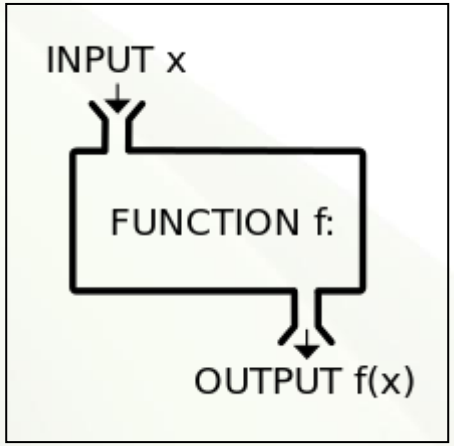
함수

함수?

- 실행문 집합
- 반복적으로 사용되는 코드가 있을 경우 효과적인 재사용을 위해 함수로 구현
- 복잡한 코드를 단위 기능별로 분리하고 효과적으로 관리하기 위한 목적으로도 함수 사용

- 형식

```
def 함수명(입력 인수):
    <수행할 문장1>
    <수행할 문장2>
    ...
    return 결과 값
```



- 구현된 함수는 호출을 통해 실행됨

```
변수 = 함수명(입력 인수)
```

함수의 입력 값과 출력 값

- 입력 값은 함수에게 전달되는 데이터이고 출력 값은 함수가 호출한 곳으로 반환하는 값
  - » 입력 값을 전달 인자 또는 입력 인수로 부름
  - » 출력은 return문을 통해서 반환

- 입력 값과 출력 값이 모두 있는 함수

```
def sum(a, b):
    result = a + b
    return result
```

```
>>> a = sum(3, 4)
>>> print(a)
7
```

- 입력 값이 없는 함수

```
>>> def say():
...     return 'Hi'
...
>>>
```

```
>>> a = say()
>>> print(a)
Hi
```

함수의 입력 값과 출력 값

- 출력 값이 없는 함수

```
>>> def sum(a, b):
...     print("%d, %d의 합은 %d입니다." % (a, b, a + b))
...
>>>
```

```
>>> sum(3, 4)
3, 4의 합은 7입니다.
```

- 입력 값과 출력 값이 모두 없는 함수

```
>>> def say():
...     print('Hi')
...
>>>
```

```
>>> say()
Hi
```

함수의 결과 값

- 함수의 결과 값은 언제나 한 개
  - » 두 개 이상의 값을 반환할 경우 반환 값을 목록으로 하는 하나의 튜플을 만들어서 반환

```
>>> def sum_and_mul(a,b):
...     return a+b, a*b
```

  - » 함수를 호출하고 튜플로 반환 값 저장

```
>>> result = sum_and_mul(3,4)
```

  - » 함수를 호출하고 각각 개별 변수에 튜플 값을 나누어서 저장

```
>>> sum, mul = sum_and_mul(3, 4)
```

스코핑 룰

- 이름 공간 (Namespace)
  - » 변수, 함수 등의 이름이 저장되는 영역

- 함수는 독립적인 이름 공간 형성
  - » 함수 내부에서 만들어진 변수는 함수의 이름 공간에 포함
  - » 함수에서 이름을 사용하면 함수 이름 공간 → 상위 이름 공간 순으로 검색

```
a = [1, 2, 3]

def scoping1():
    a = [4, 5, 6] # 함수 지역 변수 a 생성
    print("function scope : ", a) # 함수 지역 변수 a 사용

print(a) # 전역 변수 a
```

```
[1, 2, 3]
```

```
x = 1

def scoping2(a):
    return a + x # 상위 영역의 x 사용

scoping2(10)
```

```
11
```

입력 인수에 기본 값 사용

- 입력 인수 = 기본 값 형식으로 입력 인수에 기본 값 설정
  - » 입력 인수에 기본 값이 지정된 경우 호출할 때
    - › 값을 전달하지 않으면 기본 값이 사용되고
    - › 값을 전달하면 전달한 값이 사용됨

```
def say_myself(name, old, man=True):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

```
say_myself("장동건", 40)
say_myself("장동건", 40, True)
```

```
나의 이름은 장동건입니다.
나이는 27살입니다.
남자입니다.
```

입력 인수에 기본 값 사용

- 기본 값을 지정한 입력 인수는 모든 입력 인수 중 마지막에만 사용할 수 있음
  - » 일반 입력 인수 중간에 기본 값을 지정한 입력 인수를 사용할 수 없음

```
def say_myself(name, man=True, old):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

