새내기 파이썬

10장 파일과 예외처리

학습 목표

- 파일에서 텍스트를 읽고 쓸 수 있나요?
- 파일을 복사하면서 어떤 처리를 할 수 있나요?
- 작업 디렉토리 안에 100개의 파일을 생성할 수 있나요?
- 이미지와 같은 이진 파일을 복사할 수 있나요?
- 오류를 우아하게 처리할 수 있나요?

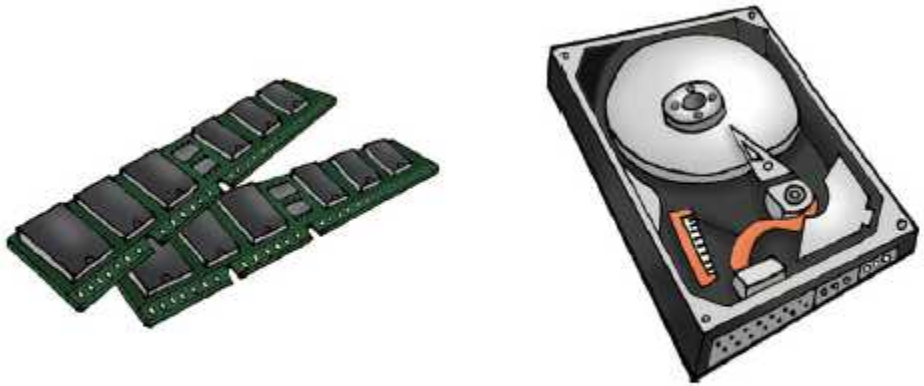
이번 장에서 만들 프로그램

```
_____
단어를 추측하시오: a
틀렸음!
9기회가 남았음!

_____
단어를 추측하시오: e
```

```
['날짜', '지점', '평균기온(℃)', '최저기온(℃)', '최고기온(℃)']
['1980-04-01', '108', '6.5', '3.2', '11.7']
['1980-04-02', '108', '6.5', '1.4', '12.9']
['1980-04-03', '108', '11.1', '4.1', '18.4']
['1980-04-04', '108', '15.5', '8.6', '21']
...
가장 추웠던 날은 –19.2 입니다.
```

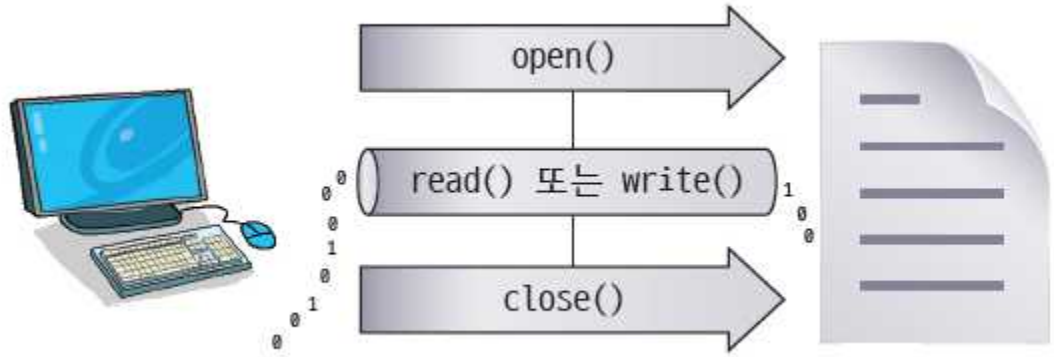
파일의 기초

- 프로그램에서 만든 데이터를 영구히 저장하고자 한다면 하드 디스크에 파일 형태로 저장하여야 한다.



파일 열고 닫기

- 파일을 사용하려면 open() 함수를 이용하여 파일을 열어야 한다. open()은 파일 이름을 받아서 파일 객체를 생성한 후에 반환한다. 파일이 열리면 우리는 파일에서 데이터를 읽거나 쓸 수 있다. 파일과 관련된 작업이 모두 종료되면 close()를 호출하여서 파일을 닫아야 한다.



파일 열고 닫기

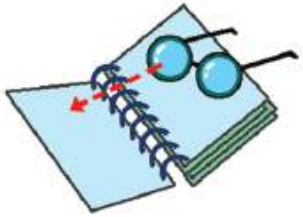
- 예를 들어서 현재 작업 디렉토리의 “input.txt" 파일을 읽기 모드로 열려면 다음과 같은 문장을 사용한다.

```
f = open("input.txt", "r")
```

파일객체 파일의 이름 파일 모드

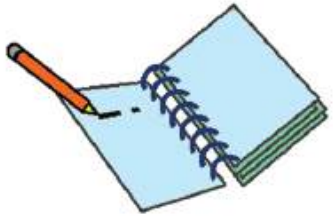
파일 모드

파일 모드	모드 이름	설명
"r"	읽기 모드(read mode)	파일의 처음부터 읽다.
"w"	쓰기 모드(write ode)	파일의 처음부터 쓴다. 파일이 없으면 생성된다. 만약 파일이 존재하면 기존의 내용은 지원진다.
"a"	추가 모드(append mode)	파일의 끝에 쓴다. 파일이 없으면 생성된다.
"r+"	읽기와 쓰기 모드	파일에 읽고 쓸 수 있는 모드이다. 모드를 변경하려면 seek()가 호출되어야 한다.



"r"

파일의 처음 부터 읽는다.



"w"

파일의 처음 부터 쓴다.
만약 파일이 존재하면 기존의
내용이 지워진다.

"a"

파일의 끝에 쓴다.
파일이 없으면 생성 된다.

