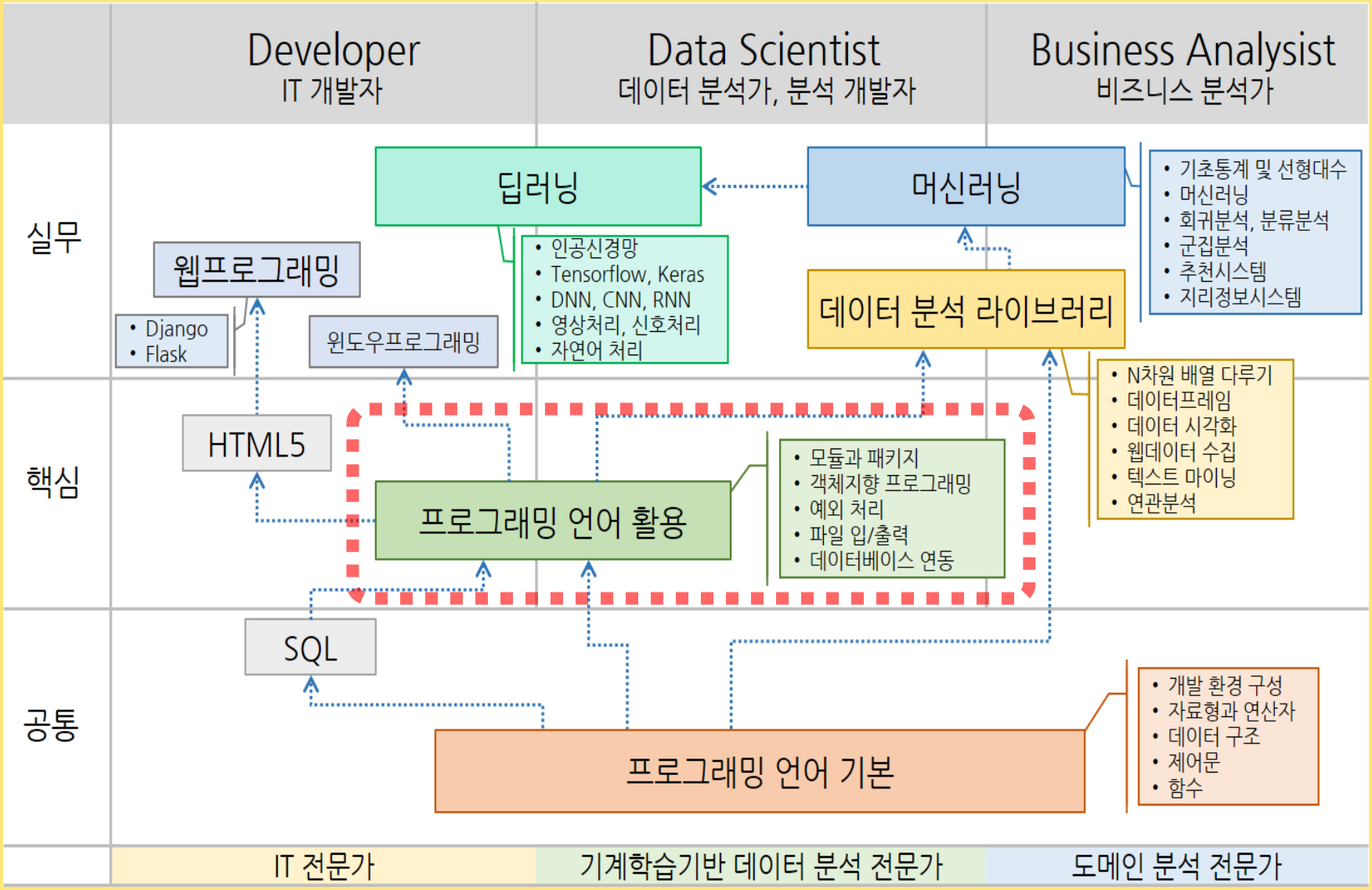

Developer
IT 개발자
Data Scientist
데이터 분석가, 분석 개발자
Business Analysist
비즈니스 분석가
실무
핵심
공통
딥러닝
머신러닝
• 기초통계 및 선형대수
• 머신러닝
• 회귀분석, 분류분석
• 군집분석
• 추천시스템
• 지리정보시스템
웹프로그래밍
• Django
• Flask
윈도우프로그래밍
• 인공신경망
• Tensorflow, Keras
• DNN, CNN, RNN
• 영상처리, 신호처리
• 자연어 처리
데이터 분석 라이브러리
• N차원 배열 다루기
• 데이터프레임
• 데이터 시각화
• 웹데이터 수집
• 텍스트 마이닝
• 연관분석
HTML5
프로그래밍 언어 활용
• 모듈과 패키지
• 객체지향 프로그래밍
• 예외 처리
• 파일 입/출력
• 데이터베이스 연동
SQL
프로그래밍 언어 기본
• 개발 환경 구성
• 자료형과 연산자
• 데이터 구조
• 제어문
• 함수
IT 전문가
기계학습기반 데이터 분석 전문가
도메인 분석 전문가

2부 교육목표 체크리스트

2부의 내용을 공부한 전과 후에 표에 있는 항목들을 체크해 보세요. 체크표시가 늘어날수록 여러분의 실력도 늘어납니다

	교육목표	✔
1	외부 패키지를 설치하고 사용할 수 있다.	
2	모듈과 패키지를 만들어 사용할 수 있다.	
3	생성자를 갖는 클래스를 정의하고 이를 이용해 객체를 만들 수 있다.	
4	상속의 개념을 이해하고 실제 코드에 상속받은 코드를 작성할 수 있다.	
5	클래스 안에 메소드를 정의하고 메소드의 올바른 사용법을 알고 있다.	
6	오류가 발생했을 때 어디에서 발생했는지 찾을 수 있다.	
7	예외의 원인을 파악하고 예외에 따른 적절한 처리를 할 수 있다.	
8	텍스트파일을 불러오고 텍스트 데이터를 파일에 저장할 수 있다.	
9	CSV파일, HDF5 형식의 파일을 불러오고 저장할 수 있다.	
10	데이터베이스 연결 프로그램을 작성할 수 있다.	

6장. 모듈과 패키지

- 1. 파이썬 표준 모듈
- 2. 사용자 정의 모듈
- 3. 패키지
