

데이터 과학을 위한
파이썬
프로그래밍



05. 함수

목차

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

01

함수 기초

01. 함수 기초

■ 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

01. 함수 기초

■ 함수의 선언

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

- ① **def** : 'definition'의 줄임말로, 함수를 정의하여 시작한다는 의미이다.
- ② **함수 이름** : 함수 이름은 개발자가 마음대로 지정할 수 있지만, 파이썬에서는 일반적으로 다음과 같은 규칙을 사용한다.
 - **소문자**로 입력한다.
 - 띄어쓰기를 할 경우에는 **_ 기호**를 사용한다. ex) save_model
 - 행위를 기록하므로 동사와 명사를 함께 사용하는 경우가 많다. ex) find_number
 - 외부에 공개하는 함수일 경우, 줄임말을 사용하지 않고 짧고 명료한 이름을 정한다.

01. 함수 기초

■ 함수의 선언

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

- ③ **매개변수(parameter)** : 함수에서 입력값으로 사용하는 변수를 의미하며, 1개 이상의 값을 적을 수 있다.
- ④ **수행문** : 수행문은 반드시 들여쓰기한 후 코드를 입력해야 한다. 수행해야 하는 코드는 일반적으로 작성하는 코드와 같다. if나 for 같은 제어문을 사용할 수도 있고, 고급 프로그래밍을 하게 되면 함수 안에 함수를 사용하기도 한다.

01. 함수 기초

■ 함수의 선언

- 함수 선언 작성 예시를 간단한 코드로 살펴보자

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

- ① 먼저 선언된 함수를 확인할 수 있다.
- ② 함수 이름은 `calculate_rectangle_area`이고, `x`와 `y`라는 2개의 매개변수를 사용하고 있다.
- ③ `return`의 의미는 값을 반환한다는 뜻으로, `x`와 `y`를 곱한 값을 반환하는 함수로 이해한다.

01. 함수 기초

■ 함수의 실행 순서

코드 5-1 rectangle_area.py

```
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))
```

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


01. 함수 기초

여기서 잠깐! 매개변수와 인수

- **매개변수**는 함수의 인터페이스 정의에 있어 어떤 변수를 사용하는지를 정의하는 것이다. 그에 반해 **인수**는 실제 매개변수에 대입되는 값을 뜻한다.

코드 5-3 parameter.py

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

11

- ➡ [코드 5-3]에서 '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 사용하는지를 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.

01. 함수 기초

■ 함수의 형태

코드 5-4 function_type.py

```
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))
```

```
-
35
35
35
35
```

02

함수 심화

02. 함수 심화

■ 함수의 호출 방식

코드 5-5 call1.py

```
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)
```

9

3

02. 함수 심화

■ 함수의 호출 방식 : [코드 5-5] 해석

- 함수 밖에 있는 변수 x의 메모리 주소와 함수 안에 있는 변수 x의 메모리 주소가 같은지 다른지 확인할 필요가 있다.
- 함수 안에 변수가 인수로 들어가 사용될 때, 변수를 호출하는 방식을 전통적인 프로그래밍에서는 다음과 같이 크게 두 가지로 나눈다.

종류	설명
값에 의한 호출 (call by value)	<ul style="list-style-type: none">• 함수에 인수를 넘길 때 값만 넘김• 함수 안의 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 안의 인수값 변경 시, 호출된 변수값도 변경됨

[함수가 변수를 호출하는 방식]

02. 함수 심화

■ 함수의 호출 방식

- 파이썬은 객체의 주소가 함수로 넘어간다는 뜻으로, **객체 호출(call by object reference)** 로 명명되는 방식을 사용한다.
- 파이썬에서는 새로운 값을 할당하거나 해당 객체를 지울 때는 영향을 주지 않고, 단순히 해당 객체에 값을 추가할 때는 영향을 준다.

02. 함수 심화

■ 함수의 호출 방식

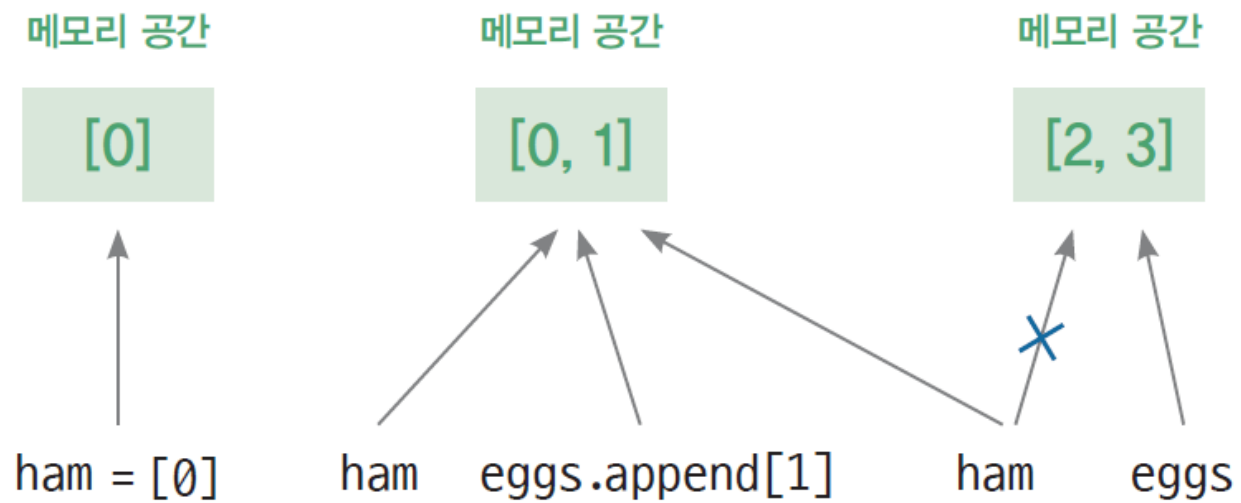
코드 5-6 call2.py

```
1 def spam(eggs):  
2     eggs.append(1)          # 기존 객체의 주소값에 [1] 추가  
  
3     eggs = [2, 3]          # 새로운 객체 생성  
4  
5     ham = [0]  
6     spam(ham)  
7     print(ham)
```

