

08 파이썬 실전 프로젝트

이번 장에서는 지금까지 배운 파이썬 기초 문법을 활용해 간단한 프로젝트를 진행해보겠습니다. 이를 통해 지금까지 배운 파이썬 문법이 실제 프로그램 개발에 어떻게 사용되는지 감을 잡을 수 있을 것입니다. 그리고 더 나아가 프로그램 개발에 대한 자신감도 얻을 수 있을 것입니다.

프로그래밍을 처음 배우면 언어의 문법은 어느 정도 알 것 같은데, 막상 프로그램을 작성하려고 하면 어디서부터 어떻게 시작해야 할지 감이 오지 않는 경우가 많습니다. 이러한 단계에서 더 발전하려면 꼭 코드를 직접 타이핑해보면서 이번 장에서 소개하는 프로젝트를 완성해 보기 바랍니다.

1) PyCharm 을 이용한 개발 환경 구축

지금까지 Python IDLE 를 이용해 파이썬 코드를 작성했습니다. Python IDLE 는 사용하기 편하고 아나콘다 같은 파이썬 배포판이나 파이썬 공식 설치 프로그램만을 설치해도 바로 사용할 수 있어서 초보자에게 적합한 개발 환경입니다. 하지만 조금 규모가 있는 프로그램을 작성할 때는 Python IDLE 만으로는 부족한 점이 있습니다.

전문 개발자는 보통 통합 개발 환경이라고 하는 IDE(Integrated Development Environment)를 이용해 프로그램을 개발합니다. 음악을 업으로 하는 사람들이 좋은 악기를 사용해서 연주하는 것을 생각해보면 더 좋은 개발 환경을 사용하는 상황이 쉽게 이해될 겁니다.

IDE 는 보통 변수나 함수 이름의 자동 완성 기능을 포함하고 있어 개발자가 더욱 빠르게 코드를 작성할 수 있도록 합니다. 또한, 코드의 실행과 결과 확인을 IDE 를 통해 한 번에 처리할 수 있어 편리합니다. 그뿐만 아니라 작성한 프로그램이 정상적으로 동작하지 않을 때는 디버거(Debugger)를 사용하여 코드를 쉽게 분석할 수 있는 기능도 제공합니다. 최근에는 SVN 이나 Git 과 같은 버전 관리 시스템도 지원하고 있어 프로그램 개발의 전반적인 과정을 IDE 를 통해 모두 처리할 수 있습니다. 여러분도 이제부터는 IDE 를 사용하여 파이썬 프로그램을 개발해보도록 합시다.

파이썬용 IDE 에는 비주얼 스튜디오 기반의 PTVS(Python Tools Visual Studio), 이클립스 기반의 PyDev, 그리고 최근 인기가 높은 JetBrains 의 PyCharm 이 있습니다. 이 책에서는 이 가운데 최근 인기가 높고 무료로 사용할 수 있는 PyCharm 을 사용하겠습니다.

- PTVS (<http://microsoft.github.io/PTVS/>)
- PyDev (<http://www.pydev.org/>)
- PyCharm (<https://www.jetbrains.com/pycharm/download/>)

1-1) PyCharm 설치 프로그램 다운로드

컴퓨터에 PyCharm 을 설치하기 위해 먼저 PyCharm 다운로드

페이지(<https://www.jetbrains.com/pycharm/download/>)로 이동한 후 그림 8.1 과 같이 Community Edition 을 다운로드합니다. 참고로 다운로드 위치나 웹 페이지의 구성은 조금씩 바뀌므로 그림 8.1 과 다른 인터넷의 다른 글을 참고해서 Community Edition 을 다운로드하면 됩니다.

Download PyCharm

OS X

WINDOWS

LINUX

Professional

Full-featured IDE
for Python & Web
development

DOWNLOAD

221 MB

Community

Lightweight IDE
for Python & Scientific
development

DOWNLOAD

179 MB

그림 8.1 PyCharm 커뮤니티 에디션 다운로드 페이지

PyCharm 은 유료인 Professional Edition 과 무료인 Community Edition 이 있습니다. 물론 Professional Edition 이 더 많은 기능을 지원하지만 Community Edition 도 기본적인 프로그램을 개발하기에는 충분한 기능을 제공합니다.

1-2) PyCharm 설치

다운로드한 Community Edition 설치 파일을 실행하면 그림 8.2 와 같은 설치 화면이 나타납니다. Next 버튼을 클릭해 다음 화면으로 이동합니다.

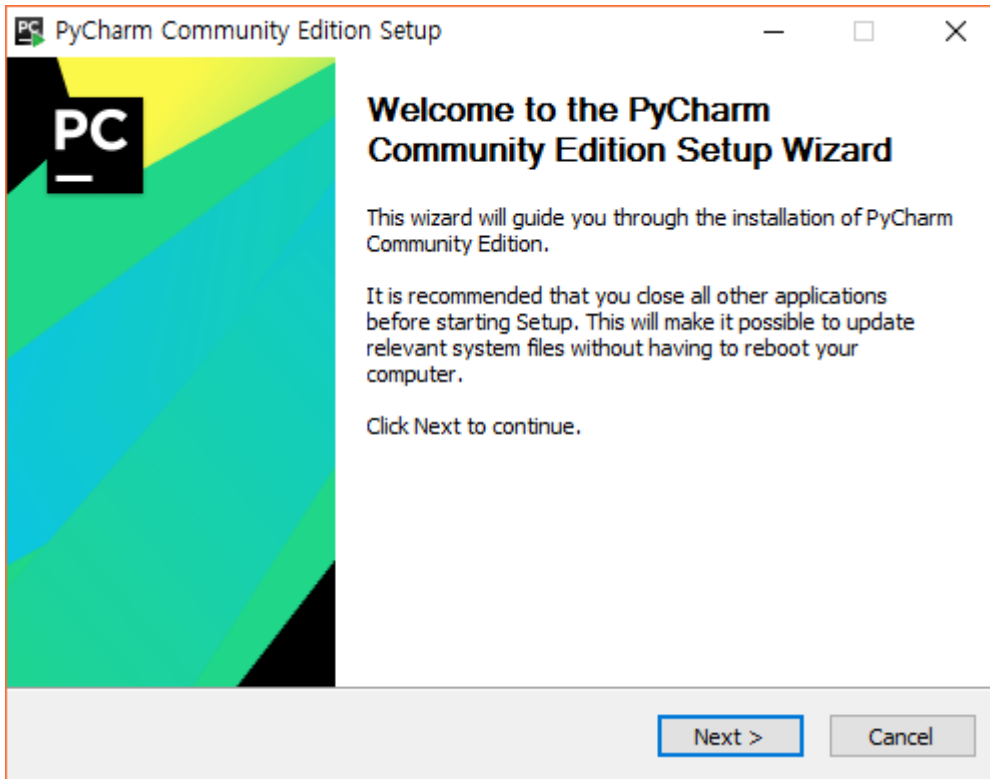


그림 8.2 PyCharm 설치 과정 1

그림 8.3 은 설치 위치를 변경할 수 있는 화면입니다. 기본 설정을 그대로 사용하는 것으로 하고 Next 버튼을 클릭해 다음 설치 화면으로 이동합니다.

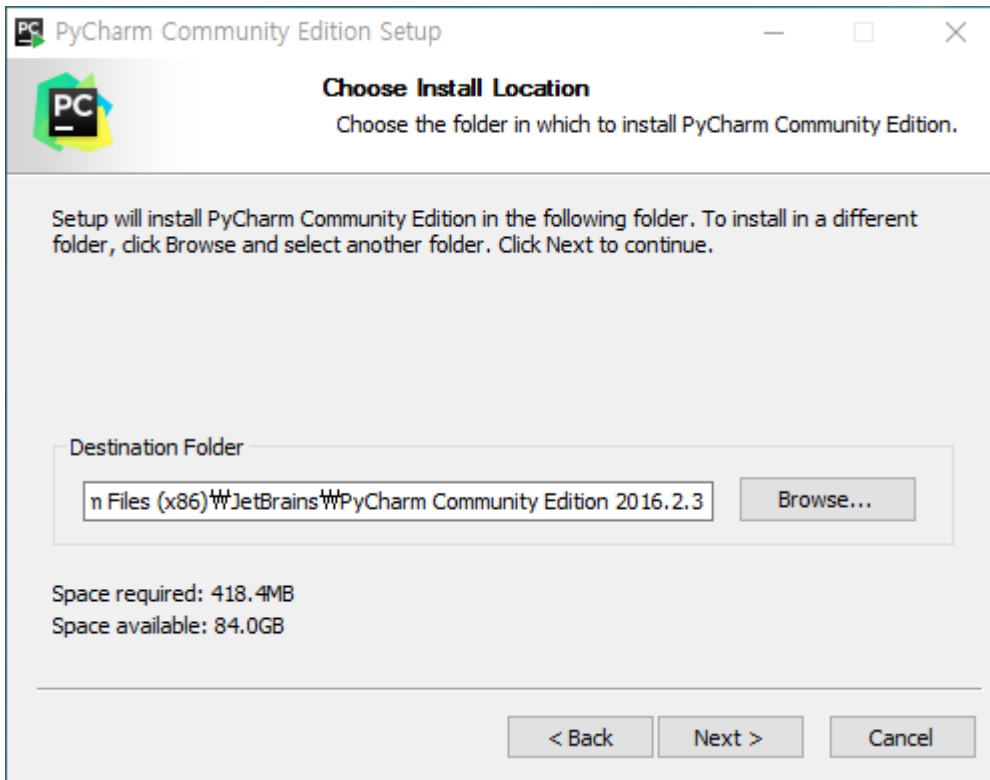


그림 8.3 PyCharm 설치 과정 2

그림 8.4 는 윈도우 바탕화면에 프로그램 아이콘과 *.py 파일에 대해 PyCharm 이 실행되도록 설정하는 화면입니다. 그림 8.4 와 같이 두 곳 모두 체크한 후 Next 버튼을 클릭합니다.

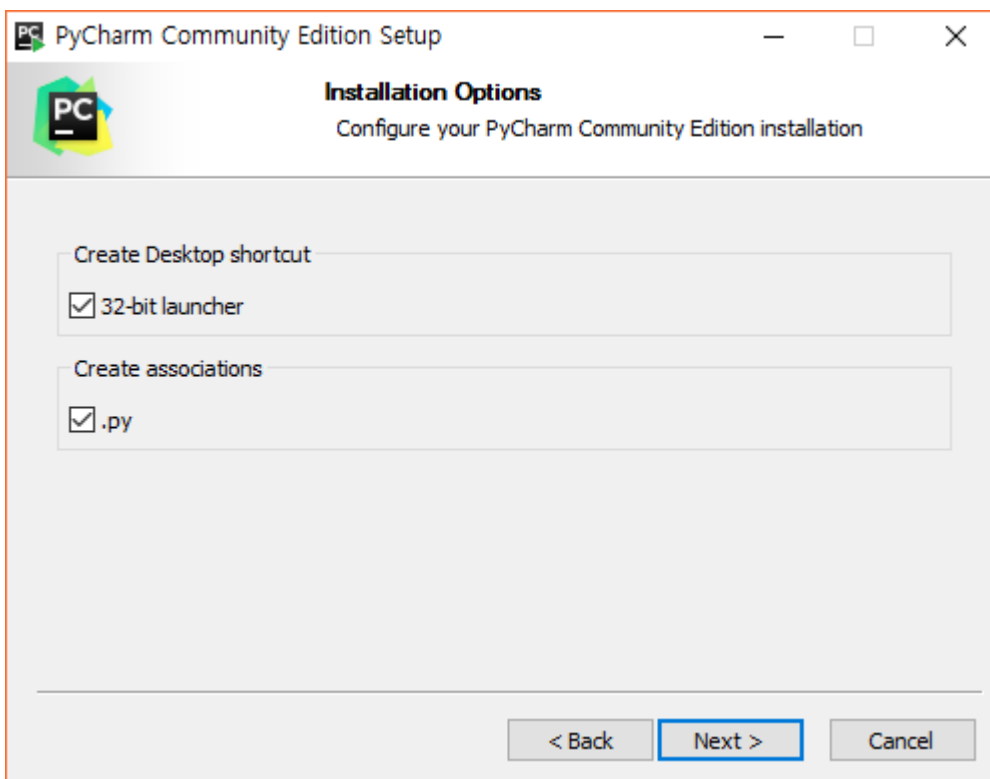


그림 8.4 PyCharm 설치 과정 3

그림 8.5 는 윈도우 시작 메뉴 폴더에 등록되는 이름을 변경할 수 있는 화면입니다. 기본값을 그대로 두고 Install 버튼을 클릭해 설치를 진행합니다.

