

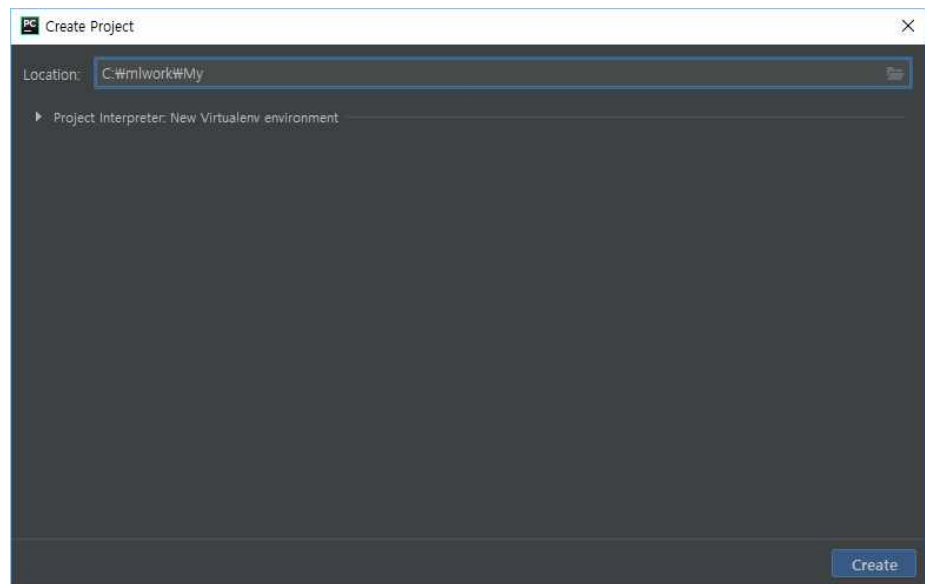
제7장

파이선 클래스 모듈 작성하기

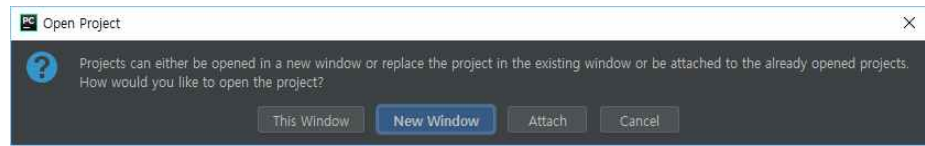
Step1

가장 간단한 파이썬 프로그램

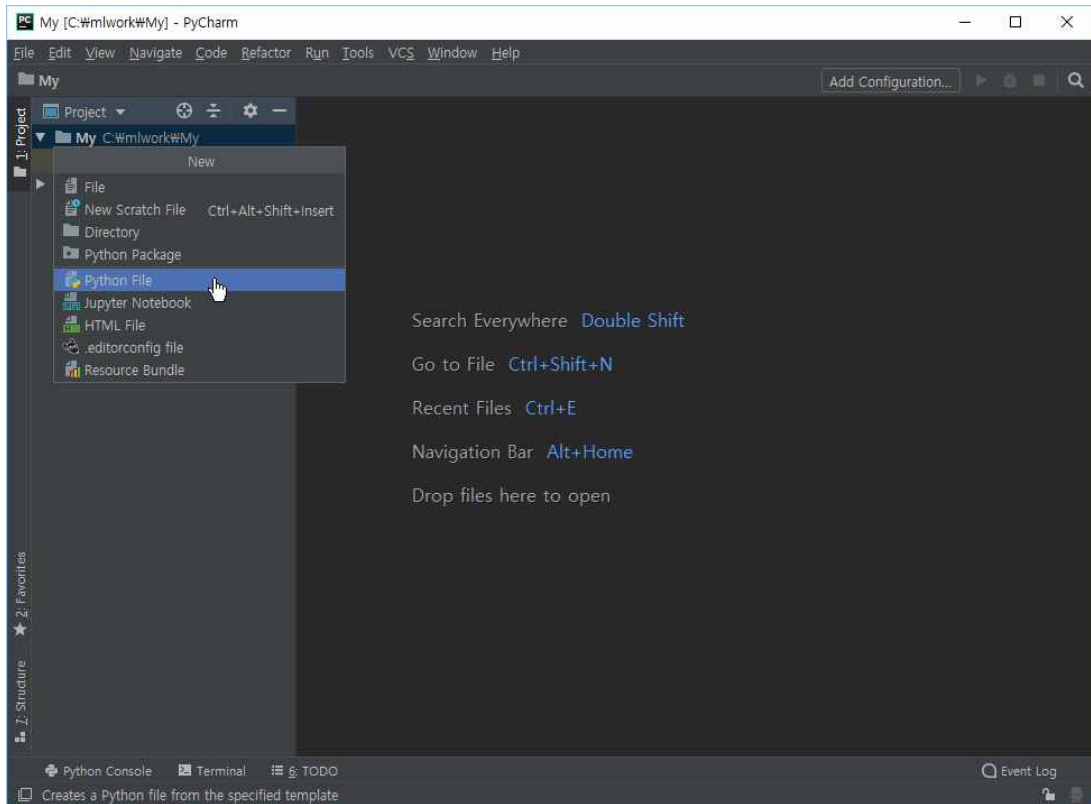
먼저 프로젝트를 만들겠습니다. PyCharm을 실행한 후 File | New Project 메뉴 항목을 선택하면 아래와 같이 대화 상자가 표시됩니다. 여기에서 여러분이 원하는 폴더 + 원하는 이름을 입력합니다. 아래의 경우 C:\mlwork\My 입력하였고, 이는 mlwork 폴더에 My 프로젝트를 생성하는 것입니다.

