



파이썬과 인공지능

7주차 파일과 예외처리

학습 목표

- 텍스트 파일을 읽고 쓰는 프로그래밍을 할 수 있다.
- 다양한 파일 모드의 사용법을 이해 할 수 있다.
- 예외(Exception)와 오류(Error)를 구분하여 설명할 수 있다.
- 예외(Exception) 처리 블록을 이해하고 상황에 맞게 프로그래밍 할 수 있다.



| Part 01

파일 처리

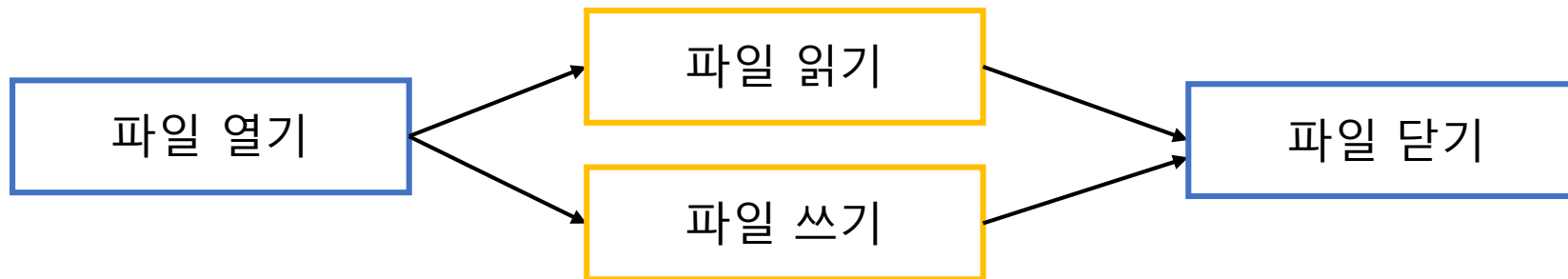




01. 파일 처리의 개요



- 지금까지 값을 입력받을 때 사용자로부터 입력받고, 출력할 때 모니터에 출력하는 방법으로 프로그래밍하는 방법을 사용
- 파일을 이용하여 프로그램에 입출력 가능
- 파일 처리는 프로그래밍에서 데이터를 외부 파일에 저장하거나 외부 파일에서 데이터를 읽어오는 작업을 의미함
- 데이터를 영구적으로 보관하거나 프로그램의 실행 상태를 저장하는데 매우 중요한 역할
- 파일을 사용(읽기, 쓰기)하기 위해서는 '파일 열기'를 하여야 함.
- 파일의 사용(읽기, 쓰기)이 모두 종료되면 '파일 닫기'를 하여야 함.





02. 파일 열기/쓰기와 닫기



- 파일을 열 때는 open() 함수를 사용한다.
- open() 함수는 다음과 같이 '파일 이름'과 '파일 열기 모드'를 입력값으로 설정한다.
- 함수에서 리턴된 파일 내용은 파일 객체에 저장하여 사용한다.

파일 객체 = open(파일이름, 파일 열기 모드)



02. 파일 열기/쓰기와 닫기



- 파일열기 모드는 다음과 같은 것들이 있다.

파일 열기 모드	의미
w	write(쓰기), 파일에 내용을 쓸 때 사용
a	append(추가하기), 파일의 마지막에 내용을 추가할 때 사용
r	read(읽기), 파일을 읽을 때 사용

- 파일을 w(write)모드로 열었을 때 지정된 파일이 존재하지 않으면 새로운 파일을 생성하고, 파일이 이미 존재하면 기존 파일의 내용을 삭제하고 새로 작성한다.



02. 파일 열기/쓰기와 닫기



- 파일 닫기는 close() 함수를 사용 열려있는 객체를 닫는다.

파일 객체.close()

- 프로그램이 종료 될 때 열려있는 파일 객체가 있다면 자동으로 닫아주기 때문에 close() 함수를 사용하지 않아도 무관하다.
- 하지만, close() 함수를 사용하지 않고 다시 사용하려고 한다면 오류가 발생하기 때문에 파일의 사용이 종료되면 close() 함수를 사용하여 파일 객체를 닫아주는 것이 좋다.



