# Ch4-1.함수 (Function)

- 함수는 재사용 가능한 프로그램의 조각을 말한다. 이것은 특정 명령어 덩어리를 묶어 이름을 짓고, 그 이름을 프로그램 어디에서건 사용함으로써 그 명령어들을 몇 번이고 다시 실행할 수 있게 하는 것이다.
- 함수를 이용한 코딩은 반복을 줄여주고 코드 좀더 구조적으로 파악할 수 있기 때문에 효율적인 프로그래밍을 위해서는 필수적인 부분이다.

# Table of Contents


- 1 함수란 무엇인가?
  - 1.0.1 함수란 무엇인가?
  - 1.0.2 함수는 왜 사용할까?
- 2 함수의 기본 구조
- 3 입력값과 결과값에 따른 함수 형태
  - 3.0.1 일반적인 함수
  - 3.0.2 입력값이 없는 함수
  - 3.0.3 결과값이 없는 함수
  - 3.0.4 입력값도 결과값도 없는 함수
- 4 입력값이 몇 개 인지 모를 경우
- 5 함수의 결과값은 언제나 하나!
- 6 매개변수에 초기값 미리 설정하기
- 7 함수 안에서 선언한 변수의 효력 범위
- 8 함수 안에서 함수 밖의 변수를 변경하는 방법
  - 8.0.1 return 사용
  - 8.0.2 global 명령어 사용
- 9 lambda (람다) 사용


## 1. 함수란 무엇인가?

#### 함수란 무엇인가?

함수를 설명하기 전에 자판기를 한번 생각해보자. 사용자가 돈을 자판기에 넣는다.
그리고 자판기에서 커피가 나온다. 이때 돈은 "입력"이고, 커피는 "출력(결과값)"이 된다.

그렇다면, 자판기는 무엇인가? 자판기는 돈을 입력받아 커피를 출력하는 "함수"와 같다.

입력값을 가지고 무언가를 수행한 뒤에 합당한 결과물을 내어 놓는 것이 바로 함수가 하는 일이다.

**함수는 왜 사용할까?**

- 프로그래밍을 하다보면 중복되는 코드가 많이 발생한다. 이때 중복되는 코드를 하나의 함수로 정의하고 가져다 쓰면 많은 **중복을 줄일수 있어 효율적**이다.
- 프로그램을 함수화하면 **프로그램 흐름을 일목요연**하게 볼 수 있다. 입력값이 여러 함수를 거치면서 원하는 결과값을 환하게 되는데, 이 과정에서 프로그램 흐름도 잘 파악할 수 있고 오류가 어디에서 나는지도 바로 알아차릴 수 있다.

# 2. 함수의 기본 구조

파이선 함수의 기본 구조는 다음과 같다.

In [11]:
```
def 함수명(매개변수):
  "수행할 문장1"
  "수행할 문장2"

  return "결과값"
```

함수를 만들때 아래의 사항을 유의해서 만들도록 하자.

* **매개변수** (parameter): 함수에 입력으로 전달되는 값을 받는 변수
* **인수** (arguments) : 함수를 호출할 때 전달하는 입력값, 매개변수에 입력하고자 하는값
* 매개변수를 정의한 후 콜론(:)을 반드시 입력해줘야 함
* 입력값 이용을 원하지 않는 경우 매개변수 부분을 공란으로 함
* 함수 내용은 탭(tab) 또는 스페이스 4개를 이용해 들여쓰기 해줘야 함
* return은 생략할 수 있음
* return이 수행되면 함수가 종료됨

위의 내용을 덧셈을 수행하는 간단한 함수를 만들어 확인해 보자.

