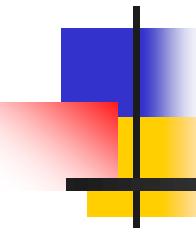
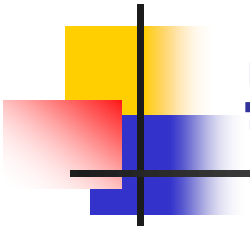


04장 프로그램의 입력과 출력은 어떻게 해야 할까?

함수, 입력과 출력, 파일 처리 방법



목차

- 함수
- 사용자 입력과 출력
- 파일 읽고 쓰기

04-1 함수

입력값을 가지고 어떤 일을 수행한 다음에 그 결과물을 내어놓는 것, 이것이 함수가 하는 일

함수를 사용하는 이유

1. 반복되는 부분이 있을 경우 "반복적으로 사용되는 가치 있는 부분"을 한 뭉치로 묶어서 "어떤 입력값을 주었을 때 어떤 결괏값을 돌려준다"라는 식의 함수로 작성
2. 자신이 만든 프로그램을 함수화하면 프로그램 흐름을 일목요연하게 볼 수 있기 때문

파이썬 함수의 구조

```
def 함수명(입력 인수):
    <수행할 문장1>
    <수행할 문장2>
    ...
```

04-1 함수

입력값과 결괏값이 있는 함수의 사용법

결괏값을 받을 변수 = 함수이름(입력인수1, 입력인수2, ...)

일반적인 함수-입력값이 있고 결괏값이 있는 함수

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

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

매개변수는 함수에 입력으로 전달된 값을 받는 변수를 의미하고 인수는 함수를 호출할 때 전달하는 입력값을 의미한다.

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

print(add(3, 4))  # 3, 4는 인수
```

함수는 들어온 입력값을 받아 어떤 처리를 하여 적절한 결괏값을 돌려준다.

입력값 ---> 함수 ----> 결괏값

함수의 형태는 입력값과 결괏값의 존재 유무에 따라 4가지 유형으로 나뉜다

04-1 함수

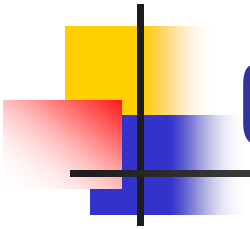
입력값이 없는 함수

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

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

입력값이 없고 결괏값만 있는 함수 사용법

결괏값을 받을 변수 = 함수이름()



04-1 함수

결과값이 없는 함수

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

```
>>> sum(3, 4)
3, 4의 합은 7입니다.
```

결괏값이 없는 함수 사용법

함수이름(입력인수1, 입력인수2, ...)

결괏값은 오직 return 명령어로만 돌려받을 수 있다.

04-1 함수

입력값도 결과값도 없는 함수

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

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

입력값도 결괏값도 없는 함수 사용법

함수이름()

04-1 함수

매개변수 지정하여 호출하기

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

>>> result = add(a=3, b=7)  # a에 3, b에 7을 전달
>>> print(result)
10

매개변수를 지정하면 순서에 상관없이 사용할 수 있다는 장점이 있다.

>>> result = add(b=5, a=3)  # b에 5, a에 3을 전달
>>> print(result)
8
```

입력값이 여러 개일 때 그 입력값을 모두 더해 주는 함수 만들기
- 몇 개가 입력될지 모를 때는 어떻게 해야 할까?

def 함수이름(*매개변수):
    <수행할 문장>
    ...

04-1 함수

여러 개의 입력값

```
>>> def sum_many(*args):
...     sum = 0
...     for i in args:
...         sum = sum + i
...     return sum
...
>>> print(sum_many(1,2,3))
>>> print(sum_many(1,2,3,4,5))
```

매개변수 이름 앞에 *을 붙이면 입력값을 전부 모아서 튜플로 만들어 줌

```
>>> def add_mul(choice, *args):
...     if choice == "add":
...         result = 0
...         for i in args:
...             result = result + i
...     elif choice == "mul":
...         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)
>>> print(result)
24
```

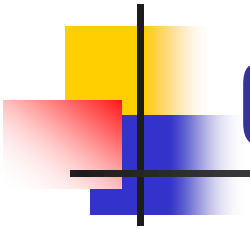
- 키워드 파라미터 kwargs

키워드 파라미터를 사용할 때는 매개변수 앞에 별 두 개(**)를 붙인다.

