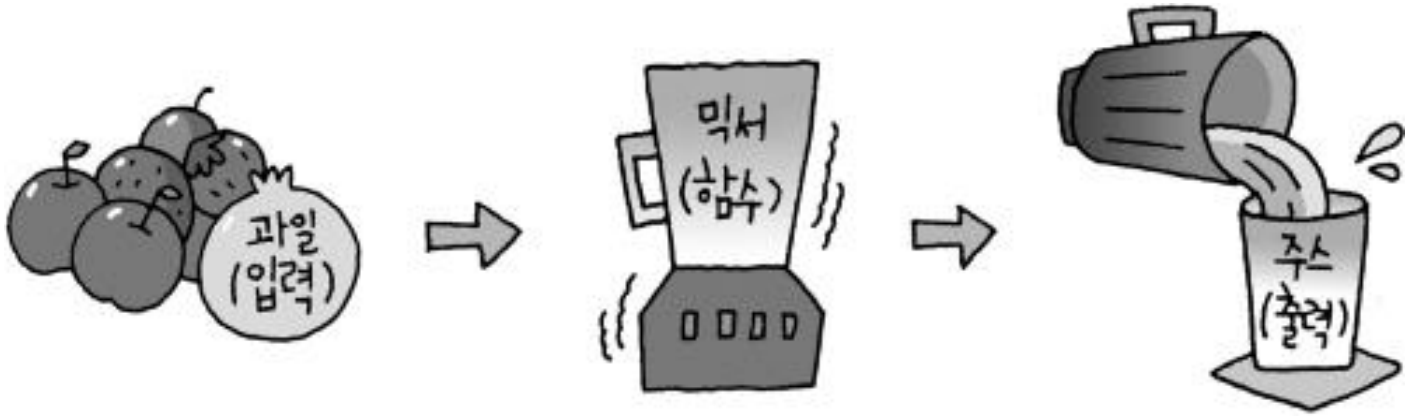
4장. 프로그램 입력과 출력

Contents

I. 함수

함수(function)는 입력 값을 가지고 어떤 일을 수행한 다음에 그 결과물을 내어놓은 것



[믹서는 과일을 입력 받아 주스를 출력하는 함수와 같다]

❖ **함수를 사용하는 이유는 무엇일까?**

- 동일한 내용을 반복해서 작성할 때 한 번만 코딩해 두고 반복해서 사용
- 프로그램을 함수화하면 프로그램의 흐름을 일목요연하게 볼 수 있음

❖ 함수의 구조

```
def 함수이름(매개변수):
    <수행할 문장1>
    <수행할 문장2>
     ...
    return 결과값
```

- def는 함수를 만들 때 사용하는 예약어임
- 함수이름은 함수를 만드는 사람이 임의로 만들 수 있음
- 매개변수는 이 함수에 입력으로 전달되는 값을 받는 변수임
- 함수를 정의한 다음, 함수에서 수행할 문장들을 입력함
- return은 함수의 결과값을 돌려주는 명령어임

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

✓ 함수이름은 add이고, 입력으로 2개의 값을 받으며, 결과값은 2개의 입력 값을 더한 값임

- 함수 사용 예

```
>>> def add(a, b):
...         return a + b
...
>>>
>>> a = 3
>>> b = 4
>>> c = add(a, b)      ← add(3, 4)의 반환 값을 c에 대입
>>> print(c)
7
```

✓ 변수 a에 3, b에 4를 대입한 다음, 앞서 만든 add 함수에 a와 b를 입력 값으로 넣어 줌

✓ 변수 c에 add 함수의 결과값을 대입하면, print(c)로 c의 값을 확인할 수 있음

❖ 매개변수와 인수

- 매개변수 : 함수에 입력으로 전달된 값을 받는 변수
- 인수 : 함수를 호출할 때 전달하는 입력 값

```
def add(a, b):          ← a, b는 매개변수
    return a + b



print(add(3,4))         ← 3, 4는 인수
```

❖ 입력값과 결과값에 따른 함수의 형태

- 함수는 들어온 입력값을 받아 어떤 처리를 하여 적절한 결과값을 돌려줌

- 함수의 형태는 입력값과 결과값의 존재 유무에 따라 4가지 유형으로 나뉨

1. 일반적인 함수

- 입력값이 있고 결과값이 있는 함수가 일반적인 함수임

```
def 함수이름(매개변수):
    수행할 문장
    ...
    return 결과값
```