In [24]:
```
"""
함수 이름 : add
매개 변수 : a, b (두개의 값을 입력 받음)
결과값 : 두개의 입력값을 더한 값
"""

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

In [25]:
```
add(1,3)
```

Out[25]: 4

In [14]:
```
a=1
```

```
b=3
c=add(a,b)
print(c)
```

4

이 예제에서 매개변수는 a,b 이고, 인수는 1,3 이 된다. 매개변수와 인수를 혼용하지 않도록 하자.

# 3. 입력값과 결과값에 따른 함수 형태

- 함수의 형태는 입력값과 결괏값의 존재 유무에 따라 4가지 유형으로 나뉜다.

### 일반적인 함수

* 입력값이 있고 결괏값이 있는 함수로, 가장 많이사용하는 일반적인 함수이다.

In [22]:
```
# 일반적인 함수의 기본 구조

def 함수이름(매개변수):
  "수행할 문장"

  return 결과값
```

In [23]:
```
# 일반적인 함수의 예시
def add(a, b):
  result = a + b
  return result
```

In [24]:
```
# 결괏값을 받을 변수 = 함수이름(입력인수1, 입력인수2, ...)
a = add(3,4)
print(a)
```

7

### 입력값이 없는 함수

* 입력값이 없으므로 함수 기본 구조에서 매개변수 부분이 공란으로 표시된다.

In [26]:
```
# 입력값이 없는 함수의 기본 구조
def 함수이름( ):
  "수행할 문장"
  return 결과값
```

In [26]:
```
# 입력값이 없는 함수의 예시
def say():
```

```
    return 'hello'
```

In [27]:
```
# 결과 값을 받을 변수 = 함수이름()
a = say()
print(a)
```

hello

### 결과값이 없는 함수

* 결과값이 없으므로 return 을 생략하고, return문 대신 print문을 사용한다.

In [31]:
```
# 결과값이 없는 함수의 기본 구조
def 함수명(매개변수):
    print("출력할 문장")
```

In [29]:
```
# 결과값이 없는 함수의 예제
def say(a):
    print("Hello, %s " % (a))
```

In [30]:
```
# 사용법 : 함수이름(입력인수1, 입력인수2, ...)
say("python")
```

Hello, python

In [31]:
```
say("Tom")
```

Hello, Tom

진짜 결과값이 없을까? 확인해 보자.

In [32]:
```
x = say("python")
```

Hello, python

In [34]:
```
x
```

In [35]:
```
print(x)
```

None

print문은 함수의 구성 요소 중 하나인 <수행할 문장>에 해당하는 부분일 뿐이다. 결과값은 당연히 없다.

결과값은 오직 return 명령어로만 돌려받을 수 있다.

### 입력값도 결과값도 없는 함수

* 결과값이 없으므로 return 생략하고, return문 대신 print문을 사용한다.

* 입력값도 없기 때문에 매개변수 공란으로 남겨둔다.

In [43]:
```
# 입력값도 결과값도 없는 함수의 기본 구조
def 함수명():
    print("출력할 문장")
```

In [36]:
```
# 입력값도 결과값도 없는 함수의 예시
def say():
    print('Hello')
```

In [38]:
```
# 사용법 : 함수이름()
a=say()
```

Hello

# 4. 입력값이 몇 개 인지 모를 경우

- 처음에 함수를 만들 때 입력값이 몇개인지 모르는 경우가 있다.
- 또는 입력값의 갯수가 경우에 따라 달라지는 경우도 존재할 것이다.
- 이런 경우 매개 변수를 지정할때 난감한 상황을 직면하게 된다.
- 파이썬은 이러한 문제를 매개변수 이름 앞에 *를 사용하여 해결할 수 있도록 한다.

In [50]:
```
# 입력값이 몇 개 인지 모를 경우 함수 기본 구조
def 함수이름(*매개변수):
    "수행할 문장"
    ...
```

다음과 같이 여러 개의 입력값을 모두 더하는 함수를 직접 만들어 보자. 이 때 여러개의 입력값의 갯수가 정해져있지 않다면, 매개변수 앞에 * 을 사용하여 함수를 만들 수 있다.

In [50]:
```
def add_many(*args): # args는 임의로 정한 매개변수의 이름 (매개변수를 뜻하는 arguments의 약자로 관례적으로 자주씀)
    result = 0
    for i in args:
        result = result + i
    return result
```

In [48]:
```
result1= add_many(1,2)
print(result1)
```

3

In [49]:
```
result2 = add_many(1,2,3,4,5,6,7,8,9,10)
print(result2)
```

55

여러개의 입력을 처리할때, 매개변수의 개수를 모르는 매개변수만 사용할 수 있는 것은 아니

다. 개수를 알고 있는 매개변수와 모르는 매개변수를 함께 사용하고자 하는 경우 아래와 같이 사용할 수 있다.

예를 위해서, 선택에서 따라 주어진 입력값들을 다 더하거나, 곱하는 함수를 만들고자 한다.

In [51]:
```
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
```

In [52]:
```
result1 = add_mul('add', 1,2,3,4,5)
print(result1)
```

15

In [55]:
```
result1 = add_mul('mul', 1,2,3,4,5)
print(result1)
```

120

**args* 처럼 매개변수 이름 앞에 * 을 붙이면 입력값을 전부 모아서 튜플 (tuple)로 만들어 주고 함수의 문장이 수행된다. 결과값은 언제나 하나의 값만이 반환된다.

참고로, 여러개의 입력값을 딕셔너리로 만들어 함수를 수행하고자 하는 경우는 키워드 파라미터를 사용할 수있다. 키워드 파라미터를 사용할 때는 매개변수 앞에 별 두 개(**)를 붙인다.

In [56]:
```
# 키워드 파라미터 예제 (매개변수를 출력하는 함수)
def print_kwargs(**kwargs):  # kwargs는 keyword arguments의 약자이며 args와 마찬가지로 관례적으로 사용
  print(kwargs)
```

In [57]:
```
print_kwargs(name='foo', age=3)  # 입력된 매개변수는 딕셔너리가 되고, key=value 형태의 결과값이 저장됨
```

{'name': 'foo', 'age': 3}

In [58]:
```
print_kwargs(a=1)
```

{'a': 1}

## 5. 함수의 결과값은 언제나 하나!

- 함수는 언제나 하나의 결과값만을 반환한다.
- 여러 값을 반환하고자 하는 경우 리스트, 튜플로 묶어 결과값 반환하는 방법이 있다.

아래의 다양한 예시들을 살펴보고 결과를 예측하여보자.

In [59]:
```
# case1
def add_and_mul(a,b):
    return a+b
    return a*b


