

# 파이썬의 함수

## 1 함수의 정의

입력 값으로 어떤 일을 수행한 뒤 결과 값을 출력하는 것

예

입력 : 2, 4, 6

결과 : 4, 8, 12



함수 :  $y = 2x$

# 파이썬의 함수

## 1 함수의 정의



주로 자주 사용하는 반복된 코드를 일반화하여 함수로 사용

- 중복된 코드를 줄일 수 있고, 코드의 가독성, 유지보수성을 향상시킬 수 있음



소프트웨어의 개발 시 수행 단위로 의미가 있는 코드도 함수로 사용

- def 함수명(매개변수):  
    코드

```
def sayHello():
    print("안녕하세요")
    print("파이썬의 함수입니다.")
```

# 파이썬의 함수

## 1 함수의 정의



함수의 이름 또한 식별자 규칙을 지켜야 함

- 특수문자, 공백, 대·소문자 구분 등



제어문과 마찬가지로 콜론과 들여쓰기를 꼭 해야 함



아무 행동도 하지 않는 함수는 pass 키워드를 적어줘야 함

```
def func():
    pass
```



함수의 설명(Docstring)을 적어둘 수 있음

```
def Hello():
    "이것은 함수 예제입니다."
    print("안녕하세요")
    print("파이썬의 함수입니다.")
```

# 파이썬의 함수

## 1 함수의 정의



**내장 함수 `help()`**를 사용해 해당 함수의 설명을 확인할 수 있음

```
help>Hello)
```

Help on function Hello in module \_\_main\_\_:

```
Hello()
```

이것은 함수 예제입니다.



흔히 사용하는 내장 함수들도 `help`를 활용해 설명을 확인할 수 있음

```
help(len)
```

Help on built-in function len in module builtins:

```
len(obj, /)
```

Return the number of items in a container.

# 파이썬의 함수



## 2 함수의 호출과 반환

### 1 함수의 호출

정의한 함수명을 불러서 호출

```
def Hello():  
    print("안녕하세요")  
    print("파이썬의 함수입니다.")
```

Hello()

안녕하세요  
파이썬의 함수입니다.

# 파이썬의 함수



## 2 함수의 호출과 반환

### 1 함수의 호출



매개변수가 있다면, 해당 인자를 호출할 때 적어줘야 함

```
def add(a,b):  
    print(a+b)
```

```
add(1,2)
```

3

```
add()
```

**TypeError**

```
<ipython-input-15-d5d29de3ed94> in  
----> 1 add()
```

**TypeError:** add() missing 2 required

# 파이썬의 함수



## 2 함수의 호출과 반환

### 2 함수의 반환

함수 실행 종료 후,  
지정한 값을 함수가 호출된 지점으로 반환할 수 있음

- def 함수명:  
    코드  
    return 반환 값

```
def add(a,b):  
    print(a+b)  
  
def add2(a,b):  
    return a+b
```

```
x = add(1,2)  
print(x)
```

3  
None

```
y = add2(1,2)  
print(y)
```

3

# 파이썬의 함수



## 2 함수의 호출과 반환

### ② 함수의 반환



두 개 이상의 값을 반환하면, 결과 값은 튜플로 반환



매개변수의 자료형은 동적으로 결정되며, 호출되는 순간 해당 인자에 전달되는 객체에 따라 자료형이 결정됨

```
def func(a,b):  
    return a,b
```

```
x = func(1,2)  
print(x)  
print(type(x))
```

```
(1, 2)  
<class 'tuple'>
```

```
y = func("문자", ['리스트', 1])  
print(y)  
print(type(y))
```

```
('문자', ['리스트', 1])  
<class 'tuple'>
```





# 지역변수와 전역변수

## 1 지역변수

### 함수 내에서 만들어진 변수

- 함수가 실행될 때 생성되며, 함수가 종료될 때 사라짐

```
def func():
    a = 10
    print(a)
a = 20
func()
print(a)
```

함수는  
종료되고,  
a를 출력하면  
20이 출력됨

func() 함수가  
호출될 때  
지역변수 a에  
10이 할당되고,  
10이 출력됨

10  
20

## 지역변수와 전역변수

## 1 지역변수

함수가  
호출될 때  
b가 생성되고  
출력됨

```
def func():
    b = 10
    print(b)
func()
print(b)
```

함수가  
종료되면  
b는 사라지므로  
b를 출력하면  
오류 발생

```
10
```

```
-----
NameError
```

```
<ipython-input-35-98ae05c55619> in
```

```
3     print(b)
```

```
4 func()
```

```
----> 5 print(b)
```

```
NameError: name 'b' is not defined
```

## 지역변수와 전역변수

## 2 전역변수

## 함수 밖에서 만들어진 변수

- 함수와 관계 없이 사용 가능하며, 함수 안에서 참조 가능

```
a = 10
def func():
    print(a)
```

```
a = 10
def func():
    b = 20
    return a+b
```

```
func()
```