- 4. 파이썬 표준 라이브러리

7장. 객체지향 프로그래밍

8장. 예외 처리

9장. 파일 입/출력 프로그래밍

10장. 데이터베이스 연동

1.1 파이썬 모듈

1절. 파이썬 모듈 사용하기

- 모듈은 파이썬 정의와 문장을 담고 있는 파일(함수 또는 변수 정의)
- 파일 이름은 접미어 .py가 추가 된 모듈 이름
- 함수 또는 변수의 정의를 파일에 넣고 스크립트 또는 인터프리터의 대화형 인스턴스에서 사용하는 방법을 가지고 있음

1.2 파이썬 표준 모듈

1절. 파이썬 모듈 사용하기

- 파이썬 라이브러리 레퍼런스(Python Library Reference)에서 설명하는 **표준 모듈 라이브러리**를 제공
- 일부 모듈은 인터프리터에 내장되어 효율성이나 시스템 호출과 같은 운영 체제 기본 요소 또는 내장되어 있는 작업에 대한 접근을 제공
- 일부 모듈 세트는 기본 플랫폼에 종속
- 문자열(string), 날짜(date), 시간(time), 수학(math), 분수(fractions), 랜덤(random), 파일(file), sqlite3, os, sys, threading, unittest, xml, email, http 등 200여개의 다양한 모듈이 존재
- 일부 모듈 세트는 기본 플랫폼에 종속될 수 있음
- 파이썬 라이브러리 레퍼런스(Python Library Reference)
  - https://docs.python.org/3/library/index.html

1.3 import 하는 방법 : import ~

- import *모듈명*
- 모듈 안의 함수들은 모듈 이름을 붙여 사용

```
import time
```

```
time.ctime()
```

1.3 import 하는 방법 : from A import B

- from *패키지명* import *모듈명 # 패키지는 directory, 모듈은 .py파일*
- from *모듈명* import *함수명*

```
from time import ctime
ctime()
```

```
from time import *
ctime()
```