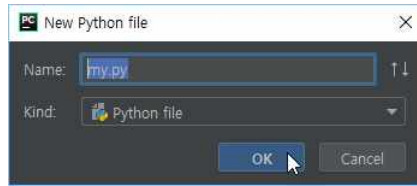


대화 상자 아래에 있는 Create 버튼을 누르면 프로젝트가 생성되는데, 이때 현재 창에서 작업할 것인지 새로운 창을 표시하여 작업할 것인지를 묻습니다.



이제 File | New... 메뉴 항목을 선택한 후 아래와 같이 Python File을 선택하고 원하는 파일 이름을 입력합니다.



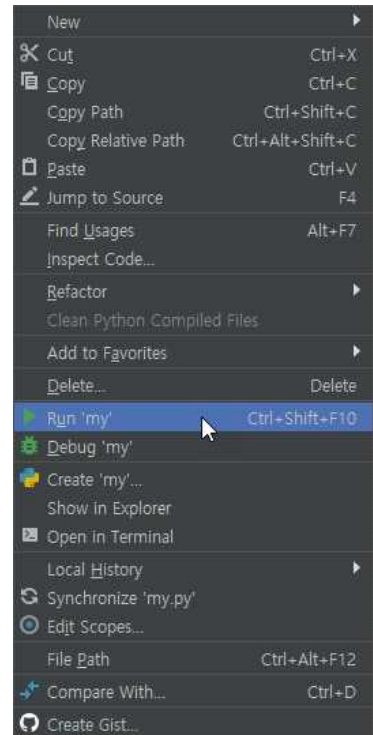


그런 다음, 표시되는 창에 아래의 코드를 입력합니다.

