4-5. 파일과 디렉토리 (1) 파일

우리는 지금까지 텍스트 데이터를 문자열로 나타낸 후 변수에 저장했습니다. 변수에 저장되는 데이터는 컴퓨터 메모리에 저장됩니다. 메모리에 저장되는 데이터는 빠르다는 장점이 있지만, 일정 전원이 공급되어야지만 데이터가 보존되죠. 즉, 프로그램이 실행되는 동안에만 데이터가 저장됩니다.

우리 일상에서 **파일**(file)은 너무나도 익숙한 존재이죠? 사용자의 중요한 정보가 담긴 파일이 매번 휘발되어 없어지면 안될 텐데요. 이러한 데이터를 영구적으로 보존할 필요가 있기 때문에 우리는 ROM(Read Only Memory)이라는 보조기억장치에 데이터를 저장합니다. 이것을 파일이라고 합니다.

파일

---

write

디스크상에 저장된 파일을 읽거나, 수정하거나, 혹은 데이터를 파일로 저장하려면 어떻게 해야 할까요? 우선 열어야 겠지요. 파일을 열고 객체로 만들어 주는 작업을 한번 해 보겠습니다.

[Input]

```
f = open("hello.txt","w")
#- open(파일명, 파일모드)
#- 파일을 열고 파일 객체를 반환합니다.
for i in range(10):
    f.write("안녕")
    #- write() 메소드로 '안녕'을 10번 씁니다.
f.close()
#- 작업이 끝나면 close() 메소드로 닫아줍니다. *필수!
```

실행

[Output]

read

위에서 `hello.txt` 파일이 잘 만들어졌나요? 정상적으로 만들어졌다면 주피터 커널을 실행했던 경로에 생성되어 있을 것입니다. 파일 탐색기를 이용해 실제로 만들어진 것을 확인해 보세요.
이번에는 파일을 읽어 보겠습니다. 방금 생성된 `hello.txt` 한번 읽어 볼까요?

[Input]

```
with open("hello.txt", "r") as f:
 print(f.read())
```

실행

[Output]

`hello.txt` 파일에 저장된 내용이 읽힌 것을 확인하실 수 있었을 거에요. 이제 파일에 대해 감이 좀 잡히셨나요?

여기서 `with` 구문을 사용해서 파일을 열어서 파일 객체를 `f` 로 받아서 `f.read()` 를 통해 내용을 읽었는데요, `f.close()` 를 명시적으로 호출하지 않고 있습니다. 이것은 실수가 아니라, `with` 를 통해 `open` 된 객체는 `with` 문이 종료될 때 자동으로 `close` 되는 것이 보장되기 때문입니다. 시스템 리소스의 안정적 사용을 위해 `with` 문 활용을 권장합니다.

마지막으로 파일 관련 메소드를 간단히 정리해 본 뒤 다음 스텝으로 넘어가 봅시다.

파일 관련 메소드

- `f.read()` : 파일을 읽는다.
- `f.readline()` : 파일을 한 줄씩 읽는다.
- `f.readlines()` : 파일 안의 모든 줄을 읽어 그 값을 리스트로 반환한다.

- `f.write(str)` : 파일에 쓴다. 문자열 타입을 인자로 받는다.
- `f.writelines(str)` : 파일에 인자를 한 줄씩 쓴다.
- `f.close()` : 파일을 닫는다.
- `f.seek(offset)` : 새 파일의 위치를 찾는다.