프로그래밍

이은섭
(Eunseop Lee)

Computer Science and Engineering,
Pohang University of Science and Technology
eseop90@gmail.ac.kr

함수(Function)

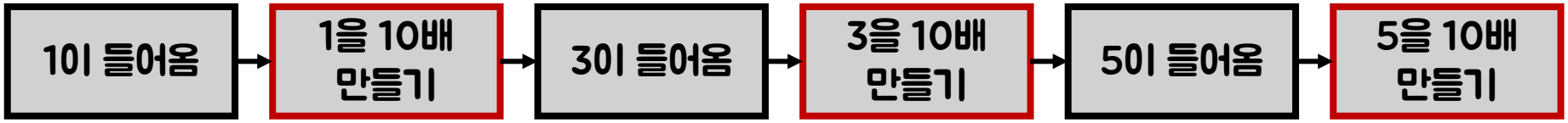
함수란 무엇인가

- 함수란 입력값을 주었을 때, 결과값을 출력하는 것

- 과일 =“입력”
- 과일 주스 = “출력(결과값)”
- 믹서 =“함수”

함수를 사용하는 이유

- 반복적으로 사용되는 가치 있는 부분이 있는 경우



- 프로그램을 기능 단위의 함수로 분리하여 프로그램 흐름을 잘 파악하기 위한 경우

```
data = read_file(file_path='data')
new_data = process_file(data)
save_file(new_data)
```

```
def read_file(file_path):
    # file_path에 해당 file이 있는지 확인 후, file을 읽음
    if not osp.isfile(file_path):
        print(f'{file_path} is not exist')
    else:
        with open(file_path, 'r') as f:
            return f.readlines()

def save_file(file, out_path):
    # file을 out_path에 저장하는 함수
    if not osp.isdir(out_path):
        os.makedirs(out_path)

    with open(out_path, 'w') as f:
        f.write(file)

def process_file(file):
    # file의 값들을 처리함
    print(file)
    return file
```

함수

- 함수의 구조

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

  - 예약어인 def로 시작함
  - 함수 이름은 함수를 만드는 사람이 임의로 만들 수 있음
  - 함수 이름 뒤 괄호 안의 매개변수는 이 함수에 입력으로 전달되는 값을 받는 변수

함수

- 함수의 구조

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

  - 이 함수의 이름은 'add'이고 입력으로 2개의 값을 받으며 출력값은 2개의 입력값을 더한 값

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

함수

- 매개변수와 인수
  - 매개변수(parameter)는 함수에 입력으로 전달된 값을 받는 변수, 인수(argument)는 함수를 호출할 때 전달하는 입력값을 의미

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

print(add(3, 4))  # 3, 4는 인수
```

함수

- 입력값과 리턴값에 따른 함수의 형태
  - 함수는 들어온 입력값을 받은 후 어떤 처리를 하여 적절한 값을 출력(리턴)함

  - 함수의 형태는 입력값과 리턴값의 존재 유무에 따라 4가지 유형을 가짐
    1. 일반적인 함수(입력값과 출력값이 있는 함수)
    2. 입력값이 없는 함수
    3. 출력값이 없는 함수
    4. 입력값도, 출력값도 없는 함수

함수

1. 일반적인 함수

- 입력값이 있고 출력값이 있는 함수

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

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

함수

2. 입력값이 없는 함수

- 입력값이 없고 출력값이 있는 함수

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

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

함수

3. 출력값이 없는 함수

- 입력값이 있고 출력값이 없는 함수

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

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

- '3, 4의 합은 7입니다' 문장은 add 함수 내에서 실행이 되는 것이며, add 함수에서 return 명령어를 사용하지 않았기에, 출력값은 없음

함수

3. 입력값도, 리턴값도 없는 함수

- 입력값과 출력값이 없는 함수

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

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

함수

- 매개변수를 지정하여 호출하기
  - 함수를 호출할 때, 매개변수를 지정할 수 있음

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

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

  - 매개변수를 지정하면 다음과 같이 순서에 상관없이 사용할 수 있음

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

함수

- 여러 개의 입력값을 받는 함수
  - 입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?

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

  - '*매개변수'를 이용하면 여러 개의 입력값을 받을 수 있음

함수

- 가변 매개변수(*args)

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

  - *args처럼 매개변수 이름 앞에 *을 붙이면 입력값을 전부 모아 튜플로 만들어 줌
  - 예를들어, add_many(1, 2, 3)면 args는 (1, 2, 3)이 되고 add_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)면 args는 (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)됨

함수

- 가변 매개변수(*args)

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

  - 여기에서 *args는 임의로 정한 변수 이름이며, *pey, *python처럼 아무 이름을 사용할 수 있음
  - 관례적으로 *args를 사용

함수

- 가변 매개변수(*args)
  - 여러 개의 입력을 처리할 때 함수의 매개변수로 *args 외에 추가적인 매개변수를 사용할 수 있음