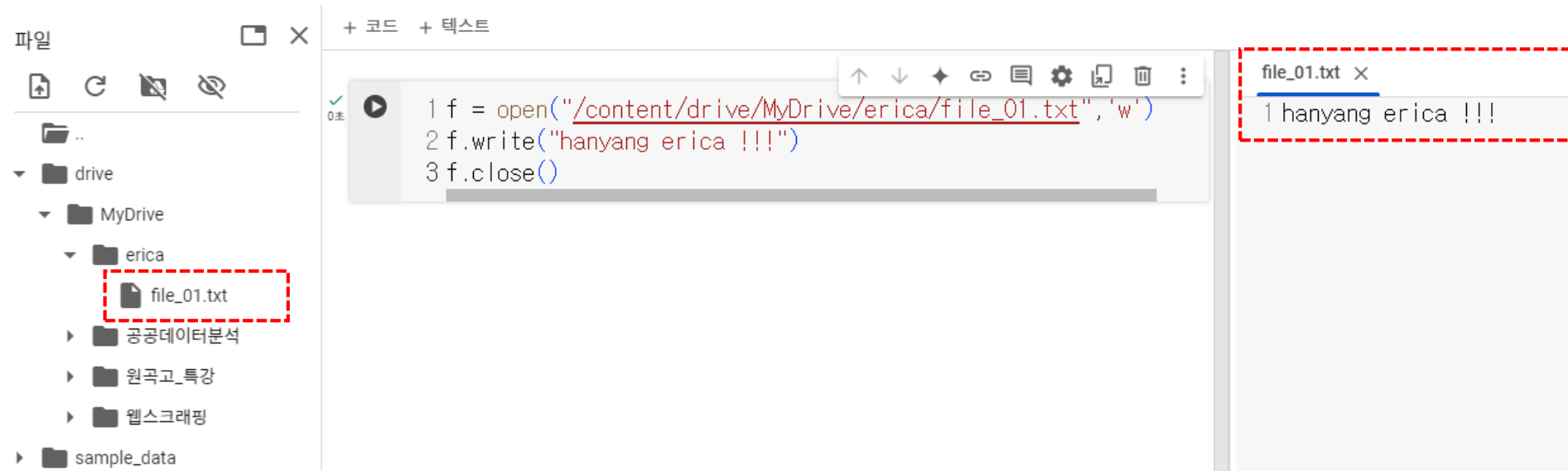
02. 파일 열기/쓰기와 닫기



- 예제 7-1

```
f = open("/content/drive/MyDrive/erica/file_01.txt", 'w')  
f.write("hanyang erica !!!")  
f.close()
```

- 결과 7-1





02. 파일 열기/쓰기와 닫기



- with 키워드를 사용하여 파일을 닫는 코드를 생략할 수 있다.
- with 키워드를 사용한 구문이 종료될 때 자동으로 파일이 닫힌다.
- with 구문은 들여쓰기 하여야 한다.

```
with open(파일이름, 파일 열기 모드) as 파일객체:  
    파일객체.write("텍스트")
```



