입력과 출력

입력과 출력

입력과 출력

- 코딩할 때 실행한 결과를 화면이나 파일로 출력해야 할 때
- 입력을 키보드로 받거나 파일에 있는 데이터를 읽어서 처리해야 할 때
- 파이썬으로 코드를 작성할 때 키보드와 화면으로 입·출력하는 방법과 파일로 입 · 출력하는 방법

화면 출력

화면 출력

- 작성한 코드의 결과를 확인하는 가장 기본적인 방법은 결과를 화면으로 출력하는 것
- 파이썬에서는 print() 함수를 이용해 원하는 내용을 화면으로 출력
- print() 함수에는 출력 형식을 지정하지 않는 기본 출력 방법과 다양한 형식으로 출력할 수 있는 형식 지정 출력 방법

화면 출력

화면 출력

- 작성한 코드의 결과를 확인하는 가장 기본적인 방법은 결과를 화면으로 출력하는 것
- 파이썬에 서는 print() 함수를 이용해 원하는 내용을 화면으로 출력
- print() 함수에는 출력 형식을 지정하지 않는 기본 출력 방법과 다양한 형식으로 출력할 수 있는 형식 지정 출력 방법

화면 출력

기본 출력

- print() 함수의 기본 출력 방법을 이용해 문자열과 숫자를 출력하는 방법
- 문자 열을 출력하려면 문자열을 pint() 함수 안에 넣으면 됨

```
[ ] print("Hello Python!!")
```

화면 출력

기본 출력

- 문자열 여러 개를 연결해 출력하려면 문자열을 콤마(,)로 구분하고 연속해서 입력
- 콤마로 구분 하면 출력에서 자동으로 빈칸(공백)이 하나씩 들어감

```
[ ] print("Best", "python", "book")
```

화면 출력

기본 출력

- 빈칸 대신 다른 문자열을 넣으려면 print() 함수 안에 두 문자열 사이의 구분하는 값을 설정하는 sep 인자를 이용
- print()함수에서 sep 인자를 지정하지 않으면 기본적으로 빈칸이 들어감
- 빈칸을 다른 문자열로 바꾸려면 'sep = 문자열'을 추가

```
[ ] print("Best", "python", "book", sep = "-:*:-")
```

  - sep 인자에 지정한 문자열로 입력한 세 개의 문자열이 연결되어 출력

화면 출력

기본 출력

- 빈칸 없이 두 문자열을 연결하려면 콤마 대신 더하기 연산자를 이용

```
[ ] print("abcd" + "efg")
```

- 문자열을 연결할 때 콤마와 더하기 연산자를 동시에 사용할 수 있음

```
[ ] print("Best", "python", "book" + ":", "This book")
```

화면 출력

기본 출력

- 변수에 저장된 내용을 출력하려면 print() 함수에 변수를 인자로 입력
- 변수 이름을 이용해 출력할 때는 따옴표가 없어야 함

```
[ ]  x = 10
     print(x)
```

화면 출력

기본 출력

- 문자열과 숫자를 함께 출력하려면 둘을 콤마로 구분해서 print() 함수의 인자로 입력
- 문자열과 문자열은 더하기 연산자로 공백 없이 연결해서 출력 가능
- 문자열과 숫자는 더하기 연산자로 연결 할 수 없음

```
[ ]  name = "James"
     ID_num = 789
     print("Name:", name + ",", "ID Number:", ID_num )
```

화면 출력

기본 출력

- print() 함수를 이용해 문자열을 출력할 때 결과가 한 줄씩 출력
- print() 함수를 하나만 이용해서 문자열을 여러 줄로 출력하려면?
- 문자열 안에 줄 바꿈 표시인 개행 문자(₩n)를 입력하면 출력할 때 줄이 바뀜

```
[ ] print("James is my friend.₩nHe is Korean.")
```

화면 출력

기본 출력

- 개행문자(₩n)를 하나 더 추하면 다시 줄 바꿈을 함
- 개행문자(₩n)를 두 개 입력하면 줄 바꿈이 두 번 발생해 한 줄을 띄움

```
[ ] print("James is my friend.₩n₩nHe is Korean.")
```

