Thread와 Sqlite3

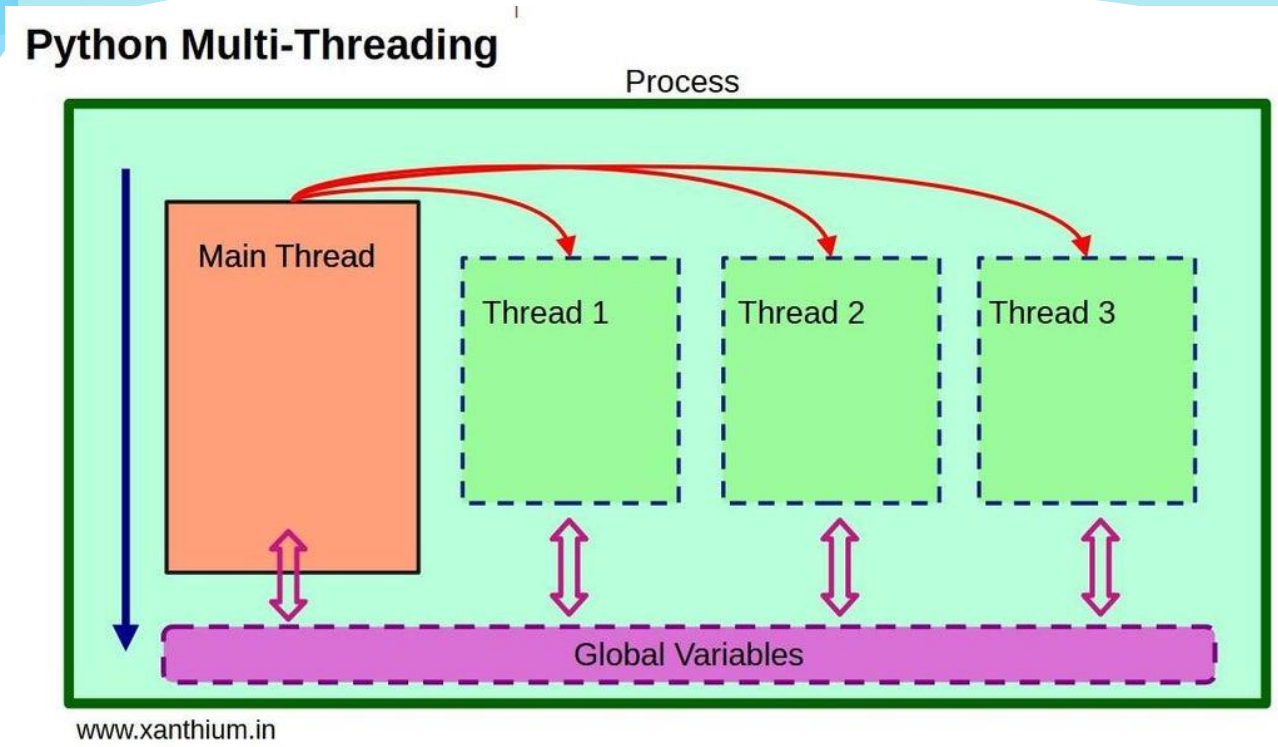
- Wifi 설정
  Rokey / rokey12345

- 방화벽 해지
  sudo ufw disable

- 같은 ROS_DOMAIN_ID에 영향을 안받게 하기
  export $ROS_LOCALHOST_ONLY=1

김루진 강사

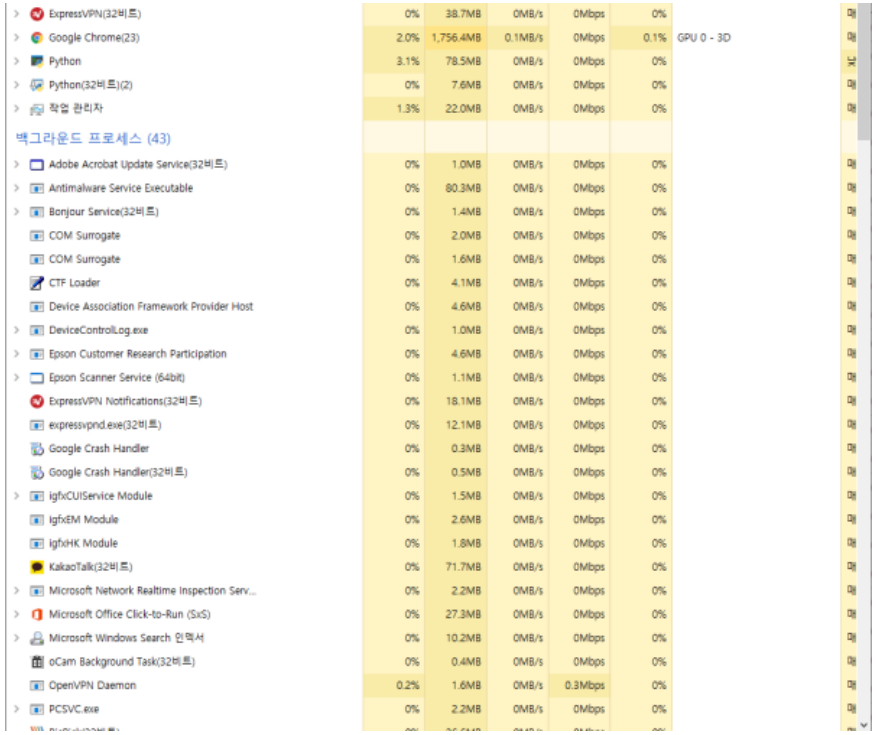
파이썬 멀티 쓰레드



쓰레드를 사용하는 주된 이유는 프로그램의 성능과 응답성을 향상시키기 위해서입니다.
특히 ROS2와 PyQt GUI를 함께 사용할 때 더욱 중요합니다.

