함수

함수의 반환 값

함수 (Function)

- 함수의 반환 형태
  - 다음과 같은 경우에는 함수의 반환 값이 없다.
    - return 구문 자체를 사용하지 않은 경우
    - return 키워드만 기재한 경우
    - return None이라고 작성한 경우

```
1  def hello1():
2      print("Hello", 123, "Python")
```

```
1  def hello1():
2      print("Hello", 123, "Python")
3      return
```

```
1  def hello1():
2      print("Hello", 123, "Python")
3      return None
```

함수 (Function)

- 함수의 반환 형태
  - 그 외의 경우에는 함수의 반환 값이 존재하며, 반환되는 값은 언제나 1개이다.
  - 반환된다는 것은 그 함수가 지금까지 하던 작업을 마치고 자신을 호출한 곳으로 되돌아 간다는 의미이며, 여러 개의 값을 하나씩 순서대로 반환할 수는 없다.

```
1  def test(a, b):
2      print("a는", a)
3      print("b는", b)
4
5      return a + 1
6      return b + 1
```

```
1  print(test(3, 7))
```

```
a는 3
b는 7
4
```

```
1  def test(a, b):
2      if a > b:
3          return a + 1
4      else:
5          return b + 1
```

```
1  print(test(3, 7))
```

```
8
```

함수 (Function)

- 함수의 반환 형태
  - return 구문 뒤에 여러 개의 값을 기재하면, 그 값들을 순서대로 항목으로 가지는 1개의 튜플이 반환된다.

```
t = divmod(7, 3)

print(t)
```

```
(2, 1)
```

```
def test1(a, b):
    return (a + 1, b + 1)

c = test1(3, 7)

print(c)
```

```
(4, 8)
```

```
def test2(data):
    print("매개변수는", data)

    return sum(data), max(data), min(data)

sum_value, max_value, min_value = test2([1, 3, 5])

print("합계는", sum_value)
print("최대값은", max_value)
print("최소값은", min_value)
```

```
매개변수는 [1, 3, 5]
합계는 9
최대값은 5
최소값은 1
```

함수

자료 구조의 인자 전달

함수 (Function)

- 변할 수 있는 자료 구조를 인자로 전달할 때
  - 리스트, 사전처럼 변할 수 있는(mutable) 자료를 인자로 사용하면 함수 내에서 자료의 원본에 접근할 수 있다.

```
1  mylist = [1, 2, 3]
2
3  def test_list(t):
4      return t.pop()
5
6  print("mylist =", mylist)
7
8  mypop = test_list(mylist)
9
10 print("pop한 value =", mypop)
11 print("mylist =", mylist)
```

```
mylist = [1, 2, 3]
pop한 value = 3
mylist = [1, 2]
```

함수 (Function)

- 변할 수 있는 자료 구조를 인자로 전달할 때
  - 리스트, 사전처럼 변할 수 있는(mutable) 자료를 인자로 사용하면 함수 내에서 자료의 원본에 접근할 수 있다.

```
1  old_dict = {"name": "홍길동", "age": 25}
2
3  def test_dict(d, name):
4      d["name"] = name
5      return d
6
7  print("old =", old_dict)
8
9  new_dict = test_dict(old_dict, "Hong Kil-Dong")
10
11 print("new =", new_dict)
12 print("old =", old_dict)
```

```
old = {'name': '홍길동', 'age': 25}
new = {'name': 'Hong Kil-Dong', 'age': 25}
old = {'name': 'Hong Kil-Dong', 'age': 25}
```

함수 (Function)

- 변할 수 있는 자료 구조를 인자로 전달할 때
  - 변할 수 있는 자료를 인자로 사용할 때는 그 자료가 수정되는 경우에 대비하여 사본을 만들어 이용한다.

```
1  mylist = [1, 2, 3]
2
3  def test_list(t):
4      t1 = list(t)
5      return t1.pop()
6
7  print("mylist =", mylist)
8
9  mypop = test_list(mylist)
10
11 print("pop한 value =", mypop)
12 print("mylist =", mylist)
```

```
mylist = [1, 2, 3]
pop한 value = 3
mylist = [1, 2, 3]
```

함수 (Function)

- 변할 수 있는 자료 구조를 인자로 전달할 때
  - 변할 수 있는 자료를 인자로 사용할 때는 그 자료가 수정되는 경우에 대비하여 사본을 만들어 이용한다.

```
1  old_dict = {"name": "홍길동", "age": 25}
2
3  def test_dict(d, name):
4      d1 = dict(d)
5      d1["name"] = name
6      return d1
7
8  print("old =", old_dict)
9
10 new_dict = test_dict(old_dict, "Hong Kil-Dong")
11
12 print("new =", new_dict)
13 print("old =", old_dict)
```

```
old = {'name': '홍길동', 'age': 25}
new = {'name': 'Hong Kil-Dong', 'age': 25}
old = {'name': '홍길동', 'age': 25}
```

