

File IOs in Python

파일 생성하기

```
f = open("새파일.txt", 'w')  
f.close()
```

파일 객체 = open(파일 이름, 파일 열기 모드)

파일 열기 모드에는 다음과 같은 것들이 있다.

파일열기모드	설명
r	읽기모드 - 파일을 읽기만 할 때 사용
w	쓰기모드 - 파일에 내용을 쓸 때 사용
a	추가모드 - 파일의 마지막에 새로운 내용을 추가 시킬 때 사용

만약 새파일.txt라는 파일을 C:/Python 이라는 디렉터리에 생성하고 싶다면 다음과 같이 작성해야 한다.

```
f = open("C:/Python/새파일.txt", 'w')  
f.close()
```

파일을 쓰기 모드로 열어 출력값 적기

위의 예에서는 파일을 쓰기 모드로 열기만 했지 정작 아무것도 쓰지는 않았다. 이번에는 에디터를 열고 프로그램의 출력값을 파일에 직접 써 보자.

```
# writedata.py
f = open("C:/Python/새파일.txt", 'w')
for i in range(1, 11):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

파일에 새로운 내용 추가하기

```
# adddata.py
f = open("C:/Python/새파일.txt", 'a')
for i in range(11, 20):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

프로그램의 외부에 저장된 파일을 읽는 여러 가지 방법

readline() 함수 이용하기

첫 번째 방법은 readline() 함수를 이용하는 방법이다. 다음의 예를 보자.

```
# readline.py
f = open("C:/Python/새파일.txt", 'r')
line = f.readline()
print(line)
f.close()
```

만약 모든 라인을 읽어서 화면에 출력하고 싶다면 다음과 같이 작성하면 된다.

```
# readline_all.py
f = open("C:/Python/새파일.txt", 'r')
while True:
    line = f.readline()
    if not line: break
    print(line)
f.close()
```

readlines() 함수 이용하기

두 번째 방법은 readlines() 함수를 이용하는 방법이다. 다음의 예를 보자.

```
f = open("C:/Python/새파일.txt", 'r')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

readlines() 함수는 파일의 모든 라인을 읽어서 각각의 줄을 요소로 갖는 리스트로 리턴한다. 따라서 위의 예에서 lines는 ["1 번째 줄입니다.\n", "2 번째 줄입니다.\n", ..., "10 번째 줄입니다.\n"] 라는 리스트가 된다. f.readlines()에서 f.readline()과는 달리 s가 하나 더 붙어 있음에 유의하자.

read() 함수 이용하기

세 번째 방법은 read() 함수를 이용하는 방법이다. 다음의 예를 보자.

```
f = open("C:/Python/새파일.txt", 'r')
data = f.read()
print(data)
f.close()
```

f.read()는 파일의 내용 전체를 문자열로 리턴한다. 따라서 위 예의 data는 파일의 전체 내용이다.

“with” statement

```
f = open("foo.txt", 'w')  
f.write("Life is too short, you need python")  
f.close()
```

```
with open("foo.txt", "w") as f:  
    f.write("Life is too short, you need python")
```

위와 같이 with문을 이용하면 with 블록을 벗어나는 순간 열린 파일 객체 f가 자동으로 close되어 편리하다. (※ with구문은 파이썬 2.5부터 지원됨)

Reading from a File: txt file

pi_digits.txt 3.1415926535
 8979323846
 2643383279

file_reader.py with open('pi_digits.txt') as file_object:
 contents = file_object.read()
 print(contents)



3.1415926535
8979323846
2643383279

with open('pi_digits.txt') as file_object:
 contents = file_object.read()
 print(contents.rstrip())



3.1415926535
8979323846
2643383279

Reading from a File: File Paths

- On Linux

```
with open('text_files/filename.txt') as file_object:
```

```
file_path = '/home/ehmatthes/other_files/text_files/filename.txt'  
with open(file_path) as file_object:
```

- On Window

```
with open('text_files\filename.txt') as file_object:
```

```
file_path = 'C:\Users\ehmatthes\other_files\text_files\filename.txt'  
with open(file_path) as file_object:
```

Reading from a File: Reading Line by Line

```
pi_digits.txt  3.1415926535
                8979323846
                2643383279
```

file_reader.py

```
❶ filename = 'pi_digits.txt'