1.3 import 하는 방법 : import A as B

- import *패키지명* as *패키지별칭*
- import *모듈명* as *모듈별칭*
- A 모듈 또는 패키지의 이름이 길 경우 별칭을 주어 짧게 쓰기 위한 용도로 사용

```
import time as t
t.ctime()
```

1.4 dir()

- 모듈이 정의한 이름을 정렬된 문자열 목록으로 반환

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__sp
ec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees',
'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot',
'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqr
t', 'tan', 'tanh', 'tau', 'trunc']
```

2.1 모듈 만들기

2절. 사용자 정의 모듈

- C:/pylib/fibonacci.py 파일로 작성

fibonacci.py

```
1 def fibo1(n):    # n까지 피보나치 수열 출력
2     a, b = 0, 1
3     while b < n:
4         print(b, end=' ')
5         a, b = b, a+b
6     print()
7
8 def fibo2(n):   # n까지 피보나치 수열 반환
9     result = []
10    a, b = 0, 1
11    while b < n:
12        result.append(b)
13        a, b = b, a+b
14    return result
15
```

2.2 모듈 가져오기

2절. 사용자 정의 모듈

- 모듈을 import 하면…
  - 해당 이름을 가진 내장 모듈 검색, sys.path 변수에 지정된 디렉토리들 검색
- sys.path 초기화
  - 입력 스크립트가 들어있는 디렉토리
  - PYTHONPATH 환경 변수에 지정한 디렉토리
  - 표준 라이브러리 디렉토리. ex) d:₩ai/IDE/Python/Python39₩Lib
- sys.path
  - sys.path.insert(*index*, *path*) 또는 sys.path.append(*path*)로 추가
  - sys.path.remove(*path*)로 제거

```
import sys
sys.path.append('C:/pylib')
```

```
import fibonacci
fibonacci.fibo1(200)
```

```
1 1 2 3 5 8 13 21 34 55 89 144
```

2.3 모듈 실행

- python fibonacci.py <arguments>

```
! python C:₩pylib₩fibonacci.py
```

if __name__ == "__main__":

- 모듈을 실행시킬 때 실행되도록 하려면

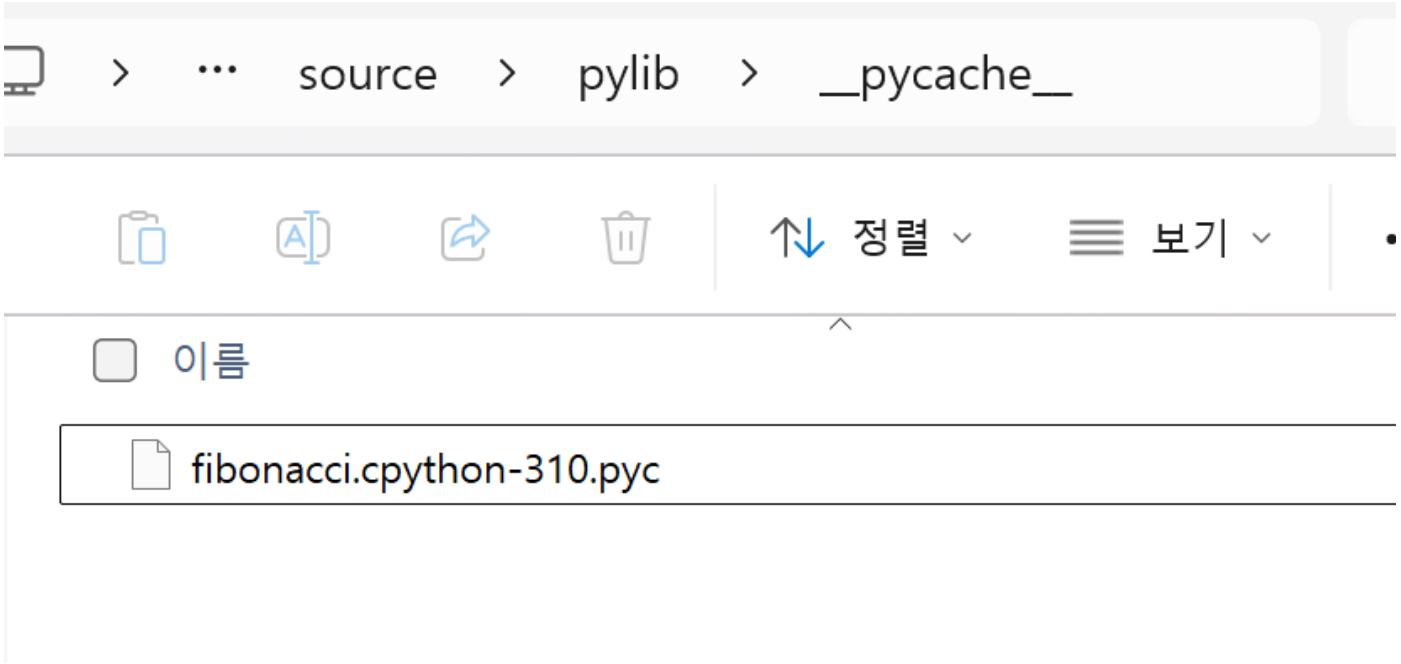
```
16 if __name__ == "__main__":
17     import sys
18     fibo1(int(sys.argv[1]))
19
```