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

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

입력값 a=1 또는 name='foo', age=3이 모두 딕셔너리로 만들어져서 출력된다
즉 **kwargs처럼 매개변수 이름 앞에 **을 붙이면 매개변수 kwargs는 딕셔너리가 되고 모든 key=value 형태의 결괏값이 그 딕셔너리에 저장된다.



04-1 함수

함수의 결과값은 언제나 하나이다

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

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

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

함수의 결괏값은 2개가 아니라 언제나 1개

add_and_mul 함수의 결괏값 a+b와 a*b는 튜플값 하나인 (a+b, a*b)로 돌려준다.

이 하나의 튜플 값을 2개의 결괏값처럼 받고 싶다면

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

이렇게 호출하면 result1, result2 = (7, 12)가 되어 result1은 7이 되고 result2는 12가 된다.

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

return문을 2번 사용하면 두 번째 return문인 return a*b는 실행되지 않는다

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

※ add_and_mul(2, 3)의 결괏값은 5 하나뿐이다. 두 번째 return문인 return a*b는 실행되지 않았다는 뜻이다.

함수는 return문을 만나는 순간 결괏값을 돌려준 다음 함수를 빠져나가게 된다.

[return의 또 다른 쓰임새]

특별한 상황일 때 함수를 빠져나가고 싶다면 return을 단독으로 써서 함수를 즉시 빠져나갈 수 있다.

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

'별명'을 입력으로 전달받아 출력하는 함수.
만약에 입력값으로 '바보'라는 값이 들어오면 문자열을 출력하지 않고 함수를 즉시 빠져나간다.

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

04-1 함수

매개 변수에 초깃값 미리 설정하기

man=True처럼 매개변수에 미리 값을 넣어 준 것

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

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

04-1 함수

초기화시키고 싶은 매개변수를 항상 뒤쪽에 놓아야 함

함수 매개 변수에 초깃값을 설정할 때 주의할 사항

함수를 실행할 때 오류가 발생

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

04-1 함수

함수 안에서 선언된 변수의 효력 범위

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

vartest(a)
print(a)   # 1
```

실행해 보면 결괏값은 1이 나온다

함수 안에서 새로 만든 매개변수는 함수 안에서만 사용하는 "함수만의 변수"이다.
즉 def vartest(a)에서 입력값을 전달받는 매개변수 a는 함수 안에서만 사용하는 변수이지 함수 밖의 변수 a가 아니라는 뜻이다.

함수 안에서 사용하는 매개변수는 함수 밖의 변수 이름과는 전혀 상관이 없다는 뜻이다.
함수 안에서 선언한 매개변수는 함수 안에서만 사용될 뿐 함수 밖에서는 사용되지 않는다.

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

vartest(3)
print(a)   # 오류발생
```

04-1 함수

함수 안에서 함수 밖의 변수를 변경하는 방법 1

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

a = vartest(a)
print(a)   #2
```

return 사용하기

여기에서도 vartest 함수 안의 a 매개변수는 함수 밖의 a와는 다른 것이다.

04-1 함수

함수 안에서 함수 밖의 변수를 변경하는 방법 2

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

vartest()
print(a)  #2
```

global 명령어 사용하기

vartest 함수 안의 **global a 문장은 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 뜻**이다.

하지만 프로그래밍을 할 때 global 명령어는 사용하지 않는 것이 좋다. 왜냐하면 함수는 독립적으로 존재하는 것이 좋기 때문이다. 외부 변수에 종속적인 함수는 그다지 좋은 함수가 아니다.