```
>>> def add_mul(choice, *args):
...     if choice == "add":   # 매개변수 choice에 "add"를 입력받았을 때
...         result = 0
...         for i in args:
...             result = result + i
...     elif choice == "mul":   # 매개변수 choice에 "mul"을 입력받았을 때
...         result = 1
...         for i in args:
...             result = result * i
...     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
```

함수

- 키워드 매개변수 (**kwargs)

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

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

  - 매개변수 이름 앞에 **을 붙이면 매개변수 kwargs는 딕셔너리가 되고 모든 Key=Value 형태의 입력값이 그 딕셔너리에 저장
  - 예를들어, 함수의 입력값으로 a=1이 사용되면 kwargs는 {'a': 1}이라는 딕셔너리가 되고, 입력값으로 name='foo', age=3이 사용되면 kwargs는 {'age': 3, 'name': 'foo'}라는 딕셔너리가 됨
  - 관례적으로, **kwargs를 사용

함수

- 함수의 리턴값은 언제나 하나!

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

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

  - 함수의 출력값은 a+b와 a*b인데, 출력값을 받아들이는 변수는 result 하나만 쓰였지만 오류가 발생하지 않음
  - add_and_mul 함수의 리턴값 a+b와 a*b는 튜플값 하나인 (a+b, a*b)로 출력함

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

  - 만약 튜플 값을 분류하고 싶다면 아래와 같이 호출함

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

함수

- 함수의 리턴값은 언제나 하나!

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

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

  - add_and_mul(2, 3)는 5만 출력하며, 두 번째 return 문인 return a * b는 실행되지 않음
  - 즉, 함수는 return 문을 만나는 순간, 출력값을 돌려 준 다음 함수를 종료함

함수

- 매개변수 초기값(디폴트값) 미리 설정하기
  - 파이썬 함수는 매개변수에 초기값을 설정할 수 있음

```
# default1.py
def say_myself(name, age, man=True):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % age)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

함수

- 매개변수 초기값(디폴트값) 미리 설정하기

```
say_myself("박응용", 27)
```

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

- 입력값으로 ("박응용", 27)처럼 2개를 주면 name에는 "박응용", age에는 27이 대입됨.
- 그리고 man이라는 변수에는 입력값을 주지 않았지만, man은 초기값 True를 할당함

함수

- 매개변수 초기값(디폴트값) 미리 설정하기

```
say_myself("박응선", 27, False)
```

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

함수

- 매개변수 초기값(디폴트값) 미리 설정하기

```
# default2.py
def say_myself(name, man=True, age):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % age)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

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

non-default argument(age)는 default argument(man) 뒤에 온다.

- 초기화하고 싶은 매개변수는 항상 뒤쪽에 놓아야 함

함수

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

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

vartest(a)
print(a)
```

