

IT 개론

9장. 파일 처리, 예외 처리

목차

1. 파일 연결 - open() 내장 함수
2. 파일 입력 - 파일로부터 데이터 읽어오기
3. 파일 출력 - 파일에 데이터 저장하기
4. 예외 처리
 - 구문 에러
 - 예외와 예외 처리

1. 파일 연결 - open() 내장 함수

◆ open() 내장 함수

파일객체 = open(file , mode)	
file	파일명
mode	<p>파일을 열 때의 모드를 의미하며, 다음 문자열 조합으로 사용함.</p> <ul style="list-style-type: none">• r : 읽기 모드 (디폴트)• w : 쓰기 모드• a : 쓰기 + 이어쓰기 모드 (append)• r+ : 읽기와 쓰기를 모두 하고자 할 때

1. 파일 연결 - open() 내장 함수

◆ file open 후에 사용할 수 있는 메소드

읽기	read()	read() - 파일 내용을 모두 읽어서 문자열(str)로 반환한다.
		read(n) - 파일에서 n 바이트 읽어서 문자열(str)로 반환한다.
	readline()	한 줄씩 읽어서 문자열(str)로 반환한다.
	readlines()	파일 전체를 리스트(list)로 반환한다.
쓰기	write()	문자열을 파일에 저장한다.
	writelines()	문자열 리스트를 파일에 저장한다.

2. 파일 입력

◆ read()와 read(n)

```
f = open('newfile.txt', 'r')
```

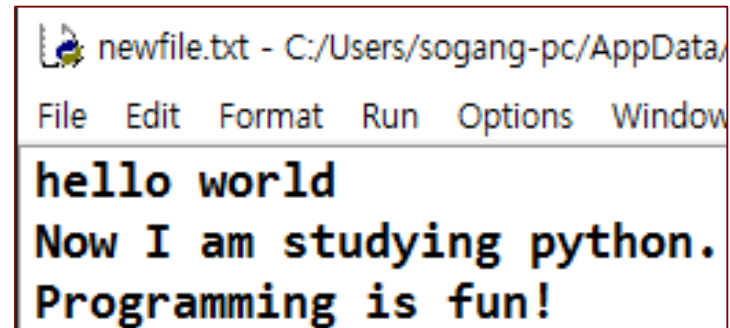
```
a = f.read(4)
```

```
print(a)
```

```
b = f.read()
```

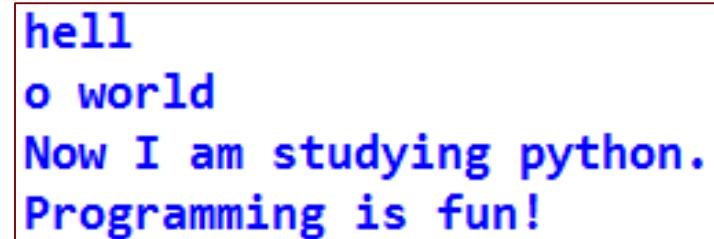
```
print(b)
```

```
f.close()
```



A screenshot of a text editor window titled 'newfile.txt - C:/Users/sogang-pc/AppData/...'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', and 'Window'. The text content is as follows:

```
hello world
Now I am studying python.
Programming is fun!
```



A screenshot showing the output of the Python code. The text is displayed in blue font:

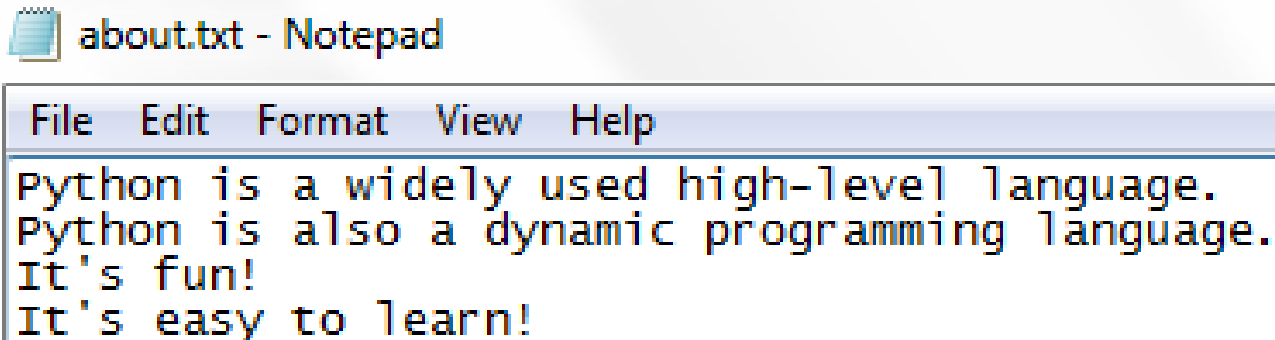
```
hell
o world
Now I am studying python.
Programming is fun!
```

