



염 희 균

목차

- 함수의 기본
- 함수에 매개변수 만들기
- 가변 매개변수
- 기본 매개변수
- 키워드 매개변수
- 리턴
- 기본적인 함수의 활용
- 파일처리
- 키워드로 정리하는 핵심 포인트
- 확인문제

시작하기 전에

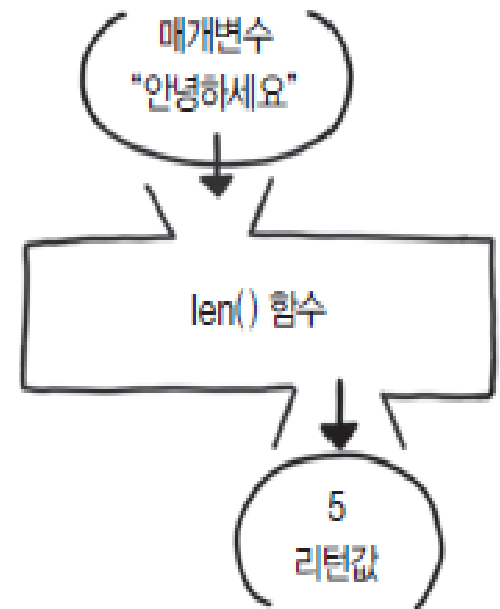
[핵심 키워드] : 호출, 매개변수, 리턴값, 가변 매개변수, 기본 매개변수

[핵심 포인트]

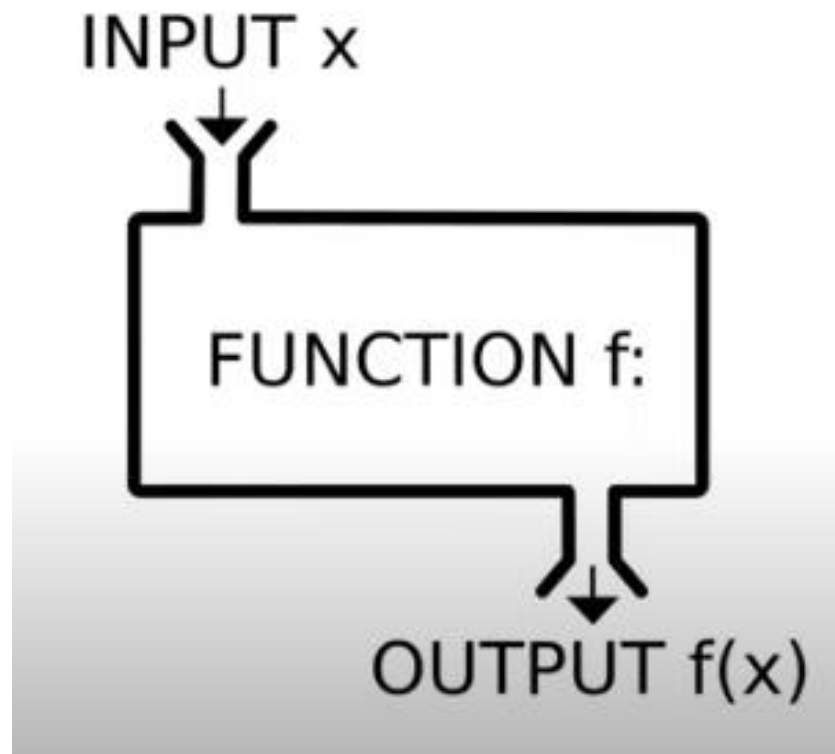
함수들을 어떻게 만들고 활용하는지 살펴본다.