실행할 경우 오류 발생

함수의 입력 값과 출력 값

- 입력 값의 개수를 미리 확정할 수 없는 경우 1 (리스트 사용)
  - » *를 사용해서 입력변수 표현

```
def 함수이름(*입력변수):
    <수행할 문장>
    ...
```

```
>>> def sum_many(*args):
...     sum = 0
...     for i in args:
...         sum = sum + i
...     return sum
...
>>>
```

```
>>> result = sum_many(1,2,3)
>>> print(result)
6
>>> result = sum_many(1,2,3,4,5,6,7,8,9,10)
>>> print(result)
55
```

함수의 입력 값과 출력 값

- 입력 값의 개수를 미리 확정할 수 없는 경우 2 (딕셔너리 사용)
  - » **를 사용해서 입력변수 표현

```
def 함수이름(**입력변수):
    <수행할 문장>
    ...
```

```
def var_args_func(**dict_data):
    for key, item in dict_data.items():
        print( (key, item), end=' ' )
    print()
```

```
var_args_func(a = 100, b = 200, c = 300, d = 400, e = 500)

data = { 'a' : 100, 'b' : 200, 'c' : 300, 'd' : 400, 'e' : 500 }
var_args_func(**data)
```

```
('a', 100) ('b', 200) ('c', 300) ('d', 400) ('e', 500)
('a', 100) ('b', 200) ('c', 300) ('d', 400) ('e', 500)
```

함수 내부에 선언된 변수의 사용 범위

- 함수 내부에서 선언된 변수는 함수 안에서만 사용할 수 있음 → 지역 변수

```
a = 1
def vartest(a):
    a = a +1

vartest(a)
print(a)
```

위 코드의 실행 결과는 1 (함수 안의 a와 함수 밖의 a는 별개의 변수)

- 함수 내부에서 함수 외부의 변수를 변경하는 방법 → global 변수 사용
  - » 이 방법은 일반적으로 권장되지 않음

```
a = 1
def vartest():
    global a
    a = a+1
vartest()
print(a)
```

위 코드의 실행 결과는 2 (함수 안의 a와 함수 밖의 a는 같은 변수)

람다 함수

- 이름이 없는 익명 함수 생성

```
lambda 전달인자목록 : 전달인자를 사용하는 실행문
```

```
def squared(x):
    return x**2

a = [1, 2, 3, 4, 5]
a2 = map(squared, a)
for d in a2:
    print(d, end=' ')
```

```
1 4 9 16 25
```

```
a = [1, 2, 3, 4, 5]
a2 = map(lambda x: x**2, a)
for d in a2:
    print(d, end=' ')
```

```
1 4 9 16 25
```

```
a = [1, 2, 3, 4, 5]
a2 = filter(lambda x: x % 2 == 0, a)
for d in a2:
    print(d, end=' ')
```

```
2 4
```

문서 주석

- 함수, 클래스 등에 대한 설명을 추가하는 방법

```
print.__doc__
```

```
"print(value, ..., sep=' ', end='\\n', file=sys.stdout, flush=False)\n\nPrints the
values to a stream, or to sys.stdout by default.\nOptional keyword …"
```

» 함수, 클래스 등의 앞부분에 작성된 문자열은 자동으로 문서 주석 처리

```
def my_func():
    """
    이 함수는 문서 주석 테스트를 위해 만든 함수입니다.
    """
my_func.__doc__
```

```
"print(value, ..., sep=' ', end='\\n', file=sys.stdout, flush=False)\n\nPrints the
values to a stream, or to sys.stdout by default.\nOptional keyword
arguments:\nfile:  a file-like object (stream); defaults to the current
sys.stdout.\nsep:   string inserted between values, default a space.\nend:
string appended after the last value, default a newline.\nflush: whether to
forcibly flush the stream."
```

Generator

- iterator (반복자)
  - » 목록의 값들을 순회할 수 있는 객체
  - » iter() 함수로 만들고 next() 함수로 다음 객체 반환

```
it = iter([1, 2, 3, 4, 5])
next(it), next(it), next(it), next(it), next(it)
```

```
(1, 2, 3, 4, 5)
```

- Generator
  - » yield를 사용해서 iterator를 만드는 함수

```
def my_generator(p):
    for i in p:
        yield i

mg = my_generator([1, 2, 3, 4, 5])
next(mg), next(mg), next(mg), next(mg), next(mg)

for x in my_generator([1, 2, 3, 4, 5]):
    print(x, end=' ')
```

```
(1, 2, 3, 4, 5)
```

```
1 2 3 4 5
```