

파이썬 프로그래밍

12주차 강의자료

나재호

CHAPTER
12

데이터 파일 입출력

01 파일이란?

02 데이터 파일의 두 가지 구성 방법

03 파일 열고 닫기

04 텍스트 파일에 쓰고 읽기

05 이진 파일에 쓰고 읽기

이 단원을 마치고 나면

- 데이터 파일을 입출력해야 하는 필요성에 대해 이해 한다.
- 텍스트 파일과 이진 파일의 차이점과 그 특징에 대해 설명할 수 있다.
- 지정된 모드로 파일을 열고 안전하게 닫을 수 있다.
- 텍스트 파일에 임의의 데이터를 쓰고, 읽어 들일 수 있다.
- 이진 파일에 임의의 데이터를 쓰고, 읽어 들일 수 있다.



01 | 파일이란

정보를 저장하고 불러오기

- ◆ 컴퓨터를 통해 처리한 데이터는 저장장치에 반영구적으로 저장
 - 추후 그 데이터가 필요할 때 다시 불러들여 사용
 - 예: 워드프로세서로 작성한 문서, 소스 파일 등



그림 12.1 저장된 정보의 사용

CPU와 협업하는 기억장치

- ◆ CPU: 각종 연산을 수행하고 각 부분 제어
- ◆ 기억장치: 처리될 명령과 데이터 저장
 - 주기억장치 (main memory) + 보조기억장치

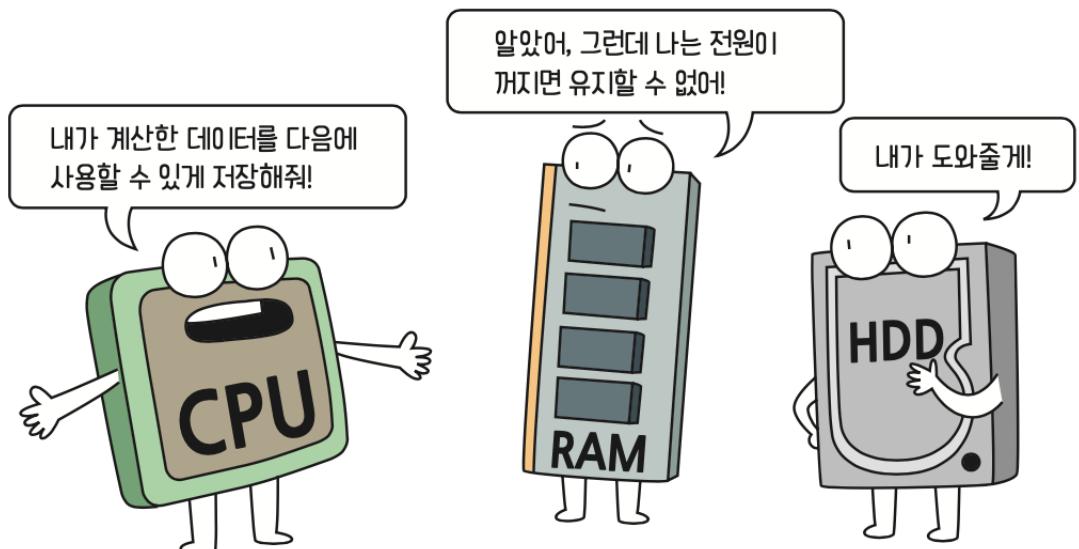


그림 12.2 다양한 저장장치

주기억장치의 역할

- ◆ 현재 실행중인 프로그램의 명령어 코드와 데이터 저장
 - CPU와 빈번하게 상호작용함
 - 빠르게 접근될 수 있어야 함
 - 전기적 신호를 기반으로 정보를 저장하는 반도체 메모리인 RAM으로 구성
 - 전원이 차단되면 저장내용을 유지하지 못함
 - 작업시 필요한 도구와 재료가 놓이는 작업실에 비유

보조기억장치의 역할

- ◆ 프로그램/시스템이 종료된 후에도 프로그램과 데이터를 **반영구적으로 저장**
 - 보조저장장치라고도 함
 - 추후 사용할 도구와 재료를 보관하는 창고에 비유



(1) 하드 디스크



(2) 블루레이 디스크



(3) SSD

그림 12.3 다양한 보조기억장치

주기억장치 vs. 보조기억장치

- ◆ 보조기억장치에 보관된 프로그램과 데이터가 CPU에 의해 실행되기 위해서는 반드시 주기억장치에 탑재되어 펼쳐 놓아져야 함



그림 12.4 주기억장치와 보조기억장치

파일(file)

- ◆ 운영체제의 관리 하에 보조저장장치에 저장된 연관된 정보들의 묶음
 - 파일의 내용은 그저 바이트들의 나열
 - 운영체제에 따라 서로 다른 방식으로 유지/관리됨
 - 역할에 따라 일반 파일과 디렉터리로 구분

디렉터리(directory)

