

Exception/File/Log Handling

TEAMLAB director
최성철

프로그램 사용할 때 일어나는 혼한 일 들



프로그램 사용할 때 일어나는 오류들

- 주소를 입력하지 않고 배송 요청
 - 저장도 안 했는데 컴퓨터 전원이 나감
 - 게임 아이템 샀는데 게임에서 튕김
- ➔ 예상치 못한 많은 일(예외) 들이 생김

Exception

- 1) 예상 가능한 예외
- 2) 예상이 불가능한 예외

- 발생 여부를 **사전에 인지할 수 있는 예외**
- 사용자의 잘못된 입력, 파일 호출 시 파일 없음
- 개발자가 반드시 **명시적으로 정의** 해야함

- 인터프리터 과정에서 발생하는 예외, 개발자 실수
- 리스트의 범위를 넘어가는 값 호출, 정수 0으로 나눔
- 수행 불가시 인터프리터가 자동 호출

- 예외가 발생할 경우 후속 조치 등 대처 필요

1) 없는 파일 호출 → 파일 없음을 알림

2) 게임 이상 종료 → 게임 정보 저장

프로그램 = 제품, 모든 잘못된 상황에 대처가 필요

Exception Handling

Exception Handling

- try ~ except 문법

try:

예외 발생 가능 코드

except <Exception Type>:

예외 발생시 대응하는 코드

-0으로 숫자를 나눌 때 예외처리 하기

```
for i in range(10):  
    try:  
        print(10 / i)  
    except ZeroDivisionError:  
        print("Not divided by 0")
```

- Built-in Exception: 기본적으로 제공하는 예외

Exception 이름	내용
IndexError	List의 Index 범위를 넘어갈 때
NameError	존재하지 않은 변수를 호출 할 때
ZeroDivisionError	0으로 숫자를 나눌 때
ValueError	변환할 수 없는 문자/숫자를 변환할 때
FileNotFoundError	존재하지 않는 파일을 호출할 때

<https://docs.python.org/3/library/exceptions.html>

- 예외 정보 표시하기

```
for i in range(10):  
    try:  
        print(10 / i)  
    except ZeroDivisionError as e:  
        print(e)  
        print("Not divided by 0")
```

- try ~ except ~ else

try:

예외 발생 가능 코드

except <Exception Type>:

예외 발생시 동작하는 코드

else:

예외가 발생하지 않을 때 동작하는 코드

- try ~ except ~ else

```
for i in range(10):  
    try:  
        result = 10 / i  
    except ZeroDivisionError:  
        print("Not divided by 0")  
    else:  
        print(10 / i)
```

- try ~ except ~ finally

try:

예외 발생 가능 코드

except <Exception Type>:

예외 발생시 동작하는 코드

finally:

예외 발생 여부와 상관없이 실행됨

- try ~ except ~ finally

```
try:
    for i in range(1, 10):
        result = 10 // i
        print(result)
except ZeroDivisionError:
    print("Not divided by 0")
finally:
    print("종료되었습니다.")
```

- 필요에 따라 강제로 Exception을 발생

raise <Exception Type>(예외정보)

```
while True:
    value = input("변환할 정수 값을 입력해주세요")
    for digit in value:
        if digit not in "0123456789":
            raise ValueError("숫자값을 입력하지  
않으셨습니다")
    print("정수값으로 변환된 숫자 -", int(value))
```

- 특정 조건에 만족하지 않을 경우 예외 발생

assert 예외조건

```
def get_binary_nmubmer(decimal_number):  
    assert isinstance(decimal_number, int)  
    return bin(decimal_number)  
  
print(get_binary_nmubmer(10))
```

File Handling

File system, 파일 시스템

OS에서 파일을 저장하는 트리구조 저장 체계

File from wiki

컴퓨터 등의 기기에서 의미 있는 정보를 담는 논리적인 단위
모든 프로그램은 파일로 구성되어 있고, 파일을 사용한다.