2. 파일 입력

◆ 한 줄씩 읽어 오기 - readline()

```
f = open('about.txt', 'r')
a = f.readline()
print(a)
b = f.readline()
print(b)
f.close()
```

```
Python is a widely used high-level language.
Python is also a dynamic programming language.
>>>
```



about.txt - Notepad

File Edit Format View Help

Python is a widely used high-level language.
Python is also a dynamic programming language.
It's fun!
It's easy to learn!

2. 파일 입력

◆ 한 줄씩 읽어 오기 - for 구문 이용하기

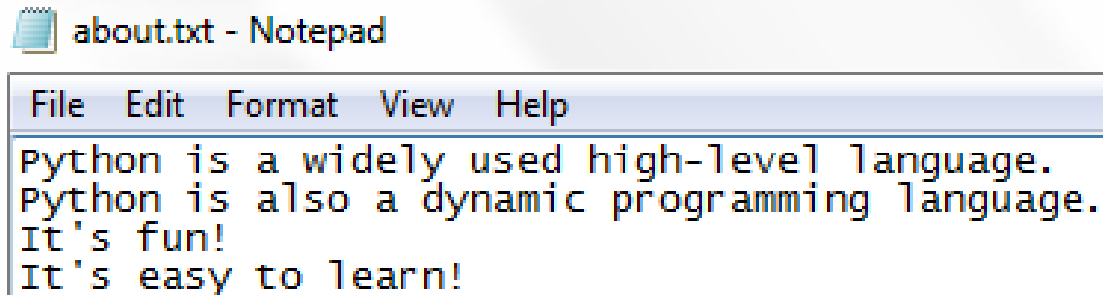
```
f = open('about.txt', 'r')
for line in f:
    print(line)
f.close()
```

```
Python is a widely used high-level language.

Python is also a dynamic programming language.

It's fun!

It's easy to learn!
```



about.txt - Notepad

File Edit Format View Help

Python is a widely used high-level language.
Python is also a dynamic programming language.
It's fun!
It's easy to learn!

2. 파일 입력

- ◆ `readlines()` - 파일 전체를 **리스트(list)**로 반환한다.

```
f = open('about.txt', 'r')
```

```
a = f.readlines()
```

```
print(a)
```

```
f.close()
```

```
['Python is a widely used high-level language.\n', 'Python is also a dynamic programming language.\n', 'It's fun!\n', 'It's easy to learn!\n']
```



about.txt - Notepad

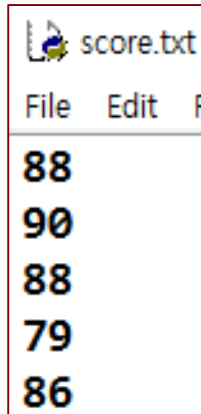
File Edit Format View Help

```
Python is a widely used high-level language.  
Python is also a dynamic programming language.  
It's fun!  
It's easy to learn!
```


2. 파일 입력

◆ 파일 내용을 통째로 읽어 들여 한 줄씩 리스트에 저장 - 1

< 입력 파일 > 한 줄에 정수 한 개씩 저장되어 있음



< 출력 결과 >

[88, 90, 88, 79, 86]

```
f = open('score.txt', 'r')
score = []
for line in f:
    score.append(int(line))
print(score)
```

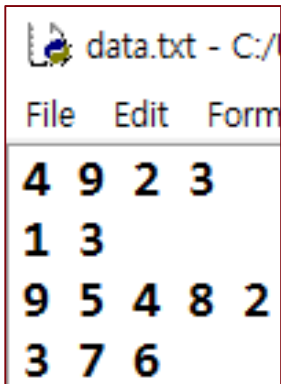
```
score = []
with open('score.txt') as f:
    for line in f:
        score.append(int(line))
print(score)
```

파일명

2. 파일 입력

◆ 파일 내용을 통째로 읽어 들여 한 줄씩 리스트에 저장 - 2

< 입력 파일 > 한 줄에 여러 개의 정수가 저장되어 있음



A screenshot of a text editor window titled 'data.txt - C:/'. The window has a menu bar with 'File', 'Edit', and 'Form'. The text content is as follows:

