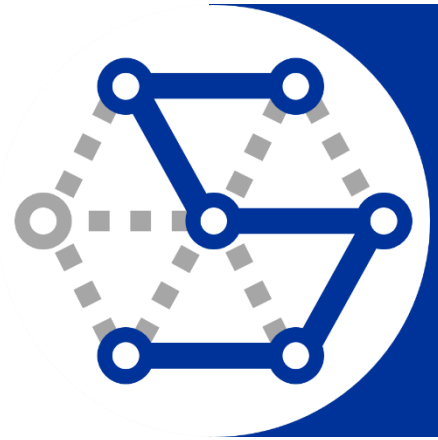
데이터 처리 프로그래밍

Data Processing Programming


07
함수
suanlab
수안컴퓨터 연구소
Suan computer laboratory

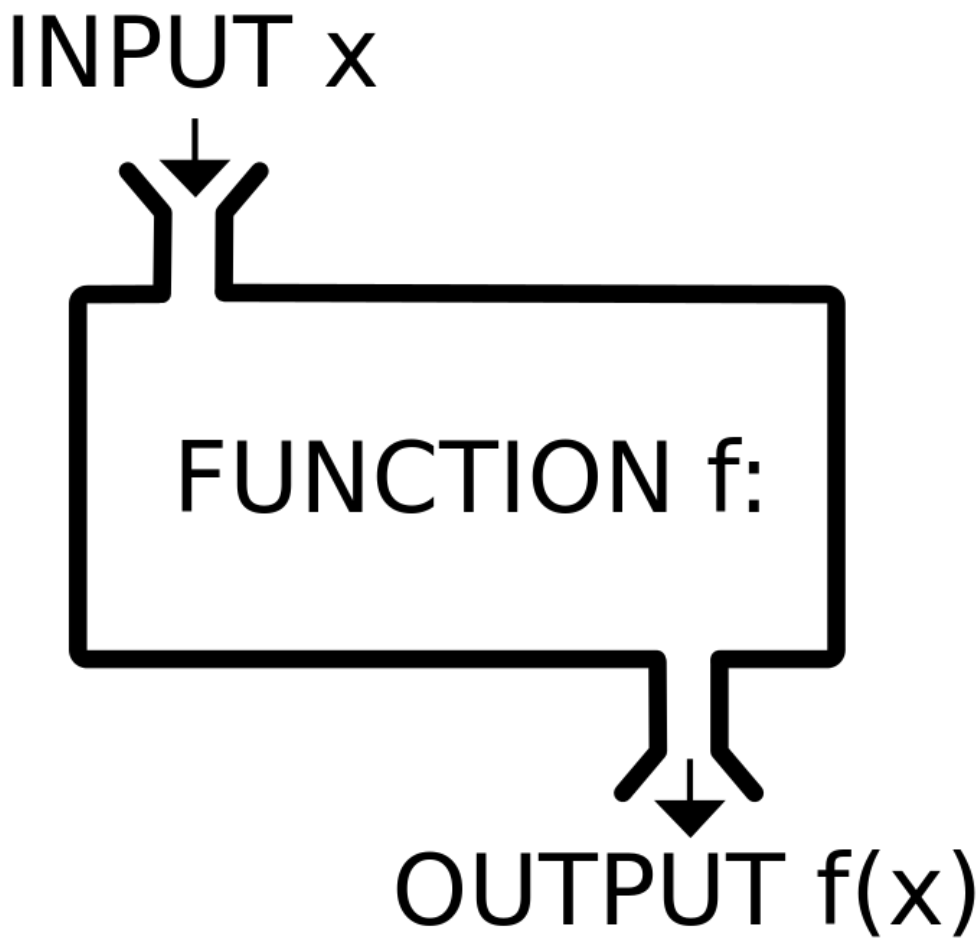
목차



1. 함수 기본

함수 개념

- 특정 값 X를 인자로 받고, 결과값을 반환: $Y = f(X)$
- 함수가 필요할 때마다 호출 가능
- 논리적인 단위로 분할 가능
- 코드의 캡슐화Capsulation
- 중복되는 소스코드를 최소화
- 소스코드의 재사용성을 높임



https://en.wikipedia.org/wiki/Function_(mathematics)