[0, 1]

02. 함수 심화

■ 함수의 호출 방식



[객체 호출 방식]

02. 함수 심화

■ 변수의 사용 범위

- 변수의 사용 범위(scoping rule) : 변수가 코드에서 사용되는 범위
- 지역 변수(local variable) : 함수 안에서만 사용
- 전역 변수(global variable) : 프로그램 전체에서 사용

02. 함수 심화

■ 변수의 사용 범위

코드 5-7 scoping_rule.py

```
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)
```

```
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
```

02. 함수 심화

■ 변수의 사용 범위

코드 5-8 local_variable.py

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

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

함수 안과 밖의 **s**는 같은 이름을 가졌지만,
다른 메모리 주소를 가진 전혀 다른 변수

02. 함수 심화

■ 변수의 사용 범위

- 그렇다면 함수 안의 변수와 함수 밖의 함수가 같은 이름을 사용하기 위해서는 어떻게 해야 할까? 함수 내에서 전역 변수로 선언된 변수를 사용하기 위해서는 **global**이라는 파이썬에서 제공하는 키워드를 사용해야 한다.

코드 5-9 global_variable.py

```
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)
```

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

02. 함수 심화

■ 변수의 사용 범위

코드 5-10 scoping_rule_final.py

```
1 def calculate(x, y):
2     total = x + y          # 새로운 값이 할당되어 함수 안 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
```

```
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
```

03

함수의 인수

03. 함수의 인수

- 파이썬에서 인수를 사용하는 방법에 대해 알아보자

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

[파이썬에서 인수를 사용하는 방법]

03. 함수의 인수

■ 키워드 인수

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

코드 5-12 keyword.py

```
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")
```

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


03. 함수의 인수

■ 디폴트 인수

- **디폴트 인수(default arguments)** : 매개변수에 기본값을 지정하여 사용하고, 아무런 값도 인수로 넘기지 않으면 지정된 기본값을 사용하는 방식이다

코드 5-13 default.py

```
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")
```

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

03. 함수의 인수

■ 가변 인수

- 함수의 매개변수 개수가 정해지지 않고 진행해야 하는 경우가 있다. 이때 사용하는 것이 바로 **가변 인수(variable-length arguments)**이다.
- 가변 인수는 ***(asterisk라고 부름)로 표현**할 수 있는데, *는 파이썬에서 기본적으로 곱셈 또는 제곱 연산 외에도 변수를 묶어 주는 가변 인수를 만든다

03. 함수의 인수

■ 가변 인수

코드 5-14 asterisk1.py

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

15

- ➡ [코드 5-14]의 `asterisk_test()` 함수는 변수 `a`, `b`를 받고, 나머지 변수는 `*args`로 받고 있다. 여기서 **`*args`를 가변 인수**라고 한다. `asterisk_test(1, 2, 3, 4, 5)`에서 1과 2는 각각 `a`와 `b`에 할당되고, 나머지 인수인 3, 4, 5가 모두 `*args`에 할당된다.

03. 함수의 인수

■ 가변 인수

- [코드 5-14]를 [코드 5-15]와 같이 변경한 후 실행하면, 다음과 같은 결과를 얻을 수 있다.

코드 5-15 asterisk2.py

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

```
(3, 4, 5)
None
```

- ➡ [코드 5-15]의 결과값이 괄호로 묶여 출력되는 것을 확인할 수 있다. 이렇게 **괄호로 묶여 출력되는 자료형을 튜플(tuple)**이라고 한다. 가변인수 *는 반드시 일반적인 키워드 인수가 모두 끝난 후 넣어야 한다. 리스트와 비슷한 튜플 형태로 함수 안에서 사용할 수 있으므로 **인덱스**를 사용하여, 즉 `args[0]`, `args[1]` 등으로 변수에 접근할 수 있다.

03. 함수의 인수

■ 가변 인수

- 언패킹 코드를 `x, y, *z = args`로 변경하면 어떤 결과가 나올까?

