

XNode로 배우는

저전력 무선 네트워크 프로그래밍

4 센서 네트워크 개발 환경

XNode 구성

- IoT 센서 노드 및 확장 모듈, 엣지 서버, USB 허브 등으로 구성
 - ▣ 목적에 따라 세분되어 있으며 제품에 따라 구성이 조금씩 다름
 - XNode Lite: IoT 센서 Zigbee 노드 3개, 확장 모듈 5개, USB 허브, 케이블
 - Zigbee V3.0 기반 IoT 센서 네트워크 입문용
 - XNode: XNode Lite 구성에 LoRa 노드 2개 및 Zigbee 노드 2개 추가 (IoT 센서 노드 7개)
 - 다양한 IoT 센서 네트워크 구성
 - XNode Plus: XNode 구성에 엣지 인공지능 모듈이 포함된 엣지 서버(게이트웨이) 추가
 - IoT 센서 네트워크의 게이트웨이 및 엣지 클라우드 지원. 인공지능 학습 가능
 - XNode PrimeX: 엣지 서버에 포함된 엣지 인공지능 모듈을 최고 사양으로 업그레이드

XNode 구성

- IoT 센서 노드는 A Type과 B Type으로 구분
 - A Type: LoRa와 Wi-Fi 통신을 지원하며 마이크로파이썬으로 프로그래밍
 - B Type: Zigbee V3.0과 BLE 통신을 지원하며 마이크로파이썬으로 프로그래밍
- 확장 모듈은 IoT 센서 노드의 확장 커넥터에 연결해 운영하며 추가 센서 지원
 - BASIC: 기본 주변 장치(LED, 버튼, 부저)
 - PIR: 움직임 감지 센서
 - IRTHERMO: 비접촉 온도 센서
 - IMU: 9축(가속도, 자이로, 지자기) 관성 센서
 - GPS: 범지구적 연재 위치 확인용 GPS 수신기

XNode Plus 구성품

■ 엣지 서버 (XNode Plus 제품에 포함)

- LoRa 및 Zigbee 모듈과 고성능 엣지 인공지능 모듈로 구성
- IoT 센서 노드를 인터넷에 연결하는 게이트웨어 역할 수행
- 내장된 클라우드 소프트웨어를 통해 수집한 센서 데이터 처리 지원

■ USB 허브

- USB 3.0 기반 7 포트 허브로 다수의 IoT 센서 노드와 PC 연결
 - 시리얼 포트로 센서 노드 구분
- IoT 센서 노드(이하 XNode)의 내장 배터리 충전
 - 연결된 XNode 개수에 따라 전원 어댑터 추가 연결 필요

배포 USB

- XNode 학습에 필요한 툴과 예제 포함
 - ▣ Code: 교재에서 다루는 예제 코드
 - 기본 센서 및 확장 모듈, 센서 네트워크로 항목별 구분
 - ▣ Library: 기본 및 확장 모듈용 Pop 라이브러리.
 - 기본 Pop 라이브러리(pop.py)는 항상 설치되어 있어야 함.
 - XNode에는 미리 설치되어 있음
 - 확장 Pop 라이브러리는 XNode의 저장소 절약을 위해 필요한 것만 선택해 설치

배포 USB

- CORE/lib/pop.py, core_a.py, core_b.py: XNode 포맷에 대비해 제공하는 기본 Pop 라이브러리
- EXT/lib/BASIC.py: BASIC 확장 모듈용
- EXT/lib/PIR.py: PIR 확장 모듈용
- EXT/lib/IRTHERMO.py: IRTHERMO 확장 모듈용
- EXT/lib/IMU.py: IMU 확장 모듈용
- EXT/lib/GPS.py: GPS 확장 모듈용

▣ Software: 호스트용 툴 모음

- Windows64: Visual Studio Code 기반 윈도우용 통합 개발환경
 - SodaIDE_Installer_Vxy.exe: 설치 파일
- SodaOS: 옛지 서버용 Soda OS 이미지
 - XNode Plus 제품에 포함된 옛지 서버에 Soda OS를 새로 설치할 때 사용

개발 환경 구축

▣ SodaIDE

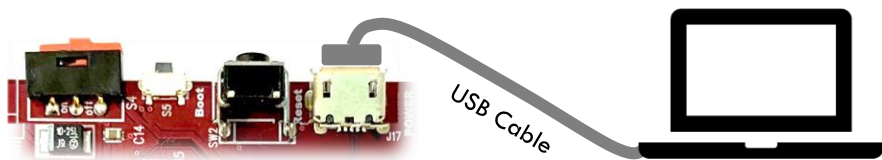
■ 다수의 오픈 소스 및 무료 툴로 구성된 XNode 통합 개발환경

- Visual Sutdio Code (이하 VS Code): XNode 용 마이크로파이썬 코드 편집
- xnode: 명령행 기반으로 마이크로파이썬 코드를 XNode에서 실행 및 XNode 파일 시스템 관리
- xmon: 플로팅 툴로 XNode에서 실시간으로 수집한 센서 데이터 시각화
- xquat: 3차원 시뮬레이션 툴로 XNode에서 실시간으로 수집한 IMU 확장 모듈의 쿼터니언 데이터 시각화
- XCTU: B Type XNode (Zigbee V3.0) 용 센서 네트워크 테스트 및 분석 툴
- python: VS Code에서 마이크로파이썬 코드 편집 시 코드 자동완성 지원을 위한 파이썬 SDK
 - 다수의 확장 모듈(PyQt5, PySerial, numpy 등)이 사전에 설치되어 있음
- 기타 Soda OS 기반 옛지 서버용 오픈 소프트웨어 포함
 - PuTTY, WinSCP, NoMachine Client, VcXsrv, Win32DiskImager 등

하드웨어 연결

- XNode 와 PC를 USB 케이블로 연결해 시리얼 통신 채널 생성

- ▣ 다수의 XNode는 USB 허브 사용
- ▣ Micro Type B(XNode) <— —> Type A(PC)