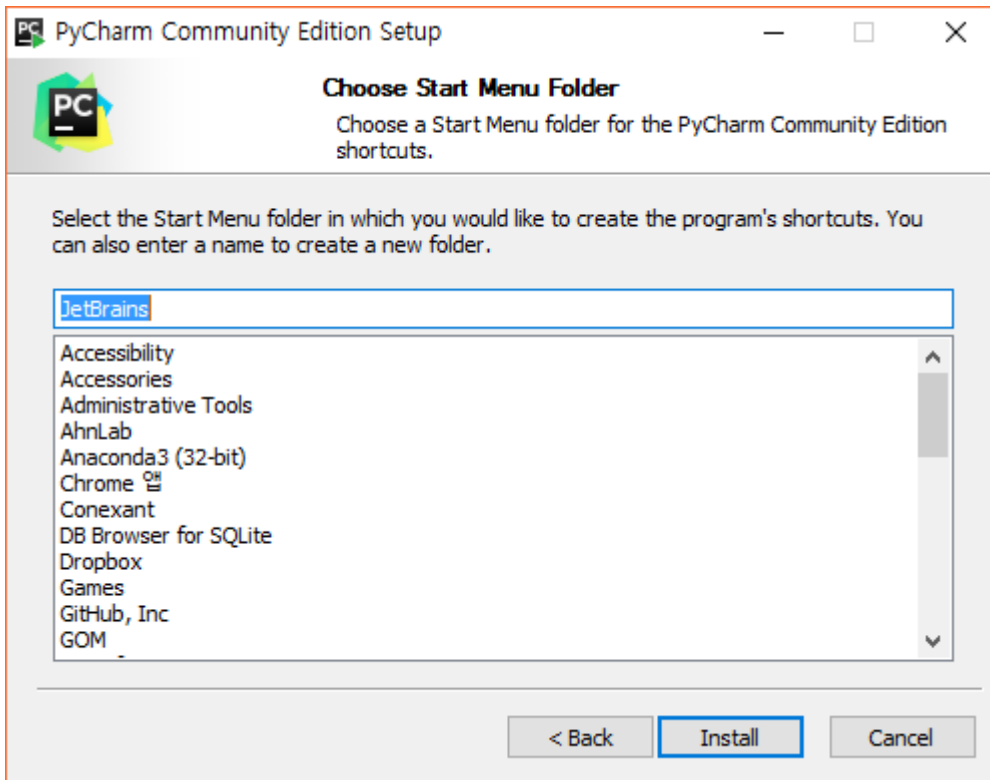


그림 8.5 PyCharm 설치 과정 4

그림 8.6 은 PyCharm 의 설치의 끝을 알리는 화면입니다. Finish 버튼을 클릭해 설치를 마칩니다.

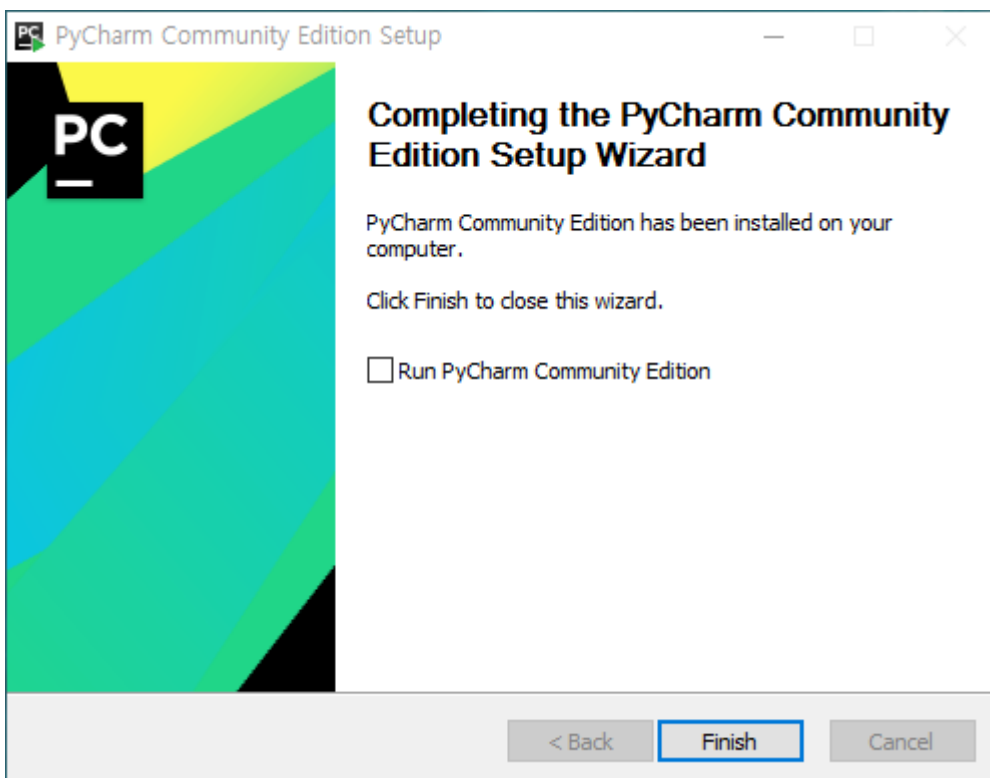


그림 8.6 PyCharm 설치 과정 5

1-3) PyCharm 초기 설정

PyCharm 설치가 완료되면 윈도우 바탕화면에 PyCharm 아이콘이 생성됩니다. 해당 아이콘을 클릭해 PyCharm 을 실행해 봅시다. PyCharm 을 설치한 후 처음 실행하면 그림 8.7 과 같은 화면이 나타납니다. 설정값을 그대로 둔 채 OK 버튼을 클릭합니다.

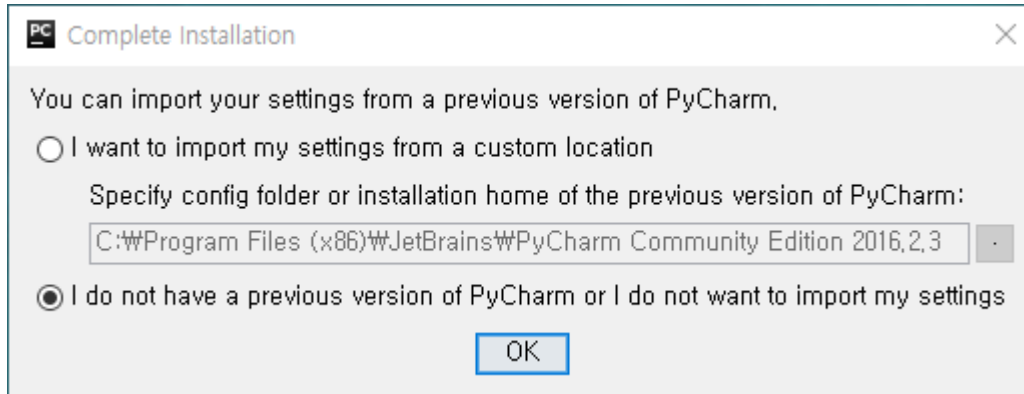


그림 8.7 PyCharm 초기 설정 과정 1

그림 8.8 은 PyCharm 키보드 단축키와 IDE 테마 등을 선택하는 화면입니다. 이 부분은 각자 취향에 맞춰 설정하는 부분인데 저자는 그림 8.8 과 같이 설정했습니다. 참고로 Darcula 테마는 약간 어두운 색상을 사용하므로 흰색 배경을 사용하려면 기본값을 사용하면 됩니다. 설정 중간에 있는 'Click to hide preview'를 누르면 그림 8.8 과 같이 현재 설정에 따른 결과를 미리 볼 수 있습니다.

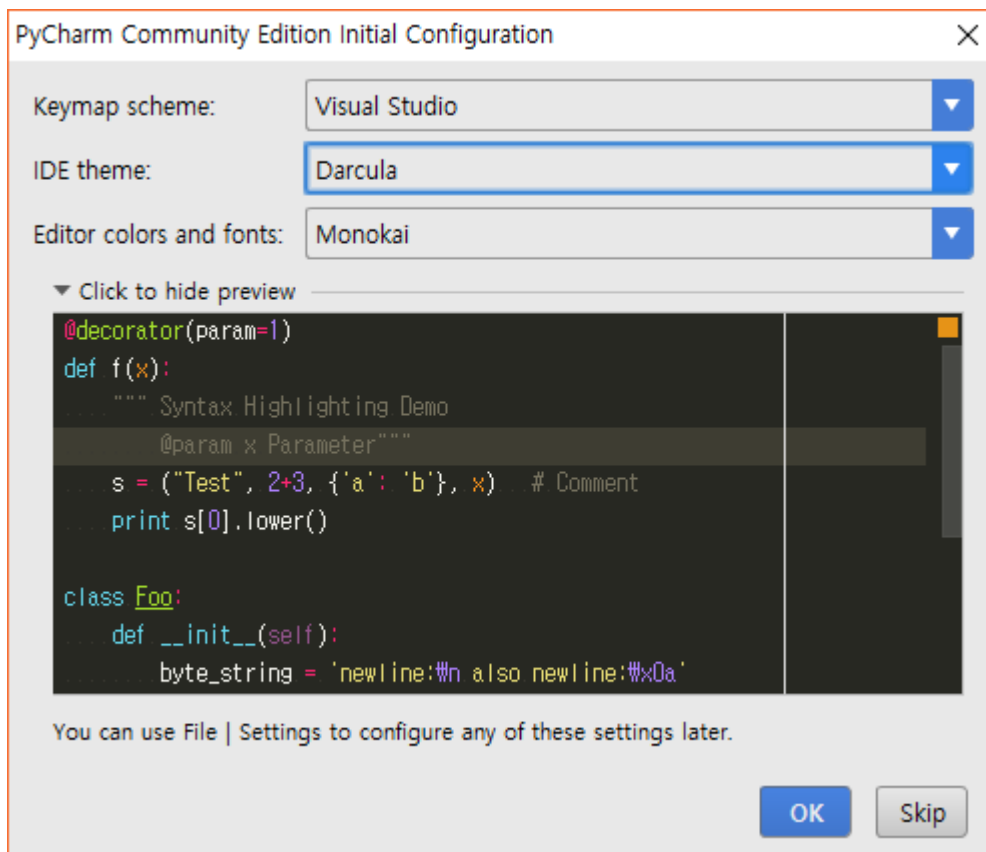


그림 8.8 PyCharm 초기 설정 과정 2

그림 8.8 에서 OK 버튼을 클릭하면 그림 8.9 와 같이 현재 설정을 적용해 PyCharm 을 재시작한다는 메시지가 나타납니다. Yes 를 눌러 PyCharm 을 재실행합니다.

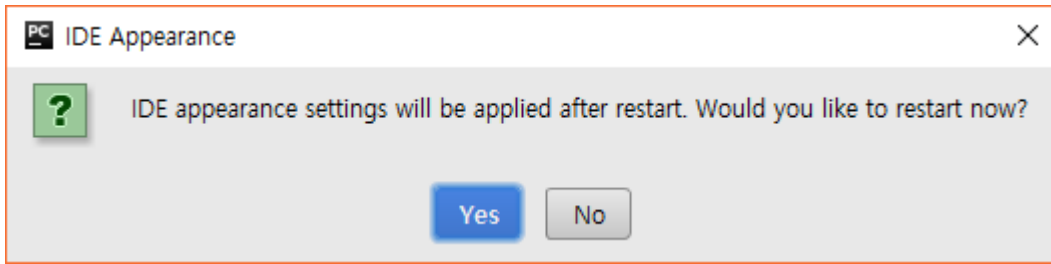


그림 8.9 PyCharm 초기 설정 과정 3

1-4) 프로젝트 생성

PyCharm 은 다른 IDE 와 마찬가지로 프로젝트 단위로 소스 파일을 관리합니다. 지금까지는 Python IDLE 를 통해 프로그래밍했기 때문에 따로 작성된 코드를 파일로 저장하지는 않았습니다. 그러나 앞으로 큰 규모의 프로그램을 개발하기 위해 작성된 코드를 여러분의 컴퓨터에 저장하겠습니다.

먼저 PyCharm 을 실행한 후 그림 8.10 과 같이 'Create New Project'를 클릭해 프로젝트 생성 화면으로 이동합니다.