result = add_and_mul(2, 3)
print(result)  # 어떤 결과가 나올까? -> 처음 return문을 만나는 순간 결괏값을 돌려주고 함수를 빠져나감 -> 하나의 결과값만 얻어짐
```

5

In [62]:
```
# Case 2
def add_and_mul(a,b):
    return a+b , a*b

# 어떤 결과가 나올까? -> 두개의 값이 나온듯하지만, 두개의 결과값이 하나의 투플형 변수에 저장된 것 뿐이다!
x=add_and_mul(1,3)
```

In [61]:
```
# Case 3 : 두개의 결과값을 각각의 변수에 저장하고자 함
result1, result2 = add_and_mul(3, 4)  # 튜플의 각 요소를 변수로 저장
print(result1)
print(result2)
```

7
12

## 6. 매개변수에 초기값 미리 설정하기

- 매개변수에 초기값을 미리 설정하여 함수의 인수를 전달할 수 있다.
- 유의사항 : 초기값 설정된 변수 뒤에는 초기값이 설정되지 않은 변수가 올 수 없음 즉, 초기값 설정된 변수들은 뒤에 위치해야 함

In [65]:
```
def say_myself(name, old , man=True):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

In [68]:
```
say_myself(name="제임스", old=28)
```

나의 이름은 제임스 입니다.
나이는 28살입니다.
남자입니다.