- ▣ XNode 프로그래밍 및 시리얼 데이터 통신과 전원 공급 수행

- USB 케이블을 연결하면 USB 전원으로 배터리 충전
- XNode 를 사용할 땐 전원 스위치로 전원 공급



하드웨어 연결

□ PC의 장치 관리자에서 자동 할당된 시리얼 포트 번호 확인

▣ 제어판 > 장치 관리자 > 포트(COM & LPT)



▣ XNode 프로그래밍할 때 필요

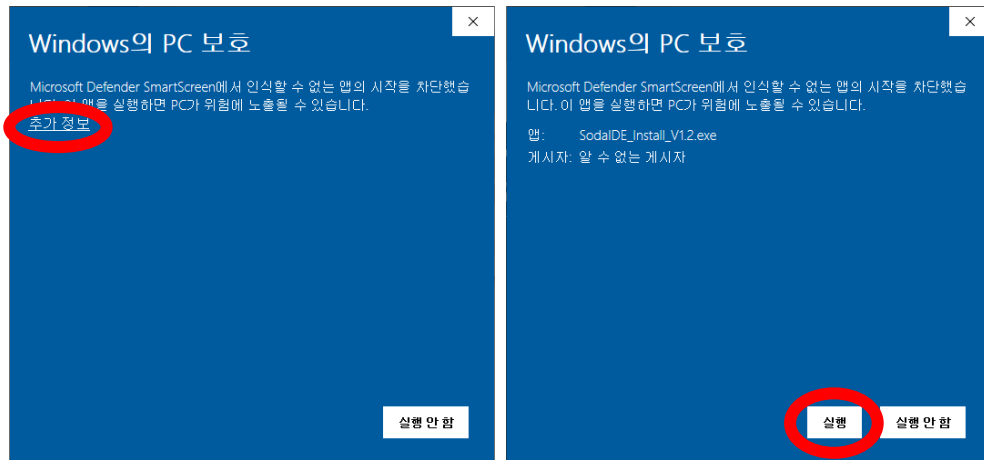
- 여러 개의 XNode를 연결할 경우 각각의 XNode 포트 번호를 파악해 둘 것
- 센서 네트워크 실습을 할 땐 3개 이상의 XNode 연결 필요
 - Coordinator, Router, End Device 역할

VSCode4Soda 설치

□ XNode와 엡지 서버를 위한 통합 개발환경

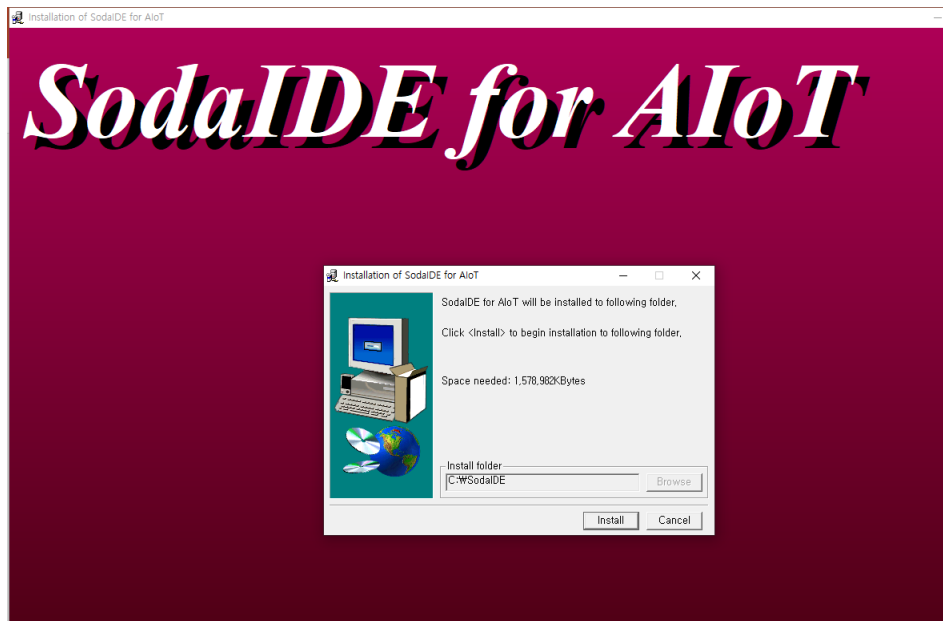
▣ 배포 USB > Software > Windows64 > SodalDE_Install_Vx.y.exe 실행

