

데이터 과학을 위한 파이썬 프로그래밍



12. 예외 처리와 파일

목차

1. 예외 처리
2. 파일 다루기

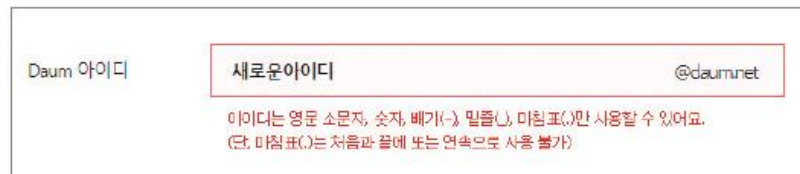
01

예외 처리

01. 예외 처리

■ 예외의 개념과 사례

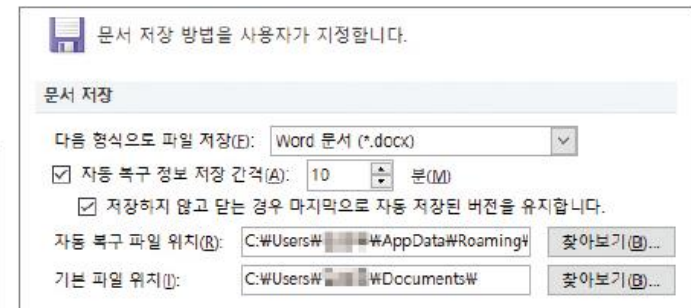
- 예외(exception) 란 프로그램을 개발하면서 예상하지 못한 상황이 발생한 것이다. 프로그래밍의 예외는 크게 예측 가능한 예외와 예측 불가능한 예외로 나눌 수 있다.



Daum 아이디 새로운아이디 @daum.net

아이디는 영문 소문자, 숫자, 배가(-), 밑줄(_) , 마침표(.)만 사용할 수 있어요.
(단, 마침표(.)는 처음과 끝에 또는 연속으로 사용 불가)

(a) 아이디 생성 오류 입력



문서 저장 방법을 사용자가 지정합니다.

문서 저장

다음 형식으로 파일 저장(E): Word 문서 (*.docx)

☒ 자동 복구 정보 저장 간격(A): 10 분(M)

☒ 저장하지 않고 닫는 경우 마지막으로 자동 저장된 버전을 유지합니다.

자동 복구 파일 위치(B): C:\Users\...#AppData#Roaming# 찾아보기(B)...

기본 파일 위치(I): C:\Users\...#Documents# 찾아보기(B)...

(b) 자동 저장 기능

[예외에 대비한 사례]

01. 예외 처리

■ 예측 가능한 예외와 예측 불가능한 예외

- **예측 가능한 예외** : 발생 여부를 개발자가 사전에 인지할 수 있는 예외이다. 개발자는 예외를 예측하여 명시적으로 예외가 발생할 때는 어떻게 대응하라고 할 수 있다. 대표적으로 사용자 입력란에 값이 잘못 들어갔다면, if문을 사용하여 사용자에게 잘못 입력하였다고 응답하는 방법이 있다. 매우 쉽게 대응할 수 있다
- **예측 불가능한 예외** : 대표적으로 매우 많은 파일을 처리할 때 문제가 발생할 수 있다. 예측 불가능한 예외가 발생했을 경우, 인터프리터가 자동으로 이것이 예외라고 사용자에게 알려준다. 대부분은 이러한 예외가 발생하면서 프로그램이 종료되므로 적절한 조치가 필요하다.

01. 예외 처리

■ 예외 처리 구문 : try -except문

- 파이썬 예외 처리의 기본 문법은 try -except문이다.

```
try:
```

```
    예외 발생 가능 코드
```

```
except 예외 타입:
```

```
    예외 발생 시 실행되는 코드
```