4	9	2	3
1	3		
9	5	4	8 2
3	7	6	

< 출력 결과 >

`[[4, 9, 2, 3], [1, 3], [9, 5, 4, 8, 2], [3, 7, 6]]`

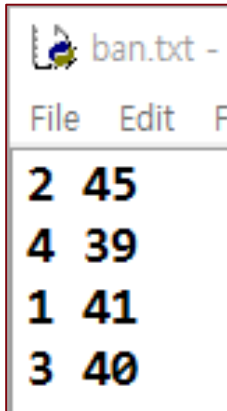
```
f = open('data.txt', 'r')
L = []

for line in f:
    L.append([int(x) for x in line.split()])
print(L)
```

2. 파일 입력

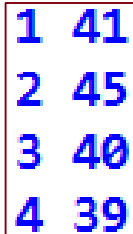
◆ 파일 내용을 통째로 읽어 들여 한 줄씩 사전에 저장 - 3

< 입력 파일 > 한 줄에 정수가 두 개씩 저장되어 있음



ban.txt -
File Edit F
2 45
4 39
1 41
3 40

< 출력 결과 >



1 41
2 45
3 40
4 39

```
D = {}  
  
with open('ban.txt') as f:  
    for line in f:  
        (key, val) = line.split()  
        D[int(key)] = val  
  
for key, val in D.items():  
    print(key, val)
```

2. 파일 입력

◆ 파일 내용을 통째로 읽어 들여 한 줄씩 사전에 저장 - 4

< 입력 파일 > 한 줄에 임의의 개수의 다양한 자료형이 저장되어 있음

```
2 Alice Paul David Bob
4 Cindy Stella Bill
1 Henry Jenny Jessica Erin Tim
3 John Joe Tom
```

```
{'1': ['Henry', 'Jenny', 'Jessica', 'Erin', 'Tim'], '3': ['John', 'Joe', 'Tom'],
 '2': ['Alice', 'Paul', 'David', 'Bob'], '4': ['Cindy', 'Stella', 'Bill']}
```

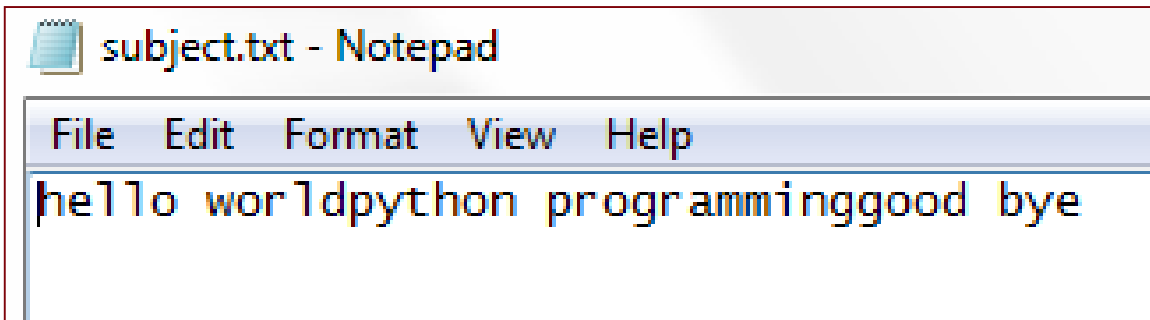
```
D = {}
```

```
f = open('ban_student.txt', 'r')
for line in f:
    items = line.split()
    key, values = items[0], items[1:]
    D[key] = values
print(D)
```

3. 파일 출력

◆ write() 예제

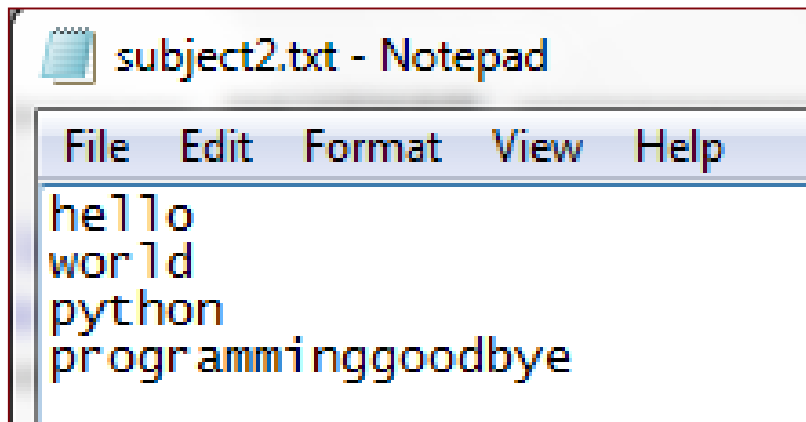
```
f = open("subject.txt", "w")  
f.write('hello world')  
f.write('python programming')  
f.write('good bye')  
f.close()
```