- 일반적인 함수의 예

```
>>> def add(a, b):
...         result = a + b
...         return result        ← a+b의 결과값 반환
>>>
```

✓ add 함수는 2개의 입력값을 받아서 서로 더한 결과값을 돌려줌

- 일반적인 함수의 사용 예

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

✓ 입력값으로 3과 4를 주고, 결과값을 돌려받음

- 입력값과 결과값이 있는 일반적인 함수의 사용법

결과값을 받을 변수 = 함수이름(입력인수 1, 입력인수 2, …)

2. 입력값이 없는 함수

- 매개변수 부분을 나타내는 함수이름 뒤의 괄호 안이 비어 있는 함수의 예

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

- 입력값이 없는 함수의 사용 예

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

  - ✓ 위의 함수를 쓰기 위해서는 say()처럼 괄호 안에 아무런 값도 넣지 않아야 함
  - ✓ 이 함수는 입력값은 없지만 결과값으로 Hi라는 문자열을 돌려 줌
  - ✓ a = say()처럼 작성하면 a에 Hi라는 문자열이 대입되는 것임

- 입력값이 없고 결과값만 있는 함수의 사용법

```
결과값을 받을 변수 = 함수명()
```

3. 결과값이 없는 함수

- 결과값이 없는 함수의 예

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

- 결과값이 없는 함수, 호출해도 돌려주는 값이 없는 함수의 사용 예

```
>>> add(3, 4)
3, 4의 합은 7입니다.
```

- 결과값이 없는 함수의 사용법

함수이름(입력인수1, 입력인수2, …)

- 결과값이 없는지 확인하기 위한 예

```
>>> a = add(3, 4)
3, 4의 합은 7입니다.
>>> print(a)
None
```

- ✓ '3, 4의 합은 7입니다.'라는 문장을 출력해 주었는데, 왜 결과값이 없다는 것인지 의아하게 생각할 것임
- ✓ print문은 함수의 구성요소 중 하나인 '수행할 문장'에 해당하는 부분임
- ✓ 결과값은 오직 return 명령어로만 돌려받을 수 있음
- ✓ 이를 확인하기 위해 돌려받을 값을 a라는 변수에 대입하여 출력해 봄
- ✓ add 함수처럼 결과값이 없을 때, 리턴값으로 a 변수에 None을 돌려 줌

4. 입력값도 결과값도 없는 함수

- 입력값과 결과값이 없는 함수의 예

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

  - ✓ 입력 인수를 받는 매개변수도 없고 return문도 없는 함수임

- 함수의 사용 예

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

- 입력값도 결과값도 없는 함수의 사용법

```
함수명()
```

❖ 매개변수 지정하여 호출하기

- 함수를 호출할 때 매개변수를 지정할 수 있음

```
>>> def add(a, b):
...         return a + b
...
>>> result = add(a=3, b=7)
>>> print(result)
10
```

- 매개변수를 지정하면 순서에 상관없이 사용할 수 있다는 장점이 있음

```
>>> result = add(b=7, a=3)
>>> print(result)
10
```

❖ **입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?**

```
def 함수이름(*매개변수)
    수행할 문장
    ...
```

← 괄호 안의 매개변수 부분이 *매개변수로 바뀌었음

1. 여러 개의 입력값을 받는 함수 만들기

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

← *args에 입력 받은 모든 값을 더함

- ✓ *args처럼 매개변수 이름 앞에 *을 붙이면 입력값을 튜플로 만들어 주기 때문에 add_many 함수는 입력값이 몇 개이든 상관이 없음
- ✓ add_many(1,2,3)처럼 함수를 쓰면 args는 (1,2,3)이 되어 6을 돌려줌
- ✓ *args는 임의로 정한 변수명임

- 여러 개의 입력값을 받는 함수 사용 예

```
>>> result = add_many(1, 2, 3)  ← add_many 함수의 결과값을 result 변수에 대입
>>> print(result)
6
>>> result = add_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> print(result)
55
```

✓ sum_many(1, 2, 3)으로 함수를 호출하면 6을 돌려주고,
sum_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)을 대입하면 55를 돌려줌