```
! python C:₩pylib₩fibonacci.py 200
```

```
1 1 2 3 5 8 13 21 34 55 89 144
```

2.4 컴파일된 파이썬 파일

- 모듈 로드 속도를 높이기 위해 파이썬은 각 모듈의 컴파일 된 버전을 __pycache__ 디렉토리에 module.version.pyc라는 이름으로 캐시
- 파이썬의 버전은 컴파일 된 파일의 형식을 인코딩하고 컴파일된 파이썬 파일의 이름은 버전 번호를 포함



3절. 패키지

- 패키지(Package)는 “점으로 구분 된 모듈 이름”을 사용하여 파이썬 모듈 네임 스페이스를 구조화하는 방법
- 모듈 이름 A.B는 A라는 패키지에 B라는 서브 모듈을 지정
- 모듈을 사용하면 다른 모듈의 작성자가 서로의 전역 변수 이름을 신경 쓰지 않아도 되므로 점이 있는 모듈 이름을 사용하면 작성자(author)를 절약 할 수 있음
- 넘파이(NumPy)나 파이썬 이미징 라이브러리(Imaging Library)와 같은 다중 모듈 패키지는 서로의 모듈 이름들이 중복 되는 것에 대해 걱정할 필요가 없음

예제 패키지 디렉토리 구조 및 코드

3절 패키지

```
sample_pac/
 +  __init__.py
 +  ab/
     +  __init__.py
     +  a.py
     +  b.py
  + cd/
     +  __init__.py
     +  c.py
```

Editor - C:₩pylib₩sample_pac₩__init__.py

__init__.py - sample_pac | __init__.py - ab

```
1 # -*- coding: utf-8 -*-
2
3 print("sample_pac 패키지를 import 합니다.")
4
```

Editor - C:₩pylib₩sample_pac₩ab₩__init__.py

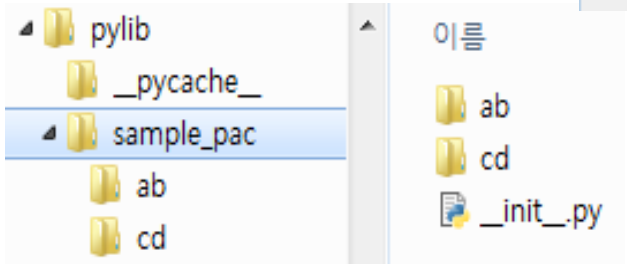
__init__.py - sample_pac | __init__.py - ab

```
1 # -*- coding: utf-8 -*-
2
3 __all__ = ["a"]
4
5 print("ab 패키지를 import 합니다.")
6
```

Editor - C:₩pylib₩sample_pac₩cd₩__init__.py

__init__.py - sample_pac | __init__.py - ab | __init__.py - cd

```
1 # -*- coding: utf-8 -*-
2
3 print("cd 패키지를 import 합니다.")
4
```



Editor - C:₩pylib₩sample_pac₩ab₩a.py

a.py | b.py | __init__.py -