시작하기 전에

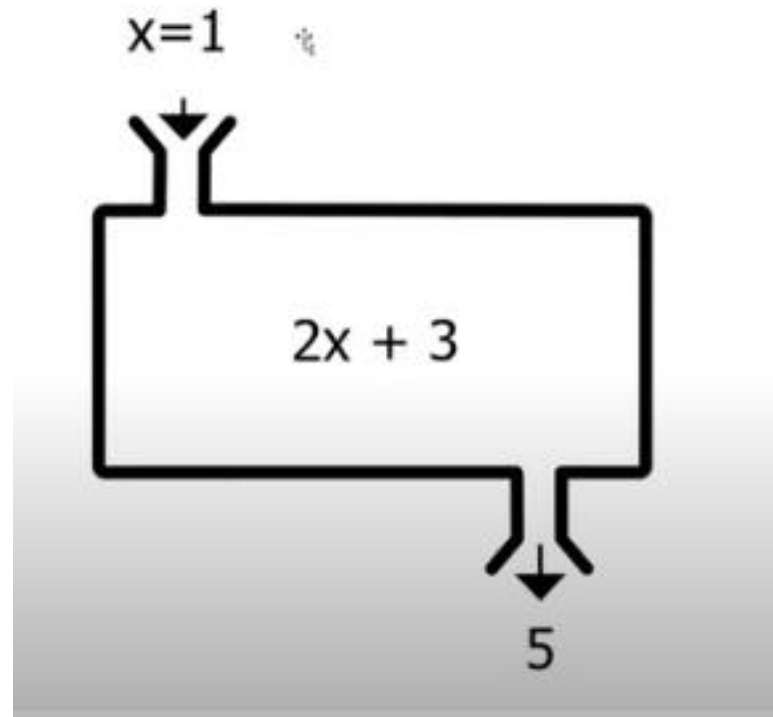
- 함수를 호출
 - 함수 사용
- 매개변수
 - 함수 호출 시 괄호 내부에 넣는 여러 가지 자료
- 리턴값
 - 함수를 호출하여 최종적으로 나오는 결과



함수



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



입력, 출력 없을 수 있음 (alert())



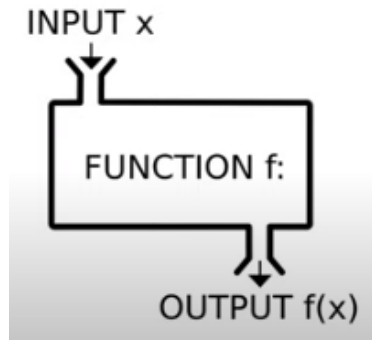
FUNCTION f:

함수의 기본

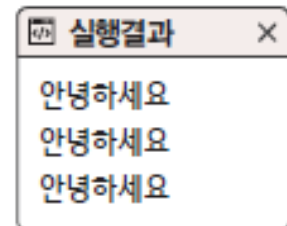
- 함수 = 코드의 집합

파이썬 함수의 구조

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



```
01 def print_3_times():  
02     print("안녕하세요")  
03     print("안녕하세요")  
04     print("안녕하세요")  
05  
06 print_3_times()
```

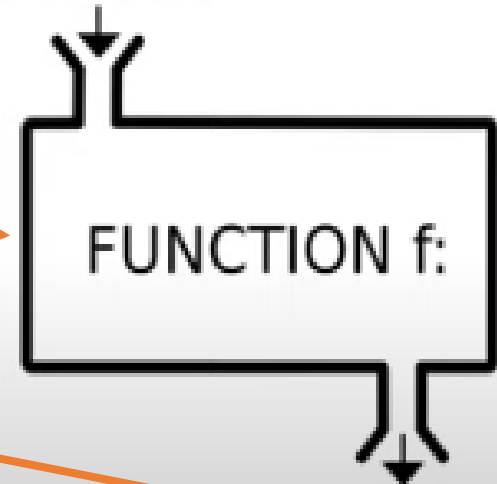


함수의 기본

파이썬 함수의 구조

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

INPUT x



OUTPUT $f(x)$

함수에 매개변수 만들기

- 매개변수

```
def 함수 이름(매개변수, 매개변수, ...):  
    문장
```

print

 print

```
def print(value, ..., sep=' ' ×  
, end='\n', file=sys.stdout,  
t, flush=False)
```

