파이썬 프로그래밍

재귀 함수

- 재귀 함수(Recursive Function)란 자기 자신을 다시 호출하는 함수를 의미
- 단순한 형태의 재귀 함수 예제
  - '재귀 함수를 호출합니다.'라는 문자열을 무한히 출력
  - 어느 정도 출력하다가 최대 재귀 깊이 초과 메시지가 출력

```
def recursive_function():
    print('재귀 함수를 호출합니다.')
    recursive_function()

recursive_function()
```

재귀 함수의 종료 조건

- 재귀 함수를 문제 풀이에서 사용할 때는 재귀 함수의 종료 조건을 반드시 명시
- 종료 조건을 제대로 명시하지 않으면 함수가 무한히 호출될 수 있다.
  - 종료 조건을 포함한 재귀 함수 예제

```
def recursive_function(i):
    # 100번째 호출을 했을 때 종료되도록 종료 조건 명시
    if i == 100:
        return
    print(i, '번째 재귀함수에서', i + 1, '번째 재귀함수를 호출합니다.')
    recursive_function(i + 1)
    print(i, '번째 재귀함수를 종료합니다.')

recursive_function(1)
```

팩토리얼 구현 예제

- $n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$
- 수학적으로 $0!$과 $1!$의 값은 1입니다.

```
# 반복적으로 구현한 n!
def factorial_iterative(n):
    result = 1
    # 1부터 n까지의 수를 차례대로 곱하기
    for i in range(1, n + 1):
        result *= i
    return result

# 재귀적으로 구현한 n!
def factorial_recursive(n):
    if n <= 1: # n이 1 이하인 경우 1을 반환
        return 1
    # n! = n * (n - 1)!를 그대로 코드로 작성하기
    return n * factorial_recursive(n - 1)
```

실행 결과

```
# 각각의 방식으로 구현한 n! 출력(n = 5)
print('반복적으로 구현:', factorial_iterative(5))
print('재귀적으로 구현:', factorial_recursive(5))
```

```
반복적으로 구현: 120
재귀적으로 구현: 120
```

최대공약수 계산 (유클리드 호제법) 실습

- 두 개의 자연수에 대한 최대공약수를 구하는 대표적인 알고리즘으로는 유클리드 호제법이 있습니다.
- 유클리드 호제법
  - 두 자연수 A, B에 대하여 (A > B) A를 B로 나눈 나머지를 R이라고 합시다.
  - 이때 A와 B의 최대공약수는 B와 R의 최대공약수와 같습니다.
- 유클리드 호제법의 아이디어를 그대로 재귀 함수로 작성할 수 있습니다.
  - 예시: GCD(192, 162)

단계	A	B
1	192	162
2	162	30
3	30	12
4	12	6

재귀 함수 사용의 유의 사항

- 재귀 함수를 잘 활용하면 복잡한 알고리즘을 간결하게 작성할 수 있습니다.
  - 단, 오히려 다른 사람이 이해하기 어려운 형태의 코드가 될 수도 있으므로 신중하게 사용해야 합니다.
- 모든 재귀 함수는 반복문을 이용하여 동일한 기능을 구현할 수 있습니다.
- 재귀 함수가 반복문보다 유리한 경우도 있고 불리한 경우도 있습니다.
- 컴퓨터가 함수를 연속적으로 호출하면 컴퓨터 메모리 내부의 스택 프레임에 쌓입니다.
  - 그래서 스택을 사용해야 할 때 구현상 스택 라이브러리 대신에 재귀 함수를 이용하는 경우가 많습니다.

