데이터 과학을 위한
파이썬 프로그래밍

2판

Chapter 11
모듈과 패키지

목차

학습목표

- 모듈과 패키지의 개념을 이해한다.
- 모듈 만들기 실습을 진행한다.
- 패키지 만들기 실습을 진행한다.
- 가상환경에 대해 이해한다.

01
모듈과 패키지의 이해

(a) 파이썬 (b) 자바

그림 11-1 파이썬과 자바의 코드 작성 비유

- 파이썬은 매우 간결한 프로그래밍 언어!

  ☞ 많은 사람들이 이미 파이썬으로 프로그램을 작성해 두었기 때문

- 파이썬에서는 이미 작성된 프로그램을 **모듈(module)**이라고 하고, 이 프로그램의 묶음을 **패키지(packages)**라고 함

1. 모듈의 개념

- 2014년 구글(Google)의 'Ara' 프로젝트 - 모듈형 휴대전화 판매
  - 휴대전화에 들어가는 카메라, 메모리, 와이파이 등을 하나의 블록으로 개별 판매하고 조립할 수 있는 환경을 만들어주는 프로젝트
  - 이미 PC를 통해 대중화되어 있는 컴퓨터 제작 방식 - 컴퓨터의 여러 부품은 하나의 완성된 제품으로, 또는 여러 컴퓨터에 호환되어 사용할 수 있는 형태로 판매되고 있음

그림 11-2 구글의 프로젝트 Ara 콘셉트

1. 모듈의 개념

- **프로그래밍에서의 모듈:** 작은 프로그램 조각

  - 하나하나 연결해 어떤 목적을 가진 프로그램을 만들기 위한 작은 프로그램.

- **인터페이스:** 함수에서 매개변수를 입력하는 약속

  - 해당 모듈을 사용하기 위해서는 모듈 간의 연결을 위한 약속이 필요한데, 이것을 인터페이스라고 함.

그림 11-3 Kakao Developers의 개발가이드

1. 모듈의 개념

- 파이썬에서 내장 모듈은 기본적으로 제공하는 대표적인 내장 모듈은 random
- random 모듈을 호출하기 위한 코드:

```
>>> import random
>>> random.randint(1, 1000)
198
```

  - 위의 코드를 실행하면 1~1000 중 임의의 숫자가 생성됨.