- 만약 Windows 10 보호를 위한 경고창이 표시되면 '추가 정보' 클릭 및 '실행' 클릭



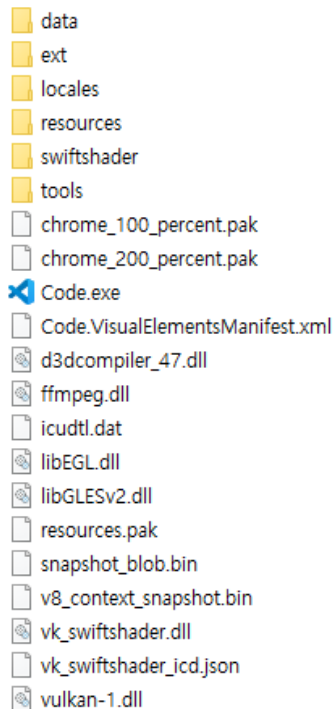
VSCode4Soda 설치

- 전체 설치 화면이 표시되면 확인 버튼을 눌러 설치 진행
 - ▣ 설치 위치는 'C:\SodaIDE'로 고정되며 변경할 수 없음



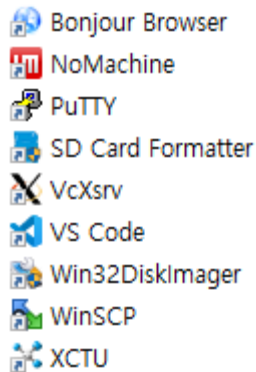
VSCode4Soda 설치

- 설치가 완료되면 C:\VSCode4Soda 경로에 해당 파일 위치



VSCode4Soda 설치

- ▣ 윈도우 시작 메뉴 > SodaIDE 에 다음의 파일들 확인
 - VS Code: XNode를 위해 미리 환경 설정된 VS Code
 - XCTU: XNode type B(Zigbee V3.0) 네트워크 설정 및 모니터링 툴
 - PuTTY: 시리얼 에뮬레이터
 - Qt Designer: GUI 호스트 응용 프로그램 개발을 위한 UI 디자이너
 - 옛지 서버용 툴
 - WinScp: 원격 파일 탐색기
 - NoMachine: 원격 데스크탑
 - VcXsrv: 윈도우용 X-Server
 - SD Card Formatter: microSD 카드 포맷 툴
 - Win32DiskImager: 옛지 서버용 이미지(Soda OS)를 microSD 카드에 설치
 - Bonjour Browser: 동적 서버 탐색



시작하기

□ 작업 폴더 설정

▣ 적당한 위치에 빈 작업 폴더 생성

- C:\XNode

▣ XNode와 함께 제공되는 USB 메모리를 PC에 연결해 배포 경로 설정

- D:\

- PC마다 다르며 D:\로 가정

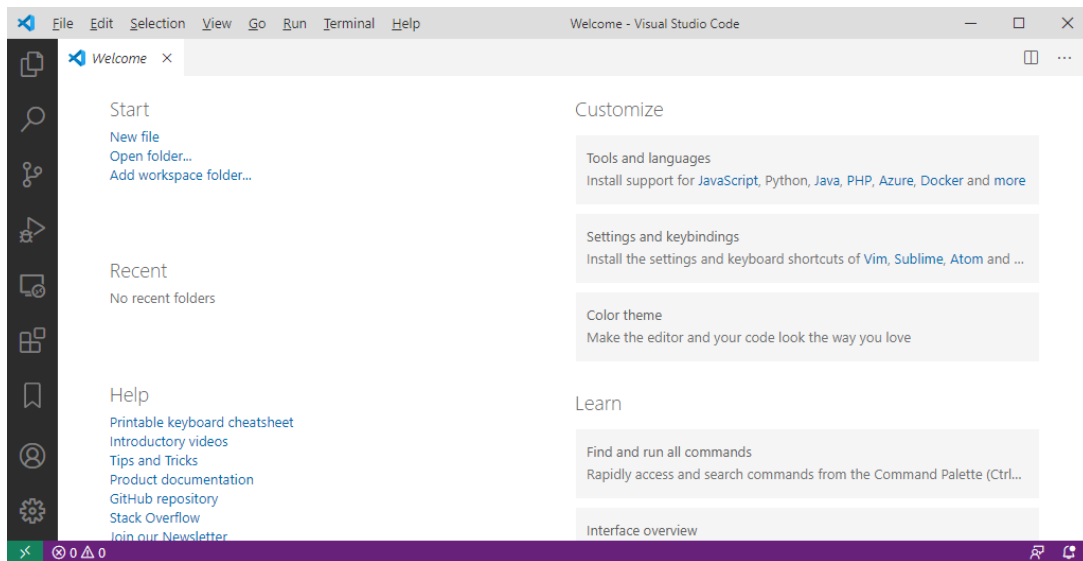
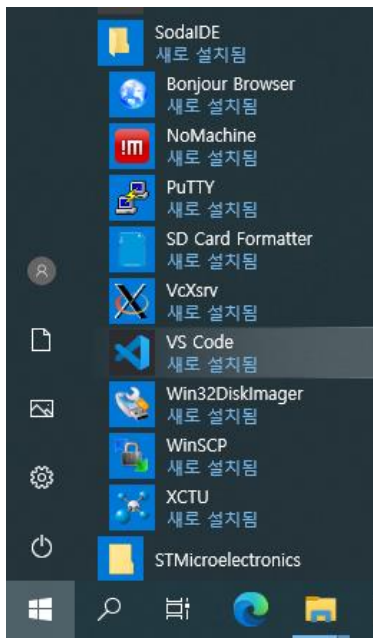
▣ 배포 경로의 Library 폴더 아래 CORE 폴더를 작업 폴더로 복사