- ◆ 연관된 파일들을 좀 더 효과적으로 관리하기 위해 파일들의 목록을 갖고 있는 특수 파일
 - 계층적으로 구성

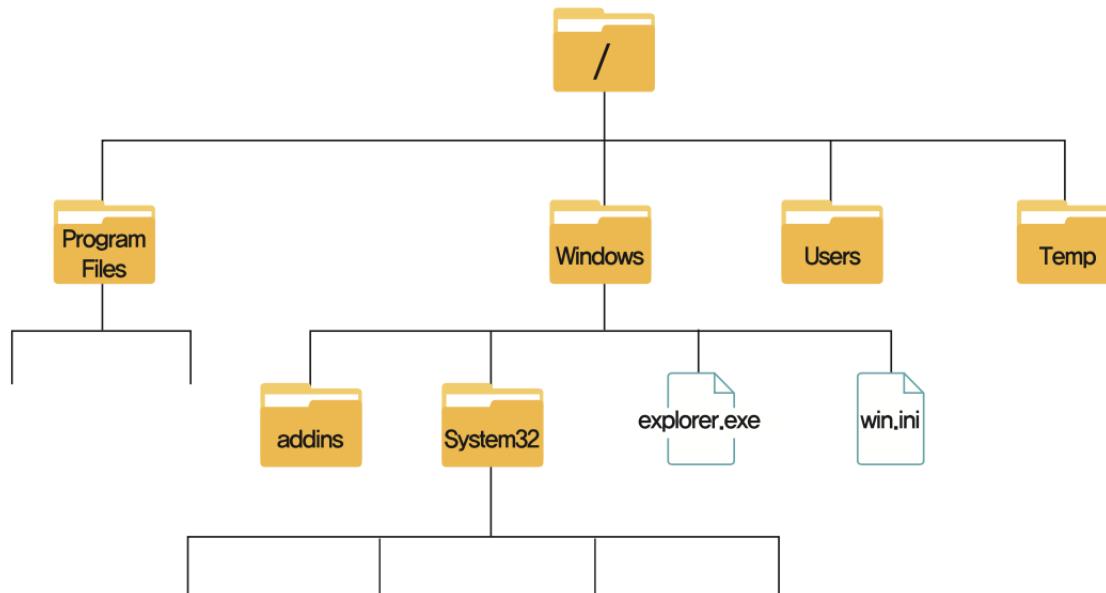


그림 12.5 계층적으로 구성된 디렉터리와 파일

일반 파일의 종류

◆ 실행 파일

- 운영체제에 의해 직접 실행될 수 있는 명령어들의 집합으로 구성된 파일
 - 운영체제 종속적 (예) .exe 파일은 윈도우에서 실행 가능)
 - 기계어 코드로 변환된 이진 파일의 형태를 가짐
- 보통 프로그램(앱) 설치 과정을 통해 보조저장장치에 저장됨
 - 시스템 시작 시 혹은 사용자가 원하는 시점에 운영체제의 도움을 받아 실행됨

일반 파일의 종류

◆ 데이터 파일

- 특정 프로그램에서 사용되는 각종 정보를 저장한 파일
 - 각 응용 프로그램에 종속적
 - 때로는 호환성을 위해 표준화된 형식을 사용하기도
- 보통 파일 입출력의 대상이 되는 파일



그림 12.6 데이터 파일의 역할

데이터 파일의 내용

- ◆ 프로그램 실행 중에 프로그램 내부적으로 혹은 사용자에 의해 생산된 정보
- ◆ 프로그램 실행을 위해 참조되는 정보



그림 12.7 실행 중에 참조되어지는 다양한 데이터 파일



02 | 데이터 파일의 두 가지 구성 방법

데이터 파일의 입출력

- ◆ 각 응용 프로그램은 데이터 파일에 어떤 내용을, 어떤 형식으로 저장할지 결정
 - 저장된 형식 그대로 다시 읽어 들여 사용하게 됨
 - 따라서 데이터 파일 내의 바이트들을 어떻게 해석하는지는 전적으로 프로그래머의 책임
- ◆ 데이터 파일의 두 형식
 - 이진 데이터 파일
 - 텍스트 파일

이진(binary) 데이터 파일

- ◆ 실행 중 메모리에 저장되는 데이터의 이진 형식 그대로 파일에 저장

- 특별한 **변환 없이 그대로 저장**하기에 효율적
 - 저장 형식을 이해할 수 있는 프로그램에 의해서만 데이터 판독 가능
 - 부적절한 접근을 차단하는 효과
 - 응용 프로그램 간 호환성을 위해 표준화된 형식을 사용하기도 함 (문서, 이미지, 동영상 등)
- 대부분의 응용 프로그램은 이 방법 이용

이진 데이터 파일의 예

- ◆ 헥스(hex) 에디터로 그 내용을 볼 수 있으나
바이트의 나열 이상의 의미를 찾기는 어려움

```
1 00000000 ef bf bd 50 4e 47 0d 0a 1a 0a 00 00 00 00 0d 49 48
2 00000010 44 52 00 00 01 ef bf bd 00 00 01 2e 08 06 00 00
3 00000020 00 ef bf bd 18 ef bf bd ef bf bd 00 00 0a ef bf
4 00000030 bd 69 43 43 50 49 43 43 20 50 72 6f 66 69 6c 65
5 00000040 00 00 48 ef bf bd ef bf bd ef bf bd 07 54 53 ef
6 00000050 bf bd 1a ef bf bd ef bf bd de 9b 5e 68 01 04 ef
7 00000060 bf bd ef bf bd de 91 4e 00 29 ef bf bd ef bf bd
8 00000070 2e 1d 44 25 24 ef bf bd ef bf bd 12 63 42 50 11
9 00000080 3b ef bf bd 2b ef bf bd 16 54 44 ef bf bd ef bf
...

```

(1) 헥스 에디터로 본 이미지 파일 내용(일부)



(2) 이미지 뷰어 표시 내용

그림 12.8 이진 데이터 파일의 예: .png 이미지 파일

텍스트 파일(text file)

- ◆ 텍스트 에디터를 통해 사람이 손쉽게 읽을 수 있도록 모든 데이터를 문자열로 변환해 저장
 - 보통 텍스트 에디터로 그 내용을 보면 연속적인 줄(line)들로 구성
 - 문자 코드 값들로 구성
 - 최근에는 가변 길이(1-4 bytes) 유니코드인 UTF-8 방식 선호
 - 이진 파일에 비해 입출력 효율성이 떨어지고, 저장공간도 많이 차지하며, 데이터 훼손 가능성도 높음
 - 예: 소스 파일, 메모장으로 작성되어진 파일 등