2. 다양한 종류의 매개변수를 받는 함수 만들기

```
>>> def add_mul(choice, *args):
...         if choice == "add":      ← 매개변수 choice에 'add'을 입력 받았을 때
...             result = 0
...             for i in args:
...                 result = result + i      ← args에 입력 받은 모든 값을 더함
...         elif choice == "mul":    ← 매개변수 choice에 'mul'을 입력 받았을 때
...             result = 1
...             for i in args:
...                 result = result * i      ← args에 입력 받은 모든 값을 곱함
...          return result
...
>>>
```

- ✓ 매개변수로 choic와 *args를 받음
- ✓ add_mul 함수는 여러 개의 입력값을 의미하는 *args 매개변수 앞에 choice 매개변수가 추가됨

- 다양한 종류의 매개변수를 받는 함수 사용 예

```
>>> result = add_mul('add', 1, 2, 3, 4, 5)
>>> print(result)
15
>>> result = add_mul('mul', 1, 2, 3, 4, 5)
>>> print(result)
120
```

- ✓ 매개변수 choice에 'add'가 입력된 경우, *args에 입력되는 모든 값을 더해서 15를 돌려줌
- ✓ 매개변수 choice에 'mul'이 입력된 경우, *args에 입력되는 모든 값을 곱해서 120를 돌려줌

❖ 키워드 파라미터

- 키워드 파라미터를 사용할 때는 매개변수 앞에 별 두 개(**)를 붙임

```
>>> def print_kwargs(**kwargs):
...         print(kwargs)
...
>>>
```

  - ✓ print_kwargs 함수는 매개변수 kwargs를 출력하는 함수임

- 키워드 파라미터 사용 예

```
>>> print_kwargs(a=1)
{'a' : 1}
>>> print_kwargs(name='foo', age=3)
{'age':3, 'name':'foo'}
```

  - ✓ 입력값이 모두 딕셔너리로 만들어져서 출력된다는 것을 확인할 수 있음
  - ✓ 매개변수 이름 앞에 **을 붙이면 매개변수는 딕셔너리가 되고, 모든 key=value 형태의 결과값이 그 딕셔너리에 저장됨

❖ 함수의 결과값은 언제나 하나이다

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

← 2개의 매개변수를 받아 더한 값과 곱한 값을 돌려준다

- 이 함수를 호출하면 어떻게 될까?

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

- ✓ 결과값은 a+b와 a*b 2개인데 받아들이는 변수는 result 하나만 쓰였으니 오류가 발생하지 않을까? 의문을 가지나 오류는 발생하지 않음
- ✓ 그 이유는 함수의 결과값은 2개가 아니라 언제나 1개라는 데 있음
- ✓ add_and_mul 함수의 결과값 a+b와 a*b는 튜플값 하나인 (a+b, a*b)로 돌려줌
- ✓ 따라서 result 변수는 다음과 같은 값을 갖게 됨

```
result = (7, 12)
```

- ✓ 결과값으로 (7, 12)라는 튜플 값을 갖게 되는 것임

- 하나의 튜플 값을 2개의 결과값처럼 받고 싶다면 다음과 같이 함수를 호출하면 됨

```
>>> result1, result2 = add_and_mul(3,4)
```

  - ✓ result1, result2 = (7, 12)가 되어 result1은 7이 되고 result2은 12가 됨

- 아래와 같이 return문을 2번 사용하면, 2개의 결과값을 돌려주지 않을까?

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

  - ✓ add_and_mul 함수를 호출하면 다음과 같음

```
>>> result = add_and_mul(2,3)
>>> print(result)
5
```

  - ✓ 두 번째 return문인 return a*b는 실행되지 않음
  - ✓ 함수는 return문을 만나는 순간 결과값을 돌려준 다음 함수를 빠져나가게 됨

❖ return의 또 다른 쓰임새

- 특별한 상황일 때 함수를 빠져나가고자 싶다면, return을 단독으로 써서 함수를 즉시 빠져 나갈 수 있음

```
>>> def say_nick(nick):
...         if nick == "바보":
...             return
...         print("나의 별명은 %s입니다." % nick)
...
>>>
```

✓ 위 함수는 '별명'을 입력으로 전달받아 출력하는 함수로 반환 값은 없음

✓ 만약 입력값으로 '바보'라는 값이 들어오면 문자열을 출력하지 않고 함수를 즉시 빠져 나감

```
>>> say_nick('야호')
나의 별명은 야호입니다.
>>> say_nick('바보')
>>>
```