- C:\XNode\CORE

시작하기

□ VS Code 실행

▣ 윈도우 시작 메뉴 > SodaIDE > VS Code

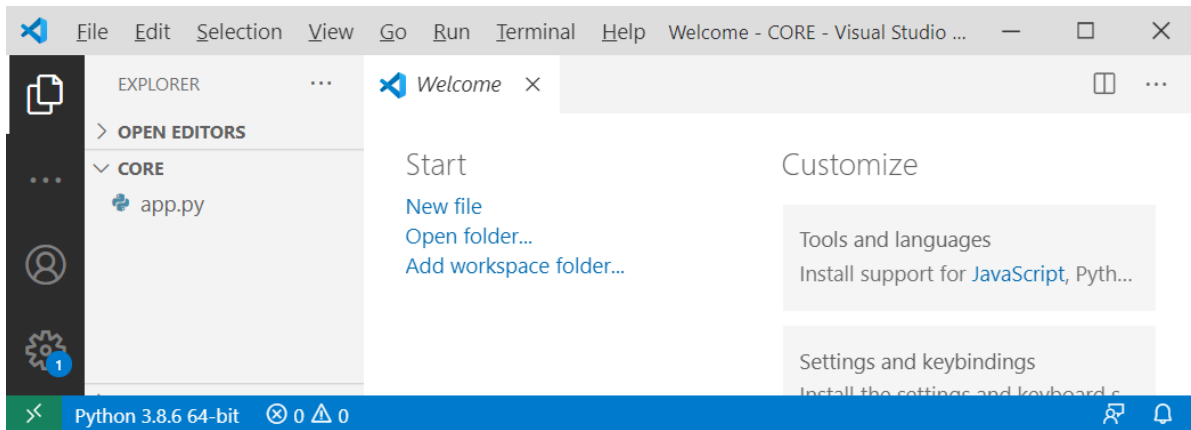


시작하기

▣ 작업 폴더 열기

■ 'File > Open Folder' 선택

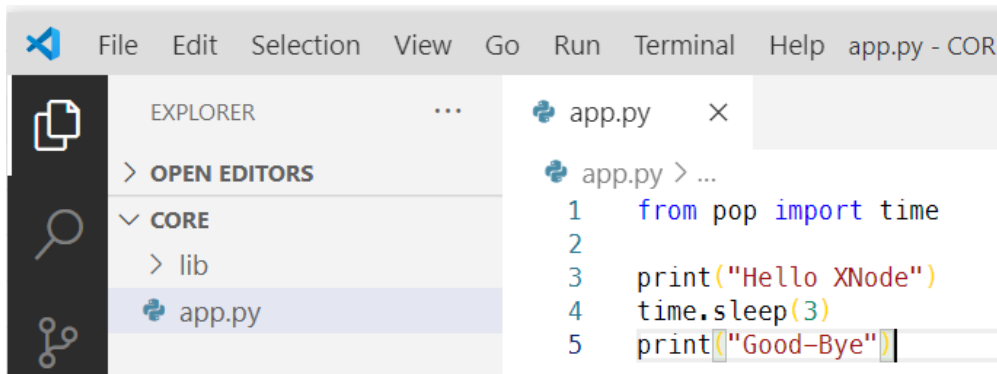
- 폴더 선택 창이 표시되면 작업 폴더 선택
- C:\XNode\CORE



시작하기

▣ 작업 파일 편집

- Explorer 창에서 app.py 선택
- 마이크로파이썬 구문 입력 후 'File > Save (Ctrl + S)'를 눌러 저장

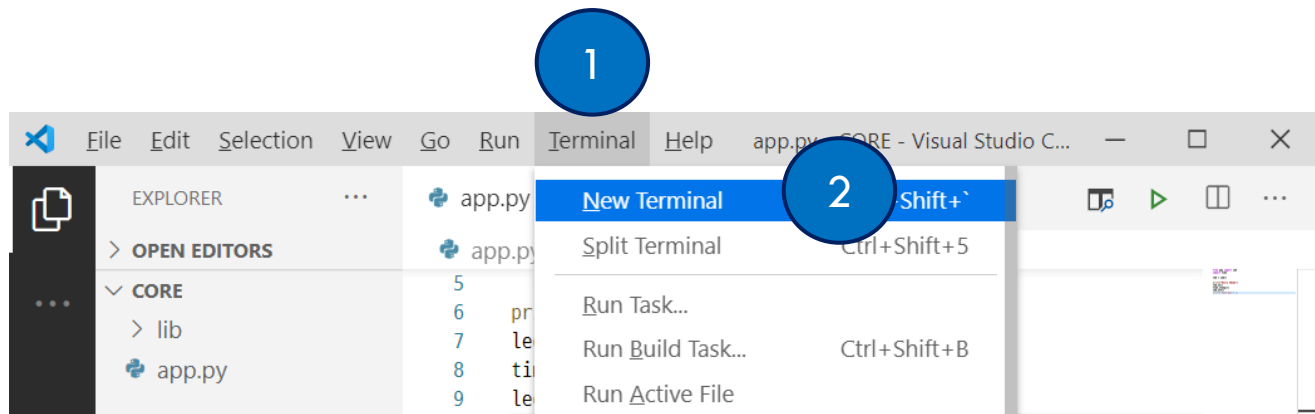


시작하기

▣ 터미널 창 표시

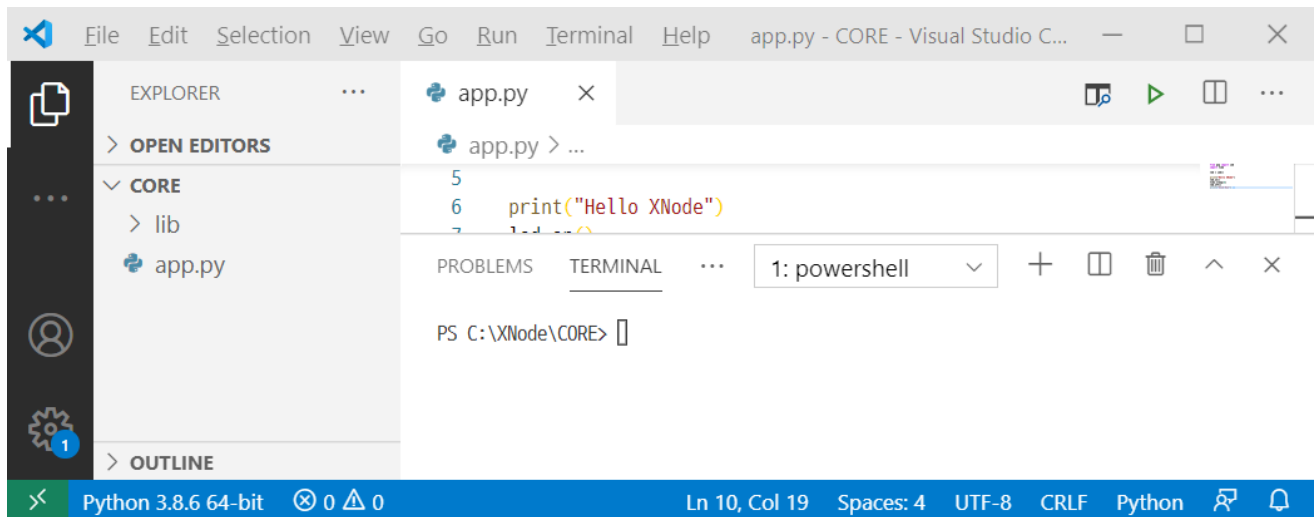
■ 'Terminal > New Terminal'