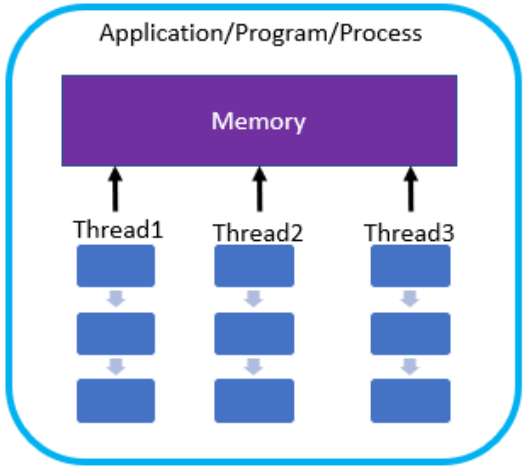
프로세스와 쓰레드

- 프로그램이 하나의 일을 처리할 때, CPU는 메모리에 프로그램에 관련된 DATA를 저장하고 처리하게 된다.
- 이렇게 일련의 프로그램을 진행하는 것을 프로세스(Process)라고 한다.
- 하나의 프로그램을 실행시키면, 그 프로그램은 한개의 프로세스를 가지게 된다.

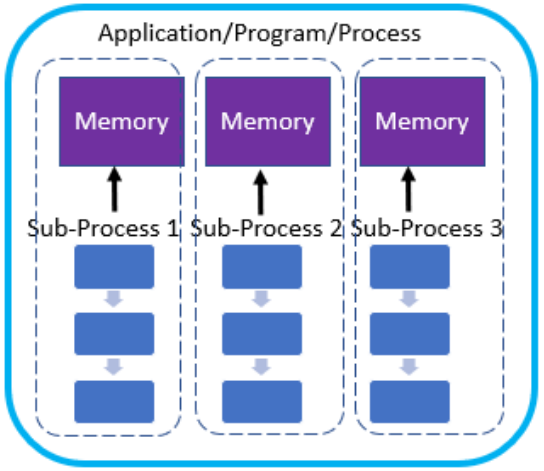


프로세스와 쓰레드

- 새로운 프로세스를 생성하는 것을 프로세스 포크(Fork)
- 프로세스는 프로그램이 작동하기에 필요한 많은 데이터를 가지고 있다.
- 프로세스 포크를 하게되면, 새로 생성된 프로세스는 이전의 프로세스와 동일한 데이터를 카피
  - 프로세스간의 데이터 공유를 위해 많은 뒤처리가 필요하고, 또한 컴퓨터 자원의 소모 야기
  - 새로운 프로세스는 필요하지 않지만, 하나의 일을 따로 처리해야할 때 사용하는 것이 쓰레드



Multi-Threading



Multi-processing

멀티쓰레드와 멀티프로세스의 가장 큰 차이는 **자원 공유**

쓰레드 사용 이유

- GUI 응답성 유지
  - GUI는 메인 쓰레드(이벤트 루프)에서 실행됨
  - 시간이 오래 걸리는 작업을 메인 쓰레드에서 실행하면 GUI가 멈춤
  - 별도 쓰레드로 실행하면 GUI가 계속 반응 가능

- ROS2 통신 처리
  - ROS2의 publish/subscribe 통신은 지속적으로 발생
  - 메인 쓰레드에서 처리하면 다른 작업 수행 불가
  - 별도 쓰레드로 분리하여 동시 처리 가능

쓰레드 사용 이유

- 자원 효율성
  - 멀티코어 CPU 활용 가능
  - 작업을 병렬로 처리하여 성능 향상
  - 시스템 리소스 효율적 사용
- 데드락 방지
  - GUI 이벤트 처리와 ROS2 통신을 분리
  - 상호 블로킹 현상 방지
  - 안정적인 프로그램 실행

```
# 잘못된 예 (쓰레드 미사용)
def publish_message():
    while True:
        publisher.publish(msg)
        # GUI가 완전히 멈춤

# 올바른 예 (쓰레드 사용)
class PublisherThread(QThread):
    def run(self):
        while self.running:
            publisher.publish(msg)
            # GUI는 계속 응답 가능
```