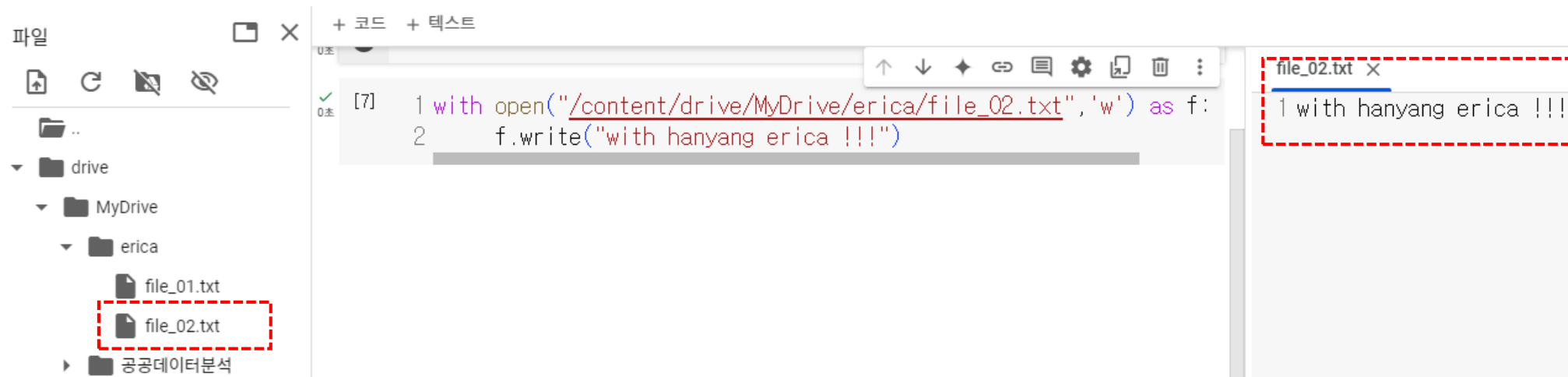
02. 파일 열기/쓰기와 닫기



- 예제 7-2

```
with open("/content/drive/MyDrive/erica/file_02.txt",'w') as f:  
    f.write("with hanyang erica !!!")
```

- 결과 7-2





03. 파일 읽기



- 파일의 전체 내용을 읽기 위해서는 read() 함수를 사용한다.
- read() 함수는 파일의 내용 전체를 문자열로 반환한다. 따라서 반환된 문자열을 저장할 객체를 사용하여 작성한다.

```
문자열 저장 객체 = 파일 객체.read()
```



03. 파일 읽기



- 예제 7-3

```
f = open("/content/drive/MyDrive/erica/file_01.txt", 'r')  
str01 = f.read()  
print(str01)  
f.close()
```

- 결과 7-3

```
파일  
+ 코드 + 텍스트  
0초  
1 f = open("/content/drive/MyDrive/erica/file_01.txt", 'r')  
2 str01 = f.read()  
3 print(str01)  
4 f.close()  
hanyang erica !!!
```



03. 파일 읽기



- readline()함수는 텍스트 파일에서 한 줄씩 데이터를 읽어오는데 사용된다.
- readline()함수를 한 번 호출하면 파일에서 한 줄을 읽어 반환하고 다시 호출을 하면 그 다음 줄의 읽어 반환한다.
- 즉, 호출될 때마다 한 줄씩 순차적으로 문서 끝까지 반환한다.

- 예제 7-4

```
f = open("/content/drive/MyDrive/erica/file_03.txt", 'r')
while True:
    str = f.readline()
    if not str:
        break
    print(str)
f.close()
```



03. 파일 읽기



- 결과 7-4

```
1 f = open("/content/drive/MyDrive/erica/file_03.txt", 'r')  
2  
3 while True:  
4     str = f.readline()  
5     if not str:  
6         break  
7     print(str)  
8 f.close()
```

⇒ 1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
5번째 줄입니다.
6번째 줄입니다.



03. 파일 읽기



- readlines()함수는 텍스트 파일에서 모든 줄을 읽어 각 줄을 요소로 갖는 리스트를 반환한다.
- 즉, readlines()함수는 예제 7-4에서의 readline()함수와는 다르게 '1번째 줄입니다.', '2번째 줄입니다'... 의 문자열을 갖는 리스트를 반환하게 되는 원리이다.
- 예제 7-5

```
f = open("/content/drive/MyDrive/erica/file_03.txt",'r')  
  
str = f.readlines()  
for i in str:  
    print(i)  
f.close()
```



03. 파일 읽기



- 결과 7-4

+ 코드 + 텍스트 | Drive로 복사

```
✓ 0초 ▶ 1 f = open("/content/drive/MyDrive/erica/file_03.txt", 'r')
      2
      3 str = f.readlines()
      4 for i in str:
      5     print(i)
      6 f.close()
```

↻ 1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
5번째 줄입니다.
6번째 줄입니다.