모듈

- 함수 정의나 클래스 등 서로 관련이 있거나 비슷한 기능을 하는 파이썬 문장들을 담고 있는 파일을 모듈(module) 이라고 하며, 필요한 것들끼리 부품처럼 잘 만드는 것을 모듈화 (modularization) 라고 합니다.
- 난수를 생성하는 함수들이 정의된 random 모듈 덕분에 우리는 별도의 난수 생성 함수를 정의할 필요 없이 random 모듈을 import 하여 사용하기만 하면 되었습니다.
- 파이썬에는 이미 굉장히 많은 유용한 모듈들이 있는데 파이썬 개발을 하다 보면 새로운 모듈을 개발해야 할 필요도 있습니다.
- theater_module 모듈을 만들어 보시다. 주의할 점은 theater_module.py 파일과 이 모듈을 사용할 파일은 서로 같은 경로상에 있어야 한다는 것입니다.

모듈

theater_module.py

```
# 일반 가격
def price(people):
    print("{0}명 가격은 {1}원 입니다.".format(people, people * 10000))
# 조조 할인 가격
def price_morning(people):
    print("{0}명 조조 할인 가격은 {1}원 입니다.".format(people, people * 6000))

# 군인 할인 가격
def price_soldier(people):
    print("{0}명 군인 할인 가격은 {1}원 입니다. ".format(people, people * 4000))
```

모듈

- 모듈을 사용하는 방법에는 여러 가지가 있는데 가장 기본적인 import 를 먼저 보겠습니다. import 구문을 쓸 때는 파일명 theater_module.py 에서 확장자 .py 를 제외한 모듈 이름 theater_module 을 그대로 적어주면 됩니다. import 를 한 이후부터는 이 모듈에 정의한 함수를 사용할 수 있는데 모듈명 뒤에 점(.) 을 찍고 나서 함수 이름을 적습니다. 3개 함수를 각각 호출하며 전달값은 3, 4, 5 로 해보겠습니다.

```
import theater_module # theater_module 을 가져다가 사용
theater_module.price(3) # 3명이 영화 보러 갔을 때 가격
theater_module.price_morning(4) # 4명이 조조 영화 보러 갔을 때
theater_module.price_soldier(5) # 5명이 군인이 영화 보러 갔을 때
```

모듈

- 그런데 theater_module 이라는 이름이 길어서 불편하기도 합니다.
- 이럴 때는 as 를 이용해서 모듈에 별명을 붙여줄 수 있습니다.

```
import theater_module as mv
# theater_module 을 새로운 별명인 mv 로 사용
mv.price(3)
mv.price_morning(4)
mv.price_soldier(5)
```

모듈

```
from theater_module import * # theater_module 내에서 모든 것을 가져다가 사용
price(3) # theater_module. 필요 없음
price_morning(4)
price_soldier(5)
from theater_module import price, price_morning # 모듈에서 일부만 가져다가 사용
price(5) # 이번에는 5명
price_morning(6)
price_soldier(7) # import 하지 않았으므로 사용 불가
from theater_module import price_soldier as price # price_soldier 를 새로운 별명인 price 로 사용
price(5) # price_soldier() 를 호출
```

sys 모듈

sys.path는 인터프리터가 모듈을 검색할 경로를 담은 문자열 리스트다.

PYTHONPATH 환경변수 또는 기본 path로 초기화된다.

```
import sys
sys.path
# 임시로 path를 추가할 수 있다.
sys.path.append('모듈_디렉터리_경로')
```

dir(): 내장 함수는 모듈의 정의하는 모든 유형의 이름(모듈, 변수, 함수)를 찾는다.

```
dir(sys)
```

package

Package 는 모듈과 __init__.py파일이 있는 디렉터리다.

__init__.py파일이 있어야 디렉터리를 패키지로 취급한다. 흔한 이름의 디렉터리에 유효한 모듈이 있는 경우 모듈이 검색되지 않는 문제를 방지한다.

__init__.py 는 빈 파일일 수도 있고 패키지의 초기화 코드를 실행하거나, __all__ 변수를 정의할 수도 있다.

__all__=[' 파일1 ' ,...]

from 폴더이름 import * :이름이 __로 시작하는 모듈을 제외한 모듈의 모든 객체를 불러온다.