코드 5-17 asterisk4.py

```
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))
```

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

03. 함수의 인수

■ 키워드 가변 인수

- **키워드 가변 인수(keyword variable-length arguments)**는 매개변수의 이름을 따로 지정하지 않고 입력하는 방법으로, 이전 가변 인수와는 달리 *****를 **2개 사용**하여 함수의 매개변수를 표시한다.
- 입력된 값은 튜플 자료형이 아닌 **딕셔너리 자료형(dictionary type)**으로 사용할 수 있다.
- 키워드 가변 인수는 반드시 모든 매개변수의 맨 마지막, 즉 가변 인수 다음에 선언되어야 한다.

03. 함수의 인수

■ 키워드 가변 인수

코드 5-18 kwargs.py

```
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)
```

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

03. 함수의 인수

■ 키워드 가변 인수 : [코드 5-18] 해석

- 키워드 가변 인수는 변수명으로 kwargs를 사용한다. 변수명 자체는 중요하지 않지만, *는 반드시 ** 이렇게 2개를 붙여야 한다.
- 먼저 print() 함수에는 3개의 키워드 인수를 넣으므로 그 인수들이 {'first': 3, 'second': 4, 'third': 5} 형태로 출력되는 것을 확인할 수 있다. 이러한 형태를 딕셔너리 자료형이라고 하며, 실행 결과 변수명과 값이 쌍으로 저장된 것을 확인할 수 있다.
- 4행의 print("Secondvalue is {second}".format(**kwargs))와 같이 개별 변수명을 따로 불러내 사용할 수도 있는데, 이 코드는 print() 함수에서 사용하는 출력 기능을 사용하여 변수 kwargs에 있는 second 변수를 print() 함수에서 사용할 수 있도록 하였다. 이미 키워드 가변 인수를 사용하여 'first = 3, second = 4, third = 5' 변수를 함수 안에 넣었기 때문에 second라는 변수를 사용할 수 있다.
- 이전 가변 인수에서 보았듯이 딕셔너리 자료형 변수에 *를 2개 붙이면, 개별 변수로 풀려 함수에 들어갈 수 있다.

03. 함수의 인수

■ 키워드 가변 인수

- 인터프리터는 다음과 같이 해석한다.

```
>>> 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
```

03. 함수의 인수

■ 키워드 가변 인수

- 다음 코드에서 3, 4는 각각 one, two에 할당되고, 나머지 5, 6, 7, 8, 9는 args에, first = 3, second = 4, third = 5는 딕셔너리형으로 kwargs에 저장된다.

```
>>> 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

04. 좋은 코드를 작성하는 방법

■ 코딩 규칙

- 들여쓰기는 4 스페이스
- 한 줄은 최대 79자까지
- 불필요한 공백은 피함
- 파이썬에서는 이러한 규칙 중 파이썬 개발자가 직접 정한 것이 있다. 이를 (PEP 8Python Enhance Proposal 8)이라고 하는데, 이는 파이썬 개발자들이 앞으로 필요한 파이썬의 기능이나 여러 가지 부수적인 것을 정의한 문서이다.

04. 좋은 코드를 작성하는 방법

■ PEP 8의 코딩 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 나눈다.

04. 좋은 코드를 작성하는 방법

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
```

- Atom에 [코드 5-19]와 같이 코드를 작성한 후, 'test_flake.py'로 저장한다.

코드 5-19 test_flake.py

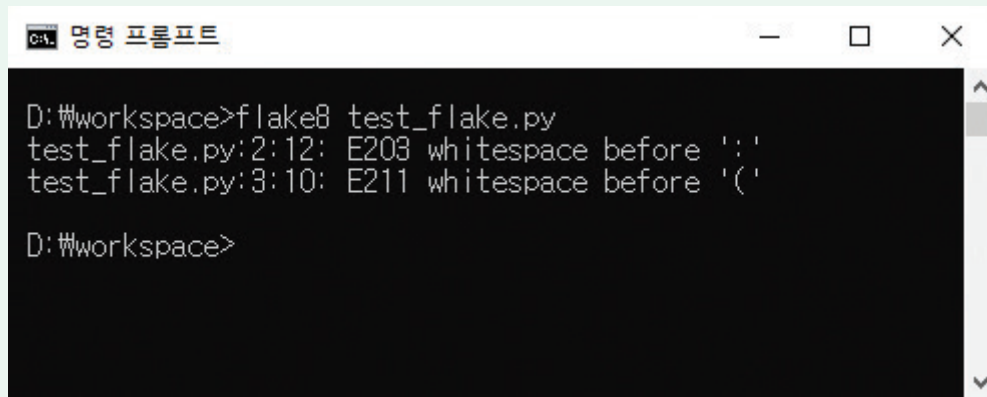
```
1 lL00 = "123"  
2 for i in 10 :  
3     print("Hello")
```

04. 좋은 코드를 작성하는 방법

여기서  잠깐! **flake8** 모듈

- 그리고 cmd 창에 다음과 같이 입력하면, 각 코드의 수정 방법을 알려 준다.

```
flake8 test_flake.py
```



```
cmd. 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>
```

[flake8 모듈로 코드 수정 방법 확인]