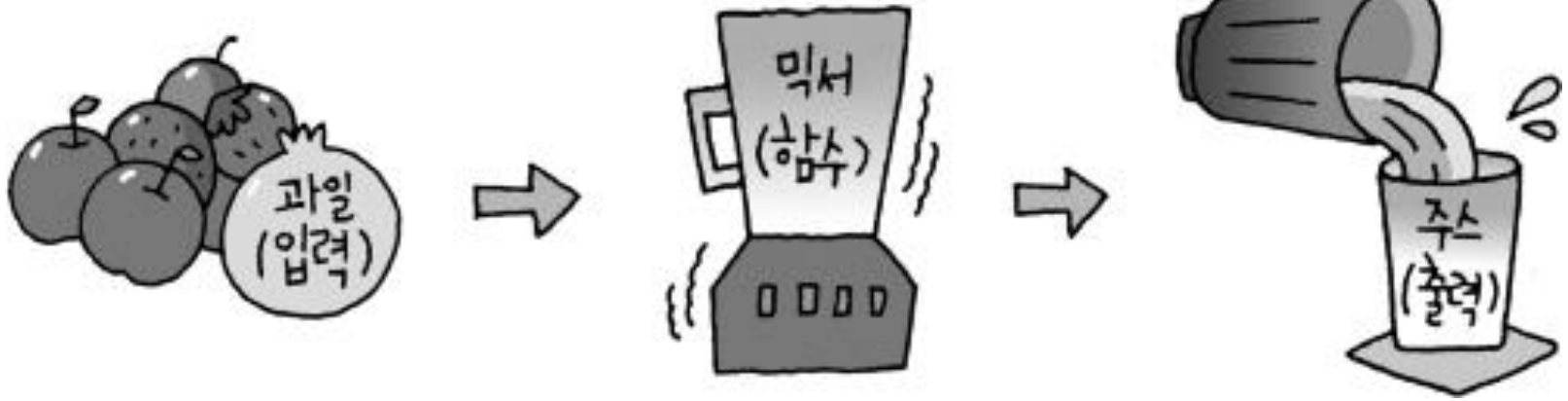
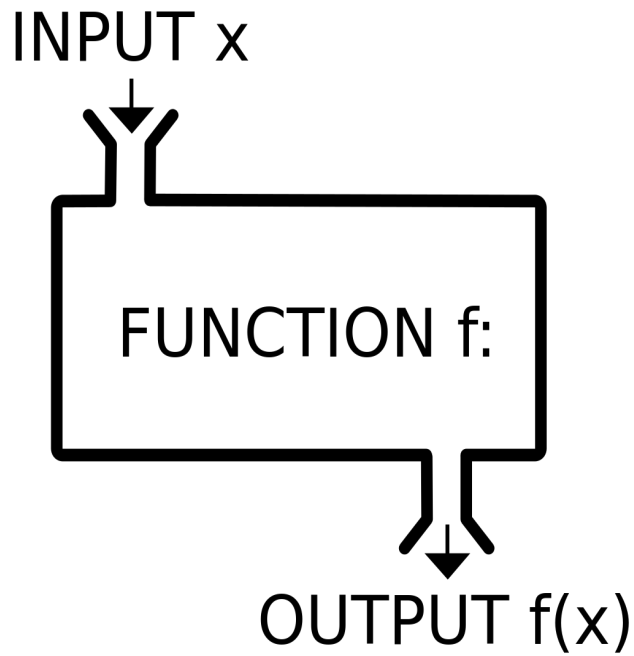
파이썬 프로그래밍

과일, 설탕

믹서(함수)