파일에 쓰기

```
>>> f = open("test.txt", "w")
>>> f.write("파이썬은 강력합니다.\n")
12

>>> f.close()
```

test.txt

```
파이썬은 강력합니다.\n
```

파일에서 읽기

test.txt

```
파이썬은 강력합니다.\n
```

```
>>> f = open("test.txt", "r")
>>> s = f.read()
>>> s
'파이썬은 강력합니다.\n'
>>> f.close()
```

파일에서 읽기

test.txt

```
파이썬은 강력합니다.\n
```

- 여러 가지 문자열 연산도 가능하다.

```
>>> f = open("test.txt", "r")
>>> s = f.read()
>>> s[:3]
'파이썬'
>>> f.close()
```

파일에 추가하기

test.txt

```
파이썬은 강력합니다.\n
파이썬은 간결합니다.\n`
```

```
>>> f = open("test.txt", "a")
>>> f.write("파이썬은 간결합니다.\n")
12

>>> f.close()
>>> f = open("test.txt", "r")
>>> f.read()
'파이썬은 강력합니다.\n파이썬은 간결합니다.\n'
>>> f.close()
```

with 문으로 파일 열기

- 이 방법에서는 with 문 내의 블록이 종료될 때 파일이 자동으로 닫친다. close()를 명시적으로 호출할 필요는 없다. close() 호출은 내부적으로 이루어진다.

```
with open("test.txt", "w") as f:
        f.write("파이썬은 간결합니다.\n")

        ....                    # f.close()를 호출하지 않아도 된다.
```

파일을 연 후에는 무엇을 하는가?

test.txt

```
파이썬은 강력합니다.\n
파이썬은 간결합니다.\n
파이썬은 배우기 쉽습니다.\n
```



```
["파이썬은 강력합니다.",
"파이썬은 간결합니다.",
"파이썬은 배우기 쉽습니다."]
```

```
f = open("test.txt", "r")
s = f.read()
s = s.rstrip(' \n')

myList = s.split("\n")
print(myList)
f.close()
```

문자 인코딩

- 텍스트 파일을 처리할 때 문자 인코딩이 중요한 이유는 인코딩에 따라서 동일한 파일이라도 파일을 이루는 바이트가 달라지기 때문이다. 파이썬에서는 운영체제로부터 문자 인코딩 설정을 가져온다. 따라서 사용자가 UTF-8 기반의 파일을 열 때는 특별히 다음과 같이 인코딩을 지정하여야 한다.

```
infile = open("input.txt", "r", encoding="utf-8")
```

중간점검

1. 파일을 처리하는 절차를 설명하라.
2. 파일에서 데이터를 읽는 가장 기본적인 함수는 무엇인가?
3. 파일에 데이터를 쓰는 가장 기본적인 함수는 무엇인가?

Lab: 행맨

- 사용자는 한 번에 하나의 글자만을 입력할 수 있으며 맞으면 글자가 보이고 아니면 시도 횟수만 하나 증가한다.

```
_________
단어를 추측하시오: a
틀렸음!
9기회가 남았음!

_________
단어를 추측하시오: e
틀렸음!
8기회가 남았음!
```

Sol: 행맨

```
import random

guesses = ''
turns = 10

infile = open("words.txt", "r")
lines = infile.readlines()
word = random.choice(lines)

while turns > 0:
    failed = 0
    for char in word:
        if char in guesses:
            print(char, end="")
        else:
            print("_", end="")
            failed += 1
    if failed == 0:
        print("^사용자 승리")
        break
```

Sol: 행맨

```
    print("")
    guess = input("단어를 추측하시오: ")
    guesses += guess
    if guess not in word:
        turns -= 1
        print ("틀렸음!")
        print (str(turns)+ "기회가 남았음!")
        if turns == 0:
            print("사용자 패배 정답은 "+word)


infile.close()
```

Lab: 매출 파일 처리

- 입력 파일에 상점의 일별 매출이 저장되어 있다고 하자. 이것을 읽어서 일별 평균 매출과 총 매출을 계산한 후에 다른 파일에 출력하는 프로그램을 작성해보자.

sales.txt

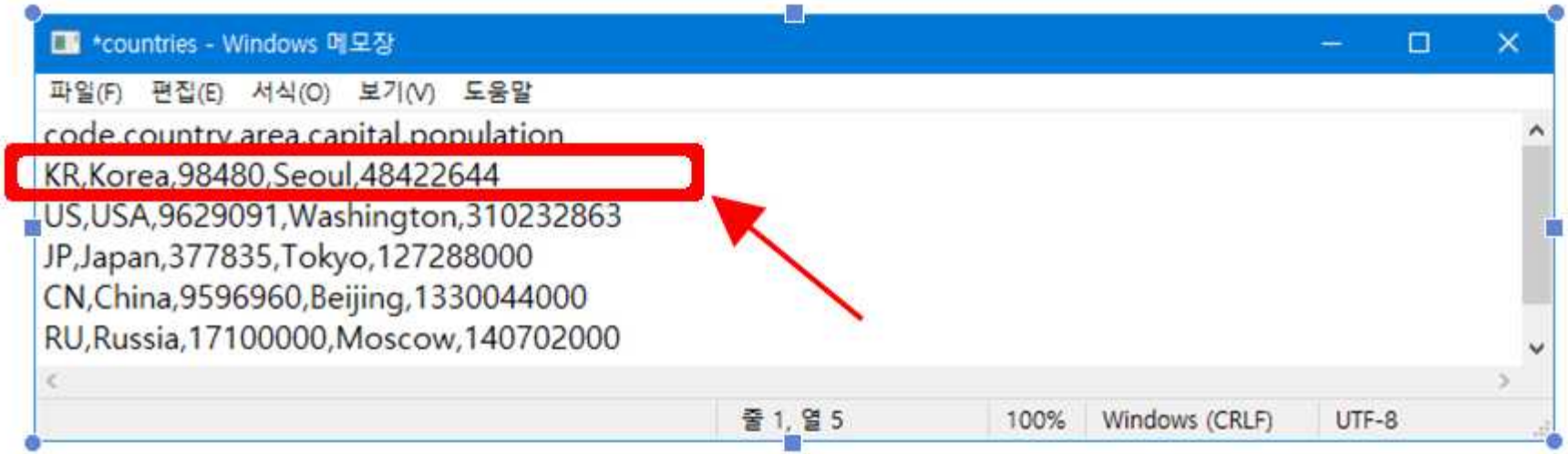
```
1000000
1000000
1000000
500000
1500000
```

