Python 5

Exception / File / Log handling

Exception handling

- 예상 가능한 예외
  - 사전에 인지 가능
  - ex. 사용자의 잘못된 입력, 파일 호출 시 파일 없음
  - 개발자가 반드시 명시적으로 정의
- 예상 불가능한 예외
  - 인터프리터 과정에서 발생, 개발자 실수
  - ex. 리스트 범위를 넘어가는 값 호출, divided by 0
  - 수행 불가 시 인터프리터가 자동 호출
- Exception handling
  - try ~ except

```
a = [1, 2, 3, 4, 5]
for i in range(10):
  try:
    print(10 / i)
    print(a[i])
    print(v)
  except ZeroDivisionError:
    print("Error")
    print("Not divided by 0")
  except IndexError as e:
    print(e)
  # 전체 exception을 잡아버리면 error가 정확하게 어디서 발생했는지 찾기 힘들다
  except Exception as e:
    print(e)

  # else: 예외가 발생하지 않을 시 실행됨
  # finally: 예외 발생 여부와 상관없이 실행됨
```

> Built-in exceptions: IndexError, NameError, ZeroDivisionError, ValueError, FileNotFoundError

  - raise

```
while True:
  value = input("Input a number:")
  for digit in value:
    if digit not in "0123456789":
      raise ValueError("Not a digit")
  print("int(number) -", int(value))
```

- assert

```
def get_binary_number(decimal_number):
  assert isinstance(decimal_number, int)
  # 특정 조건(isinstance)을 만족하지 않을 경우 예외 발생시킴
  return bin(decimal_number)

print(get_binary_number(10))
```

File handling

- File: 의미 있는 정보를 담는 논리적 단위. 기본적인 파일 종류로 text file, binary file이 있음.
- File system: OS에서 파일을 저장하는 트리구조 저장 체계
- python file I/O
  - r (읽기 모드), w (쓰기 모드), a (추가 모드, 파일 마지막에 새로운 내용 추가)

```
f = open("sample.txt", "r") # 주소 연결
contents = f.read()
print(contents)
f.close()

with open("sample.txt", "r", encoding="utf8") as f:
  contents = f.read()
  print(contents)
```

  - readlines(): 파일 전체를 list로 반환
  - 실행 시 마다 한 줄씩 읽어오기

```
with open("sample.txt", "r") as f:
  i = 0
  while True:
    line = f.readline()
      if not line:
        break
      print(str(i) + " === " + line.replace("\n", ""))
      i += 1
```

- os module

```
import os
try:
  os.mkdir("folder")
except FileExistsError as e:
  print("Already created")

os.path.exists("folder")
os.path.isfile("file.py")
```

- shell util module

```
import shutil

source = "sample.txt"
dest = os.path.join("folder", "file.py")

shutil.copy(source, dest) # 파일 복사
```

- pathlib module

```
import pathlib

# path를 객체로 다룰 수 있음
cwd = pathlib.Path.cwd()
cwd.parent
list(cwd.glob("*"))
```

- create log file

```
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("Log starts\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) + "created log" + "\n"
    f.write(log_line)
```

- pickle
  - python 객체를 persistent(영속화)하게 하는 built-in 객체

- data, object 등 실행 중 (메모리에 있는) 정보/계산결과(모델) 등 저장

```
import pickle

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

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

Log handling

- logging
  - 프로그램이 실행되는 동안 일어나는 정보를 기록
  - 유저 접근, 프로그램 exception, 특정 함수 사용 등
  - console 화면에 출력, 파일 남기기, db에 남기기 등
- logging module

```
import logging

logging.debug() # 개발시 처리 기록
logging.info() # 처리가 진행되는 동안의 정보 알림
logging.warning() # 사용자 wrong input 등 발생 시 알림
logging.error() # 잘못된 처리로 에러 발생 시 (프로그램 동작에는 문제 없음)
logging.critical() # 잘못된 처리로 데이터 손실이나 프로그램 동작 불가 시 알림
```

  - 사전 설정: 데이터 파일 위치, 파일 저장 장소, operation type 등
- configparser
  - 프로그램의 실행 설정을 file에 저장
  - section, key, value 값의 형태로 설정된 설정 파일 사용
  - 설정파일을 dict type으로 호출 후 사용
  - key: value 혹은 key=value 형식으로 parse
- argparser
  - console 창에서 프로그램 실행 시 setting 정보 저장
  - 대부분의 console 기반 python program은 기본으로 제공
  - command-line option
  - “-v”, “-l” 등 실행 시 옵션 추가
- logging formmater
  - formatter = logging.Formatter(‘%(asctime)s %(levelname)s %(process)d %(message)s’)
- log config file

[loggers]
keys=root

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter

Python data handling

CSV (comma separate value)

- 엑셀 양식의 데이터를 프로그램에 상관없이 쓰기 위한 데이터 형식
- TSV(tab), SSV(space)
- open(), readline() 으로 불러와 처리도 가능하지만 한계가 있다
- csv 객체 활용 (import csv)
    - csv.reader(file), for row in csv_data
    - delimiter='\t'

Web

- WWW (World Wide Web)
    - HTTP protocal: 데이터 송수신
    - HTML 형식: 데이터 표시
    - 요청(웹주소, form, header) -> 처리(db) -> 응답(html, xml) -> 렌더링(html, xml)
- HTML
    - 웹 상의 정보를 구조적으로 표현
    - tag 형태로 구성
    - 트리 모양의 포함관계
    - 일반적으로 html 소스파일을 컴퓨터가 다운로드 후 웹 브라우저가 해석
    - 태그 특징을 이용해 정규식으로 파싱 가능
        - regular expression (정규식)

          regexp, regex

          복잡한 문자열 패턴을 정의하는 문자 표현 공식

          특정한 규칙을 가진 문자열 집합을 추출

          regexr.com
    - urllib, re

```
import urllib.request
import re

url = "https://..."
html = urllib.request.urlopen(url)
html_contents = str(html.read().decode("ms929"))

stock_results = re.findall("(\dl class=\"blind\">)([\s\S])...", html_contents)
```

- XML (eXtensible Markup Language)
  - 데이터의 구조의 의미를 설명하는 tag (MarkUp)를 사용해 표시하는 언어
  - 정보의 구조에 대한 정보인 스키마와 DTD 등으로 메타정보가 표현됨
  - 용도에 따라 다양한 형태로 변경가능
  - pc to mobile, mobile to pc 정보 교류에 매우 유용한 저장 방식
  - html과 같은 구조적 markup 언어
  - 정규표현식으로 parsing 가능하지만, beautifulsoup 등 편리한 parser 존재

```
from bs4 import BeautifulSoup

with open("~.xml", "r", encoding="utf8") as file:
  file_xml = file.read()

soup = BeautifulSoup(file_xml, "lxml") # lxml parser로 데이터 분석

for info in soup.find_all("search"):
  print(info)
  print(info.get_text())
```

- JSON (JavaScript Object Notation)
  - javascript의 데이터 객체 표현 방식
  - 간결하고 데이터 용량이 작고 code로 전환이 쉬움
  - {“key”:”value”} 형태로 dict type과 상호 호환
  - import json (json.loads())