2. 패키지의 개념

- **패키지(packages):** 모듈의 묶음
- **from:** 모듈을 호출하기 위해 패키지부터 호출하는 명령어

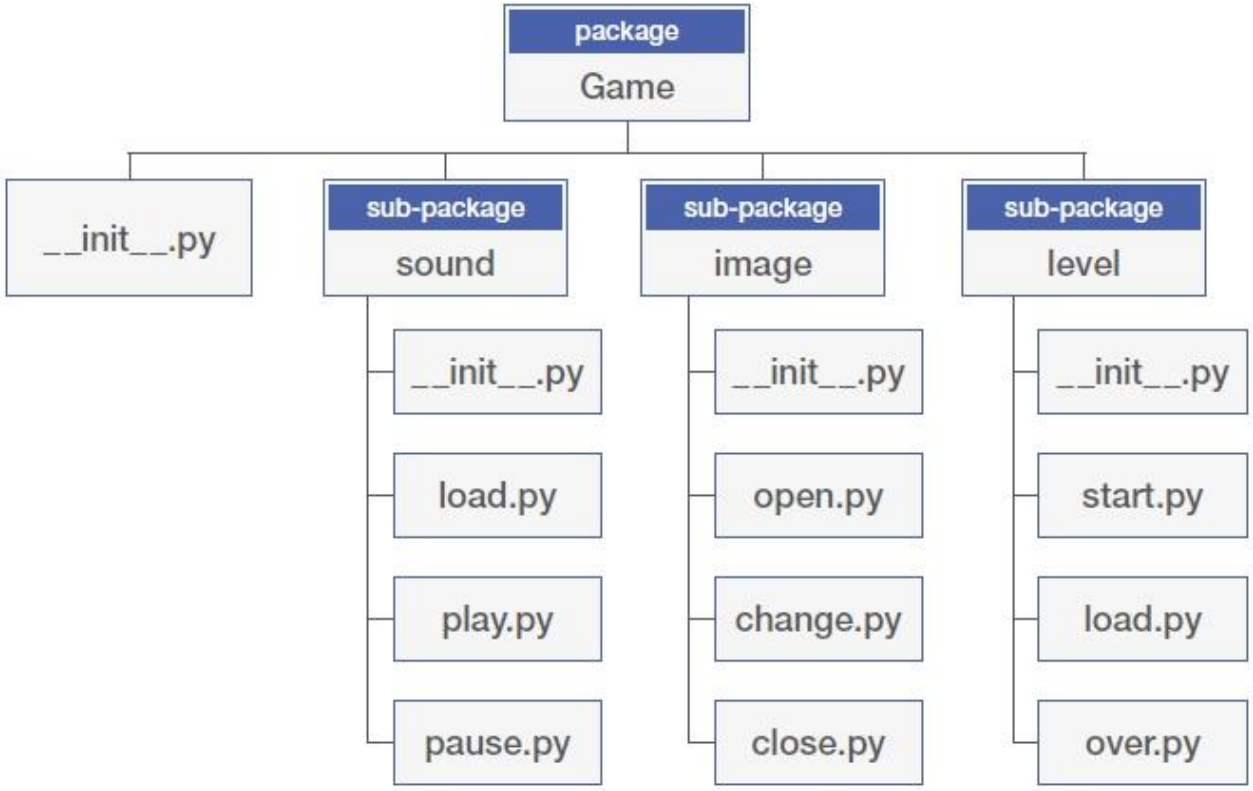


그림 11-4 모듈과 패키지의 관계

02
모듈 만들기

1. 모듈 만들기 실습

- 파이썬에서는 .py 파일 자체가 모듈임.

- [코드 11-1]과 같은 코드를 작성하여 'fah_converter.py'로 저장

[코드 11-1]

```
1 def covert_c_to_f(celcius_value):
2     return celcius_value * 9.0 / 5 + 32
```

1. 모듈 만들기 실습

- 해당 모듈을 사용하는 코드를 [코드 11-2]와 같이 작성하여 'module_ex.py'로 저장

[코드 11-2]

```
1 import fah_converter
2
3 print("Enter a celsius value:")
4 celsius = float(input())
5 fahrenheit = fah_converter.covert_c_to_f(celsius)
6 print("That's", fahrenheit, "degrees Fahrenheit.")
```

[실행결과]

```
Enter a celsius value:
10                                    ← 사용자 입력
That's 50.0 degrees Fahrenheit.       ← 결과값 출력
```

2. 네임스페이스

- **네임스페이스(namespace):** 모듈 호출의 범위를 지정하는 것
  - 하나의 .py 파일 안에는 클래스·함수·변수가 있으며 import를 사용하여 이들을 호출할 수 있는데 때로는 클라이언트 프로그램의 함수 이름과 호출된 모듈의 함수 이름이 같은 경우가 있음
  - 익숙하지 않아 발생하는 실수일 수도 있고 필요에 따라 기존 클래스를 상속받아 사용하다가 발생할 수도 있음.
  - 이 경우 호출된 모듈의 사용 범위를 명확히 지정해야 하는데 이때 사용하는 개념이 바로 네임스페이스

2. 네임스페이스

- **네임스페이스를 만드는 방법 ①**

: 모듈 이름에 알리아스(alias)를 생성하여 모듈 안으로 코드를 호출하는 방법

[코드 11-3]

```
1 import fah_converter as fah
2 print(fah.covert_c_to_f(41.6))
```

[실행결과]

```
106.88000000000001
```

2. 네임스페이스

- **네임스페이스를 만드는 방법 ②**

: from 구문을 사용하여 모듈에서 특정 함수 또는 클래스만 호출하는 방법

[코드 11-4]

```
1 from fah_converter import covert_c_to_f
2 print(covert_c_to_f(41.6))
```

[실행결과]

```
106.88000000000001
```

2. 네임스페이스

- **네임스페이스를 만드는 방법 ③**

: 해당 모듈 안에 있는 모든 함수, 클래스, 변수를 가져오는 *를 사용하는 것

[코드 11-5]

```
1 from fah_converter import *
2 print(covert_c_to_f(41.6))
```

[실행결과]

```
106.88000000000001
```

3. 내장 모듈의 사용

내장 모듈(built-in module)

- 파이썬은 프로그래밍을 개발하기 위해 기본적으로 사용해야 하는 문자 처리, 웹, 수학과 관련된 다양한 내장 모듈을 제공하며, 별다른 조치 없이 import문 한 줄로 사용할 수 있음.

