세상의 속도를
따라잡고 싶다면

점프 투 파이썬

박응용 지음(위키독스 운영자)

이지스퍼블리싱

04

파이썬의 입출력

04-1 함수

04-1 함수

▪ 함수란 무엇인가?

- 우리는 믹서에 과일을 넣고, 믹서를 사용해서 과일을 갈아 과일 주스를 만듦

- 믹서에 넣는 과일 = 입력
- 과일주스 = 출력(결괏값)
- 믹서 = ?

믹서는 과일을 입력받아 주스를 출력하는 함수와 같다.

04-1 함수

- **함수를 사용하는 이유는 무엇일까?**
  - 반복되는 부분이 있을 경우 '반복적으로 사용되는 가치 있는 부분'을 한 뭉치로 묶어 '어떤 입력값을 주었을 때 어떤 결괏값을 리턴해 준다'라는 식의 함수로 작성하는 것이 현명함
  - 프로그램의 흐름을 파악하기 좋고 오류 발생 지점도 찾기 쉬움

04-1 함수

▪ 파이썬 함수의 구조

- def : 함수를 만들 때 사용하는 예약어
- 함수 이름은 임의로 생성 가능
- 매개변수는 함수에 입력으로 전달되는 값을 받는 변수
- return : 함수의 결괏값(리턴값)을 돌려주는 명령어

```
def 함수_이름(매개변수):
    수행할_문장1
    수행할_문장2
    ...
```

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

- 함수의 풀이

이 함수의 이름은 add이고 입력으로 2개의 값을 받으며 리턴값(출력값)은 2개의 입력값을 더한 값이다.

04-1 함수

파이썬 함수의 구조

- 예) add 함수
  - add 함수 만들기

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

  - add 함수 사용하기

```
>>> a = 3
>>> b = 4
>>> c = add(a, b)  <-- add(3, 4)의 리턴값을 c에 대입
>>> print(c)
7
```

04-1 함수

매개변수와 인수

- **매개변수**와 인수는 혼용해서 사용되는 헷갈리는 용어로 잘 구분하는 것이 중요!

- **매개변수(parameter)**
  - 함수에 입력으로 전달된 값을 받는 변수

- **인수(arguments)**
  - 함수를 호출할 때 전달받는 입력값

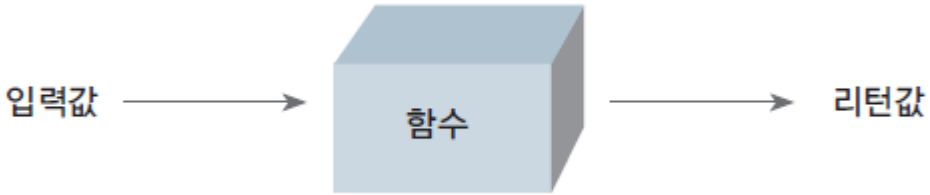
```
def add(a, b): ← a, b는 매개변수
    return a + b

print(add(3, 4)) ← 3, 4는 인수
```

04-1 함수

▪ 입력값과 리턴값에 따른 함수의 형태

- 함수는 들어온 입력값을 받은 후 어떤 처리를 하여 적절한 값을 리턴해 줌



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

04-1 함수

▪ 입력값과 리턴값에 따른 함수의 형태

- 일반적인 함수
  - 입력값이 있고 리턴값이 있는 함수
  - 일반 함수의 전형적인 예
    - add 함수는 2개의 입력값을 받아 서로 더한 결괏값을 리턴함

```
def 함수_이름(매개변수):
    수행할_문장
    ...
    return 리턴값
```

```
>>> def add(a, b):
...     result = a + b
...     return result  <— a + b의 결괏값 리턴
```

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

```
리턴값을_받을_변수 = 함수_이름(입력_인수1, 입력_인수2, ...)
```

04-1 함수

입력값과 리턴값에 따른 함수의 형태

- 입력값이 없는 함수
  - 입력값이 없는 함수도 존재함

  - say 함수는 매개변수 부분을 나타내는 함수 이름 뒤의 괄호 안이 비어있음
  - 함수 사용 시 say( )처럼 괄호 안에 아무 값도 넣지 않아야 함

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

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

```
리턴값을_받을_변수 = 함수_이름()
```

04-1 함수

입력값과 리턴값에 따른 함수의 형태

리턴값이 없는 함수

- 리턴값이 없는 함수는 호출해도 리턴되는 값이 없음

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

```
>>> add(3, 4)
3, 4의 합은 7입니다.
```

- 사용 방법

```
함수_이름(입력_인수1, 입력_인수2, ...)
```

04-1 함수

▪ 입력값과 리턴값에 따른 함수의 형태

- 리턴값이 없는 함수
  - 리턴값이 진짜 없을까?
    - 리턴받을 값을 a 변수에 대입하여 출력하여 확인
    - None이란 거짓을 나타내는 자료형으로, 리턴값이 없을 때 쓰임

```
>>> a = add(3, 4) <— add 함수의 리턴값을 a에 대입
3, 4의 합은 7입니다.
>>> print(a) <— a 값 출력
None
```