파일의 기본 체계 - 파일 vs 디렉토리

디렉토리 (Directory)

- 폴더 또는 디렉토리로 불림
- 파일과 다른 디렉토리를 포함할 수 있음

파일 (File)

- 컴퓨터에서 정보를 저장하는 논리적인 단위 (wikipedia)
- 파일은 파일명과 확장자로 식별됨 (예: hello.py)
- 실행, 쓰기, 읽기 등을 할 수 있음

- 기본적인 파일 종류로 text 파일과 binary 파일로 나눔
- 컴퓨터는 text 파일을 처리하기 위해 binary 파일로 변환시킴 (예: pyc 파일)
- 모든 text 파일도 실제로는 binary 파일,
ASCII/Unicode 문자열 집합으로 저장되어 사람이 읽을 수 있음

Binary 파일	Text 파일
<ul style="list-style-type: none">- 컴퓨터만 이해할 수 있는 형태인 이진(법)형식으로 저장된 파일- 일반적으로 메모장으로 열면 내용이 깨져 보임 (메모장 해설 불가)- 엑셀파일, 워드 파일 등등	<ul style="list-style-type: none">- 인간도 이해할 수 있는 형태인 문자열 형식으로 저장된 파일- 메모장으로 열면 내용 확인 가능- 메모장에 저장된 파일, HTML 파일, 파이썬 코드 파일 등

ASCII CODE TABLE

10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자
0	0x00	NULL	22	0x16	STN	44	0x2C	,	66	0x42	B	88	0x58	X	110	0x6E	n
1	0x01	SOH	23	0x17	ETB	45	0x2D	-	67	0x43	C	89	0x59	Y	111	0x6F	o
2	0x02	STX	24	0x18	CAN	46	0x2E	.	68	0x44	D	90	0x5A	Z	112	0x70	p
3	0x03	ETX	25	0x19	EM	47	0x2F	/	69	0x45	E	91	0x5B	[113	0x71	q
4	0x04	EOT	26	0x1A	SUB	48	0x30	0	70	0x46	F	92	0x5C	₩	114	0x72	r
5	0x05	ENQ	27	0x1B	ESC	49	0x31	1	71	0x47	G	93	0x5D]	115	0x73	s
6	0x06	ACK	28	0x1C	FS	50	0x32	2	72	0x48	H	94	0x5E	^	116	0x74	t
7	0x07	BEL	29	0x1D	GS	51	0x33	3	73	0x49	I	95	0x5F	_	117	0x75	u
8	0x08	BS	30	0x1E	RS	52	0x34	4	74	0x4A	J	96	0x60	`	118	0x76	v
9	0x09	HT	31	0x1F	US	53	0x35	5	75	0x4B	K	97	0x61	a	119	0x77	w
10	0x0A	₩n	32	0x20	SP	54	0x36	6	76	0x4C	L	98	0x62	b	120	0x78	x
11	0x0B	VT	33	0x21	!	55	0x37	7	77	0x4D	M	99	0x63	c	121	0x79	y
12	0x0C	FF	34	0x22	"	56	0x38	8	78	0x4E	N	100	0x64	d	122	0x7A	z
13	0x0D	₩r	35	0x23	#	57	0x39	9	79	0x4F	O	101	0x65	e	123	0x7B	{
14	0x0E	SO	36	0x24	\$	58	0x3A	:	80	0x50	P	102	0x66	f	124	0x7C	
15	0x0F	SI	37	0x25	%	59	0x3B	;	81	0x51	Q	103	0x67	g	125	0x7D	}
16	0x10	DLE	38	0x26	&	60	0x3C	<	82	0x52	R	104	0x68	h	126	0x7E	~
17	0x11	DC1	39	0x27	'	61	0x3D	=	83	0x53	S	105	0x69	i	127	0x7F	DEL
18	0x12	DC2	40	0x28	(62	0x3E	>	84	0x54	T	106	0x6A	j			
19	0x13	DC3	41	0x29)	63	0x3F	?	85	0x55	U	107	0x6B	k			
20	0x14	DC4	42	0x2A	*	64	0x40	@	86	0x56	V	108	0x6C	l			
21	0x15	NAK	43	0x2B	+	65	0x41	A	87	0x57	W	109	0x6D	m			

출처:

<http://sexy.pe.kr>

- 파이썬은 파일 처리를 위해 "open" 키워드를 사용함

```
f = open("<파일이름>", "접근 모드")  
f.close()
```

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

read() txt 파일 안에 있는 내용을 문자열로 반환

```
f = open("i_have_a_dream.txt", "r" )  
contents = f.read()  
print(contents)  
f.close()
```

대상파일이 같은 폴더에 있을 경우

with 구문과 함께 사용하기

```
with open("i_have_a_dream.txt", "r") as my_file:  
    contents = my_file.read()  
    print (type(contents), contents)
```