01. 예외 처리

■ 예외 처리 구문 : try -except문

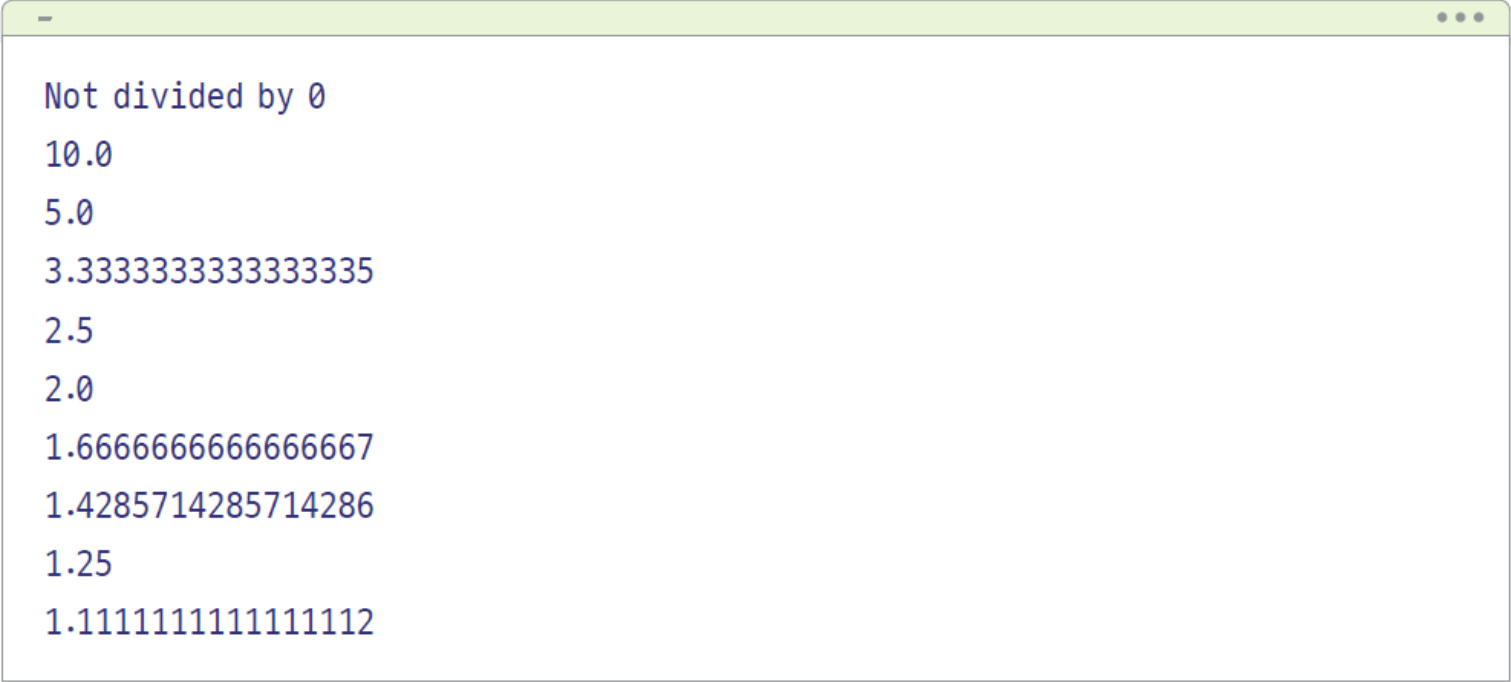
- 간단한 코드를 만들어 보자. [코드 12-1]은 0부터 9까지의 숫자를 i에 하나씩 할당하면서 10으로 나눈 값을 출력하는 코드이다. 이 프로그램은 1이 아닌 0부터 시작하다 보니 10을 0으로 나누는 계산이 가장 먼저 실행된다. 처음에 '10÷0(10/0)'을 하면 0으로는 10을 나눌 수 없으므로 예외가 발생한다. 하지만 이미 이러한 예외의 발생은 예상 가능하므로 try문으로 해당 예외가 발생할 때를 대비할 수 있다. ZeroDivisionError, 즉 0으로 나뉘진 경우에는 except 문 안으로 들어가 해당 구문에서 처리하는 코드가 정의된다. 여기서는 print("Not divided by 0") 코드가 실행된다.

코드 12-1 try-except.py

```
1 for i in range(10):
2     try:
3         print(10 / i)
4     except ZeroDivisionError:
5         print("Not divided by 0")
```

01. 예외 처리

■ 예외 처리 구문 : **try -except**문



```
Not divided by 0
10.0
5.0
3.3333333333333335
2.5
2.0
1.6666666666666667
1.4285714285714286
1.25
1.1111111111111112
```

- 그런데 만약 여기서 try문이 for문 밖으로 나가면 어떤 일이 발생할까? 이 경우, 이 반복문 전체가 종료된다. 즉, try문 내부에서 예외가 발생하면 except문 영역에서 코드가 실행되고, try except문이 종료된다. 이러한 이유로 try문을 적당한 곳에 삽입하여 예외 처리를 해야 한다.

01. 예외 처리

여기서 잠깐! 예외의 종류와 예외 에러 메시지

- 예외의 종류

예외	내용
IndexError	리스트의 인덱스 범위를 넘어갈 때
NameError	존재하지 않는 변수를 호출할 때
ZeroDivisionError	0으로 숫자를 나눌 때
ValueError	변환할 수 없는 문자나 숫자를 변환할 때
FileNotFoundError	존재하지 않는 파일을 호출할 때

01. 예외 처리

여기서 잠깐! 예외의 종류와 예외 에러 메시지

- **예외 에러 메시지** : 내장 예외와 함께 사용하기 좋은 것이 예외 에러 메시지이다. [코드 12-2]와 같이 except문의 마지막에 'as e' 또는 'as 변수명'을 입력하고, 해당 변수명을 출력하면 된다. 실행 결과 'division by zero'라는 에러 메시지를 확인할 수 있는데, 이 에러 메시지는 파이썬 개발자들이 사전에 정의한 것으로, 특정한 에러를 빠르게 이해할 수 있도록 돕는다.