함수를 쓰레드로 만드는 방법

threading 모듈의 Thread 클래스를 사용
Thread 객체를 생성할 때 target 매개변수에 실행할 함수를 지정하고, 필요에 따라 args 매개변수를 사용

```python
import threading
import time

def my_function(name, delay):
    for i in range(3):
        time.sleep(delay)
        print(f"{name}: {i}")

# 쓰레드 생성
thread1 = threading.Thread(target=my_function, args=("Thread 1", 1))
thread2 = threading.Thread(target=my_function, args=("Thread 2", 2))

# 쓰레드 시작
thread1.start()
thread2.start()

# 쓰레드 완료 대기
thread1.join()
thread2.join()
```

```python
import threading
import time

class MyThread(threading.Thread):
    def __init__(self, name, delay):
        super().__init__()
        self.name = name
        self.delay = delay

    def run(self):
        for i in range(3):
            time.sleep(self.delay)
            print(f"{self.name}: {i}")

# 쓰레드 객체 생성
thread1 = MyThread("Thread 1", 1)
thread2 = MyThread("Thread 2", 2)

# 쓰레드 시작
thread1.start()
thread2.start()

# 쓰레드 완료 대기
thread1.join()
thread2.join()
```

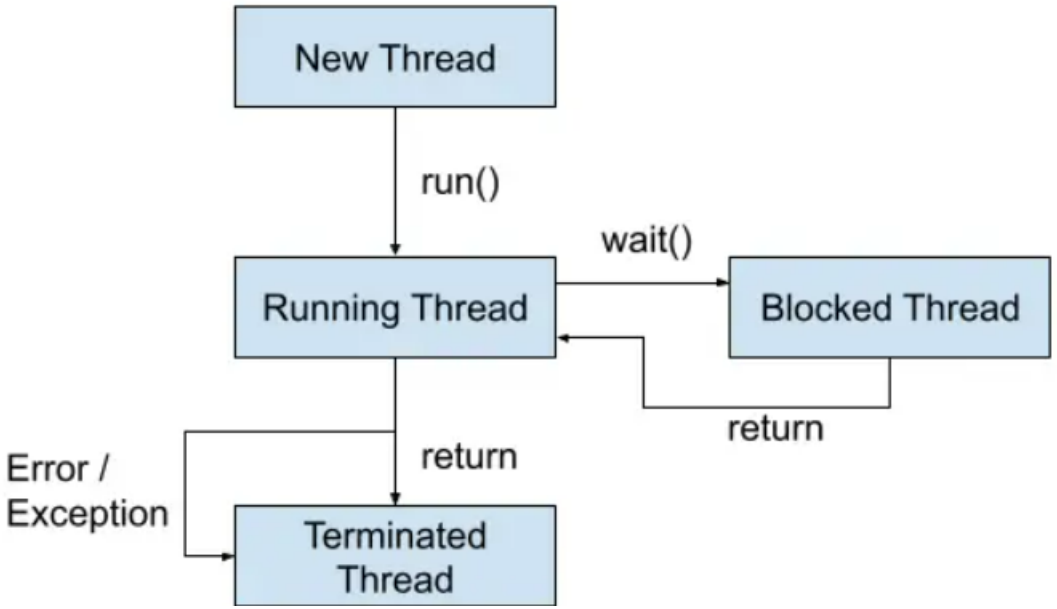
클래스를 쓰레드로 만드는 방법

파이썬에서 클래스를 쓰레드로 만들기 위해서는 threading 모듈의 Thread 클래스를 상속하거나, Thread 인스턴스를 생성할 때 target 매개변수에 함수를 지정하는 방법을 사용

```python
import threading
import time

class MyThread(threading.Thread):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def run(self):
        for i in range(5):
            print(f"{self.name} is running: {i}")
            time.sleep(1)

# 쓰레드 객체 생성 및 시작
thread1 = MyThread("Thread-1")
thread2 = MyThread("Thread-2")

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```

```python
import threading
import time

class MyTask:
    def __init__(self, name):
        self.name = name

    def run(self):
        for i in range(5):
            print(f"{self.name} is running: {i}")
            time.sleep(1)

# 인스턴스를 생성하고, run 메서드를 쓰레드로 실행
task1 = MyTask("Task-1")
task2 = MyTask("Task-2")

thread1 = threading.Thread(target=task1.run)
thread2 = threading.Thread(target=task2.run)

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```

쓰레드 라이프싸이클



- 생성단계

```python
import threading

def my_task():
    print("Thread is running")

# 쓰레드 객체 생성
thread = threading.Thread(target=my_task)
```

- 시작 (Started) 단계

```python
# start() 메서드 호출로 쓰레드 시작
thread.start()  # 이때 my_task 함수가 실행됨
```

쓰레드 라이프싸이클

- 실행 (Running) 단계

```python
class MyThread(threading.Thread):
    def run(self):
        # 실행 중인 상태
        print("Thread is running")
        for i in range(5):
            print(f"Working... {i}")
```