3.1 random 모듈

- 난수 생성 모듈로, 정수 모듈을 생성하는 randint( ) 함수와 임의의 난수를 생성하는 random( ) 함수가 있음.

```
>>> import random
>>> print(random.randint (0, 100))        # 0~100 사이의 정수 난수를 생성
7
>>> print(random.random())                # 일반적인 난수 생성
0.0565504217895318 46
```

3.2 time 모듈

- 시간과 관련된 모듈로, 일반적으로 시간을 변경하거나 현재 시각을 알려줌.
- 대표적으로 프로그램이 동작하는 현재 시각을 알 수 있음.

```
>>> import time
>>> print(time.localtime())                          # 현재 시각 출력
time.struct_time(tm_year=2018, tm_mon=8, tm_mday=19, tm_hour=22,
tm_min=9, tm_sec=21, tm_wday=6, tm_yday=231, tm_isdst=0)
```

3.3 urllib 모듈

- 웹과 관련된 모듈로, 웹 주소의 정보를 불러옴.
- 대표적으로 urllib의 request 모듈을 사용하면 특정 URL의 정보를 불러올 수 있음.
- urllib.request.urlopen() 코드에서 괄호 안에 특정 웹 주소를 입력하면 해당 주소의 HTML 정보를 가져옴.

```
>>> import urllib.request
>>> response = urllib.request.urlopen("http://theteamlab.io")
>>> print(response.read())
```

3. 내장 모듈의 사용

여기서 잠깐! 파이썬 모듈 검색

이외에도 많은 파이썬 모듈이 있다. 그렇다면 이 모듈들은 어떻게 불러와 사용할 수 있을까?

가장 좋은 방법은 구글에서 검색하는 것이다. 특히 영어로 검색하는 것이 좋다. 예를 들어 프로그래밍 수행에 걸린 시간을 알아내는 모듈을 찾고 싶다면 다음과 같은 방식으로 검색어를 입력하면 된다.

```
python time module run time
```

다른 방법으로는 우리나라의 대표 사이트인 파이썬 코리아에 문의해도 된다. 파이썬 코리아의 페이스북 페이지는 파이썬 개발자에게 많은 정보를 제공하는 대표적인 커뮤니티이다. 다음 주소에서 여러 가지 질문을 하면 많은 개발자가 친절히 알려줄 것이다.

```
https://www.facebook.com/groups/pythonkorea
```

모듈은 여러모로 유용하기 때문에 다양한 사용 방법에 대해서 알아두고, 필요할 때마다 불러와 사용하길 바란다.

03
패키지 만들기

1. 패키지의 구성

- **패키지:** 하나의 대형 프로젝트를 수행하기 위한 모듈의 묶음
  - 모듈은 하나의 파일로 이루어져 있고, 패키지는 파일이 포함된 디렉터리로 구성됨

  → 즉, 여러 개의 .py 파일이 하나의 디렉터리에 들어가 있는 것을 '패키지'라고 함
  - 다른 사람이 만든 프로그램을 불러와 사용하는 것을 라이브러리(library)라고 하는데, 파이썬에서는 패키지를 하나의 라이브러리로 이해하면 됨.

2. 패키지 만들기 실습

2.1 1단계: 디렉터리 구성하기

- 패키지 이름: 'roboadvisor' - 패키지 이름 대부분은 소문자를 사용
- roboadvisor에는 세 가지 기능이 있다고 가정

❶ crawling(크롤링): 주식 관련 데이터를 인터넷에서 가져오는 기능

❷ database(데이터베이스): 가져온 데이터를 데이터베이스에 저장하는 기능

❸ analysis(분석): 해당 정보를 분석하여 의미 있는 값을 뽑아내는 기능

2. 패키지 만들기 실습

- 패키지를 구성하기 위한 첫 번째 단계는 각 패키지 내에서 다시 세부 패키지에 맞춰 디렉터리 를 구성하는 것.

```
mkdir roboadvisor
cd roboadvisor
mkdir crawling
mkdir database
mkdir analysis
```

그림 11-5 세부 패키지에 맞춰 디렉터리 생성

- [그림 11-6]과 같이 cmd 창에서 만든 디렉터리가 구성됨

그림 11-6 roboadvisor 디렉터리 구성

2. 패키지 만들기 실습

2.2 2단계: 디렉터리별로 필요한 모듈 만들기

- 2단계에서는 만들어진 디렉터리에 필요한 모듈을 만든다.
- 하나의 패키지는 중첩된 구조로, 패키지 안에 또 하나의 패키지가 들어갈 수 있음.
- 각각의 디렉터리를 하나의 패키지로 선언하기 위해서는 예약된 파일을 만들어야 함.