텍스트 파일의 예

- ◆ 텍스트 편집기만 있으면 누구든 그 내용을 보고 수정할 수 있음
 - 보통 행 단위의 문자열로 구분

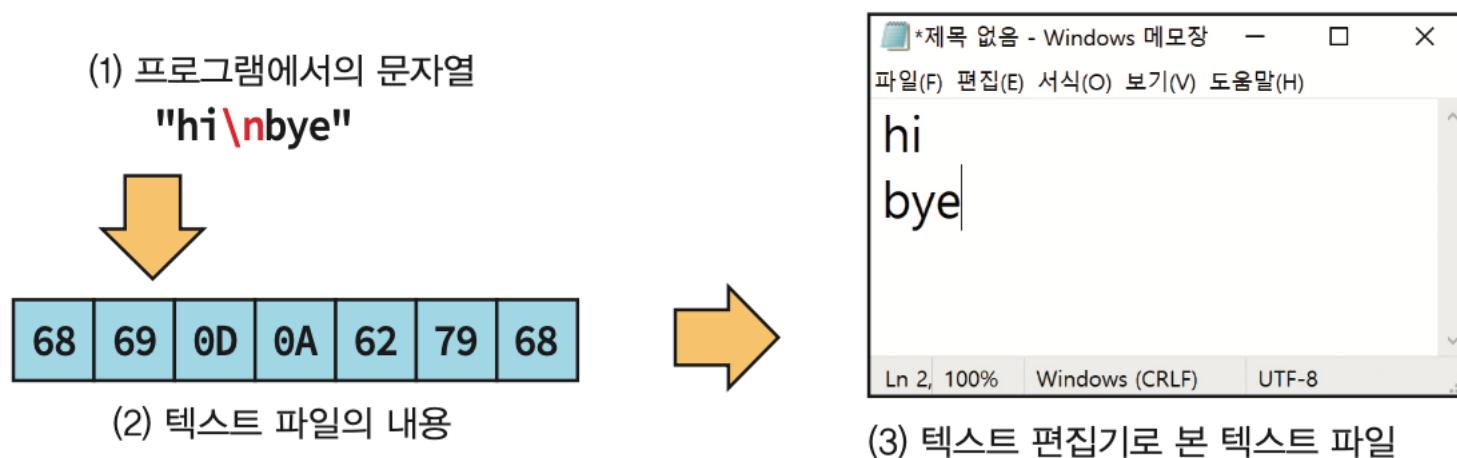
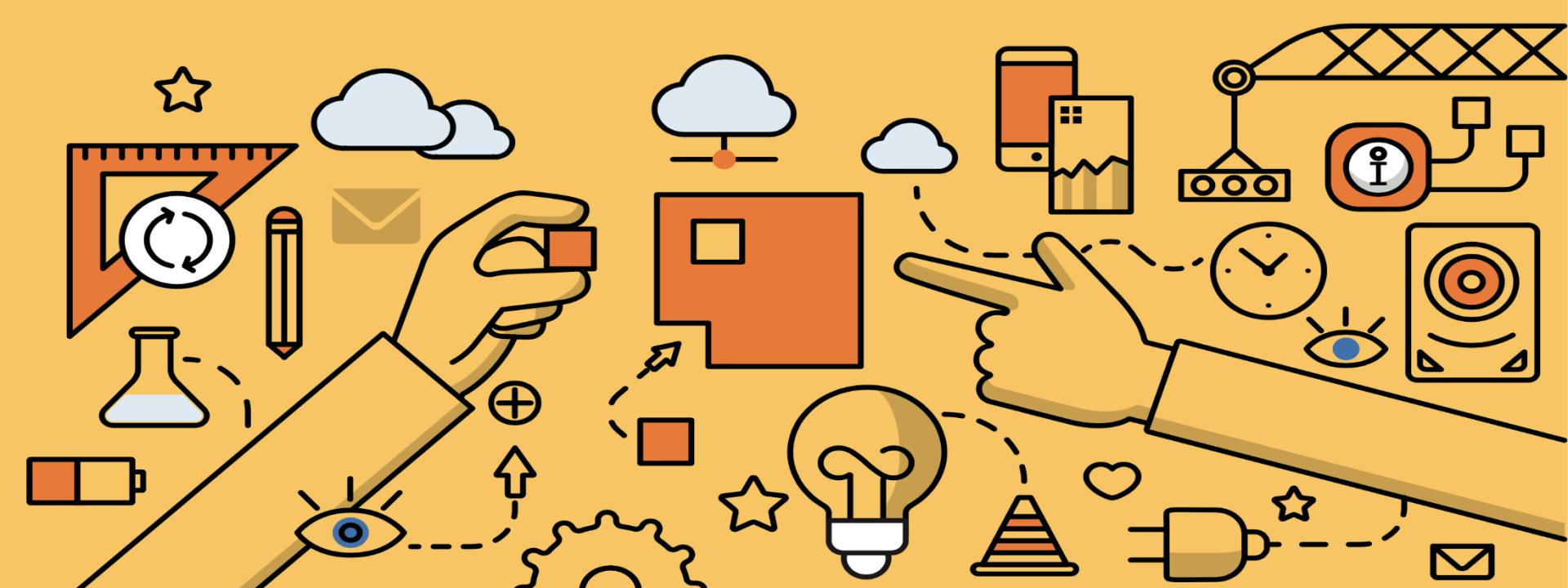


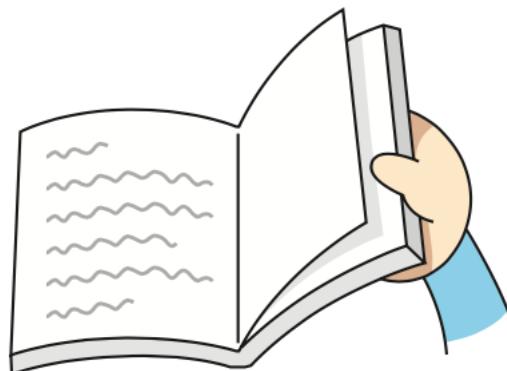
그림 12.9 텍스트 파일



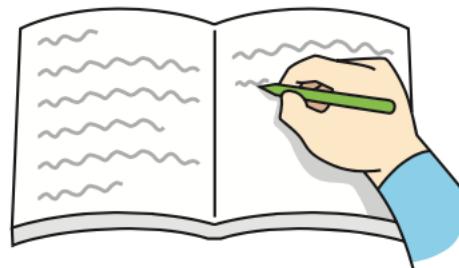
03 | 파일 열고 닫기

글을 쓰거나 읽기 위해서는

- ◆ 일상 생활 속에서 일기를 쓰기 위해서는,
 1. 책꽂이에서 일기장을 찾아 일기를 작성할 쪽을 펼침
 2. 일기를 작성
 3. 일기장을 덮어 책꽂이의 제자리에 끌음



(1) 일기장 찾아 펴기



(2) 일기 쓰기



(3) 일기장 덮고 정리하기

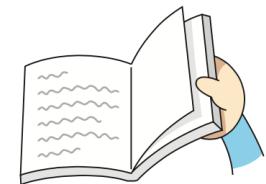
그림 12.10 일기쓰기 과정

파일에 쓰거나 읽기 위해서는

- ◆ 일상에서와 마찬가지로 준비 작업과 마무리 작업이 동반됨

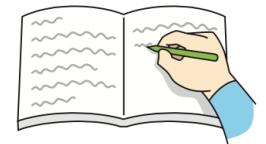
- 파일 열기(file open)