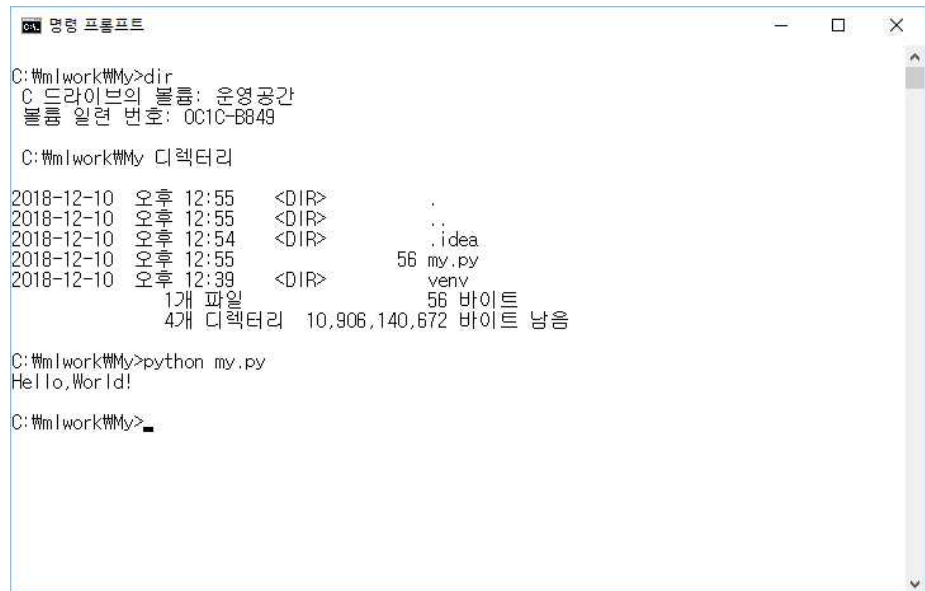
```
print('Hello,World!')
```

파이선에서는 실행이 시작되는 메인 함수가 따로 없습니다. 대신 들여쓰기하지 않은 모든 코드, 이를 레벨 0 코드라고 하는데요, 이러한 모든 코드가 실행됩니다. 우리의 경우에는 한 줄만 있게 됩니다. 코드 작성 후 파일 이름 위에서 오른쪽 버튼을 클릭하면 오른쪽과 같이 표시됩니다. 여기에서 Run 메뉴 항목을 선택하면 우리가 작성한 코드를 실행할 수 있습니다. 프로그램을 실행한 결과 'Hello,World!' 문자열이 아래 작은 창에 표시됨을 알 수 있습니다.

이로써 문자열을 표시하는 아주 간단한 파이선 프로그램을 작성해 보았습니다.



PyCharm에서 프로그램을 실행할 수도 있지만, 아래와 같이 도스 창에서 실행할 수도 있습니다.



```
C:\work\My>dir
C 드라이브의 볼륨: 운영공간
볼륨 일련 번호: 0C1C-B849

C:\work\My 디렉터리
2018-12-10 오후 12:55 <DIR>          .
2018-12-10 오후 12:55 <DIR>          .idea
2018-12-10 오후 12:54 <DIR>          .
2018-12-10 오후 12:55          56 my.py
2018-12-10 오후 12:39 <DIR>          venv
                        1개 파일          56 바이트
                        4개 디렉터리 10,906,140,672 바이트 남음

C:\work\My>python my.py
Hello,World!

C:\work\My>
```

이처럼 우리는 두 가지 방법으로 실행해보았습니다. 1) PyCharm에서 직접 실행하는 방법, 2) 콘솔 창에서 실행하는 방법. 사실 더 정확한 프로그램 코드는 아래와 같습니다. 하지만 우리가 만든 my.py가 사용되는 방법이 또 있습니다. 그것은 바로 다른 파이썬 프로그램에서 임포트(import)하여 사용하는 것입니다. 프로그램을 직접 실행하는 것과 다른 프로그램에서 import되어 사용되는 것을 구분하기 위하여 아래와 같이 코드를 작성할 수도 있습니다.

```
if __name__ == '__main__':
    print('Hello,World!')
```

이는 아래와 같은 의미가 있습니다.

‘이 프로그램을 여러분이 직접 실행할 경우 print 문 실행한다.’ 여러분이 직접 실행하지 않았을 때는 print 문이 실행되지 않는다.

그렇다면 여러분이 직접 실행하지 않는 경우는 어떤 경우인가요? 바로 import 할 경우입니다. 즉, 임포트될 경우에는 print 문이 실행되지 않습니다.

여기에서 `__name__`은 시스템에 내장되어있는 변수로서 콘솔 창이나 PyCharm에서 직접 실행할 경우 참이 됩니다.