그림 11-7 패키지의 구조

2. 패키지 만들기 실습

여기서 잠깐! 패키지의 구조 설계

[그림 11-7]의 패키지 구조는 임의로 작성한 것이다. 패키지의 구조를 만들기 위해 프로그램 개발자는 설계 과정을 거쳐야 한다. 하위 패키지별로 해야 하는 일과 하위 패키지에 소속된 모듈들이 해야 할 일을 따로 정의해 각 모듈에 역할을 부여하는 것이다. 이 과정에는 많은 경험과 지식이 필요하다. 지금 단계에서는 어려울 수 있으니 이 예제에서는 대략적인 구조와 역할을 임의로 지정하였다.

2. 패키지 만들기 실습

➢각 하위 패키지에 포함된 모듈에 필요한 기능을 구현하기 위해 코드 작성

[코드 11-6] series.py(analysis 디렉터리)

```
1 def series_test():
2     print("series")
```

[코드 11-7] statics.py(analysis 디렉터리)

```
1 def statics_test():
2     print("statics")
```

- 작성한 모듈을 실제로 사용하기 위해서 파이썬 셸에 다음과 같이 입력.

```
>>> from roboadvisor.analysis import series
>>> series.series_test()
series
```

2.3 3단계: 디렉터리별로 _ _init_ _.py 구성하기

- _ _init_ _은 해당 디렉터리가 파이썬의 패키지라고 선언하는 초기화 스크립트임.
- _ _init_ _.py 파일은 파이썬의 거의 모든 라이브러리에 있음.

그림 11-8 scikit-learn에서 확인할 수 있는 __init__.py 파일

- _init_ _.py 파일은 패키지 개발자나 설치 시 확인해야 할 내용 등의 메타데이터라고 할 수 있음.

2. 패키지 만들기 실습

[코드 11-8] _ _init_ _.py(roboadvisor 디렉터리)

```
1 import analysis
2 import crawling
3 import database
4
5 __all__ = ['analysis', 'crawling', 'database']
```

- oboadvisor 디렉터리에는 3개의 하위 패키지 analysis, crawling, database가 있음.

- 각각의 패키지를 _ _init_ _.py 안에 _ _all_ _과 import문을 사용해 선언해야 함.

- 따라서 _ _all_ _이라는 리스트형의 변수를 만들어 차례대로 하위 패키지의 이름을 입력하고, 같은 방법으로 각 하위 패키지를 import문으로 호출함.

2. 패키지 만들기 실습

[코드 11-9] _ _init__.py(analysis 디렉터리)

```
1 from . import series
2 from . import statics
3
4 __all__ = ['series', 'statics']
```

- analysis 디렉터리의 _ _init_ _.py 파일은 각 패키지에 포함된 모듈명을 모두 작성해야 함. 상당히 번거로운 작업이지만 패키지로 표시하기 위해 꼭 해야 하는 작업이며 패키지별로 모두 처리해야 함.

- crawling과 database 디렉터리의 _ _init_ _.py 파일에도 같은 방식으로 코드를 입력하고 저장.

여기서 잠깐! __init__.py 파일을 만들지 않으면 어떤 문제가 발생할까?

파이썬 3.6 버전 이상에서는 __init__.py 파일을 만들지 않아도 큰 문제가 생기지 않는다. 하지만 파이썬 3.3 버전 이하에서는 __init__.py가 없을 경우 해당 디렉터리를 패키지로 인정하지 않는다. 물론 많은 사람이 상위 버전의 파이썬을 사용하기 때문에 문제되지 않을 수도 있지만, 간혹 하위 버전의 파이썬을 사용할 수도 있으니 __init__.py 파일은 패키지를 만들 때 반드시 만드는 것이 좋다.