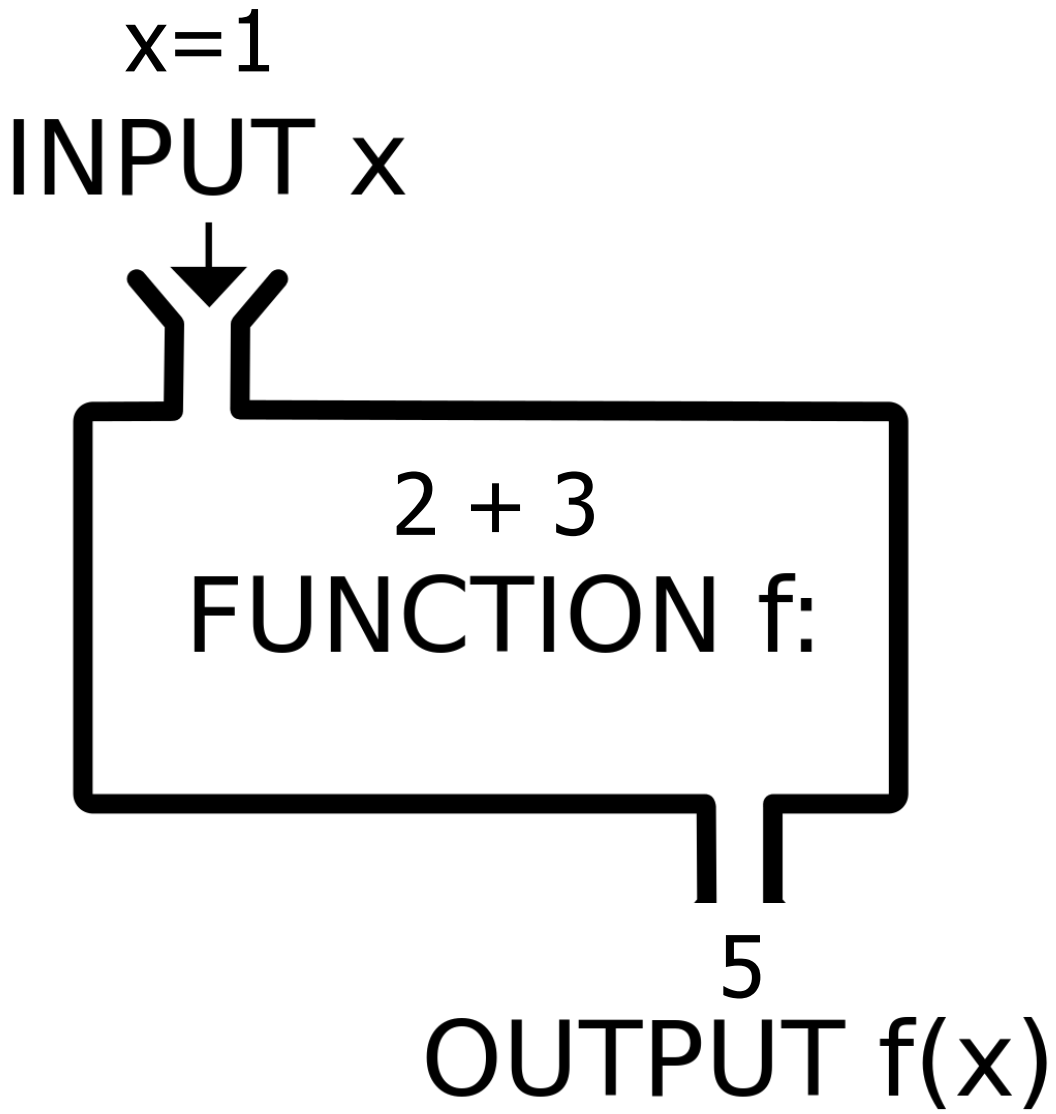
과일 주스





$$f(x) = 2x + 3$$

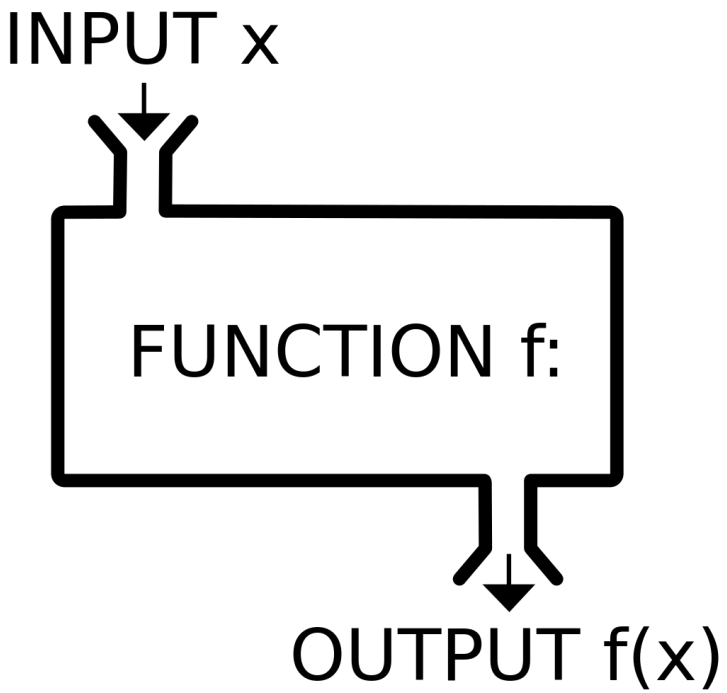
$$f(x) = 2x + 3$$



함수구조

파이썬 함수의 구조

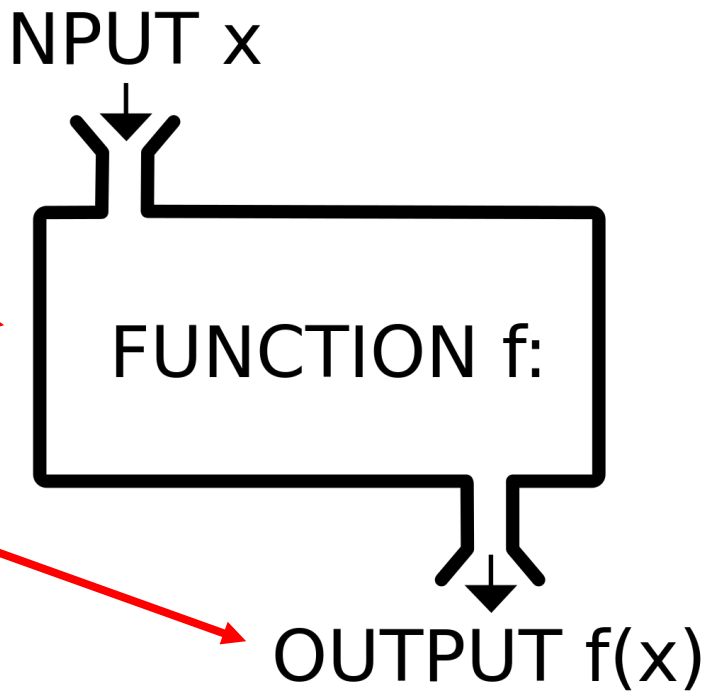
```
def 함수명(매개변수):
    <수행할 문장1>
    <수행할 문장2>
    …
    return 리턴 값
```



함수구조

파이썬 함수의 구조

```
def 함수명 (매개변수):
    <수행할 문장1>
    <수행할 문장2>
    ...
    return 리턴 값
```



일반적인 함수

```
def sum(a, b):

   result = a +b
   return result
a = sum(3, 4)
print(a)

7
```

- def를 사용하여 정의한다. def가 실행되면 함수의 객체와 참조가 같이 생성된다.
- 리턴값을 정의하지 않으면, None을 반환한다.
- 아무런 값을 반환하지 않는 함수를 *procedure*라 한다.

입력 값이 없는 함수

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

a = say()
print(a)

 Hi
```

결과 값이 없는 함수

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

sum(3, 4)

3, 4의 합은 7입니다.
```

입력 값도 결과 값도 없는 함수

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

say()
```

```
Hi
```

함수 이름

- 함수의 이름은 그 함수의 동작을 대표하는 것으로 짓는 게 좋습니다.