화면 출력

기본 출력

- 줄을 바꿔서 출력해야 할 때도 있지만 줄을 바꾸지 않고 출력해야 할 때도 있음
- 파이썬 print() 함수는 기본적으로 출력을 위해서 줄을 바꿔 놓으므로 print() 함수를 두 개 사용해 문자 열을 출력하면 두 줄로 출력

```
[ ] print("Welcome to ")
    print("python!")
```

화면 출력

기본 출력

- 두 줄로 출력된 결과를 한 줄로 출력하려면 print() 함수 안에 라인 끝의 값을 지정할 수 있는 end 인자를 추가 함
- print() 함수에서 end 인자를 지정하지 않으면 기본적으로 라인 끝에 개행문자(₩n)가 들어감
- 개행문자(₩n) 대신 다른 문자열을 입력하려면 'end = 문자열' 형태로 end 인자에 다른 값을 입력

```
[ ] print("Welcome to ", end="")
    print("python!")
```

형식 지정 출력

형식 지정 출력

- print() 함수 안에 [end=""] 를 추가하면 라인 끝에 어떤 문자도 넣지 않았으므로 그 다음 print() 함수를 실행하면 줄 바꿈 없이 연결되어 출력
- 파이썬의 print() 함수에서 문자열에 데이터가 출력될 위치와 형식을 지정하는 방식으로도 데이터를 출력

형식 지정 출력

나머지 연산자(%)를 이용한 형식 및 위치 지정

- 나머지 연산자는 print() 함수에서 데이터의 출력 형식과 위치를 지정할 때도 사용
- print() 함수에서 나머지 연산자를 이용해 데이터의 출력 형식과 위치를 지정하는 기본 구조
- print("%type" % data)

형식 지정 출력

나머지 연산자(%)를 이용한 형식 및 위치 지정

- print("%type" % data)
  - print() 함수의 인자에는 두 개의 %가 있음
  - 첫 번째는 따옴표로 둘러싼 문자열 '%type'에서 사용했고 두 번째는 '%data'에서 사용
  - 따옴표와 '%data' 사이에 콤마가 없고 공백이 있다는 점에 유의
  - '%type'은 data 형식에 따라 다른 값을 지정
  - data가 문자열이면 %s를, 정수이면 %d(혹은 %i)를, 실수이면 %f(혹은 %F)를 지정
  - 실수의 경우 %f로 표시 하면 기본적으로 소수점 6자리까지 출력

형식 지정 출력

나머지 연산자(%)를 이용한 형식 및 위치 지정

- data가 두 개 이상이면
- 따옴표로 둘러싼 문자열 안에 data의 개수에 맞게 '%type'를 순서대로 입력하고 튜플 형식으로 data를 묶어서 이용
- print() 함수에서 data가 두 개일 때 나머지 연산자를 이용해 data를 출력하는 형식과 위치를 지정하는 구조
- print("%type %type" % (datal, data2))

형식 지정 출력

나머지 연산자(%)를 이용한 형식 및 위치 지정

- %s를 이용해 문자열을 대입한 변수를 출력

```
[ ]  name = "광재"
     print("%s는 나의 친구입니다." % name)
```

  - %s 자리에 변수 name에 할당된 문자열이 출력

- %d와 %f를 이용해 정수와 실수를 출력

```
[ ]  r = 3  # 변수 r에 정수 데이터 할당
     PI = 3.14159265358979 # 변수 PI에 실수 데이터 할당
     print("반지름: %d, 원주율: %f" % (r, PI)) # 지정된 위치에 데이터 출력
```

  - 출력 결과에서 %d 자리에 변수 r에 할당된 정수가
  - %f 자리에 변수 PI에 할당된 실수가 각 형식에 맞게 출력
  - 실수의 경우는 소수점 6자리까지만 출력

형식 지정 출력

형식 지정 문자열에서 출력 위치 지정

- 출력 양식을 좀 더 자유롭게 표현할 수 있는 형식 지정 문자열을 이용
- print() 함수에서 'string.format()'을 이용하는 형식 지정 문자열의 기본 구조
- print("{0} {1} {2} ••• {n}".format(data_0, data_1, data_2, •••, data_n))
  - {N}의 N은 0부터 시작하는 숫자로 format()에서 데이터의 위치(0부터 시작)를 의미
  - {N}에는 format()에서 N에 해당하는 위치의 데이터가 들어가서 출력
  - {0}에는 data_0가 출력되고 {1}에는 data_1이 출력하고 {n}에는 data_n이 출력