I Part 02

예외 처리





01. 오류의 종류



- 프로그램의 오류에는 크게 두 가지 종류가 있다.
 - ✓ 구문 오류(syntax error): 프로그램을 번역하는 과정에서 발생하는 오류
 - ✓ 오타자, 들여쓰기 오류, 잘못된 예약어의 사용 등과 같이 프로그램이 실행되기 전에 발생하는 오류를 말한다.

```
1 print("안녕하세요.")
```

File "<ipython-input-6-28341a6c09b1>", line 1
print("안녕하세요.")
^
SyntaxError: unterminated string literal (detected at line 1)

다음 단계: [오류 수정](#)

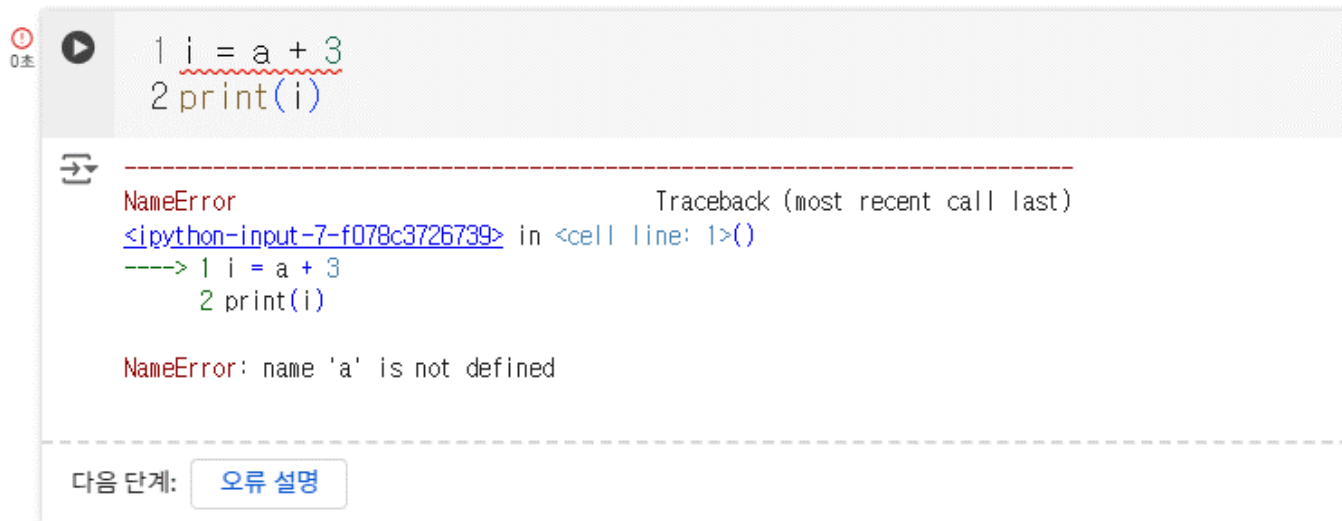
- ✓ print()함수의 매개변수에서 닫는 큰따옴표(")의 누락으로 오류가 발생한다.
- ✓ 오류의 설명에 'SyntaxError'라고 표시되며, 이는 구문의 오류로 프로그램이 실행되기 전에 발생하는 오류에 해당된다.



01. 오류의 종류



- 프로그램의 오류에는 크게 두 가지 종류가 있다.
 - ✓ 런타임 오류(runtime error): 프로그램을 실행 하는 중에 발생하는 오류, 예외(exception)라고 부른다.
 - ✓ 할 수 없는 계산, 존재하지 않는 변수명 사용, 파일, 인덱스(리스트 등에서)의 접근 등과 같이 프로그램의 실행 과정에서 발생하는 오류를 말한다.



```
1 i = a + 3
2 print(i)
```

NameError Traceback (most recent call last)

<ipython-input-7-f078c3726739> in <cell line: 1>()

----> 1 i = a + 3

2 print(i)

NameError: name 'a' is not defined

다음 단계: [오류 설명](#)

- ✓ name 'a' is not defined: 변수 a가 정의 되지 않아 런타임 오류가 발생하였다.