```
1 # -*- coding: utf-8 -*-
2
3 def hello():
4     print("Hello")
5
```

Editor - C:₩pylib₩sample_pac₩ab₩b.py

a.py | b.py | __init__.py - sa

```
1 # -*- coding: utf-8 -*-
2
3 def world():
4     print("World")
5
```

패키지 경로 추가하기

3절 패키지

```
1  import sys
2  sys.path.append("C:/pylib")
```

```
1  sys.path
```

```
['',
 'C:₩₩ProgramData₩₩Anaconda3₩₩python36.zip',
 'C:₩₩ProgramData₩₩Anaconda3₩₩DLLs',
 'C:₩₩ProgramData₩₩Anaconda3₩₩lib',
 'C:₩₩ProgramData₩₩Anaconda3',
 'C:₩₩ProgramData₩₩Anaconda3₩₩lib₩₩site-packages',
 'C:₩₩ProgramData₩₩Anaconda3₩₩lib₩₩site-packages₩₩win32',
 'C:₩₩ProgramData₩₩Anaconda3₩₩lib₩₩site-packages₩₩win32₩₩lib',
 'C:₩₩ProgramData₩₩Anaconda3₩₩lib₩₩site-packages₩₩Pythonwin',
 'C:₩₩ProgramData₩₩Anaconda3₩₩lib₩₩site-packages₩₩IPython₩₩extensions',
 'C:₩₩Users₩₩JK₩₩.ipython',
 'C:/pylib']
```

import ~

3절 패키지

- import 패키지명
- 패키지 사용자는 패키지에서 개별 모듈을 가져올 수 있음
- 상위 패키지를 import 한다고 해서 하위 패키지까지 로드되지 않음
- 한번 로드된 패키지는 다시 로드되지 않음
  - 다시 로드시키려면 커널을 재시작

```
1  import sample_pac
```

```
sample_pac 패키지를 import 합니다.
```

```
1  import sample_pac.ab.a
```

```
ab 패키지를 import 합니다.
```

패키지 리로드

3절 패키지

- importlib.reload(*모듈_또는_패키지명*)

```
1  import importlib
```

```
1  importlib.reload(sample_pac)
```

```
sample_pac 패키지를 import 합니다.

<module 'sample_pac' from 'C:₩₩pylib₩₩sample_pac₩₩__init__.py'>
```

```
1  importlib.reload(sample_pac.ab)
```

```
ab 패키지를 import 합니다.

<module 'sample_pac.ab' from 'C:₩₩pylib₩₩sample_pac₩₩ab₩₩__init__.py'>
```

from ~ import ~

3절 패키지

- from 패키지명 import 모듈명
- 패키지의 하위 패키지 또는 하위 모듈을 가져오는데 사용

```
1  from sample_pac.ab import b
2  b.world()
```

```
World
```

- from 패키지명.모듈명 import 함수명
- 원하는 함수 또는 변수를 직접 가져 오는 것

```
1  from sample_pac.ab.b import world
2  world()
```

```
World
```

from ~ import *

3절 패키지

```
1  import sys
2  sys.path.append("C:/pylib")
3  from sample_pac.ab import *
```

```
sample_pac 패키지를 import 합니다.
ab 패키지를 import 합니다.
```

```
1  a.hello()
```

```
Hello
```

```
1  b.hello()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-ffdd9487a12b> in <module>()
----> 1 b.hello()

NameError: name 'b' is not defined
```

__init__.py의 __all__ 속성

3절 패키지

```
1  import sys
2  sys.path.append("C:/pylib")
3  from sample_pac.ab import *
```