✓ return으로 함수를 빠져나가는 방법은 실제 프로그래밍에서 자주 사용됨

❖ 매개변수에 초기값 미리 설정하기

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

- ✓ 위 함수는 매개변수가 name, old, man=True 이렇게 3개임
- ✓ man=True처럼 매개변수에 미리 값을 넣어주는 것이 함수의 매개변수 초기값을 설정하는 방법임
- ✓ 함수의 매개변수에 들어갈 값이 항상 변하는 것이 아닐 경우에는 함수의 초기값을 미리 설정해 두면 유용함

- 매개변수에 초기값 설정하는 사용 예

```
say_myself("박응용", 27)
say_myself("박응용", 27, True)
```

✓ 입력값으로 "박응용", 27처럼 2개를 주면, name에는 "박응용"이 old에는 27이 대입되고 man이라는 변수에는 입력값을 주지 않았지만 초기값인 True를 갖게 됨

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

✓ 위의 예에서 함수를 사용한 2가지 방법은 모두 동일한 결과를 출력함

```
say_myself("박응용", 27, False)
```

✓ 초기값이 설정된 부분을 False로 바꿈

✓ man 변수에 False 값이 대입되어 다음과 같은 결과가 출력됨

```
나의 이름은 박응용입니다.
나이는 27살입니다.
여자입니다.
```

- 함수의 매개변수에 초기값 설정할 때 주의사항

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

```
say_myself("박응용", 27)
```

```
Syntax Error: non-default argument follows default argument
```

- ✓ 이전 함수와 바뀐 부분은 초기값을 설정한 매개변수의 위치이며, 함수를 실행할 때 오류가 발생함
- ✓ 위의 함수를 호출한다면 name 변수에는 "박응용"이 들어가고, 27을 man 변수와 old 변수 중 어느 곳에 대입해야 할지 알 수 없게 됨
- ✓ 오류 메시지는 초기값을 설정해 놓은 매개변수 뒤에 초기값을 설정해 놓지 않은 매개변수는 사용할 수 없다는 뜻임
- ✓ **초기화시키고 싶은 매개변수는 항상 뒤쪽에 놓아야 함**

❖ 함수 안에서 선언한 변수의 효력 범위

- 함수 안에서 사용할 변수의 이름을 함수 밖에서도 동일하게 사용한다면 어떻게 될까?

```
#vartest.py
a = 1        ← 함수 밖의 변수 a
def vartest(a):
    a = a +1     } vartest 함수 선언


vartest(a)     ← vartest 함수의 입력값으로 a를 줌
print(a)       ← a의 값 출력
```

- ✓ vartest 함수에서 매개변수 a의 값에 1을 더했으니까 2가 출력될 것 같지만 실행시켜보면 결과값은 1이 나옴
- ✓ 함수 안에서 새로 만든 매개변수는 함수 안에서만 사용하는 '함수만의 변수'이기 때문임
- ✓ 즉, def vartest(a)에서 입력값을 전달받는 매개변수 a는 함수 안에서만 사용하는 변수이지 함수 밖의 변수 a가 아니라는 뜻임
- ✓ 함수 안에서 사용하는 매개변수는 함수 밖의 변수 이름과는 전혀 상관이 없음

```
#vartest_error.py
def vartest(a):
    a = a +1

vartest(3)
print(a)
```

- ✓ vartest(3)을 수행하면 vartest라는 함수 안에서 a는 4가 되지만 함수를 호출하고 난 뒤에 print(a)라는 문장은 에러가 발생하게 됨
- ✓ 이유는 print(a)에서 입력받아야 하는 a 변수를 어디에서도 찾을 수 없기 때문임
- ✓ 함수 안에서 선언된 변수는 함수 안에서만 사용될 뿐 함수 밖에서는 사용되지 않음

❖ 함수 안에서 함수 밖의 변수를 변경하는 방법

- vartest라는 함수를 사용해서 함수 밖의 변수 a를 1만큼 증가시킬 수 있는 방법

1. return 사용하기

```
#vartest_return.py
a = 1
def vartest(a):
    a = a +1
    return a

a = vartest(a)      ← vartest(a) 결과값을 함수 밖의 변수 a에 대입
print(a)
```

- ✓ vartest 함수는 입력으로 들어온 값에 1을 더한 값을 돌려 줌
- ✓ 따라서 a = vartest(a)라 대입하면, a가 vartest 함수의 결과값으로 바뀜
- ✓ 여기서도 함수 안의 a 매개변수는 함수 밖의 a와는 다른 것임