- 명령행 기반의 xnode 툴 실행



시작하기

- XNode 개발용 xnode 툴은 명령행 기반이므로 터미널 창에서 실행
 - XNode에서 마이크로파이썬 코드를 실행하거나 XNode 파일 시스템 관리



시작하기

□ xnode 툴 사용

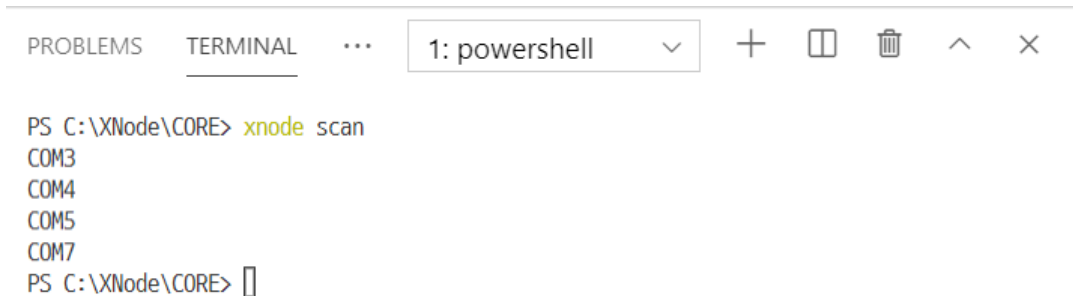
▣ XNode 연결 시리얼 포트 탐색

■ port 번호로 PC에 연결된 XNode 식별

- 다수의 XNode를 한꺼번에 프로그래밍할 경우 XNode를 하나씩 연결하면서 port 번호를 확인할 것

■ scan 인자로 시리얼 포트 탐색

- xnode scan



```
PROBLEMS  TERMINAL  ...  1: powershell  +  [ ]  [ ]  ^  x
PS C:\XNode\CORE> xnode scan
COM3
COM4
COM5
COM7
PS C:\XNode\CORE> [ ]
```

시작하기

▣ XNode의 파일 시스템 목록 표시

■ /: XNode 파일 시스템 루트 경로

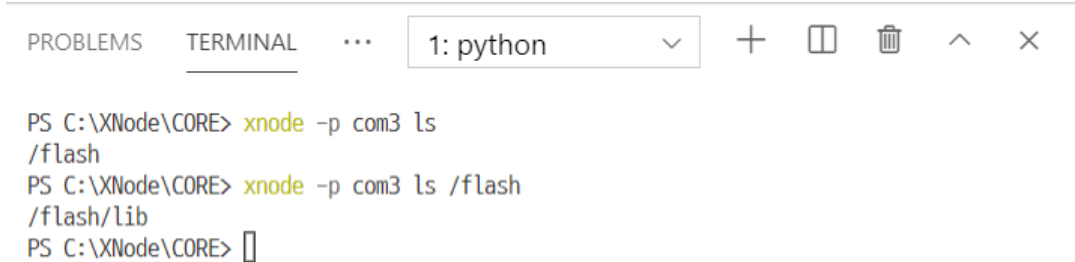
- 하위 flash 폴더만 가지며 이곳에 파일이나 폴더 생성 불가능

■ /flash: 사용자 작업 경로

- /flash/lib: 확장 모듈 경로. 포맷하면 빈 폴더

■ ls 인자는 XNode의 파일(폴더 포함) 목록 확인. 포트 번호는 필수이며 경로를 생략하면 ‘/’ 자동 할당

- `xnode -p <포트 번호> ls [XNode 경로]`



```
PROBLEMS  TERMINAL  ...  1: python  v  +  [ ]  [ ]  ^  x

PS C:\XNode\CORE> xnode -p com3 ls
/flash
PS C:\XNode\CORE> xnode -p com3 ls /flash
/flash/lib
PS C:\XNode\CORE> [ ]
```

시작하기

▣ PC의 폴더를 XNode 파일 시스템에 복사

■ Pop을 포함한 외부 라이브러리는 반드시 lib 경로(/flash/lib)에 위치해야 함

- XNode를 포맷했다면 반드시 기본 Pop 라이브러리(배포 USB/Library/CORE/lib/pop.py, core_a.py, core_b.py)를 lib에 복사해야 함

■ 폴더 복사는 put 인자 사용. XNode 경로를 생략하면 /flash 아래 같은 폴더 이름으로 복사

- XNode의 해당 경로에 동일한 폴더가 없으면 만들고, 있으면 그 안에 PC 폴더에 포함된 모든 파일 복사
 - 이미 같은 이름의 파일이 있으면 덮어씀

- `xnode -p <포트 번호> put <PC 폴더 경로> [XNode 경로]`

PROBLEMS TERMINAL ...

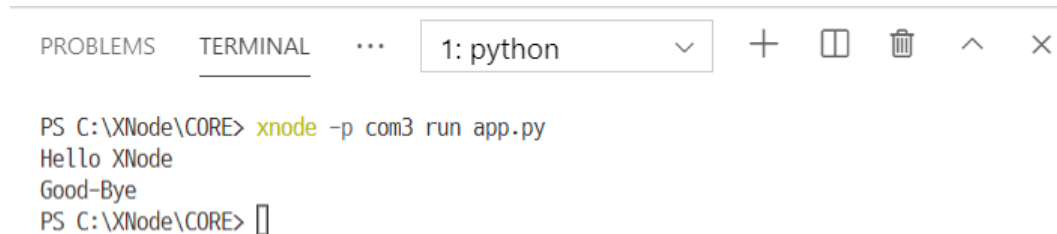
1: python



```
PS C:\XNode\CORE> xnode -p com3 put lib
PS C:\XNode\CORE> xnode -p com3 ls /flash/lib
/flash/lib/core_a.py
/flash/lib/core_b.py
/flash/lib/pop.py
PS C:\XNode\CORE>
```