한 줄씩 읽어 List Type으로 반환함

```
with open("i_have_a_dream.txt", "r") as my_file:  
    content_list = my_file.readlines() #파일 전체를 list로 반환  
    print(type(content_list)) #Type 확인  
    print(content_list) #리스트 값 출력
```

실행 시 마다 한 줄 씩 읽어 오기

```
with open("i_have_a_dream.txt", "r") as my_file:
    i = 0
    while True:
        line = my_file.readline()
        if not line:
            break
        print (str(i) + " == " + line.replace("\n","")) #한줄씩 값 출력
        i = i + 1
```

단어 통계 정보 산출

```
with open("i_have_a_dream.txt", "r") as my_file:
    contents = my_file.read()
    word_list = contents.split(" ")
    #빈칸 기준으로 단어를 분리 리스트
    line_list = contents.split("\n")
    #한줄 씩 분리하여 리스트

print("Total Number of Characters :", len(contents))
print("Total Number of Words:", len(word_list))
print("Total Number of Lines :", len(line_list))
```

mode는 "w", encoding="utf8"

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

mode는 "a"는 추가 모드

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

os 모듈을 사용하여 Directory 다루기

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

디렉토리가 있는지 확인하기

```
if not os.path.isdir("log"):  
    os.mkdir("log")
```

최근에는 pathlib 모듈을 사용하여 path를 객체로 다룸

```
>>> import pathlib
>>>
>>> cwd = pathlib.Path.cwd()
>>> cwd
WindowsPath('D:/workspace')
>>> cwd.parent
WindowsPath('D:/')
>>> list(cwd.parents)
[WindowsPath('D:/')]
>>> list(cwd.glob("*"))
[WindowsPath('D:/workspace/ai-pnpp'), WindowsPath('D:/workspace/cs50_auto_grader'),
WindowsPath('D:/workspace/data-academy'), WindowsPath('D:/workspace/DSME-AI-Smart-
Yard'), WindowsPath('D:/workspace/introduction_to_python_TEAMLAB_M00C'),
```


1) 디렉토리가 있는지, 2) 파일이 있는지 확인 후

```
import os
if not os.path.isdir("log"):
    os.mkdir("log")
if not os.path.exists("log/count_log.txt"):
    f = open("log/count_log.txt", 'w', encoding="utf8")
    f.write("기록이 시작됩니다\n")
    f.close()

with open("log/count_log.txt", 'a', encoding="utf8") as f:
    import random, datetime
    for i in range(1, 11):
        stamp = str(datetime.datetime.now())
        value = random.random() * 1000000
        log_line = stamp + "\t" + str(value) + "값이 생성되었습니다" + "\n"
        f.write(log_line)
```

- 파이썬의 객체를 영속화(persistence)하는 built-in 객체
- 데이터, object 등 실행중 정보를 저장 ➔ 불러와서 사용
- 저장해야하는 정보, 계산 결과(모델) 등 활용이 많음

```
import pickle
```

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

```
f = open("list.pickle", "rb")  
test_pickle = pickle.load(f)  
print(test_pickle)  
f.close()
```

```
import pickle

class Mutltiply(object):
    def __init__(self, multiplier):
        self.multiplier = multiplier

    def multiply(self, number):
        return number * self.multiplier

multiply = Mutltiply(5)
multiply.multiply(10)

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)
```

Logging Handling

**게임을 만들었는데
HACK쓰는 애들 때문에 망했어요...**

어떻게 잡을 수 있을까?

일단은 기록부터!!

- 프로그램이 실행되는 동안 일어나는 정보를 기록을 남기기
- 유저의 접근, 프로그램의 Exception, 특정 함수의 사용
- Console 화면에 출력, 파일에 남기기, DB에 남기기 등등
- 기록된 로그를 분석하여 의미있는 결과를 도출 할 수 있음
- 실행시점에서 남겨야 하는 기록, 개발시점에서 남겨야하는 기록

- 기록을 print로 남기는 것도 가능함
- 그러나 Console 창에만 남기는 기록은 분석시 사용불가
- 때로는 레벨별(개발, 운영)로 기록을 남길 필요도 있음
- 모듈별로 별도의 logging을 남길필요도 있음
- 이러한 기능을 체계적으로 지원하는 모듈이 필요함