2. global 명령어 이용하기

```
#vartest_global.py
a = 1
def vartest():
    global a
    a = a +1

vartest()
print(a)
```

- ✓ vartest 함수 안의 global a 문장은 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 의미임
- ✓ 함수는 독립적으로 존재하는 것이 좋기 때문에 프로그래밍을 할 때 global 명령어는 사용하지 않는 것이 좋음

❖ lambda

- lambda(람다)는 함수를 생성할 때 사용하는 예약어로 def와 동일한 역할을 함
- 함수를 한 줄로 간결하게 만들 때 사용함
- def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰임

lambda 매개변수1, 매개변수2, ... : 매개변수를 사용한 표현식

```
>>> add = lambda a, b: a+b
>>> result = add(3, 4)
>>> print(result)
7
```

def 함수와 동일

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

- ✓ add는 두 개의 인수를 받아 서로 더한 값을 돌려주는 lambda 함수임
- ✓ lambda 예약어로 만든 함수는 return 명령어가 없어도 결과값을 돌려줌

응용예제01. 커피 자판기 프로그램

✓ 커피 자판기가 없는 상황에서 손님에게 커피를 서비스하는 과정이다. 커피 한 잔을 서비스하려면 여러 과정을 거쳐야 하는데, 함수를 만들어 프로그램으로 구현해 보자.

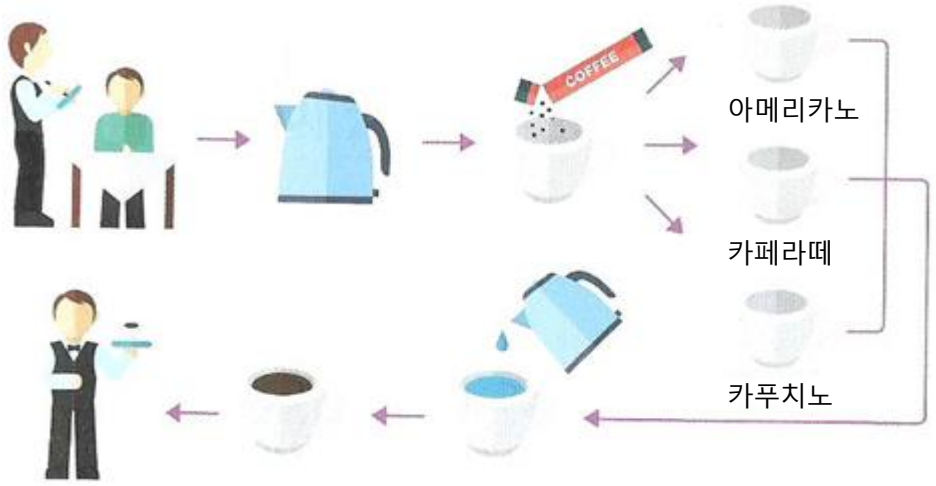
```
#coffee01.py
coffee = 0

coffee = int(input("어떤 커피 드릴까요?(1:아메리카노, 2:카페라떼, 3:카푸치노)"))

print()
print("#1. 뜨거운 물을 준비한다.")
print("#2. 종이컵을 준비한다.")

if coffee ==1:
    print("#3. 아메리카노를 탄다.")
elif coffee == 2:
    print("#3. 카페라떼를 탄다.")
elif coffee == 3:
    print("#3. 카푸치노를 탄다.")
else:
    print("#3. 아무거나 탄다.")

print("#4. 물을 붓는다.")
print("#5. 스푼으로 젓는다.")
print()
print("손님~커피 여기 있습니다.")
```



- 표준 입력(standard input) : 키보드로 입력
- 표준 출력(standard output) : 모니터로 출력

❖ 사용자 입력

사용자가 입력한 값을 어떤 변수에 대입하고 싶을 때는 어떻게 해야 할까?