❷ with open(filename) as file_object:
❸     for line in file_object:
        print(line)
```



```
3.1415926535
8979323846
2643383279
```

```
filename = 'pi_digits.txt'

with open(filename) as file_object:
    for line in file_object:
        print(line.rstrip())
```



```
3.1415926535
8979323846
2643383279
```

Reading from a File: Making a List of Lines from a File

```
filename = 'pi_digits.txt'
```

```
with open(filename) as file_object:
```

❶

```
    lines = file_object.readlines()
```

❷

```
for line in lines:  
    print(line.rstrip())
```



```
3.1415926535  
8979323846  
2643383279
```

```
filename = 'pi_digits.txt'
```

```
with open(filename) as file_object:  
    lines = file_object.readlines()
```

❶

```
pi_string = ''
```

❷

```
for line in lines:  
    pi_string += line.rstrip()
```

❸

```
print(pi_string)  
print(len(pi_string))
```



```
3.1415926535 8979323846 2643383279  
36
```

pi_million_digits.txt having 1000000 decimal digits

pi_million_digits.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230E
019898 3809525720106548586327886593615338182796823030195203530185296899577362259941389124972177528347913151 55748572424
47802759009 9465764078951269468398352595709825822620522489407726719478268482601476990902640136394437455305068203 4962E
40907186494231961 5679452080951465502252316038819301420937621378559566389377870830390697920773467221825625996615014215
16201052652272111660396 66557309254711055785376346682065310989652691862056476931257058635662018558100729360659876486117E
26312986080998886874132604721 56951623965864573021631598193195167353812974167729478672422924654366800980676928238280689E
61190625454337213153595845068772460 29016187667952406163425225771954291629919306455377991403734043287526288896399587947E
57811196358330059408730681216028764962867 44604774649159950549737425626901049037781986835938146574126804925648798556145E
8202734209222453398562647669149055628425039127 57710284027998066365825488926488025456610172967026640765590429099456815C
32326092752496035799646925650493681836090032380929345 95889706953653494060340216654437558900456328822505452556405644824E
49887275846101264836999892256959688159205600101655256375678 56672279661988578279484885583439751874454551296563443480396E
55690347119729964090894180595343932512362355081349490043642785271 38315912568989295196427287573946914272534366941532361C
93860381017121585527266830082383404656475880405138080163363887421637140 64354955618689641122821407533026551004241048967E
21025941737562389942075713627516745731891894562835257044133543758575342698699 472547031656613991999682628247270641336222
48639688369032655643642166442576079147108699843157337496488352927693282207629472823 815374099615455987982598910937171262
16975774183023986006591481616404944965011732131389574706208847480236537103115089842799275 442685327797431139514357417221
38363185745698147196210841080961884605456039038455343729141446513474940784884423772175154334260 30669883176833100113310E
9025100158882721647450068207041937615845471231834800726293395505482395571372568402322682130124767945 226448209102356477E
064 7842645676338818807565612168960504161139039063960162022153684941092605387688714837989559999112099164 646441191856E
843271922 3223810158744450528665238022532843891375273845892384422535472653098171578447834215822327020690287232 330053E
721268099226911 0277722610254414922157650450812067717957120271802429681062037765788971669091094180744878140490755178 5
```

```
filename = 'pi_million_digits.txt'

with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ''
for line in lines:
    pi_string += line.strip()


print(pi_string[:52] + "...")
print(len(pi_string))
```

```
3.14159265358979323846264338327950288419716939937510...
1000002
```

Writing to an Empty File

```
write_message.py  filename = 'programming.txt'


❶ with open(filename, 'w') as file_object:
❷     file_object.write("I love programming.")
```



programming.txt I love programming.

```
filename = 'programming.txt'


with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
    file_object.write("I love creating new games.")
```



I love programming.I love creating new games.

```
filename = 'programming.txt'

with open(filename, 'w') as file_object:
    file_object.write("I love programming.\n")
    file_object.write("I love creating new games.\n")
```



I love programming.
I love creating new games.

Writing to an Empty File: Appending to a File

programming.txt

```
I love programming.  
I love creating new games.
```

*write_
message.py*

```
filename = 'programming.txt'  
  
❶ with open(filename, 'a') as file_object:  
❷     file_object.write("I also love finding meaning in large datasets.\n")  
     file_object.write("I love creating apps that can run in a browser.\n")
```

programming.txt

```
I love programming.  
I love creating new games.  
I also love finding meaning in large datasets.  
I love creating apps that can run in a browser.
```

Exception Handling in Python

Interpreter 방식의 언어에서
Run-time에 flexible한
상황전개를 위해서...