- 프로그램에서 입출력 하고자 하는 데이터 파일에 연결하는 과정



- 파일 조작

- 열린 파일에 데이터를 입출력하는 과정



- 파일 닫기(file close)

- 프로그램에서 입출력 조작을 마친 파일의 사용을 안전하게 마무리하기 위해 연결을 끊는 과정



스트림(Stream)

- ◆ 일반적으로 데이터의 흐름, 혹은 데이터 흐름을 형성해 주는 통로를 가리킴
 - 연속된 바이트의 흐름



- ◆ 종류
 - 출력 스트림(Output stream)
 - 데이터를 쓰는 대상이 되는 스트림
 - 입력 스트림(Input stream)
 - 데이터를 읽어들이는 대상이 되는 스트림

프로그램 입출력(I/O)

◆ 프로그램이 외부와 상호작용

- 스트림 기반

- 입출력 장치의 종류에 관계없이 동일한 방법으로 입출력 수행 가능(장치 독립적)
 - 예: 표준 입출력 라이브러리

- 버퍼링

- CPU가 다양한 I/O를 일관된 방법으로 접근할 수 있게 함
 - CPU와 I/O 장치간의 속도 차이 보완



파일 열기

- ◆ 대상 파일에 대해 접근할 수 있는 객체 반환
 - 이후 이 객체를 이용해 손쉽고 효율적으로 파일 입출력 처리 수행

```
파일객체이름 = open('대상파일이름', mode='열기모드')
```

- 첫 번째 인수
 - 입출력 대상이 되는 파일의 이름(경로명)을 나타내는 문자열
- mode 키워드 인수
 - 어떤 용도로 파일을 열 것인지를 지정하는 문자열

대상 파일은 어디에?

- ◆ `open()`의 첫 번째 인수로 '파일 이름'만 지정하면 현재 경로에서 대상 파일을 찾음
 - 현재 경로란?
 - 대화식 모드에서 실행시: 파이썬이 설치된 디렉터리 경로
 - C:\Users\사용자계정명\AppData\Local\Programs\Python\Python310
 - 스크립트 모드에서 실행시: 현재 소스 파일이 위치한 경로

경로명이 포함된 파일 이름

◆ 특정 경로 상의 파일을 입출력하기

```
파일객체이름 = open('디렉터리이름/대상파일이름', mode='열기모드')
```

- `open()`의 첫 번째 인수에 경로명을 포함한 파일이름을 문자열로 지정
 - 경로 구분자는 / 기호 사용
 - 절대경로
 - 시스템의 루트 디렉터리를 기반으로 한 경로명
 - 경로명 맨 앞에 / 로 시작
 - 상대경로
 - 현재 경로를 기반으로 한 상대적인 경로명

파일 열기 모드

◆ 입출력 방식과 접근 동작을 문자열로 기술

표 12.1 파일 열기 모드 기본값

	모드값	의미		동작
입출력 방식	't'	text		텍스트 파일 입출력 (기본값)
	'b'	binary		이진 파일 입출력
접근 동작	'r'	read		읽기 모드 (기본값)
	'w'	write	overwrite	덮어쓰기 모드
	'a'		append	추가 쓰기 모드
	'x'	exclusive creation		파일 존재 시 에러, 파일 부재 시 생성

표 12.2 일반적인 파일 열기 모드

모드값	동작	동일 표현
'r'	텍스트 파일 읽기 모드 (기본값)	'rt' 혹은 모드값 지정 생략
'w'	텍스트 파일 덮어쓰기 모드	'wt'
'a'	텍스트 파일 추가 쓰기 모드	'at'
'rb'	이진 파일 읽기 모드	'b'
'wb'	이진 파일 덮어쓰기 모드	

파일 닫기

◆ 파일 입출력 처리가 끝난 파일은 바로 닫자

파일객체이름.close()

- 안전한 파일 사용 위해 파일 입출력 작업을 마침
 - 다른 모드로 파일을 열기 위해
 - 저장을 마무리 하기 위해
- 필요시 적절한 모드로 다시 열어 사용

읽기 모드로 파일 열고 닫기

- ◆ 존재하지 않는 파일을 읽기 모드로 열기
 - FileNotFoundError 발생

코드 12-1

읽기 모드로 파일 열기

```
01 file = open('example.txt', 'r')      # 읽기모드로 텍스트파일 열기  
02 file.close()  
03 print('파일 열고 닫기 완료')
```



실행 결과

```
Traceback (most recent call last):  
  File "C:/pyWork/fileopen_test.py", line 1, in <module>  
    file = open('example.txt', 'r')  
FileNotFoundError: [Errno 2] No such file or directory:  
'example.txt'
```

쓰기 모드로 파일 열고 닫기(1/2)

- ◆ 존재하지 않는 파일을 쓰기 모드로 열기
 - 오류 없이 대상 파일을 새로 생성

#

코드 12-2

쓰기 모드로 파일 열기

```
01 file = open('example.txt', 'w')      # 쓰기모드로 텍스트파일 열기
02 file.close()
03 print('파일 열고 닫기 완료')
```



실행 결과

파일 열고 닫기 완료

쓰기 모드로 파일 열고 닫기(2/2)

- ◆ 쓰기 모드로 여는 경우에도 대상 경로명이 유효하지 않으면 FileNotFoundError 발생

코드 12-3

쓰기 모드로 파일 열기

```
01 file = open('data/example.txt', 'w')      # 텍스트 파일 쓰기 모드  
02 file.close()  
03 print('파일 열고 닫기 완료')
```



실행 결과

Traceback (most recent call last):

```
File "C:\pyWork\fileopen_test.py", line 1, in <module>  
    file = open('data/example.txt', 'w')  
FileNotFoundError: [Errno 2] No such file or directory: 'data/  
example.txt'
```

파일/디렉터리의 존재 여부 검사

- ◆ 파일 열기 작업 후 예외 발생시 예외처리
 - 이 방법이 좀 더 일반적이나
 - 이 강의에서는 예외처리를 다루지 않음
- ◆ 파일 접근 전에 먼저 대상 경로 검사

```
import os  
os.path.exists('검사할파일경로명')      # True / False의 반환값
```