시작하기

- XNode에서 마이크로파이썬 코드(이하 프로그램) 실행
 - PC에서 개발한 프로그램을 XNode의 메모리에 복사한 후 실행
 - 파일 시스템에 저장하지 않으므로 시리얼 케이블을 분리하거나 XNode 전원이 차단되면 종료함
 - run 인자의 추가 옵션을 생각하면 print() 결과를 시리얼로 읽어 터미널 창에 출력
 - `xnode -p <포트 번호> run [옵션] main.py`
 - "Hello XNode"를 출력하고 3초 후 "Goodbye" 출력
 - [주의] XNode의 /flash/lib 경로에 Pop 라이브러리가 설치되어 있어야 함



The screenshot shows a terminal window with a title bar containing '1: python' and standard window controls. The terminal output shows a command being executed in a PowerShell prompt, followed by two lines of output and a new prompt line.

```
PROBLEMS  TERMINAL  ...  1: python  v  +  □  □  ^  x

PS C:\XNode\CORE> xnode -p com3 run app.py
Hello XNode
Good-Bye
PS C:\XNode\CORE> 
```

시작하기

■ run 인자의 추가 옵션

- run 인자로 프로그램을 실행하면 종료 전까지 시리얼을 통해 print() 문의 출력을 기다림
- -n: print() 문의 출력을 기다리지 않고 명령 종료.
 - 프로그램은 XNode에서 계속 실행될 수 있음
 - XNode에서 시리얼로 출력하는 데이터를 다른 툴(PuTTY, smon 등)로 확인할 때 사용
- -i: PC의 키보드 입력을 시리얼을 통해 XNode에 실행 중인 프로그램에 전달
 - XNode에서 실행 중인 프로그램은 시리얼로부터 데이터를 읽는 구문이 구현되어 있어야 함
 - 누른 키를 터미널 창에 표시함 (Echo on)
- -ni (또는 -n -i): -i와 같으나 누른 키를 터미널 창에 표시하지 않음 (Echo off)

시작하기

▣ XNode에 파일 복사

- XNode가 시작될 때 /flash/main.py가 존재하면 이를 자동으로 실행
 - XNode에 전원을 공급하거나 시작된 상태에서 리셋 버튼을 누름
 - PC에서 개발한 프로그램을 XNode의 /flash 경로에 main.py로 복사
- 폴더 복사처럼 put 인자 사용. XNode 경로를 생각하면 /flash 아래 같은 파일명으로 복사
 - `xnode -p <포트 번호> put <PC 파일 경로> [XNode 경로]`
 - `print()` 문의 출력 결과는 PuTTY와 같은 별도의 시리얼 툴을 연결해 확인

PROBLEMS

TERMINAL

...

1: powershell

▼

+

□

🗑

^

×

```
PS C:\XNode\CORE> xnode -p com3 put app.py /flash/main.py
```

```
PS C:\XNode\CORE> xnode -p com3 ls /flash
```

```
/flash/lib
```

```
/flash/main.py
```

```
PS C:\XNode\CORE> █
```

시작하기

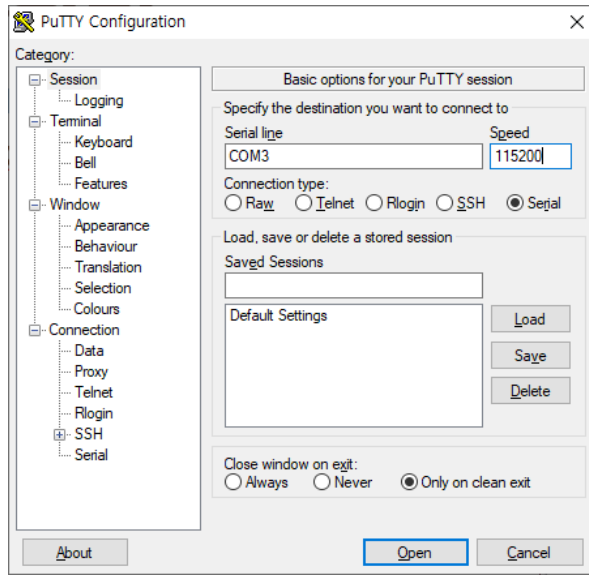
- XNode의 print() 출력 결과 확인

- XNode의 시리얼 출력 결과를 확인하기 위해 PuTTY 실행

- Connection type에서 Serial을 선택한 후 포트 번호와 전송 속도 설정 후 Open으로 실행

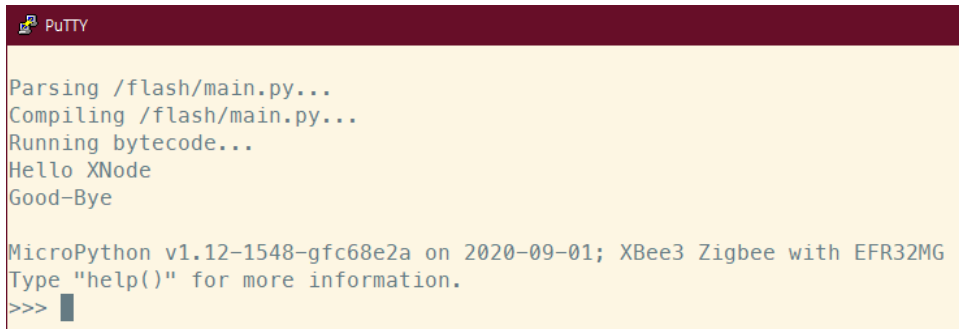
- Serial line to connect to: COM 3 (포트 번호는 다를 수 있음)