04-1 함수

입력값과 리턴값에 따른 함수의 형태

- 입력값도, 리턴값도 없는 함수
  - 입력 인수를 받는 매개변수도 없고 return 문도 없는, 즉 입력값도 리턴값도 없는 함수

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

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

```
함수_이름()
```

04-1 함수

▪ 매개변수를 지정하여 호출하기

- 함수 호출 시 매개변수 지정 가능
  - 예) sub 함수

```
>>> def sub(a, b):
...     return a - b
```

  - 매개변수를 지정하여 사용

```
>>> result = sub(a=7, b=3) <— a에 7, b에 3을 전달
>>> print(result)
4
```

- 매개변수를 지정하면 매개변수 순서에 상관없이 사용할 수 있는 장점

```
>>> result = sub(b=5, a=3) <— b에 5, a에 3을 전달
>>> print(result)
-2
```

04-1 함수

- **입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?**
  - 파이썬에서의 해결 방법
    - 일반 함수 형태에서 괄호 안의 매개변수 부분이 *매개변수로 바뀜

```
def 함수_이름(*매개변수):
    수행할_문장
    ...
```

04-1 함수

- **입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?**
  - 여러 개의 입력값을 받는 함수 만들기
    - 매개변수 이름 앞에 *을 붙이면 입력값을 전부 모아 튜플로 만들어 줌

```
>>> def add_many(*args):
...     result = 0
...     for i in args:
...         result = result + i  <- *args에 입력받은 모든 값을 더한다.
...     return result
```

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

04-1 함수

▪ 입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?

- 여러 개의 입력값을 받는 함수 만들기
  - *args 매개변수 앞에 choice 매개변수를 추가할 수도 있음

```
>>> def add_mul(choice, *args):
...     if choice == "add":  <- 매개변수 choice에 "add"를 입력받았을 때
...         result = 0
...         for i in args:
...             result = result + i  <- args에 입력받은 모든 값을 더한다.
...     elif choice == "mul":  <- 매개변수 choice에 "mul"을 입력받았을 때
...         result = 1
...         for i in args:
...             result = result * i  <- *args에 입력받은 모든 값을 곱한다.
...     return result
```

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

04-1 함수

키워드 매개변수, kwargs

- 매개변수 앞에 별 2개(**)를 붙임

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

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

04-1 함수

▪ 함수의 리턴값은 언제나 하나이다

- 2개의 입력 인수를 받아 리턴하는 함수

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

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

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

- 하나의 튜플 값을 2개의 값으로 분리하여 리턴

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

- return 문을 2번 사용하면 리턴값은 하나뿐

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

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

04-1 함수

▪ 매개변수에 초깃값 미리 설정하기

- 매개변수에 초깃값을 미리 설정

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

- man=True처럼 매개변수에 미리 값을 넣어 함수의 매개변수 초깃값을 설정

04-1 함수

▪ 매개변수에 초깃값 미리 설정하기

- 매개변수에 들어갈 값이 항상 변하는 것이 아니면, 초깃값을 미리 설정하는 것이 유용함
- say_myself 함수 사용법

```
say_myself("박응용", 27)
```

```
say_myself("박응용", 27, True)
```

실행 결과

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

```
say_myself("박응선", 27, False)
```

실행 결과

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

- 입력값으로 ("박응용", 27)처럼 2개를 주면 name에는 "박응용"이 old에는 27이 대입됨
- man이라는 변수에는 입력값을 주지 않았지만 초깃값 True를 갖게 됨

04-1 함수

매개변수에 초깃값 미리 설정하기

- 주의할 점
  - 초기화하고 싶은 매개변수는 항상 뒤쪽에 놓아야 함

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

- 파이썬 인터프리터는 27을 man 매개변수와 age 매개변수 중 어느 곳에 대입해야 할지 판단이 어려워 오류 발생

```
say_myself("박응용", 27)
```

실행 결과

```
SyntaxError: non-default argument follows default argument
```

04-1 함수

▪ 함수 안에서 선언한 변수의 효력 범위

- 함수 안에서 사용할 변수의 이름을 함수 밖에서도 동일하게 사용한다면?

```
a = 1              # 함수 밖의 변수 a
def vartest(a):    # vartest 함수 선언
    a = a + 1

vartest(a)         # vartest 함수의 입력값으로 a를 대입
print(a)           # a 값 출력
```

- 함수를 실행해 보면, 결괏값은 1이 나옴
- 함수 안에서 사용하는 매개변수는 함수 안에서만 사용하는 '함수만의 변수'이기 때문
- 즉, 매개변수 a는 함수 안에서만 사용하는 변수일 뿐, 함수 밖의 변수 a와는 전혀 상관없음
- 따라서 vertest 함수는 매개변수 이름을 바꾸어도 이전 함수와 동일하게 동작함