2.4 4단계: _ _main_ _.py 파일 만들기

- 패키지를 한 번에 사용하기 위해 roboadvisor 디렉터리에 _ _main_ _.py 파일 생성

[코드 11-10] _ _main_ _.py

```
1 from analysis.series import series_test
2 from crawling.parser import parser_test
3
4 if __name__ == '__main__':
5     series_test()
6     parser_test()
```

2.5 5단계: 실행하기(패키지 이름만 호출)

- 모든 코드를 작성한 후 해당 패키지의 최상위 디렉터리(이 예시에서는 roboadvisor의 상위 디렉터리)에서 'python 패키지명'을 입력하여 실행함.

```
python roboadvisor
series
parser
```

TIP [그림 11-9]의 첫 번째 줄과 같이 roboadvisor 디렉터리에서 패키지를 실행하면 오류가 발생. 반드시 최상위 디렉터리에서 패키지를 실행해야 함.

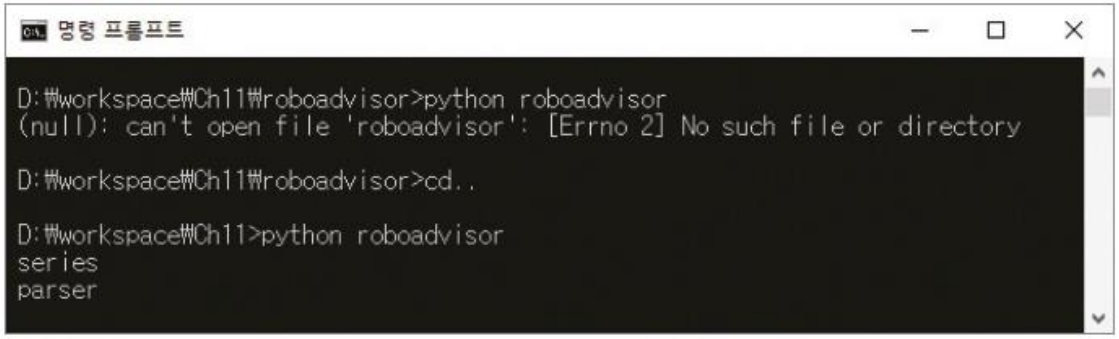


그림 11-9 패키지 실행

3. 패키지 네임스페이스

3.1 절대 참조

- 전체 패키지의 구조를 생각해 모듈의 경로(path)를 모두 호출하는 것

```
from roboadvisor.analysis import series
```

- 위 코드에서 from은 패키지 이름 roboadvisor부터 시작하여 series까지 모든 경로를 입력함. ['from 전체 패키지.서브 패키지 import 모듈'의 형식] ☞ 절대 참조

- [코드 11-11]처럼 작성해야 다른 서브 모듈에서도 모듈을 쉽게 호출할 수 있음.

[코드 11-11] reference1.py

```
1 __all__ = ['analysis', 'crawling', 'database']
2
3 from roboadvisor import analysis
4 from roboadvisor import crawling
5 from roboadvisor import database
```

3.2 상대 참조

- 현재의 디렉터리를 기준으로 모듈을 호출하는 것

[코드 11-12] reference2.py

```
1 from .series import series_test
2 from ..crawling.parser import parser_test
```

- 가장 중요한 코드는 .series와 ..crawling.parser : 점 1개(.)는 현재 디렉터리를, 점 2개(..)는 부모 디렉터리를 의미

- [그림 11-7]과 같은 패키지 구조라면 현재 디렉터리에서 series 모듈 안의 series_test 함수를 호출하라는 의미

04
가상환경 사용하기

1. 가상환경의 개념

- **가상환경:** 어떤 프로젝트를 수행할 때 파이썬 코드를 수행할 기본 인터프리터뿐 아니라 프로젝트별로 필요한 추가 패키지까지 설치해야 하는데 이러한 패키지를 설치할 때 서로 다른 프로젝트가 영향을 주지 않도록 독립적인 프로젝트 수행 환경을 구성하는 것

- **대표적인 가상환경 도구: virtualenv와 conda**

- **Virtualenv:** 파이썬에서 기본적으로 제공하는 가상환경 도구로, pip을 이용하여 새로운 패키지를 설치할 수 있음.

- **Conda:** miniconda의 전용 패키지 관리 도구로, 가상환경 관리와 패키지 설치를 같이 할 수 있음.

표 11-1 가상환경 도구

가상환경 도구	특징
virtualenv + pip	• 가장 대표적인 가상환경 관리 도구 • 레퍼런스와 패키지가 가장 많음
conda	• 상용 가상환경 도구인 miniconda의 기본 가상환경 도구 • 설치가 쉬워 윈도에서 유용함

2. 가상환경 설정하기

2.1 가상환경 만들기

- 다음과 같은 명령을 cmd 창에 입력.

```
conda create -n my_project python=3.4
```

그림 11-10 가상환경 만들기 명령어

- conda는 실행 명령어, create는 가상환경을 만드는 인수(argument).
- -n my_project에서 -n은 name의 줄임말이고, my_project는 구성할 가상환경 이름.
- python=3.4는 설치되는 파이썬의 버전

2. 가상환경 설정하기

- 이 명령을 실행하면 [그림 11-11]과 비슷한 화면이 나타남.