summary.txt

```
총매출 = 5000000
평균 일매출 = 1000000.0
```

Lab: CVS 파일 처리

- CSV는 테이블 형식의 데이터를 저장하고 이동하는 데 사용되는 구조화된 텍스트 파일 형식이다. CSV는 Microsoft Excel와 같은 스프레드시트에 적합한 형식이다.



CSV 데이터 처리하기

- CSV 파일은 13장에서 학습하게 될 판다스를 이용해서 읽는 것이 최선이다.
- 하지만 여기서는 순수 파이썬을 이용한 처리만 살펴보자. 앞에서 파이썬 모듈 csv는 CSV reader와 CSV writer를 제공한다. 두 객체 모두 파일 핸들을 첫 번째 매개 변수로 사용한다. 필요한 경우 delimiter 매개 변수를 사용하여 구분자를 제공할 수 있다.

CVS 파일 처리

- 일단 CSV 파일을 읽는 코드를 작성해보자.

weather.csv

```
날짜,지점,평균기온(℃),최저기온(℃),최고기온(℃)
1980-04-01,108,6.5,3.2,11.7
1980-04-02,108,6.5,1.4,12.9
1980-04-03,108,11.1,4.1,18.4
1980-04-04,108,15.5,8.6,21
1980-04-05,108,15.4,12.5,18.2
1980-04-06,108,7.1,4.3,12.5
1980-04-07,108,8.5,4.7,13.3
1980-04-08,108,10.8,8.4,15.2
…
```

Sol:

```
import csv

f = open('weather.csv')                                  # CSV 파일을 열어서 f에 저장한다.
data = csv.reader(f)
for row in data:
        print(row)
f.close()
```

```
['날짜', '지점', '평균기온(°C)', '최저기온(°C)', '최고기온(°C)']
['1980-04-01', '108', '6.5', '3.2', '11.7']
['1980-04-02', '108', '6.5', '1.4', '12.9']
['1980-04-03', '108', '11.1', '4.1', '18.4']
['1980-04-04', '108', '15.5', '8.6', '21']
...
```

CVS 파일 처리

- 가장 추웠던 날의 온도를 찾아보자.

weather.csv

```
날짜,지점,평균기온(℃),최저기온(℃),최고기온(℃)
1980-04-01,108,6.5,3.2,11.7
1980-04-02,108,6.5,1.4,12.9
1980-04-03,108,11.1,4.1,18.4
1980-04-04,108,15.5,8.6,21
1980-04-05,108,15.4,12.5,18.2
1980-04-06,108,7.1,4.3,12.5
1980-04-07,108,8.5,4.7,13.3
1980-04-08,108,10.8,8.4,15.2
…
```

Sol:

```
import csv

f = open('d://weather.csv')
data = csv.reader(f)
header = next(data)
temp = 1000
for row in data:
    if temp > float(row[3]):
        temp = float(row[3])
print(temp)
f.close()
```

```
-19.2
```

Lab: 인구 데이터

- 2019년 4월 현재 우리나라의 행정구역별 인구 분포가 저장된 ages.csv 파일이 있다. 이 파일에서 서울의 인구 구조만을 추출해보자.

ages - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
행정구역,2019년04월_계_총인구수,2019년04월_계_연령구간인구수,2019년04월_계_0~9세,2019년04월_계_10~19세,2019년04월_계_20~29세,2019년04월_계_30~39세,2019년04월
전국 (0000000000),"51,836,763","51,836,763","4,252,921","5,068,750","6,815,660","7,198,993","8,438,563","8,658,999","6,070,217","3,524,973","1,567,865","220,388","19,434"
서울특별시 (1100000000),"9,766,886","9,766,886","680,098","843,949","1,458,045","1,535,973","1,579,826","1,549,531","1,159,875","667,968","247,754","37,855","6,012"
부산광역시 (2600000000),"3,431,750","3,431,750","248,549","291,745","441,953","447,924","522,444","590,864","494,077","276,664","102,489","13,323","1,718"
대구광역시 (2700000000),"2,454,154","2,454,154","192,597","245,182","328,309","313,937","399,715","431,461","297,457","166,821","69,727","8,298",650
인천광역시 (2800000000),"2,956,700","2,956,700","248,107","288,855","409,266","429,231","492,213","516,940","323,497","166,561","70,374","10,797",859
광주광역시 (2900000000),"1,458,940","1,458,940","129,244","169,087","206,651","197,954","245,937","232,778","147,936","87,809","36,134","5,032",378
대전광역시 (3000000000),"1,485,509","1,485,509","126,303","157,801","213,885","205,895","246,611","244,338","162,156","85,768","37,208","5,218",326
울산광역시 (3100000000),"1,152,293","1,152,293","106,946","119,309","151,514","164,088","192,791","211,282","128,127","54,950","20,345","2,781",160
세종특별자치시 (3600000000),"324,417","324,417","45,618","37,984","34,845","59,706","60,619","39,997","25,344","12,826","6,444",979,55
경기도 (4100000000),"13,129,508","13,129,508","1,200,597","1,373,979","1,754,439","1,940,021","2,285,992","2,170,600","1,327,051","718,326","309,784","44,976","3,743"
강원도 (4200000000),"1,540,794","1,540,794","111,278","147,872","181,271","173,485","231,350","271,670","212,545","136,287","64,455","9,832",749
충청북도 (4300000000),"1,599,488","1,599,488","132,016","157,987","200,429","203,465","246,165","273,262","198,738","118,363","60,461","8,056",546
충청남도 (4400000000),"2,125,912","2,125,912","186,524","211,490","247,176","285,662","329,783","340,549","252,582","164,559","94,159","12,616",812
전라북도 (4500000000),"1,829,273","1,829,273","141,184","187,390","215,583","204,252","278,583","305,612","236,060","162,419","85,484","11,923",783
전라남도 (4600000000),"1,873,183","1,873,183","142,949","180,002","203,878","200,761","269,034","319,797","249,472","191,037","101,881","13,581",791
경상북도 (4700000000),"2,670,375","2,670,375","204,935","241,542","296,155","315,047","395,784","468,033","371,092","235,602","124,577","16,713",895
경상남도 (4800000000),"3,368,933","3,368,933","292,679","341,522","390,403","434,824","548,317","582,708","413,205","234,962","114,687","14,926",700
제주특별자치도 (5000000000),"668,648","668,648","63,297","73,054","81,858","86,768","113,399","109,577","71,003","44,051","21,902","3,482",257
```

Windows (CRLF) Ln 1, Col 1 100%

Sol:

```
import csv
import matplotlib.pyplot as plt

f = open('d://ages.csv')
data = csv.reader(f)
header = next(data)

for row in data:
    if '서울특별시' in row[0]:
        print(row)
f.close()
```

```
['서울특별시  (1100000000)', '9,766,886', '9,766,886', '680,098', '843,949',
'1,458,045', '1,535,973', '1,579,826', '1,549,531', '1,159,875', '667,968',
'247,754', '37,855', '6,012']
```

디렉토리 작업

- 파이썬으로 쉽게 할 수 있는 작업 중의 하나가 파일이나 디렉토리를 정리하는 작업이다. 이것을 손으로 하려면 상당한 시간이 소요되고 또 아주 지루한 작업이 된다.

현재 작업 디렉토리

- 작업 디렉토리를 얻으려면 다음과 같은 함수 호출을 사용한다.

```
>>> import os
>>> os.getcwd()
'D:\\test'
>>> os.getcwdb()
b'D:\\test'
```

디렉토리 변경

- chdir() 메소드를 사용하여 현재 작업 디렉토리를 변경할 수 있다. 변경하려는 새 경로는 이 메소드에 문자열로 제공되어야 한다. 슬래시 / 또는 백슬래시 \를 모두 사용하여 경로 요소를 구분할 수 있다

```
>>> os.chdir('D:\\sources')
>>> print(os.getcwd())
D:\source
```

디렉토리 안의 파일 나열

- 디렉토리 내의 모든 파일은 listdir() 메소드를 사용하여 검색할 수 있다. 이 메소드는 경로를 가져와 해당 경로의 하위 디렉터리 및 파일 리스트를 반환한다.

```
>>> os.listdir()
['kr', 'PackageTest.java’]

>>> os.listdir('D:\\')
['$RECYCLE.BIN', '.metadata', '10.1.1.335.3398.pdf', ... ]
```

```
for filename in os.listdir() :
        print(filename)
```

작업 디렉토리에서 확장자가 ".jpg"인 파일을 전부 찾아서 파일 이름을 출력하는 프로그램

```
import os

cwd = os.getcwd()
files = os.listdir()
for name in files :
        if os.path.isfile(name) :
                if name.endswith(".jpg") :
                        print(name)
```

```
DSC04886_11.jpg
DSC04886_12.jpg
DSC04886_13.jpg
```

새 디렉토리 만들기

- mkdir() 메소드를 사용하여 새 디렉토리를 만들 수 있다. 이 메소드는 새 디렉터리의 경로를 사용한다.

```
>>> os.mkdir('test')
>>> os.listdir()
['kr', 'PackageTest.java', 'test']
```

디렉토리 또는 파일 이름 바꾸기

- rename() 메소드는 디렉터리나 파일의 이름을 바꿀 수 있다. 디렉토리나 파일의 이름을 바꾸기 위해 rename()는 두 가지 인수를 사용한다.

```
>>> os.listdir()
['kr', 'PackageTest.java', 'test']
>>> os.rename('test','test2')
>>> os.listdir()
['kr', 'PackageTest.java', 'test2']
```

디렉토리 또는 파일 제거

- remove() 메소드를 사용하여 파일을 제거할 수 있다. rmdir()는 빈 디렉터리를 제거한다. 이 rmdir()는 빈 디렉터리만 제거할 수 있다.

```
>>> os.listdir()
['kr', 'PackageTest.java', 'test2']
>>> os.remove('PackageTest.java')
>>> os.listdir()
['kr', 'test2']
>>> os.rmdir('test2')
['kr']
```

중간점검

1. 현재 작업 디렉토리에 있는 모든 파일 이름을 얻으려면 어떻게 하는가?
2. 파일과 디렉토리를 어떻게 구분하는가?

Lab: 디렉토리 안의 파일 처리

- 파일 중에서 ”Python"을 포함하고 있는 줄이 있으면 파일의 이름과 해당 줄을 출력한다.