```
sample_pac 패키지를 import 합니다.
ab 패키지를 import 합니다.
```

```
1  a.hello()
```

```
Hello
```

```
1  b.hello()
```

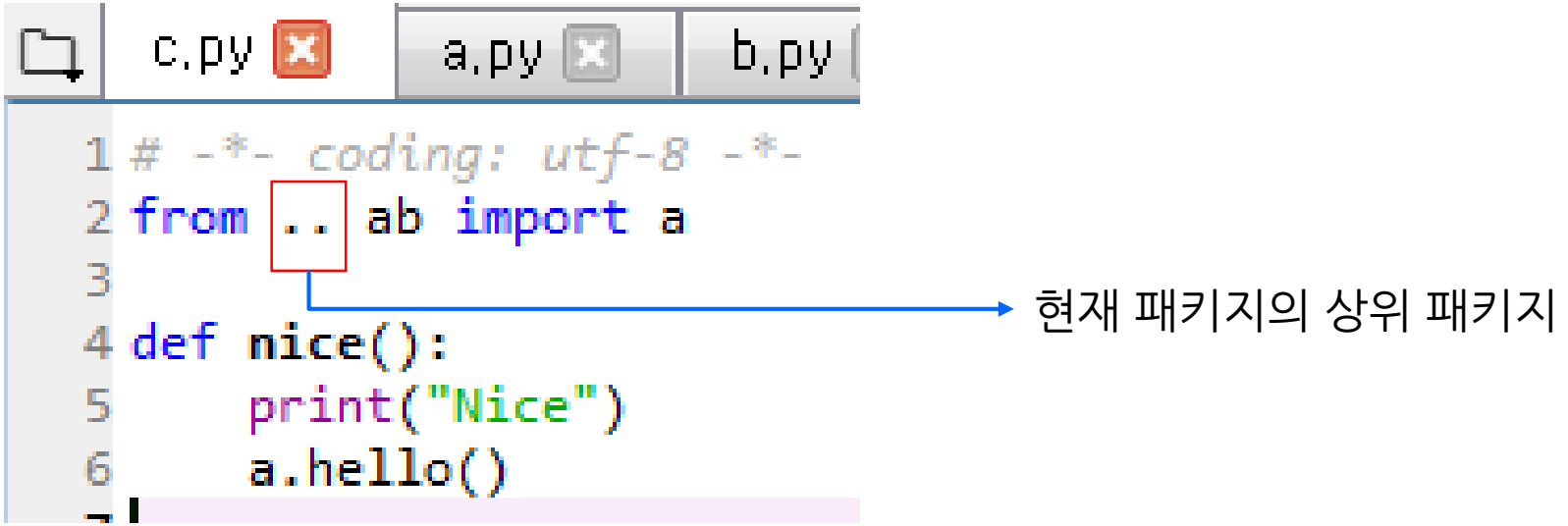
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent
 call last)
<ipython-input-3-ffdd9487a12b> in <module>()
----> 1 b.hello()

NameError: name 'b' is not defined
```

패키지의 __init__.py 파일의 코드에 __all__ 속성으로 모듈의 목록을 정의하면 패키지 import *가 발생할 때 가져와야 하는 모듈 이름의 목록으로 간주함

내부 패키지 참조

3절 패키지

- from module import name 형식의 import 문을 사용하여 상대적인 가져오기를 작성할 수도 있음
- 선행 점(. 또는 ..)을 사용하여 상대 가져 오기에 관련된 현재(.) 및 부모(..) 패키지를 나타냄



3.5 패키지 설치 및 삭제

3절 패키지

- 패키지 이름으로 설치
  - pip install 패키지명
  - conda install 패키지명
  - pip install 패키지명 == 1.3.5(버전)