3. 파일 출력

◆ writelines() 예제

```
f = open("subject2.txt", "w")  
f.writelines(['hello\n', 'world\n', 'python\n',  
              'programming', 'good', 'bye'])  
f.close()
```



4. 예외 처리

Exception(예외)

REPL

```
>>> list = []
```

```
>>> list[0]
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
IndexError: list index out of range
```

빈 리스트에서 값을 가져오려
하자 IndexError 색인 오류가
발생하였다.

```
>>> text = 'abc'
```

```
>>> number = int(text)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ValueError: invalid literal for int() with base 10: 'abc'
```

문자열 abc를 정수로 변형하려
하자 ValueError 값 오류가 발
생하였다.

4. 예외 처리

코드 try_except.py

```
text = '100%'

try:
    number = int(text)  # text를 정수형 변수로 바꾸어 number에 넣는다
except ValueError:     # ValueError가 발생하면
    print('{}는 숫자가 아니에요'.format(text))  # 출력한다
```

100%는 숫자가 아니에요

4. 예외 처리

try: —————→ try 블록의 코드를 실행한다.

코드1
코드2
...

except (예외): —————→ 만약 조건으로 걸어둔 예외가 발생하면

코드a
코드b
...

—————→ 이쪽 블록의 코드를 실행한다.

4. 예외 처리

코드

```
def safe_pop_print(list, index):  
    try:  
        print(list.pop(index)) # index에 해당하는 값을 지우면서 출력합니다.  
    except IndexError:  
        print('{} index의 값을 가져올 수 없습니다.'.format(index))  
  
safe_pop_print([1, 2, 3], 5) # [1, 2, 3]에서 5번째 값을 출력하고 지우도록 명령을 내립니다.
```

5 index의 값을 가져올 수 없습니다.

4. 예외 처리

코드

```
def safe_pop_print(list, index):  
    if index < len(list):          # index가 list 길이보다 작으면  
        print(list.pop(index))    # 출력한다.  
    else:                          # 그렇지 않으면 오류 메시지를 출력한다.  
        print('{} index의 값을 가져올 수 없습니다.'.format(index))  
  
safe_pop_print([1,2,3], 5)
```

5 index의 값을 가져올 수 없습니다.

`try-except`와 `if-else` 중 더 간결하고 읽기 쉬운 방법을 선택하여 코딩한다. 코드의 복잡도가 무엇을 사용하던 비슷하다면 `if`를 사용하는 것이 권장된다.

4. 예외 처리

코드

```
try:  
    import my_module  
except ImportError:  
    print('모듈이 없습니다')
```

모듈이 없습니다



잠깐만요

예외와 오류의 차이

파이썬에서 예외(exception)는 오류(error)를 처리하기 위한 수단입니다. 그래서 여러 가지 오류(값 오류, 색인 오류, 가져오기 오류 등)를 예외 문법(try-except)으로 처리합니다. 하지만 지금은 예외와 오류를 엄밀하게 구분해서 기억하지 않아도 괜찮습니다.

4. 예외 처리 – 모든 예외를 처리하고 싶을 때

코드

```
try:  
    list = []  
    print(list[0])  
  
    text = 'abc'  
  
    number = int(text)  
except:  
    print('오류가 발생했습니다.')
```

오류가 발생했습니다.

4. 예외 처리

코드

```
try:
    # list = []
    # print(list[0])
    text = 'abc'
    number = int(text)
except:
    print('오류가 발생했습니다.')
```

박스 부분을 주석처리

오류가 발생했습니다.

무슨 **오류**가 발생하였는지 모른다.

