

강의교안 이용 안내

- 본 강의교안의 저작권은 도서출판 홍릉에 있습니다.
- 불법한 PDF 파일이 사용되지 않도록 지도 바랍니다.
- 이 자료를 무단으로 전제하거나 배포할 경우 저작권법 136조에 의거하여 벌금에 처할 수 있고 이를 병과(併科)할 수도 있습니다.

단원 07

모듈과 패키지

인공지능소프트웨어학과

강환수 교수



Section 1. 모듈의 이해와 활용

-



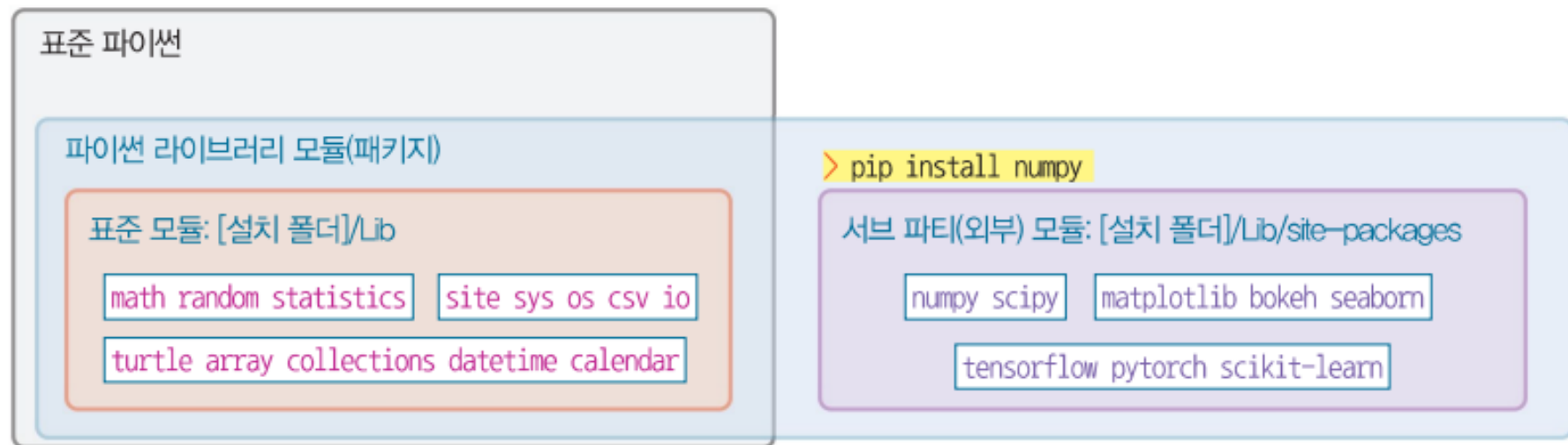
모듈

재사용 가능한 변수, 클래스, 함수 등이 정의된 파이썬 소스 파일

- **import 구문을 통해 다른 프로그램에서 불러와 사용**
 - `import math`
- **표준 모듈**
 - 파이썬에는 기본적으로 제공되는 표준 모듈(`math`, `random`, `sys`, `os` 등)
 - **sys 모듈 활용**
 - 파이썬 시스템 정보 접근에 사용되며, `sys.path`를 통해 모듈 검색 경로를 확인하고 변경
 - `sys.modules`를 통해 현재 메모리에 로드된 모듈 정보를 확인
- **외부 모듈(external module)**
 - 서드 파티 모듈
 - 설치 필요
 - 여러 회사나 전문가 등 외부에서 개발해 배포하는 모듈

표준 모듈과 서드 파티(외부) 모듈

- 표준 모듈은 따로 설치가 필요 없으나
- 서드 파티 모듈은 설치가 필요



▲ 그림 1 표준 모듈과 서드 파티(외부) 모듈

모듈 site

Python 설치 폴더와 사용자 지정 패키지 폴더 등 확인 및 관리

- 모듈 site를 불러와 함수 `getsitepackages()`
 - 현재 시스템의 파이썬 설치 폴더와 설치된 패키지 폴더를 확인

```
>>> import site
>>> site.getsitepackages()
['C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311',
 'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages']
```