- Speed (baud): 115200



시작하기

- XNode의 리셋 버튼을 눌러 재시작
 - 프로그램이 종료하면 자동으로 마이크로파이썬 인터프리터가 실행됨



```
PuTTY

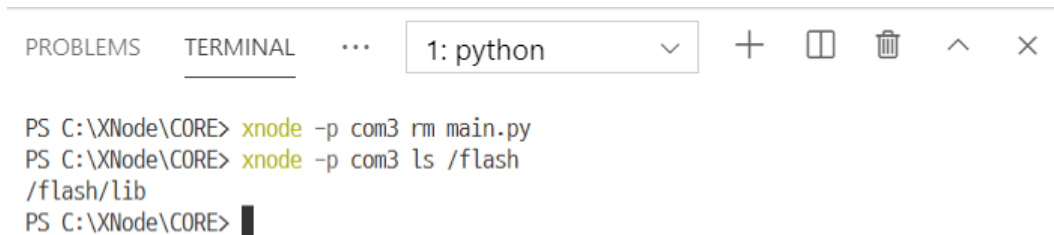
Parsing /flash/main.py...
Compiling /flash/main.py...
Running bytecode...
Hello XNode
Good-Bye

MicroPython v1.12-1548-gfc68e2a on 2020-09-01; XBee3 Zigbee with EFR32MG
Type "help()" for more information.
>>> █
```

시작하기

▣ XNode의 파일 삭제

- 한 번에 하나의 파일만 삭제할 수 있음
- rm 인자로 해당 파일 삭제
 - `xnode -p <포트 번호> rm <파일명>`



The screenshot shows a VS Code terminal window with the 'TERMINAL' tab selected. The title bar indicates the terminal is running '1: python'. The terminal output shows three commands being executed in a Windows command prompt environment (PS C:\XNode\CORE>):

```
PS C:\XNode\CORE> xnode -p com3 rm main.py
PS C:\XNode\CORE> xnode -p com3 ls /flash
/flash/lib
PS C:\XNode\CORE> 
```

시작하기

▣ XNode의 폴더 삭제

- 한 번에 하나의 폴더만 삭제할 수 있음
- rmdir 인자로 폴더 삭제
 - `xnode -p <포트 번호> rmdir <폴더명>`



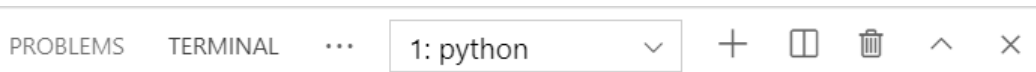
The screenshot shows a terminal window with a title bar containing '1: python' and standard window controls. The terminal output shows a PowerShell prompt at 'C:\XNode\CORE' where the user runs 'xnode -p com3 rmdir /flash/lib' and 'xnode -p com3 ls /flash', followed by a new prompt line.

```
PROBLEMS  TERMINAL  ...  1: python  +  [icon]  [icon]  ^  x
PS C:\XNode\CORE> xnode -p com3 rmdir /flash/lib
PS C:\XNode\CORE> xnode -p com3 ls /flash
PS C:\XNode\CORE> 
```

시작하기

▣ XNode의 파일 시스템 초기화

- 수 초 정도 소요되며, 사용자 작업을 모두 삭제하고 초기 상태로 만들
 - /flash 경로의 모든 파일과 폴더를 삭제한 후 /flash 아래 빈 lib 폴더 생성
- format 인자를 사용하며 Type A(LoRa)는 a, Type B(Zigbee V3.0)는 b 옵션 사용
 - `xnode -p <포트 번호> format <a | b>`



```
PS C:\XNode\CORE> xnode -p com3 format b
PS C:\XNode\CORE> xnode -p com3 ls /flash
/flash/lib
_
```

- [주의] 초기화가 끝나면 반드시 XNode의 lib 경로에 기본 Pop 라이브러리(pop.py, core_a.py, core_b.py)를 다시 복사해 둘 것
 - `xnode -p com3 put lib`

시작하기

- XNode에서 마이크로파이썬 REPL (read-eval-print loop) 실행
 - 간단한 테스트 목적으로 사용하며, 마이크로파이썬 구문 입력하면 즉시 실행
 - repl 인자를 사용하며 종료는 <Ctrl>x
 - `xnode -p <포트 번호> repl`

```
PROBLEMS  OUTPUT  TERMINAL  ...
1: python

PS C:\XNode\CORE> xnode -p com3 repl
...

>>>
>>> import sys
>>> sys.path
['', '/flash', '/flash/lib']
>>> help('modules')
__main__      sys          uhashlib      ustruct
builtins      uarray       uio           utime
digi          ubinascii    ujson         xbee
gc            ucryptolib   umachine
micropython   uerrno       uos
```

시작하기

□ Pop과 같은 외부 모듈 로드 구문 비교

▣ `import pop`: `pop`만 메모리에 로드. 식별 이름을 사용할 땐 접두사로 `pop.` 사용

■ `led = pop.Led()`

■ 이후 동적으로 해당 식별 이름(예: `Led`)을 메모리에 로드

▣ `from pop import *`: `pop`에 포함된 모든 식별 이름을 메모리에 로드

■ `led = Led()`

■ 모든 이름을 사용하지 않는다면 메모리 낭비가 있으므로 권장하지 않음

▣ `from pop import <식별 이름>`: `pop`에 포함된 특정 식별 이름만 메모리에 로드

■ `led = Led()`

시작하기

□ 시리얼 통신

- UART로 불리는 범용 직렬 통신으로 PC와 같은 지능형 장치와 데이터 교환
 - XNode는 USB to Serial 칩이 내장되어 USB 케이블을 통해 시리얼 통신 수행
 - 통신 설정은 115200bps, 8 Data bits, 1 Stop bits, None Parity
- XNode에서 print()로 출력하는 문자열은 모두 시리얼 출력으로 전달됨
 - Pop은 Uart 클래스 추가 제공
 - Uart(): Uart 객체 생성
 - Uart.read(size): 시리얼로부터 데이터 읽기. size 매개변수에 아무런 값을 주지 않을 경우 반환할 값이 없을 때 None 반환. size 매개변수가 있으면 해당하는 byte만큼 대기하여 반환
 - size: 읽은 데이터 수
 - Uart.write(n): 시리얼에 데이터 쓰기
 - n: 쓸 데이터