  - 결과값은 '1'이 나옴
  - 함수 안에서 사용하는 매개변수는 함수 안에서만 사용하는 '함수만의 변수'이기 때문임

함수

- 함수 안에서 함수 밖의 변수를 변경하는 방법
  - global 명령어

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

vartest()
print(a)
```

  - 'global a' 문장은 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 뜻이며 변수 a를 전역 변수라 부름

함수

- lambda 예약어
  - lambda는 함수를 생성할 때 사용하는 예약어로, def와 동일한 역할을 함
  - 함수를 한 줄로 간결하게 만들 때 사용

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

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

  - lambda로 만든 함수는 return 명령어가 없어도 표현식의 결과값을 리턴함

사용자 입출력

사용자 입력

- input() 함수

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

  - 프롬프트를 띄워 사용자 입력을 받고 싶을 경우, input()의 괄호 안에 안내 문구를 입력하면 됨.

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

파일 읽고 쓰기

파일 생성하기

- open() 함수
  - open 함수는 '파일 이름'과 '파일 열기 모드'를 입력값으로 받고 결과값으로 파일 객체를 리턴

```
파일_객체 = open(파일_이름, 파일_열기_모드)
```

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

파일열기모드	설명
r	읽기 모드: 파일을 읽기만 할 때 사용한다.
w	쓰기 모드: 파일에 내용을 쓸 때 사용한다.
a	추가 모드: 파일의 마지막에 새로운 내용을 추가할 때 사용한다.

파일을 쓰기 모드로 열어 내용 쓰기

```
for i in range(1, 11):
    data = "%d번째 줄입니다.\n" % i
    print(data)
```

```
# write_data.py
f = open("C:/doit/새파일.txt", 'w')
for i in range(1, 11):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

파일을 쓰기 모드로 열어 내용 쓰기

- readline() 함수
  - 파일의 첫 번째 줄을 읽어 출력

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

  - 한 줄씩 읽어 출력할 때 줄 끝에 '₩n' 문자가 있으므로 빈 줄도 같이 출력
  - '₩n' 문자를 제거하고 싶을 경우, strip() 함수를 사용
    - line = line.strip()

파일을 쓰기 모드로 열어 내용 쓰기

- readline() 함수
  - 만약 모든 줄을 읽어 화면에 출력하고 싶다면?

```
# 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: 무한 루프 안에서 f.readline()을 사용해 파일을 계속 한 줄씩 읽어 들임
  - 만약 더 이상 읽을 줄이 없으면 break를 수행
    - readline()은 더 이상 읽을 줄이 없을 경우, 빈 문자열('')을 리턴

파일을 쓰기 모드로 열어 내용 쓰기

- readlines() 함수
  - 파일의 모든 줄을 읽어서 각각의 줄을 요소로 가지는 리스트를 리턴
  - (예) ["1번째 줄입니다.₩n", "2번째 줄입니다.₩n", ..., "10번째 줄입니다.₩n"]

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

파일을 쓰기 모드로 열어 내용 쓰기

- read() 함수
  - 파일의 내용 전체를 문자열로 리턴

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

파일에 새로운 내용 추가하기

- 원래 있던 값을 유지하면서 단지 새로운 값만 추가해야 할 경우 파일을 추가 모드('a')로 열면 됨

```
# add_data.py
f = open("C:/doit/새파일.txt",'a')
for i in range(11, 20):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

with 문과 함께 사용하기

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

```
# file_with.py
with open("foo.txt", "w") as f:
    f.write("Life is too short, you need python")
```

- with 문을 사용하면 with 블록을 벗어나는 순간, 열린 파일 객체 f가 자동으로 닫힘(close)

프로그램의 입출력

프로그램의 입출력

- sys 모듈

```
# sys1.py
import sys

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

- sys.argv는 프로그램 실행 시 전달된 인수를 의미

프로그램의 입출력

- sys 모듈

```
# sys1.py
import sys

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

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

Q&A