모듈 sys

• 인터프리터와 상호작용할 수 있는 다양한 기능을 제공하는 표준 라이브러리 모듈

```
import sys
dir(sys)
```

```
sys.path
sys.platform
sys.version
sys.version_info
```

```
import sys
sys.stdlib_module_names
```

```
sys.builtin_module_names
```

```
sys.modules
sys.modules.keys()
```

```
import math
sys.modules['math']
import random
sys.modules['random']
```

- ❖ Python 표준 라이브러리(standard library)에 포함된 모듈들의 이름을 집합(set) 형태로 반환
- ❖ Python 설치 시 기본 제공되는 표준 라이브러리 모듈 목록을 조회
- ❖ 집합(set) 형태로 반환되므로 중복이 없으며, 정렬되지 않은 상태

- ❖ C로 구현된 Python 내장 모듈(built-in module)들의 이름을 튜플(tuple) 형태로 반환
- ❖ 튜플(tuple) 형태이므로 변경할 수 없으며, 정렬된 상태

- ❖ 현재 메모리에 로드된 모든 모듈을 저장하는 딕셔너리
- ❖ 키(Key): 모듈의 이름 (str)
- ❖ 값(Value): 해당 모듈 객체 (module object)
- ❖ import된 모듈이 sys.modules에 존재하는지 확인

- ❖ 교재 p320에서 math를 random으로 수정

```
>>> sys.modules['random']
<module 'random' from
'C:\Users\WWPC\AppData\Local\Programs\Python\Python312\Lib\random.py'>
```

사용자 정의 모듈

재사용 가능한 변수, 클래스, 함수 등이 정의된 파이썬 소스 파일

- 사용자 정의 모듈 생성 및 관리

- .py 확장자를 가지는 파이썬 파일로 모듈을 생성
- import 구문을 통해 다른 프로그램에서 사용 가능
- 모듈의 검색 경로 설정 및 `__name__` 변수를 통한 스크립트 실행 조건 제어

- 모듈화의 장점

- 모듈화는 코드 재사용성, 유지보수 용이성, 확장성 향상, 협업 개발 용이성 등 다양한 장점을 제공

직접 모듈을 작성하여 프로그램에서 가져오기

모듈은 확장자가 .py인 파이썬 파일

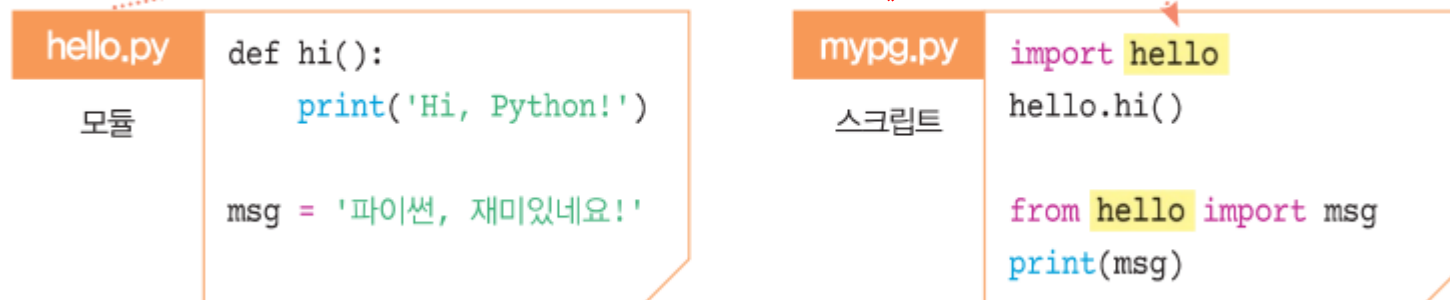
• 모듈 이름은 파일 이름과 동일

- 모듈에는 일련의 함수, 클래스 또는 변수가 정의되고 구현
- 모듈의 파일 이름이 식별자가 되므로 모듈 파일 이름에 공백이나 특수 문자를 넣지 말도록

• 다음 좌측 모듈 hello의 파이썬 소스를 원하는 폴더에 저장

- 폴더 'D:\Wpycode\Wch07'에 저장한다고 가정
- 모듈 이름을 hello로 하려면 소스 이름은 hello.py로 저장

- ❖ 파이썬 소스가 일반적인 프로그램으로 실행되는 경우, 이 파일을 스크립트(script)라고 부름
- ❖ 특히 스크립트(script)라는 표현은 모듈로서의 역할의 반대 개념