10

```
x = func()
print(x)
```

30

a는 전역변수  
b는 지역변수

## 지역변수와 전역변수

## 2 전역변수



global 키워드를 사용해 함수 안에서 전역변수 활용 가능

a는 전역변수로  
값은 10

```
a = 10
print(a)
def func():
    global a
    a = 20
    print(a)
func()
print(a)
```

a의 값을  
20으로 변경

전역변수 a의 값이  
20으로 변경됨

```
10
20
20
```

# 지역변수와 전역변수

## 3 함수에서의 값 전달



함수의 매개변수로 전달 받은 값을 함수 내에서 변경했을 때, 인자로 사용된 호출함수 내에서의 변수의 값은 변경되지 않음

- 자료형에 따라 다르지만 변경 불가능한 객체인 경우 값을 복사하여 전달

a는 전역변수로  
값은 10

```
a = 10
print(a)
def func(b):
    b = 20
    print(b)
func(a)
print(a)
```

매개변수 b를  
20으로 변경

함수의 인자로  
a를 전달

a의 값은  
변경되지 않음

```
10
20
10
```

## 지역변수와 전역변수

## 3 함수에서의 값 전달



전달받은 객체 자체의 변경이 아닌 객체의 요소를 변경하는 것은 가능

```
def func(b):
    b = [1,2,3]
a = [4,5,6]
func(a)
print(a)
```

[4, 5, 6]

```
def func(b):
    b[1] = 10
a = [4,5,6]
func(a)
print(a)
```

매개변수 b의  
1번째 요소를  
10으로 변경

전역변수 a는  
리스트  
(변경 가능한 자료형)

a의 요소 값이  
변경됨

[4, 10, 6]

# 함수의 매개변수

## 1 기본 매개변수



매개변수에 기본 값(Default)을 설정해 값이 없어도 오류가 발생하지 않음

```
def func(a,b):
    print(a+b)
func(10,20)
```

30

```
def func(a,b):
    print(a+b)
func(10)
```

```
-----
TypeError
<ipython-input-59-2a308d67:
      1 def func(a,b):
      2     print(a+b)
----> 3 func(10)

TypeError: func() missing 1
```

```
def func(a,b=10):
    print(a+b)
func(10)
```

20

# 함수의 매개변수

## 1 기본 매개변수



함수 생성 및 호출 시 기본 값이 있는 매개변수가 일반 매개변수보다 앞에 올 수 없음

```
def func(a=10,b):  
    print(a+b)  
func(10,20)
```

File "<ipython-input-61-009171fccb9c>", line 1

```
def func(a=10,b):  
    ^
```

**SyntaxError:** non-default argument follows default argument



# 함수의 매개변수

## 2 키워드 매개변수



함수를 호출할 때 인자는 순서대로 전달됨

```
def func(a,b,c,d,e):  
    print(a+b-c-d+e)  
func(1,2,3,4,5)  
func(5,4,3,2,1)
```

1  
5

# 함수의 매개변수

## 2 키워드 매개변수



순서와 상관 없이 매개변수의 이름과 함께 값을 전달할 수 있음

```
def func(a,b,c,d,e):  
    print(a+b-c-d+e)  
func(a=1,b=2,c=3,d=4,e=5)  
func(e=5,d=4,c=3,b=2,a=1)
```

1

1

# 함수의 매개변수

## 3 가변 매개변수



일반 매개변수 다음에, \*매개변수로 가변 인자를 전달할 수 있음



일반 매개변수에 할당되는 인자를 제외한 나머지 인자는 튜플로 할당

```
def func(a, *arg):  
    print(a, arg)
```

```
func(1)  
func(2,3)  
func(2,3,4,5,6)
```

```
1 ()  
2 (3,)  
2 (3, 4, 5, 6)
```

# Run! 프로그래밍

## Mission 1

## 매개변수 활용

```
def calculator(a, b, c=0):
    if int(c) == 0:
        print(int(a) + int(b))
    elif int(c) == 1:
        print(int(a) - int(b))
    elif int(c) == 2:
        print(int(a) * int(b))
    else:
        print(int(a) / int(b))

a = input("정수를 입력 해 주세요 : ")
b = input("정수를 입력 해 주세요 : ")
c = input("연산을 입력 해 주세요(0:덧셈, 1: 뺄셈,
2: 곱셈, 3: 나눗셈) :")

calculator(a,b,c)
```

## Run! 프로그래밍



## Mission 2 |

## 가변 매개변수 활용

```
def gugu(a):  
    for i in range(1,10):  
        print("{} X {} = {}".format(a,i,a*i))  
  
gugu(4)
```

## 정답

```
def gugu(*a):  
    for j in range(len(a)):  
        print(a[j], "단")  
        for i in range(1,10):  
            print("{} X {} = {}".format(a[j],i,a[j]*i))  
        print()  
gugu(4,2,3)
```