```
file.py :        if "Python" in e:
summary.txt : The joy of coding Python should be in seeing short
summary.txt : Python is executable pseudocode.
```

Sol:

```
import os
arr = os.listdir()

for f in arr:
    infile = open(f, "r", encoding="utf-8")
    for line in infile:
        e = line.rstrip()                    # 오른쪽 줄바꿈 문자를 없앤다.
        if "Python" in e:
            print(f, ":", e)
    infile.close()
```

Lab: 수학문제지 100개 만들기

- 우리가 초등학생들을 위한 수학학원을 운영한다고 하자. 간단한 사칙 연산 문제가 10개가 들어 있는 문제지가 필요하다. 100명의 초등학생에게 서로 다른 문제지를 주고 싶다. 파이썬으로 만들 수 있을까?

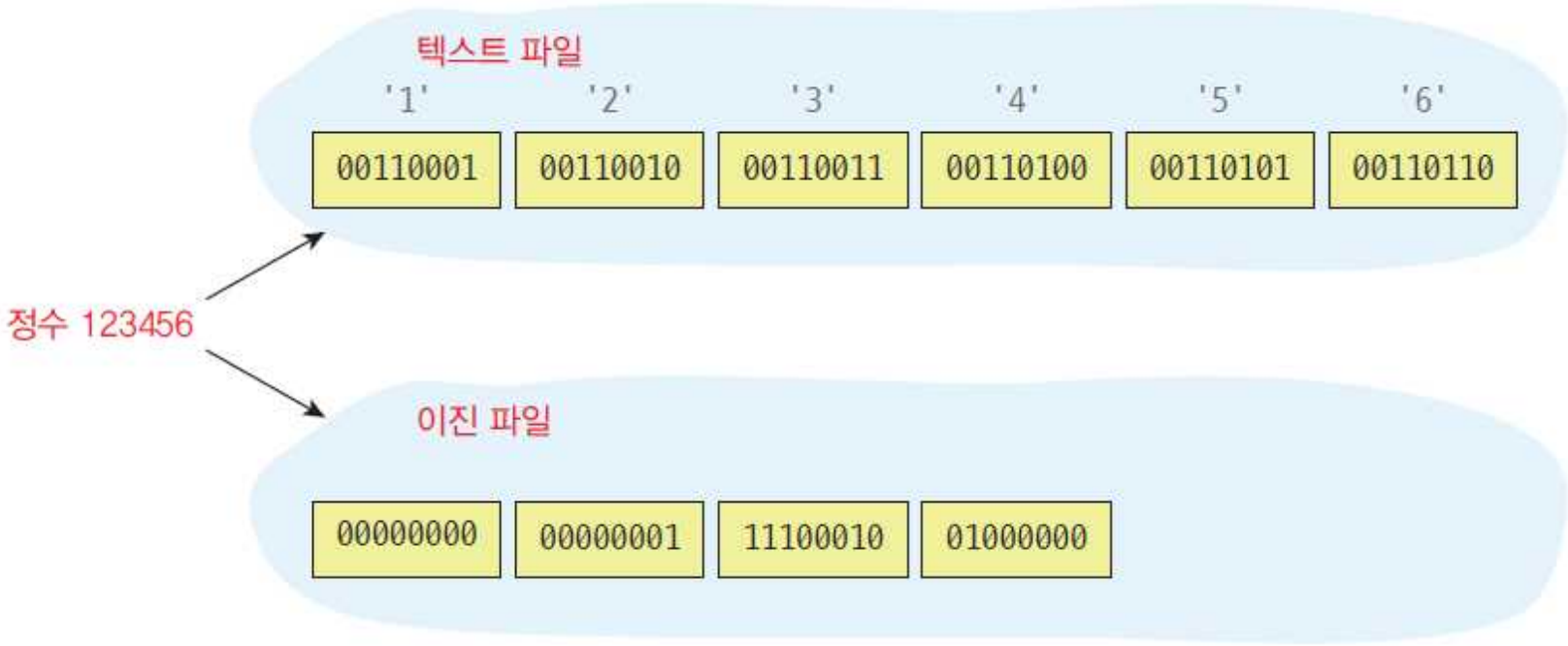
```
다음의 문제를 풀어서 제출하세요
이름:       점수:

30 - 17 =
82 + 47 =
69 * 11 =
88 / 40 =
80 / 35 =
2 / 73 =
70 * 87 =
13 * 93 =
85 - 35 =
4 + 11 =
```

Sol:

```
import random
import os

os.mkdir("math_ex")

for i in range(100):
        f = open(f"math_ex/ex{i}.txt", "w")
        f.write("다음의 문제를 풀어서 제출하세요\n")
        f.write("이름:      점수:      \n\n")

        for k in range(10):
                x = random.randint(0, 100)
                y = random.randint(0, 100)
                op = random.choice("+-*/")
                f.write(f"{x} {op} {y} = \n")
        f.close()
```

이진 파일

- 이진 파일(binary file)은 데이터가 직접 저장되어 있는 파일이다. 즉 정수 123456는 문자열로 변환되지 않고 0 1 226 64와 같은 이진수 형태로 그대로 파일에 기록되는 것이다.



이진 파일에서 읽기

이진 파일에서 데이터를 읽으려면 다음과 같이 파일을 열어야 한다.

```
>>> infile = open(filename, "rb")
```

입력 파일에서 8 바이트를 읽으려면 다음과 같은 문장을 사용한다.

```
>>> bytesArray = infile.read(8)
```

첫 번째 바이트를 꺼내려면 다음과 같은 문장을 사용하면 된다.

```
>>> byte1 = bytesArray[0]
```

이진 파일에 바이트들을 저장하려면 다음과 같이 한다.