조금 복잡한 파이썬 프로그램

이제 다음과 같이 조금 복잡한 프로그램을 작성해 봅니다.

```
if __name__ == '__main__':  
    iX = 2  
    iY = 3  
    iResult = iX + iY  
    print('Sum =', iResult)
```

2와 3을 각각 iX 변수와 iY 변수에 넣은 후 두 값을 더한 후 출력하는 코드입니다. 프로그램을 실행할 경우 아래와 같이 결과가 잘 출력됨을 알 수 있습니다.

```
C:\Wmlwork\WMy\Wvenv\WScripts\Wpython.exe C:/mlwork/My/my.py  
Sum= 5
```

```
Process finished with exit code 0
```

앞의 코드를 다음과 같이 main 함수로 추상화하면 어떨까요?

```
def main():  
    iX = 2  
    iY = 3  
    iResult = iX + iY  
    print('Sum =', iResult)  
  
if __name__ == '__main__':  
    main()
```

main 함수로 작성하였습니다. 참고로 파이선에서는 변수에 어떤 값을 할당하기만 하면 값에 맞는 자료형으로 변수가 자동으로 생성됩니다. 예를 들어 아래와 같이 코드를 작성하면 자동으로 a, b 변수가 생성됩니다.

```
a = 1  
b = [1, 2, 3]
```

따라서 앞에서 작성하였던 코드에서 iX와 iY에 값을 할당만 해도 변수 iX와 iY가 자동으로 생성됩니다. 아래의 경우 main 함수 내에서 변수를 만드는 것이므로 지역변수로서 iX와 iY가 생성됩니다.

```
def main():  
    iX = 2  
    iY = 3  
    iResult = iX + iY  
    print('Sum =', iResult)
```

```
if __name__ == '__main__':  
    main()
```

만일 iX와 iY를 함수 **밖으로 꺼내면** 전역변수가 됩니다.

```
iX = 2  
iY = 3
```

```
def main():  
  
    iResult = iX + iY  
    print('Sum =', iResult)
```

```
if __name__ == '__main__':  
    main()
```

전역변수는 main 함수 내에서 읽을 수는 있지만, 기본적으로 어떤 값을 넣을 수는 없습니다. 예를 들어 다음과 같이 하면 어떤 일이 발생할까요?

```
iX = 2  
iY = 3
```

```
def main():  
    iX = 4  
    iY = 5  
    iResult = iX + iY  
    print('Sum =', iResult)
```



```
if __name__ == '__main__':  
    main()
```

전역변수 iX와 iY에 4, 5를 저장하는 것이 아니라 또 다른 지역변수 iX와 iY를 넣는 모양이 됩니다. 즉, 전혀 다른 변수가 새로 생성이 되는 셈이지요. 이 경우에는 global 키워드를 사용할 수 있습니다.

```
iX = 2  
iY = 3
```

```
def main():  
    global iX, iY  
    iX = 4  
    iY = 5  
    iResult = iX + iY  
    print('Sum =', iResult)
```

```
if __name__ == '__main__':  
    main()
```

즉, global 키워드를 이용하면 새로운 지역변수를 만들지 않고 전역변수를 사용한다는 의미입니다. 이제 값을 할당하는 코드를 assign 함수로 추상화하겠습니다.

```
iX = 2  
iY = 3
```

```
def assign(a, b):
```

```

global iX, iY
iX = a
iY = b

def main():
    global iX, iY
    assign(2, 3)
    iResult = iX + iY
    print('Sum =', iResult)

if __name__ == '__main__':
    main()

```

새로운 지역변수가 아닌 전역변수를 이용하기 위하여 global로 선언해줍니다.



코드 추상화와 함수

두 수를 더하는 코드를 add 함수로 추상화합니다.

```

iX = 2
iY = 3

def assign(a, b):
    global iX, iY
    iX = a
    iY = b

```

```
def add():
    return iX + iY

def main():
    global iX, iY
    assign(2, 3)
    iResult = add()
    print('Sum =', iResult)

if __name__ == '__main__':
    main()
```

앞에서 add 함수를 볼까요? 값을 넣는 코드는 없고 값을 읽는 부분만 있지요?
따라서 값을 할당하는 코드가 없으므로 global 선언이 필요 없습니다.

이제 main 함수에서 선언한 global 문은 필요가 없게 되었습니다. 왜냐하면
main 함수에서 더 이상 iX와 iY를 사용하고 있지 않으니까요. 따라서 global
iX, iY 문은 삭제하는 것이 맞습니다.