파일/디렉터리의 존재 여부 검사 예

◆ 유효한 경우에만 파일을 열어 사용

#

코드 12-4

파일 존재 여부 확인

```
01 import os
02
03 filename = 'example.txt'
04 if os.path.exists(filename):      # 파일이 존재하는지 검사
05     file = open(filename, 'r')      # 텍스트 파일읽기모드로 열기
06     file.close()                  # 파일 닫기
07     print('파일 열고 닫기 완료')
08 else:
09     print(f'ERROR: {filename}이 존재하지 않습니다! ')
10
11 print('끝')
```

참고: 디렉터리 만들기

```
import os  
if not(os.path.isdir('data')):  
    os.makedirs('data')
```

- `isdir()`
 - 디렉터리의 존재 여부를 반환
- `makedirs()`
 - 현재 경로상에 인수로 지정한 이름의 서브 디렉터리 생성

with 키워드로 안전하게 파일 닫기

- ◆ close() 메서드 호출 없이도 with 블록이 종료될 때 언제나 파일 닫기 처리가 자동으로 수행됨

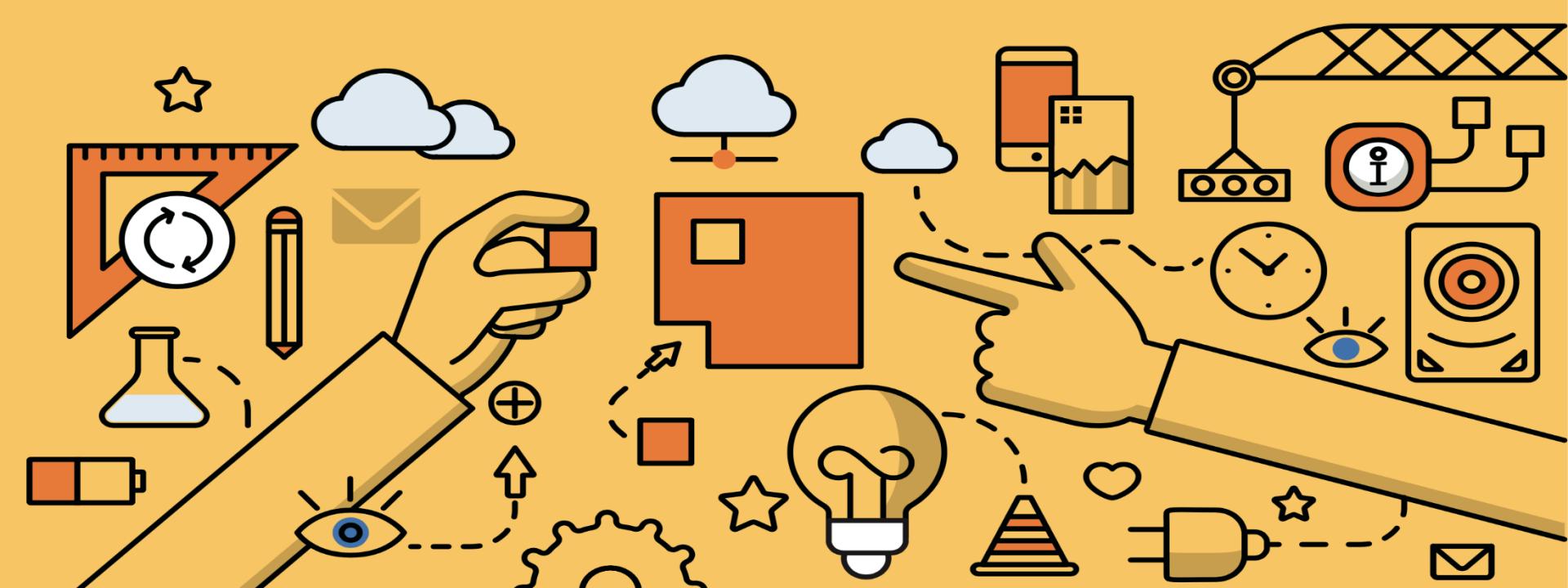
```
with open('파일이름', '열기모드') as 파일객체이름:  
    # 파일객체를 가지고 파일 입출력 처리
```

- 오류가 발생하는 상황에서도 닫는 처리 자동 수행

#

코드 12-5 with 키워드로 쓰기 모드의 파일 열기

```
01 with open('data/example.txt', 'w') as file:  
02     pass      # file에 대해 파일 입출력 처리 문장으로 대체  
03  
04 print('파일 열고 닫기 완료')
```



04

텍스트 파일에 쓰고 읽기

텍스트 파일에 쓰기

◆ 문자열로 텍스트 파일에 저장하기

파일객체이름.write(문자열)

- 인수로 파일에 저장할 문자열 전달
 - 텍스트 편집기로 보기 좋게 구성하기 위해 적절한 곳에 줄바꿈 처리를 위한 개행 문자를 추가로 저장 필요
 - 저장할 데이터에 개행 문자가 포함되어 있지 않은 경우
- 문자열이 아닌 데이터는 문자열 형식화를 통해 하나의 문자열로 변환해 저장

텍스트 파일에 쓰기 예

◆ 두 사람의 번호와 이름의 저장 예

#

코드 12-6

텍스트 파일에 정보 저장

```
01 num = 2
02 name = '홍길동'
03
04 with open('data/example.txt', 'w') as file:
05     file.write('1 이찬수\n')      # 하나의 문자열을 저장
06     file.write(f'{num} {name}\n') <-----
07
08 print('끝')
```

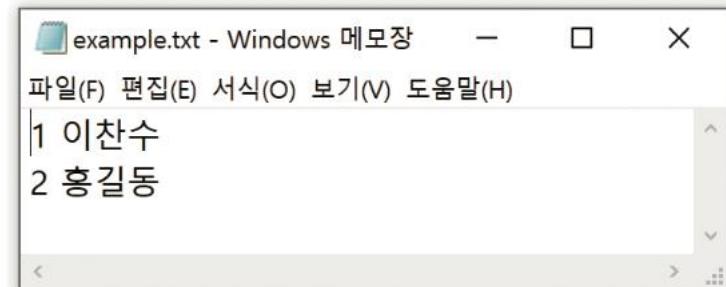


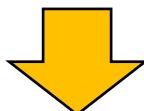
그림 12.11 메모장으로 열기

변수 num과 name을 문자열 형식화로
하나의 문자열로 표현

저장되는 문자열의 코드화 종류

◆ 기본적으로 ASCII 방식으로 영문자 위주의 인코딩 방법 사용

- 기타 다양한 나라의 문자는 MS-윈도우 자체적으로 각 문자셋에 대한 코드 페이지를 별도로 제공
 - 단, 특정 운영체제에서 제공하는 비 표준적인 방법
 - 한글의 경우 **EUC-KR** 방식의 인코딩 사용