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream);
defaults to the current sys.stdout.

sep: string inserted between values,
default a space.

end: string appended after the last

함수에 매개변수 만들기

```
01 def print_n_times(value, n):  
02     for i in range(n):  
03         print(value)  
04  
05 print_n_times("안녕하세요", 5)
```

실행결과

안녕하세요
안녕하세요
안녕하세요
안녕하세요
안녕하세요

일반적인 함수

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

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

7

함수에 매개변수 만들기

- 매개변수와 관련된 TypeError
 - 매개변수를 넣지 않은 경우

```
def print_n_times(value, n): → 매개변수를 2개 지정했는데
    for i in range(n):
        print(value)

# 함수를 호출합니다.
print_n_times("안녕하세요") → 하나만 넣었습니다.
```

오류

```
Traceback (most recent call last):
  File "test5_01.py", line 6, in <module>
    print_n_times("안녕하세요")
TypeError: print_n_times() missing 1 required positional argument: 'n'
```

함수에 매개변수 만들기

- 매개변수를 더 많이 넣은 경우

```
def print_n_times(value, n): → 매개변수를 2개 지정했는데
    for i in range(n):
        print(value)

# 함수를 호출합니다.
print_n_times("안녕하세요", 10, 20) → 3개를 넘었습니다.
```

오류

```
Traceback (most recent call last):
  File "test5_02.py", line 6, in <module>
    print_n_times("안녕하세요", 10, 20)
TypeError: print_n_times() takes 2 positional arguments but 3 were given
```

가변 매개변수

- *가변 매개변수

- 매개변수를 원하는 만큼 받을 수 있는 함수

```
def 함수 이름(매개변수, 매개변수, ..., *가변 매개변수):  
    문장
```

- 제약

- 가변 매개변수 뒤에는 일반 매개변수 올 수 없음
- 가변 매개변수는 하나만 사용할 수 있음

가변 매개변수

- 예시 - 가변 매개변수 함수

```
01 def print_n_times(n, *values):  
02     # n번 반복합니다.  
03     for i in range(n):  
04         # values는 리스트처럼 활용합니다.  
05         for value in values:  
06             print(value)  
07         # 단순한 줄바꿈  
08         print()  
09  
10 # 함수를 호출합니다.  
11 print_n_times(3, "안녕하세요", "즐거운", "파이썬 프로그래밍")
```

실행결과

안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍

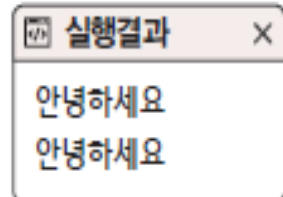
기본 매개변수

- 기본 매개변수

- 매개변수 값 입력하지 않았을 경우 매개변수에 들어가는 기본값

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
01 def print_n_times(value, n=2):  
02     # n번 반복합니다.  
03     for i in range(n):  
04         print(value)  
05  
06 # 함수를 호출합니다.  
07 print_n_times("안녕하세요")
```



실행결과

안녕하세요
안녕하세요

키워드 매개변수

- 키워드 매개변수
 - 매개변수 이름을 지정해서 입력하는 매개변수

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
# while 반복문을 사용합니다.
```

```
while True:
```

```
    # "."을 출력합니다.
```


```
    # 기본적으로 end가 "\n"이라 줄바꿈이 일어나는데,
```

```
    # 빈 문자열 ""로 바꿔서 줄바꿈이 일어나지 않게 합니다.
```

```
    print(".", end="") → 키워드 매개변수입니다.
```

키워드 매개변수

```
01 def print_n_times(*values, n=2):  
02     # n번 반복합니다.  
03     for i in range(n):  
04         # values는 리스트처럼 활용합니다.  
05         for value in values:  
06             print(value)  
07         # 단순한 줄바꿈  
08         print()  
09  
10 # 함수를 호출합니다.  
11 print_n_times("안녕하세요", "즐거운", "파이썬 프로그래밍", n=3)
```