가급적 global 명령어를 사용하는 이 방법은 피하고 첫 번째 방법을 사용하기를 권장.

lambda

- **lambda**는 함수를 생성할 때 사용하는 예약어로 def와 동일한 역할을 한다.
- 보통 함수를 한줄로 간결하게 만들 때 사용.
- def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰인다.

- lambda 매개변수1, 매개변수2, ... : 매개변수를 이용한 표현식

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

add는 두 개의 인수를 받아 서로 더한 값을 돌려주는 lambda 함수.
def를 사용한 다음 함수와 하는 일이 완전히 동일

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

※ lambda 예약어로 만든 함수는 return 명령어가 없어도 결괏값을 돌려준다.

04-2 사용자 입력과 출력

input의 사용

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

input은 입력되는 모든 것을 문자열로 취급한다.

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

04-2 사용자 입력과 출력

프롬프트를 띄워서 사용자 입력 받기

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

사용자에게 입력받을 때 "숫자를 입력하세요"라든지 "이름을 입력하세요"라는 안내 문구 또는 질문이 나오도록 하고 싶을 때는 input()의 괄호 안에 질문을 입력하여 프롬프트를 띄워주면 된다.

input("질문 내용")

- 큰따옴표(")로 둘러싸인 문자열은 + 연산과 동일하다
- 콤마(,)를 사용하면 문자열 사이에 띄어쓰기를 할 수 있다.
- 한 줄에 결괏값을 계속 이어서 출력하려면 매개변수 end를 사용해 끝 문자를 지정해야 한다.

04-2 사용자 입력과 출력

print 문

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

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

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

04-3 파일 읽고 쓰기

파일 생성하기

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

프로그램을 실행한 디렉터리에 새로운 파일이 하나 생성됨
파일을 생성하기 위해 파이썬 내장 함수 open 사용.
open 함수는 "파일 이름"과 "파일 열기 모드"를 입력값으로 받고 결괏값으로 파일 객체를 돌려준다.

파일 객체 = open(파일 이름, 파일 열기 모드)

파일열기모드	설명
r	읽기모드 - 파일을 읽기만 할 때 사용
w	쓰기모드 - 파일에 내용을 쓸 때 사용
a	추가모드 - 파일의 마지막에 새로운 내용을 추가시킬 때 사용

04-3 파일 읽고 쓰기

- 파일을 쓰기 모드로 열면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고, 해당 파일이 존재하지 않으면 새로운 파일이 생성된다.
- 위 예에서는 디렉터리에 파일이 없는 상태에서 새파일.txt를 쓰기 모드인 'w'로 열었기 때문에 새파일.txt라는 이름의 새로운 파일이 현재 디렉터리에 생성됨
- 새파일.txt 파일을 C:/doit 디렉터리에 생성하고 싶다면

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

- f.close()는 열려 있는 파일 객체를 닫아 주는 역할을 함.
- 이 문장은 생략해도 된다. 프로그램을 종료할 때 파이썬 프로그램이 열려 있는 파일의 객체를 자동으로 닫아주기 때문
- 하지만 close()를 사용해서 열려 있는 파일을 직접 닫아 주는 것이 좋다. 쓰기모드로 열었던 파일을 닫지 않고 다시 사용하려고 하면 오류가 발생하기 때문

04-3 파일 읽고 쓰기

파일을 쓰기 모드로 열어 출력값 적기

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

f.close()

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

모니터 화면 대신 파일에 결괏값을 적는 방법

모니터 화면에 출력하는 방법

04-3 파일 읽고 쓰기

프로그램의 외부에 저장된 파일을 읽는 여러 가지 방법

readline() 함수

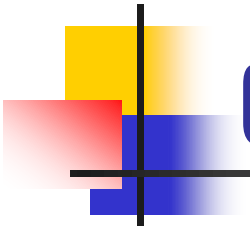
```
f = open("C:/Python/새파일.txt", 'r')
line = f.readline()
print(line)
f.close()
```

파일을 읽기 모드로 연 후 readline()을 사용해서 파일의 첫 번째 줄을 읽어 출력

```
f = open("C:/Python/새파일.txt", 'r')
while True:
    line = f.readline()
    if not line: break
    print(line)
f.close()
```

모든 줄을 읽어서 화면에 출력

무한 루프 안에서 f.readline()을 사용해 파일을 계속해서 한 줄씩 읽어 들인다. 더 이상 읽을 줄이 없으면 break를 수행 (readline()은 더 이상 읽을 줄이 없을 경우 None을 출력).