4. 예외 처리

코드

```
try:
    list = []
    print(list[0])

    text = 'abc'
    number = int(text)
except Exception as ex:
    print('오류가 발생했습니다.', ex)
```

오류가 발생했습니다. list index out of range

4. 예외 처리

코드

```
try:
    # list = []
    # print(list[0])
    # 박스 부분을 주석처리

    text = 'abc'
    number = int(text)
except Exception as ex:
    print('오류가 발생했습니다.', ex)
```

오류가 발생했습니다. invalid literal for int() with base 10: 'abc'

4. 예외 처리

모든 예외를 잡아야 하는 상황이라면 항상 Exception이라는 예외를 잡으면 된다. 하지만 상위 예외 이름을 써 하위 예외를 잡으면 의도하지 않은 예외까지 함께 잡을 수 있기 때문에 디버깅하기가 어려워질 수 있다. 따라서 의도하지 않았다면 Exception을 쓰지 않고 특정 예외를 잡는 것이 좋다.

잡고자 하는 예외의 이름을 모른다면, try-except 없이 그대로 예외를 일으켰을 때 출력되는 오류 메시지를 보면 된다.

REPL

```
>>> int('abc')
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ValueError: invalid literal for int() with base 10: 'abc'
```

4. 예외 처리

raise

특정 상황에서 직접 예외를 일으키는 명령어.

4. 예외 처리

코드 raise.py

```
def rsp(mine, yours):  
    allowed = ['가위', '바위', '보']  
    if mine not in allowed:      # allowed 안에 없다면(잘못된 값이라면)  
        raise ValueError        # 값 오류를 발생시킵니다.  
    if yours not in allowed:  
        raise ValueError  
  
rsp('가위', '바')              # 일부러 '바'라는 잘못된 값을 넣습니다.
```

Traceback (most recent call last):

File "raise.py", line 8, in <module>

rsp('가위', '바')

File "raise.py", line 6, in rsp

raise ValueError

ValueError

4. 예외 처리

코드 raise.py

(코드 줄임)

```
try:  
    rsp('가위', '바')  
except ValueError:  
    print("잘못된 값을 넣었습니다")
```

잘못된 값을 넣었습니다

4. 예외 처리

코드 raise.py

```
def rsp(mine, yours):
    allowed = ['가위', '바위', '보']
    if mine not in allowed: # allowed 안에 없다면(잘못된 값이라면)
        # 값 오류를 발생시킵니다.
        raise ValueError("'가위', '바위', '보' 가운데 하나의 값만 입력받을 수 있습니다.")
    if yours not in allowed:
        raise ValueError("'가위', '바위', '보' 가운데 하나의 값만 입력받을 수 있습니다.")

rsp('가위', '바') # '바'라는 잘못된 값을 일부러 넣습니다.
```

Traceback (most recent call last):

File "raise.py", line 8, in <module>

rsp('가위', '바')

File "raise.py", line 6, in rsp

raise ValueError

ValueError: '가위', '바위', '보' 가운데 하나의 값만 입력받을 수 있습니다.

4. 예외 처리

코드

```
classrooms = {'1반': [162, 175, 198, 137, 145, 199], '2반': [165, 177, 157, 160, 191]}

# for문으로 각 반을 순회합니다.
# class_id에는 반 이름, heights에는 학생들의 키 리스트가 들어갑니다.
for class_id, heights in classrooms.items():
    for height in heights:
        if height > 190: # 학생의 키가 190보다 크면
            print(class_id, '에 190이 넘는 학생이 있습니다') # 메시지를 출력합니다.
            break
```

2반 에 190이 넘는 학생이 있습니다
1반 에 190이 넘는 학생이 있습니다

break는 해당 반복문만을 종료하므로, 상위 for문은 종료되지 않아 '2반'으로 넘어가서 한번 더 실행된다.

4. 예외 처리

코드

```
classrooms = {'1반': [172, 185, 198, 177, 165, 199], '2반': [165, 177, 167, 180, 191]}

# for문으로 각 반을 순회합니다.
# class_id에는 반 이름, heights에는 학생들의 키 리스트가 들어갑니다.
for class_id, heights in classrooms.items():
    for height in heights:
        if height > 190: # 학생의 키가 190보다 크면
            print(class_id, '에 190이 넘는 학생이 있습니다') # 메시지를 출력합니다.
            raise StopIteration
```

raise를 이용하여
즉시 종료하였다.