형식 지정 출력

형식 지정 문자열에서 출력 위치 지정

- 형식 지정 문자열의 구조인 'string.format()'으로 데이터를 출력

```
[ ] animal_0 = "cat"
    animal_1 = "dog"
    animal_2 = "fox"

    print("Animal: {0}".format(animal_0))
    print("Animal: {0},{1},{2}".format(animal_0, animal_1, animal_2))
```

형식 지정 출력

형식 지정 문자열에서 출력 위치 지정

- {N}의 위치를 변경하면 데이터의 출력 위치를 변경할 수 있음

```
[ ] print("Animal: {1},{2},{0}".format(animal_0, animal_1, animal_2))
```

- '{1}, {2}, {0}'는 format()안의 변수를 '두 번째, 세 번째, 첫 번째' 순서로 출력하라는 의미
- 결과를 보면 지정한 순서대로 출력된 것을 볼 수 있음

형식 지정 출력

형식 지정 문자열에서 출력 위치 지정

- format() 안에 데이터의 내용 중 일부만 출력할 수도 있음

```
[ ] print("Animal: {0},{2}".format(animal_0, animal_1, animal_2))
```

- 출력된 결과를 보면 첫 번째와 세 번째 데이터만 출력

형식 지정 출력

형식 지정 문자열에서 출력 위치 지정

- {N}으로 format()안의 데이터 순서를 지정
- 만약 format()안의 데이터를 순차적으로 지정하려면 {N}에 N 없이 {}만 써도 됨
- 세 개의 {}를 이용해 데이터를 순서대로 출력

```
[ ] print("Animal: {}, {}, {}".format(animal_0, animal_1, animal_2))
```

형식 지정 출력

형식 지정 문자열에서 출력 위치 지정

- ’string.format() 방식을 이용해 문자열을 출력하는 방법
- 문자열뿐만 아니라 정수나 실수도 ‘string.format()’ 방식을 이용해 출력할 수 있음
- 기본 출력 양식을 그대로 이용 하면 출력 형식을 별도로 지정하지 않아도 됨
- 단지 {N}를 이용해 변수를 출력할 위치만 지정하면 됨

형식 지정 출력

형식 지정 문자열에서 출력 위치 지정

```
[ ]  name = "Tomas"
     age = 10
     a = 0.12345678901234567890
     fmt_string = "String: {0}, Integer Number: {1}, Floating Number: {2}"
     print(fmt_string.format(name, age, a))
```

- print() 함수에서 이용할 문자열을 fmt_string 변수에 대입한 후 이를 이용
- {N}으로 데이터의 출력 위치만 지정하면 변수의 데이터 타입을 지정하지 않아도 데이터의 타입에 따라 자동으로 알아서 출력
- 실수의 경우 기본적으로 소수 점 17자리까지 표시
- 그 이상의 소수점 숫자는 반올림되어 출력

형식 지정 출력

형식 지정 문자열에서 숫자 출력 형식 지정

- string.format()'을 이용해 형식 지정 문자열을 이용하는 방법
- 데이터를 출력할 경우 출력 형식을 지정하지 않아도 데이터의 타입에 맞게 출력
- 데이터가 문자열이 아니라 숫자인 경우 {N: '출력 형식 '} 형태로 좀 더 다양하게 출력 형식을 지정할 수 있음
- N은 format()에서 N번째 데이터의 위치

형식 지정 출력

형식 지정 문자열에서 숫자 출력 형식 지정

```
[ ]  a = 0.12345678901234567890
     print("{0:.2f}, {0:.5f}".format(a))
```