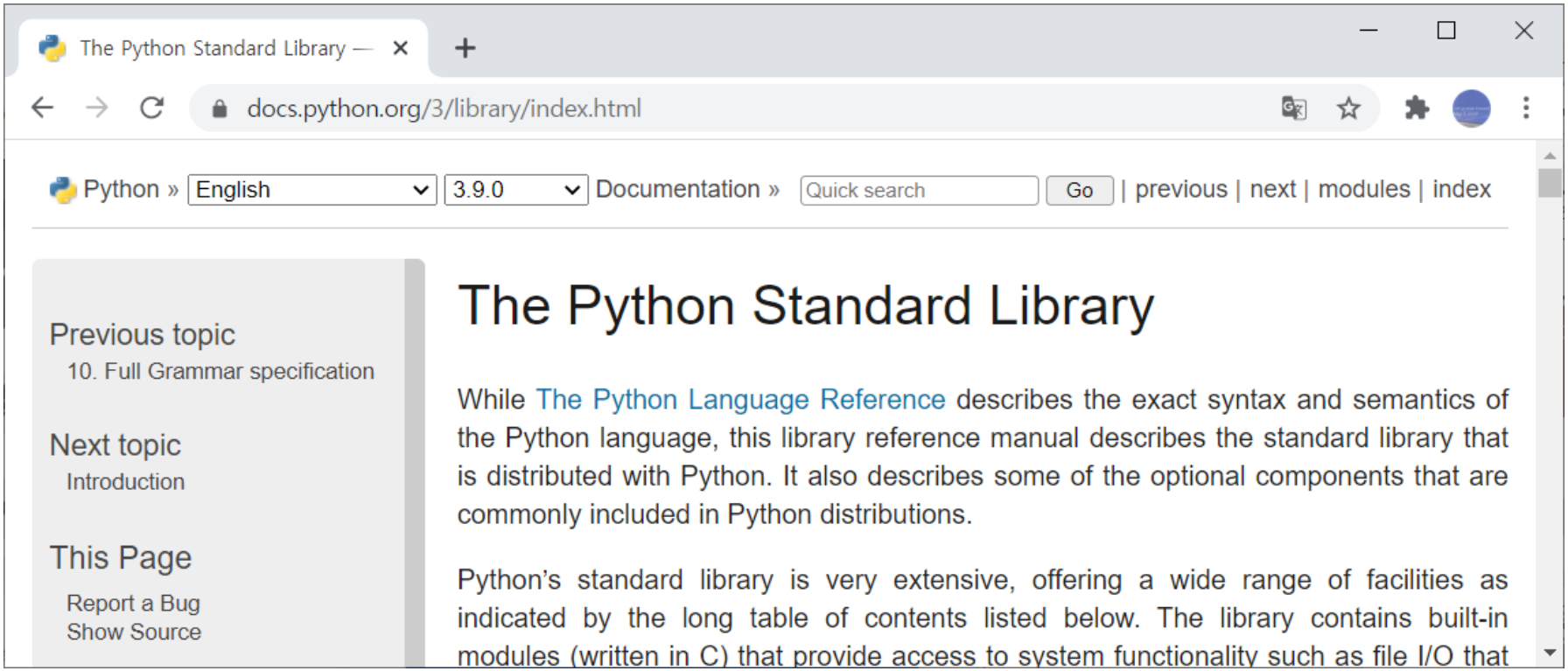
- whl 파일을 이용한 설치
  - python -m pip install whl파일명

- 패키지 관리
  - conda list 패키지명
  - conda remove 패키지명
  - pip show 패키지명
  - pip uninstall 패키명
  - help(패키지이름(.메서드이름))
  - dir(패키지이름(.메서드이름))
- 패키지 위치
  - C:₩Users₩사용자이름₩AppData₩Local₩Programs₩Python₩Lib₩site-packages나
  - Virtual Environment