__all__있는 경우 해당 리스트의 객체를 불러온다.

터미널에서 특정 모듈이 있는지 확인하려면

생각해보기

함수를 생성할 때, 함수 또는 메서드에서 mutable 객체를 기본값으로 사용하면 안된다.

 # 나쁜 예

```
def append (number ,number_list =[]):
    number_list .append (number )
    return number_list
>>>append(5)
[5]
>>>append(7)
[5,7]
```

 # 좋은 예

```
def append (number ,number_list =None ):
    if number_list is None :
        number_list =[]
    number_list .append (number )
    return number_list
>>>append(5)
[5]
>>>append(7)
[7]
```

참고자료 : return vs. yield

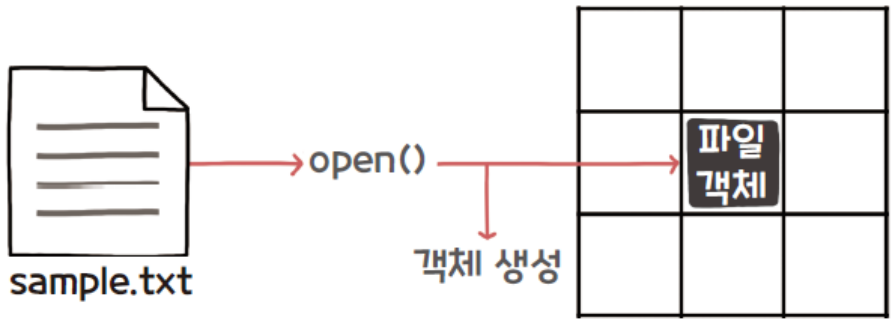
yield 키워드는 각 반환값을 호출자에게 반환하고, 반환값이 모두 소진되었을 때에만 메서드가 종료된다.

```
>>> a=[1,2,3]
>>> def f(a):
        while a:
            yield a.pop()

>>>def fib_generator():
        a, b = 0, 1
        while True:
                yield b
                a, b = b, a+b

>>>fib = fib_generator()
>>>print(next(fib))
>>>print(next(fib))
>>>print(next(fib))
>>>print(next(fib))
```

텍스트 파일 다루기

파일 다루기 3단계

- 텍스트 파일을 다루는 3단계
  - [1단계] 파일 열기
    - 첫 번째, 파일을 여는 단계입니다.
    - 파일을 열기 위해서는 open() 함수를 이용함
    - 파일 열기에 성공하면 파일은 객체로 만들어져 메모리에 생성됨



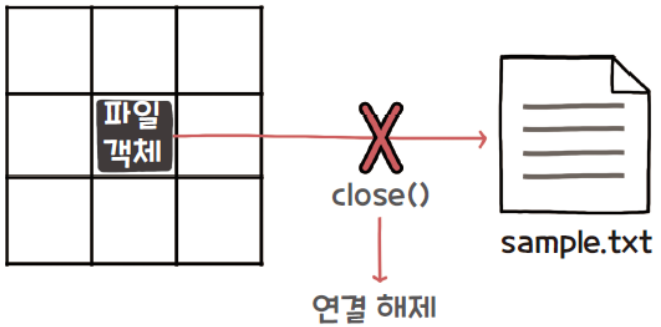
open( )에 의해서 객체로 생성됨

파일 다루기 3단계

- 텍스트 파일을 다루는 3단계
  - [2단계] 파일 쓰기/읽기
    - 두 번째, 문자열을 쓰거나 읽는 단계
    - 문자열을 쓸 때는 write() 함수를, 읽을 때는 read() 함수를 이용함