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

04-1 함수

▪ 함수 안에서 선언한 변수의 효력 범위

- 함수 안에서 선언한 매개변수는 함수 안에서만 사용될 뿐,
  함수 밖에서는 사용되지 않음

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

vartest(3)
print(a)
```

  - print(a)에서 사용한 a 변수는 어디에도 선언되지 않았기 때문

04-1 함수

▪ 함수 안에서 함수 밖의 변수를 변경하는 방법

1. return 사용하기

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

a = vartest(a)    # vartest(a)의 리턴값을 함수 밖의 변수 a에 대입
print(a)
```

2. global 명령어 사용하기

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

vartest()
print(a)
```

- 단, 함수는 독립적으로 존재하는 것이 좋기 때문에 이 방법은 피하는 것이 좋음

04-1 함수

lambda 예약어

- 함수를 생성할 때 사용하는 예약어
- 함수를 한 줄로 간결하게 만들 때 사용
- def와 동일한 역할
- 우리말로 '람다'
- def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰임

```
함수_이름 = lambda 매개변수1, 매개변수2, ... : 매개변수를_이용한_표현식
```

04-1 함수

lambda 예약어

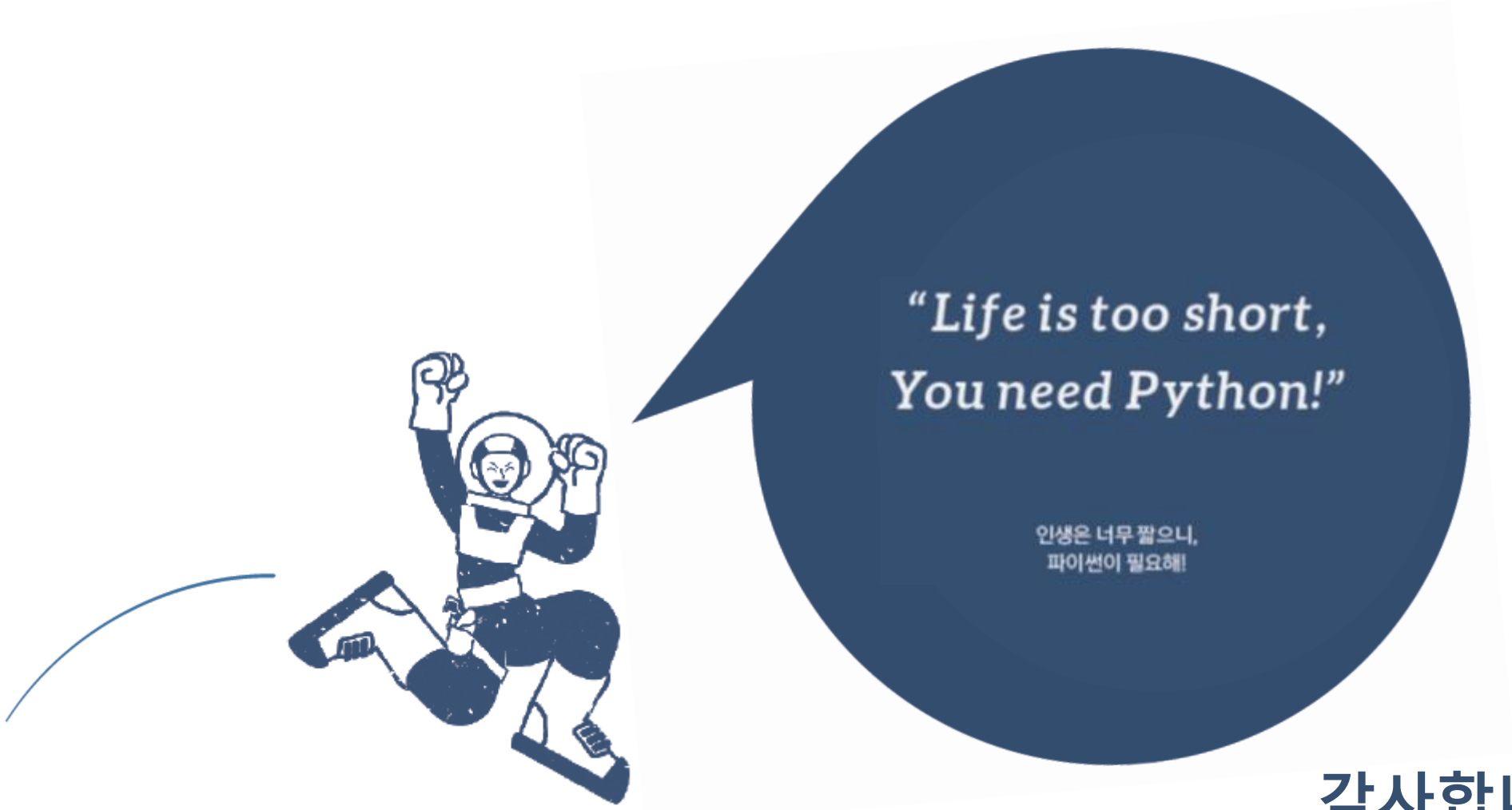
- 사용 예시

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

  - add는 2개의 인수를 받아
    서로 더한 값을 리턴하는 lambda 함수

- def를 사용한 경우와 하는 일이 완전히 동일함

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



감사합니다.