c))) / (2 * a))
5 print((-b - math.sqrt(b ** 2 - (4 * a * c))) / (2 * a))
```

[실행결과]

```
1.0
1.0
```

3. 함수 개발 가이드라인

[코드 5-23]

```
1 import math
2
3 def get_result_quadratic_equation(a, b, c):
4     values = [ ]
5     values.append((-b + math.sqrt(b ** 2 - (4 * a * c)) ) / (2 * a))
6     values.append((-b - math.sqrt(b ** 2 - (4 * a * c)) ) / (2 * a))
7     return values
8
9 print(get_result_quadratic_equation(1,-2,1))
```

[실행결과]

```
[1.0, 1.0]
```

Thank you!