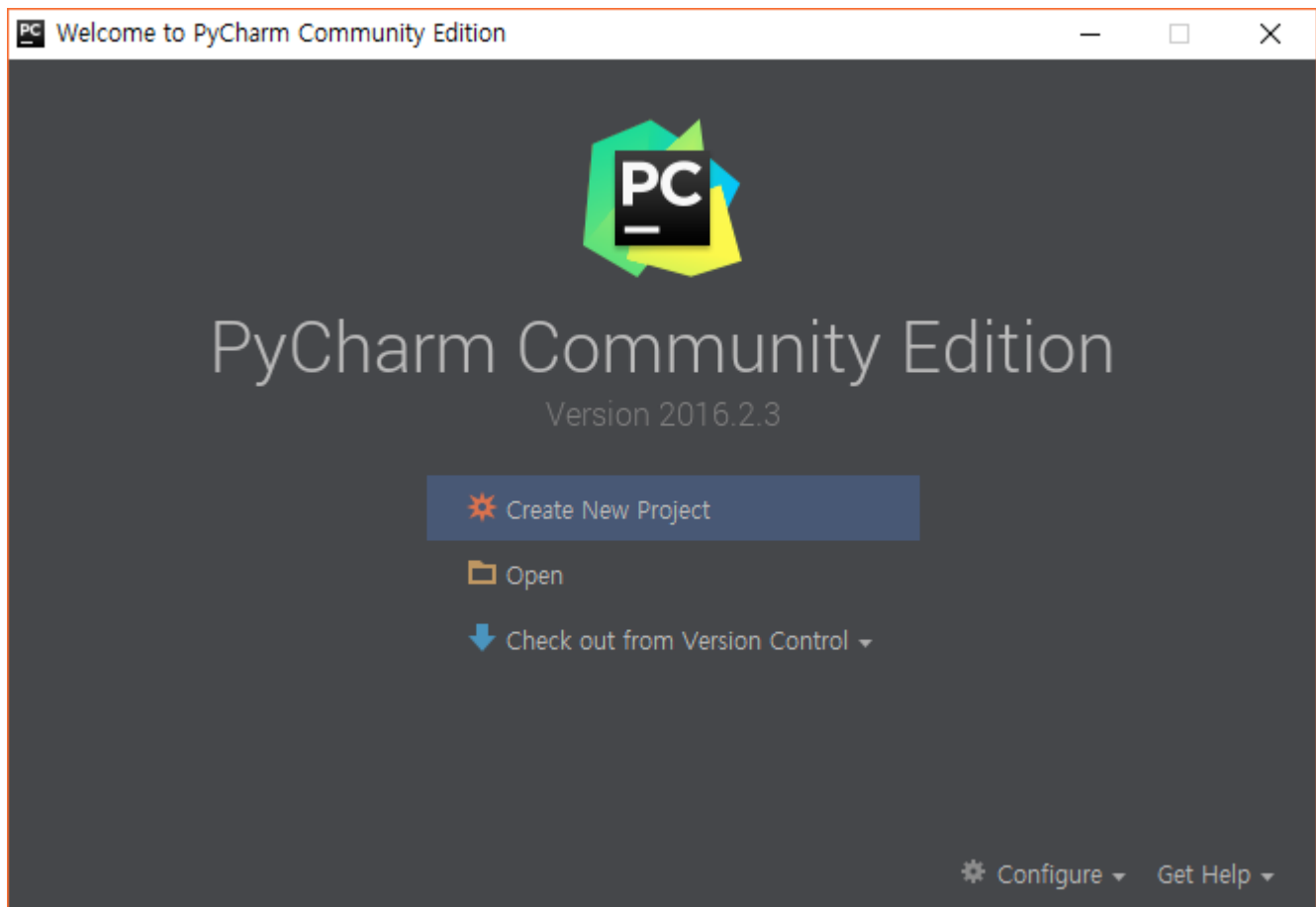


그림 8.10 PyCharm 프로젝트 생성 1

그림 8.11 은 프로젝트 저장 위치와 파이썬 인터프리터를 설정하는 화면입니다. 그림 8.11 과 같이 Location 의 마지막 부분을 'HelloWorld'라는 이름으로 변경한 후 Create 버튼을 클릭합니다.

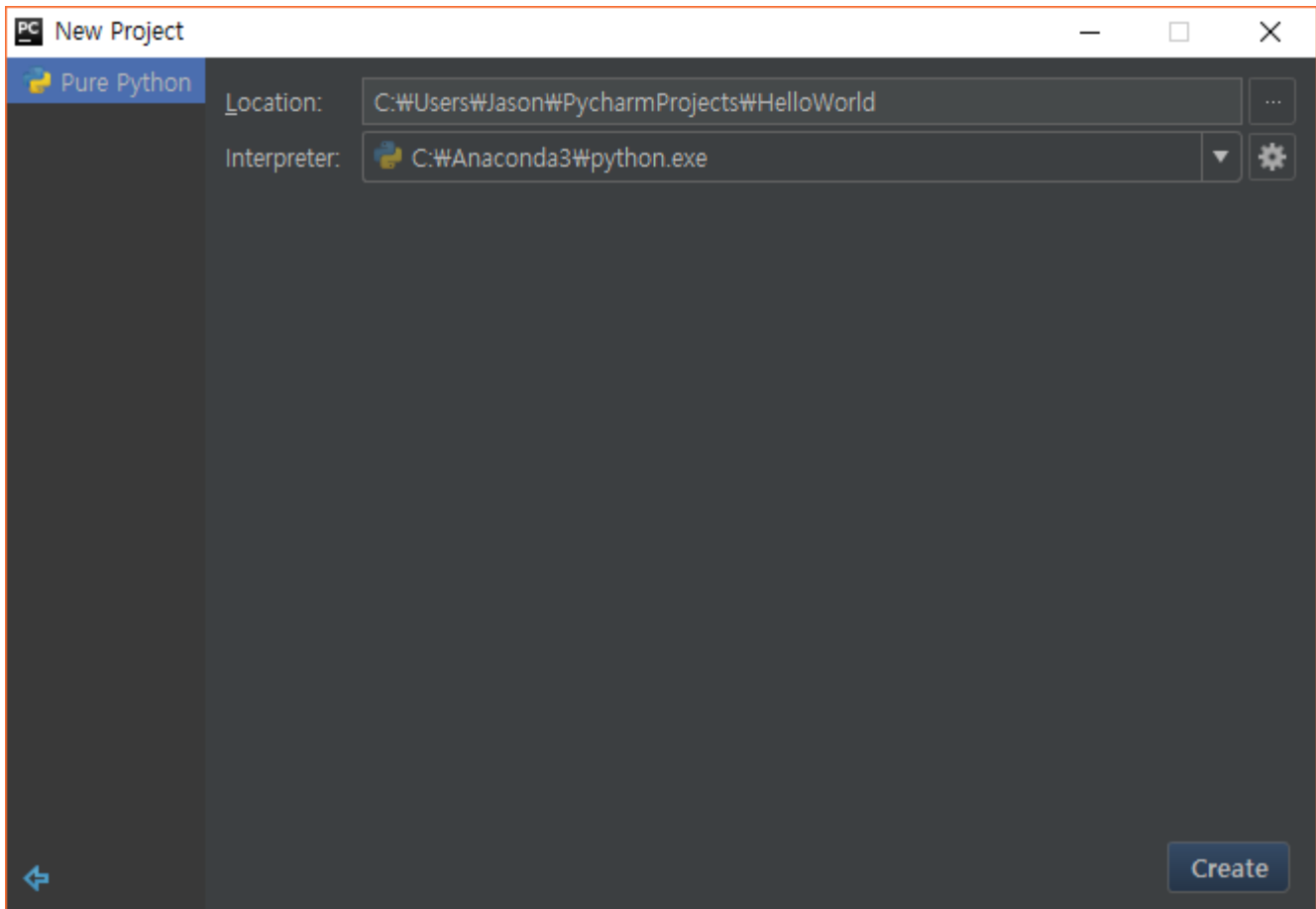


그림 8.11 PyCharm 프로젝트 생성 2

PyCharm 이 실행되면 그림 8.12 와 같이 화면에 Tip of the Day 라는 화면이 나타납니다. PyCharm 은 실행될 때마다 유용한 팁을 이와 같은 방식으로 알려줍니다. 일단 Close 버튼을 눌러 닫습니다.

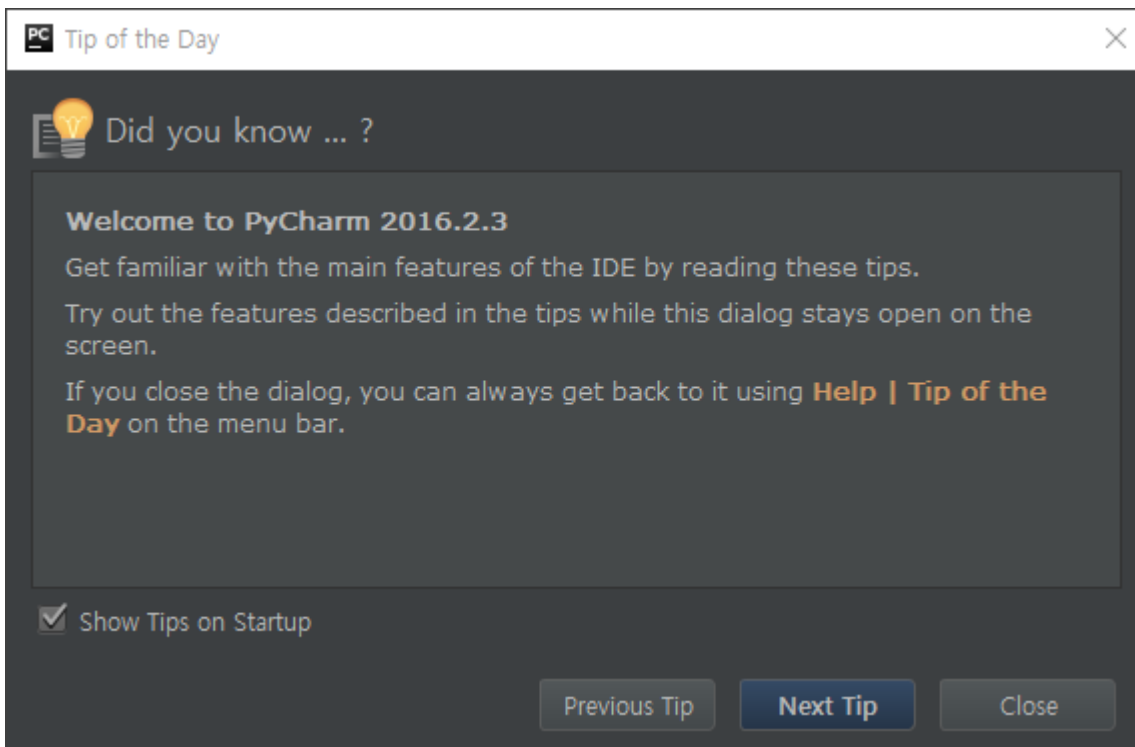


그림 8.12 PyCharm 프로젝트 생성 3

그림 8.13 은 실행된 PyCharm 화면 구성입니다. 프로젝트 부분에 HelloWorld 가 보입니다. 이는 그림 8.11 의 Location 부분에서 HelloWorld 를 사용했기 때문입니다.

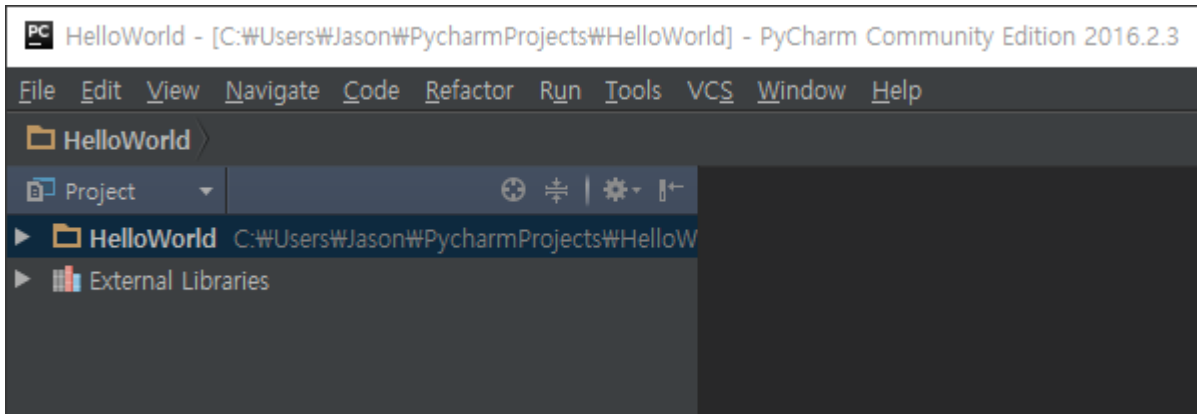


그림 8.13 PyCharm 프로젝트 생성 4

생성된 HelloWorld 라는 프로젝트는 현재 비어있는 상태입니다. 프로젝트에 파이썬 파일을 추가하려면 그림 8.14 와 같이 HelloWorld 라는 프로젝트명에 마우스 오른쪽 버튼을 클릭한 후 New 와 Python File 메뉴를 순서대로 클릭합니다.