 실행결과 x

안녕하세요
즐거운
파이썬 프로그래밍

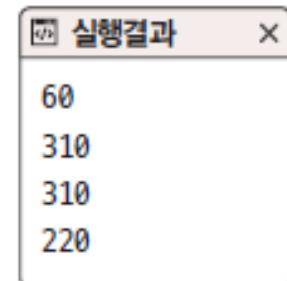
안녕하세요
즐거운
파이썬 프로그래밍

안녕하세요
즐거운
파이썬 프로그래밍

키워드 매개변수

- 기본 매개변수 중에서 필요한 값만 입력하기
 - 예시 - 여러 함수 호출 형태

```
01 def test(a, b=10, c=100):  
02     print(a + b + c)  
03  
04 # 1) 기본 형태  
05 test(10, 20, 30)  
06 # 2) 키워드 매개변수로 모든 매개변수를 지정한 형태  
07 test(a=10, b=100, c=200)  
08 # 3) 키워드 매개변수로 모든 매개변수를 마구잡이로 지정한 형태  
09 test(c=10, a=100, b=200)  
10 # 4) 키워드 매개변수로 일부 매개변수만 지정한 형태  
11 test(10, c=200)
```



실행결과

60
310
310
220

키워드 매개변수

- 첫 번째 매개변수 a : 일반 매개변수이므로 해당 위치에 반드시 입력해야 함
- 8행 3번 : 키워드 지정하여 매개변수 입력하는 경우 매개변수 순서를 원하는 대로 입력할 수 있음
- 10행 4번 : b를 생략한 형태. 키워드 매개변수 사용하여 필요한 매개변수에만 값을 전달

리턴

- 리턴값 (return value)
 - 함수의 결과

```
# input() 함수의 리턴값을 변수에 저장합니다.  
value = input("> ")  
  
# 출력합니다.  
print(value)
```

리턴

- 자료 없이 리턴하기
 - return 키워드 : 함수를 실행했던 위치로 돌아가게 함

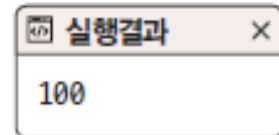
```
01  # 함수를 정의합니다.  
02  def return_test():  
03      print("A 위치입니다.")  
04      return                # 리턴합니다.  
05      print("B 위치입니다.")  
06  
07  # 함수를 호출합니다.  
08  return_test()
```

실행결과 ×
A 위치입니다.

리턴

- 자료와 함께 리턴하기
 - 리턴 뒤에 자료 입력하면 자료 가지고 돌아감

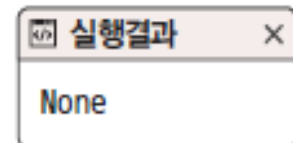
```
01  # 함수를 정의합니다.  
02  def return_test():  
03      return 100  
04  
05  # 함수를 호출합니다.  
06  value = return_test()  
07  print(value)
```



리턴

- 아무것도 리턴하지 않기
 - None : '없다'라는 의미

```
01  # 함수를 정의합니다.  
02  def return_test():  
03      return  
04  
05  # 함수를 호출합니다.  
06  value = return_test()  
07  print(value)
```



기본적인 함수의 활용

- 일반적인 형태
 - 값을 만들어 리턴

```
def 함수(매개변수):
```

```
    변수 = 초깃값
```

```
    # 여러 가지 처리
```

```
    # 여러 가지 처리
```

```
    # 여러 가지 처리
```

```
    return 변수
```

기본적인 함수의 활용

- 예시 - 범위 내부의 정수를 모두 더하는 함수

```
01  # 함수를 선언합니다.
02  def sum_all(start, end):
03      # 변수를 선언합니다.
04      output = 0
05      # 반복문을 돌려 숫자를 더합니다.
06      for i in range(start, end + 1):
07          output += i
08      # 리턴합니다.
09      return output
10
11  # 함수를 호출합니다.
12  print("0 to 100:", sum_all(0, 100))
13  print("0 to 1000:", sum_all(0, 1000))
14  print("50 to 100:", sum_all(50, 100))
15  print("500 to 1000:", sum_all(500, 1000))
```