In [67]:
```
say_myself("제임스", 28, False)
```

```
나의 이름은 제임스 입니다.
나이는 28살입니다.
여자입니다.
```

In [ ]:
```
# 어떻게 될까?
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("제임스", 28)  # 28 을 man 과 old 중에 어디에 입력해야할지 모르겠다...(파이썬 입장)
```

## 7. 함수 안에서 선언한 변수의 효력 범위

- 함수 안에서 선언한 매개변수는 함수 안에서만 사용될 뿐 함수 밖에서는 사용되지 않는다.

다음의 함수가 있을 때 아래 두개의 결과를 예상해보자.

In [71]:
```
a = 1
def vartest(a):
    a = a +1      # 함수에서 사용된 a는 함수 밖에서는 사용되지 않음
    return(a)
```

In [72]:
```
vartest(a)
```

Out[72]: 2

In [73]:
```
a
```

Out[73]: 1

자, 그럼 아래의 함수에 대해 print(k)의 결과는 무엇일까?

In [74]:
```
def vartest(k):
    k = k + 1

vartest(3)
```

In [ ]:
```
print(k)
```

# 8. 함수 안에서 함수 밖의 변수를 변경하는 방법

**return 사용**

- return을 사용한 뒤 그 함수를 특정한 변수로 a = vartest(a) 와 같이 대입한다. 그럼 a 가 기존 1에서, vartest 함수 결과 값으로 바뀐다. 여기에서도 물론 vartest 함수 안의 a 매개변수는 함수 밖의 a와는 다른 것이다.

In [1]:
```
a = 1
def vartest(a):
    a = a +1
    return a

a = vartest(a)
print(a)
```

2

**global 명령어 사용**

- global 명령어를 사용하여 함수 안에서 함수 밖의 변수를 변경 가능하지만 사용하지 않는 것이 좋다.
- 왜냐하면, 함수는 독립적으로 존재하는 것이 좋고, 외부 변수에 종속적인 함수는 좋은 함수가 아니기 때문이다. (언제나 쉽게 가져다 쓰고 떼어낼 수 있는 함수가 좋다!)
- 가급적 global 명령어를 사용하는 이 방법은 피하고 첫 번째 방법을 사용하기를 권한다.

In [2]:
```
a = 1
def vartest():
    global a    # 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 뜻
    a = a+1

vartest()
print(a)
```

2

# 9. lambda (람다) 사용

- lambda는 함수를 생성할 때 사용하는 예약어로, def와 동일한 역할을 한다.
- 주로 함수를 한줄로 간결하게 만들 때 사용한다.
- lambda 예약어로 만든 함수는 return 명령어가 없어도 결과값을 반환한다.
- def를 사용해야 할 정도로 복잡하지 않거나 def를 사용할 수 없는 곳에 주로 쓰인다.

1) 기본 구조

lambda 매개변수1, 매개변수2, ... : 매개변수를 이용한 표현식(수행식)

2) 예시

In [3]:
```
add = lambda a, b: a+b      # 매개변수 a,b,를 이용해서 a+b 한 것을 결과값으로 반환, return이 없이도 간단하게 함수 표현
result = add(3, 4)
print(result)
```

7

## 10. Example

Q1) 함수의 입력으로 들어오는 모든 수의 평균 값을 계산해 주는 함수를 작성해 보자.

- 단, 입력으로 들어오는 수의 개수는 정해져 있지 않다.
- 평균값을 구할 때 len 함수를 사용해 보자.

Q2) 주어진 자연수가 홀수인지 짝수인지 판별해 주는 함수(is_odd)를 작성해 보자.

- 홀수인 경우 True, 짝수인 경우 False가 되도록 한다.

Q3) 세개의 숫자를 입력받아 가장 큰수를 출력하는 print_max 함수를 정의하라.

- 단 if 문을 사용해서 수를 비교할 수 있다.

Q4) 어떤 수(number)를 입력받아 각 자리 숫자(digit)의 합을 계산하는
sumOfDigits 라는 이름의 함수를 작성하자.