Exceptions

division.py

```
print(5/0)
```

Of course Python can't do this, so we get a traceback:

```
Traceback (most recent call last):  
  File "division.py", line 1, in <module>  
    print(5/0)
```

❶ `ZeroDivisionError: division by zero`

```
>>> f = open("나없는파일", 'r')
```

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

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

```
FileNotFoundError: [Errno 2] No such file or directory: '나없는파일'
```

```
>>> a = [1,2,3]
```

```
>>> a[4]
```

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

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

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

Exception Handling

- Full list of [standard built-in exceptions](https://docs.python.org/3/library/exceptions.html) (users may create their own) is listed here
<https://docs.python.org/3/library/exceptions.html>
- In the quadratic equation example, other types of exceptions may arise
 - not entering the right number of parameters (“[unpack tuple of wrong size](#)”)
 - entering an identifier instead of a number ([NameError](#))
 - entering an invalid Python expression ([TypeError](#))
 - Refer to sample code *quadratic6.py*

Exception Handling: Intuition [1/3]

```
x = 5 + "ham"
```

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

```
File "<pyshell#0>", line 1, in <module>
```

```
x = 5 + 'ham'
```

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

```
try:
```

```
    x = 5 + "ham"
```

```
except:
```

```
    print("darn it")
```

TRY

- 'try' TO EXECUTE the code below...
- May be used anywhere that keyboarduser input is required

EXCEPT

- CATCHES all ERRORS or can just catch a specific error
- May be used anywhere that keyboarduser input is required

darn = damn (비난하다)

Exception Handling: Intuition [2/3]

```
>>> try:
    x = 5 + 'ham'
except:
    pass

>>>
```



- says to IGNORE and move on
- may be used in For, While, Try/Except instances

18

```
Python 2.7.4 (default, Apr 6 2013, 19:55:15) [MSC v
64)] on win32
Type "copyright", "credits" or "license()" for more in
>>> def doesNothing():
    pass

>>> doesNothing()
```

Exception Handling: Intuition [3/3]

RAISE

- FORCE AN ERROR
to occur

```
raise TypeError("hahaha")
```

```
>>> raise TypeError("hahahaha")  
  
Traceback (most recent call last):  
  File "<pyshell#0>", line 1, in <module>  
    raise TypeError("hahahaha")  
TypeError: hahahaha  
>>>
```

finally: 절은
exception
발생여부와 관계없이
꼭 수행

```
try:  
    x = 5 + "ham"  
  
except ZeroDivisionError:  
    print("darn it")  
  
finally:  
    print("Let's go further!")
```

•Exception Handling: Syntax

```
try: <body>
except:
    <exception handling>
```

- To explicitly filter out all error types

```
try: <body>
except <error_1> :
    <exception handling>
...
except <error_n> :
    <exception handling>
```

- To explicitly filter out error types and store the error as a variable

```
try:
    <body>
except <error_1> as <variable_1> :
    <exception handling>
...
except <error_n> as <variable_n> :
    <exception handling>
```

Exception Handling: Code Example

```
# quadratic5.py
#     A program that computes the real roots of a quadratic equation.
#     Illustrates exception handling to avoid crash on bad inputs

import math

def main():
    print("This program finds the real solutions to a quadratic\n")

    try:
        a, b, c = eval(input("Please enter the coefficients (a, b, c): "))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\n The solutions are:", root1, root2)
    except ValueError:
        print("\n No real_number roots")
```

Using Exceptions to Prevent Crashed Situation [1/2]

```
division.py print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    ❶ first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    ❷ second_number = input("Second number: ")
    if second_number == 'q':
        break
    ❸ answer = int(first_number) / int(second_number)
    print(answer)
```

```
Give me two numbers, and I'll divide them.
Enter 'q' to quit.
```

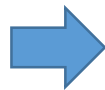


```
First number: 5
Second number: 0
Traceback (most recent call last):
  File "division.py", line 9, in <module>
    answer = int(first_number) / int(second_number)
ZeroDivisionError: division by zero
```

Using Exceptions to Prevent Crashed Situation [2/2]

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    second_number = input("Second number: ")
    ❶ try:
        answer = int(first_number) / int(second_number)
    ❷ except ZeroDivisionError:
        print("You can't divide by 0!")
    ❸ else:
        print(answer)
```



```
Give me two numbers, and I'll divide them.
Enter 'q' to quit.
```

```
First number: 5
Second number: 0
You can't divide by 0!