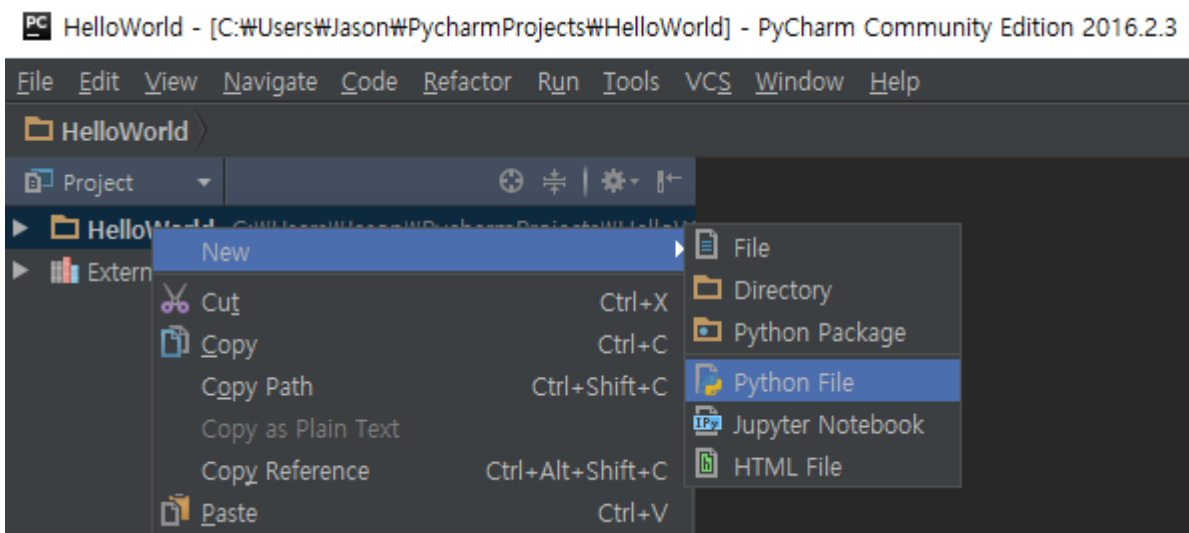


그림 8.14 PyCharm 파일 추가 1

그림 8.15 와 같이 새 파일을 생성하는 창에서 Name 에 파일명을 입력합니다. OK 버튼을 클릭해 hello.py 라는 파일을 생성해 봅시다.

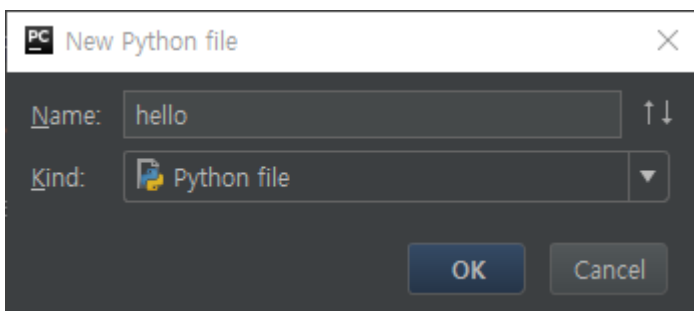


그림 8.15 PyCharm 파일 추가 2

hello.py 파일에 그림 8.16 과 같이 `print("hello world")`를 입력해 봅시다. 지금까지는 Python IDLE 를 통해 명령어를 입력했지만, 앞으로는 그림 8.16 과 같이 파이썬 코드 파일에 입력하면 됩니다.

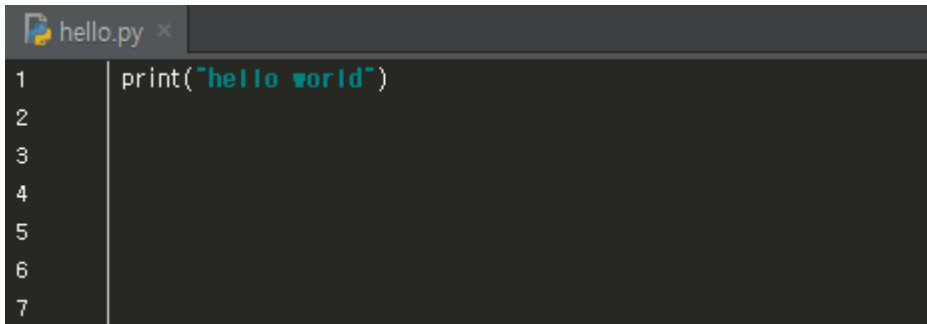


그림 8.16 파이썬 코드 작성

Python IDLE 는 인터프리터라서 명령어를 입력할 때마다 해당 명령어를 바로 실행했습니다. 이와 달리 PyCharm 은 코드를 작성한 후 실행 명령을 내리는 시점에 프로그램을 실행합니다. PyCharm 에서 소스코드를 실행하려면 그림 8.17 과 같이 소스코드에 마우스 오른쪽 버튼을 눌러 Run 메뉴를 클릭하거나 단축키인 'Ctrl+F9'를 누르면 됩니다.

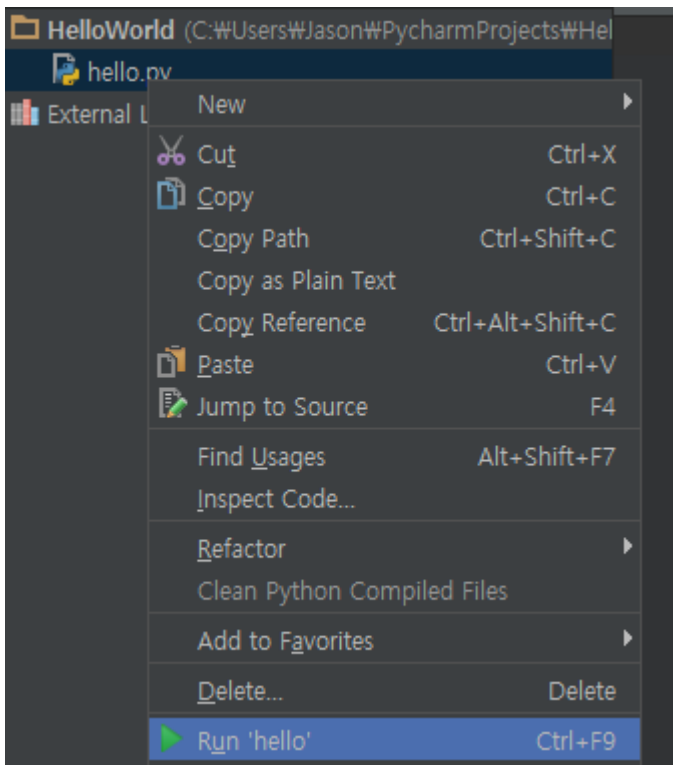


그림 8.17 파이썬 코드 실행 1

Ctrl + F9 단축키 또는 Run 메뉴를 선택해 파이썬 코드를 실행하면 그림 8.18 과 같이 PyCharm 하단부의 실행 창에 'hello world'라는 문자열이 출력됩니다. 참고로 단축키는 그림 8.8 의 설정에 따라 달라지므로 저자와 다를 수도 있습니다.

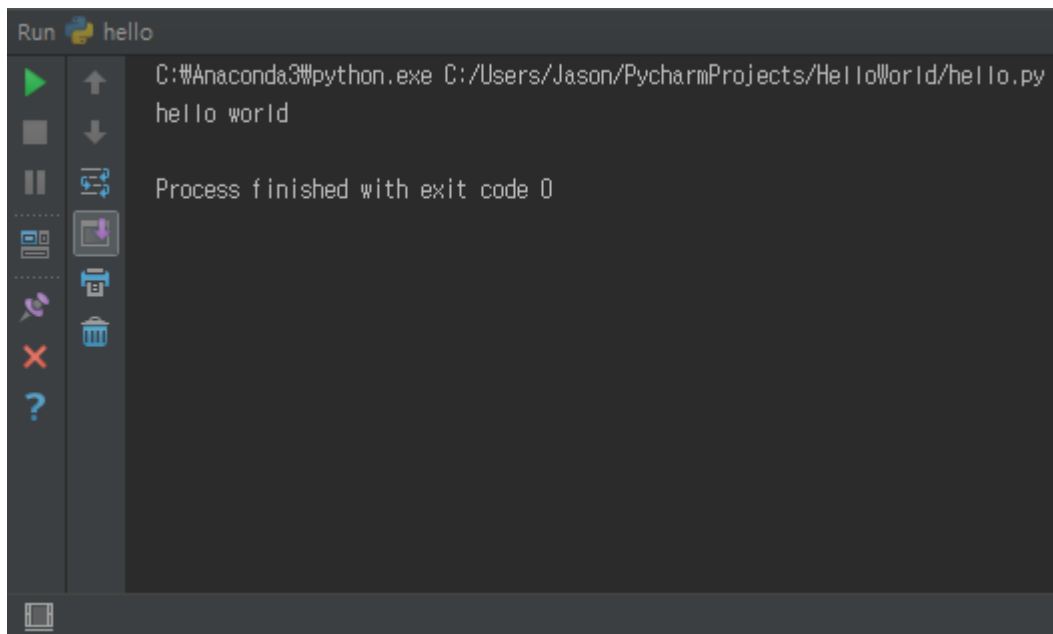


그림 8.18 파이썬 코드 실행 2

1-5) PyCharm 기타 설정

이번 절에서는 PyCharm 에서 프로그램을 작성할 때 유용하게 사용할 수 있는 몇 가지 설정을 소개하겠습니다.

첫 번째로 소개할 내용은 파이썬 에디터 창에서 소스코드의 라인 넘버를 보이게 하는 설정입니다. 가끔 다른 누군가에게 코드를 설명할 때 '몇 번째 줄에 있는 코드를 보라' 같은 표현을 하곤 합니다. PyCharm 의 기본 설정은 에디터에 라인 넘버가 보이지 않습니다. PyCharm 의 코드 에디터에서 라인 넘버를 보이게 하려면 File → Settings → Editor → General → Appearance 메뉴를 차례로 선택한 후 그림 8.19 와 같이 'Show line numbers' 항목을 체크하면 됩니다.

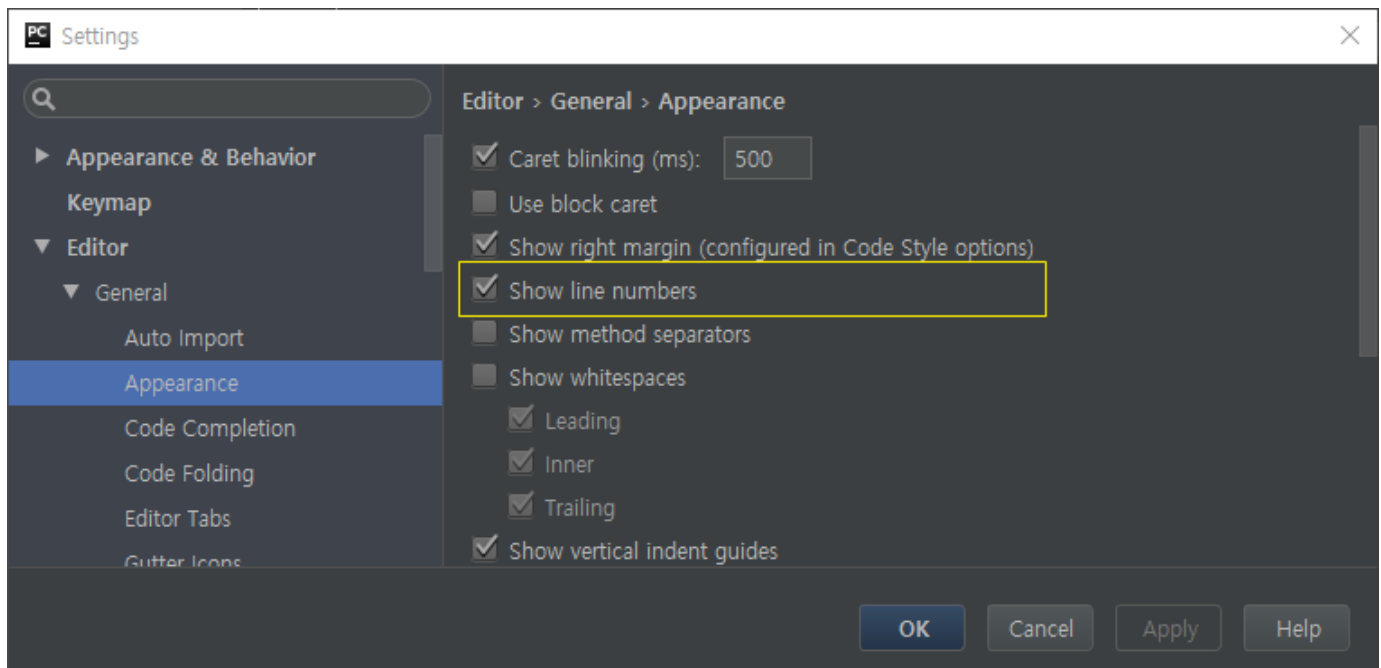


그림 8.19 라인 넘버 설정

두 번째로 소개할 내용은 PyCharm 에서 파이썬 인터프리터를 사용하는 기능입니다. PyCharm 으로 프로그램을 작성하다 보면 간단한 코드를 파이썬 인터프리터로 확인해보고 싶을 때가 있습니다. Tools → Python Console 메뉴를 차례로 선택하면 그림 8.20 과 같이 PyCharm 의 하단부에 Python IDLE 와 같은 Python Console 이 나타납니다.