함수 선언

- 함수 선언 문법

```
def 함수명(매개변수):
        <수행문>
        return <반환값>
```

- def: 정의definition의 줄임으로 사용
- 함수명: 사용자가 임의로 지정
- 함수명 convention
  - 짧고 명료한 이름
  - 소문자로 입력
  - 띄어쓰기는 '_' 기호 사용 (hello_world)
  - 동사와 명사를 함께 사용 (find_name)
- 매개변수parameter: 함수에서 입력값으로 사용하는 변수
- 반환값: 함수에서 반환할 결과값 지정

매개변수와 반환값이 없는 함수

- 함수에 매개변수와 반환값이 없이 사용 가능
- 함수를 호출하면 함수의 수행문이 실행

```
>>> def hello():
...     print("Hello Python")
...
>>> hello()
Hello Python
```

매개변수만 있는 함수

- 문자열 매개변수를 사용한 함수

```
>>> def hello(string):
...     print("Hello " + string)
...
>>> hello("World")
Hello World
```

반환값만 있는 함수

- 문자열 반환값을 사용한 함수

```
>>> def hello():
...     return "Hello Python"
...
>>> hello()
'Hello Python'
```

매개변수와 반환값이 있는 함수

- 정수형 매개변수와 반환값을 사용한 함수

```
>>> def square(num):
...     return num*num
...
>>> square(5)
25
```

매개변수가 여러개 있는 함수

- 정수형 매개변수 여러개를 사용한 함수
- 매개변수를 지정하여 호출 가능

```
>>> def add(n1, n2):
...     return n1 + n2
...
>>> add(5, 8)
13
>>> add(n2=5, n1=8)
13
```

키워드 매개변수

- 함수의 매개변수를 변수명을 지정하여 호출 가능

```
>>> def add(n1, n2):
...     return n1 + n2
...
>>> add(n2=5, n1=8)
13
```

가변 매개변수

- 매개변수가 몇 개인지 알 수 없을 때 사용
- 매개변수 앞에 '*'을 표시

```
>>> def sum(*args):
...     result = 0
...     for i in args:
...         result = result + i
...     return result
...
>>> sum(1, 2, 3, 4, 5)
15
>>> sum(1, 2, 3, 4, 5, 6, 7, 8, 9)
45
```

키워드 매개변수

- 매개변수의 이름을 따로 지정하지 않고 사용
- 매개변수 앞에 '**'을 표시

```
>>> def print_kwargs(**kwargs):
...     print(kwargs)
...
>>> print_kwargs(n1=5, n2=8)
{'n1': 5, 'n2': 8}
>>> print_kwargs(id="Suan", pw="1234")
{'id': 'Suan', 'pw': '1234'}
```

초기값 매개변수

- 매개변수에 초기값을 설정하여 사용
- 함수에 매개변수를 사용하지 않을 때 초기값을 사용

```
>>> def power(b=2, n=2):
...     return pow(b, n)
...
>>> power()
4
>>> power(3)
9
>>> power(5, 2)
25
>>> power(n=3)
8
```

여러 반환값이 있는 함수

- 함수의 반환값은 하나
- 여러 반환값을 사용할 경우 튜플 형태로 반환

```
>>> def plus_and_minus(n1, n2):
...     return n1+n2, n1-n2
...
>>> plus_and_minus(8, 5)
(13, 3)
>>> result = plus_and_minus(8, 5)
>>> result
(13, 3)
>>> result1, result2 = plus_and_minus(8, 5)
>>> result1
13
>>> result2
3
```

Lab. 계산기 함수

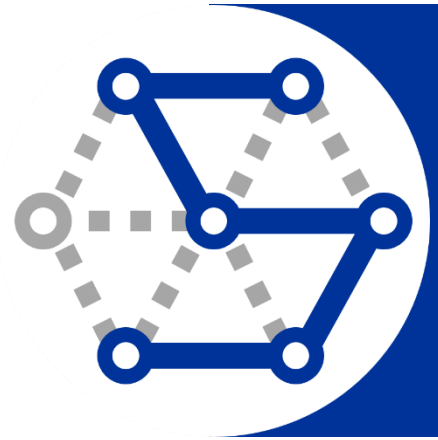
- 두 수에 대해서 덧셈, 뺄셈, 곱셈, 나눗셈을 수행하는 함수

```
>>> def calc(op, n1, n2):
...     result = 0
...     if op == '+':
...         [redacted]
...     elif op == '-':
...         [redacted]
...     elif op == '*':
...         [redacted]
...     elif op == '/':
...         [redacted]
...
...     return result
...
>>> calc('+', 8, 5)
13
>>> calc('*', 8, 5)
40
```