▲ 그림 2 모듈 hello.py와 모듈을 불러 사용하는 프로그램 mypg.py

셸에서 자신이 만든 모듈을 실행

- 셸에서 자신이 만든 모듈 `hello`를 `import`하면 모듈을 찾지 못하고 오류가 발생
 - 상위 폴더인 폴더 'D:\pycode'에서 셸을 실행해 `import hello`에서의 오류 발생

❖ 교재 p322의 D:\pycode\ch07에서 D:\pycode로 수정

```
D:\pycode\ch07>cd ..
```

```
D:\pycode>python
```

```
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)] on win32
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import hello
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ModuleNotFoundError: No module named 'hello'
```

```
>>> █
```

모듈을 가져오는 첫 번째 방법

셸에서 자신이 만든 모듈을 실행

- 서드 파티 모듈이 설치되는 폴더 site-packages에 자신이 만든 모듈을 복사해 놓는 방법

- 모듈 site의 함수 getsitepackages()로 site-packages의 폴더 확인

```
>>> import site
```

```
>>> site.getsitepackages()
```

```
['C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311',  
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages']
```

- 자신이 만든 모듈 hello.py를 다음 폴더에 복사
 - 폴더 파이썬 설치 폴더 하부 Lib\site-packages 폴더
- 이제 셸에서 실행 가능

```
>>> import hello
```

```
>>> hello.hi()
```

```
Hi, Python!
```

```
>>> hello.msg
```

```
'파이썬, 재미있네요!'
```

두 번째 방법

셸에서 자신이 만든 모듈을 실행

- 파이썬의 모듈 경로인 `sys.path`에 `append()` 메소드로 추가

- `hello.py`가 저장된 폴더인 `'D:\pycode\ch07'`
- 문자열 앞에 `raw`를 의미하는 `r`을 붙이면 역슬래시 문자를 문자 그대로 처리

```
>>> import sys
```

```
>>>
```

```
>>> sys.path.append(r'D:\pycode\ch07')
```

raw string 처리 방법인 r'string'은 특수 문자를 이스케이프
하지 않고 문자열을 표현하는 방법이다.

❖ Raw string 대신에 `\\`를 사용 가능

```
>>> sys.path
```

```
['', 'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311\\python311.zip',  
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311\\DLLs', 'C:\\Users\\  
PC\\AppData\\Local\\Programs\\Python\\Python311\\Lib', 'C:\\Users\\PC\\AppData\\  
Local\\Programs\\Python\\Python311', <중략> ... 'C:\\Users\\PC\\AppData\\Local\\  
Programs\\Python\\Python311\\Lib\\site-packages', 'D:\\pycode\\ch07']
```

```
>>>
```

```
>>> import hello
```

```
>>> hello.hi()
```

```
Hi, Python!
```

```
>>> hello.msg
```

```
'파이썬, 재미있네요!'
```