코드 12-2 error_message.py

```
1 for i in range(10):
2     try:
3         print(10 / i)
4     except ZeroDivisionError as e:
5         print(e)
6         print("Not divided by 0")
```

01. 예외 처리

여기서  잠깐! 예외의 종류와 예외 에러 메시지

```
division by zero
Not divided by 0
10.0
5.0
3.3333333333333335
2.5
2.0
1.6666666666666667
1.4285714285714286
1.25
1.1111111111111112
```

01. 예외 처리

■ 예외 처리 구문 : try-except-else문

- try-except-else문은 if-else문과 비슷한데, 해당 예외가 발생하지 않을 경우 수행할 코드를 else문에 작성하면 된다.

```
try:  
    예외 발생 가능 코드  
except 예외 타입:  
    예외 발생 시 실행되는 코드  
else:  
    예외가 발생하지 않을 때 실행되는 코드
```

01. 예외 처리

■ 예외 처리 구문 : **try-except-else문**

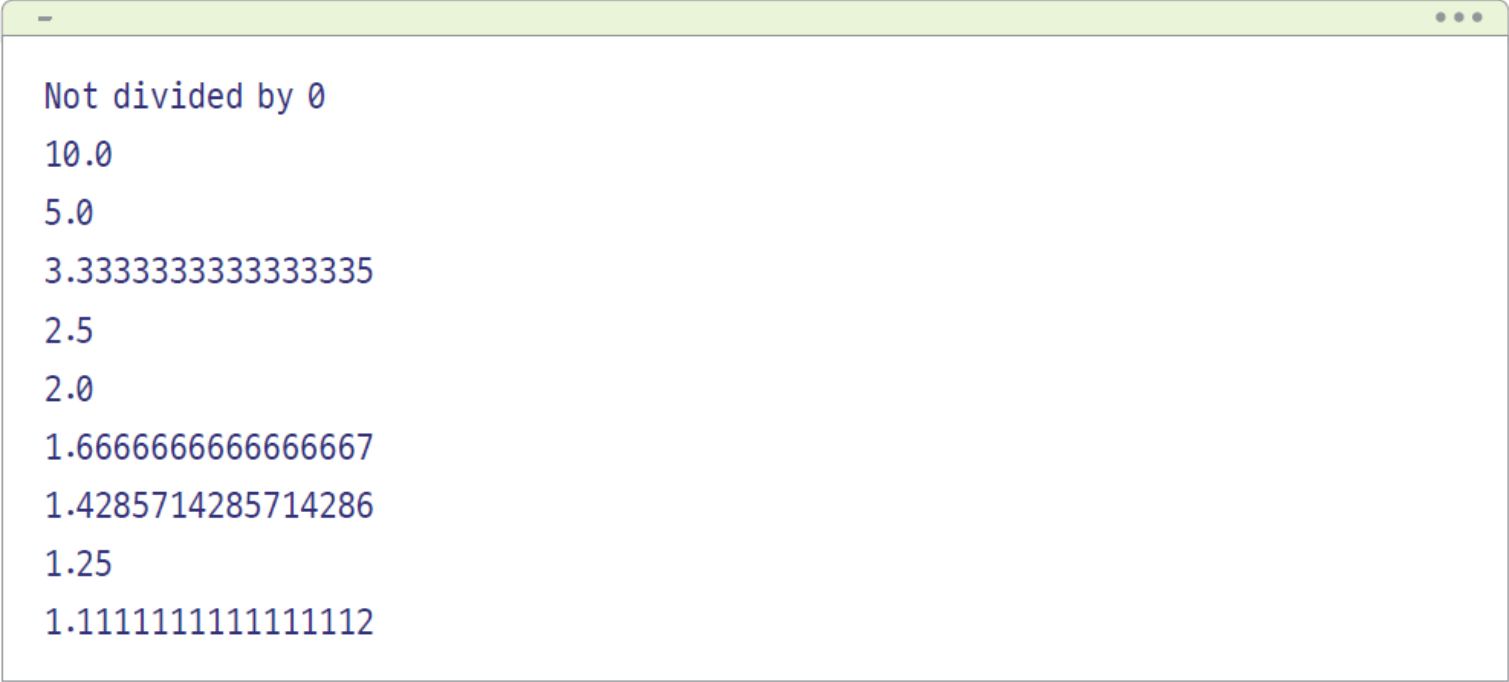
- [코드 12-3]은 10을 i로 나누는 코드를 실행하여 제대로 나누었을 경우 else문에 의해 결과가 화면에 출력되고, 그렇지 않을 경우 사전에 정의된 except문에 의해 에러가 발생하는 코드이다.