```
>>> outfile = open(filename, "wb")
>>> bytesArray = bytes([255, 128, 0, 1])
>>> outfile.write(bytesArray)
```

Lab: 이미지 파일 복사하기

- 하나의 이미지 파일을 다른 이미지 파일로 복사하는 프로그램을 작성하여 보자.

123.png를 kkk.png로 복사하였습니다.

Sol:

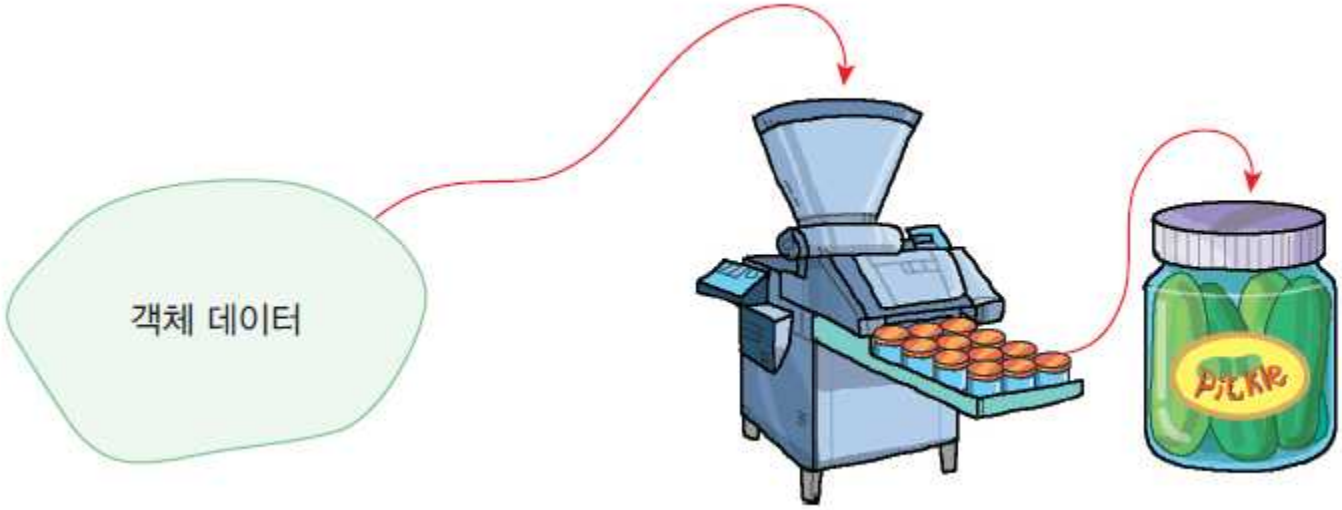
```
infile = open("123.png", "rb")
outfile = open("kkk.png", "wb")

# 입력 파일에서 1024 바이트씩 읽어서 출력 파일에 쓴다.
while True:
    copy_buffer = infile.read(1024)
    if not copy_buffer:
        break
    outfile.write(copy_buffer)

infile.close()
outfile.close()
print(filename1+"를 " +filename2+"로 복사하였습니다. ")
```

중간점검

1. 이진 파일을 열려면 어떻게 파일을 열어야 하는가? 무엇이 달라지는가?
2. 파일 포인터를 이동시키는 함수 이름은?
3. 파일 포인터를 파일의 처음으로 보내는 명령문은?

객체 입출력

- pickle 모듈의 dump()와 load() 메소드를 사용하면 객체를 쓰고 읽을 수 있다.



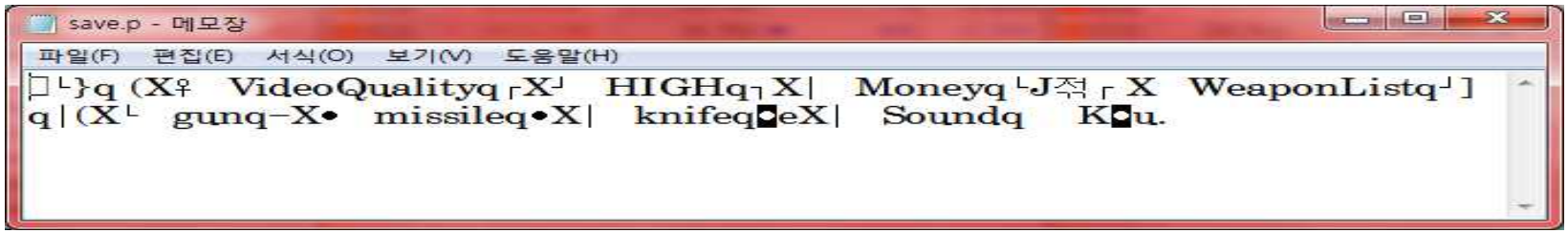
예제: 딕셔너리 저장하기

- 딕셔너리에 저장된 데이터들을 pickle 모듈을 이용하여서 파일에 기록하는 예제를 살펴보자.

```
import pickle

gameOption = {
                                "Sound": 8,
                                "VideoQuality": "HIGH",
                                "Money": 100000,
                                "WeaponList": ["gun", "missile", "knife" ]
}
```

Sol: 딕셔너리 저장하기

```
import pickle

gameOption = {
                                    "Sound": 8,
                                    "VideoQuality": "HIGH",
                                    "Money": 100000,
                                    "WeaponList": ["gun", "missile", "knife" ]
}

file = open( "save.p", "wb" )     # 이진 파일 오픈
pickle.dump( gameOption, file )            # 딕셔너리를 피클 파일에 저장
file.close()                                                # 파일을 닫는다.
```



예제: 딕셔너리 복원하기

- 이번에는 이진 파일에 저장된 딕셔너리를 읽어보자. pickle 객체의 load() 함수를 호출하면 파일에 저장된 딕셔너리를 복원할 수 있다.