- '.2f '와 '.5f는 모두 '출력 형식'을 지정한 것으로서 각각 실수를 소수점 둘째 자리와 다섯째 자리까지 표시하라는 뜻

형식 지정 출력

형식 지정 문자열에서 숫자 출력 형식 지정

- 숫자의 출력 형식 지정

데이터	출력 형식	출력 결과	설명
3	{N:2d}	<->3	정수를 공백 포함해 두 자리로 표시(<->은 공백 한 칸을 의미함)
3	{N05d}	00003	정수를 다섯 자리로 표시. 앞의 공백은 0으로 채움
12	{N:>5d}	<-><-><->121	정수를 다섯 자리로 표시. 숫자는 오른쪽으로 정렬
0.12345	{N:.3f}	0.123	실수를 소수점 셋째 자리까지 표시
7456000	{N:,}	7,456,000	통화 표시처럼 끝에서 셋째 자리마다 콤마(,)를 표시
0.3258	{N:1%}	32.6%	소수를 퍼센트(%)로 표시. 퍼센트 표시에서 소수점 자리 수는 ':'다음 숫자로 표시
92500000000	{N:.2e}	9.25e+10	숫자를 지수로 표시. 지수 표시에서 소수점 자리 수는 다음 숫자로 표시
16	{N:#x}	0x10	숫자를 16진수로 표시. #기호가 없으면 0x 없이 출력됨
8	{N:#0}	0o10	숫자를 8진수로 표시. #기호가 없으면 0o 없이 출력됨
2	{N:#b}	0b10	숫자를 2진수로 표시. #기호가 없으면 0b 없이 출력됨

키보드 입력

키보드 입력

- 키보드로 데이터를 입력하기 위해서는 input() 함수를 이용
- input() 함수를 이용해 데이터를 입력하고 그 값을 받아서 처리하는 방법
- input() 함수로 데이터를 입력하기 위한 기본 구조
- data = input ("문자열")
  - input() 함수 안의 '문자열'은 화면에 표시되고
  - 키보드로 데이터를 입력한 후 Enter를 누르면
  - 입력된 데이터는 문자열 형태로 data 변수에 대입

키보드 입력

키보드 입력

- input()함수로부터 입력 받은 데이터를 print() 함수로 출력

```
[ ]  yourName = input("당신의 이름은? ")
     print("당신은 {}이군요.".format(yourName))
```

키보드 입력

키보드 입력

- 숫자를 입력

```
[ ]  num = input("숫자를 입력하세요: ")
     print("당신이 입력한 숫자는 {}입니다.".format(num))
```

- input() 함수로부터 입력 받은 숫자를 그대로 출력해서 문제가 없었지만
- 숫자를 연산에 이용 한다면 연산 전에 입력 받은 숫자를 정수 혹은 실수로 변환
- input() 함수로부터 입력 받은 데이터는 모두 문자열로 처리되기 때문

키보드 입력

키보드 입력

- 사각형의 변의 길이를 입력 받아 넓이를 구하기

```
[ ] a = input("정사각형 한 변의 길이는?: ")
    area = int(a) ** 2
    print("정사각형의 넓이: {}".format(area))
```

- 정수가 입력될 것이라는 가정하에 int() 함수를 이용해 문자열을 정수로 변환한 후에 제곱 연산으로 넓이를 구했음

키보드 입력

키보드 입력

- 입력하려고 하는 숫자가 실수면 어떻게 할까요?

```
[ ] b = input("정사각형 한 변의 길이는?:")
    area = float(b) ** 2
    print("정사각형의 넓이: {}".format(area))
```

- 입력 숫자를 실수로 가정해 float()함수로 문자열을 실수로 변환해 연산을 수행
- 입력하려는 숫자가 정수인지 실수인지 모를 때는 무조건 float() 함수를 쓰면 됨

키보드 입력

키보드 입력

- 정수를 입력해도 float() 함수를 이용해 실수로 변환한 후에 연산을 수행
- 실수로 변환했으므로 결과는 실수로 출력

```
[ ] c = input("정사각형 한 변의 길이는?: ")
    area = float(c) ** 2
    print("정사각형의 넓이: {}".format(area))
```

파일 읽고 쓰기

파일 읽고 쓰기

- 출력 결과를 화면이 아니라 파일로 출력하거나 키보드가 아닌 파일에서 데이터를 읽어야 할 때
- 데이터를 파일로 출력(즉, 파일로 저장)하는 방법
- 데이터가 저장된 파일에서 데이터를 읽는 방법