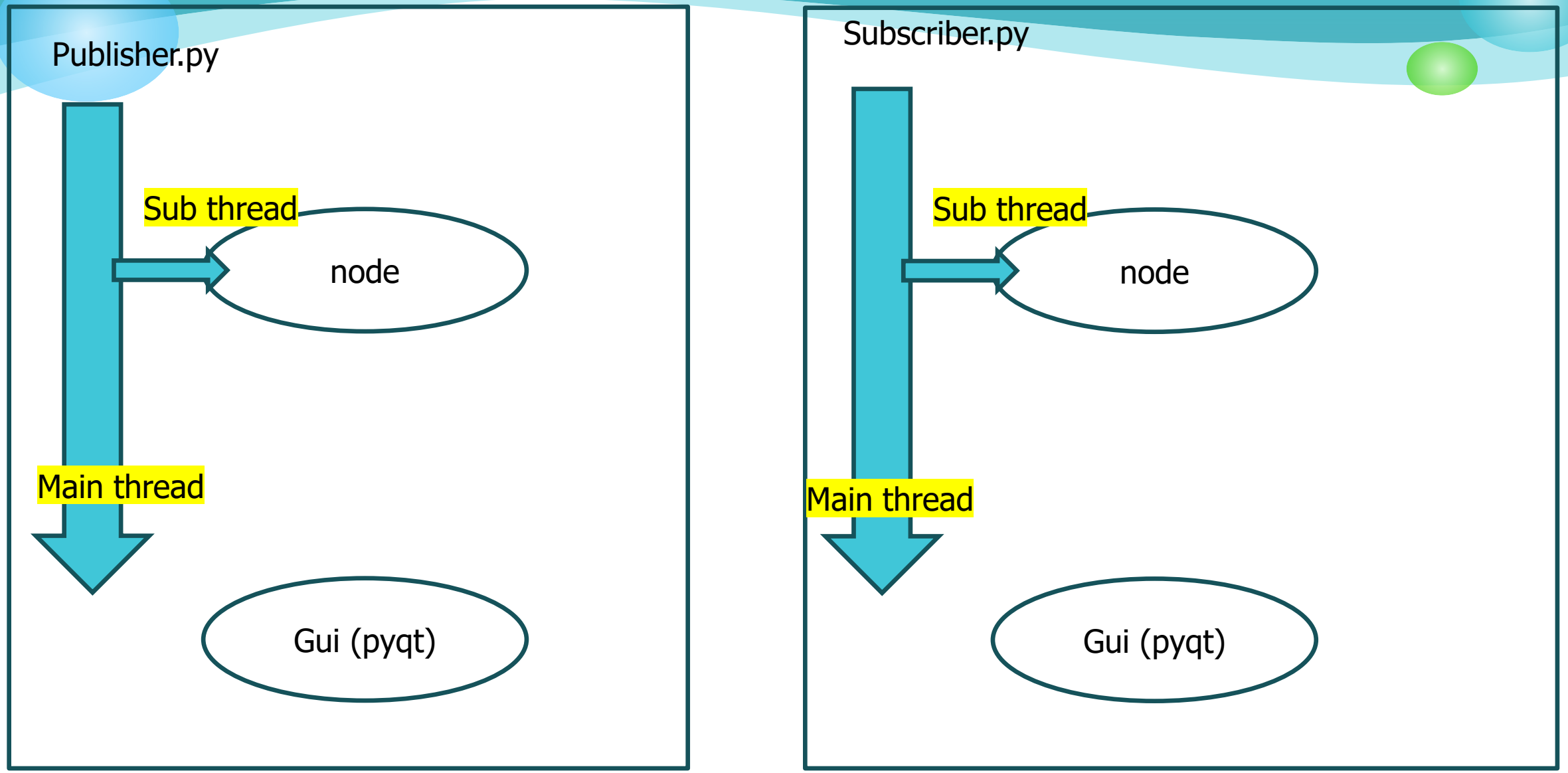
- 대기 (Waiting) 단계

```python
# join()으로 쓰레드 완료 대기
thread.join()  # 메인 쓰레드가 이 쓰레드의 완료를 기다림

# sleep으로 일시 정지
import time
time.sleep(1)  # 1초 동안 대기
```

- 종료 (Terminated) 단계

```python
# 쓰레드 작업이 완료되면 자동으로 종료
# 또는 강제 종료를 위한 플래그 사용
class StoppableThread(threading.Thread):
    def __init__(self):
        super().__init__()
        self.stop_flag = False

    def run(self):
        while not self.stop_flag:
            # 작업 수행
            pass

    def stop(self):
```