4절. 파이썬 표준 라이브러리

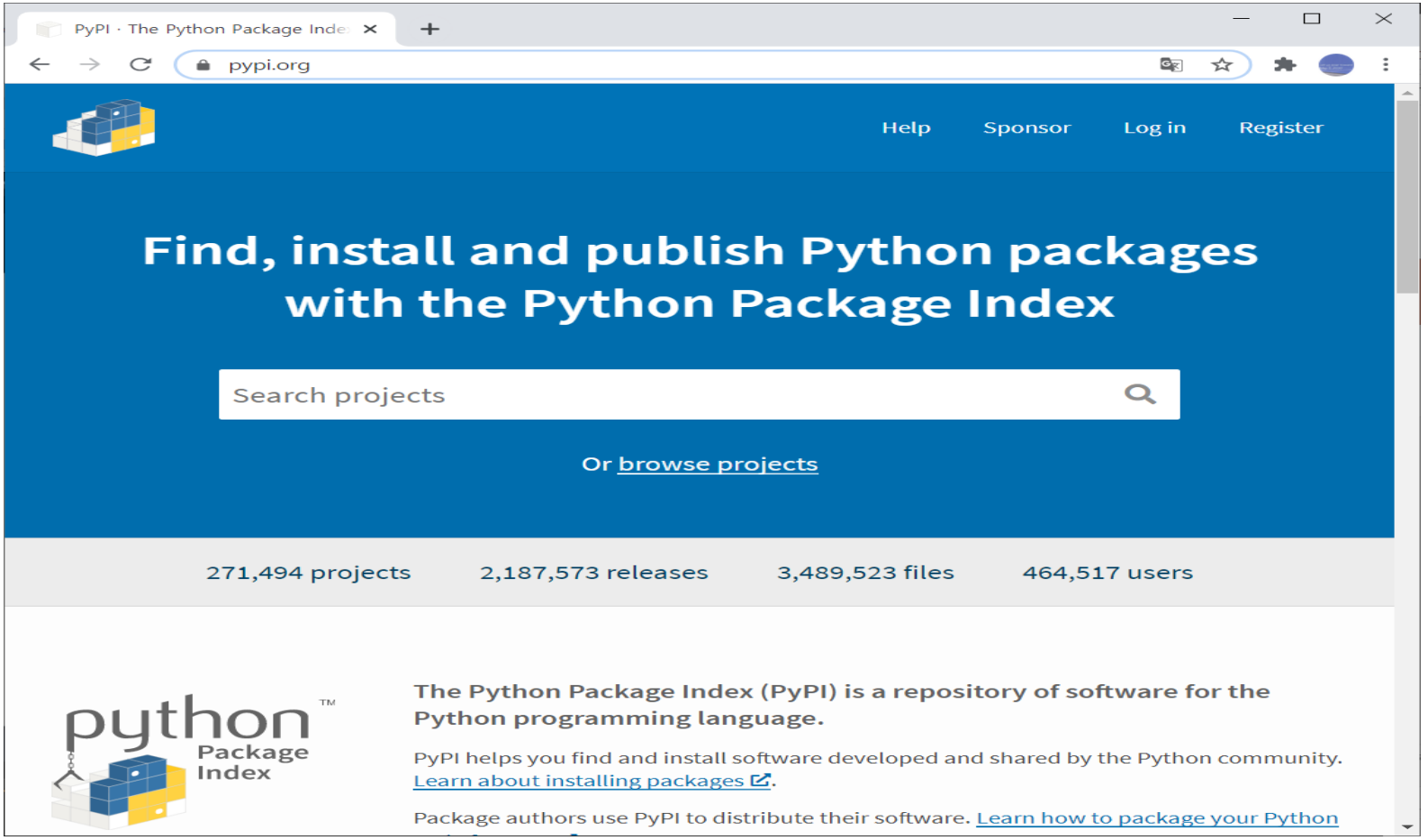
많은 문제를 해결하기 위한 표준화 된 솔루션을 제공

https://docs.python.org/3/library/index.html



많은 문제를 해결하기 위한 표준화 된 솔루션을 제공

https://pypi.org/



5절. 연습문제

1. 파이썬의 모듈과 패키지에 대해 잘 못 설명한 것은?

① 모듈은 파일 단위로 작성하며 확장자가 .py인 파이썬 파일이다.

② 패키지는 모듈들의 모음이며 디렉터리 단위로 존재한다.

③ 파이썬의 표준 라이브러리는 import 하지 않고 사용할 수 있다.

④ 일부 모듈 세트는 특정 플랫폼에서만 사용할 수 있다.

5절. 연습문제

2. 다음 중 모듈의 import와 import후 사용 방법이 잘 못 된 것은?

① import time
time.ctime()

② import time as t
t.ctime()

③ from time import ctime
ctime()

④ from time import ctime as ct
ctime()

5절. 연습문제

3. 파이썬이 디렉터리를 패키지에 포함하도록 처리하기 위해 필요한 파일은?

4. import *을 이용하면 패키지에 있는 서브 모듈을 찾아서 모두 가져옵니다. 그렇기 때문에 패키지 작성자가 패키지의 명시적 색인을 제공하기 위해 사용하는 속성의 이름은 무엇인가요?

5. 다음 중 파이썬 패키지를 설치하는 방법이 아닌 것은?
   ① python -m pip install --upgrade pip
   ② pip install 패키지명
   ③ conda install 패키지명
   ④ pip install 휠(whl)파일경로