이처럼 Python IDLE 과 비슷한 기능을 수행하는 Python Console 을 동시에 사용함으로써 더욱 편리하게 개발할 수 있습니다.

```
Python Console
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
PyDev console: using IPython 5.1.0
?
import sys: print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\Jason\\PycharmProjects\\HelloWorld'])
Python 3.5.2 |Anaconda 4.2.0 (32-bit)| (default, Jul  5 2016, 11:45:57) [MSC v.1900 32 bit (Intel)] on win32
+
In[2]:
```

그림 8.20 Python Console 실행 화면

2) 주소록 프로젝트

아이폰이나 안드로이드폰을 보면 이름, 전화번호, 이메일 등을 저장하고 관리할 수 있는 주소록이 기본으로 제공됩니다. 이번 절에서는 파이썬을 이용하여 주소록을 관리하는 프로그램을 직접 개발해 보겠습니다.

PyCharm 을 실행한 후 그림 8.21 과 같이 Contact 라는 이름의 프로젝트를 생성합니다.

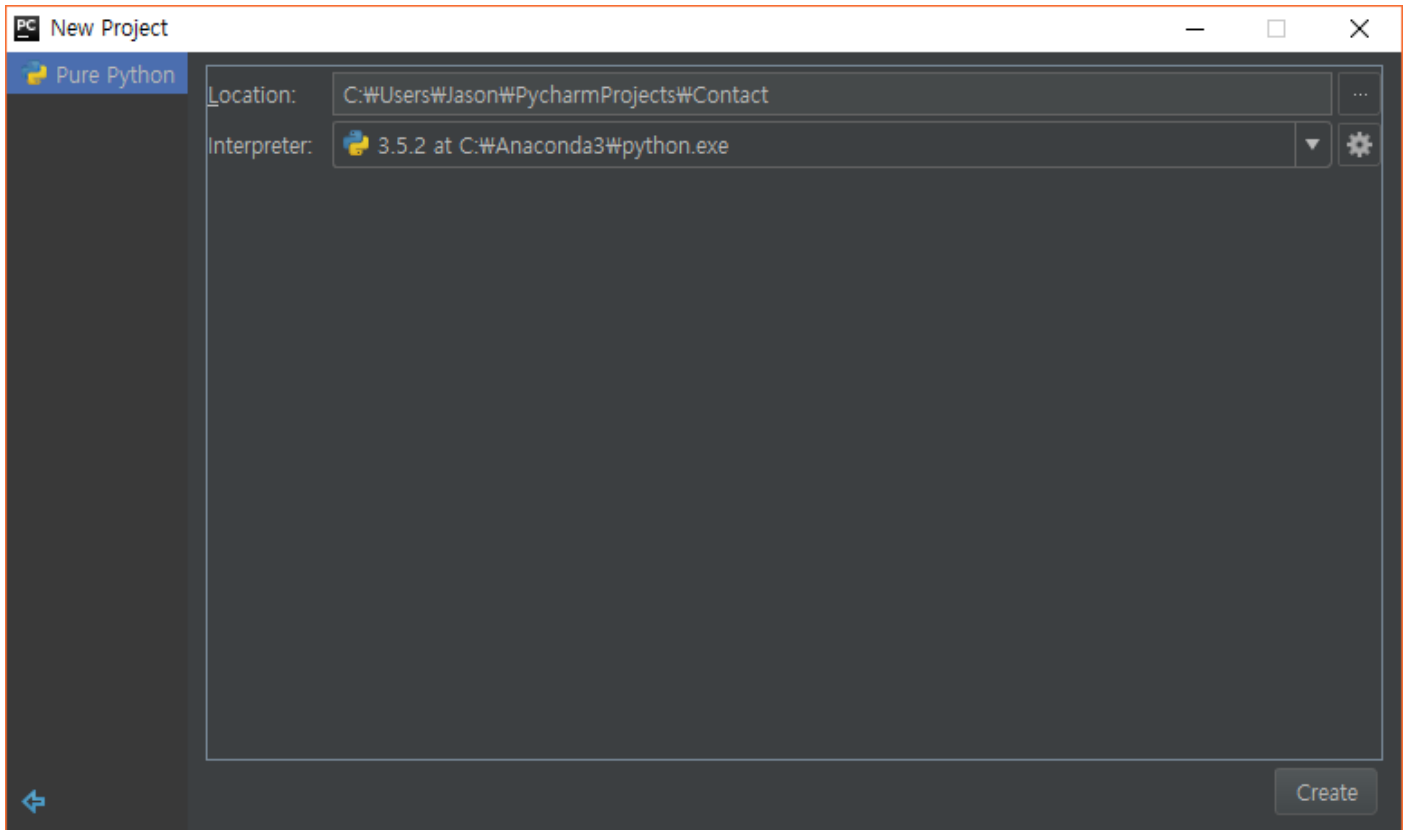


그림 8.21 Contact 프로젝트 생성

Contact 프로젝트에 contact.py 파일을 추가한 후 다음과 같이 기본 코드를 작성합니다. 그리고 Ctrl + F9 키를 눌러 화면에 'Contact'라는 문자열이 정상적으로 출력되는지 확인합니다.

```
def run():
```

```
    print("Contact")
```

```
if __name__ == "__main__":
```

```
    run()
```


2-1) Contact 클래스 만들기

아이폰이나 안드로이드폰의 연락처 프로그램을 살펴보면 연락처에는 이름, 전화번호, 이메일, 주소 등을 입력할 수 있습니다. 이를 파이썬에서 제공하는 기본 자료형으로 표현하기보다 하나의 클래스로 정의하는 것이 좋습니다.

6장에서 배운 클래스를 이용해 다음과 같이 'Contact'라는 이름의 클래스를 정의하겠습니다. 클래스 인스턴스를 생성할 때 이름, 전화번호, 이메일, 주소를 입력받을 수 있게 생성자를 선언하고 인스턴스 변수에 저장된 정보를 화면에 출력하기 위해 print_info 라는 메서드도 정의했습니다.

```
class Contact:
```

```
    def __init__(self, name, phone_number, e_mail, addr):
```

```
        self.name = name
```

```
        self.phone_number = phone_number
```

```
        self.e_mail = e_mail
```

```
        self.addr = addr
```

```
    def print_info(self):
```

```
        print("Name: ", self.name)
```

```
        print("Phone Number: ", self.phone_number)
```

```
        print("E-mail: ", self.e_mail)
```

```
        print("Address: ", self.addr)
```

정의한 Contact 클래스가 제대로 동작하는지 확인하기 위해 앞서 작성한 run() 함수를 수정해 연락처를 입력하고 입력된 정보를 화면에 출력해 보겠습니다. 이를 위해 다음과 같이 Contact 클래스에 대한 인스턴스를 생성합니다.

```
def run():
```

```
    kim = Contact('김일구', '010-8812-1193', 'ilgu.kim@python.com', 'Seoul')
```

```
    kim.print_info()
```

지금까지 작성한 전체 코드는 다음과 같습니다.

```
class Contact:
```

```
    def __init__(self, name, phone_number, e_mail, addr):
```

```
        self.name = name
```

```
        self.phone_number = phone_number
```

```
        self.e_mail = e_mail
```

```
        self.addr = addr
```

```
    def print_info(self):
```

```
        print("Name: ", self.name)
```

```
        print("Phone Number: ", self.phone_number)
```

```
        print("E-mail: ", self.e_mail)
```

```
        print("Address: ", self.addr)
```

```
def run():
```

```
    kim = Contact('김일구', '010-8812-1193', 'ilgu.kim@python.com', 'Seoul')
```

```
    kim.print_info()
```

```
if __name__ == "__main__":
```

```
    run()
```

PyCharm 에서 Ctrl + F9 또는 Ctrl + F5 를 눌러 지금까지 작성한 프로그램이 정상적으로 동작하는지 확인해보기 바랍니다. 그림 8.22 와 같이 run() 함수에서 입력한 값이 정상적으로 출력되면 됩니다.

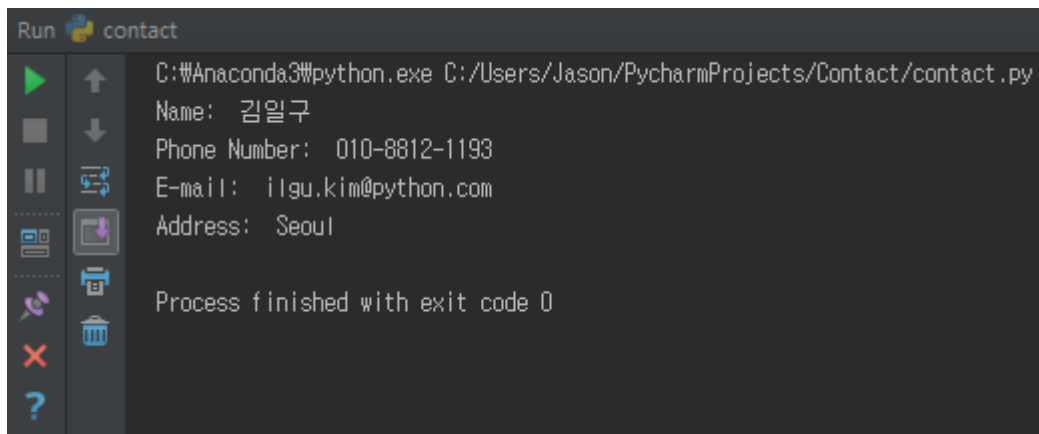


그림 8.22 연락처 정보 출력

2-2) 사용자로부터 데이터 입력받기

이번 절에서는 사용자로부터 데이터를 입력받아 보겠습니다. 파이썬에서 사용자로부터 데이터를 입력받을 때는 `input()` 함수를 사용합니다. 사용자로부터 데이터를 입력받는 함수인 `set_contact()` 함수를 새로 정의하고 `run()` 함수에서 `set_contact()` 함수를 호출하도록 코드를 변경해 봅시다.

```
class Contact:
```

```
    def __init__(self, name, phone_number, e_mail, addr):
```

```
        self.name = name
```

```
        self.phone_number = phone_number
```

```
        self.e_mail = e_mail
```

```
        self.addr = addr
```

```
    def print_info(self):
```

```
        print("Name: ", self.name)
```

```
        print("Phone Number: ", self.phone_number)
```

```
        print("E-mail: ", self.e_mail)
```

```
        print("Address: ", self.addr)
```

```
def set_contact():
```

```
    name = input("Name: ")
```

```
    phone_number = input("Phone Number: ")
```

```
    e_mail = input("E-mail: ")
```

```
    addr = input("Address: ")
```

```
    print(name, phone_number, e_mail, addr)
```

```
def run():
    set_contact()
```

```
if __name__ == "__main__":
    run()
```

수정한 코드를 실행하면 PyCharm의 아래쪽에 위치하는 실행 창을 통해 데이터를 입력할 수 있습니다. 데이터 입력이 완료되면 set_contact() 함수의 마지막 라인에 있는 print 코드로 인해 지금까지 입력된 데이터가 화면에 출력됩니다. 그림 8.23과 같이 입력한 데이터와 출력된 데이터를 비교해 정상적으로 데이터가 변수에 저장되는지 확인해 봅시다.

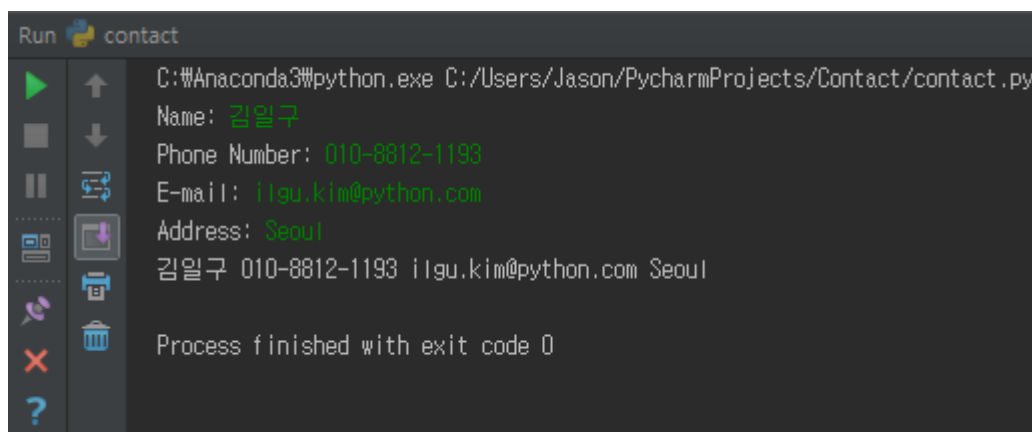


그림 8.23 데이터 입력하기

2-3) 메인 메뉴 구성하기

여러분이 작성할 연락처 프로그램은 기본적으로 연락처 입력, 연락처 출력, 연락처 삭제, 종료 기능을 제공합니다. 작성된 연락처 프로그램은 계속 실행 상태를 유지하다가 사용자로부터 종료 명령을 받으면 프로그램이 종료해야 합니다. 이번 절에서는 이러한 기본 실행 구조를 파이썬 코드로 작성해 보겠습니다.