2반 에 190을 넘는 학생이 있습니다

Traceback (most recent call last):

File "raise.py", line 9, in <module>

raise StopIteration

StopIteration

4. 예외 처리

코드

```
classrooms = {'1반': [172, 185, 198, 177, 165, 199], '2반': [165, 177, 167, 180, 191]}  
  
try:  
    for class_id, heights in classrooms.items():  
        for height in heights:  
            if height > 190:  
                print(class_id, '에 190이 넘는 학생이 있습니다')  
                raise StopIteration # StopIteration을 발생시켰다  
  
except StopIteration: # StopIteration이 발생했으므로  
    print('정상 종료') # 출력한다
```

1반 에 190이 넘는 학생이 있습니다

정상 종료

4. 예외 처리

4. 예외 처리

◆ 구문 에러(syntax error)

- 문법 에러

```
>>> print('hello world)
```

SyntaxError: EOL while scanning string literal

```
>>> a = 100
```

```
>>> if a > 100 ; print(a)
```

SyntaxError: invalid syntax

4. 예외 처리

◆ 예외(exception)

- 구문 에러가 없이 잘 작성된 코드라도 실행 도중에 에러가 발생할 수 있다. 이러한 잠재적인 에러를 '예외'라고 한다.

```
>>> print(x)
```

```
Traceback (most recent call last):
```

```
File "<pyshell#3>", line 1, in <module>
```

```
    print(x)
```

```
NameError: name 'x' is not defined
```

```
>>> a = 1 ; b = 'A'
```

```
>>> a + b
```

```
Traceback (most recent call last):
```

```
File "<pyshell#13>", line 1, in <module>
```

```
    a + b
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

4. 예외 처리

◆ 예외(exception)

```
>>> a = 10; b = 0
>>> a / b
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    a / b
ZeroDivisionError: division by zero
```

```
>>> L = [1,2,3]
>>> print(L[3])
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    print(L[3])
IndexError: list index out of range
```

4. 예외 처리

◆ 예외 처리

try :

< 예외 발생 가능성이 있는 문장 >

except < 예외 종류 > :

< 예외 처리 문장 >

except < 예외 종류 > :

< 예외 처리 문장 >

else :

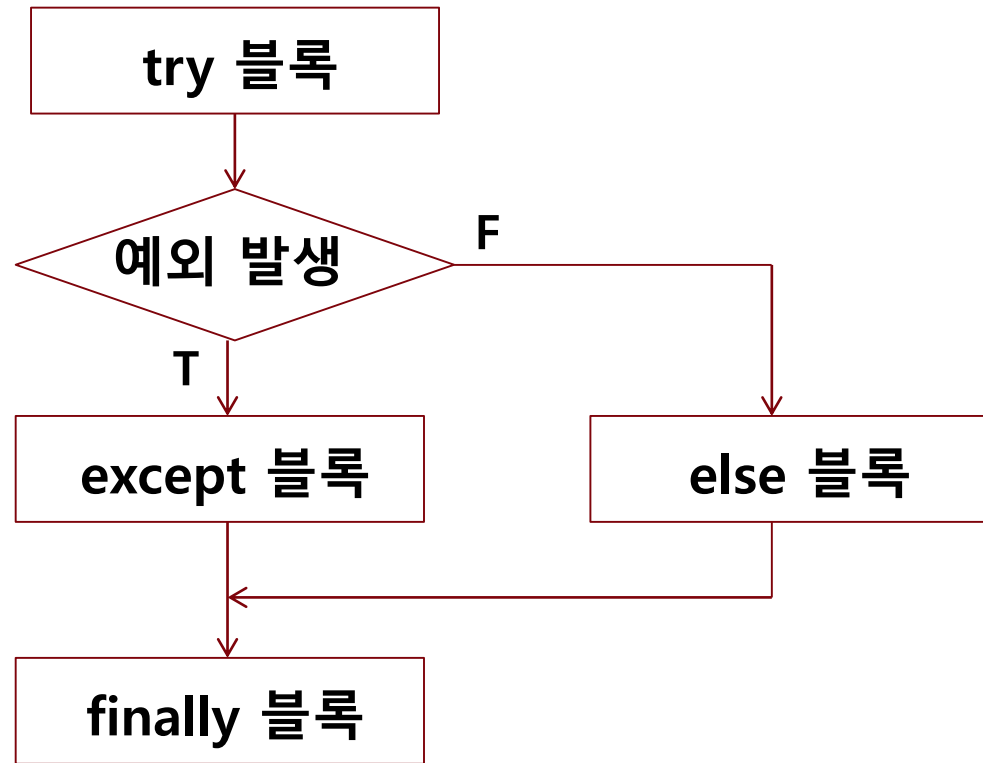
< 예외가 발생하지 않은 경우, 수행할 문장 >

finally :

< 예외 발생 유무에 상관없이 try 블록 이후 수행할 문장 >

4. 예외 처리

◆ 예외 처리



4. 예외 처리

◆ 예외 처리 예