코드 12-3 try-except-else.py

```
1 for i in range(10):
2     try:
3         result = 10 / i
4     except ZeroDivisionError:
5         print("Not divided by 0")
6     else:
7         print(10 / i)
```

01. 예외 처리

■ 예외 처리 구문 : **try-except-else**문



```
Not divided by 0
10.0
5.0
3.3333333333333335
2.5
2.0
1.6666666666666667
1.4285714285714286
1.25
1.1111111111111112
```

01. 예외 처리

■ 예외 처리 구문 : **try-except-finally**문

- try-except-finally문에서 finally문은 try-except문 안에 있는 수행 코드가 아무런 문제 없이 종료되었을 경우, 최종으로 호출하는 코드이다

try:

예외 발생 가능 코드

except 예외 타입:

예외 발생 시 실행되는 코드

finally:

예외 발생 여부와 상관없이 실행되는 코드

01. 예외 처리

■ 예외 처리 구문 : **try-except-finally**문

코드 12-4 try-except-finally.py

```
1 try:
2     for i in range(1, 10):
3         result = 10 // i
4         print(result)
5 except ZeroDivisionError:
6     print("Not divided by 0")
7 finally:
8     print("종료되었다.")
```

```
10
5
3
2
```


01. 예외 처리

■ 예외 처리 구문 : **try-except-finally**문

```
2  
1  
1  
1  
1  
1  
종료되었다.
```

- ➡ 이 코드는 try문이 for문 밖으로 나가 i가 1부터 시작한다. 사실상 ZeroDivisionError가 발생할 수 없는 코드이다. 이러한 코드를 작성하면 except문을 사용할 수 없고, 마지막으로 finally문만 실행된다. try-except-finally문도 for문에서 finally문을 사용하는 것과 동일하게 예외 발생 여부와 상관없이 반드시 실행되는 코드이다.

01. 예외 처리

■ 예외 처리 구문 : **raise**문

- raise문은 try-except문과 달리 필요할 때 예외를 발생시키는 코드이다.

```
raise 예외 타입(예외 정보)
```

01. 예외 처리

■ 예외 처리 구문 : **raise**문

코드 12-5 raise.py

```
1 while True:
2     value =input("변환할 정수값을 입력해 주세요: ")
3     for digit in value:
4         if digit not in "0123456789":
5             raise ValueError("숫자값을 입력하지 않았습니다.")
6     print("정수값으로 변환된 숫자 -", int(value))
```

변환할 정수값을 입력해 주세요: 10

정수값으로 변환된 숫자 - 10

변환할 정수값을 입력해 주세요: ab

Traceback (most recent call last):

File "raise.py", line 5, in <module>

raise ValueError("숫자값을 입력하지 않았습니다.")

ValueError: 숫자값을 입력하지 않았습니다.

01. 예외 처리

■ 예외 처리 구문 : **raise**문

- ➡ [코드 12-5]는 while True문으로 반복문이 계속 돌아가면서 사용자에게 입력을 받는다. 하지만 사용자가 입력한 값이 숫자가 아닌 경우에는 숫자값을 입력받지 않았다고 출력하면서 프로그램을 종료하는 것을 목적으로 작성된 프로그램이다. 이때, 예외의 종료는 ValueError로 화면에 출력된다. 사용자가 입력을 잘못했을 때, 입력이 잘못된 것을 알려 주면서 종료하는 프로그램이다.

01. 예외 처리

■ 예외 처리 구문 : **assert문**

- assert문은 미리 알아야 할 예외 정보가 조건에 만족하지 않을 경우, 예외를 발생시키는 구문이다.

assert 예외 조건

01. 예외 처리

■ 예외 처리 구문 : **assert**문

코드 12-6 assert.py

```
1 def get_binary_nmubmer(decimal_number):  
2     assert isinstance(decimal_number, int)  
3     return bin(decimal_number)  
4 print(get_binary_nmubmer(10))  
5 print(get_binary_nmubmer("10"))
```

0b1010

← 5행 실행 결과

Traceback (most recent call last):

← 6행 실행 결과

File "<assert.py>", line 5, in <module>

print(get_binary_nmubmer("10"))

File "assert.py", line 2, in get_binary_nmubmer

assert isinstance(decimal_number, int)

AssertionError

01. 예외 처리

■ 예외 처리 구문 : **assert문**

- ➡ 1행에서 `get_binary_nmubmer()` 함수에 십진수가 들어온다. 하지만 함수를 사용하는 사용자가 잘못된 인수 `argument`, 예를 들어 문자열값을 입력할 수도 있다. 이를 방지하기 위해 2행에서 `assert문`을 사용하였다. `isinstance()` 함수는 입력된 값이 뒤에 있는 클래스의 인스턴스인지를 확인하는 함수이다. 이 코드에서 `decimal_number` 변수가 정수형인지는 5~6행에서 확인할 수 있다.
- `assert문`은 코드를 작성할 때 잘못된 입력 여부를 사전에 확인하여 나중에 필요 없는 연산을 막아 주며, 다른 사람이 만든 코드를 사용하는 데 좋은 가이드가 될 수 있다.

02

파일 다루기

02. 파일 다루기

■ 파일의 개념

- 파일(file)은 컴퓨터를 실행할 때 가장 기본이 되는 단위이다.