01. 오류의 종류



0초



```
1 a = 3
2 i = a / 0
3 print(i)
```



```
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-8-82639a816703> in <cell line: 2>()
      1 a = 3
----> 2 i = a / 0
      3 print(i)

ZeroDivisionError: division by zero
```

✓ ZeroDivisionError: 0으로 나누어 런타임 오류가 발생

0초



```
1 f = open("없는파일.txt", 'r')
2 f.write("hanyang erica !!!")
3 f.close()
```



```
FileNotFoundError                                Traceback (most recent call last)
<ipython-input-5-d9fd94129b8e> in <cell line: 1>()
----> 1 f = open("없는파일.txt", 'r')
      2 f.write("hanyang erica !!!")
      3 f.close()

FileNotFoundError: [Errno 2] No such file or directory: '없는파일.txt'
```

✓ FileNotFoundError: 파일을 찾을 수 없어 런타임
오류가 발생



02. 예외 처리의 개요



- 컴퓨터 프로그램은 사용자 입력, 네트워크 연결, 파일 입출력 등 다양한 외부 환경과 상호작용하면서 예기치 않은 상황에 직면할 수 있다. 예를 들어, 존재하지 않는 파일을 열거나 숫자가 아닌 값을 정수로 변환하려고 하면 프로그램은 오류를 발생시키고 종료될 수 있다.
- 이러한 오류는 프로그램의 안정성을 저해하고, 사용자 경험을 해칠 수 있다.
- 파이썬에서는 예외 처리(Exception Handling)라는 강력한 메커니즘을 통해 이러한 문제를 효과적으로 관리할 수 있다.
- 예외 처리는 오류가 발생하더라도 프로그램이 예기치 않게 종료되지 않도록 하고, 개발자가 정의한 방식으로 문제를 해결하거나 사용자에게 알리는 방법을 제공한다.



02. 예외 처리의 개요



- 예외처리를 해야 하는 이유는 다음과 같다.

- 프로그램의 안정성 및 신뢰도 향상: 예외 처리를 통해 프로그램이 예기치 않게 종료되는 것을 방지할 수 있다.
- 오류 분석 용이: 발생한 오류를 잘 처리하고, 사용자에게 명확한 오류 메시지를 제공함으로써 문제를 쉽게 파악할 수 있다.
- 프로그램 흐름 제어: 예외 발생 시 적절한 대체 흐름을 제어하여 프로그램을 계속 실행할 수 있도록 할 수 있다.



02. 예외 처리의 개요



- 예외 처리는 크게 두 가지 방법으로 나누어 설명할 수 있다.
 - ✓ if문을 사용한 예외 처리: 전통적인 방법으로 예외가 발생할 경우를 모두 if문을 사용하여 처리 하는 방법
 - ✓ try, except를 사용한 예외 처리: 예외 처리만을 위한 구문으로 발생 가능한 다양한 예외를 처리하기 위한 방법



03. if문을 사용한 예외 처리



- if문을 사용한 예외 처리는 예외가 발생할 경우가 비교적 분명한 경우에 사용한다.
- 예제 7-6

```
1 loan = input("대출금을 입력하세요:")
2 month = input("상환 개월을 입력하세요:")
3 result = int(loan) / int(month)
4 print(int(result), end="")
5 print("을 상환합니다.")
```

☞ 대출금을 입력하세요:1000000
상환 개월을 입력하세요:0

```
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-22-5eca000ecbf4> in <cell line: 3>()
      1 loan = input("대출금을 입력하세요:")
      2 month = input("상환 개월을 입력하세요:")
----> 3 result = int(loan) / int(month)
      4 print(int(result), end="")
      5 print("을 상환합니다.")
```

ZeroDivisionError: division by zero

- ✓ 매월 상환 금액은 $\text{loan} / \text{month}$ 의 식으로 구할 수 있으나, month에 0을 입력하여 예외가 발생하게 되며, 0이하의 숫자를 month에 입력하게 되면 정확한 결과를 얻을 수 없다.