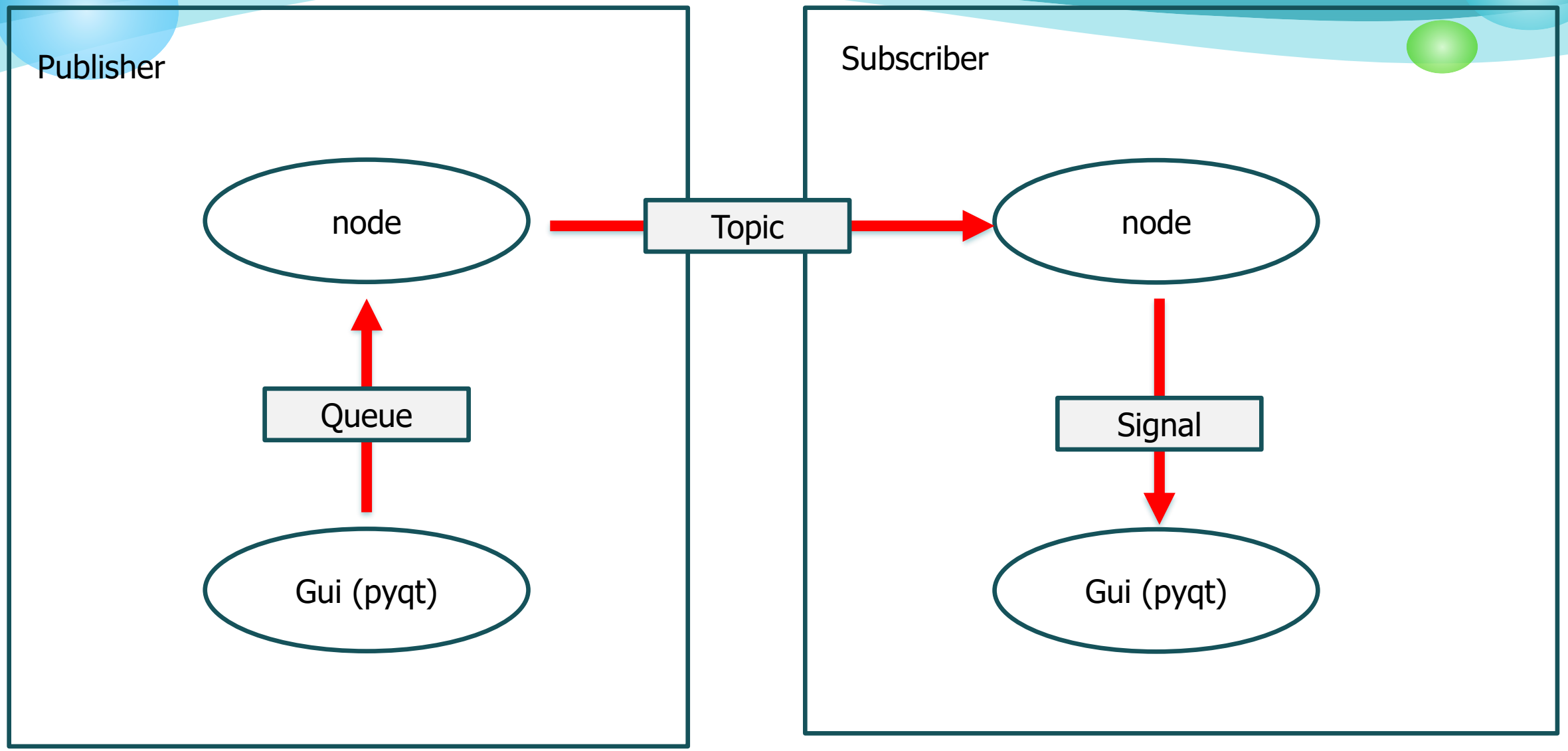
Ros2 publisher, subscriber



Ros2 publisher, subscriber

```python
def main():
    rclpy.init()
    node = NODE()
    ros_thread = threading.Thread(target=lambda : rclpy.spin(node), daemon=True)
    ros_thread.start()

    app = QApplication(sys.argv)
    gui = GUI(node)
    gui.window.show()

    try:
        sys.exit(app.exec_())

    except KeyboardInterrupt:
        sys.exit(0)

    finally:
        node.destroy_node()
        rclpy.shutdown()
```

Ros2 publisher, subscriber



Ros2 publisher, subscriber

Publisher

```
class GUI():
    def __init__(self, node):···

    def setupUi(self):···

    def button_clicked(self):
        self.message = self.lineEdit.text()
        self.node.queue.put(self.message)
        self.lineEdit.clear()
```

```
class NODE(Node):
    def __init__(self):
        super().__init__('node')
        qos_profile = QoSProfile(depth=5)
        self.message_publisher = self.create_publisher(String, 'message', qos_profile)

        self.queue = queue.Queue()
        self.timer = self.create_timer(0.1, self.publish_message)

    def publish_message(self):
        while not self.queue.empty():
            message = self.queue.get()
            msg = String()
            msg.data = message
            self.message_publisher.publish(msg)
            self.get_logger().info(f'Published message: {message}')
```

Ros2 publisher, subscriber

Subscriber

```python
class NODE(QThread, Node):
    message_received = Signal(str)

    def __init__(self, node_name='ros_subscriber_node'):
        QThread.__init__(self)
        Node.__init__(self, node_name)

        self.subscription = self.create_subscription(
            String, 'message', self.subscription_callback, 10)

    def subscription_callback(self, msg):
        message = msg.data
        self.get_logger().info(f'Received message: {message}')
        self.message_received.emit(message)

    def run(self):
        rclpy.spin(self)
```

```python
class GUI(QMainWindow):
    def __init__(self, ros_thread):
        super().__init__()
        self.ros_thread = ros_thread
        self.ros_thread.message_received.connect(self.add_message)
        self.setupUi()
```