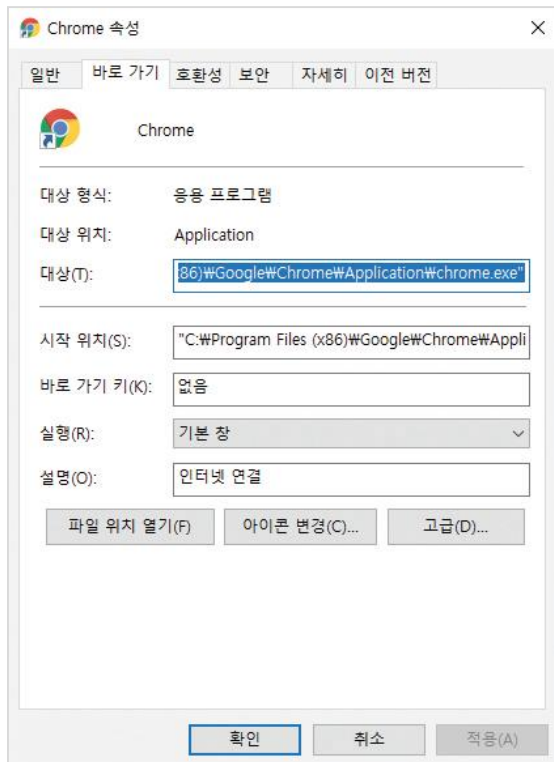


[윈도 GUI 환경의 아이콘]

02. 파일 다루기

■ 파일의 개념

- 사실 이러한 아이콘을 클릭하여 프로그램을 실행하는 것이 아니라, 실제로는 아이콘과 연결된 파일이 실행되는 구조이다. 아이콘에서 마우스 오른쪽 버튼을 클릭하고, 속성을 선택하면 다음과 같은 화면을 볼 수 있다.



[아이콘의 속성]

02. 파일 다루기

여기서 잠깐! 파일과 디렉터리

- 파일을 이해하기 위해 파일과 디렉터리에 대해 알아보자. 윈도우에서 사용하는 탐색기는 윈도우와 E 키를 함께 누르면 확인할 수 있다. 이것이 기본 파일 시스템이다. 기본적으로 파일 시스템은 파일과 디렉터리로 구분하는데, 윈도우에서는 디렉터리라는 용어 대신 폴더라는 용어를 사용한다.
- 디렉터리는 파일을 담는 또 하나의 파일로, 여러 파일을 포함할 수 있는 그릇이다. 파일과 다른 디렉터리를 포함할 수 있으므로 직접 프로그램을 실행하지는 않지만, 다른 파일들을 구분하고 논리적인 단위로 파일을 묶을 수 있다.
- 파일은 컴퓨터에서 정보를 저장하는 가장 작은 논리적인 단위이다. 파일은 일반적으로 파일명과 확장자로 식별한다. 예를 들어, 파이썬 파일로 저장 관리한 파일들은 py라는 확장자를 가지고 있다. 확장자는 그 파일의 쓰임을 구분하는 글자로, hwp, ppt, doc 같은 것이다. 파일은 다른 정보를 저장하거나 프로그램을 실행하거나 다른 프로그램이 실행될 때 필요한 정보를 제공하는 등의 역할을 한다.
- 흔히 탐색기 프로그램에서 파일과 디렉터리는 트리 구조로 표현되는데, 그 이유가 바로 디렉터리와 파일이 서로 포함 관계를 가지기 때문이다.

02. 파일 다루기

■ 파일의 종류

- 컴퓨터에서 파일의 종류는 다양하지만, 기본적으로 바이너리 파일(binary file)과 텍스트 파일(text file), 두 가지로 분류할 수 있다.

바이너리 파일	텍스트 파일
<ul style="list-style-type: none">컴퓨터만 이해할 수 있는 형태인 이진(법) 형식으로 저장된 파일일반적으로 메모장으로 열면 내용이 깨져 보임(메모장에서 해석 불가)엑셀 파일, 워드 파일 등	<ul style="list-style-type: none">사람도 이해할 수 있는 형태인 문자열 형식으로 저장된 파일메모장으로 열면 내용 확인이 가능메모장에 저장된 파일, HTML 파일, 파이썬 코드 파일 등

[바이너리 파일과 텍스트 파일]

02. 파일 다루기

■ 파일 읽기

- 파이썬에서는 텍스트 파일을 다루기 위해 `open()` 함수를 사용한다.

```
f = open("파일명", "파일 열기 모드")  
f.close()
```

종류	설명
r	읽기 모드: 파일을 읽기만 할 때 사용
w	쓰기 모드: 파일에 내용을 쓸 때 사용
a	추가 모드: 파일의 마지막에 새로운 내용을 추가할 때 사용


[파일 열기 모드]

02. 파일 다루기

■ 파일 읽기 : 파일 읽기 실행하기

코드 12-7 fileopen1.py

```
1 f = open("dream.txt", "r")
2 contents = f.read()
3 print(contents)
4 f.close()
```



```
I have a dream a song to sing
to help me cope with anything
if you see the wonder of a fairy tale
you can take the future even
if you fail I believe in angels
something good in everything
```

02. 파일 다루기

■ 파일 읽기 : 파일 읽기 실행하기

- ➔ 1행에서 `open()` 함수 다음에 파일명과 `r`을 사용하면 파일의 정보를 변수 `f`에 저장할 수 있다. 이를 일반적으로 파일 객체라고 한다. 2행에서 변수 `f`에서 `read()` 함수를 실행하면 해당 텍스트 파일의 텍스트를 `contents` 변수에 문자열로 저장한다. 3행에서는 'dream.txt' 파일을 불러와 화면에 출력한다. 4행에서는 최종으로 `close()` 함수를 사용하여 파일을 종료한다. 때때로 텍스트 파일을 수정할 때 이미 수정하고 있는 파일을 다른 프로그램이 함께 호출하면 에러가 발생하는데, 이렇게 하나의 파이썬 프로그램이 하나의 파일을 쓰고 있을 때 사용을 완료하면 반드시 해당 파일을 종료해야 한다.