작성한 프로그램이 한 번 실행됐을 때 종료하지 않은 상태로 계속 실행되게 하려면 무한 루프를 사용해야 합니다. 파이썬에서는 'while 1'이라는 구문을 사용하면 프로그램이 계속 실행되도록 만들 수 있습니다. 하지만 사용자로부터 종료 명령을 받았을 때는 프로그램이 종료해야겠지요? 이를 위해서는 while 문 내부에서 if 문을 사용해 조건에 따라 while 문이 종료하도록 만들면 됩니다.

여기서 while 문은 결국 반복문이기 때문에 이를 종료하려면 반복문을 빠져나오면 됩니다. 이러한 용도로 사용하는 구문이 바로 break 문입니다. 지금까지 설명한 내용을 코드로 작성하면 다음과 같습니다. 참고로 지금부터는 지면상 새로 추가된 부분이나 수정된 코드만 표시하겠습니다.

```
def print_menu():
```

```
    print("1. 연락처 입력")
```

```
    print("2. 연락처 출력")
```

```
    print("3. 연락처 삭제")
```

```
    print("4. 종료")
```

```
    menu = input("메뉴선택: ")
```

```
    return int(menu)
```

```
def run():
```

```
    while 1:
```

```
        menu = print_menu()
```

```
        if menu == 4:
```

```
            break
```

지금까지 작성한 프로그램을 실행해 봅시다. 그림 8.24 는 현재까지 작성된 프로그램의 실행 결과 화면입니다. run() 함수는 print_menu() 함수를 호출하기 때문에 화면에 메인 메뉴가 출력됩니다.

print_menu() 함수는 input() 함수를 통해 사용자로 부터 메뉴를 입력 받은 후 int()라는 내장 함수를 통해 정수로 변환한 후 반환합니다.

run() 함수에서는 사용자가 입력한 메뉴 값이 4와 같은지 확인하고 메뉴 값이 4 일 때는 break 을 사용해 무한 루프를 빠져나옵니다. 그림 8.24 와 같이 여러분이 1 번 메뉴를 선택했다면 while 문의 반복 구조로 인해 다시 한 번 화면에 메뉴가 출력됩니다. 이때 4 를 입력하면 정상적으로 프로그램이 종료됨을 확인할 수 있습니다.

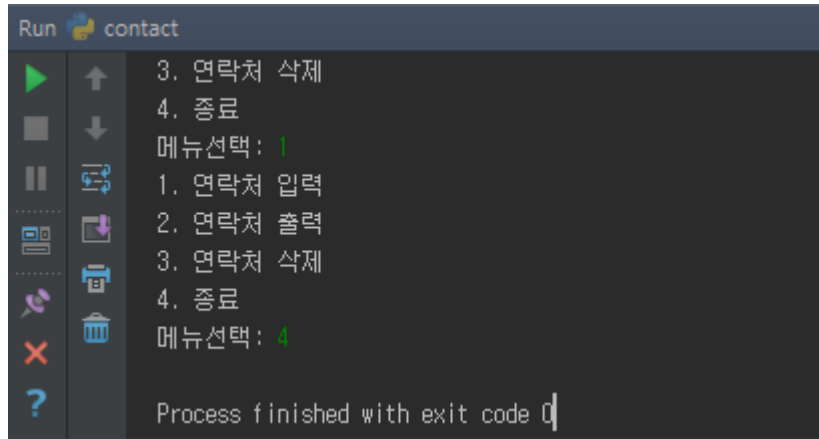


그림 8.24 프로그램의 메인 루프 테스트

2-4) 연락처 입력 동작 시키기

이번에는 1 번 메뉴인 '연락처 입력' 기능이 정상적으로 동작하도록 프로그램을 변경해 보겠습니다. 먼저 사용자가 입력한 연락처 정보를 이용해 Contact 클래스 인스턴스를 생성하고 생성된 인스턴스를 반환하도록 set_contact() 함수를 수정하겠습니다. set_contact() 함수의 마지막 두 줄에서 Contact 클래스의 인스턴스를 생성하고 이를 반환하는 것을 확인할 수 있습니다.

```
def set_contact():  
    name = input("Name: ")  
    phone_number = input("Phone Number: ")  
    e_mail = input("E-mail: ")  
    addr = input("Address: ")  
    contact = Contact(name, phone_number, e_mail, addr)  
    return contact
```

현재 set_contact() 함수는 단순히 함수가 정의된 상태입니다. 이 함수가 정상적으로 동작하려면 어디선가 해당 함수를 호출해야 합니다. set_contact() 함수는 사용자로부터 연락처를 입력받을 때 사용하는 함수이므로 사용자가 '1. 연락처 입력' 메뉴를 선택했을 때 run() 함수에서 set_contact() 함수를 호출해주면 되겠습니다.

다음과 같이 기존에 작성된 run() 함수를 수정해 사용자가 1 번 메뉴를 선택했을 때 set_contact() 함수를 호출하도록 변경해 봅시다. 그리고 set_contact() 함수의 반환값인 Contact 인스턴스를 저장하기 위해 contact_list 라는 이름의 리스트 자료구조를 생성하고 해당 리스트에 생성된 인스턴스를 추가해 봅시다.

```
def run():  
    contact_list = []  
    while 1:  
        menu = print_menu()  
        if menu == 1:  
            contact = set_contact()
```



```
contact_list.append(contact)
```

```
elif menu == 4:
```

```
break
```

예제 8.1 은 지금까지 작성된 코드입니다. 중간중간 코드를 수정하다가 흐름을 놓친 분들은 다음의 전체 코드를 살펴보시기 바랍니다.

```
class Contact:
```

```
    def __init__(self, name, phone_number, e_mail, addr):
```

```
        self.name = name
```

```
        self.phone_number = phone_number
```

```
        self.e_mail = e_mail
```

```
        self.addr = addr
```

```
    def print_info(self):
```

```
        print("Name: ", self.name)
```

```
        print("Phone Number: ", self.phone_number)
```

```
        print("E-mail: ", self.e_mail)
```

```
        print("Address: ", self.addr)
```

```
def set_contact():
```

```
    name = input("Name: ")
```

```
    phone_number = input("Phone Number: ")
```

```
    e_mail = input("E-mail: ")
```

```
    addr = input("Address: ")
```

```
contact = Contact(name, phone_number, e_mail, addr)
```

```
return contact
```

```
def print_menu():
```

```
    print("1. 연락처 입력")
```

```
    print("2. 연락처 출력")
```

```
    print("3. 연락처 삭제")
```

```
    print("4. 종료")
```

```
    menu = input("메뉴선택: ")
```

```
    return int(menu)
```

```
def run():
```

```
    contact_list = []
```

```
    while 1:
```

```
        menu = print_menu()
```

```
        if menu == 1:
```

```
            contact = set_contact()
```

```
            contact_list.append(contact)
```

```
        elif menu == 4:
```

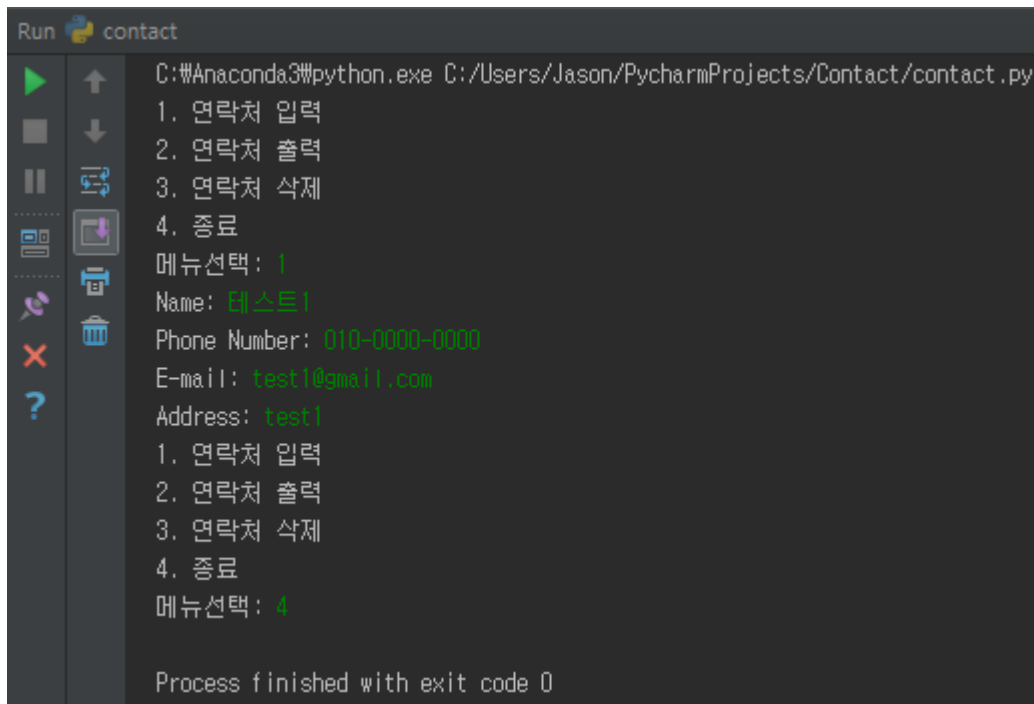
```
            break
```

```
if __name__ == "__main__":
```

```
    run()
```

예제 8.1 Contact 관리 프로그램 소스코드

예제 8.1의 코드를 실행한 후 그림 8.25와 같이 1번 메뉴를 선택해 연락처를 입력하고 4번 메뉴를 선택해 프로그램을 종료시켜 봅시다. 아직은 연락처 출력 기능이 없으므로 정상적으로 입력됐는지는 확인할 수 없지만 현 시점에서는 일단 에러가 발생하지 않는 것이 중요합니다.



```
Run contact
C:\Anaconda3\python.exe C:/Users/Jason/PycharmProjects/Contact/contact.py
1. 연락처 입력
2. 연락처 출력
3. 연락처 삭제
4. 종료
메뉴선택: 1
Name: 테스트1
Phone Number: 010-0000-0000
E-mail: test1@gmail.com
Address: test1
1. 연락처 입력
2. 연락처 출력
3. 연락처 삭제
4. 종료
메뉴선택: 4
Process finished with exit code 0
```

그림 8.25 Contact 프로그램의 중간 실행 결과

앞서 프로그램이 실행될 때 `contact_list`는 그림 8.26과 같이 입력된 '테스트 1'에 대한 데이터를 가진 인스턴스를 바인딩하게 됩니다. 물론 4번 메뉴를 통해 프로그램을 종료시키면 메모리에 할당된 모든 데이터가 자동으로 삭제됩니다.

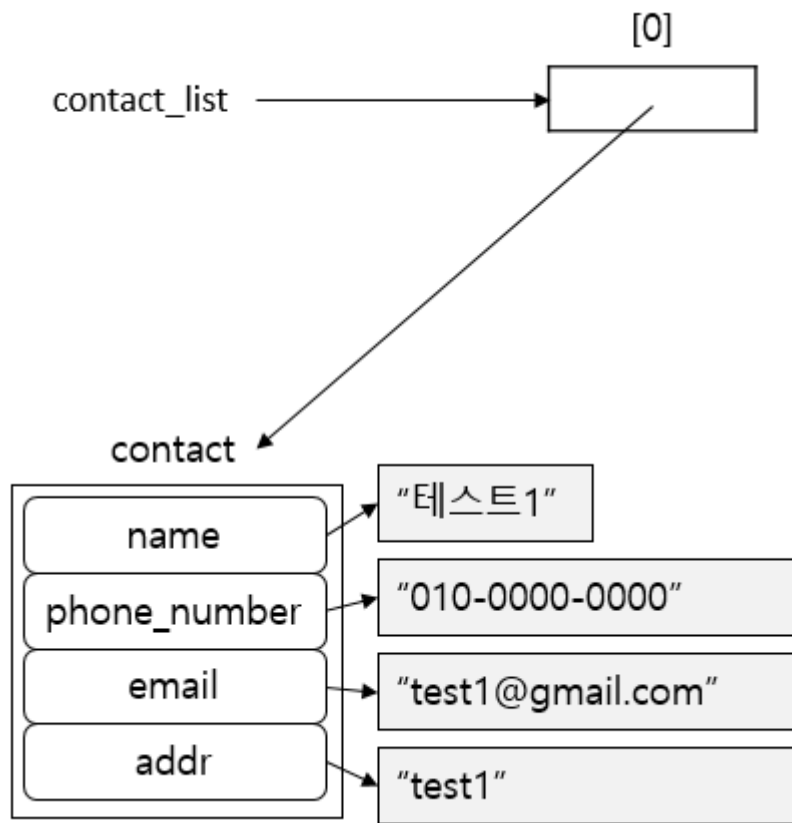


그림 8.26 `contact_list` 상태

2-5) 연락처 출력하기

이번에는 연락처를 출력하는 기능을 추가해 보겠습니다. 앞서 작성한 코드에서 연락처는 Contact 라는 클래스의 인스턴스 형태로 저장되며, 인스턴스는 contact_list 라는 이름의 리스트를 통해 순서대로 저장됩니다. 즉, 입력된 연락처를 모두 출력하려면 contact_list 에 있는 인스턴스에 저장된 정보를 출력하면 되는 것입니다.