Lab. 가변 매개변수의 평균값 계산

- 가변 매개변수로 들어오는 모든 수의 평균값 계산

```
>>> def avg(*args):
...     sum = 0
...     for i in args:
...         sum += i
...     return [redacted]
...
>>> avg(1, 2, 3, 4, 5)
3.0
```



2. 변수의 유효범위

유효 범위

- 변수는 유효한 범위가 존재
- 함수 안에서 선언된 변수는 함수 내부에서 유효함

```
>>> def var_scope(a):
...     a = a + 1
...
>>> a = 10
>>> var_scope(a)
>>> print(a)
10
```

변수의 종류

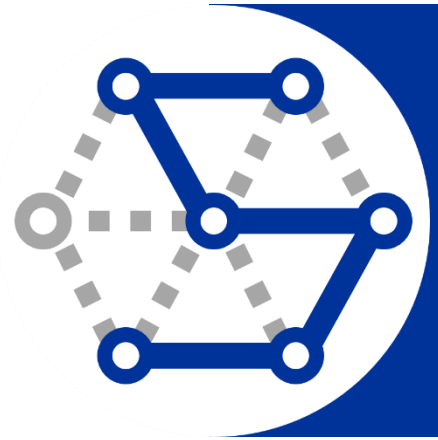
- 지역변수: 한정된 지역에서만 사용되는 변수
- 전역 변수: 프로그램 전체에서 사용되는 변수

```
>>> a = 10
>>> def func1():
...     a = 20
...     print(a)
...
>>> def func2():
...     print(a)
...
>>>
>>> func1()
20
>>> func2()
10
```

전역 변수 사용 global

- 함수 내부에서 전역 변수를 사용하기 위한 global 키워드

```
>>> a = 10
>>> def func1():
...     global a
...     a = 20
...     print(a)
...
>>> def func2():
...     print(a)
...
>>> func1()
20
>>> func2()
20
```



3. 함수 심화

내부 함수

- 함수 안에 함수가 존재
- 내부 함수는 외부에서 호출 불가

```
>>> def func1(n1, n2):
...     def func2(num1, num2):
...         return num1 + num2
...     return func2(n1, n2)
...
>>> func1(5, 8)
13
```

재귀 함수recursive function

- 함수가 자기 자신을 다시 부르는 함수
- count() 함수 내부에서 count() 함수를 호출
- 재귀적으로 카운트 수를 출력

```
>>> def count(n):
...     if n >= 1:
...         print(n, end=' ')
...         count(n-1)
...     else:
...         return
...
>>> count(10)
10 9 8 7 6 5 4 3 2 1 >>>
```

팩토리얼 함수factorial function

- 팩토리얼factorial 함수는 대표적인 재귀 함수
- 팩토리얼 함수

$$n! = 1 \cdot 2 \cdot 3 \cdots (n-2) \cdot (n-1) \cdot n$$

$$n! = \prod_{i=1}^{n} i$$

$$n! = n \cdot (n-1)!$$

```
>>> def factorial(n):
...     if n == 1:
...         return 1
...     else:
...         return n * factorial(n-1)
...
>>> factorial(10)
3628800
>>> factorial(5)
120
>>> factorial(3)
6
```

람다 함수lambda function

- 함수를 한 줄로 간결하게 만들어 사용

```
>>> def add(n1, n2):
...     return n1 + n2
...
>>> add(5, 8)
13
>>> add2 = lambda n1, n2 : n1 + n2
>>> add2(5, 8)
13
```

map() 함수map function

- map() 함수: built-in 함수로 list 나 dictionary 와 같은 iterable 한 데이터를 인자로 받아 list 안의 개별 item을 함수의 인자로 전달하여 결과를 list로 형태로 반환해 주는 함수
- 람다 함수와 map() 함수를 이용한 리스트 계산

```
>>> l = [1, 2, 3, 4, 5]
>>> square = lambda n : n * n
>>> l = list(map(square, l))
>>> l
[1, 4, 9, 16, 25]
```

```
>>> l1 = [1, 2, 3, 4, 5]
>>> l2 = [6, 7, 8, 9, 10]
>>> l = list(map(lambda n1, n2 : n1 + n2, l1, l2))
>>> l
[7, 9, 11, 13, 15]
```