03. if문을 사용한 예외 처리



- 예제 7-7

```
1 loan = input("대출금을 입력하세요:")
2 month = input("상환 개월을 입력하세요:")
3 result = int(loan) / int(month)
4 print(int(result), end="")
5 print("을 상환합니다.")
```

대출금을 입력하세요:1000000
상환 개월을 입력하세요:12개월

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-23-5eca000ecbf4> in <cell line: 3>()
      1 loan = input("대출금을 입력하세요:")
      2 month = input("상환 개월을 입력하세요:")
----> 3 result = int(loan) / int(month)
      4 print(int(result), end="")
      5 print("을 상환합니다.")
```

ValueError: invalid literal for int() with base 10: '12개월'

✓ 뿐만 아니라 상환 개월에 숫자만 입력하지 않고 '12개월'의 형식으로 입력하여도 오류가 발생하게 된다..



03. if문을 사용한 예외 처리



- 이와 같이 발생할 예외가 비교적 예상되는 경우에는 if문을 사용하여 예외 처리를 할 수 있다.
- 예제 7-8

```
1 loan = input("대출금을 입력하세요:")
2 month = input("상환 개월을 입력하세요:")
3
4 if month.isdigit() == False:
5     print("숫자를 입력하세요.")
6 elif int(month) <= 0:
7     print("0이하는 사용할 수 없습니다.")
8 else:
9     result = int(loan) / int(month)
10    print(int(result), end="")
11    print("을 상환합니다.")
```

↔ 대출금을 입력하세요:1000000
상환 개월을 입력하세요:0
0이하는 사용할 수 없습니다.



04. try except를 사용한 예외 처리



- try except를 사용한 예외 처리는 예외가 발생할 경우가 비교적 많아 모두 처리할 수 없을 때 사용한다.
- 프로그램에서 예외(오류)가 발생할 수 있는 상황이 있는 경우, 오류가 발생해도 프로그램이 종료되지 않고 정상적으로 처리될 수 있도록 한다.
- try 블록: 예외가 발생할 가능성이 있는 코드를 포함하는 부분이다. try 블록 안의 코드가 실행되는 동안 예외가 발생하면, 해당 예외는 except 블록으로 전달된다. 예외가 발생하지 않으면 except 블록은 무시된다.
- except 블록: try 블록 내에서 예외가 발생했을 때 이를 처리하는 부분이다.
- except 블록은 발생한 다양한 예외에 대해 동일한 방법으로 처리할 수 있을 뿐만 아니라 발생한 예외에 따라 다르게 처리할 수 있다.

```
try :  
    예외가 발생할 가능성이 있는 코드  
except [발생 예외[as 오류 객체]]:  
    예외가 발생했을 때 실행할 코드
```



04. try except를 사용한 예외 처리



- 모든 예외 한번에 처리하기: except 블록을 사용하여 try 블록에서 어떠한 예외가 발생하더라도 모두 처리 가능하다.
- 예제 7-9

```
1 try:
2     # 예외가 발생할 수 있는 코드
3     num = int(input("숫자를 입력하세요: "))
4     result = 10 / num
5 except :
6     print("유효한 숫자를 입력하세요.")
```

숫자를 입력하세요: 10입니다.
유효한 숫자를 입력하세요.



04. try except를 사용한 예외 처리



- 특정 예외를 처리하기: except 블록을 사용할 때 특정 예외를 명시하여 특정 예외에 대해 다르게 예외를 처리하도록 한다.
- 예제 7-10