04-3 파일 읽고 쓰기

readlines() 함수 이용하기

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

readlines 함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트로 돌려준다.

04-3 파일 읽고 쓰기

read() 함수 이용하기

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

f.read()는 파일의 내용 전체를 문자열로 돌려준다

04-3 파일 읽고 쓰기

파일에 새로운 내용 추가하기

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

쓰기 모드('w')로 파일을 열 때 이미 존재하는 파일을 열면 그 파일의 내용이 모두 사라지게 된다.

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

새파일.txt 파일을 추가 모드('a')로 열고 write를 사용해서 결괏값을 기존 파일에 추가해 적는다.
추가 모드로 파일을 열었기 때문에 새파일.txt 파일이 원래 가지고 있던 내용 바로 다음부터 결괏값을 적기 시작한다.

04-3 파일 읽고 쓰기

with문과 함께 사용하기

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

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

파일을 열면 항상 close해 주는 것이 좋다.

with문
 - 파일을 열고 닫는 것을 자동으로 처리하는 역할을 한다
with문을 사용하면 with 블록을 벗어나는 순간 열린 파일 객체 f가 자동으로 close됨

- Q1
  주어진 자연수가 홀수인지 짝수인지 판별해 주는 함수(is_odd) 작성

- Q2
  입력으로 들어오는 모든 수의 평균 값을 계산해 주는 함수 작성.
  (단 입력으로 들어오는 수의 개수는 정해져 있지 않다.)
  ※ 평균 값을 구할 때 len 함수를 사용

- Q3
  다음은 두 개의 숫자를 입력 받아 더하여 돌려주는 프로그램이다.

```
input1 = input("첫 번째 숫자를 입력하세요:")
input2 = input("두 번째 숫자를 입력하세요:")
total = input1 + input2
print("두 수의 합은 %s 입니다" % total)

이 프로그램을 수행해 보자.

첫번째 숫자를 입력하세요:3
두번째 숫자를 입력하세요:6
두 수의 합은 36 입니다
3과 6을 입력했을 때 9가 아닌 36이라는 결괏값을 돌려주었다. 이 프로그램의
오류를 수정해 보자. (※ int 함수를 사용)
```

- Q4. 다음 중 출력 결과가 다른 것 한 개를 골라 보자.

```
print("you" "need" "python")
print("you"+"need"+"python")
print("you", "need", "python")
print("".join(["you", "need", "python"]))
```

- Q5. 다음은 "test.txt"라는 파일에 "Life is too short" 문자열을 저장한 후 다시 그 파일을 읽어서 출력하는 프로그램이다.

```
f1 = open("test.txt", 'w')
f1.write("Life is too short")
f2 = open("test.txt", 'r')
print(f2.read())
```

이 프로그램은 우리가 예상한 "Life is too short"라는 문장을 출력하지 않는다. 우리가 예상한 값을 출력할 수 있도록 프로그램을 수정해 보자.

- Q6.사용자의 입력을 파일(test.txt)에 저장하는 프로그램을 작성해 보자. (단 프로그램을 다시 실행하더라도 기존에 작성한 내용을 유지하고 새로 입력한 내용을 추가해야 한다.)

- Q7. 다음과 같은 내용을 지닌 파일 test.txt가 있다. 이 파일의 내용 중 "java"라는 문자열을 "python"으로 바꾸어서 저장해 보자. (※ replace 함수를 사용)

```
Life is too short
you need java
```