write()와 read()를 이용한 쓰기, 읽기

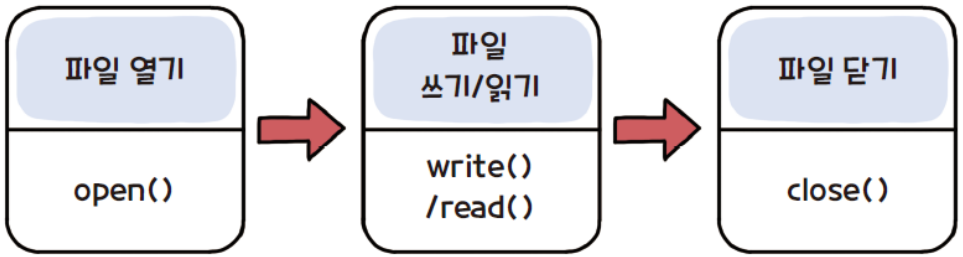
  - [3단계] 파일 닫기
    - 세 번째, 파일을 닫는 단계
    - 쓰기 또는 읽기가 끝난 파일은 close() 함수를 이용해서 연결을 해제함



close()에 의해서 연결 해제됨

파일 다루기 3단계

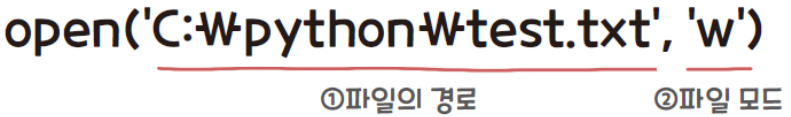
- 텍스트 파일을 다루는 3단계 정리



파일을 다루는 3단계

파일 열기/닫기

- 파일 열기
  - 파일을 열 때 사용하는 함수 open()은 두 개의 인수가 필요함
  - 첫 번째 인수는 파일의 경로이고, 두 번째 인수는 파일 모드임