함수

함수의 축약된 형태

함수 (Function)

- 람다 (Lambda) 함수

  lambda 매개변수(들) : 표현식

  - 1개의 구문(표현식)으로 정의하는 축약된 형태의 함수
  - 기본적으로 이름 없이 사용될 수 있기 때문에 익명 함수라고도 한다.
  - 매개변수를 받아서 표현식에 따라서 연산한 뒤 그 결과를 반환한다.

```
1  f = lambda x : x + 1
2
3  f(10)
```

```
11
```

```
1  (lambda x, y : x * y)(2, 3)
```

```
6
```

함수 (Function)

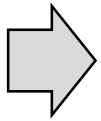
- 람다 (Lambda) 함수

  lambda 매개변수(들) : 표현식

  – 람다 함수가 필요한 경우
    - 함수를 간단하게 한번만 사용하고 버릴 때
    - 다른 함수의 매개변수로 함수를 사용할 때

```
1  def test(x):
2      return 2 * x
3
4  a = [1, 2, 3, "A"]
5  b = list(map(test, a))
6
7  print(b)
```

```
[2, 4, 6, 'AA']
```



```
1  c = list(map(lambda x: 2*x, a))
2
3  print(c)
```

```
[2, 4, 6, 'AA']
```

함수

다양한 매개변수 및 인자의 형태

함수 (Function)

- 매개변수의 기본값 지정하기 (Default Argument)

```
def 함수명(매개변수1, … , 매개변수n=기본값):
    이 함수가 수행할 구문1
    이 함수가 수행할 구문2
    …
```

  - 매개변수의 이름 뒤에 기본값을 지정해 둘 수 있다.
  - 이렇게 하면 함수를 호출할 때 해당 매개변수에 대응하는 인자를 전달하지 않아도 자동으로 기본값으로 설정된다.

```
1  def test(a, b, c=0):
2      return a + b + c
```

```
1  print(test(1, 2, 3))
```

```
6
```

```
1  print(test(1, 2))
```

```
3
```

함수 (Function)

- 매개변수의 기본값 지정하기 (Default Argument)

  def 함수명(매개변수1, … , 매개변수n=기본값):

  이 함수가 수행할 구문1

  이 함수가 수행할 구문2

  …

  – 기본값을 지정하려는 매개변수들은 반드시 매개변수들의 목록에서 뒤 쪽부터 위치해야 한다.

```
1  def test1(a, b=1, c=0):
2      return a + b + c
3
4  print(test1(1, 2))
5  print(test1(1))
```

```
3
2
```

```
1  def test2(a=2, b=1, c):
2      return a + b + c
3
4  print(test2(1, 2))
5  print(test2(1))
```

```
  File "<ipython-input-221-1d283f2054c5>", line 1
    def test2(a=2, b=1, c):
                        ^
SyntaxError: non-default argument follows default argument
```

함수 (Function)

- 매개변수 지정하여 호출하기 (Keyword Argument)

  호출 대상 함수명(인자값1, … , 매개변수명=인자값n)

  - 함수를 호출할 때, 매개변수의 이름을 직접 명시하여 호출할 수 있다.
  - 이렇게 하면 인자 값을 위치에 맞추어 기재하지 않아도 해당 매개변수로 값이 전달된다.

```
1  def test(a, b, c):
2      return a + b + c
```

```
1  print(test(c=3, a=1, b=2))
```

```
6
```

```
1  print(test(1, c=0, b=9))
```

```
10
```

함수 (Function)

- 매개변수 지정하여 호출하기 (Keyword Argument)

  호출 대상 함수명(인자값1, … , 매개변수명=인자값n)

  – 매개변수를 지정하여 전달되는 인자들은 반드시 인자들의 목록에서 뒤 쪽부터 위치해야 한다.

```
1  def test1(a, b, c):
2      return a + b + c
3
4  print(test(1, c=0, b=9))
```

```
10
```

```
1  print(test(c=3, b=1, 2))
```

```
  File "<ipython-input-244-0920b97d3057>", line 1
    print(test(c=3, b=1, 2))
                         ^
SyntaxError: positional argument follows keyword argument
```

함수 (Function)

```python
def test(a, b=1, c=0):
    print("a는", a, end=" / ")
    print("b는", b, end=" / ")
    print("c는", c)

    return a + b + c
```

```python
print(test(3, 7))
print()

print(test(25, c=24))
print()

print(test(c=50, a=99))
print()
```

```
a는 3 / b는 7 / c는 0
10

a는 25 / b는 1 / c는 24
50

a는 99 / b는 1 / c는 50
150
```

함수 (Function)