# Ch4-2. 사용자 입출력 (User Input and Output)

- 우리는 이미 함수 부분에서 입력과 출력이 어떤 의미를 가지는지 알아보았다.
- 지금부터는 좀 더 다양하게 사용자의 입력을 받는 방법과 출력하는 방법을 알아보자.

# Table of Contents


- 1 사용자 입력
- 2 print 자세히 알기
  - 2.0.1 큰따옴표(")로 둘러싸인 문자열은 + 연산과 동일하다
  - 2.0.2 문자열 띄어쓰기는 콤마로 한다
  - 2.0.3 한 줄에 결괏값 출력하기
- 3 Example


## 1. 사용자 입력

- 사용자가 입력한 값을 어떤 변수에 대입하고 싶을 때 input 을 사용할 수 있다.
- input은 입력되는 모든 것을 문자열로 취급한다.

In [5]:
```
a = input()  # 네모 박스에 입력값을 넣으면, 문자열로 저장된다.
```

```
Life is too short, you need python
```

In [6]:
```
print(a)
```

```
Life is too short, you need python
```

- 사용자에게 입력받을 때 안내 문구 또는 질문이 나오도록 하고 싶을 때 input() 괄호 안에 질문내용/안내 문구를 넣을 수 있다.

In [7]:
```
number = input("숫자를 입력하세요: ")
```

```
숫자를 입력하세요: 6
```

In [8]:
```
print(number)
```

```
6
```

## 2. print 자세히 알기

- 우리가 지금껏 써 왔던 print문이 수행해 온 일은 우리가 입력한 자료형을 출력하는 것이

었다. print의 사용예는 다음과 같다.

In [9]:
```
a = 123
print(a)

b = "Python"
print(b)

c = [1, 2, 3]
print(c)
```

```
123
Python
[1, 2, 3]
```

이제 print문으로 할 수 있는 일에 대해서 조금 더 자세하게 알아보자.

큰따옴표(")로 둘러싸인 문자열은 + 연산과 동일하다

In [10]:
```
print("life" "is" "too short")
print("life"+"is"+"too short")
```

```
lifeistoo short
lifeistoo short
```

문자열 띄어쓰기는 콤마로 한다

In [11]:
```
print("life", "is", "too short")
```

```
life is too short
```

**한 줄에 결괏값 출력하기**

기본적으로 아래와 같이 for문을 이용한 print 결과값은 반복마다 줄바꿈을 실시한다.

In [12]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

한 줄에 결과값을 계속 이어서 출력하려면 매개변수 end를 사용해 끝 문자를 지정해야 한다.

In [17]:
```
for i in range(5):
    print(i, end=' ')  # 해당 시점의 i를 출력하고, blank 를 붙임
```

```
0 1 2 3 4
```

In [16]:
```
for i in range(5):
    print(i, end=",")
```

0,1,2,3,4,

## 3. Example

Q1) 다음은 두 개의 숫자를 입력받아 더하여 돌려주는 프로그램이다.

In [16]:
```
input1 = input("첫번째 숫자를 입력하세요:")
input2 = input("두번째 숫자를 입력하세요:")

total = input1 + input2
print("두 수의 합은 %s 입니다" % total)
```

첫번째 숫자를 입력하세요:10
두번째 숫자를 입력하세요:5
두 수의 합은 105 입니다

이 프로그램을 수행해 보고, 이 프로그램의 오류를 수정해보자.

Q2) 다음 중 출력 결과가 다른 것 한 개를 골라 보자.

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

Q3) 아래 예시와 같이 사용자 입력을 통해 순차적으로 입력받은 정수를 계속 더해나가다가, 음수가 입력되면 중단하고 그 전까지 계산한 값을 출력하는 프로그램을 작성해보자.