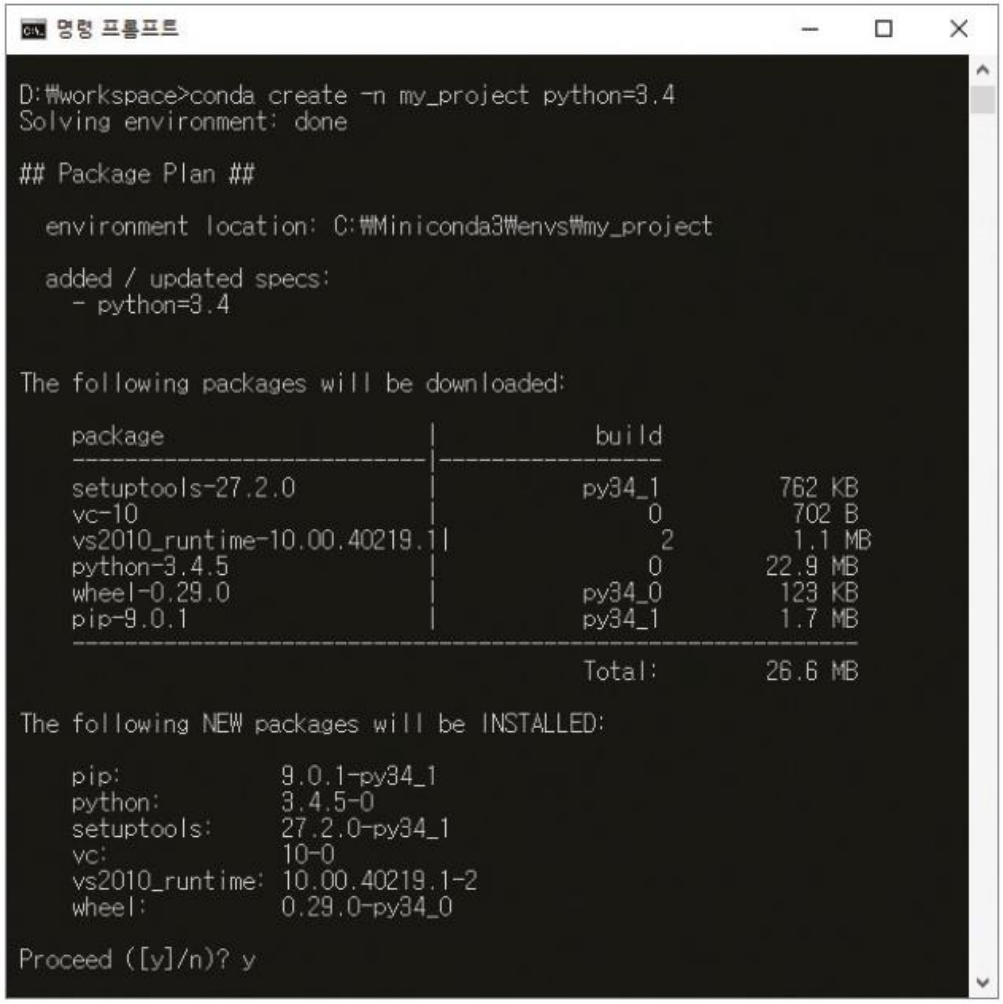


그림 11-11 가상환경 설치 명령 입력

2.2 가상환경 실행하기

- 구성한 가상환경을 실행하는 방법

```
#
# To activate this environment, use:
# > activate my_project
#
# To deactivate an active environment, use:
# > deactivate
#
# * for power-users using bash, you must source
```

그림 11-12 가상환경 설치 화면

```
activate my_project
```

- 이 코드는 my_project라는 가상환경을 활성화(activate)하라는 뜻.

2. 가상환경 설정하기

- 구성된 가상환경의 이름을 activate 다음에 넣으면 해당 가상환경이 실행되고, [그림 11-13]과 같이 프롬프트 앞에 (my_project)라는 가상환경 이름이 붙음.