- 여러 개의 인자를 하나의 매개변수에 전달 (Packing)

  def 함수명(매개변수1, … , *매개변수n):
  이 함수가 수행할 구문1
  이 함수가 수행할 구문2
  …

  – 인자의 수가 상황에 따라 변할 수 있다면, 함수를 정의할 때 매개변수 이름 앞에 '*' 기호를 덧붙인다.

```
1  def test(*args):
2      print("매개변수 args는", args)
3
4      return sum(args)
```

```
1  print(test(1, 2, 3))
```

```
매개변수 args는 (1, 2, 3)
6
```

```
1  print(test(3, -10, 999, 54, -267))
```

```
매개변수 args는 (3, -10, 999, 54, -267)
779
```

함수 (Function)

- 여러 개의 인자를 하나의 매개변수에 전달 (Packing)

  def 함수명(매개변수1, … , *매개변수n):
  이 함수가 수행할 구문1
  이 함수가 수행할 구문2
  …

  – 이렇게 하면 해당 매개변수는 여러 개의 인자 값을 1개의 튜플로 모아서 전달받는다.

```
1  def test(a, *b):
2      print("a는", a, end=" / ")
3      print("b는", b)
4
5      return len(b)
```

```
1  print(test(1, 2, 3))
```

```
a는 1 / b는 (2, 3)
2
```

```
1  print(test(3, -10, 999, 54, -267))
```

```
a는 3 / b는 (-10, 999, 54, -267)
4
```

함수 (Function)

- 하나의 인자를 여러 매개변수에 전달 (Unpacking)

  호출 대상 함수명(인자값1, … , *인자값n)

  - 함수를 호출할 때, 1개의 인자 값을 여러 개로 나누어 전달하려면 인자 앞에 '*' 기호를 덧붙인다.
  - 이렇게 하면 나뉘어진 여러 개의 인자 값이 순서대로 매개변수로 전달된다.

```
1  def test(a, b, c):
2      return a + b + c
```

```
1  t1 = (10, 20, 30)
2
3  print(test(*t1))
```

```
60
```

```
1  t2 = [2, 3]
2
3  print(test(1, *t2))
```

```
6
```

함수 (Function)

- 여러 개의 변수명과 값을 하나의 매개변수에 전달

```
def 함수명(매개변수1, … , **매개변수n):
    이 함수가 수행할 구문1
    이 함수가 수행할 구문2
    …
```

  - 인자가 여러 개의 '변수명=값' 형태로 존재한다면, 함수를 정의할 때 매개변수 이름 앞에 '**' 기호를 덧붙인다.

```
1  def test(**kwargs):
2      print("매개변수 kwargs는", kwargs)
3
4      return list(kwargs.values())
```

```
1  print(test(a=1, b=2, c=3))
```

```
매개변수 kwargs는 {'a': 1, 'b': 2, 'c': 3}
[1, 2, 3]
```

```
1  print(test(name="홍길동", age=25))
```

```
매개변수 kwargs는 {'name': '홍길동', 'age': 25}
['홍길동', 25]
```

함수 (Function)

- 여러 개의 변수명과 값을 하나의 매개변수에 전달

  def 함수명(매개변수1, … , **매개변수n):

  이 함수가 수행할 구문1

  이 함수가 수행할 구문2

  …

  – 이렇게 하면 해당 매개변수는 여러 개의 '변수명=값'을 1개의 사전으로 모아서 전달받는다.

```
1  def test(a, **b):
2      print("a는", a, end=" / ")
3      print("b는", b)
4
5      return len(b)
```

```
1  print(test(1, b=2, c=3))
```

```
a는 1 / b는 {'b': 2, 'c': 3}
2
```

```
1  print(test(a="A", b="B", c="C", z="Z"))
```

```
a는 A / b는 {'b': 'B', 'c': 'C', 'z': 'Z'}
3
```

함수 (Function)

- 하나의 사전을 여러 이름과 값으로 매개변수에 전달

  호출 대상 함수명(인자값1, … , **인자값n)

  - 함수를 호출할 때, 1개의 사전을 여러 개로 나누어 전달하려면 인자 앞에 '**' 기호를 덧붙인다.
  - 이 때 사전의 키와 매개변수의 이름은 모두 각각 일치해야 한다.

```
1  def test(a, b, c):
2      return a + b + c
```

```
1  d1 = {'a':10, 'b':20, 'c': 30}
2
3  print(test(**d1))
```

```
60
```

```
1  d2 = {'a':1, 'c':3}
2
3  print(test(b=2, **d2))
```

```
6
```

함수 (Function)

```python
def test(a=1, *b, **c):
    print("a는", a)
    print("b는", b)
    print("c는", c)

    d = list(c.values())

    return a + sum(b) + sum(d)
```

```python
print(test(3, 7, c=5, d=15))
print()

print(test(1, 2, 3, 4, b=10))
print()
```

```
a는 3
b는 (7,)
c는 {'c': 5, 'd': 15}
30

a는 1
b는 (2, 3, 4)
c는 {'b': 10}
20
```