이미 여러분은 Contact 클래스를 정의할 때 인스턴스 변수의 데이터를 화면에 출력하는 기능을 제공하는 메서드인 print_info()를 정의해 놓았습니다. 따라서 연락처를 출력하는 함수는 다음과 같이 간단히 구현할 수 있습니다.

```
def print_contact(contact_list):  
  
    for contact in contact_list:  
  
        contact.print_info()
```

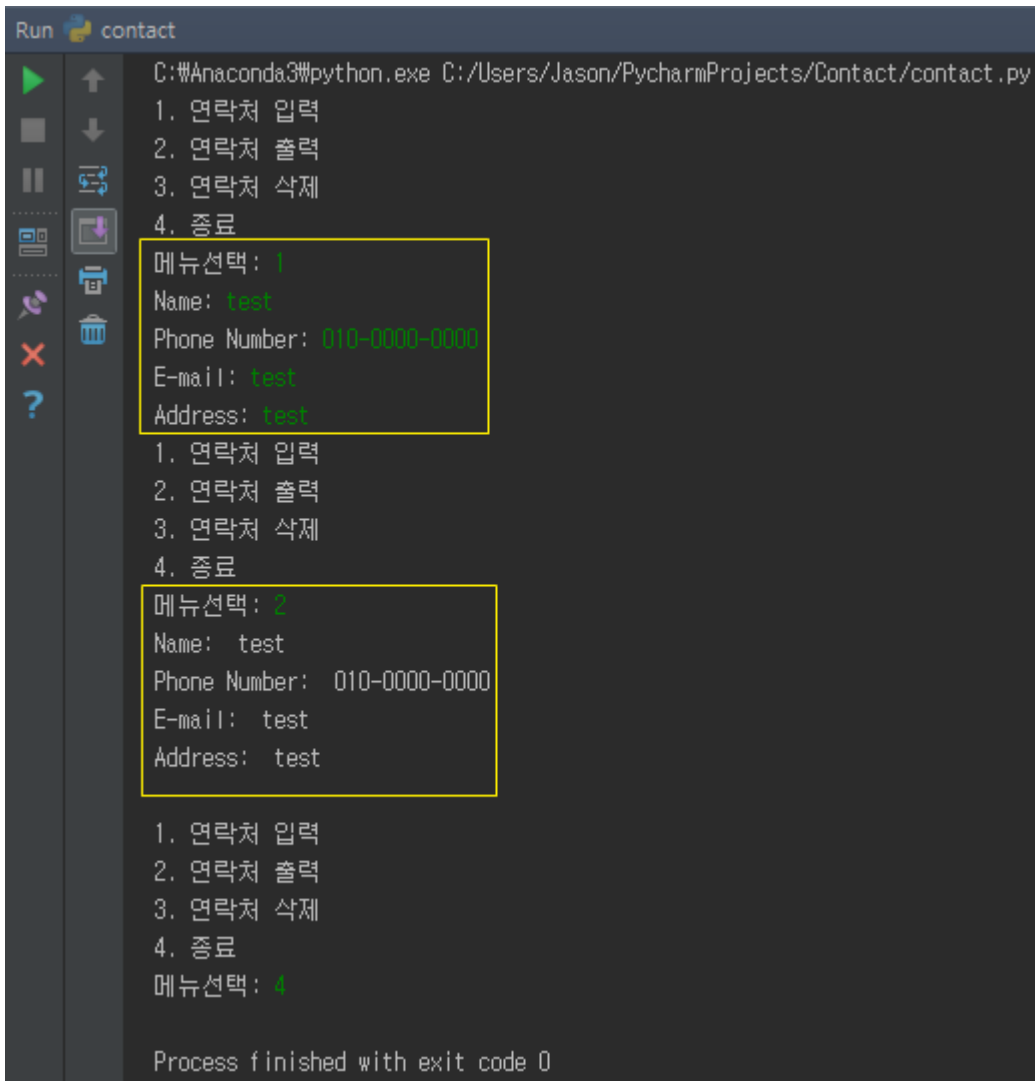
print_contact() 함수는 Contact 인스턴스를 저장하고 있는 리스트를 인자로 입력받은 후 for 문을 이용해 리스트에 저장된 인스턴스를 순회합니다. 이때 각 인스턴스에서 print_info() 메서드를 호출합니다.

연락처 출력 기능은 메인 메뉴에서 '2. 연락처 출력'을 선택했을 때 수행돼야 합니다. 따라서 다음과 같이 run() 함수도 수정합니다.

```
def run():  
  
    contact_list = []  
  
    while 1:  
  
        menu = print_menu()  
  
        if menu == 1:  
  
            contact = set_contact()  
  
            contact_list.append(contact)  
  
        elif menu == 2:  
  
            print_contact(contact_list)  
  
        elif menu == 4:
```

break

지금까지 작성한 프로그램을 실행해 봅시다. 먼저 1 번 메뉴를 선택한 후 임의로 연락처 정보를 입력하고 2 번 메뉴를 선택하면 그림 8.27 과 같이 입력된 정보가 정상적으로 출력되는 것을 확인할 수 있습니다.



```
Run contact
C:\Anaconda3\python.exe C:/Users/Jason/PycharmProjects/Contact/contact.py
1. 연락처 입력
2. 연락처 출력
3. 연락처 삭제
4. 종료
메뉴선택: 1
Name: test
Phone Number: 010-0000-0000
E-mail: test
Address: test
1. 연락처 입력
2. 연락처 출력
3. 연락처 삭제
4. 종료
메뉴선택: 2
Name: test
Phone Number: 010-0000-0000
E-mail: test
Address: test
1. 연락처 입력
2. 연락처 출력
3. 연락처 삭제
4. 종료
메뉴선택: 4
Process finished with exit code 0
```

그림 8.27 연락처 출력 기능 확인

2-6) 연락처 삭제하기

이번에는 입력된 연락처를 삭제하는 기능을 추가해 보겠습니다. 연락처를 삭제하려면 삭제하고자하는 연락처의 이름을 입력받은 후 연락처 리스트에서 해당 이름을 찾아서 지우면 됩니다.

연락처 리스트에서 연락처를 삭제하기 위해 다음과 같이 `delete_contact()` 함수를 작성합니다. 이 함수는 연락처 리스트와 삭제할 이름을 인자로 입력받습니다. `delete_contact()` 함수는 연락처 리스트에 저장된 인스턴스 중 삭제하고자 하는 연락처 이름과 같은 연락처가 있는지 확인한 후 같은 연락처가 있을 때 해당 연락처를 삭제합니다.

```
def delete_contact(contact_list, name):  
    for i, contact in enumerate(contact_list):  
        if contact.name == name:  
            del contact_list[i]
```

연락처를 삭제하는 함수를 작성했다면 이번에는 `run()` 함수를 수정하겠습니다. 연락처 삭제는 '3. 연락처 삭제' 메뉴를 선택했을 때 동작하는데, 이때 삭제할 연락처의 이름(Name)을 한 번 더 입력받습니다.

삭제할 연락처의 이름을 입력받았다면 연락처 리스트와 삭제할 이름을 `delete_contact()` 함수의 인자로 전달하면 해당 함수를 통해 연락처가 삭제됩니다.

```
def run():  
    contact_list = []  
    while 1:  
        menu = print_menu()  
        if menu == 1:  
            contact = set_contact()  
            contact_list.append(contact)  
        elif menu == 2:  
            name = input("삭제할 연락처 이름을 입력하세요: ")  
            delete_contact(contact_list, name)  
        print_contact(contact_list)
```

```

elif menu == 3:

    name = input("Name: ")

    delete_contact(contact_list, name)

elif menu == 4:

    break

```

지금까지 작성한 주소록 프로젝트의 전체 소스코드는 예제 8.2와 같습니다. 여러분이 입력한 코드와 틀린 부분이 없는 확인하고 틀린 부분이 없다면 작성한 프로그램을 실행해 보기 바랍니다.

프로그램이 정상적으로 동작하는지 확인하기 위해 여러 명의 연락처를 입력해 보기 바랍니다. 그리고 그중 일부 연락처를 삭제한 후 연락처도 출력해 보기 바랍니다. 어떨까요? 지금까지 배운 파이썬의 기본 문법을 활용하고 프로그램을 단계별로 나눠서 작성하니 조금 복잡해 보이는 프로그램도 어렵지 않게 작성할 수 있지요?

```

class Contact:

    def __init__(self, name, phone_number, e_mail, addr):

        self.name = name

        self.phone_number = phone_number

        self.e_mail = e_mail

        self.addr = addr

```

```

def print_info(self):

    print("Name: ", self.name)

    print("Phone Number: ", self.phone_number)

    print("E-mail: ", self.e_mail)

    print("Address: ", self.addr)

```

```

def set_contact():

```



```
name = input("Name: ")
```

```
phone_number = input("Phone Number: ")
```

```
e_mail = input("E-mail: ")
```

```
addr = input("Address: ")
```

```
contact = Contact(name, phone_number, e_mail, addr)
```

```
return contact
```

```
def print_contact(contact_list):
```

```
    for contact in contact_list:
```

```
        contact.print_info()
```

```
def delete_contact(contact_list, name):
```

```
    for i, contact in enumerate(contact_list):
```

```
        if contact.name == name:
```

```
            del contact_list[i]
```

```
def print_menu():
```

```
    print("1. 연락처 입력")
```

```
    print("2. 연락처 출력")
```

```
    print("3. 연락처 삭제")
```

```
    print("4. 종료")
```

```
    menu = input("메뉴선택: ")
```

```
    return int(menu)
```

```
def run():
```

```
    contact_list = []
```

```
    while 1:
```

```
        menu = print_menu()
```

```
        if menu == 1:
```

```
            contact = set_contact()
```

```
            contact_list.append(contact)
```

```
        elif menu == 2:
```

```
            print_contact(contact_list)
```

```
        elif menu == 3:
```

```
            name = input("Name: ")
```

```
            delete_contact(contact_list, name)
```

```
        elif menu == 4:
```

```
            break
```

```
if __name__ == "__main__":
```

```
    run()
```

예제 8.2 주소록 프로젝트의 전체 소스코드

3) 주소록 프로젝트의 기능 향상

이번 절에서는 8.2 절에서 작성한 연락처 관리 프로그램에 저장 기능을 추가해 보겠습니다. 지금까지 작성한 주소록 프로젝트는 프로그램은 프로그램을 종료하면 입력된 모든 연락처가 사라집니다. 이는 입력받은 연락처가 Contact 클래스의 인스턴스로서 메모리에 할당되기 때문에 프로그램이 종료되면 메모리에 있는 모든 변수가 삭제되기 때문입니다.

이러한 문제를 해결하려면 어떻게 해야 할까요? 가장 간단한 방법은 프로그램이 종료되는 시점을 기준으로 유효한 연락처의 정보를 모두 파일로 저장하는 것입니다. 파일 입/출력은 7장에서 살펴봤기 때문에 바로 해당 기능을 추가해 보겠습니다.

8.2 절에서 구현한 메인 메뉴의 기능과 달리 파일 출력 기능은 메인 메뉴를 통해 제공되는 기능이 아닙니다. 단지 프로그램이 종료될 때 종료 시점에 유효한 데이터를 파일로 출력하면 됩니다. 반대로 프로그램이 실행될 때는 기존에 저장된 연락처를 불러와야(Load) 합니다.