실행결과

0 to 100:	5050
0 to 1000:	500500
50 to 100:	3825
500 to 1000:	375750

기본적인 함수의 활용

- 예시 - 기본 매개변수와 키워드 매개 변수를 활용해 범위의 정수를 더하는 함수

```
01  # 함수를 선언합니다.
02  def sum_all(start=0, end=100, step=1):
03      # 변수를 선언합니다.
04      output = 0
05      # 반복문을 돌려 숫자를 더합니다.
06      for i in range(start, end + 1, step):
07          output += i
08      # 리턴합니다.
09      return output
10
11  # 함수를 호출합니다.
12  print("A.", sum_all(0, 100, 10))
13  print("B.", sum_all(end=100))
14  print("C.", sum_all(end=100, step=2))
```

실행결과	
A.	550
B.	5050
C.	2550

키워드로 정리하는 핵심 포인트

- **호출** : 함수를 실행하는 행위
- **매개변수** : 함수의 괄호 내부에 넣는 것
- **리턴값** : 함수의 최종적인 결과
- **가변 매개변수 함수** : 매개변수를 원하는 만큼 받을 수 있는 함수
- **기본 매개 변수** : 매개변수에 아무 것도 넣지 않아도 들어가는 값

기본적인 함수의 활용

- 다음과 같이 방정식을 파이썬 함수로 만들어 보세요.

예: $f(x)=x$

```
def f(x):  
    return x  
print(f(10))
```

① $f(x)=2x+1$

```
def f(x):  
    return   
print(f(10))
```

② $f(x)=x^2+2x+1$

```
def f(x):  
    return   
print(f(10))
```

기본적인 함수의 활용

- 다음 빈칸을 채워 매개변수로 전달된 값들을 모두 곱해서 리턴하는 가변 매개변수 함수를 만들어 보세요.

```
def mul(*values):
```

```
# 함수를 호출합니다.
```

```
print(mul(5, 7, 9, 10))
```

실행결과

3150

기본적인 함수의 활용

- 다음 중 오류가 발생하는 코드를 고르세요.

①

```
def function(*values, valueA, valueB):  
    pass  
function(1, 2, 3, 4, 5)
```

②

```
def function(*values, valueA=10, valueB=20):  
    pass  
function(1, 2, 3, 4, 5)
```

③

```
def function(valueA, valueB, *values):  
    pass  
function(1, 2, 3, 4, 5)
```

④

```
def function(valueA=10, valueB=20, *values):  
    pass  
function(1, 2, 3, 4, 5)
```


시작하기 전에

- 튜플 (tuple)
 - 함수와 함께 많이 사용되는 리스트와 비슷한 자료형으로, 한번 결정된 요소를 바꿀 수 없다는 점이 리스트와 다름
- 람다 (lambda)
 - 매개변수로 함수를 전달하기 위해 함수 구문을 작성하는 것이 번거롭고 코드 낭비라 생각될 때 함수를 간단하고 쉽게 선언하는 방법

튜플

- 튜플 (tuple)
 - 리스트와 유사한 자료형
 - 한번 결정된 요소는 바꿀 수 없음

(데이터, 데이터, 데이터, ...)

```
>>> tuple_test = (10, 20, 30)
```

```
>>> tuple_test[0]
```

```
10
```

```
>>> tuple_test[1]
```

```
20
```

```
>>> tuple_test[2]
```

```
30
```

튜플

```
>>> tuple_test[0] = 1
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#1>", line 1, in <module>
```

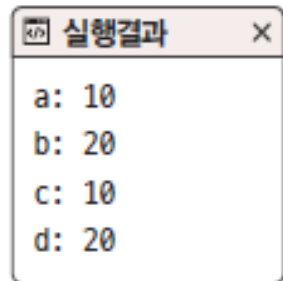
```
    tuple_test[0] = 1
```

```
TypeError: 'tuple' object does not support item assignment
```