데이터 추상화와 클래스

이제까지 작성한 코드를 객체지향 프로그램으로 바꿔보겠습니다. 다음 프로그램
에서 서로 관련된 코드는 무엇일까요? 전역변수를 기준으로 이 전역변수를 사
용하는 코드, 이

```

iX = 2
iY = 3

def assign(a, b):
    global iX, iY
    iX = a
    iY = b

def add():
    return iX + iY

def main():
    assign(2, 3)
    iResult = add()
    print('Sum =', iResult)

if __name__ == '__main__':
    main()

```

서로 관련된 것들을 묶는다, 이것이 추상화의 시작입니다. 아래와 같이 말이죠.

```

class XXX:
    iX = 2
    iY = 3

    def assign(self, a, b):
        global iX, iY
        iX = a

```

```
iY = b
```

```
def add(self):  
    return iX + iY
```

```
def main():  
    assign(2, 3)  
    iResult = add()  
    print('Sum =', iResult)
```

```
if __name__ == '__main__':  
    main()
```

클래스로 묶으면서 조금 달라진 부분이 보입니다. 즉, assign, add 함수의 첫 번째 파라미터로 self가 추가되었습니다. 이제 main 함수도 오류가 발생하지 않도록 수정해 봅니다. 바로 아래와 같습니다.

```
def main():  
    gildong = XXX()  
    gildong.assign(2, 3)  
    iResult = gildong.add()  
    print('Sum =', iResult)
```

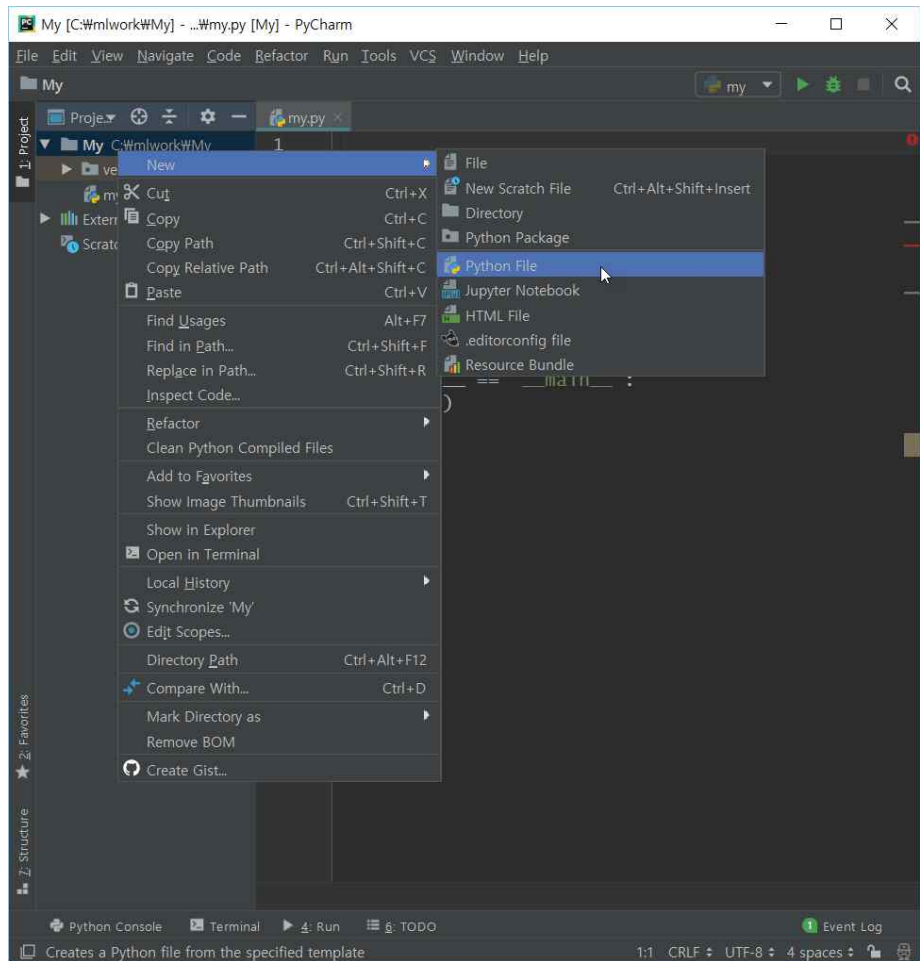
gildong 객체를 통하여 assign 멤버 함수와 gildong 멤버 함수를 호출하면 멤버 함수의 첫 번째 파라미터로 자동으로 객체의 주소가 전달됩니다. 그렇다고 무조건 첫 번째 파라미터로 객체의 주소가 전달되는 것은 아닙니다. 예를 들어 첫 번째 파라미터로 self를 주지 않으면 gildong 객체를 통해 호출하면 오류가 발생