파일 읽고 쓰기

파일 열기

- 파일에서 데이터를 읽거나 파일에 데이터를 쓰려면 우선 파이썬 내장 함수인 open()을 이용해 파일을 열어야 함
- f = open ('filename', 'mode')
  - open() 함수는 file_name과 mode를 입력 인자로 받아서 파일을 열고 파일 객체인 f를 반환
  - 반환된 파일 객체를 이용해 파일을 읽고 쓰고 닫음
  - open()의 첫 번째 인자인 file_name은 열고 자 하는 파일 이름
  - 두 번째 인자인 모드(mode)

파일 읽고 쓰기

파일 열기

- 파일 열기의 속성

mode	의미
r	읽기 모드로 파일 열기(기본). 모드를 지정하지 않으면 기본적으로 읽기 모드로 지정됨
w	쓰기 모드로 파일 열기. 같은 이름의 파일이 있으면 기존 내용은 모두 삭제됨
x	쓰기 모드로 파일 열기. 같은 이름의 파일이 있을 경우 오류가 발생함
a	추가 모드로 파일 열기. 같은 이름의 파일이 없으면 w와 기능 같음
b	바이너리 파일 모드로 파일 열기
t	텍스트 파일 모드로 파일 열기(기본). 지정하지 않으면 기본적으로 텍스트 모드로 지정됨

파일 읽고 쓰기

파일 열기

- mode는 혼합해서 사용할 수도 있음
- 바이너리 파일을 읽기 모드로 열고 싶으면

  mode에 'bw' 혹은 'wb'를 입력
- mode에 아무것도 쓰지 않으면

  'rt'와 같은 기능 (읽기 모드이면서 텍스트 모드로 파일을 오픈)
- mode에 'w'만 입력하면 'wt' 모드로 파일을 오픈

파일 읽고 쓰기

파일 쓰기

- 파일 쓰기를 하려면 우선 파일 을 쓰기 모드로 열어야 함
- 파일이 텍스트 파일인지 바이너리 파일인지도 지정할 수 있는데 특별히 지정하지 않으면 기본적으로 텍스트 파일 모드로 파일을 염
- 파일을 열고 지정한 내용을 쓴 후에는 파일을 닫아야 함

파일 읽고 쓰기

파일 쓰기

- 파일 쓰기를 위한 코드의 구조

```
f = open('file_name', 'w')

f.write(str)

f.close()
```

  - open()에서 쓰기 모드('w')로 파일을 열면 지정된 이름으로 파일을 생성한 후에 파일을 오픈
  - 같은 이름의 파일이 있다면 기존 파일의 내용을 모두 삭제하고 파일을 오픈
  - 파일을 열 때 파일 객체인 f를 반환하는데, 이를 이용해 열린 파일을 구분

파일 읽고 쓰기

파일 쓰기

- write(str)에서 str은 문자열을 의미
- write()에서는 print() 함수에서 사용하는 출력 방식을 그대로 이용할 수 있음
- 따옴표를 이용해 문자열을 파일로 출력할 수도 있고
- 형식 지정 출력 방식을 이용해 문자열을 파일로 출력할 수도 있음
- 파일에 원하는 내용을 다 썼다면 close()를 이용해 파일을 닫음
- 이 닫히면 파일 객체인 f도 사라짐

파일 읽고 쓰기

파일 쓰기

- 파일을 쓰기 모드로 연 후 문자열을 쓰고 파일을 닫는 코드를 작성

```
[ ] f = open('myFile.txt', 'w')          # (1)'myFile.txt' 파일 쓰기 모드로 열기
    f.write('This is my first file.')    # (2) 연 파일에 문자열 쓰기
    f.close()                            # (3) 파일 닫기
```

- 문자열이 저장된 파일 하나가 만들어짐
- '!cat 파일 이름'을 이용해 파일이 있는지 확인
- 파일이 있으면 내용을 화면에 출력하고 없으면 오류 메시지를 출력

```
!cat myFile.txt
```

파일 읽고 쓰기

파일 읽기

- 파일을 읽으려면 우선 파일을 읽기 모드로 열어야 함
- 그 후에 파일의 내용을 읽고 마지막으로 파일을 닫음
- 파일을 읽기 모드이면서 텍스트 모드로 연 후에 파일 내용을 읽고 파일을 닫는 구조

```
f = open('file_name', 'r')    # f = open ('filename') 도 가능
data = f.read()
f.close()
```