▪ input의 사용

```
>>> a = input()
Life is too short, you need python
>>> a
'Life is too short, you need python'
>>>
```

사용자가 입력한 문장을 a에 대입

✓ input은 입력되는 모든 것을 문자열로 취급함

▪ 프롬프트를 띄워서 사용자 입력 받기

```
input("질문 내용")
```

✓ 사용자에게 입력 받을 때 안내문구 또는 질문이 나오도록 하고 싶을 때는 input() 괄호 안에 질문을 입력하여 프롬프트를 띄워 줌

- **프롬프트를 띄워서 사용자 입력 받기 사용 예**

```
>>> number = input("숫자를 입력하세요: ")
숫자를 입력하세요: 3
>>> print(number)
3
>>>
```

  - ✓ 숫자를 입력하라는 프롬프트에 3을 입력하면 변수 number에 3이 대입됨
  - ✓ print(number)로 출력해서 입력을 확인함

❖ print 자세히 알기

- 지금까지 print문이 수행해 온 일은 입력한 자료형을 출력하는 것임

```
>>> a = 123
>>> print(a)  ← 숫자 출력하기
123
>>> a = "Python"  ← 문자열 출력하기
>>> print(a)
Python
>>> a = [1, 2, 3]
>>> print(a)  ← 리스트 출력하기
[1, 2, 3]
```

- **큰따옴표(“)로 둘러싸인 문자열은 +연산과 동일하다**

```
>>> print("life" "is" "too short")
lifeistoo short
>>> print("life"+"is"+"too short")
lifeistoo short
```

  - ✓ 위의 예들은 동일한 결과값을 출력함
  - ✓ 즉, 따옴표로 둘러싸인 문자열을 연속해서 쓰면 +연산을 한 것과 같음

- **문자열 띄어쓰기는 콤마로 한다**

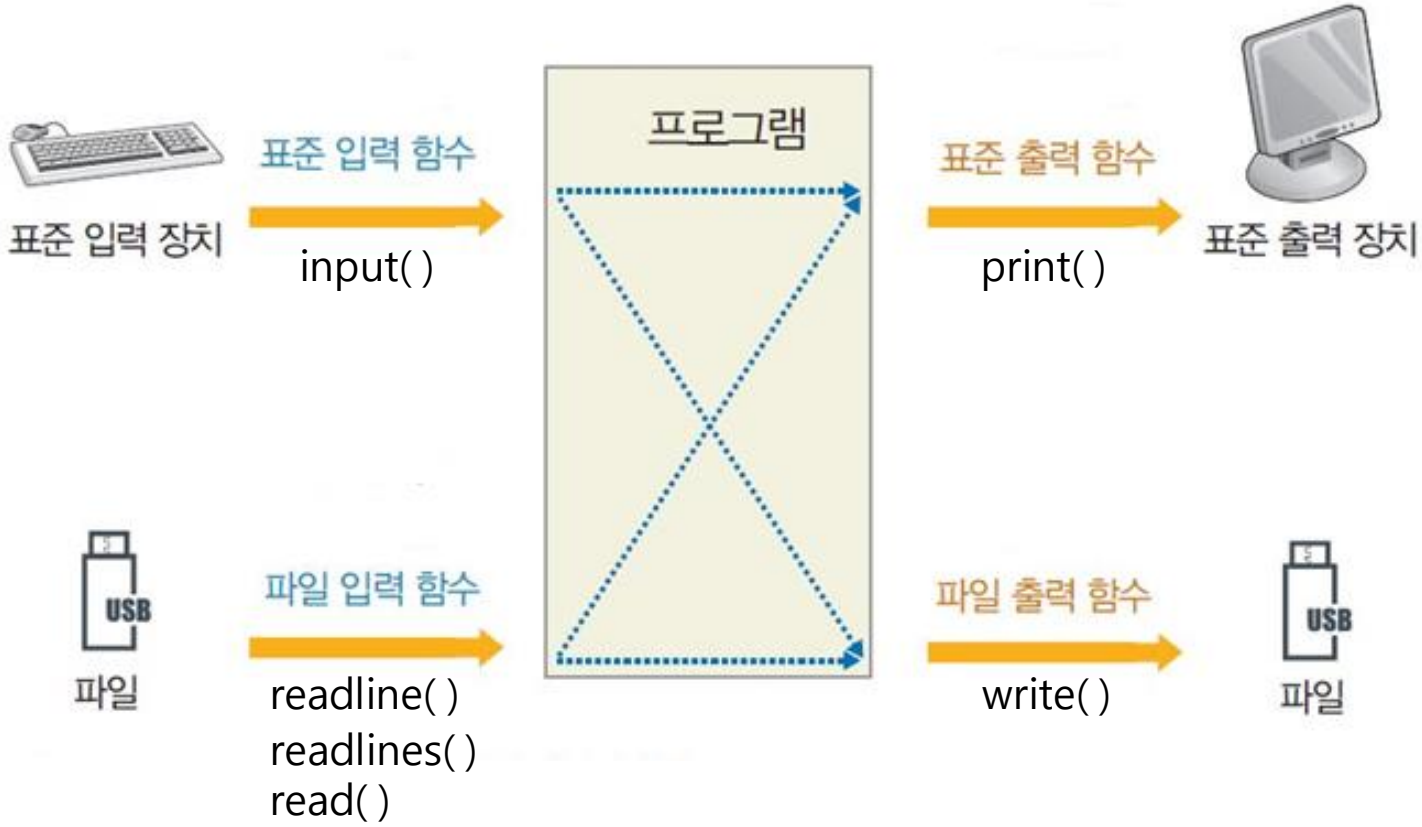
```
>>> print("life", "is", "too short")
life is too short
```