02. 파일 다루기

■ 파일 읽기 : with문과 함께 사용하기

- with문과 함께 open() 함수를 사용할 수 있다. with문은 들여쓰기를 사용해 들여쓰기가 있는 코드에서는 open() 함수가 유지되고, 들여쓰기가 종료되면 open() 함수도 끝나는 방식이다.

코드 12-8 fileopen2.py

```
1 with open("dream.txt","r") as my_file:
2     contents = my_file.read()
3     print(type(contents), contents)
```

```
<class 'str'> I have a dream a song to sing
to help me cope with anything
if you see the wonder of a fairy tale
you can take the future even
if you fail I believe in angels
something good in everything
```


02. 파일 다루기

■ 파일 읽기 : 한 줄씩 읽어 리스트형으로 반환하기

- 파일 전체의 텍스트를 문자열로 반환하는 read() 함수 대신, readlines() 함수를 사용하여 한 줄씩 내용을 읽어 와 문자열 형태로 저장할 수 있다.

코드 12-9 fileopen3.py

```
1 with open("dream.txt","r") as my_file:
2     content_list = my_file.readlines()           # 파일 전체를 리스트로 반환
3     print(type(content_list))                   # 자료형 확인
4     print(content_list)                         # 리스트값 출력
```

```
<class 'list'>
['I have a dream a song to sing \n', 'to help me cope with anything \n', 'if you
see the wonder of a fairy tale \n', 'you can take the future even \n', 'if you
fail I believe in angels \n', 'something good in everything \n']
```

02. 파일 다루기

■ 파일 읽기 : 실행할 때마다 한 줄씩 읽어 오기

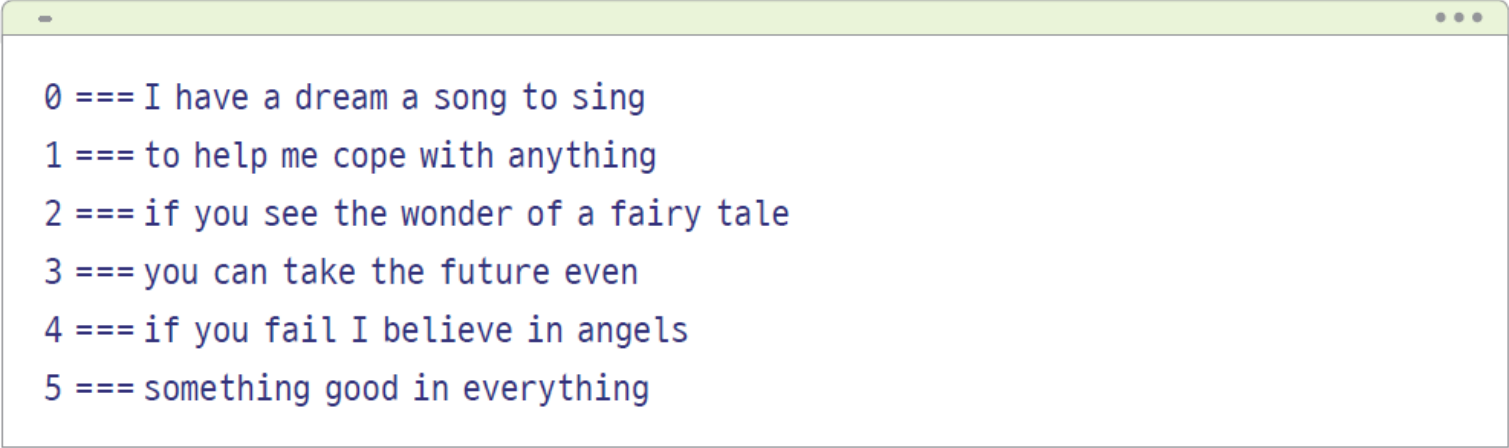
- readline() 함수는 실행할 때마다 차례대로 한 줄씩 읽어오는 함수이다.