open( ) 함수의 형식

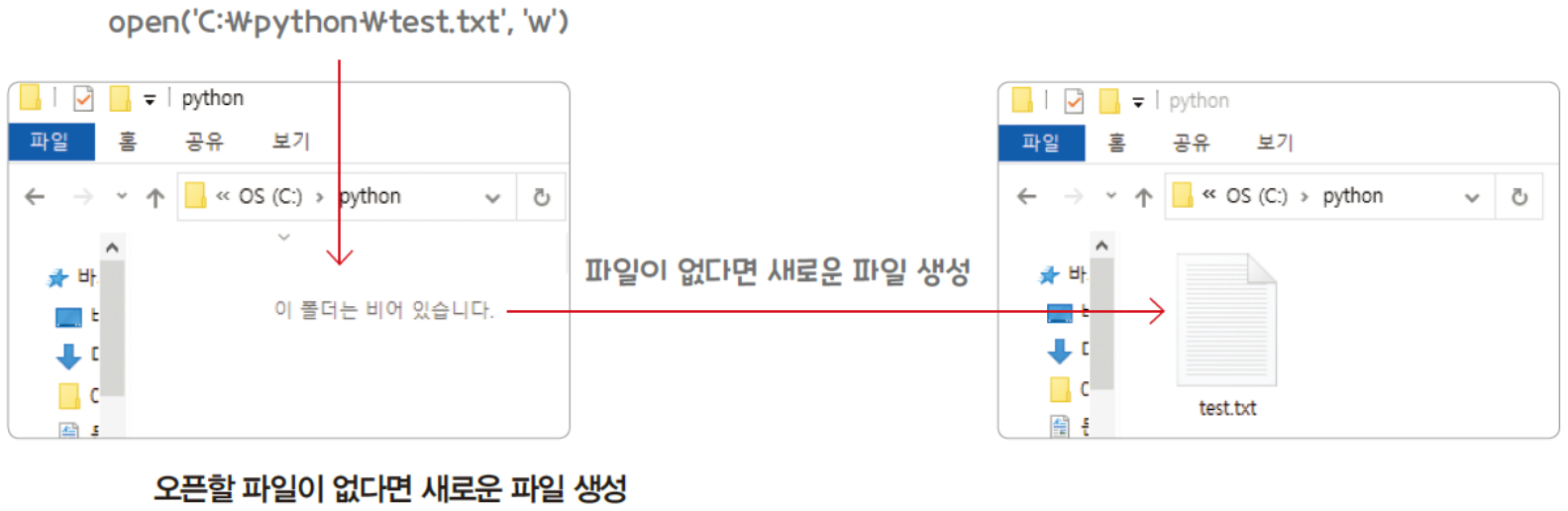
  - 파일 모드는 사용 목적에 따라서 파일을 여는 모드를 설정함

파일 모드

모드	사용 목적	특징
w	쓰기	기존 파일이 존재하면 새로운 파일로 덮어쓴다.
a	쓰기	기존 파일이 존재하면 기존 파일에 덧붙인다.
x	쓰기	기존 파일이 존재하면 에러가 발생한다.
r	읽기	파일을 읽는다. 파일이 없으면 에러가 발생한다.

파일 열기/닫기

- 파일 열기
  - 쓰기(w) 모드로 파일을 열 때, 만약 해당 경로에 '*.txt' 파일이 없다면 새로운 파일(*.txt)을 생성함



오픈할 파일이 없다면 새로운 파일 생성

파일 열기/닫기

- 파일 닫기
  - 작업을 마치고 파일을 닫을 때는 close() 함수를 사용함

```
file.close()
```

close( ) 함수의 형식

하나 더 알기 **파일 작업이 끝난 후에 close()를 호출하지 않으면 어떻게 되나요?**

파일을 제대로 닫지 않으면 파일을 필요로 하는 다른 곳에서 사용하지 못 할 수 있습니다. 또한 사용이 끝난 외부 파일이 메모리에 계속 남아 있기 때문에 시스템에 부하가 걸려 최악의 경우 시스템이 멈추는 현상까지 발생할 수 있습니다. 따라서 파일을 열어 쓰거나 읽기한 후에는 반드시 파일을 닫아야 합니다.

확인문제

확인문제

1. 텍스트 파일을 읽기 또는 쓰기 위한 과정으로 옳은 것은 무엇인가?

   ① save → read → open → close

   ② read → open → close

   ③ open → write / read → close

   ④ close → write / read → open

2. 다음 중 파일을 다루는 함수에 대한 설명으로 옳은 것은 무엇인가?

   ① open() : 파일을 저장하기 위한 함수이다.

   ② open() : 파일을 열기 위한 함수이다.

   ③ close() : 파일을 삭제하기 위한 함수이다.

   ④ close() : 파일을 암호화 처리하기 위한 함수이다.