  - ✓ 콤마(,)를 사용하면 문자열 사이에 띄어쓰기를 할 수 있음

- **한 줄에 결과값 출력하기**

```
>>> for i in range(10):
...         print(i, end=' ')
...
0 1 2 3 4 5 6 7 8 9
```

  - ✓ 한 줄에 결과값을 출력하려면, 매개변수 end를 사용해 끝 문자를 지정해야 함


표준 입력 장치
표준 입력 함수
input( )
프로그램
표준 출력 함수
print( )
표준 출력 장치
USB
파일
파일 입력 함수
readline( )
readlines( )
read( )
파일 출력 함수
write( )
USB
파일

❖ 파일 생성하기

- 파일을 생성하기 위해 open이라는 내장 함수를 사용함

```
f = open("새파일.txt", 'w')
f.close()
```

- open 함수 사용법

파일 객체 = open(파일 이름, 파일 열기 모드)

✓ open 함수는 '파일 이름'과 '파일 열기 모드'를 입력값으로 받고, 결과값으로 파일 객체를 돌려 줌

- 파일 열기 모드

파일 열기 모드	설명
r	읽기 모드 - 파일을 읽기만 할 때 사용
w	쓰기 모드 - 파일에 내용을 쓸 때 사용
a	추가 모드 - 파일의 마지막에 새로운 내용을 추가할 때 사용

  - ✓ 파일을 쓰기 모드로 열게 되면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고, 해당 파일이 존재하지 않으면 새로운 파일이 생성됨

- 파일을 특정 디렉터리에 생성하는 예

```
f = open("C:₩doit₩새파일.txt", 'w')
f.close()
```

  - ✓ f.close()는 열려 있는 파일 객체를 닫아 주는 역할을 함
  - ✓ 쓰기 모드로 열었던 파일을 닫지 않고 다시 사용하려고 하면 오류가 발생하기 때문에 close()를 사용해서 파일을 닫아 주는 것이 좋음

❖ 파일을 쓰기 모드로 열어 출력값 적기

```
#writedata.py
f = open("C:₩doit₩새파일.txt", 'w')
for i in range(1, 11):          ← 1부터 10까지 i에 대입
    data = "%d번째 줄입니다.₩n" % i
    f.write(data)               ← data를 파일 객체 f에 씀
f.close()
```

- 위의 예제를 실행함

```
C:₩doit>python writedata.py
C:₩doit>
```

✓ 프로그램을 실행한 C:₩doit 디렉터리에 새파일.txt라는 파일이 생성되었고, 새파일.txt 파일에는 모니터에 출력될 내용이 들어 있는 것을 볼 수 있음

III. 파일 읽고 쓰기

❖ **프로그램의 외부에 저장된 파일을 읽는 여러 가지 방법**

1. readline() 함수 이용하기

```
#readline.py
f = open("C:₩doit₩새파일.txt", 'r')
line = f.readline()
print(line)
f.close()
```

- f.open("새파일.txt", 'r')로 파일을 읽기 모드로 연 후 readline()을 이용해서 **파일의 첫 번째 줄을 읽어 출력**하는 경우임
- 앞에서 만들었던 새파일.txt를 수정하거나 삭제하지 않았다면 프로그램을 실행시켰을 때 새파일.txt의 첫 번째 줄이 화면에 출력될 것임

```
1번째 줄입니다.
```