- 최근에는 UTF-8 방식의 유니코드 사용 추세
 - ASCII 방식과 호환
 - 다양한 문자를 표현하는 표준적인 방법

MS-윈도우의 메모장에서 인코딩 방식 지정

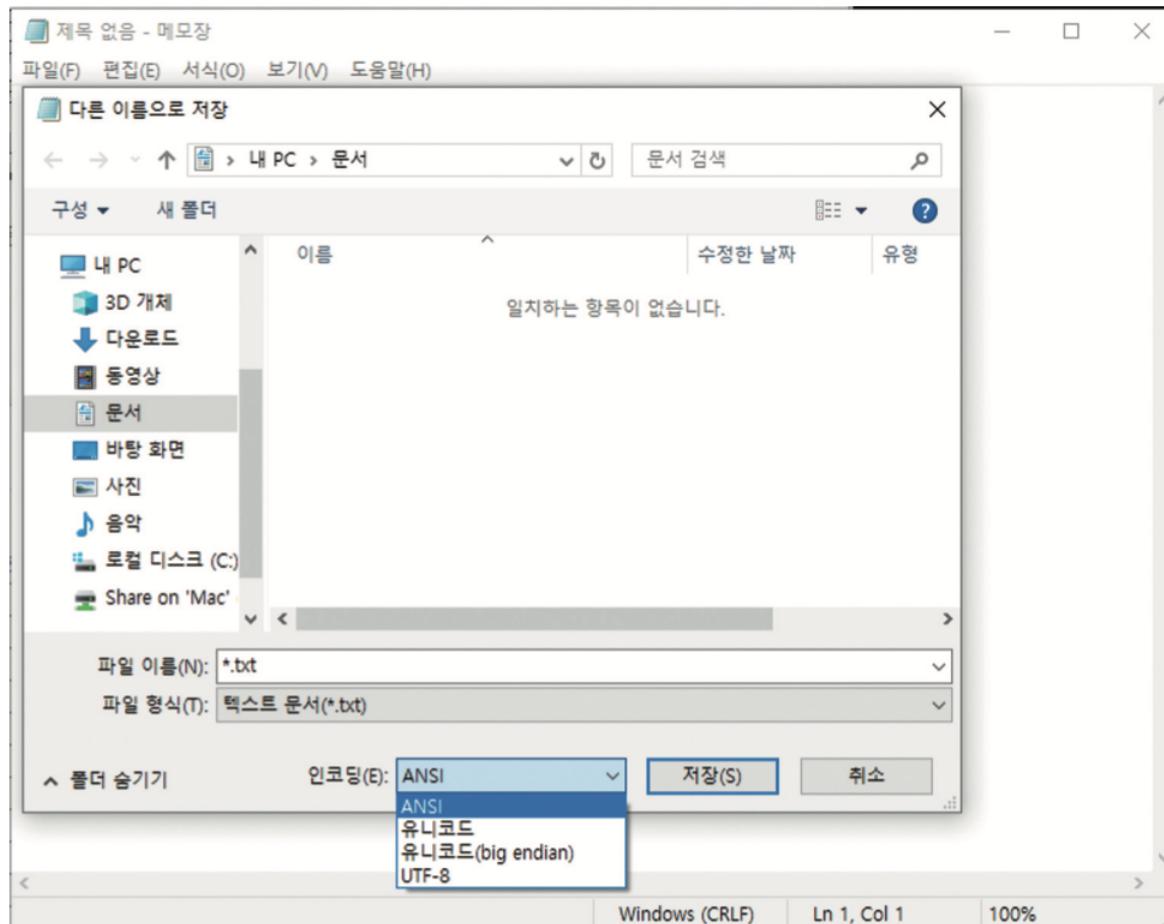


그림 12.12 메모장의 다양한 인코딩 방법

저장되는 문자열의 문자셋 지정하기

◆ encoding 키워드 인수로 문자셋 문자열 지정

```
open('경로명/대상파일명', mode='열기모드', encoding='문자셋')
```

- 'utf8' : UTF-8 방식의 인코딩 사용
 - 권장
- 'cp949' : EUC-KR 방식의 인코딩 사용
- 같은 운영체제에서만 사용한다면 문자셋 지정은 생략해도 무관함

참고: 개행문자의 저장

- ◆ 각 운영체제마다 ' Wn '를 저장하는 방식이 다름
 - 'hi $\text{Wn}bye$ '를 저장한 경우

68	69	0D	0A	62	79	65
----	----	----	----	----	----	----

윈도우에서 저장한 경우
(CR LF)

68	69	0A	62	79	65
----	----	----	----	----	----

macOS에서 저장한 경우
(LF)

- ◆ 텍스트 파일에 개행 문자 저장시 플랫폼 의존적인 줄종료로 변환해 저장됨
 - 파일로부터 읽어들일 때에도 마찬가지
 - 때때로 텍스트 편집기에 따라 다른 플랫폼에서 작성한 텍스트 문서의 줄바꿈 처리가 제대로 표시되지 않을 수 있음

텍스트 파일로부터 한 번에 읽기

◆ 파일의 모든 내용을 하나의 문자열로 반환

```
파일객체이름.read()      # 텍스트 파일로부터 읽어들인 내용을 문자열로 반환
```

#

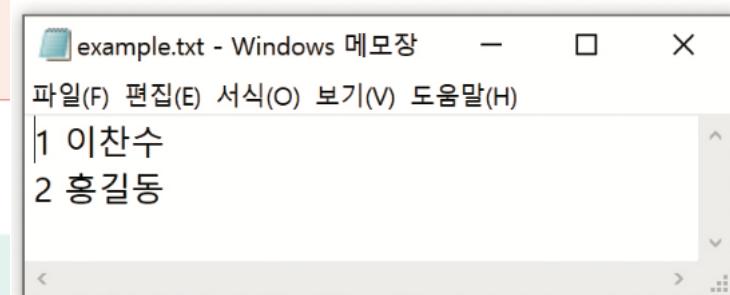
코드 12-7

텍스트 파일의 내용을 한 번에 읽어들여 화면에 출력

```
01 with open('data/example.txt', 'r') as file:  
02     all = file.read()      # 전체 내용을 하나의 문자열로 반환  
03  
04 print(all)
```