  (예)
  변수 : account_number → 계좌 번호 (명사)
  함수 : open_account() → 계좌를 개설하다 (동사)

```
def open_account():
    print("새로운 계좌가 생성되었습니다.")

open_account() # 앞에서 정의된 open_account() 함수 호출
```

함수 매개변수(parameter)

```
def deposit(balance, money): # 입금
    print("입금이 완료되었습니다. 잔액은 {0} 원입니다.".format(balance + money))
    return balance + money # 입금 후 잔액 정보 반환

balance = 0 # 최초 잔액
balance = deposit(balance, 1000) # 1000원 입금
print(balance) # 현재 잔액
```

- 함수를 정의할 때 전달값을 받기 위해 사용되는 balance, money 와 같은 변수를 매개변수(parameter) 라고 부르기도 합니다.

함수 반환값 처리

```
def deposit(balance, money): # 입금
    print("입금이 완료되었습니다. 잔액은 {0} 원입니다.".format(balance + money))
    return balance + money # 입금 후 잔액 정보 반환

balance = 0 # 최초 잔액
deposit(balance, 1000) # 1000원 입금
print(balance) # 현재 잔액
```

- 만약 앞에 balance = 란 부분이 없다면 어떻게 될까요?
- 반환 값이 있는 함수를 이용하며 반환 값을 사용하고자 할 때에는 반드시 값을 반환 받을 변수를 명시해야 합니다.

함수 반환값 처리