- 모든 줄을 읽어서 화면에 출력하고 싶다면 다음과 같이 작성하면 됨

```
#readline_all.py
f = open("C:₩doit₩새파일.txt", 'r')
while True:
    line = f.readline()
    if not line: break
    print(line)
f.close()
```

✓ while True: 라는 무한 루프 안에서 f.readline()을 이용해 파일을 계속해서 한 줄씩 읽어 들이고, 만약 더 이상 읽을 줄이 없으면 break를 수행함

```
while 1:
    data = input()
    if not data: break
    print(data)
```

✓ 사용자 입력을 받아서 출력하는 경우와 파일을 읽어서 출력하는 예제를 비교함

2. readlines() 함수 이용하기

```
#readlines.py
f = open("C:₩doit₩새파일.txt", 'r')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

- **파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트로 돌려줌**
- lines는 리스트 ["1 번째 줄입니다.", "2 번째 줄입니다.", , , "10 번째 줄입니다."]가 됨

3. read() 함수 이용하기

```
#read.py
f = open("C:₩doit₩새파일.txt", 'r')
data = f.read()
print(data)
f.close()
```

- **파일의 내용 전체를 문자열로 돌려줌**
- data는 파일의 전체 내용임

❖ 파일에 새로운 내용 추가하기

- 원래 있던 값을 유지하면서 새로운 값만 추가할 때는 파일을 추가 모드('a')로 열면 됨

```
#adddata.py
f = open("C:₩doit₩새파일.txt", 'a')
for i in range(11, 20):      ← 11부터 19까지 i에 대입
    data = "%d번째 줄입니다.₩n" % i
    f.write(data)
f.close()
```

- 파일을 추가 모드('a')로 열고, write를 이용해서 결과값을 기존 파일에 추가함
- 추가 모드로 열었기 때문에 새파일.txt 파일이 원래 가지고 있던 내용 바로 다음부터 결과값을 적기 시작함
- 위의 예제를 실행함

```
C:₩doit>python adddata.py
C:₩doit>
```

✓ 새파일.txt 파일을 보면 원래 있던 내용 뒤에 새로운 내용이 추가됨

❖ with문과 함께 사용하기

- 파일을 열면 아래와 같이 항상 close해 주는 것이 좋음

```
f = open("C:₩doit₩foo.txt", 'w')
f.write("Life is too short, you need python")
f.close()
```

- 하지만 이렇게 파일을 열고 닫는 것을 자동으로 처리하는 것이 편리하지 않을까?

```
with open("C:₩doit₩foo.txt", 'w') as f:
    f.write("Life is too short, you need python")
```

✓ 위와 같이 with문을 사용하면 with 블록을 벗어나는 순간 열린 파일 객체 f가 자동으로 close되어 편리함

❖ sys 모듈로 매개변수 주기

- 명령 프롬프트에서 명령어는 명령행(명령 프로프트 창)에서 매개변수를 직접 주어 프로그램을 실행하는 방식을 따름

명령 프롬프트 명령어 [인수1 인수2 ...]

- sys 모듈을 사용하여 입력 인수를 직접 줄 수 있음
- sys 모듈을 사용하려면 import sys처럼 import라는 명령어를 사용해야 함

```
#sys1.py
import sys

args = sys.argv[1:]
for i in args:
    print(i)
```

- ✓ 입력받은 인수를 for문을 사용해 차례대로 하나씩 출력하는 예임
- ✓ sys 모듈의 argv는 명령어 창에서 입력한 인수를 의미함
- ✓ argv[0]는 파일 이름인 sys1.py가 되고, argv[1]부터는 뒤에 따라오는 인수가 차례로 argv의 요소가 됨

- 매개변수를 함께 주어 실행시키면 다음과 같은 결과값을 얻을 수 있음

```
C:₩doit>python  sys1.py  aaa  bbb  ccc
aaa
bbb
ccc
```

- 문자열 관련 함수인 upper()를 이용하여 명령 행에 입력된 소문자를 대문자로 바꾸어 주는 프로그램임

```
#sys2.py
import sys

args = sys.argv[1:]
for i in args:
    print(i.upper(), end=' ')
```

- 명령 프롬프트 창에서 다음과 같이 입력해 봄

```
C:₩doit>python sys2.py  life is too short, you need python
```

- 결과는 다음과 같음

```
LIFE IS TOO SHORT, YOU NEED PYTHON
```