```
import pickle

file = open( "save.p", "rb" )                # 이진 파일 오픈
obj = pickle.load( open( "save.p", "rb" ) )       # 피클 파일에 딕셔너리를 로딩
print(obj)
```

```
{'WeaponList': ['gun', 'missile', 'knife'], 'Money': 100000, 'VideoQuality':
'HIGH', 'Sound': 8}
```

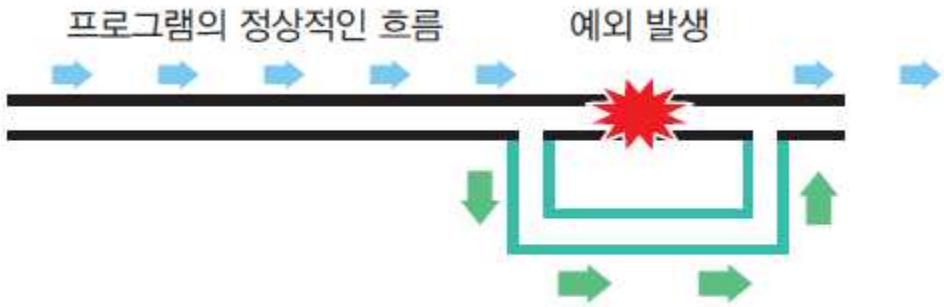
예외처리

- 사용자들은 잘못된 데이터를 입력할 수도 있고, 우리가 오픈하고자 하는 파일이 컴퓨터에 존재하지 않을 수도 있으며 인터넷이 다운될 수도 있다.

```
>>> (x, y)=(2, 0)
>>> z=x/y
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    z=x/y
ZeroDivisionError: division by zero
>>>
```

예외처리

- 오류가 발생했을 때 오류를 사용자에게 알려주고 모든 데이터를 저장하게 한 후에 사용자가 **우아하게(gracefully)** 프로그램을 종료할 수 있도록 하는 것이 바람직하다.



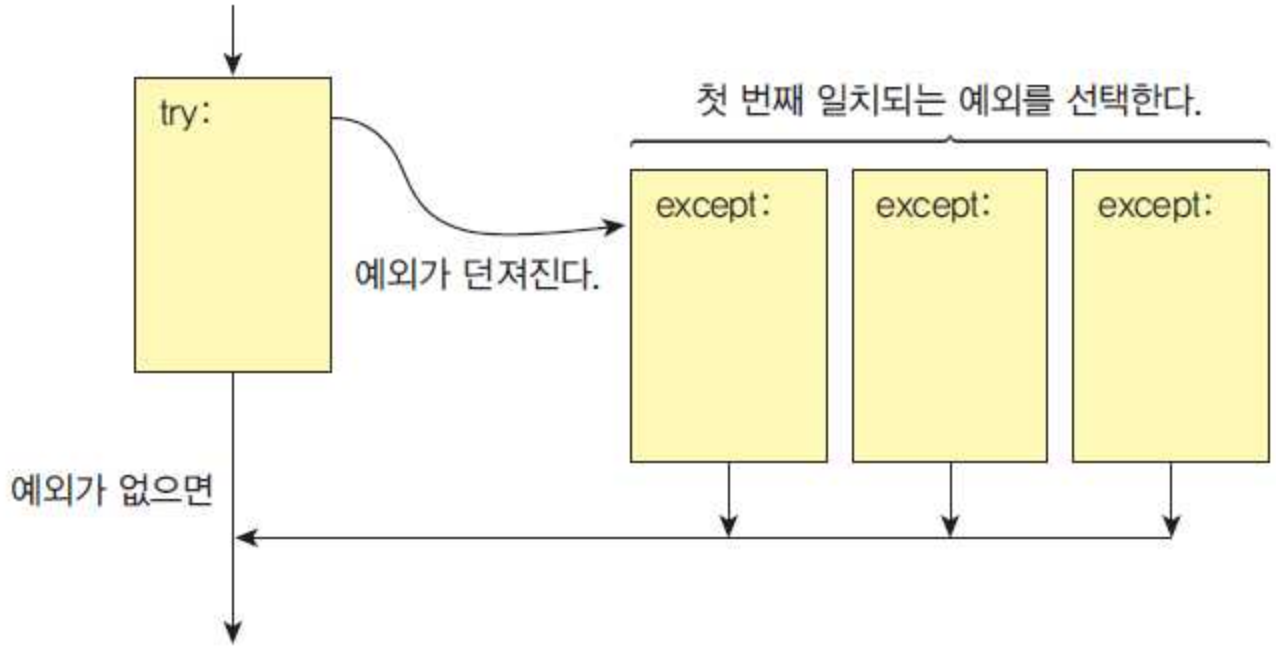
예외 처리는 프로그램의 실행을
계속할 수 있는 다른 경로를 제공한다.

오류의 종류

- 사용자 입력 오류: 사용자가 정수를 입력하여야 하는데 실수를 입력할 수 있다.
- 장치 오류: 네트워크가 안 된다거나 하드 디스크 작동이 실패할 수 있다.
- 코드 오류: 잘못된 인덱스를 사용하여서 배열에 접근할 수 있다.

- IOError: 파일을 열 수 없으면 발생한다.
- importError: 파이썬이 모듈을 찾을 수 없으면 발생한다.
- ValueError: 연산이나 내장 함수에서 인수가 적절치않은 값을 가지고 으면 발생한다.
- KeyboardInterrupt: 사용자가 인터럽트 키를 누르면 발생한다. (Control-C나 Delete)
- EOFError: 내장 함수가 파일의 끝을 만나면 발생한다.

Try-catch 구조



Try-catch 구조

Syntax

예외 처리

형식

```
try:
    예외가 발생할 수 있는 문장
except(오류):
    예외를 처리하는 문장
```

예

```
try:
    z = x/y
except ZeroDivisionError:
    print ("0으로 나누는 예외")
```

예외가 발생할 수 있는 문장

예외

예제

```
(x,y) = (2,0)
try:
    z = x/y
except ZeroDivisionError:
    print ("0으로 나누는 예외")
```

```
0으로 나누는 예외
```

```
(x,y) = (2,0)
try:
    z = x/y
except ZeroDivisionError as e:
    print (e)
```

```
division by zero
```

예제

```
>>> n = int(input("숫자를 입력하시오 : "))
숫자를 입력하시오 :  23.5
...
ValueError: invalid literal for int() with base 10: '23.5'
```

예제

- 사용자가 숫자를 입력할 때도 오류가 발생할 수 있다. 예를 들어서 정수를 받아야 되는데 사용자가 실수를 입력하면 다음과 같이 오류가 발생한다.