```
def withdraw(balance, money): # 출금
    if balance >= money: # 잔액이 출금보다 많으면
        print("출금이 완료되었습니다. 잔액은 {0} 원입니다.".format(balance - money))
        return balance - money # 출금 후 잔액 반환
    else:
        print("출금이 완료되지 않았습니다. 잔액은 {0} 원입니다.".format(balance))
        return balance # 기존 잔액 반환

balance = 0 # 최초 잔액
balance = deposit(balance, 1000) # 1000원 입금

# 출금 시도
balance = withdraw(balance, 2000) # 2000원 출금 시도 시 잔액이 부족하므로 실패
balance = withdraw(balance, 500) # 500원 출금 시도 시 성공
print(balance) # 현재 잔액
```

함수

```
def withdraw_night(balance, money): # 저녁에 출금
    commission = 100 # 출금 수수료 100원
    return commission, balance - money - commission # 튜플 형식으로 반환

balance = 0 # 최초 잔액
balance = deposit(balance, 1000) # 1000원 입금

# 저녁에 출금 시도
commission, balance = withdraw_night(balance, 500)
print("수수료 {0} 원이며, 잔액은 {1} 원입니다.".format(commission, balance))
```

함수 기본값

```
def profile(name, age, main_lang):
    print("이름 : {0}₩t나이 : {1}₩t주 사용 언어 : {2}".format(name, age, main_lang))

profile( " 김은연", 20, "파이썬") # 김은연씨(20세)의 주 사용 언어는 파이썬
profile( " 백설희", 25, "자바") # 백설의씨(25세)의 주 사용 언어는 자바

def profile(name, age=17, main_lang="파이썬"):
# 전달값을 따로 받지 않을 때 기본으로 사용할 값
    print("이름 : {0}₩t나이 : {1}₩t주 사용 언어 : {2}".format(name, age, main_lang))

profile("김은연") # age, main_lang 을 기본값으로 사용
profile("김은연", 20) # main_lang 을 기본값으로 사용
profile("김은연", 20, "파이썬") # 기본값 사용하지 않음
```

함수 기본값

```
def profile(name, age, main_lang): # 키워드 인자 : name, age, main_lang
    print(name, age, main_lang)

# profile("김은연", 20, "파이썬")
# profile("백설희", 25, "자바")

profile(name="김은연", main_lang="파이썬", age=20)
profile(main_lang="자바", age=25, name="백설희")
```

- 키워드 인자는 보통 어떤 함수에 전달값들이 많고 기본값들이 잘 정의되어 있을때, 대부분 기본값을 쓰고 필요한 부분만 값을 전달하고자 하는 경우에 유용합니다.
- 무엇보다 순서에 구애받지 않으므로 함수에서 사용 가능한 키워드의 종류만 알고 있다면 아주 편리하게 사용 가능하다는 장점이 있습니다.

함수 키워드 인자

```
def profile(name, age, main_lang): # 키워드 인자 : name, age, main_lang
    print(name, age, main_lang)
# profile("김은연", 20, "파이썬")
# profile("백설희", 25, "자바")
```

- 함수를 호출할 때 일반적인 전달값과 키워드 인자를 함께 사용하는 경우에는 반드시 일반 전달값들을 순서대로 먼저 적고 나서 키워드 인자들을 적어야 합니다.

```
profile("김은연", age=20, main_lang="파이썬")
# (O) 올바른 함수 호출 방법 (일반 전달값을 먼저 작성)

profile(name="백설희", 25, "파이썬")
# (X) 잘못된 함수 호출 방법 (키워드 인자 먼저 작성 후 일반 전달값 작성)
```

함수 가변 인자

```
def 함수이름(전달값1, 전달값2, ..., *가변인자):
    실행 명령문1
    실행 명령문2     ....
    return 반환값
```

```
def profile(name, age, *language): # 언어 정보를 전달하고 싶은 갯수 만큼 전달 가능
    print("이름 : {0}₩t나이 : {1}₩t".format(name, age), end=" ")
        # print(type(language)) # tuple
    for lang in language:
        print(lang, end=" ") # 언어들을 모두 한 줄에 표시
    print() # 줄바꿈 목적
```

함수 가변 인자