파일 쓰기/읽기

- 파일 쓰기
  - 파일에 문자열을 쓸 때는 write() 함수를 사용함

```
file.write('Hello world~')
```

write() 함수의 형식

- write()함수
  - 파일에 '쓰기'를 실행한 후 데이터를 반환함
  - 반환값은 파일에 '쓰기'한 문자의 개수로 공백 문자까지 포함됨

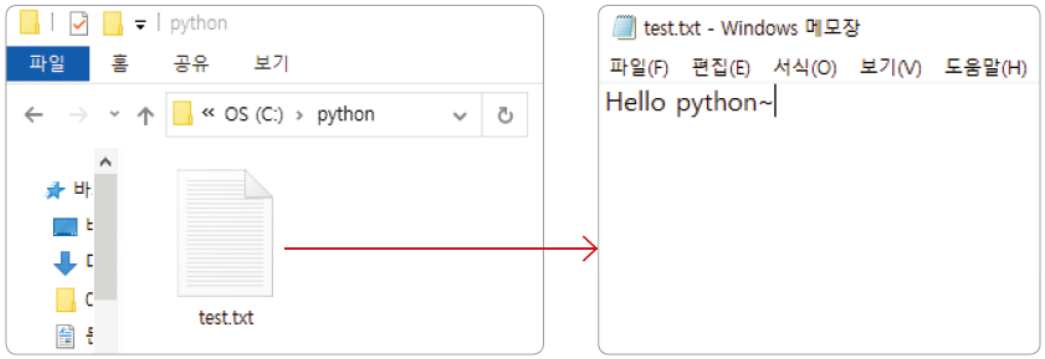
파일 쓰기/읽기

- 파일 쓰기

```
01   file = open('C:/python/test.txt', 'w')      # 파일 열기(쓰기 모드)
02   result = file.write('Hello python~')        # 쓰기
03   print('type of result :', type(result))     # 반환값의 데이터 타입 출력
04   print('result :', result)                   # 반환값 출력
05   file.close()                                # 파일 닫기
```

```
type of result : <class 'int'>
result : 13
```

write()의 반환값은 꼭 사용해야 하는 것은 아닙니다. 주로 문자열이 파일에 정상적으로 쓰였는지 확인하는 용도로 사용합니다.



test.txt에 기록된 문자열 확인

파일 쓰기/읽기

- 파일 읽기
  - 파일을 읽을 때는 read() 함수를 사용함

read() 함수의 형식

  - read()를 이용해서 파일에 문자열을 읽기 위한 코드

```
01   file = open('C:/python/test.txt', 'r')      # 파일 열기(읽기 모드)
02   result = file.read()                        # 읽기
03   print('type of result :', type(result))     # 반환값의 데이터 타입 출력
04   print('result :', result)
05   file.close()
```

```
type of result : <class 'str'>
result : Hello python~
```

파일 쓰기/읽기

- 파일 읽기
  - read()로 읽어 들인 데이터 타입은 항상 str(문자열)임

```
01   file = open('C:/python/number.txt', 'r')
02   result = file.read()
03   print('result :', result)
04   sum = result + 1
05   print('sum :', sum)
06   file.close()
```

```
result : 123
Traceback (most recent call last):
    sum = result + 1
TypeError: can only concatenate str (not "int") to str
```

  - 123을 read()로 읽어 들이면 숫자 123이 아닌 문자열 '123'으로 읽기 때문에 문자열에 정수 1을 더할 수 없어 에러가 발생함

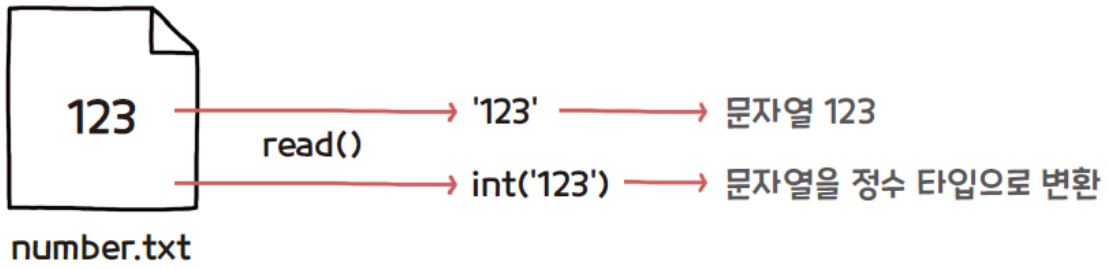
파일 쓰기/읽기

- 파일 읽기
  - 문자열을 정수로 변환하고 1을 더해 에러를 없앤 코드

```
01   file = open('C:/python/number.txt', 'r')
02   result = file.read()
03   print('result :', result)
04   sum = int(result) + 1
05   print('sum :', sum)
06   file.close()
```