- read()로 읽은 파일의 내용은 모두 data 변수에 할당

파일 읽고 쓰기

파일 읽기

- 생성한 'myFile.txt' 파일을 열고
- 파일 내의 문자열을 읽은 후에 파일을 닫음
- 파일의 문자열을 잘 읽었는지 확인하기 위해 파일에서 읽은 데이터를 print() 함수로 화면에 출력

```
[ ]  f = open('myFile.txt', 'r') # (1)'myFile.txt' 파일 읽기 모드로 열기
     file_text = f.read()        # (2) 파일 내용 읽은 후에 변수에 저장
     f.close()                   # (3) 파일 닫기

     print(file_text)            # 변수에 저장된 내용 출력
```

반복문을 이용해 파일 읽고 쓰기

파일에 문자열 한 줄씩 쓰기

- 텍스트 파일의 데이터를 읽거나 데이터를 텍스트 파일로 쓸 때
- 반복문을 이용해 파일의 내용을 한 줄 씩 처리해야 할 때
- 반복문을 이용해 파일을 읽고 쓰는 방법
  - for 문을 이용해 문자열을 한 줄 씩 파일에 쓰는 방법
  - 파일을 연 후에 for 문에서 지정된 범위만큼 반복해서 문자열을 한 줄 씩 파일에 쓰고 마지막으로 파일을 닫음

반복문을 이용해 파일 읽고 쓰기

파일에 문자열 한 줄 씩 쓰기

- 구구단 2단의 일부를 파일로 저장

```
[ ] f = open('two_times_table.txt','w') # (1)파일을 쓰기 모드로 열기
    for num in range(1,6):              # (2) for문: num이 1~5까지 반복
        format_string = "2 x {0} = {1}₩n".format(num,2*num) # 저장할 문자열 생성
        f.write(format_string)          # (3) 파일에 문자열 저장
    f.close()                           # (4) 파일 닫기
```

- open()함수를 이용해 쓰기 모드로 파일을 연 후(1)
- for 문으로 반복할 범위를 지정하고(2)
- for 문에서<반복 범위>만큼 반복해서 파일에 쓰기를 수행(3)한 후
- 파일을 닫음 (4)
- write() 함수를 이용해 데이터를 파일로 쓸 때 print() 함수에서 사용한 출력 양식 이용
- write() 함수는 자동으로 줄 바꿈이 되지 않으므로 파일에서 줄을 바꾸기 위해서는 문자열 끝에 개행문자(₩n)를 추가

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- 반복문을 이용해 파일 내용을 한 줄 씩 읽는 방법
- 'f = open(‘file_name')’으로 파일을 연 후 'read()’를 이용해 파일 내용을 읽음
- 파일 내용 전체를 반환하므로 내용을 한 줄 씩 읽어서 처리해야 할 때 사용하기 어려움
- 파일 내용을 한 줄 씩 읽고 처리하려면 readline()나 readlines()를 이용

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- readline()
- 파일을 연 후 readline()을 수행하면 파일로부터 문자열 한 줄을 읽음
- readline()을 사용하면 바로 그 다음 문자열 한 줄을 읽음
- readline()은 실행한 횟수만큼 문자열을 한 줄씩 읽음
- 파일의 마지막 한 줄을 읽고 나서 다시 readline()을 수행하면 빈 문자열을 반환

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- readline()을 이용해 파일에서 문자열을 한 줄 씩 읽어 옴

```
[ ]  f = open("two_times_table.txt")    # 파일을 읽기 모드로 열기
     line1 = f.readline()               # 한 줄씩 문자열을 읽기
     line2 = f.readline()
     f.close()                          # 파일 닫기
     print(line1, end="")               # 한 줄씩 문자열 출력(줄 바꿈 안 함)
     print(line2, end="")
```

- 파일을 연 후 readline()을 이용해 문자열을 한 줄 씩 두 번 읽고 print()로 출력
- readline()으로 읽은 문자열에는 이미 개행문자(₩n)가 포함됐으므로 print()에서는 줄 바꿈이 중복되지 않게 [end=""] 처럼 end 인자를 빈 문자열("")로 설정
- 출력 결과에서 readline()으로 한 줄 씩 읽은 문자열이 잘 출력