```
a = 5
b = 0
print('a :', a)
print('b :', b)
c = a / b
print(c)
```

```
a = 5
b = 0
print('a :', a)
print('b :', b)

try:
    c = a / b
except ZeroDivisionError:
    print('cannot divide by zero')
else:
    print(c)
```

```
a : 5
b : 0
cannot divide by zero
```

4. 예외 처리

◆ 예외 처리 예

```
a = 5
b = 0
L = [1,2,3]

print('hello')

c = a / b
print(x)
print(L[3])

print('good bye')
```

```
a = 5
b = 0
L = [1,2,3]
print('hello')
```

```
try :
    c = a / b
    print(x)
    print(L[3])
except ZeroDivisionError:
    print('cannot divide by zero !!!')
except NameError:
    print('no variable named "x"')
except IndexError:
    print('out of indexing in list')
else:
    print('else part')

print('good bye')
```

```
hello
cannot divide by zero !!!
good bye
```


4. 예외 처리

◆ 예외 처리 예

```
a = 5
b = 1 <----- 수정하였음.
L = [1,2,3]
print('hello')

try :
    c = a / b
    print(x)
    print(L[3])
except ZeroDivisionError:
    print('cannot divide by zero !!!')
except NameError:
    print('no variable named "x"')
except IndexError:
    print('out of indexing in list')
else:
    print('else part')

print('good bye')
```

hello
no variable named "x"
good bye

4. 예외 처리

- ◆ 예외 발생하지 않는 경우 - **else** 가 있다면 수행함.

```
a = 5
b = 1
L = [1,2,3]
print('hello')
try :
    c = a / b
    print(L[2])
except ZeroDivisionError:
    print('cannot divide by zero !!!')
except IndexError:
    print('out of indexing in list')
else:
    print('else part')
print('good bye')
```

```
hello
3
else part
good bye
```

4. 예외 처리

◆ finally 구문이 있는 경우 - 무조건 수행되는 구문

```
L = [1,2,3]

print('hello')

try:
    print(L[3])
except IndexError:
    print('out of indexing in list')
else:
    print('else part')
finally:
    print('finally part')

print('good bye')
```

```
hello
out of indexing in list
finally part
good bye
```

4. 예외 처리

◆ finally 구문이 있는 경우 - 무조건 수행되는 구문

```
L = [1,2,3]

print('hello')

try:
    print(L[2]) <..... 수정하였음.
except IndexError:
    print('out of indexing in list')
else:
    print('else part')
finally:
    print('finally part')

print('good bye')
```

```
hello
3
else part
finally part
good bye
```

4. 예외 처리 – Syntax Error

문법(Syntax)에 맞지 않는 코드를 썼을 때 발생한다.
처음에는 파이썬 문법을 잘 몰라 발생하고, 익숙해지면 오타자 때문에 발생한다.

REPL 따옴표의 짝이 맞지 않을 때

```
>>> print('Hello, World!)  
File "<stdin>", line 1  
    print('Hello, World!)  
                ^  
SyntaxError: EOL while scanning string literal
```

REPL 사용할 수 없는 이름(예약어)을 변수 이름으로 썼을 때 ①

```
>>> True = 20  
File "<stdin>", line 1  
SyntaxError: can't assign to keyword
```

REPL 사용할 수 없는 이름(리터럴)을 변수 이름으로 썼을 때 ②

```
>>> 1 = 'text'  
File "<stdin>", line 1  
SyntaxError: can't assign to literal
```

4. 예외 처리 – Indentation Error, TabError

들여쓰기를 유지해야 하는 부분에서 들여쓰기 칸 수가 맞지 않으면 들여쓰기 (Indentation) 오류가 발생한다.

REPL 덜 들여 썼을 때

```
>>> if True:
...     print(1)
...     print(2)
      File "<stdin>", line 3
        print(2)
        ^
IndentationError: unindent does not match any outer indentation level
```

REPL `Tab`과 `Space`를 섞어 썼을 때