UI의 저장과 Python코드와의 연결

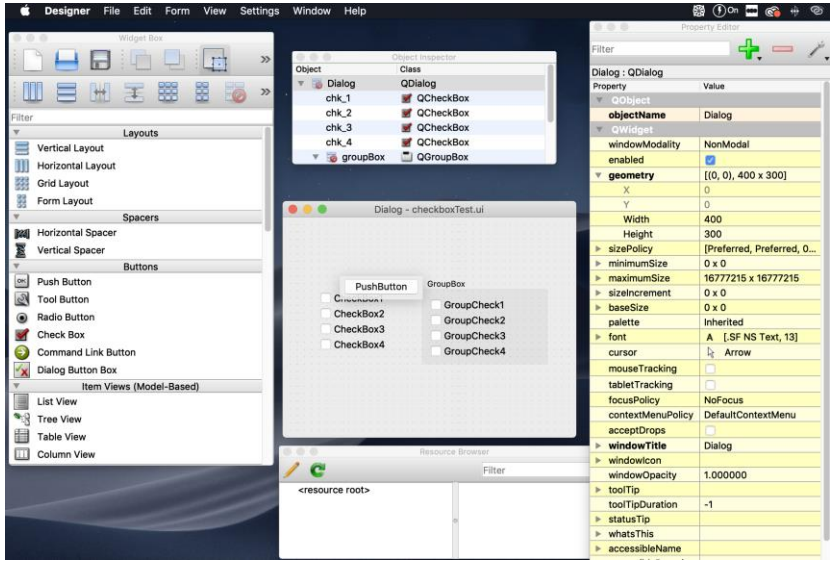
$ sudo pip3 install PySide2

$ designer

Qt Designer의 UI는 저장버튼(Control + S / Command + S)를 누르면 저장을 할 수 있습니다. 저장된 UI파일은 XML의 형식을 가짐

Python 코드에서 이 XML 파일을 Import한 후 위젯들에 기능을 할당해주면 실제로 기능을 가지고 작동하는 GUI프로그램이 완성

직접 XML형식의 ui파일을 수정하여 레이아웃을 수정할 수도 있습니다.

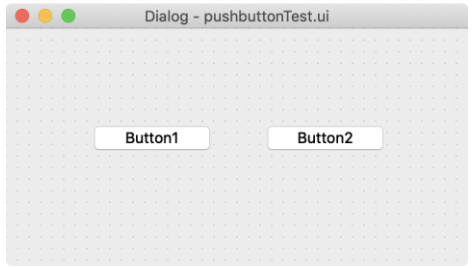


UI의 저장과 Python코드와의 연결

UI파일을 Python에 Import하여 사용하는 방법

https://wikidocs.net/35481

```python
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic

#UI파일 연결
#단, UI파일은 Python 코드 파일과 같은 디렉토리에 위치해야한다.
form_class = uic.loadUiType("UI파일이름.ui")[0]

#화면을 띄우는데 사용되는 Class 선언
class WindowClass(QMainWindow, form_class) :
    def __init__(self) :
        super().__init__()
        self.setupUi(self)

if __name__ == "__main__" :
    #QApplication : 프로그램을 실행시켜주는 클래스
    app = QApplication(sys.argv)

    #WindowClass의 인스턴스 생성
    myWindow = WindowClass()

    #프로그램 화면을 보여주는 코드
    myWindow.show()

    #프로그램을 이벤트루프로 진입시키는(프로그램을 작동시키는) 코드
    app.exec_()
```




```python
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic

#UI파일 연결
#단, UI파일은 Python 코드 파일과 같은 디렉토리에 위치해야한다.
form_class = uic.loadUiType("pushbuttonTest.ui")[0]

#화면을 띄우는데 사용되는 Class 선언
class WindowClass(QMainWindow, form_class) :
    def __init__(self) :
        super().__init__()
        self.setupUi(self)

        #버튼에 기능을 연결하는 코드
        self.btn_1.clicked.connect(self.button1Function)
        self.btn_2.clicked.connect(self.button2Function)

    #btn_1이 눌리면 작동할 함수
    def button1Function(self) :
        print("btn_1 Clicked")

    #btn_2가 눌리면 작동할 함수
    def button2Function(self) :
        print("btn_2 Clicked")


if __name__ == "__main__" :
    app = QApplication(sys.argv)
    myWindow = WindowClass()
    myWindow.show()
    app.exec_()
```


SQLite

데이터 베이스 정의

- 체계적으로 구조화된 데이터의 집합으로, 효율적인 데이터 관리와 검색을 위한 시스템

주요 특징:
1.데이터 독립성
•물리적 독립성: 저장 구조가 변경되어도 응용 프로그램에 영향 없음
•논리적 독립성: 논리적 구조 변경이 응용 프로그램에 영향 없음

2.데이터 무결성
•정확성과 일관성 보장
•제약조건을 통한 데이터 품질 유지
•중복 최소화

3.동시성 제어
•여러 사용자의 동시 접근 관리
•데이터 일관성 유지
•트랜잭션 처리

주요 구성 요소

- 테이블(릴레이션)
- 필드(속성, 컬럼)
- 레코드(튜플, 행)
- 키(기본키, 외래키)
- 인덱스
- 뷰

DDL(Data Definition Language)

```
CREATE: 데이터베이스 객체 생성
DROP: 데이터베이스 객체 삭제
ALTER: 데이터베이스 객체 구조 변경
TRUNCATE: 테이블 내용 전체 삭제
```