- Python의 기본 Log 관리 모듈

```
import logging

logging.debug("틀렸잖아!")
logging.info("확인해")
logging.warning("조심해!")
logging.error("에러났어!!!")
logging.critical ("망했다...")
```

- 프로그램 진행 상황에 따라 다른 Level의 Log를 출력함
- 개발 시점, 운영 시점 마다 다른 Log가 남을 수 있도록 지원함
- DEBUG > INFO > WARNING > ERROR > Critical
- Log 관리시 가장 기본이 되는 설정 정보

Level	개요	예시
debug	개발시 처리 기록을 남겨야하는 로그 정보를 남김	<ul style="list-style-type: none"> - 다음 함수로 A 를 호출함 - 변수 A 를 무엇으로 변경함
info	처리가 진행되는 동안의 정보를 알림	<ul style="list-style-type: none"> - 서버가 시작되었음 - 서버가 종료됨 - 사용자 A가 프로그램에 접속함
warning	사용자가 잘못 입력한 정보나 처리는 가능하나 원래 개발시 의도치 않는 정보가 들어왔을 때 알림	<ul style="list-style-type: none"> - Str입력을 기대했으나, Int가 입력됨 ➔ Str casting으로 처리함 - 함수에 argument로 이차원 리스트를 기대했으나 ➔ 일차원 리스트가 들어옴, 이차원으로 변환후 처리
error	잘못된 처리로 인해 에러가 났으나, 프로그램은 동작할 수 있음을 알림	<ul style="list-style-type: none"> - 파일에 기록을 해야하는데 파일이 없음 --> Exception 처리후 사용자에게 알림 - 외부서비스와 연결 불가
critical	잘못된 처리로 데이터 손실이나 더이상 프로그램이 동작할 수 없음을 알림	<ul style="list-style-type: none"> - 잘못된 접근으로 해당 파일이 삭제됨 - 사용자의 의한 강제 종료

```
import logging
```

```
logger = logging.getLogger("main")
```

```
stream_handler = logging.StreamHandler()
```

```
logger.addHandler(stream_handler)
```

Logger 선언

Logger의 output 방법 선언

Logger의 output 등록

```
logger.setLevel(logging.DEBUG)
```

```
logger.debug("틀렸잖아!")
```

```
logger.info("확인해")
```

```
logger.warning("조심해!")
```

```
logger.error("에러났어!!!")
```

```
logger.critical("망했다...")
```

```
logger.setLevel(logging.CRITICAL)
```

```
logger.debug("틀렸잖아!")
```

```
logger.info("확인해")
```

```
logger.warning("조심해!")
```

```
logger.error("에러났어!!!")
```

```
logger.critical("망했다...")
```

**실제 프로그램을 실행할 땐
여러 설정이 필요**

데이터 파일 위치 파일 저장 장소 Operation Type 등

**이러한 정보를
설정해줄 방법이 필요**

1) configparser - 파일에

2) argparse - 실행시점에

configparser

- 프로그램의 실행 설정을 file에 저장함
- Section, Key, Value 값의 형태로 설정된 설정 파일을 사용
- 설정파일을 Dict Type으로 호출후 사용

[SectionOne]
Status: Single
Name: Derek
Value: Yes
Age: 30
Single: True

Section- 대괄호

속성 - Key : Value

[SectionTwo]
FavoriteColor = Green

[SectionThree]
FamilyName: Johnson

```
import configparser
config = configparser.ConfigParser()
config.sections()

config.read('example.cfg')
config.sections()

for key in config['SectionOne']:
    print(key)

config['SectionOne']['status']
```

`'example.cfg'`

`[SectionOne]`
`Status: Single`
`Name: Derek`
`Value: Yes`
`Age: 30`
`Single: True`

`[SectionTwo]`
`FavoriteColor = Green`

`[SectionThree]`
`FamilyName: Johnson`

argparser

- Console 창에서 프로그램 실행시 Setting 정보를 저장함
- 거의 모든 Console 기반 Python 프로그램 기본으로 제공
- 특수 모듈도 많이 존재하지만(TF), 일반적으로 argparse를 사용
- Command-Line Option 이라고 부름