▶ 실행 결과

```
1 이찬수  
2 홍길동
```



인코딩 형식이 다른 경우

- ◆ EUC-KR로 저장된 파일을 UTF-8로 읽은 경우
 - UnicodeDecodeError 발생



실행 결과

```
Traceback (most recent call last):
  File "C:\pyWork\fileopen_err.py", line 2, in <module>
    all = file.read()
  File "C:\Users\ChanSu\AppData\Local\Programs\Python\Python310\lib\codecs.py", line 322, in decode
      (result, consumed) = self._buffer_decode(data, self.
errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb1 in
position 22: invalid start byte
```

인코딩 형식이 다른 경우

- ◆ UTF-8로 저장된 파일을 EUC-KR로 읽은 경우
 - UnicodeDecodeError 발생



실행 결과

```
Traceback (most recent call last):
  File "C:\pyWork\fileopen_err.py", line 2, in <module>
    all = file.read()
UnicodeDecodeError: 'cp949' codec can't decode byte 0xed in
position 20: illegal multibyte sequence
```

텍스트 파일로부터 한 줄씩 구분해 읽기

◆ 개행문자까지 한 줄씩 구분해 모두 읽어옴

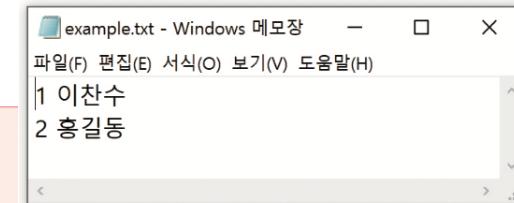
```
파일객체이름.readlines()      # 텍스트 파일의 각 행을 리스트로 반환
```

- 읽어들인 각 행을 리스트로 만들어 반환
 - 한 번의 호출로 모든 행의 내용을 읽어 반환

코드 12-8

텍스트 파일의 내용 읽어들여 리스트에 저장

```
01 lines = []
02 with open('data/example.txt', 'r') as file:
03     lines = file.readlines()      # 전체를 하나의 리스트로 반환
04
05 print(lines)
```



▶ 실행 결과

```
['1 이찬수\n', '2 홍길동\n']
```

- 각 행 요소에 개행문자가 포함됨
 - → 제거 필요

텍스트 파일로부터 한 줄만 읽기

◆ 파일에서 개행문자까지 한 줄만 읽어옴

```
파일객체이름.readline()      # 텍스트 파일로부터 한 줄을 읽어 문자열로 반환
```

- 현재 파일 위치에서 처음 만나는 개행문자까지만 읽어 문자열로 반환
 - 전체 파일을 읽기 위해 반복 호출 필요
 - 호출 시마다 자동으로 다음 행에 대해 처리
- 파일 끝에 도착하면 빈 문자열 " 반환

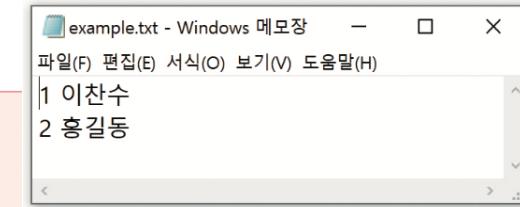
텍스트 파일로부터 한 줄만 읽기 예1

◆ 한 줄만 읽어오는 과정을 while 문으로 반복

코드 12-9

텍스트 파일의 내용 읽어들여 리스트에 저장

```
01 lines = []
02 with open('data/example.txt', 'r') as file:
03     while True:
04         line = file.readline()      # 한 줄을 읽어 반환
05
06         if line == '':
07             break
08
09         lines.append(line.strip())
10
11 print(lines)
```



실행 결과

```
['1 이찬수', '2 홍길동']
```

문자열 line 앞뒤의 공백문자를 제거해서
리스트 lines에 추가

텍스트 파일로부터 한 줄만 읽기 예2

- ◆ 한 줄만 읽어오는 과정을 for 문으로 반복
 - 반복 범위에 파일 객체를 직접 지정

코드 12-10

텍스트 파일의 내용 읽어들여 리스트에 저장

```
01 lines = []
02 with open('data/example.txt', 'r') as file:
03     for line in file:<-----
04         lines.append(line.strip('\n'))
05
06 print(lines)
```

파일 객체 file로부터 한 행씩 문자열로
읽어들여 line에 저장



실행 결과

```
['1 이찬수', '2 홍길동']
```



05

이진 파일에 쓰고 읽기

텍스트 파일 입출력의 한계

- ◆ 행 단위로 읽어 들인 문자열에서 원래 데이터인 번호/이름을 얻기 위해선 추가의 분리 작업 필요
 - 한 행 내에 공백문자와 같은 구분자를 이용해 분리



(1) 텍스트 파일에 저장하기



(2) 텍스트 파일로부터 불러오기

그림 12.13 텍스트 파일에 데이터 입출력

- ◆ 텍스트 편집기로 그 저장 내용을 쉽게 보고 편집 가능하나 이진 파일 입출력에 비해 비효율적

이진 파일(Binary File)

- ◆ 이진 데이터값 **그대로** 저장
 - 텍스트 에디터를 통해 의미 있는 내용을 읽을 수 없음
- ◆ 이 형식을 이해할 수 있는 프로그램에 의해서만 데이터 판독이 가능
 - 보통 이 형식의 파일을 저장한 프로그램에 의해 사용
- ◆ 텍스트 파일에 비해 데이터를 좀 더 **효율적으로** 저장 가능

pickle 모듈

- ◆ 기본 자료형의 데이터나 객체의 직렬화(serialization)/역직렬화(deserialization) 방법 제공
 - 피클링(pickling)
 - 데이터를 바이트 나열로 변환하는 직렬화
 - 언피클링(unpickling)
 - 바이트 나열로부터 데이터를 복원하는 역직렬화
- ◆ 바이트 나열로 변환된 데이터를 데이터베이스나 파일에 저장하거나 네트워크 전송에 사용

pickle 모듈을 이용한 입출력

- ◆ 피클링하여 이진 파일에 저장하기
 - 인수로 지정한 데이터를 바이트 나열로 변환해 저장

```
import pickle  
pickle.dump(저장할_데이터, 이진쓰기모드로_열린_파일객체)
```