* 입력예시 :
  1
  2
  3
  -1
* 출력예시 :
  6

# Ch4-3. 파일 읽고 쓰기 (Reading and Writing files)

- 지금까지는 입력과 출력에 대해 "입력"은 사용자가 직접 입력하는 방식을 사용했고 "출력"은 모니터 화면에 결괏값을 출력하는 방식으로 코딩해보았다. 하지만, 입출력 방식은 이것이 전부가 아니다.
- 이번에는 파일을 통한 입출력 방법에 대해 알아보고자 한다. 파일의 내용을 읽고, 프로그래밍 결과를 새로운 내용을 추가하여 저장해 보자.

# Table of Contents


- 1 파일 생성하기
- 2 파일을 쓰기 모드로 열어 출력값 적기
- 3 프로그램의 외부에 저장된 파일을 읽는 여러 가지 방법
  - 3.0.1 readline() 함수 이용하기
  - 3.0.2 readlines 함수 사용하기
  - 3.0.3 read 함수 사용하기
- 4 파일에 새로운 내용 추가하기
- 5 open(), close() 함수의 간략화 : with문 사용


## 1. 파일 생성하기

- 파일을 생성하기 위해 파이썬 내장 함수 open을 사용한다.
- open 함수는 다음과 같이 "파일 이름"과 "파일 열기 모드"를 입력값으로 받고 결과값으로 파일 객체를 반환한다.

    파일 객체 = open(파일 이름, 파일 열기 모드)

- open 함수 파일 열기모드

| Mode | Description |
|---|---|
| r | 읽기 모드 - 파일을 읽기만 할 때 사용 |
| w | 쓰기 모드 - 파일에 내용을 쓸 때 사용 |
| a | 추가 모드 - 파일의 마지막에 새로운 내용을 추가 시킬 때 사용 |

- Tips : 파일을 쓰기 모드로 열면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고 다시 쓰임, 해당 파일이 존재하지 않으면 새로운 파일이 생성

In [4]:
```
f = open("새파일.txt", 'w') # 쓰기 모드(w) 로 파일 객체 f 를 생성, 새파일.txt 라는 이름의 파일로 현재 디렉토리에 저장됨
f.close()                   # 파일 객체 f 를 닫아 주는 역할
# 생략 가능하지만 사용해주는 것이 좋음 (쓰기모드로 열었던 파일을 닫지 않고 다시 사용하려고 하면 오류가 발생하기 때문)
```

위 예에서는 디렉토리에 파일이 없는 상태에서 새파일.txt를 쓰기 모드인 'w'로 열었기 때문에 새파일.txt라는 이름의 새로운 파일이 현재 디렉토리에 생성되는 것이다. (보통 현재 디텍토리는 현재 노트북이 저장되고 있는 디렉토리이다.)

만약 새파일.txt 파일을 특정 디렉토리(C:/user/my) 에 생성하고 싶다면 경로와 함께 다음과 같이 작성하면 된다.

In [ ]:
```
f = open("C:/user/my/새파일.txt", 'w') # 파일 경로 지정
f.close()
```

## 2. 파일을 쓰기 모드로 열어 출력값 적기

위 예에서는 파일을 쓰기 모드로 열기만 했지 정작 아무것도 쓰지는 않았다. 이번에는 프로그램의 출력값을 파일에 직접 써 보자.

In [24]:
```
f = open("새파일.txt", 'w')        # 파일 생성해 쓰기 모드
for i in range(1, 5):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)                  # 파일에 쓸 내용
f.close()                          # 완료 후 객체 닫기
```

새파일.txt 을 열어보면 아래 프로그램의 출력 내용과 동일한 것을 확인 할 수 있다. 프로그램의 출력값을 파일에 직접 출력하고자 하는 경우 close()전에 출력하고자 하는 내용을 적으면 된다.