```
1 try:
2     # 예외가 발생할 수 있는 코드
3     num = int(input("숫자를 입력하세요: "))
4     result = 10 / num
5 except ValueError:
6     print("유효한 숫자를 입력하세요.")
7 except ZeroDivisionError:
8     print("0으로 나눌 수 없습니다.")
```

↻ 숫자를 입력하세요: 0
0으로 나눌 수 없습니다.

- 0으로 나누었기 때문에 'ZeroDivisionError' 오류가 발생하는 입력 값이므로 그에 대한 except 블록을 수행한다.



04. try except를 사용한 예외 처리



- 특정 예외를 처리하기: except 블록을 사용할 때 특정 예외를 명시하여 특정 예외에 대해 다르게 예외를 처리하도록 한다.
- 예제 7-11

```
1 try:
2     # 예외가 발생할 수 있는 코드
3     num = int(input("숫자를 입력하세요: "))
4     result = 10 / num
5 except ValueError:
6     print("유효한 숫자를 입력하세요.")
7 except ZeroDivisionError:
8     print("0으로 나눌 수 없습니다.")
```

☞ 숫자를 입력하세요: 5입니다.
유효한 숫자를 입력하세요.

- 숫자가 아닌 값을 입력하였기 때문에 'ValueError' 오류가 발생하는 입력 값이므로 그에 대한 except 블록을 수행한다.



04. try except를 사용한 예외 처리



- 발생한 오류를 출력을 통해 확인하고자 할 경우에는 'as 오류메세지 객체'와 같은 형식을 사용하여 확인할 수 있다.

예제 7-12

```
1 try:
2     num = int(input("숫자를 입력하세요: "))
3     result = 10 / num
4 except ValueError:
5     print("유효한 숫자를 입력하세요.")
6 except ZeroDivisionError as errorM:
7     print("0으로 나눌 수 없습니다.")
8     print(errorM)
9 else:
10    print(f"결과는 {result}입니다.")
```

☞ 숫자를 입력하세요: 0
0으로 나눌 수 없습니다.
division by zero

- errorM 은 예외객체이다. 예외 객체 (Exception object)는 파이썬에서 예외가 발생할 때 생성되는 객체이다. 예외가 발생하면 해당 예외의 종류에 따라 예외 객체가 생성되고 예외 정보를 담고 있다.
- 이 예외 객체에는 예외의 유형, 메시지, 발생 위치 등과 같은 정보가 포함된다.



04. try except를 사용한 예외 처리



- try, except 외에도 else, finally는 파이썬에서 예외 처리를 위한 구조이다.
- else 블록: try 블록에서 예외가 발생하지 않았을 경우 실행된다. 예외가 발생하지 않으면 else 블록의 코드가 실행되며, 이는 일반적으로 예외가 발생하지 않은 경우에 수행할 후속 작업을 처리하는 데 사용된다.
- finally 블록: 예외 발생 여부와 관계없이 무조건 실행되는 코드이다. finally는 리소스 해제나 정리 작업을 할 때 유용하다. 예를 들어, 파일을 열었으면 파일을 닫는 작업을 finally 블록에서 할 수 있다.

```
try :  
    예외가 발생할 가능성이 있는 코드  
except [발생 예외[as 오류 객체]]:  
    예외가 발생했을 때 실행할 코드  
else :  
    예외가 발생하지 않았을 때 실행할 코드  
finally :  
    예외 발생 유무와 관계없이 실행할 코드
```




04. try except를 사용한 예외 처리



- else 블록은 try 블록에서 예외가 발생하지 않았을 때 실행된다.
- 예제 7-13

```
1 try:
2     num = int(input("숫자를 입력하세요: "))
3     result = 10 / num
4 except ValueError:
5     print("유효한 숫자를 입력하세요.")
6 except ZeroDivisionError:
7     print("0으로 나눌 수 없습니다.")
8 else:
9     print(f"결과는 {result}입니다.")
```

➡ 숫자를 입력하세요: 5
결과는 2.0입니다.

- 예외가 발생하지 않았기 때문에 except블록은 실행되지 않고 else블록만 수행되어 연산 된 결과가 출력된다.



04. try except를 사용한 예외 처리



- finally 블록은 예외가 발생했는지 여부와 관계없이 마지막에 항상 실행된다.

- 예제 7-14

