logging

: 현재 프로그램이 어떤 상태를 가지고 있는지 출력하는 모듈

직접 로깅을 생성하거나 기존에 있던 라이브러리를 불러올 수 있습니다.

Level

Aa level	설명	숫자 값
DEBUG	간단히 문제를 진단하고 싶을 때 필요한 정보를 자세하게 기록	10
INFO	계획대로 작동하고 있음을 알리는 확인 메시지	20
WARNING	소프트웨어가 작동 하고 있지만, 예상치 못한 일이 발생했거나 예측하는 알림	30
ERROR	소프트웨어가 몇몇 기능들을 수행하지 못함을 알림	40
CRITICAL	작동이 불가능한 수준의 심각한 에러가 발생함을 알림	50

1) logger

어플리케이션 코드가 직접 사용할 수 있는 인터페이스를 제공함

각 logger는 name을 가지는데 이 name들은 마침표(.)를 통해 계층적 관계를 형성하게 된다.

ex) test인 로거가 있다면 test.ex1, test.ex1.ex2 는 모두 test의 자식이고 test는 부모이다.

- logger 생성

```
test_logger = logging.getLogger("test")     # -- "test"에 아무것도 입력하지 않을 경우
                                            # -- root logger가 생성됩니다.
```

- level 부여하기

```
test_logger.setLevel(logging.INFO)          # -- INFO 이상의 메시지를 출력할 수 있습니다.
```

2) `handler`

level에 따라 적절한 log 메시지를 지정된 위치에 전달하는 역할

handler는 기능과 목적에 따라 여러 개일 수 있으며, 각 handler는 다른 level 과 다른 format을 가질 수도 있다.

handler의 종류는 15가지가 있는데, 가장 기본적인 것은 **StreamHandler** / **FileHandler** 2가지가 있다.

> StramHandler : 콘솔에 메시지를 전달
> FileHandler : 파일에 메시지를 전달

- handler 생성

```
#StreamHandler

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

```
# FileHandler

file_handler = logging.FileHandler("test.log")
test_logger.addHandler(file_handler)
```

- 메시지 출력

```
test_logger.debug("debug message")
test_logger.info("info message")
test_logger.warning("warning message")
test_logger.error("error message")
test_logger.critical("critical message")
```

`StreamHandler` 는 콘솔에 메시지가 출력되고, `FileHandler`는 생성한 파일(test.log)를 확인하면 메시지가 출력된 것을 확인할 수 있습니다. FileHandler의 파일 모드 기본값은 a모드입니다.

참고 : 파일 모드 정리

logger / handler 실행 결과

```
import logging

if __name__ == '__main__':

  # logger 생성
  test_logger = logging.getLogger("test")
  test_logger.setLevel(logging.INFO)

  # handler 생성
  stream_handler = logging.StreamHandler()           # -- StreamHandler
  test_logger.addHandler(stream_handler)

  file_handler = logging.FileHandler("test.log")     # -- FileHandler
  test_logger.addHandler(file_handler)               # -- 디렉토리에 test.log 파일 생성

  # 각 level 메시지 출력
  test_logger.debug("debug message")
  test_logger.info("info message")
  test_logger.warning("warning message")
  test_logger.error("error message")
  test_logger.critical("critical message")

  # 실행 결과
  # info message
  # warning message
  # error message
  # critical message

  # test.log 파일
  # info message
  # warning message
  # error message
  # critical message
```

logger 생성에서 level을 info로 설정하였기 때문에 info 이상의 메시지만 출력하는 걸 볼 수 있다.

3) fomatter

앞 서 설정한 메시지를 어떤 형식으로 출력할지 정한다.

> asctime : 시간
> name : logger 이름
> levelname : logging 레벨
> message : 메시지

- formatter 생성

```
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

stream_handler.setFormatter(formatter)
```

logger / handler + formatter 실행 결과

```
import logging

if __name__ == '__main__':

  # logger 생성
  test_logger = logging.getLogger("test")
  test_logger.setLevel(logging.INFO)

  formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

  # handler 생성
  stream_handler = logging.StreamHandler()           # -- StreamHandler
  stream_handler.setFormatter(formatter)
  test_logger.addHandler(stream_handler)

  file_handler = logging.FileHandler("test.log")     # -- FileHandler
  test_logger.addHandler(file_handler)               # -- 디렉토리에 test.log 파일 생성

  # 각 level 메시지 출력
  test_logger.debug("debug message")
  test_logger.info("info message")
  test_logger.warning("warning message")
  test_logger.error("error message")
  test_logger.critical("critical message")

  # 실행 결과
  # 2021-08-02 14:32:48,336 - test - INFO - info message
  # 2021-08-02 14:32:48,337 - test - WARNING - warning message
  # 2021-08-02 14:32:48,338 - test - ERROR - error message
  # 2021-08-02 14:32:48,338 - test - CRITICAL - critical message
```

로깅 파일 불러오기

파일 포맷 : JSON, INI, YAML

위 파일들을 py로 불러올 수 있습니다.

예시 - logging.json

```
{
  "version": 1,
  "formatters": {
  "simple": {
  "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
  }
  },

  "handlers": {
  "console": {
  "class": "logging.StreamHandler",
  "level": "INFO",
  "formatter": "simple",
  "stream": "ext://sys.stdout"
  },

  "info_file_handler": {
  "class": "logging.FileHandler",
  "level": "DEBUG",
  "formatter": "simple",
  "filename": "info.log"
  }
  },

  "root": {
  "level": "DEBUG",
  "handlers": ["console", "info_file_handler"]
  }
  }
```

logging_json.py - 실행 파일

```
import logging
import logging.config
import json
import os

if __name__ == '__main__':
  with open('logging\logging.json', 'rt') as f:
    config = json.load(f)

    logging.config.dictConfig(config)
```

```
    logger = logging.getLogger()
    logger.warning("warning message")
```

logger.debug("debug message") 입력 시 FileHandler 실행

logger.warning("warning message") 입력 시 ScreamHandler 실행

→ ScreamHandler level이 info 이기 때문에 info 이상의 메시지만 출력합니다.