First number: 5
Second number: 2
2.5
```

```
First number: q
```

try .. finally

try문에는 finally절을 사용할 수 있다. finally절은 try문 수행 도중 예외 발생 여부에 상관없이 항상 수행된다. 보통 finally절은 사용한 리소스를 close해야 할 경우에 많이 사용된다.

```
f = open('foo.txt', 'w')
try:
    # 무언가를 수행한다.
finally:
    f.close()
```

foo.txt라는 파일을 쓰기 모드로 연 후에 try문이 수행된 후 예외 발생 여부에 상관없이 finally절에서 f.close()로 열린 파일을 닫을 수 있다.

여러개의 오류처리하기

try문 내에서 여러개의 오류를 처리하기 위해서는 다음과 같은 구문을 이용한다.

```
try:
    ...
except 발생 오류1:
    ...
except 발생 오류2:
    ...
```

```
try:
    a = [1,2]
    print(a[3])
    4/0
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except IndexError:
    print("인덱싱 할 수 없습니다.")
```

a는 2개의 요소값을 가지고 있기 때문에 `a[3]` 는 `IndexError` 를 발생시키므로 "인덱싱 할 수 없습니다."라는 문자열이 출력될 것이다. 인덱싱 오류가 먼저 발생했으므로 `4/0` 으로 발생하는 `ZeroDivisionError` 는 발생하지 않았다.

오류 회피하기

프로그래밍을 하다 보면 특정 오류가 발생할 경우 그냥 통과시켜야 할 때가 있을 수 있다. 다음의 예를 보자.

```
try:
    f = open("나없는파일", 'r')
except FileNotFoundError:
    pass
```

try문 내에서 FileNotFoundError가 발생할 경우 pass를 사용하여 오류를 그냥 회피하도록 한 예제이다.

오류 일부러 발생시키기

이상하게 들리겠지만 프로그래밍을 하다 보면 종종 오류를 일부러 발생시켜야 할 경우도 생긴다. 파이썬은 raise라는 명령어를 이용해 오류를 강제로 발생시킬 수 있다.

예를 들어 Bird라는 클래스를 상속받는 자식 클래스는 반드시 fly라는 함수를 구현하도록 만들고 싶은 경우(강제로 그렇게 하고 싶은 경우)가 있을 수 있다. 다음 예를 보자.

```
class Bird:
    def fly(self):
        raise NotImplementedError
```

Handling the FileNotFoundError Exception

alice.py

```
filename = 'alice.txt'

with open(filename) as f_obj:
    contents = f_obj.read()
```

Traceback (most recent call last):
File "alice.py", line 3, in <module>
 with open(filename) as f_obj:
FileNotFoundError: [Errno 2] No such file or directory: 'alice.txt'

```
filename = 'alice.txt'

try:
    with open(filename) as f_obj:
        contents = f_obj.read()
except FileNotFoundError:
    msg = "Sorry, the file " + filename + " does not exist."
    print(msg)
```

Sorry, the file alice.txt does not exist.

Working with Multiple Files and Exception Handling Routine

```
word_count.py def count_words(filename):
    """Count the approximate number of words in a file."""
    try:
        with open(filename) as f_obj:
            contents = f_obj.read()
    except FileNotFoundError:
        msg = "Sorry, the file " + filename + " does not exist."
        print(msg)
    else:
        # Count approximate number of words in the file.
        words = contents.split()
        num_words = len(words)
        print("The file " + filename + " has about " + str(num_words) +
              " words.")

filename = 'alice.txt'
count_words(filename)
```

```
def count_words(filename):
    --snip--
```

```
filenames = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt', 'little_women.txt']
for filename in filenames:
    count_words(filename)
```

```
The file alice.txt has about 29461 words.
Sorry, the file siddhartha.txt does not exist.
The file moby_dick.txt has about 215136 words.
The file little_women.txt has about 189079 words.
```