코드 12-10 fileopen4.py

```
1 with open("dream.txt", "r") as my_file:
2     i = 0
3     while 1:
4         line = my_file.readline()
5         if not line:
6             break
7         print(str(i)+" == "+ line.replace("\n",""))    # 한 줄씩 값 출력
8         i = i + 1
```

02. 파일 다루기

■ 파일 읽기 : 실행할 때마다 한 줄씩 읽어 오기



```
0 === I have a dream a song to sing  
1 === to help me cope with anything  
2 === if you see the wonder of a fairy tale  
3 === you can take the future even  
4 === if you fail I believe in angels  
5 === something good in everything
```

- ➡ [코드 12-10]을 보면 while 1로 코드가 항상 작동하게 만든 다음, 4행의 line = my_file.readline()으로 한 줄씩 파일을 읽어 온다. 만약 읽어 온 줄에 내용이 없다면 5행의 if not line: break 코드에 의해 반복문이 종료되어 파일을 그만 읽게 된다. 하지만 파일에 남은 내용이 있다면 while이 계속 실행되면서 모든 코드를 다 읽어 오게 된다. 일반적으로 파일의 내용을 찾다가 중간에 멈춰야 할 필요가 있는 대용량 데이터는 [코드 12-10]과 같은 코드를 많이 사용한다.

02. 파일 다루기

■ 파일 읽기 : 파일 안 글자의 통계 정보 출력하기

- 때로는 파일 안 텍스트의 통계 정보를 읽어 와야 할 때가 있다. 이를 위해 많이 사용하는 방법은 이미 배운 `split()` 함수와 `len()` 함수를 함께 사용하는 것이다.

코드 12-11 fileopen5.py

```
1 with open("dream.txt", "r") as my_file:
2     contents = my_file.read()
3     word_list = contents.split(" ")           # 빈칸 기준으로 단어를 분리 리스트
4     line_list = contents.split("\n")         # 한 줄씩 분리하여 리스트
5
6 print("총 글자의 수:", len(contents))
7 print("총 단어의 수:", len(word_list))
8 print("총 줄의 수:", len(line_list))
```

```
총 글자의 수: 188      ← 6행 실행 결과
총 단어의 수: 35      ← 7행 실행 결과
총 줄의 수: 7          ← 8행 실행 결과
```

02. 파일 다루기

■ 파일 쓰기

- 텍스트 파일을 저장하기 위해서는 텍스트 파일을 저장할 때 사용하는 표준을 지정해야 하는데, 이것을 인코딩(encoding)이라고 한다.

코드 12-12 filewrite1.py

```
1 f = open("count_log.txt", 'w', encoding = "utf8")
2 for i in range(1,11):
3     data = "%d번째 줄이다.\n"% i
4     f.write(data)
5 f.close()
```

02. 파일 다루기

■ 파일 쓰기 : 파일 열기 모드 a로 새로운 글 추가하기

- 상황에 따라 파일을 계속 추가해야 하는 작업이 있을 수도 있으므로, 기존 파일에 추가 작업을 해야 하는 일이 있다. 이 경우, 많이 사용하는 방법은 추가 모드a 를 사용하는 것이다.

코드 12-13 filewrite2.py

```
1 with open("count_log.txt", 'a', encoding = "utf8") as f:
2     for i in range(1, 11):
3         data = "%d번째 줄이다.\n"% i
4         f.write(data)
```

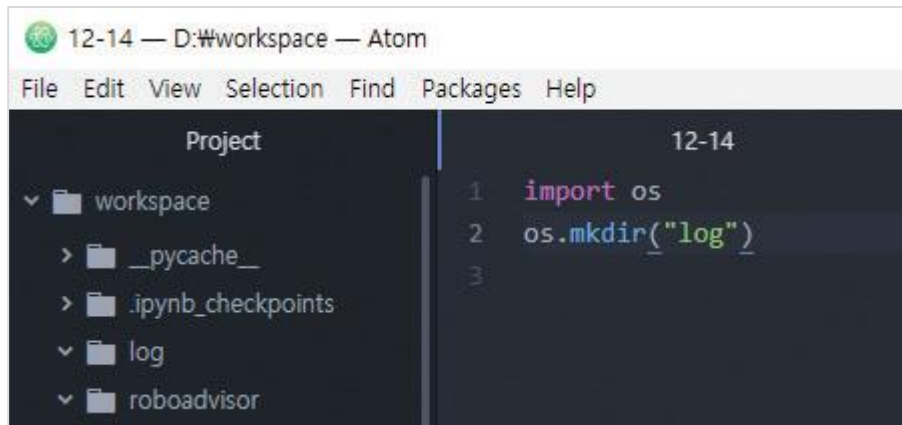
02. 파일 다루기

■ 파일 쓰기 : 디렉터리 만들기

- 파이썬으로는 파일만 다루는 것이 아니라, 디렉터리도 함께 다룰 수 있다. os 모듈을 사용하면 디렉터리를 쉽게 만들 수 있다.

코드 12-14 mkdir1.py

```
1 import os
2 os.mkdir("log")
```



[log 폴더 생성]

02. 파일 다루기

■ 파일 쓰기 : 디렉터리 만들기

- 프로그램 대부분이 새로 실행되므로 기존에 해당 디렉터리가 있는지 확인하는 코드가 필요하다. 이 경우 [코드 12-15]와 같이 `os.path.isdir` 모듈을 사용하여 기존 디렉터리의 존재 여부를 확인하면 된다

코드 12-15 mkdir2.py

```
1 import os
2 os.mkdir("log")
3
4 if not os.path.isdir("log"):
5     os.mkdir("log")
```

Traceback (most recent call last):

File "mkdir2.py", line 2, in <module>

os.mkdir("log")

FileExistsError: [WinError 183] 파일이 이미 있으므로 만들 수 없습니다: 'log'

02. 파일 다루기

■ 파일 쓰기 : 로그 파일 만들기

- 로그 파일(log file)은 프로그램이 동작하는 동안 여러 가지 중간 기록을 하는 파일이다.