In [25]:
```
for i in range(1, 5):
    data = "%d번째 줄입니다." % i
    print(data)  # print는 줄바꿈 (\n)이 없어도 각 반복에 대해서 줄바꿈이 됨
```

```
1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
```

## 3. 프로그램의 외부에 저장된 파일을 읽는 여러 가지 방법

파이썬에는 외부 파일을 읽어 들여 프로그램에서 사용할 수 있는 여러 가지 방법이 있다. 이번에는 그 방법을 자세히 알아보자.

### readline() 함수 이용하기

- 파일의 첫 번째 줄을 읽어 출력하는 함수 readline() 을 사용해 보자.

In [35]:
```
f = open("새파일.txt", 'r') # 읽기 모드
line = f.readline() # 첫 번째 줄을 읽어 출력
print(line)
f.close()
```

```
1번째 줄입니다.
```

위의 결과는 새파일.txt의 가장 첫 번째 줄이 화면에 출력될 것이다.

만약 모든 줄을 읽어서 화면에 출력하고 싶다면 다음과 같이 작성하면 된다.

In [27]:
```
f = open("새파일.txt", 'r')
while True:                 # 무한 루프
    line = f.readline()     # 계속해서 한 줄씩 읽어 출력
    if not line: break      # 더 이상 읽을 줄이 없으면 break 수행
                            # readline()은 더 이상 읽을 줄이 없을 경우 빈 문자열 반환
    print(line)
f.close()
```

```
1번째 줄입니다.

2번째 줄입니다.

3번째 줄입니다.

4번째 줄입니다.
```

### readlines 함수 사용하기

- 두 번째 방법은 readlines 함수를 사용하는 방법이다.
- readlines 함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트로 출력하는 함수이다. 다음 예를 보자.

In [40]:
```
f = open("새파일.txt", 'r')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

```
1번째 줄입니다.

2번째 줄입니다.

3번째 줄입니다.

4번째 줄입니다.
```

위예에서 lines는 아래와 같은 리스트가 된다.

In [30]:
```
lines
```

Out[30]:
```
['1번째 줄입니다.\n', '2번째 줄입니다.\n', '3번째 줄입니다.\n', '4번째 줄입니다.\n']
```

### read 함수 사용하기

- 마지막은 read 함수를 사용하는 방법이다.
- read() 함수는 파일의 내용 전체를 문자열로 출력하는 함수이다.

In [41]:
```
f = open("새파일.txt", 'r')
data = f.read()
print(data)
f.close()
```

```
1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
```

In [42]:
```
data
```

Out[42]:
```
'1번째 줄입니다.\n2번째 줄입니다.\n3번째 줄입니다.\n4번째 줄입니다.\n'
```

## 4. 파일에 새로운 내용 추가하기

- 파일의 원래 있던 값을 유지하면서 단지 새로운 값만 추가하고 싶은 경우, 파일을 추가 모드('a')로 열면 된다.

In [43]:
```
f = open("새파일.txt", 'a')    # 추가 모드 : 'a'
for i in range(5, 10):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

In [44]:
```
# 새로운 내용이 추가된 파일을 다시 열어서 추가된 내용을 확인해보자.
f = open("새파일.txt", 'r')
data = f.read()
print(data)
f.close()
```

```
1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
5번째 줄입니다.
```

```
6번째 줄입니다.
7번째 줄입니다.
8번째 줄입니다.
9번째 줄입니다.
```

## 5. open(), close() 함수의 간략화 : with문 사용

- 파일 생성 또는 파일 읽기를 할 때마다 open()함수와 close()함수를 함께 매번 사용해야 하는 번거로움이 있다.
- 이러한 번거로움을 해소하기 위해 with문 사용한다.
- with문은 파일을 열고 닫는 것을 자동으로 처리해준다는 장점이 있다.

In [45]:
```
# with 가 없을때... 매번 open()-close()
f = open("foo.txt", 'w')
f.write("Life is too short, you need python")
f.close()
```

In [39]:
```
# with를 사용하면, 파일을 열고 닫는것을 한번에 코딩 할 수 있다.
with open("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 close된다.