그림 11-13 가상환경 활성화

- 이 상태에서 'where python'을 입력하면 현재 파이썬 위치가 어디인지 출력

그림 11-14 파이썬의 위치 확인

2. 가상환경 설정하기

- 실행된 가상환경을 종료하기 위해서는 'deactivate'를 입력하면 됨.

그림 11-15 가상환경 종료

2.3 가상환경 패키지 설치하기

- 패키지의 종류는 매우 많고, 테스트할 수 있는 종류도 많음.
- 일단 패키지를 설치하기 위해서 다음과 같은 명령어 입력.

```
conda install matplotlib
```

그림 11-16 matplotlib 설치

2.4 가상환경 패키지 실습하기

- 앞에서 설치한 matplotlib은 대표적인 파이썬 그래프 관리 패키지로, 엑셀과 같은 그래프를 화면에 출력하고 데이터 분석을 할 때 다양한 데이터 분석 도구와 함께 사용됨.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot([1, 2, 3, 4])
[<matplotlib.lines.Line2D object at 0x000001E8CC52C080>]
>>> plt.ylabel('some numbers')
Text(0, 0.5, 'some numbers')
>>> plt.show()
```

- 위 코드를 실행하면 [그림 11-17]과 같은 깔끔한 그래프 화면을 볼 수 있음.

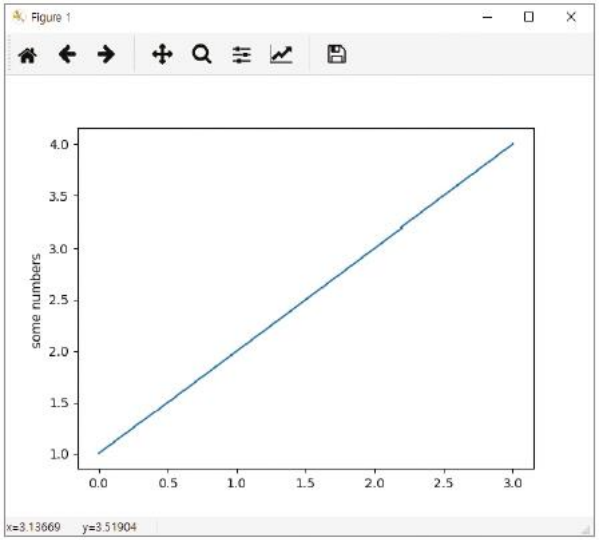


그림 11-17 matplotlib 실행 결과 화면

2. 가상환경 설정하기

여기서 잠깐! jupyter 패키지

데이터를 분석할 때 매우 유용한 패키지로 jupyter가 있다. 먼저 패키지를 설치하기 위해 cmd 창에서 다음과 같은 명령어를 입력한다.

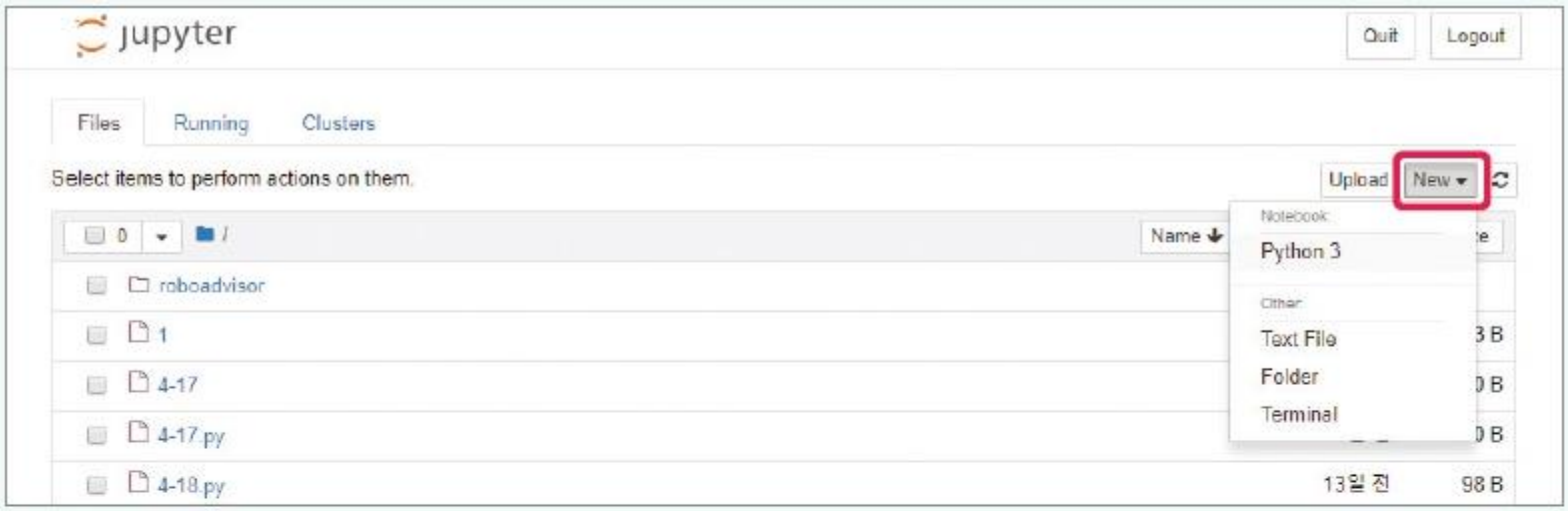
```
conda install jupyter
```

설치가 끝나면 'jupyter notebook'을 입력하여 실행시킨다. 이제 jupyter 환경에서 코딩할 수 있다.

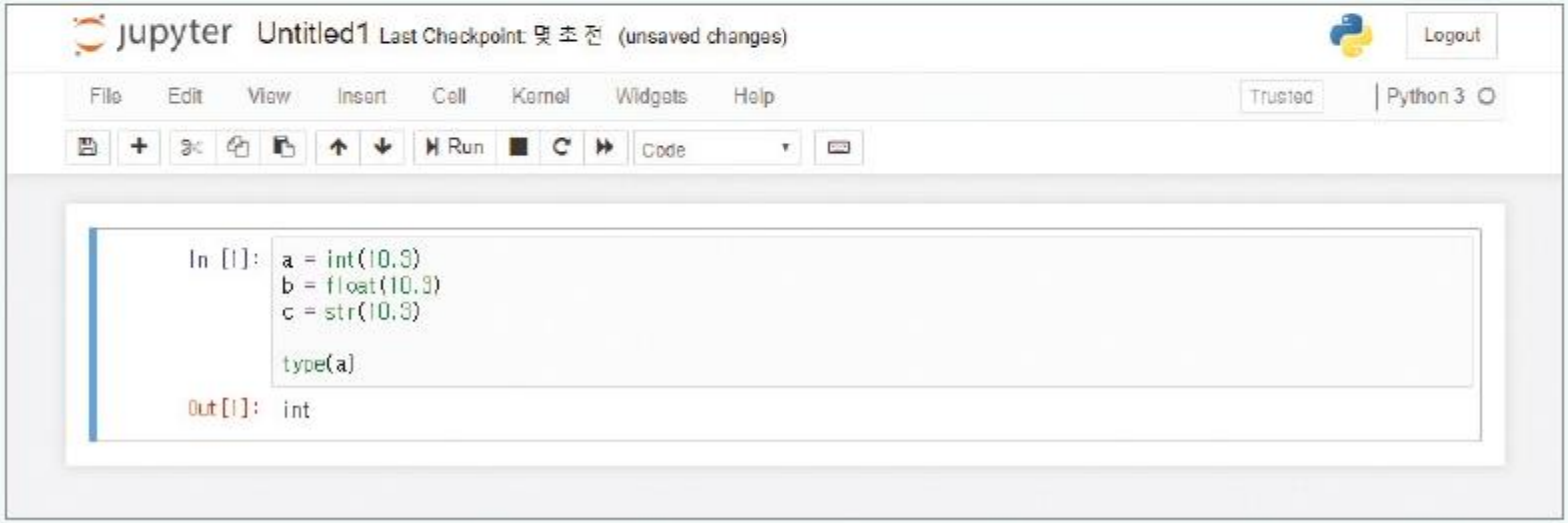
```
jupyter notebook
```

jupyter를 실행하면 웹에서 코딩할 수 있는 환경이 나타난다. 여기에서 [New] 버튼을 클릭하여 새로운 Notebook을 생성한 후 코딩하고 Ctrl + Enter를 누르면 결과를 볼 수 있다.

2. 가상환경 설정하기



(a) jupyter 메인 화면



(b) jupyter 웹 코딩

그림 11-18 jupyter 패키지 환경

Thank you!

한빛아카데미 Hanbit Academy, Inc.