```
while True:
    try:
        n = input("숫자를 입력하시오 :  ")
        n = int(n)
        break
    except ValueError:
        print("정수가 아닙니다. 다시 입력하시오. ")
print("정수 입력이 성공하였습니다!")
```

```
숫자를 입력하시오 : 23.5
정수가 아닙니다. 다시 입력하시오.
숫자를 입력하시오 : 10
정수 입력이 성공하였습니다!
```

예제

- 파일을 열 때도 오류가 많이 발생한다. 파일 오류를 처리하는 문장을 작성해보면 다음과 같다.

```
try:
    fname = input("파일 이름을 입력하세요: ")
    infile = open(fname, "r")
except IOError:
    print("파일 " + fname + "을 발견할 수 없습니다.")
```

```
파일 이름을 입력하세요: kkk.py
파일 kkk.py을 발견할 수 없습니다.
```

try/execpt 블록에서의 실행 흐름

```
try:
  result = 10 / 20

except ZeroDivisionError:
  print("오류발생")
```

예외가 발생하지 않은 경우

```
try:
  result = 10 / 0

except ZeroDivisionError:
  print("오류발생")
```

예외가 발생하은 경우

중간점검

1. try 블록에 놓아야 하는 코드는 어떤 코드인가?
2. catch 블록이 하는 일은 무엇인가?

Lab: 파일 암호화

- 로마의 유명한 정치가였던 쥴리어스 시저(Julius Caesar, 100-44 B.C.)는 친지들에게 비밀리에 편지를 보내고자 할 때 다른 사람들이 알아보지 못하도록 문자들을 다른 문자들로 치환하였다.

평 문	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
암호문	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

원문: the language of truth is simple.
암호문: wkh odqjxdjh ri wuxwk lv vlpsoh.

Sol.

```
key = 'abcdefghijklmnopqrstuvwxyz'

# 평문을 받아서 암호화하고 암호문을 반환한다.
def encrypt(n, plaintext):
    result = ''
    for l in plaintext.lower():
        try:
            i = (key.index(l) + n) % 26
            result += key[i]
        except ValueError:
            result += l

    return result.lower()

f = open("test.txt", "r")
s = f.read()
s = s.rstrip()

encrypted = encrypt(3, s)                    # 3은 이동거리이다.
print ('평문: ' ,  s)
print ('암호문: ',  encrypted)
f.close()
```

Mini Project: 파일 압축

- 파이썬을 사용하면 파일을 ZIP 형식으로 압축하거나 압축해제할 수 있다. zipfile 모듈을 사용하면 개별 또는 여러 파일을 한 번에 추출하거나 압축할 수 있다. 아주 간단하다. 먼저 zipfile 모듈을 가져온 다음 두 번째 매개 변수를 'w'로 지정하여 쓰기 모드에서 ZipFile 객체를 연다. 첫 번째 매개 변수는 파일 경로이다.

```
어떤 작업을 하시겠습니까(압축 또는 해제): 압축
파일 이름을 입력하세요: test.txt
test.zip 파일로 압축되었습니다.
```

Mini Project: 파일 암호화

- cryptography 라이브러리는 파일을 암호화하는 대칭 알고리즘을 사용한다. 대칭 알고리즘에서는 동일한 키를 사용하여 파일을 암호화하고 해독한다. cryptography 패키지의 fernet 모듈에는 키 생성, 평문을 암호문으로 암호화하는 함수와 해독 메소드가 제공된다.

```
어떤 작업을 하시겠습니까(암호 또는 해제): 암호
파일 이름을 입력하세요: test.txt
test.bin 파일로 암호화되었습니다.
```

이번 장에서 배운 것

- 파일을 읽을 때는 파일을 열고, 데이터를 읽은 후에, 파일을 닫는 절차가 필요하다.
- 파일을 열기 위해서는 open() 함수에서 파일 이름을 지정하고 읽기를 의미하는 “r"을 전달한다. 파일에서 데이터를 읽는 함수는 read() 함수이다. 모든 작업이 끝나면 close() 함수로 파일을 닫아야 한다.
- 파일에 데이터를 쓰기 위해서는 open() 함수에서 파일 이름을 지정하고 쓰기를 의미하는 “w"을 전달한다. 파일에 데이터를 쓰는 함수는 write() 함수이다. 모든 작업이 끝나면 close() 함수로 파일을 닫아야 한다.
- 파일 모드에서 “r”, “w”, “a”가 있다. 각각 읽기모드, 쓰기모드, 추가모드를 의미한다.
- 파일은 텍스트 파일과 이진 파일로 나누어진다. 텍스트 파일을 사람이 내용을 볼 수 있는 파일이다. 이진 파일에는 이진 데이터가 저장되어 있다. 이진 파일은 사람이 내용을 볼 수 없다.
- 텍스트 파일에서 한 줄을 읽으려면 for 루프를 사용한다. readline() 함수를 사용하면 한 줄씩 읽을 수 있다.
- 예외 처리는 오류가 발생했을 때 프로그램을 우아하기 종료하는 방법이다. try 블록과 except 블록으로 이루어진다.

Q & A