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- readline()은 파일의 맨 끝 줄을 읽고 난 후 다시 실행하면 빈 문자열을 반환
- while 문과 readline()으로 파일 전체에서 문자열을 한 줄 씩 읽어 올 수 있음

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

```
[ ] f = open("two_times_table.txt")  # 파일을 읽기 모드로 열기
    line = f.readline()              # 문자열 한 줄 읽기
    while line:                     # line이 공백인지 검사해서 반복 여부 결정
        print(line, end = "")        # 문자열 한 줄 출력(줄 바꿈 안 함)
        line = f.readline()          # 문자열 한 줄 읽기
    f.close() # 파일 닫기
```

- line에는 readline()의 수행 결과로 가져온 한 줄 문자열이 대입되고 'while line:'에서 line이 빈 문자열인지를 검사해서
- 빈 문자열이 아니면 while 문을 계속 수행하고
- 빈 문자열이면 while 문을 빠져 나옴

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- readlines()
- readline()은 파일에서 문자열을 한 줄 씩 읽었지만
- readlines()는 파일 전체의 모든 줄을 읽어서 한 줄 씩을 요소로 갖는 리스트 타입으로 반환

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- readlines()를 이용해 파일의 전체 내용을 읽어 옴

```
[ ] f = open("two_times_table.txt") # (1) 파일을 읽기 모드로 열기
    lines = f.readlines()           # (2) 파일 전체 읽기(리스트로 반환)
    f.close()                       # (3) 파일 닫기

    print(lines)                    # 리스트 변수 내용 출력
```

- 지정된 파일을 열고 (1)
- readlines()를 이용해 파일의 전체 문자열을 읽어서 변수 lines에 할 당한 후 (2)
- 파일을 닫는 (3)
- 리스트 항목에는 파일에서 한 줄 씩 읽은 문자열이 들어가 있음
- 개행문자(₩n)도 포함 됨

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- lines 리스트에 할당된 문자열은 for 문을 이용해 항목을 하나씩 처리할 수 있음

```
[ ]  f = open("two_times_table.txt")  # 파일을 읽기 모드로 열기
     lines = f.readlines()            # 파일 전체 읽기(리스트로 반환)
     f.close()                        # 파일 닫기
     for line in lines:               # 리스트를 <반복 범위>로 지정
         print(line, end="")          # 리스트 항목을 출력(줄 바꿈 안 함)
```

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- for 문의 〈반복 범위〉에 lines 변수 대신 바로 f.readlines()를 쓰면 코드가 더 간단해 짐

```
[ ] f = open("two_times_table.txt")  # 파일을 읽기 모드로 열기
    for line in f.readlines():       # 파일 전체를 읽고, 리스트 항목을 line에 할당
        print(line, end="")          # 리스트 항목을 출력(줄 바꿈 안 함)
    f.close()
```

반복문을 이용해 파일 읽고 쓰기

파일에서 문자열 한 줄 씩 읽기

- for문의 <반복 범위>에 있는 f.readlines() 대신 f만 입력
- for 문의〈반복 범위〉에 파일 객체만 써도 '파일객체.readlines()'를 쓴 것과 같음

```
[ ] f = open("two_times_table.txt") # 파일을 읽기 모드로 열기
    for line in f:                  # 파일 전체를 읽고, 리스트 항목을 line에 할당
        print(line, end="")          # line의 내용 출력(줄 바꿈 안 함)
    f.close()                       # 파일 닫기
```

with 문을 활용해 파일 읽고 쓰기

with 문을 활용해 파일 읽고 쓰기

- open() 함수를 이용해 파일을 연 후에는 읽기나 쓰기가 끝난 후 close()로 파일을 닫음
- with 문을 이용한다면 수행이 끝난 후에 자동으로 파일을 닫기 때문에 close()를 수행하지 않아도 됨
- 파일을 읽고 쓰려면

    1. 파일 열기
    2. 파일 읽고/쓰기
    3. 파일 닫기

with 문을 활용해 파일 읽고 쓰기

with 문을 활용해 파일 읽고 쓰기

- 텍스트 파일에 쓰는 코드

```
[ ] f = open('myTextFile.txt', 'w')    # (1) 파일 열기
    f.write('File write/read test.')   # (2) 파일 쓰기
    f.close()                          # (3) 파일 닫기
```