filter() 함수filter function

- filter() 함수: iterable 한 데이터를 인자로 개별 item을 특정 조건에 해당하는 값으로만 필터링
- 람다 함수와 filter() 함수를 이용한 리스트 필터링

```
>>> l = list(range(10))
>>> evens = filter(lambda n : n % 2 is 0, l)
>>> list(evens)
[0, 2, 4, 6, 8]
```

reduce() 함수reduce function

- reduce() 함수: iterable 한 데이터를 인자로 받아 개별 item을 축약하여 하나의 값으로 만들어 가는 과정
- 람다 함수와 reduce() 함수를 이용한 리스트 계산

```
>>> import functools
>>> l = list(range(10))
>>> sum = functools.reduce(lambda x, y: x + y, l)
>>> sum
45
>>> len = functools.reduce(lambda x, y: x + 1, l, 0)
>>> len
10
>>> max = functools.reduce(lambda x, y: x if x > y else y, l)
>>> max
9
```

제너레이터generator와 yield

- 함수 안에서 yield를 사용하면 제너레이터
- yield: 함수를 끝내지 않고 값을 계속 반환

```
>>> def gen():
...     yield 1
...     yield 2
...     yield 3
...
>>> gen()
<generator object gen at 0x0000026B9AA700C0>
>>> print(list(gen()))
[1, 2, 3]
```

```
>>> for i in gen():
...     print(i)
...
1
2
3
```

```
>>> g = gen()
>>> a = next(g)
>>> print(a)
1
>>> b = next(g)
>>> print(b)
2
>>> c = next(g)
>>> print(c)
3
```

Lab. 재귀 함수를 이용한 합계

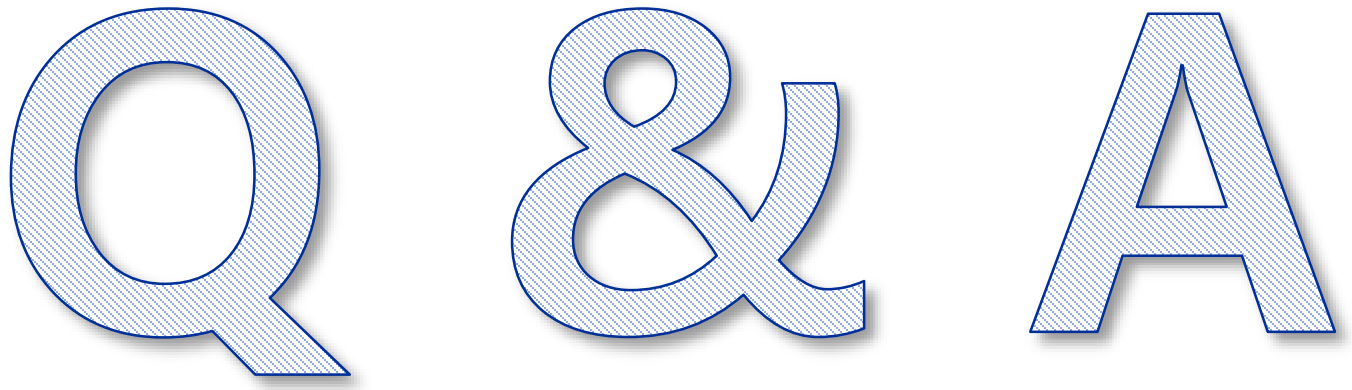
- sum() 함수 내부에서 sum() 함수를 호출
- 재귀적으로 합계를 계산

```
>>> def sum(n):
...     if n == 1:
...         [          ]
...     else:
...         [                    ]
...
>>> sum(10)
55
>>> sum(100)
5050
```

Lab. 짝수만 생성하는 제너레이터

- 0~n개의 숫자 중에서 짝수만 생성하는 제너레이터 함수 생성

```
>>> def gen_even(n):
...     for i in range(n):
...         if [redacted]:
...             [redacted]
...
>>> for i in gen_even(10):
...     print(i)
...
0
2
4
6
8
```


Q & A