```
1 try:
2     num = int(input("숫자를 입력하세요: "))
3     result = 10 / num
4 except ValueError:
5     print("유효한 숫자를 입력하세요.")
6 except ZeroDivisionError:
7     print("0으로 나눌 수 없습니다.")
8 else:
9     print(f"결과는 {result}입니다.")
10 finally:
11     print("프로그램 실행이 종료되었습니다.")
```

숫자를 입력하세요: 5
결과는 2.0입니다.
프로그램 실행이 종료되었습니다.

- try 블록에서 예외 발생 유무와 관계없이 finally 블록의 명령이 수행되어, "프로그램 실행이 종료되었습니다."라는 메시지가 출력된다.



04. 예외 통과하기



- 예외가 발생하더라도 아무런 코드를 작성하고 싶지 않는 경우 pass 키워드를 사용하여 처리할 수 있다.
- pass는 문법적으로 아무 코드도 작성하지 않겠다는 의미를 전달하며, 다른 명령어들과 마찬가지로 프로그램 흐름에 영향을 미치지 않는다.
- 예외가 발생하더라도 무시하고 프로그램을 실행하고자 할 경우 except의 블록 내에 pass 키워드를 사용한다.
- 예제 7-15

```
1 try:
2     # 일부 코드에서 예외가 발생할 수 있음
3     num = int(input("숫자를 입력하세요: "))
4 except ValueError:
5     pass # ValueError가 발생해도 아무 작업도 하지 않음
```

숫자를 입력하세요: 3입니다.



05. 예외 발생 시키기



- raise는 파이썬에서 예외를 강제로 발생시키는 명령어이다.
- raise를 사용하면 프로그램 실행 중에 명시적으로 예외를 발생시킬 수 있다.
- 사용자 정의 예외를 발생시키거나, 특정 조건이 충족되었을 때 예외를 발생시켜 프로그램의 흐름을 제어할 때 유용하게 사용된다.

```
raise Exception("에러 메시지")
```



05. 예외 발생 시키기



- raise 뒤에 예외 클래스(Exception, ValueError, TypeError 등)를 지정하여 예외를 발생시킨다. 즉, 예외 클래스의 인자로 오류 메시지나 추가적인 정보를 전달할 수 있다.

- 예제 7-16

```
1 def divide(a, b):  
2     if b == 0:  
3         raise ZeroDivisionError("0으로 나눌 수 없습니다!")  
4     return a / b  
5  
6 try:  
7     result = divide(10, 0)  
8 except ZeroDivisionError as e:  
9     print(f"에러 발생: {e}")
```

에러 발생: 0으로 나눌 수 없습니다!

- 함수 divide()의 두 번째 인자가 0일 때 ZeroDivisionError 오류가 발생되고, 예외 처리 블록에서 해당 예외를 처리하여 사용자에게 오류 메시지를 출력한다.



05. 예외 발생 시키기



- raise는 이미 발생한 예외를 다시 발생시킬 때도 사용된다. 이는 예외를 처리한 후, 예외를 다시 전달하고 싶을 때 유용하다.

예제 7-17

```
1 def process_data(data):  
2     try:  
3         if not data:  
4             raise ValueError("데이터가 비어 있습니다.")  
5         # 데이터 처리 코드  
6     except ValueError as e:  
7         print(f"에러: {e}")  
8         raise # 예외를 다시 발생시킴  
9  
10 try:  
11     process_data("")  
12 except ValueError as e:  
13     print(f"최종 에러 처리: {e}")
```

↗ 에러: 데이터가 비어 있습니다.
↗ 최종 에러 처리: 데이터가 비어 있습니다.

- ValueError 예외를 잡은 후, raise를 사용하여 예외를 다시 발생시키고, 호출된 위치에서 이를 처리하도록 한다.