- 텍스트 파일을 읽는 코드

```
[ ] f = open('myTextFile.txt', 'r')    # (1) 파일 열기
    test = f.read()                    # (2) 파일 읽기
    f.close()                          # (3) 파일 닫기
    print(test)
```

with 문을 활용해 파일 읽고 쓰기

with 문을 활용해 파일 읽고 쓰기

- with 문을 이용해 파일 열기를 할 경우 코드 구조

```
with open('file_name', 'mode') as f :
    〈코드 블록〉
```

  - file_name은 파일 이름이고 , mode는 파일의 속성 지정을 위한 옵션
  - f는 open() 함수의 반환 결과인 파일 객체로 〈코드 블록〉에서 파일을 읽거나 쓸 때 이용
  - <코드 블록>의 코드가 모두 끝나면 open()으로 열린 파일 객체는 자동으로 닫힘

with 문을 활용해 파일 읽고 쓰기

with 문을 활용해 파일 읽고 쓰기

- 텍스트 파일 쓰기를 위해서 작성

```
with open('file_name', 'w') as f :
    f.write(str)
```

- 텍스트 파일을 읽기

```
with open('file_name', 'r') as f :
    data = f.read()
```

  - 읽기 모드를 표시한 'r'은 써도 되고, 안 써도 됨

with 문을 활용해 파일 읽고 쓰기

with 문을 활용

- with 문을 이용해 파일에 문자열

```
[ ] with open('myTextFile2.txt', 'w') as f:                    # (1) 파일 열기
        f.write('File read/write test2: line1₩n')              # (2) 파일 쓰기
        f.write('File read/write test2: line2₩n')
        f.write('File read/write test2: line3₩n')
```

- ‘myTextFile2.txt’ 파일을 생성해서 문자열을 사용
- 파일명 뿐만 아니라 폴더의 위치까지 포함해서 파일명을 쓰는 경우 코드의 실행 위치와 상관 없이 파일을 열 수 있어서 유용

with 문을 활용해 파일 읽고 쓰기

with 문을 활용

- with 문을 반복문과 함께 이용하면 한 줄 씩 문자열을 읽고 쓸 수 있음

```
[ ]  with open('myTextFile2.txt') as f:                    # (1) 파일 열기
         file_string = f.read()                            # (2) 파일 읽기
         print(file_string)
```

with 문을 활용해 파일 읽고 쓰기

with 문을 활용

- 구구단 3단의 일부가 저장된 텍스트 파일인 myTextFile3.txt'를 만들기
- with 문에 for 문을 적용해 <반복 범위>만큼 문자열을 생성하고 파일에 쓰는 코드

```
[ ] with open('myTextFile3.txt', 'w') as f: # 파일을 쓰기 모드로 열기
        for num in range(1,6):                  # for문에서 num이 1~5까지 반복
            format_string = "3 x {0} = {1}₩n".format(num,3*num) # 문자열 생성
            f.write(format_string)              # 파일에 문자열 쓰기
```

with 문을 활용해 파일 읽고 쓰기

with 문을 활용

- with 문과 for 문을 이용해 앞에서 만든 파일 'myTextFile3.txt'의 문자열을 한 줄 씩 읽어서 출력하는 코드를 작성
- for 문의 〈반복 범위〉에는 f.readlines() 대신 f만 입력

```
[ ] with open('myTextFile3.txt', 'r') as f: # 파일을 읽기 모드로 열기
        for line in f:                      # 파일 전체를 읽고 리스트 항목을 line에 할당
            print(line, end="")  # line에 할당된 문자열 출력(줄 바꿈 안 함)
```

정리

정리

- print()를 이용해 화면에 다양한 형태의 데이터를 출력하는 방법
- 화면으로 출력하는 방법에서 기본 출력 방법과 형식을 지정해서 출력하는 방법
- 키보드로 데이터를 입력 받는 방법
- 파일을 열어서 읽고 쓰는 방법
- 문자열로 된 텍스트 파일의 경우 줄 단위로 처리하는 방법
- 입출력 방법은 원하는 형식으로 화면이나 파일로 결과를 출력하거나 데이터가 있는 파일을 읽고 처리할 때 이용