시작하기

■ 에코 테스트

- CORE 폴더를 선택한 후 메뉴에서 'File > New File (Ctrl + N)을 눌러 새로운 파일 생성
 - core_uart.py
- PC에서 전송한 문자를 XNode가 읽으면 다시 반송하는 코드 작성
 - write()는 print()와 달리 문자열 끝에 줄 바꿈 문자('\n') 문자를 직접 추가해야 함

```
01: from pop import Uart
02:
03: uart = Uart()
04: uart.write("Start UART\n")
05:
06: for _ in range(15):
07:     data = uart.read(1)
08:     uart.write(data)
```

시작하기

- 프로그램을 실행한 후 15개의 문자(예: 'abcdefghijklmno') 입력
 - 프로그램을 실행할 때 run 인자와 함께 -i 또는 -ni 옵션 사용해 PC 입력을 시리얼로 XNode에 전달
 - 입력 에코 포함 실행: `xnode -p com3 run -i uart.py`

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\XNode\CORE> xnode -p com3 run -i core_uart.py
Start UART
aabbccddeeffgghhiijjkkllmmnnoo
PS C:\XNode\CORE> █
```

- 입력 에코 제거 실행: `xnode -p com3 run -ni uart.py`

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\XNode\CORE> xnode -p com3 run -ni core_uart.py
Start UART
abcdefghijklmno
PS C:\XNode\CORE> █
```

시작하기

- **센서 모니터링 툴(xmon)을 이용해 센서 데이터를 실시간으로 PC에서 시각화**
 - 환경 센서로부터 수집한 온도, 기압, 습도, 가스값을 print()로 PC에 실시간 전달하면서 xmon으로 확인

```
01: from pop import time
02: from pop import Tphg
03:
04: tphg = Tphg()
05:
06: t0 = time.time()
07:
08: while True:
09:     if time.time() - t0 >= 100:
10:         t0 = time.time()
11:
12:         t, p, h, g = tphg.read()
13:         print("%.2f %.2f %.2f %d"%(t, p, h, g))
```

시작하기

- 먼저 정상적으로 센서 데이터를 시리얼로 PC에서 읽을 수 있는지 확인

- 강제 종료는 <Ctrl>c

```
PS C:\XNode\CORE> xnode -p com3 run core_uart_xmon.py
26.71, 1017.13, 10.98, 1134632
26.86, 1017.14, 11.03, 409126
27.12, 1017.15, 11.08, 429333
27.40, 1017.15, 11.13, 450728
27.65, 1017.17, 11.17, 465136
27.87, 1017.17, 11.21, 479807
```

Aborted!

- 정상적인 데이터 출력이 확인되면 -n 옵션으로 실행한 후 센서 모니터링 툴 실행

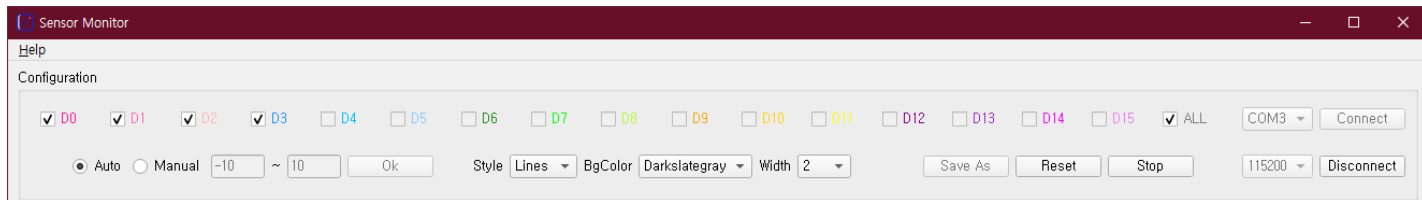
```
PS C:\XNode\CORE> xnode -p com3 run -n core_uart_xmon.py
PS C:\XNode\CORE> xmon
```

■ XNode 연결 포트를 확인한 후 'Connect' 버튼을 눌러 모니터링 시작



시작하기

- Configuration를 통해 모니터링 방법 설정
 - 최대 16개 항목을 플롯에 커브로 시각화하면서 값 테이블에도 함께 출력
 - 항목 범위는 XNode의 출력 데이터를 분석에 파악
 - XNode의 출력은 d0, d1, d2, ... 형식의 바이트 문자열로 각 항목 구분은 쉼표(,) 또는 공백(' ')
 - XNode의 출력 반복은 10Hz (0.1초) ~ 20Hz (0.05초) 권장
 - 플롯 영역에 커브로 센서 데이터 시각화할 때 배경 및 커브 모양과 굵기, y축 표시 방법 선택 가능
 - x축: 모니터링을 시작한 시점을 기준으로 차례로 증가한 밀리초 단위 시간
 - y축: XNode에서 읽은 센서 데이터로 항목 개수에 대응하여 커브가 그려짐



시작하기

■ 항목 선택

- D0 ~ D15: 표시 항목. 최초 실행 시 자동으로 범위가 결정되며 결정된 범위 내에서 표시 항목 선택 가능
- ALL: 인식 범위의 해당 항목 모두 선택 또는 모두 해제

■ y축 표시

- Auto: 데이터값에 의해 자동 조절
- Manual: 사용자가 최솟값과 최댓값을 입력한 후 'Ok' 버튼을 누르면 해당 범위로 고정

■ 플롯 표시 방법

- Style: 커브 종류 선택
- BgColor: 플롯 배경색 선택
- Width: 커브 굵기 선택

■ 표시 제어 및 저장

- Save As: 종료 또는 중단 상태일 때 테이블에 저장된 데이터를 파일로 저장 후 초기화
- Reset: 누를 때마다 시각화 초기화
- Stop: 시각화 중단. 중단하면 Start로 변경되며 다시 누르면 재개