3-1) 연락처 저장 함수 작성하기

맨 먼저 작성해볼 함수는 프로그램이 종료될 때 유효한 연락처 정보를 파일로 저장하는 역할을 하는 `store_contact()`라는 함수입니다. 이 함수는 `contact_list`라는 리스트를 입력받은 후 해당 리스트에 있는 `Contact` 인스턴스를 순회하면서 데이터를 파일로 저장합니다.

```
def store_contact(contact_list):  
    f = open("contact_db.txt", "wt")  
    for contact in contact_list:  
        f.write(contact.name + '\n')  
        f.write(contact.phone_number + '\n')  
        f.write(contact.e_mail + '\n')  
        f.write(contact.addr + '\n')  
    f.close()
```

`store_contact()` 함수는 함수가 호출될 때 먼저 'contact_db.txt'라는 이름의 텍스트 파일을 'wt' 모드로 엽니다. 그런 다음 `contact_list`를 순회하면서 각 인스턴스의 정보를 `write` 함수를 통해 출력합니다. 이때 출력되는 각 정보가 파일에서 각 라인 단위로 저장되도록 '\n'을 사용했습니다. `contact_list`의 모든 데이터를 파일로 출력했다면 `f.close()`를 통해 파일을 닫습니다.

파일로 저장하는 기능은 프로그램이 종료될 때 호출하면 되므로 기존에 작성된 `run()` 함수를 수정합니다.

```
def run():  
    contact_list = []  
    while 1:  
        menu = print_menu()  
        if menu == 1:  
            contact = set_contact()  
            contact_list.append(contact)
```

```

elif menu == 2:
    print_contact(contact_list)

elif menu == 3:
    name = input("Name: ")
    delete_contact(contact_list, name)

elif menu == 4:
    store_contact(contact_list)

break

```

수정된 프로그램을 실행한 후 그림 8.28 와 같이 연락처를 하나 입력하고 프로그램을 종료해 봅시다.

```

Run contact
C:\#Anaconda3\python.exe C:/Users/Jason/PycharmProjects/Contact/contact.py
1. 연락처 입력
2. 연락처 출력
3. 연락처 삭제
4. 종료
메뉴선택: 1
Name: test1
Phone Number: 010-0000-0000
E-mail: test1@gmail.com
Address: seoul
1. 연락처 입력
2. 연락처 출력
3. 연락처 삭제
4. 종료
메뉴선택: 4
Process finished with exit code 0

```

그림 8.28 연락처 입력 후 종료

그림 8.29 와 같이 PyCharm 의 프로젝트 화면을 보면 contact_db.txt 라는 이름의 파일이 생성된 것을 확인할 수 있습니다.

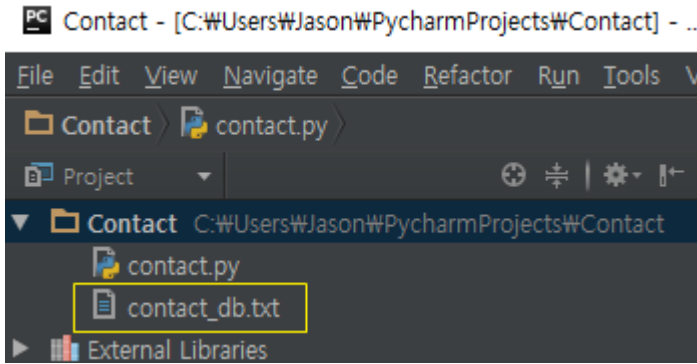


그림 8.29 생성된 파일 확인

해당 파일의 이름을 더블클릭하면 그림 8.30 과 같이 PyCharm 의 에디터 화면에서 생성된 텍스트 파일의 내용을 확인할 수 있습니다. 참고로 contact_db.txt 파일은 프로젝트 소스코드가 존재하는 디렉터리에 내에 생성되는데, PyCharm 에서는 이를 프로젝트 뷰에서 바로 보여주기 때문에 프로그램을 작성할 때보다 편리하게 파일 내용을 확인할 수 있습니다.

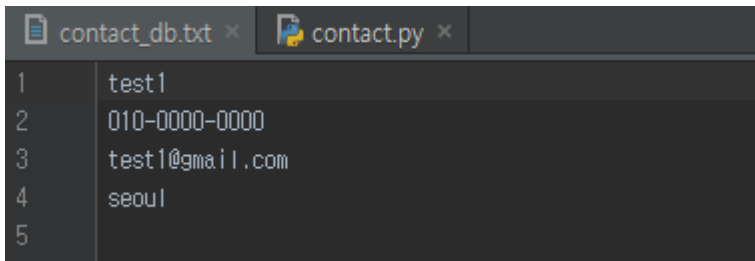


그림 8.30 생성된 파일 내용 확인

3-2) 연락처 불러들이기

앞서 프로그램이 종료될 때 유효한 연락처를 파일로 저장하는 기능을 구현해 보았습니다. 이번에는 이와 반대로 이전에 파일로 저장돼 있던 연락처 정보를 프로그램이 시작될 때 불러들이는 기능을 구현해 보겠습니다.

파일로부터 연락처를 로드하는 함수의 이름은 `load_contact` 입니다. 이 함수 역시 `contact_list` 를 인자로 받으며, 함수 내부에서 'contact_db.txt' 파일을 열어서 해당 파일을 라인 단위로 저장된 이름, 전화번호, 이메일, 주소로 읽어 들여 `Contact` 클래스의 인스턴스를 생성합니다. 그런 다음 생성한 인스턴스를 `contact_list` 에 추가합니다.

```
def load_contact(contact_list):  
  
    f = open("contact_db.txt", "rt")  
  
    lines = f.readlines()  
  
    num = len(lines) / 4  
  
    num = int(num)  
  
    for i in range(num):  
  
        name = lines[4*i].rstrip('\n')  
  
        phone = lines[4*i+1].rstrip('\n')  
  
        email = lines[4*i+2].rstrip('\n')  
  
        addr = lines[4*i+3].rstrip('\n')  
  
        contact = Contact(name, phone, email, addr)  
  
        contact_list.append(contact)  
  
    f.close()
```

`load_contact()` 함수는 먼저 `readlines()` 함수를 이용해 파일에 있는 모든 데이터를 읽습니다. 연락처 하나당 4 줄의 데이터가 존재하므로 파일에서 읽어들이는 전체 라인 수를 4로 나누어 몇 개의 데이터가 존재하는지 확인합니다. 나눗셈 연산을 수행하면 `num` 값이 실수가 되는데, 이 값을 `int()` 내장 함수를 사용해 정수형으로 형변환합니다.

for 문에서는 num 의 개수만큼 루프를 돌면서 lines 리스트에 저장된 데이터를 읽어 들여 Contact 클래스의 인스턴스를 생성하고 생성한 인스턴스를 contact_list 에 추가합니다.

앞서 설명해 드린 것처럼 파일로 저장된 연락처의 로드는 연락처 관리 프로그램이 실행될 때 이뤄져야 합니다. 따라서 다음과 같이 run() 함수의 시작 부분에서 load_contact() 함수를 호출하면 됩니다.

```
def run():  
    contact_list = []  
    load_contact(contact_list)  
    while 1:  
        menu = print_menu()  
        if menu == 1:  
            contact = set_contact()  
            contact_list.append(contact)  
        elif menu == 2:  
            print_contact(contact_list)  
        elif menu == 3:  
            name = input("Name: ")  
            delete_contact(contact_list, name)  
        elif menu == 4:  
            store_contact(contact_list)  
        break
```

예제 8.3 은 지금까지 작성한 주소록 프로젝트의 최종 코드입니다. 해당 코드는 파일 입/출력 기능을 추가로 수행함으로써 프로그램이 종료되더라도 프로그램이 실행된 시점의 데이터를 하드디스크에 저장할 수 있습니다. 또한 프로그램이 시작될 때 하드디스크에 저장된 파일을 읽어 들여 프로그램이 종료되던 시점의 데이터를 그대로 복원하는 기능을 제공합니다.


```
class Contact:
```

```
    def __init__(self, name, phone_number, e_mail, addr):
```

```
        self.name = name
```

```
        self.phone_number = phone_number
```

```
        self.e_mail = e_mail
```

```
        self.addr = addr
```

```
    def print_info(self):
```

```
        print("Name: ", self.name)
```

```
        print("Phone Number: ", self.phone_number)
```

```
        print("E-mail: ", self.e_mail)
```

```
        print("Address: ", self.addr)
```

```
def set_contact():
```

```
    name = input("Name: ")
```

```
    phone_number = input("Phone Number: ")
```

```
    e_mail = input("E-mail: ")
```

```
    addr = input("Address: ")
```

```
    contact = Contact(name, phone_number, e_mail, addr)
```

```
    return contact
```

```
def print_contact(contact_list):
```

```
    for contact in contact_list:
```

```
contact.print_info()
```

```
def delete_contact(contact_list, name):
```

```
    for i, contact in enumerate(contact_list):
```

```
        if contact.name == name:
```

```
            del contact_list[i]
```

```
def load_contact(contact_list):
```

```
    f = open("contact_db.txt", "rt")
```

```
    lines = f.readlines()
```

```
    num = len(lines) / 4
```

```
    num = int(num)
```

```
    for i in range(num):
```

```
        name = lines[4*i].rstrip('\n')
```

```
        phone = lines[4*i+1].rstrip('\n')
```

```
        email = lines[4*i+2].rstrip('\n')
```

```
        addr = lines[4*i+3].rstrip('\n')
```

```
        contact = Contact(name, phone, email, addr)
```

```
        contact_list.append(contact)
```

```
    f.close()
```

```
def store_contact(contact_list):
```

```
    f = open("contact_db.txt", "wt")
```

```
for contact in contact_list:
```

```
    f.write(contact.name + '\n')
```

```
    f.write(contact.phone_number + '\n')
```

```
    f.write(contact.e_mail + '\n')
```

```
    f.write(contact.addr + '\n')
```

```
f.close()
```

```
def print_menu():
```

```
    print("1. 연락처 입력")
```

```
    print("2. 연락처 출력")
```

```
    print("3. 연락처 삭제")
```

```
    print("4. 종료")
```

```
    menu = input("메뉴선택: ")
```

```
    return int(menu)
```

```
def run():
```

```
    contact_list = []
```

```
    load_contact(contact_list)
```

```
    while 1:
```

```
        menu = print_menu()
```

```
        if menu == 1:
```

```
            contact = set_contact()
```

```
            contact_list.append(contact)
```

```
elif menu == 2:
```

```
    print_contact(contact_list)
```

```
elif menu == 3:
```

```
    name = input("Name: ")
```

```
    delete_contact(contact_list, name)
```

```
elif menu == 4:
```

```
    store_contact(contact_list)
```

```
break
```

```
if __name__ == "__main__":
```

```
    run()
```

예제 8.3 주소록 프로젝트의 최종 코드

3) 명령 프롬프트에서 프로그램 실행하기

지금까지 PyCharm 을 통해 프로그램을 개발하고 동시에 실행해 보면서 프로그램을 작성했습니다.

PyCharm 을 이용해 프로그램을 작성하면 하드디스크에 파이썬 소스코드 형태로 프로그램이 저장됩니다.

이번 절에서는 윈도우의 명령 프롬프트를 이용해 직접 파이썬 코드를 실행해 보겠습니다.

윈도우에서 명령 프롬프트를 실행하려면 윈도우 시작 버튼 -> 실행 -> cmd 를 입력하면 됩니다. 명령 프롬프트가 나타나면 지금까지 작성한 Contact 프로젝트 경로로 이동합니다. 저자의 PC 를 기준으로 설명해 드리면 현재 프로젝트 파일은 'C:\Users\Jason\PyCharmProjects\Contact'에 있습니다. 따라서 다음과 같이 cd(change directory) 명령을 이용해 해당 디렉터리 경로로 이동합니다. 파이썬 프로젝트가 들어있는 곳으로 이동하고 나면 dir 명령어를 입력해 현재 디렉터리에 contact.py 라는 파일이 있는지 확인합니다.

Microsoft Windows [Version 10.0.14393]

(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Jason>cd PycharmProjects

C:\Users\Jason\PycharmProjects>cd Contact

C:\Users\Jason\PycharmProjects\Contact>dir

C 드라이브의 볼륨에는 이름이 없습니다.

볼륨 일련 번호: 3E7A-5817

C:\Users\Jason\PycharmProjects\Contact 디렉터리

2016-10-05 오전 08:18 <DIR> .

2016-10-05 오전 08:18 <DIR> ..

2016-10-05 오전 08:15 2,319 contact.py

1 개 파일

2,319 바이트

2 개 디렉터리 93,350,182,912 바이트 남음

C:\Users\Jason\PycharmProjects\Contact>

cd 명령어를 이용해 정상적으로 프로젝트 경로로 이동했다면 'python contact.py'를 입력해 파이썬 프로그램을 실행할 수 있습니다.

C:\Users\Jason\PycharmProjects\Contact>python contact.py

1. 연락처 입력

2. 연락처 출력

3. 연락처 삭제

4. 종료

메뉴선택:

프로그램을 개발하는 과정에서는 PyCharm 과 같은 IDE 상에서 파이썬 프로그램을 직접 실행하지만 프로그램 개발이 완료된 후에는 방금 설명해 드린 바와 같이 직접 파이썬 소스코드를 사용해 프로그램을 실행하게 됩니다.