튜플

- 괄호 없는 튜플
 - 예시

```
01  # 리스트와 튜플의 특이한 사용
02  [a, b] = [10, 20]
03  (c, d) = (10, 20)
04
05  # 출력합니다.
06  print("a:", a)
07  print("b:", b)
08  print("c:", c)
09  print("d:", d)
```



실행결과

a: 10
b: 20
c: 10
d: 20

튜플

- 괄호를 생략

```
01  # 괄호가 없는 튜플
02  tuple_test = 10, 20, 30, 40
03  print("# 괄호가 없는 튜플의 값과 자료형 출력")
04  print("tuple_test:", tuple_test)
05  print("type(tuple_test:)", type(tuple_test))
06  print()
07
08  # 괄호가 없는 튜플 활용
09  a, b, c = 10, 20, 30
10  print("# 괄호가 없는 튜플을 활용한 할당")
11  print("a:", a)
12  print("b:", b)
13  print("c:", c)
```

→ 튜플을 입력한 것입니다.

실행결과

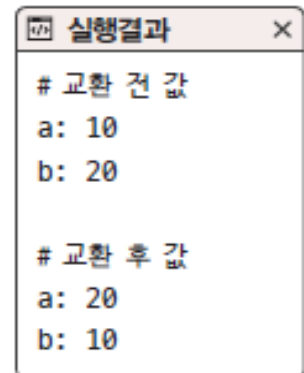
```
# 괄호가 없는 튜플의 값과 자료형 출력
tuple_test: (10, 20, 30, 40)
type(tuple_test:) <class 'tuple'>

# 괄호가 없는 튜플을 활용한 할당
a: 10
b: 20
c: 30
```

튜플

- 활용 예시 - 변수의 값을 교환하는 튜플

```
01  a, b = 10, 20
02
03  print("# 교환 전 값")
04  print("a:", a)
05  print("b:", b)
06  print()
07
08  # 값을 교환합니다.
09  a, b = b, a
10
11  print("# 교환 후 값")
12  print("a:", a)
13  print("b:", b)
14  print()
```



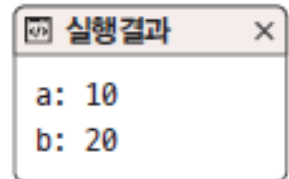
실행결과

```
# 교환 전 값
a: 10
b: 20

# 교환 후 값
a: 20
b: 10
```

- 튜플과 함수
 - 예시 - 여러 개의 값 리턴하기

```
01  # 함수를 선언합니다.
02  def test():
03      return (10, 20)
04
05  # 여러 개의 값을 리턴받습니다.
06  a, b = test()
07
08  # 출력합니다.
09  print("a:", a)
10  print("b:", b)
```



실행결과

a: 10
b: 20

람다

- 람다 (lambda)
 - 기능(함수)을 매개변수로 전달하는 코드를 더 효율적으로 작성
- 함수의 매개변수로 함수 전달하기

```
01 # 매개변수로 받은 함수를 10번 호출하는 함수
02 def call_10_times(func):
03     for i in range(10):
04         func()
05
06 # 간단한 출력하는 함수
07 def print_hello():
08     print("안녕하세요")
09
10 # 조합하기
11 call_10_times(print_hello)
```

↓
매개변수로 함수를 전달합니다.

실행결과

안녕하세요
안녕하세요
안녕하세요
안녕하세요
안녕하세요
안녕하세요
안녕하세요
안녕하세요
안녕하세요
안녕하세요

람다

- 람다란 '간단한 함수를 쉽게 선언하는 방법'
- def와 동일한 역할을 하는 예약어 - lambda
 - return 명령어가 없어도 결과값을 돌려준다.

lambda 매개변수 : 리턴값

```
1
2 # 람다 함수를 선언합니다.
3 add = lambda a, b : a+b
4 result = add(3, 4)
5 print(result)
6
```