코드 12-16 logfile.py

```
1 import os
2
3 if not os.path.isdir("log"):
4     os.mkdir("log")
5
6 if not os.path.exists("log/count_log.txt"):
7     f = open("log/count_log.txt", 'w', encoding = "utf8")
8     f.write("기록이 시작된다.\n")
9     f.close()
10
11 with open("log/count_log.txt", 'a', encoding = "utf8") as f:
12     import random, datetime
13     for i in range(1, 11):
```

02. 파일 다루기

■ 파일 쓰기 : 로그 파일 만들기

```
14         stamp = str(datetime.datetime.now())
15         value = random.random() * 1000000
16         log_line = stamp + "\t" + str(value) + "값이 생성되었다." + "\n"
17         f.write(log_line)
```

- ➡ 3~4행에서는 log 디렉터리가 존재하지 않을 경우, 새롭게 디렉터리를 만든다. 6~9행에서는 기존에 한 번도 로그 기록이 없었다면, w 모드로 count_log.txt 파일을 생성하고 기록의 시작을 알리는 문구를 저장한다. 11~17행은 예시를 만들기 위해 임의로 계속 시간 기록과 함께 임의의 숫자를 문구 안에 기록하여 저장한다. [코드 12-16]을 실행하면 딱 10번의 기록을 실행하지만, 실제로는 해당 코드가 호출할 때마다 시간과 함께 임의의 숫자가 계속 기록된다.

02. 파일 다루기

■ pickle 모듈

- 파이썬은 pickle 모듈을 제공하여 메모리에 로딩된 객체를 영속화할 수 있도록 지원한다.
- pickle 모듈을 사용하기 위해서는 다음 코드와 같이 호출한 후, 객체를 저장할 수 있는 파일을 열고 저장하고자 하는 객체를 넘기면(dump)된다. 파일을 생성할 때는 w가 아닌 wb로 열어야 하는데, 여기서 b는 바이너리(binary)를 뜻하는 약자로, 텍스트 파일이 아닌 바이너리 파일이 저장된 것을 확인할 수 있다. dump() 함수에서는 저장할 객체, 저장될 파일 객체를 차례대로 인수로 넣으면 해당 객체가 해당 파일에 저장된다.

```
>>> import pickle
>>>
>>> f = open("list.pickle", "wb")
>>> test = [1, 2, 3, 4, 5]
>>> pickle.dump(test, f)
>>> f.close()
```

02. 파일 다루기

■ pickle 모듈

- 저장된 pickle 파일을 불러오는 프로세스도 저장 프로세스와 같다. 먼저 list.pickle 파일을 rb 모드로 읽어 온 후, 해당 파일 객체를 pickle 모듈을 사용하여 load() 함수를 불러오면 된다. 다음 파이썬 셸 코드는 앞에서 리스트 객체를 list.pickle 파일에 저장했기 때문에 해당 파일을 불러 사용할 때도 동일하게 리스트 객체가 반환된 것을 확인할 수 있다.

```
>>> f = open("list.pickle", "rb")
>>> test_pickle = pickle.load(f)
>>> print(test_pickle)
[1, 2, 3, 4, 5]
>>> f.close()
```

02. 파일 다루기

■ pickle 모듈

- pickle 모듈은 단순히 생성된 객체를 저장하는 기능도 있지만, 사용자가 직접 생성한 클래스의 객체도 저장한다. 다음 코드와 같이 곱셈을 처리하는 클래스를 생성한다고 가정하자. 이 코드의 클래스는 처음 객체를 생성할 때 초깃값을 생성하고, multiply() 함수를 부를 때마다 '초깃값 * number'의 값을 호출하는 클래스이다. 일종의 곱셈기 클래스라고 생각하면 된다.

```
>>> class Mutltiply(object):
...     def __init__(self, multiplier):
...         self.multiplier = multiplier
...     def multiply(self, number):
...         return number * self.multiplier
...
>>> multiply = Mutltiply(5)
>>> multiply.multiply(10)
50
```

02. 파일 다루기

■ pickle 모듈

- 프로그램을 작성하다 보면 매우 복잡한 연산도 따로 저장하여 사용할 때가 있다. 이러한 저장 모듈을 효율적으로 사용하기 위해 다음 코드처럼 pickle 모듈을 사용할 수 있다.

```
>>> import pickle
>>>
>>> f = open("multiply_object.pickle", "wb")
>>> pickle.dump(multiply, f)
>>> f.close()
>>>
>>> f = open("multiply_object.pickle", "rb")
>>> multiply_pickle = pickle.load(f)
>>> multiply_pickle.multiply(5)
25
```

Thank You !