문자열을 정수로 변환

```
result : 123
sum : 124
```



텍스트의 기본 반환 값인 문자열

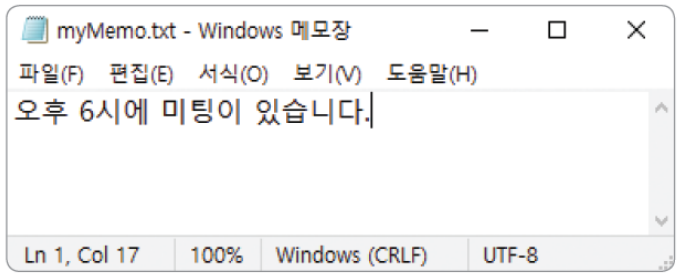
실습

일정 관리 메모를 입력하여 텍스트 파일에 저장하기

길동이는 미팅 약속을 잊어버리지 않기 위해 메모장에 기록으로 남겨두려고 합니다. 길동이가 입력한 내용을 텍스트 파일로 저장하는 프로그램을 만들어봅시다.

```
01    memo = input('메모를 입력하세요. ')
02
03
04
05
```

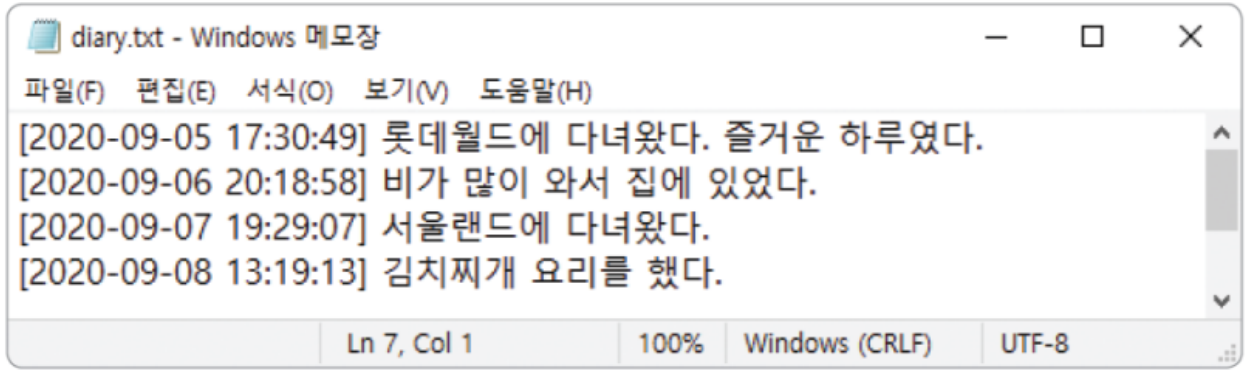
```
메모를 입력하세요. 오후 6시에 미팅이 있습니다.
```



실습

문제

하루 중 인상 깊었던 일을 파일에 한 줄로 작성하는 '한 줄 일기장' 프로그램을 만들어봅시다.



실습

당신의 회사에서는 매주 1회 작성해야 하는 보고서가 있습니다.
보고서는 항상 아래와 같은 형태로 출력되어야 합니다.

(예 : 35주차.txt 파일 내용)
- 35 주차 주간보고 -
부서 :
이름 :
업무 요약 :

1주차.txt
2주차.txt
3주차.txt
...
50주차.txt

1주차부터 50주차까지의 보고서 파일을 만드는 프로그램을 작성하시오.

조건 : 파일명은 '1주차.txt', '2주차.txt', ... 와 같이 만듭니다.
완성된 코드를 실행시키면 소스코드와 동일한 위치에 다음과 같이 50개의 파일이 생기면 됩니다.
그리고 각 파일에는 해당 주차에 해당하는 주간보고 내용이 포함됩니다.

실습

프로젝트 내에 나만의 시그니처를 남기는 모듈을 만드시오

조건: 모듈 파일명은 byme.py 로 작성

(모듈 사용 예제)

import byme

byme.sign()

(출력 예제)

이 프로그램은 김은연에 의해 만들어졌습니다.

유튜브: http://youtube.com

이메일: eunkim@gmail.com