```
>>> if True:
...     print(1)  스페이스 4번
...     print(2)  탭 1번
      File "<stdin>", line 3
        print(2)
        ^
IndentationError: unindent does not match any outer indentation level
```

4. 예외 처리 – Name Error

변수 이름이 틀렸거나 없는 변수를 사용할 때 발생한다. 대부분 오타자 문제이다.

REPL 변수 이름이 틀렸을 때

```
>>> nama = '파이썬'
```

```
>>> print(name)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 'name' is not defined
```

4. 예외 처리 – Attribute Error

없는 속성을 가져오려 할 때 발생한다. 이름 오류와 마찬가지로 오타자 때문에 주로 발생한다.

REPL 인스턴스의 속성 이름이 잘못됐을 때 ①

```
>>> [1, 2, 3].indax(2)

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

AttributeError: 'list' object has no attribute 'indax'
```

REPL 모듈의 속성 이름이 잘못됐을 때 ②

```
>>> import math

>>> math.phi

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

AttributeError: module 'math' has no attribute 'phi'

>>>
```


4. 예외 처리 – Attribute Error

REPL 인스턴스에 없는 속성을 호출했을 때

```
>>> year_str = '2016'
>>> year_int = int(year_str)
>>> year_int.replace('6', '7')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute 'replace'
```

4. 예외 처리 – Type Error

자료형이 맞지 않거나 함수 호출 규약을 틀리면 발생한다.

REPL 실행인자 자료형이 맞지 않을 때 ①

```
>>> round('10')  
  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: type str doesn't define __round__ method
```

REPL in 연산자의 자료형이 맞지 않을 때 ②

```
>>> if 20 in 2007:  
...     print('it has 20!')  
...  
  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: argument of type 'int' is not iterable
```

4. 예외 처리 – Type Error

REPL 실행인자 개수를 잘못 넣었을 때 ①

```
>>> round()
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: Required argument 'number' (pos 1) not found
```

REPL 실행인자 개수를 잘못 넣었을 때 ②

```
>>> round(10, 2, 3)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: round() takes at most 2 arguments (3 given)
```

4. 예외 처리 – Value Error

자료형은 맞으나 값이 틀린 경우 발생한다.

REPL

```
>>> int('이천')
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ValueError: invalid literal for int() with base 10: '이천'
```

4. 예외 처리 – Unbound Local Error

변수의 스코프(scope)가 적절하지 않을 때 발생한다.

REPL

```
>>> name = '파이썬'
>>> def rename(new_name):
...     print('name:', name)
...     name = new_name
...
>>> rename('신난다')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in rename
UnboundLocalError: local variable 'name' referenced before assignment
```

4. 예외 처리 – Unbound Local Error

REPL 권장하는 코드

```
>>> name = '파이썬'
>>> def rename(new_name):
...     print('name:', name)
...     name = new_name
...
>>> name = rename('신난다')
```

REPL 어쩔 수 없이 이런 코드를 써야 할 때

```
>>> name = '파이썬'
>>> def rename(new_name):
...     global name
...     print('name:', name)
...     name = new_name
...
>>> rename('신난다')
```

4. 예외 처리 – Key Error와 Index Error

컨테이너에 참조가 없을 때 발생한다.

REPL

```
>>> d = {'a': 100}
>>> d['b']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'b'
```

REPL

```
>>> a = [1, 2, 3]
>>> a[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

4. 예외 처리 ImportError와 ModuleNotFoundError

모듈 이름이 틀렸거나 사용자가 만든 파일을 실행 경로에서 가져올 수 없을 때 주로 발생한다.

REPL

```
>>> import wrong_module  
  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ImportError: No module named 'wrong_module'
```

파이썬 버전 3.6 이상부터는 ImportError 대신 ModuleNotFoundError가 발생한다.

4. 예외 처리 RuntimeError와 RecursionError

REPL

```
>>> def factorial(n):  
...     return n * factorial(n-1)  
...  
>>> factorial(5)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "<stdin>", line 2, in factorial  
  File "<stdin>", line 2, in factorial  
  File "<stdin>", line 2, in factorial  
  [Previous line repeated 995 more times]  
RecursionError: maximum recursion depth exceeded
```

4. 예외 처리 – ZeroDivisionError

숫자를 0으로 나누려고 하면 발생한다.

REPL

```
>>> def inverse(x):  
...     return 1.0 / x  
  
...  
>>> inverse(1)  
1.0  
>>> inverse(0)  
  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
    File "<stdin>", line 2, in inverse  
ZeroDivisionError: float division by zero
```