파일 처리

- 텍스트 파일의 처리
- 파일 열기 (open) – 파일 읽기 (read) – 파일 쓰기 (write)
- 파일 열고 닫기
 - `open()` 함수

파일 객체 = `open`(문자열: 파일 경로, 문자열: 읽기 모드)

- 모드에 다음을 지정할 수 있음

모드	설명
w	write 모드(새로 쓰기 모드)
a	append 모드(뒤에 이어서 쓰기 모드)
r	read 모드(읽기 모드)

파일 처리

- `closed()` 함수

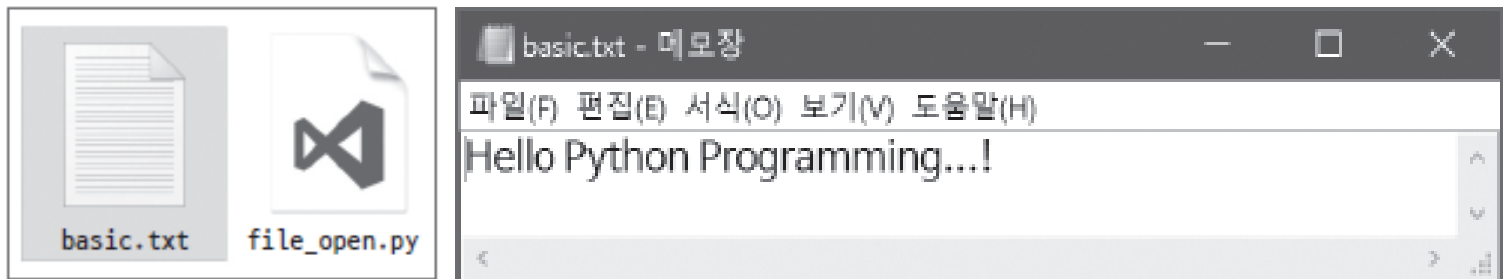
파일 객체.`close()`

- 예시

```
01  # 파일을 엽니다.  
02  file = open("basic.txt", "w")  
03  
04  # 파일에 텍스트를 씁니다.  
05  file.write("Hello Python Programming...!")  
06  
07  # 파일을 닫습니다.  
08  file.close()
```

파일 처리

- 프로그램 실행 시 폴더에 basic.txt 파일 생성
- 실행 시 다음 형태



- open() 함수로 열면 close() 함수로 닫아야 함

파일 처리

- with 키워드
 - 조건문과 반복문 들어가다 보면 파일을 열고서 닫지 않는 실수 하는 경우 생길 수 있음
 - with 구문 종료 시 파일을 자동으로 닫음

```
with open(문자열: 파일 경로, 문자열: 모드) as 파일 객체:  
    문장
```

```
# 파일을 엽니다.  
with open("basic.txt", "w") as file:  
    # 파일에 텍스트를 씁니다.  
    file.write("Hello Python Programming...!")
```

파일 처리

- 텍스트 읽기
 - read() 함수

파일 객체.read()

```
01  # 파일을 엽니다.
02  with open("basic.txt", "r") as file:
03      # 파일을 읽고 출력합니다.
04      contents = file.read()
05  print(contents)
```

→ 읽기 모드로 변경했습니다!

실행결과

Hello Python Programming...!

파일 처리

- 텍스트 한 줄씩 읽기

- CSV, XML, JSON 방법 등으로 텍스트를 사용해 데이터를 구조적으로 표현

- CSV 예시

```
이름, 키, 몸무게  
윤인성, 176, 62  
연하진, 169, 50
```

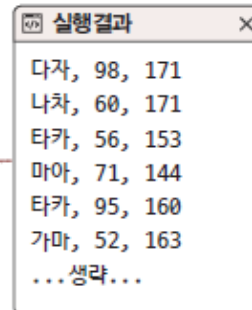
- 한 줄에 하나의 데이터 나타내며 각 줄은 쉼표 사용해 데이터 구분함
- 첫 줄에 헤더 넣어 각 데이터가 나타내는 바 설명
- 한 번에 모든 데이터 올려놓고 사용하는 것이 컴퓨터 성능에
영향 미칠 수도 있음