DML(Data Manipulation Language)

```
SELECT: 데이터 조회
INSERT: 데이터 삽입
UPDATE: 데이터 수정
DELETE: 데이터 삭제
```

Relation Map

One-to-One (1:1)

```
CREATE TABLE user (
    user_id INT PRIMARY KEY,
    name VARCHAR(50)
);

CREATE TABLE passport (
    passport_id INT PRIMARY KEY,
    user_id INT UNIQUE,
    FOREIGN KEY (user_id) REFERENCES user(user_id)
);
```

Relation Map

One-to-Many (1:N)

```
CREATE TABLE department (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50)
);

CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    dept_id INT,
    name VARCHAR(50),
    FOREIGN KEY (dept_id) REFERENCES
department(dept_id)
);
```

Relation Map

Many-to-Many (N:M)

```
CREATE TABLE student (
    student_id INT PRIMARY KEY,
    name VARCHAR(50)
);

CREATE TABLE course (
    course_id INT PRIMARY KEY,
    title VARCHAR(100)
);

CREATE TABLE enrollment (
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES student(student_id),
    FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```

sqlite3

- SQLite3는 파일 기반의 경량 관계형 데이터베이스 시스템

장점
- 서버가 필요 없음 (파일 기반)
- 설치가 필요 없음 (Python 기본 내장)
- 가볍고 빠름
- 단일 파일로 관리
- 트랜잭션 지원
- Zero-configuration

제한사항
- 동시 접근 제한
- 대규모 데이터에는 부적합
- 복잡한 쿼리에 제한적

테이블 주문 정보 테이블 설계

```
-- 테이블 정보 (table_order)
CREATE TABLE IF NOT EXISTS table_order (
    table_id INTEGER PRIMARY KEY AUTOINCREMENT,
    table_number INTEGER NOT NULL,
    capacity INTEGER NOT NULL,
    status TEXT DEFAULT 'empty' CHECK(status IN ('empty', 'occupied', 'reserved')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

테이블 주문 정보 테이블 설계

```
-- 메뉴 정보 (menu)
CREATE TABLE IF NOT EXISTS menu (
    menu_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL NOT NULL,
    category TEXT NOT NULL,
    description TEXT,
    available BOOLEAN DEFAULT 1,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

테이블 주문 정보 테이블 설계

```
-- 주문 목록 (order_list)
CREATE TABLE IF NOT EXISTS order_list (
    order_id INTEGER PRIMARY KEY AUTOINCREMENT,
    table_id INTEGER,
    total_amount REAL DEFAULT 0,
    order_status TEXT DEFAULT 'pending' CHECK(order_status IN ('pending', 'preparing', 'served', 'completed',
'cancelled')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (table_id) REFERENCES table_order(table_id)
);
```

테이블 주문 정보 테이블 설계

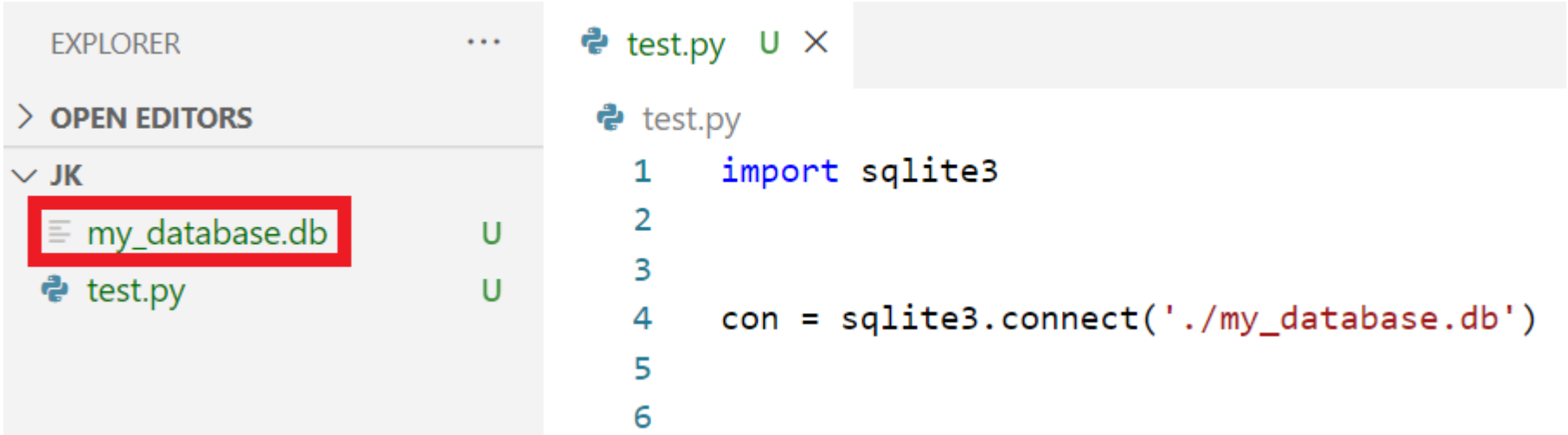
```
-- 주문 상세 항목 (order_items)
CREATE TABLE IF NOT EXISTS order_items (
    order_item_id INTEGER PRIMARY KEY AUTOINCREMENT,
    order_id INTEGER,
    menu_id INTEGER,
    quantity INTEGER NOT NULL,
    price REAL NOT NULL,
    subtotal REAL NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (order_id) REFERENCES order_list(order_id),
    FOREIGN KEY (menu_id) REFERENCES menu(menu_id)
);
```

테이블 주문 정보 테이블 설계