<https://github.com/abseil/abseil-py>

```
C:\WINDOWS\system32\cmd.exe

F:\workspace\introduction_to_pythoy_TEAMLAB_MOOC\code>python --v
Unknown option: --
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Try 'python -h' for more information.

F:\workspace\introduction_to_pythoy_TEAMLAB_MOOC\code>python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-b      : issue warnings about str(bytes_instance), str(bytearray_instance)
         and comparing bytes/bytearray with str. (-bb: issue errors)
-B      : don't write .py[co] files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd  : program passed in as string (terminates option list)
-d      : debug output from parser; also PYTHONDEBUG=x
-E      : ignore PYTHON* environment variables (such as PYTHONPATH)
-h      : print this help message and exit (also --help)
-i      : inspect interactively after running script; forces a prompt even
         if stdin does not appear to be a terminal; also PYTHONINSPECT=x
-l      : isolate Python from the user's environment (implies -E and -s)
-m mod  : run library module as a script (terminates option list)
-O      : optimize generated bytecode slightly; also PYTHONOPTIMIZE=x
-OO     : remove doc-strings in addition to the -O optimizations
-q      : don't print version and copyright messages on interactive startup
-s      : don't add user site directory to sys.path; also PYTHONNOUSERSITE
-S      : don't imply 'import site' on initialization
-u      : unbuffered binary stdout and stderr, stdin always buffered;
         also PYTHONUNBUFFERED=x
         see man page for details on internal buffering relating to '-u'
-v      : verbose (trace import statements); also PYTHONVERBOSE=x
         can be supplied multiple times to increase verbosity
```



```
import argparse
```

```
parser = argparse.ArgumentParser(description='Sum two integers.')
```

Argument	짧은 이름	긴 이름	표시명	Help 설명	Type
----------	-------	------	-----	---------	------

```
parser.add_argument('-a', "--a_value", dest="A_value", help="A integers", type=int)
```

```
parser.add_argument('-b', "--b_value", dest="B_value", help="B integers", type=int)
```

```
args = parser.parse_args()
```

```
print(args)
```

```
print(args.a)
```

```
print(args.b)
```

```
print(args.a + args.b)
```

```
def main():
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=64, metavar='N', help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N', help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=10, metavar='N', help='number of epochs to train (default: 10)')
    parser.add_argument('--lr', type=float, default=0.01, metavar='LR', help='learning rate (default: 0.01)')
    parser.add_argument('--momentum', type=float, default=0.5, metavar='M', help='SGD momentum (default: 0.5)')
    parser.add_argument('--no-cuda', action='store_true', default=False, help='disables CUDA training')
    parser.add_argument('--seed', type=int, default=1, metavar='S', help='random seed (default: 1)')
    parser.add_argument('--save-model', action='store_true', default=False, help='For Saving the current Model')

    args = parser.parse_args()

if __name__ == '__main__':
    main()
```

<https://ddiri01.tistory.com/302>

Logging 적용하기

- Log의 결과값의 format을 지정해줄 수 있음

```
formatter = logging.Formatter('%(asctime)s %(levelname)s %(process)d %(message)s')
```

```
2018-01-18 22:47:04,385 ERROR 4410 ERROR occurred
2018-01-18 22:47:22,458 ERROR 4439 ERROR occurred
2018-01-18 22:47:22,458 INFO 4439 HERE WE ARE
2018-01-18 22:47:24,680 ERROR 4443 ERROR occurred
2018-01-18 22:47:24,681 INFO 4443 HERE WE ARE
2018-01-18 22:47:24,970 ERROR 4445 ERROR occurred
2018-01-18 22:47:24,970 INFO 4445 HERE WE ARE
```

```
logging.config.fileConfig('logging.conf')  
logger = logging.getLogger()
```

```
[loggers]  
keys=root  
  
[handlers]  
keys=consoleHandler  
  
[formatters]  
keys=simpleFormatter  
  
[logger_root]  
level=DEBUG  
handlers=consoleHandler  
  
[handler_consoleHandler]  
class=StreamHandler  
level=DEBUG  
formatter=simpleFormatter  
args=(sys.stdout,)
```

Logging examples

Log handling

```
logger.info('Open file {0}'.format("customers.csv",))
try:
    with open("customers.csv", "r") as customer_data:
        customer_reader = csv.reader(customer_data, delimiter=',', quotechar='"')
        for customer in customer_reader:
            if customer[10].upper() == "USA": #customer 데이터의 offset 10번째 값
                logger.info('ID {0} added'.format(customer[0],))
                customer_USA_only_list.append(customer) #즉 country 필드가 “ USA ” 것만
except FileNotFoundError as e:
    logger.error('File NOT found {0}'.format(e,))
```

End of Document
Thank You.