파일 처리

- 예시 - 랜덤하게 1000명의 키와 몸무게 만들기

```
01  # 랜덤한 숫자를 만들기 위해 가져옵니다.  
02  import random  
03  # 간단한 한글 리스트를 만듭니다.  
04  hanguls = list("가나다라마바사아자차카타파하")  
05  # 파일을 쓰기 모드로 엽니다.  
06  with open("info.txt", "w") as file:  
07      for i in range(1000):  
08          # 랜덤한 값으로 변수를 생성합니다.  
09          name = random.choice(hanguls) + random.choice(hanguls)  
10          weight = random.randrange(40, 100)  
11          height = random.randrange(140, 200)  
12          # 텍스트를 씁니다.  
13          file.write("{} {}, {} \n".format(name, weight, height))
```

info.txt에 생성된 데이터입니다. ←



파일 처리

- 데이터를 한 줄씩 읽어들이는 때는 for 반복문을 다음과 같이 사용

```
for 한 줄을 나타내는 문자열 in 파일 객체:  
    처리
```

- 키와 몸무게로 비만도 계산

```
01  with open("info.txt", "r") as file:  
02      for line in file :  
03          # 변수를 선언합니다.  
04          (name, weight, height) = line.strip().split(", ")  
05  
06          # 데이터가 문제없는지 확인합니다: 문제가 있으면 지나감  
07          if (not name) or (not weight) or (not height):  
08              continue  
09          # 결과를 계산합니다.  
10          bmi = int(weight) / (int(height) * int(height))
```

파일 처리

```
11     result = ""
12     if 25 <= bmi:
13         result = "과체중"
14     elif 18.5 <= bmi:
15         result = "정상 체중"
16     else:
17         result = "저체중"
18
19     # 출력합니다.
20     print('\n'.join([
21         "이름: {}".format(name),
22         "몸무게: {}".format(weight),
23         "키: {}".format(height),
24         "BMI: {}".format(bmi),
25         "결과: {}".format(result)
26     ]).format(name, weight, height, bmi, result))
27     print()
```

실행결과

이름: 타나
몸무게: 63
키: 165
BMI: 0.0023140495867768596
결과: 과체중

이름: 마나
몸무게: 58
키: 187
BMI: 0.0016586119134090194
결과: 저체중

이름: 바타
몸무게: 53
키: 161
BMI: 0.0020446742023841674
결과: 저체중

...생략...

확인문제

1. 주어진 자연수가 홀수 인지 짝수인지 판별해 주는 함수(is_odd)를 작성해 보자.

```
def is_odd(number):  
    if number % 2 == 1 : # 2로 나누었을 때 나머지가 1이면 홀수이다.  
        return True  
    else :  
        return False
```

확인문제

2. 입력으로 들어오는 모든 수의 평균값을 계산하는 함수를 작성해 보자.

```
def avg_number(      ):
    result = 0
    for i in args:
        result += i
    return              # 평균값을 구 할때 len()함수를 사용해보자

print(avg_number(1,2))      #1.5 출력
print(avg_number(1,2,3,4,5) )  #3.0출력
```

확인문제

3. 다음은 두 개의 숫자를 입력 받아 더하여 돌려주는 프로그램이다.

```
input1 = input("첫번째 숫자를 입력하세요:")  
input2 = input("두번째 숫자를 입력하세요:")  
  
total = input1+input2  
print("두수의 합은 %s 입니다. " % total)
```

이 프로그램을 수행해 보자

```
첫번째 숫자를 입력하세요:3  
두번째 숫자를 입력하세요:6  
두수의 합은 36 입니다.
```

3과 6을 입력했을 때 9가 아닌 36이라는 결과값을 돌려주었다. 오류를 수정해 보자. **힌트: int()함수 사용**

확인문제

4. 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.