합니다. 이 경우에는 마치 정적 멤버 함수처럼 클래스로 호출하면 오류가 발생하지 않습니다. self를 파라미터로 전달하는 경우에는 클래스로 호출하면 오류가 발생합니다. 아무튼 이제 멤버 함수의 파라미터로 self를 넣을 수도 있고 그렇지 않을 수도 있다는 것을 알게되었습니다. 다만 self가 있을 경우에는 객체를 통하여 멤버 함수를 호출해야 하는 것이고, 없을 경우에는 정적 멤버 함수처럼 클래스 이름을 통하여 멤버 함수를 호출해야 한다는 것입니다.

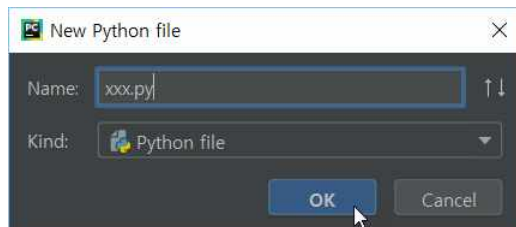


클래스 모듈 재사용

클래스 XXX를 독립된 파이썬 파일에 저장합니다. 이를 위해 다음과 같이 메뉴를 선택합니다.



그런 다음, xxx.py 파일을 생성합니다.



그리고 아래와 같이 XXX 클래스 코드만을 위 파일로 옮깁니다.

```
class XXX:
    iX = 2
    iY = 3

    def assign(self, a, b):
        global iX, iY
        iX = a
        iY = b

    def add(self, ):
        return iX + iY
```

XXX 클래스를 다른 파일로 옮겼으므로 나머지 코드에서 오류가 발생합니다.
xxx.py 파일 혹은 모듈로부터 XXX 클래스를 가져오려면 아래와 같이 from
import 문으로 선언해야 합니다.

```
from xxx import XXX

def main():
    gildong = XXX()
    gildong.assign(2, 3)
    iResult = gildong.add()
    print('Sum =', iResult)

if __name__ == '__main__':
    main()
```


혹은 다음과 같이 모듈을 가져온 후 클래스 XXX 앞에 모듈 이름 xxx를 붙여도 됩니다.

```
import xxx

def main():
    gildong = xxx.XXX()
    gildong.assign(2, 3)
    iResult = gildong.add()
    print('Sum =', iResult)

if __name__ == '__main__':
    main()
```