```
def profile(name, age, *language): # 언어 정보를 전달하고 싶은 갯수 만큼 전달 가능
    print("이름 : {0}₩t나이 : {1}₩t".format(name, age), end=" ")
        # print(type(language)) # tuple
    for lang in language:
        print(lang, end=" ") # 언어들을 모두 한 줄에 표시
    print() # 줄바꿈 목적

profile("김은연", 20, "Python", "Java", "C", "C++", "C#", "JavaScript")
# JavaScript 추가
profile("백설희", 25, "Kotlin", "Swift")
```

함수 변수(전역, 지역)

```
gun = 10 # 총 10자루

def checkpoint(soldiers): # 경계근무 나가는 군인 수
    gun = gun - soldiers # 전체 총에서 경계근무 나가는 군인 수만큼 뺀 잔여 총
    print("[함수 내] 남은 총 : {0}".format(gun))

print("전체 총 : {0}".format(gun)) # 10 자루
checkpoint(2) # 2명이 경계 근무 출발
print("남은 총 : {0}".format(gun)) # 몇 자루?
```

함수 변수(전역, 지역)

```
gun = 10

def checkpoint(soldiers):
    gun = 20 # 변수 선언 추가
    gun = gun - soldiers
    print("[함수 내] 남은 총 : {0}".format(gun))

print("전체 총 : {0}".format(gun))
checkpoint(2)
print("남은 총 : {0}".format(gun))
```

함수 변수(전역, 지역)

```
gun = 10

def checkpoint(soldiers):
    global gun # 전역공간에 있는 gun 이라는 변수를 사용
    gun = gun - soldiers
    print("[함수 내] 남은 총 : {0}".format(gun))

print("전체 총 : {0}".format(gun))
checkpoint(2)
print("남은 총 : {0}".format(gun))
```

함수 변수(전역, 지역)

```
gun = 10
def checkpoint_ret(gun, soldiers): # 전체 총 수와 군인 수를 전달받음
    gun = gun - soldiers # 전달받은 gun 을 사용
    print("[함수 내] 남은 총 : {0}".format(gun))
    return gun


print("전체 총 : {0}".format(gun))
gun = checkpoint_ret(gun, 2) # gun 값을 함수에 전달
print("남은 총 : {0}".format(gun))
```

함수를 정의할 때 가급적 외부 상황은 몰라도 되도록 전달받은 값만 이용하게끔 작성하면 보다 간결하면서도 함수의 역할에 충실할 수 있습니다.

실습

표준 체중을 구하는 프로그램을 작성하시오

* 표준 체중 : 각 개인의 키에 적당한 체중

(성별에 따른 공식)

남자 : 키(m) * 키(m) * 22

여자 : 키(m) * 키(m) * 21

조건1 : 표준 체중은 별도의 함수 내에서 계산

* 함수명 : std_weight
* 전달값 : 키(height), 성별(gender)

조건2 : 표준 체중은 소수점 둘째자리까지 표시

(출력 예제)

키 175cm 남자의 표준 체중은 67.38kg 입니다.

실습

- 입력으로 들어오는 모든 수의 평균 값을 계산해 주는 함수를 작성해 보 시오. (단 입력으로 들어오는 수의 개수는 정해져 있지 않다.)

- $n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$ 함수를 구현 하시오.

- 첫번째 매개변수에 'add'가 입력된 경우 그 다음으로 입력되는 모든 값을 더해서 돌려주고, 'mul'이 입력된 경우 그 다음으로 입력되는 모든 값을 곱해서 돌려주는 함수를 구현 하시오.

실습: 최대공약수 계산 (유클리드 호제법)

- 두 개의 자연수에 대한 최대공약수를 구하는 대표적인 알고리즘으로는 유클리드 호제법이 있습니다.
- 유클리드 호제법
  - 두 자연수 A, B에 대하여 (A > B) A를 B로 나눈 나머지를 R이라고 합시다.
  - 이때 A와 B의 최대공약수는 B와 R의 최대공약수와 같습니다.
- 유클리드 호제법의 아이디어를 그대로 함수로 작성해 보시오.
  - 예시: GCD(192, 162)

단계	A	B
1	192	162
2	162	30
3	30	12
4	12	6

실행 결과

```
6
```