- ◆ 이진 파일로부터 언피클링하여 복원하기
 - 저장 순서대로 읽어들여 복원 후 반환

```
import pickle  
pickle.load(이진읽기모드로_열린_파일객체)      # 읽은 데이터 반환
```

이진 파일 입출력 예(1/2)

코드 12-11

다양한 데이터를 피클링하여 이진 파일로 저장

```
01 import pickle
02
03 filepath = 'data/example.bin'      # 데이터파일에 대한 파일경로명
04
05 # 사용자정의함수부
06 def save_data(num, name, height, scores):  # 피클링하여 저장
07     with open(filepath, 'wb') as file:
08         pickle.dump(num, file)
09         pickle.dump(name, file)
10         pickle.dump(height, file)
11         pickle.dump(scores, file)
12
13 def load_data():      # 언피클링해 반환
14     with open(filepath, 'rb') as file:
15         num = pickle.load(file)
16         name = pickle.load(file)
17         height = pickle.load(file)
18         scores = pickle.load(file)
19
20     return num, name, height, scores      # 튜플로 반환
```

네 변수의 값을 차례로 저장

저장된 순서대로
네 부분으로
나눠 읽기

이진 파일 입출력 예(2/2)

```
21  
22 # 주 프로그램부  
23 num, name, height = 123, '홍길동', 180.5  
24 scores = {'mid': 90, 'fin': 95, 'rep': 10, 'att': 10}  
25  
26 # 저장하기  
27 save_data(num, name, height, scores)  
28  
29 # 불러오기  
30 r_num, r_name, r_height, r_scores = load_data()  
31  
32 # 불러온 값 출력  
33 print(r_num)  
34 print(r_name)  
35 print(r_height)  
36 print(r_scores)
```



실행 결과

123

홍길동

180.5

{'mid': 90, 'fin': 95, 'rep': 10, 'att': 10}

사용자 정의 클래스 객체의 입출력

코드 12-12

사용자 정의 클래스의 객체를 피클링하여 이진 파일로 저장

```
01 import pickle
02
03 filepath = 'circle.bin'      # 데이터 파일에 대한 파일 경로명
04
05 class Circle :
06     __PI = 3.14159265
07
08     def getPI(self) :
09         return __PI
10
11     def __init__(self, radius) :
12         self.__radius = radius
13
14     def getCircumference(self) :
15         result = 2 * Circle.__PI * self.__radius
16         return result
17
18     def getArea(self) :
19         result = Circle.__PI * self.__radius ** 2
20         return result
```

28	# 주 프로그램부
29	with open(filepath, 'wb') as file:
30	c1 = Circle(10)
31	pickle.dump(c1, file) # 파일에 객체를 저장
32	
33	with open(filepath, 'rb') as file:
34	c2 = pickle.load(file) # 파일에서 객체를 가져옴
35	print(f'원의 넓이는 {c2.getArea()}')

- 객체 단위로 입출력 가능
- 피클링 시 해당 클래스의 정의가 없으면 `AttributeError` 예외 발생



실행 결과

원의 넓이는 314.159265

연습문제 12.1

- ◆ 텍스트 파일을 입력받아 그 파일의 내용을 행번호와 함께 출력하는 프로그램 작성

- 아래 내용을 가지는 텍스트 파일을 작성

안녕하세요, 반갑습니다.

이 파일은 테스트 파일 저작을 위해 작성된 텍스트 문서입니다.

조금 낯설더라도 포기하지 마세요.

- 인코딩 문제가 있을 경우 open()에서 인코딩 형식 직접 지정
 - 실행 결과

파일명: **readme.txt**

1: 안녕하세요, 반갑습니다.

2: 이 파일은 테스트 파일 저작을 위해 작성된 텍스트 문서입니다.

3: 조금 낯설더라도 포기하지 마세요.

연습문제 12.3

◆ 연습문제 10.5 장바구니 프로그램에 save/load 기능 추가

- 프로그램 종료시 장바구니 정보를 shoppingbag.txt에 저장
- 프로그램 시작시 지정된 경로에 위 파일이 존재하면, 그 내용을 장바구니에 추가
- 실행 결과(첫 실행시) (두 번째 실행시)

[구입]

상품명? 바나나

장바구니에 바나나가(이) 담겼습니다.

[파일 읽기]

>>> 장바구니 보기: ['바나나', '사과', "]

[구입]

상품명? 사과

장바구니에 사과가(이) 담겼습니다.

[구입]

상품명?

[구입]

상품명?

>>> 장바구니 보기: ['바나나', '사과']

연습문제 12.4

◆ 프로그램이 종료된 마지막 시각을 last.dat에 save/load

- 프로그램의 현재 시각은 아래 코드를 사용해 구함

```
from datetime import datetime
now = datetime.now()
hour = now.hour
minute = now.minute
```

- 연습문제 11.2에서 작성한 Time 클래스를 이용하여, 아래 세 개의 사용자 정의 함수 작성

- get_current_time() - 현재 시각을 return
- save_time(time) – last.dat에 현재 시각 저장 (텍스트 또는 이진파일 형식)
- load_time() – last.dat에서 읽어온 시각으로 Time 객체를 만들어 반환

- 실행 결과(첫 실행시)

안녕하세요, 처음 실행되었습니다.
지금은 **00:37** 입니다.

- (1분 후 실행시)

안녕하세요, 마지막으로 **00:37**에 실행되었습니다.
지금은 **00:38** 입니다.

개념 확인 과제(연습문제 12.2)

- ◆ 연습문제 10.3 학생 정보 프로그램에 save/load 기능 추가
 - 프로그램 종료시 학생의 점수들을 이진 파일 score.bin에 저장
 - 프로그램 시작시 지정된 경로에 위 파일이 존재하면, 이 파일 내의 점수들을 읽어 들여 화면에 출력 (또 입력받지는 않음)
 - 실행 결과(첫 실행시) (두번째 실행시)
[점수 입력]
#1? 30
#2? 5
#3? -1

[점수 출력]
개인점수: 30 5
평균: 17.5
- ◆ 제출 기한 및 방법
 - 다음 수업날 전까지 본인의 repository에 hw10.py를 업로드