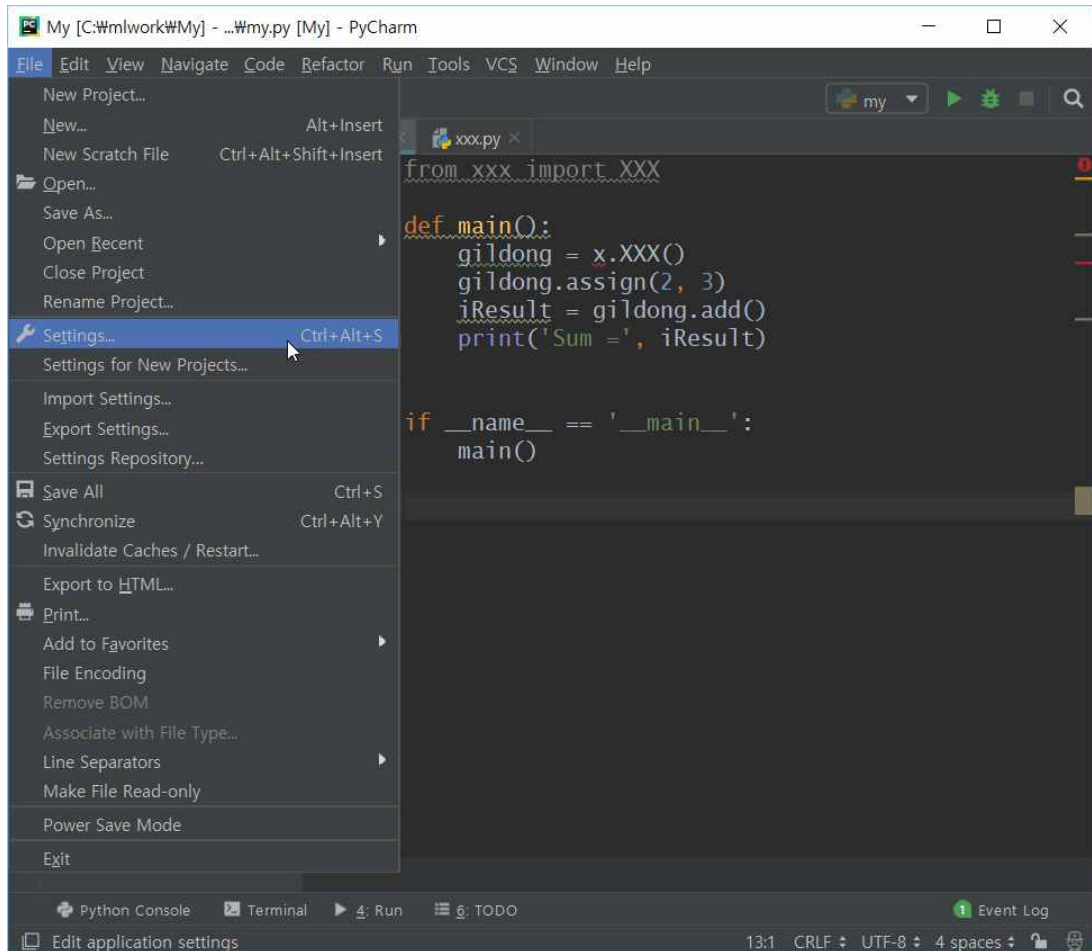
조금은 더 간편하게 아래와 같이 작성할 수도 있습니다. 이 경우에는 모듈 xxx를 x라는 이름으로 가져옵니다. 따라서 XXX 클래스를 이용할 때에는 x.XXX로 사용할 수 있습니다.

```
import xxx as x

def main():
    gildong = x.XXX()
    gildong.assign(2, 3)
    iResult = gildong.add()
    print('Sum =', iResult)

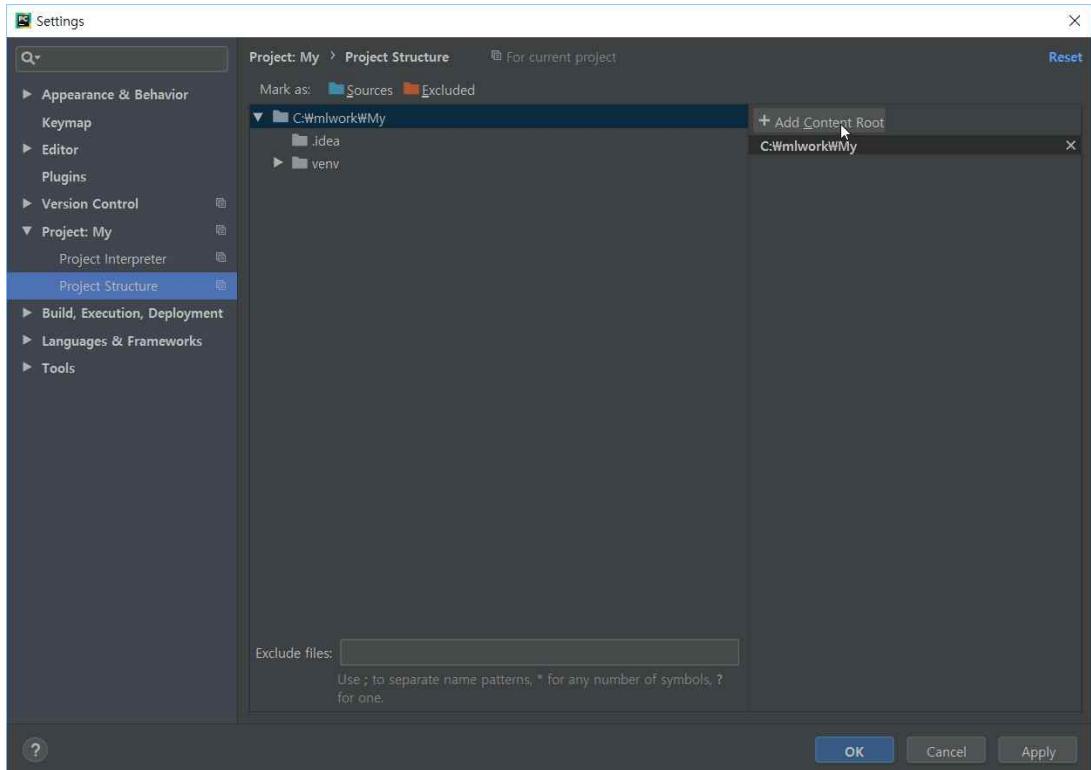
if __name__ == '__main__':
    main()
```