```
-- 취소 정보 (cancel)
CREATE TABLE IF NOT EXISTS cancel (
    cancel_id INTEGER PRIMARY KEY AUTOINCREMENT,
    order_id INTEGER,
    cancel_reason TEXT NOT NULL,
    refund_amount REAL DEFAULT 0,
    cancel_status TEXT DEFAULT 'pending' CHECK(cancel_status IN ('pending', 'approved', 'rejected')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (order_id) REFERENCES order_list(order_id)
);
```

데이터베이스 생성

데이터베이스를 생성할 때는 간단히 **connect() 메서드**를 사용하면 된다.
이때 메서드의 인자로 넣은 값이 데이터베이스 파일의 경로가 된다.
예를 들어 'my_database.db'라는 파일을 생성하고자 한다면 다음과 같이 한다.
현재 '**my_database.db**'라는 데이터베이스 파일이 없기 때문에, 새롭게 생성된 것을 확인할 수 있다.



테이블 생성

SQL 구문을 이용하기 위해서는 **cursor 객체**가 필요하다.
cursor 객체를 이용하여 실제로 데이터베이스에 테이블(table)을 삽입하거나, 테이블(table)을 조회할 수 있다.
예를 들어 다음의 명령어는 Course(과목)라는 이름의 테이블을 생성한다.
이후에 **생성된 테이블 정보를 조회**한다.

```
cursor = con.cursor()

SQL = "CREATE TABLE Course (Course_ID int primary key not null, Course_Name text,
Course_Date date);"
cursor.execute(SQL)

SQL = "SELECT name FROM sqlite_master WHERE type='table';"
cursor.execute(SQL)
print(cursor.fetchall())

SQL = "SELECT sql FROM sqlite_master WHERE type='table';"
cursor.execute(SQL)
print(cursor.fetchall())

con.close()
```

데이터 삽입

```
SQL = "INSERT INTO Course VALUES(1, 'Algorithm', '2021-03-01');"
cursor.execute(SQL)
SQL = "INSERT INTO Course VALUES(2, 'Data Structure', '2021-03-02');"
cursor.execute(SQL)
SQL = "INSERT INTO Course VALUES(3, 'Computer Architecture', '2021-03-05');"
cursor.execute(SQL)
con.commit()

SQL = "SELECT * FROM Course;"
cursor.execute(SQL)
print(cursor.fetchall())

con.close()
```

데이터 삭제

데이터 삭제는 DELETE 구문을 사용하면 된다. 예를 들어 Course 테이블에 있는 모든 데이터(row)를 삭제하기 위해서는 다음과 같이 하면 된다. INSERT 구문과 마찬가지로 실행 뒤에 commit() 메서드를 이용해 DB에 반영할 수 있다. 위 코드를 실행하면 Course 테이블에 존재하는 모든 데이터가 삭제되기 때문에, 조회 결과가 없다.

```
cursor = con.cursor()

SQL = "DELETE FROM Course;"
cursor.execute(SQL)
con.commit()

SQL = "SELECT * FROM Course;"
cursor.execute(SQL)
print(cursor.fetchall())

con.close()
```

데이터 입력

```
def insert_course(course_id, course_name, course_date):
    con = sqlite3.connect(database_name)
    cursor = con.cursor()

    SQL = "INSERT INTO Course VALUES(?, ?, ?);"
    cursor.execute(SQL, (course_id, course_name,
course_date))

    con.commit()
    con.close()
```

데이터 입력

```
def insert_course_list(course_list):
    con = sqlite3.connect(database_name)
    cursor = con.cursor()

    SQL = "INSERT INTO Course VALUES(?, ?, ?);"
    cursor.executemany(SQL, course_list)

    con.commit()
    con.close()
```

데이터 검색

```
def search_course_by_name(course_name):
    con = sqlite3.connect(database_name)
    cursor = con.cursor()

    SQL = "SELECT * FROM Course WHERE course_name = ?;"
    cursor.execute(SQL, (course_name, ))

    return cursor.fetchall()
```

데이터 업데이트

```
def update_course_by_id(course_id, course_name, course_date):
    con = sqlite3.connect(database_name)
    cursor = con.cursor()

    SQL = "UPDATE Course SET course_name = ?, course_date = ? WHERE course_id = ?;"
    cursor.execute(SQL, (course_name, course_date, course_id))

    con.commit()
    con.close()
```

데이터 삭제

```
def delete_course_by_id(course_id):
    con = sqlite3.connect(database_name)
    cursor = con.cursor()

    SQL = "DELETE FROM Course WHERE course_id = ?;"
    cursor.execute(SQL, (course_id, ))

    con.commit()
    con.close()
```

감사합니다.