하지만 현재 작성되어 있는 xxx.py 파일은 프로그램을 작성하는 프로젝트 폴더에 들어있습니다. 이를 외부 폴더에 보관해두면 나중에 사용하기 편리합니다. 즉, 나중에 라이브러리 모듈로 사용할 파일들을 한 폴더에 모아두고 사용하는 것이지요. 이를 위해 다음과 같이 File | Settings 메뉴 항목을 선택합니다.

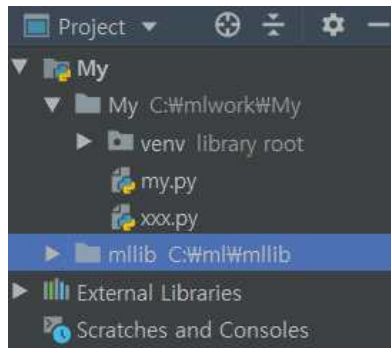


왼쪽 Project: My 메뉴를 확장하고 Project Structure를 선택한 후 오른쪽에서 Add Content Root를 선택합니다. 그런 다음 여러분의 라이브러리 폴더를 선택

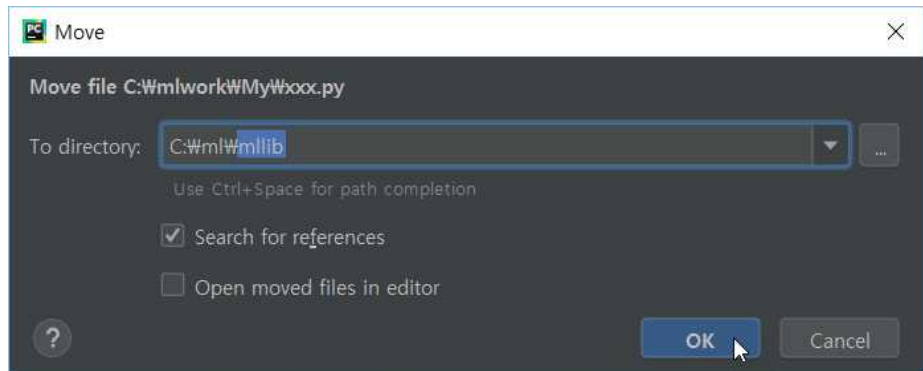
합니다. 만일 따로 없다면 아무 폴더나 생성하여 추가합니다. 저의 경우에는 c:\WmlWmllib 폴더를 선택합니다.



그랬더니 아래와 같이 mllib 폴더가 My 프로젝트에 추가되었습니다. 이제 마우스를 이용하여 xxx.py 모듈 파일을 mllib로 드래그-앤-드롭합니다.



그러면 아래와 같이 파일을 옮길 것인지를 묻습니다.



OK 버튼을 누르면 이동합니다. 이제 모듈 파일이 외부 mllib 폴더로 이동되었습니다. 나중에 필요할 때 바로 사용할 수 있습니다.