모듈의 스크립트 실행 방법

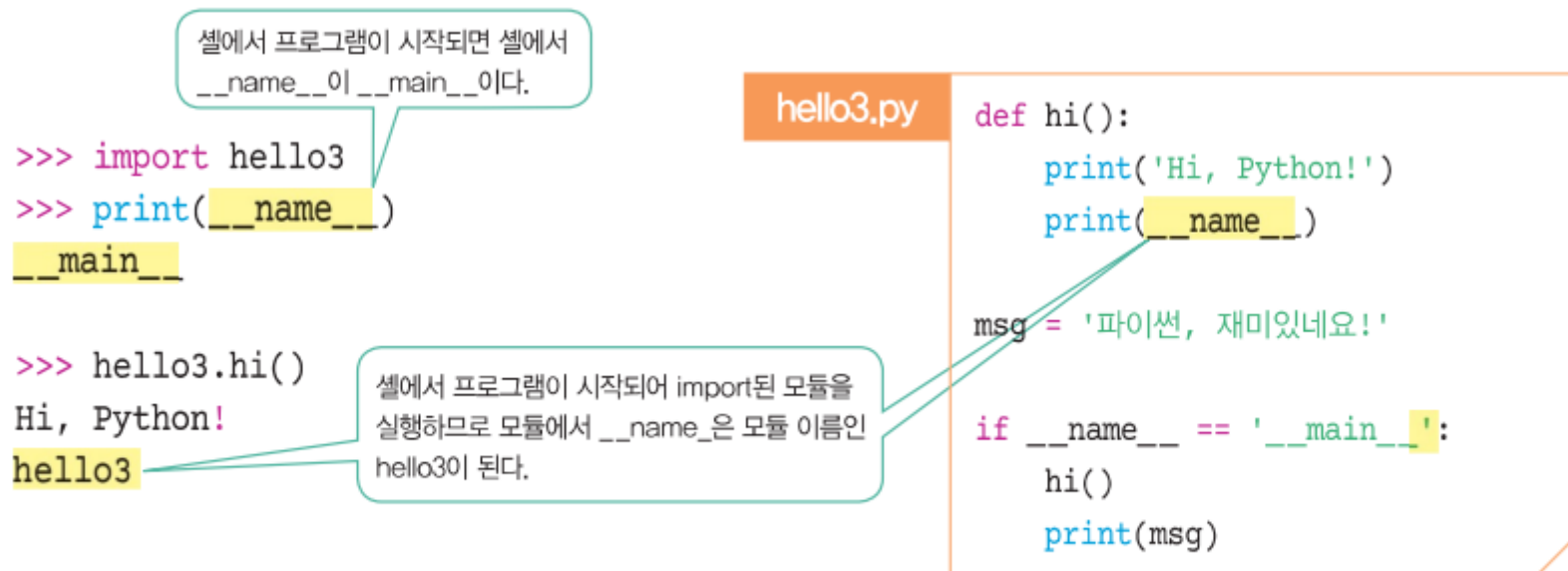
파이썬 소스가 일반 프로그램으로 실행되는 경우, 이 파일을 스크립트(script)라고 부름

- 문장 `import`로는 실행되지 않도록 하는 방법
 - 특수 변수 `__name__` 사용
 - 실행되는 모듈의 이름이 저장되는 변수
 - 모듈 자체에서 시작되어 실행되면 `__main__` 이 저장됨
 - 다른 곳에서 호출이 된다면 자신의 모듈 이름이 저장
 - 모듈 소스 파일 코드에 다음을 추가
 - `if __name__ == '__main__':`
 - 소스 파일 자체를 실행한다면 True
 - 위 if 블록이 실행됨
 - 소스 파일을 모듈로 이용해 import 한다면 False
 - 위 if 블록이 실행되지 않음

모듈에서 변수 `__name__`의 적절한 사용

• 소스 `hello3.py` 실행 예

- `python hello3.py`
 - 소스 `hello3.py`가 직접 실행되는 경우
 - `__name__` 내장 변수에는 `__main__`으로 설정
- `import hello3`
 - `hello3.py`가 다른 스크립트에 의해 `import` 되었을 경우
 - `__name__`은 모듈 이름인 `hello3`가 저장



▲ 그림 7 모듈에서 변수 `__name__`의 적절한 사용

7-1 7-2 예제 코딩

- 모듈 `todolist`

- 할 일 목록 `tasks`를 만들고 목록에 항목을 추가하며, 현재 할 일 목록을 출력하는 함수를 제공

- 스크립트 `todomain.py`

- 모듈 `todolist`에서 함수 `create_task`, `list_tasks`와 리스트 객체 `tasks`를 불러와 할 일 목록을 관리하는 스크립트 코드

```
# todolist.py
def create_task(task_list, task_name):
    task_list.append(task_name)

def list_tasks(task_list):
    for index, task in enumerate(task_list, start=1):
        print(f"{index}. {task}")

tasks = []
```

```
# todomain.py
from todolist import create_task, list_tasks, tasks

# 할 일 목록 작업
create_task(tasks, "청소하기")
create_task(tasks, "식료품 쇼핑하기")
create_task(tasks, "운동하기")

print("할 일 목록:")
list_tasks(tasks)
```

모듈화 프로그래밍(modular programming)

프로그램을 적절한 단위의 모듈로 나누어 코딩하는 방식

- 파이썬에서 모듈화 프로그래밍을 도와주는 방법

- 함수(functions)
- 모듈(modules)
- 패키지(packages)

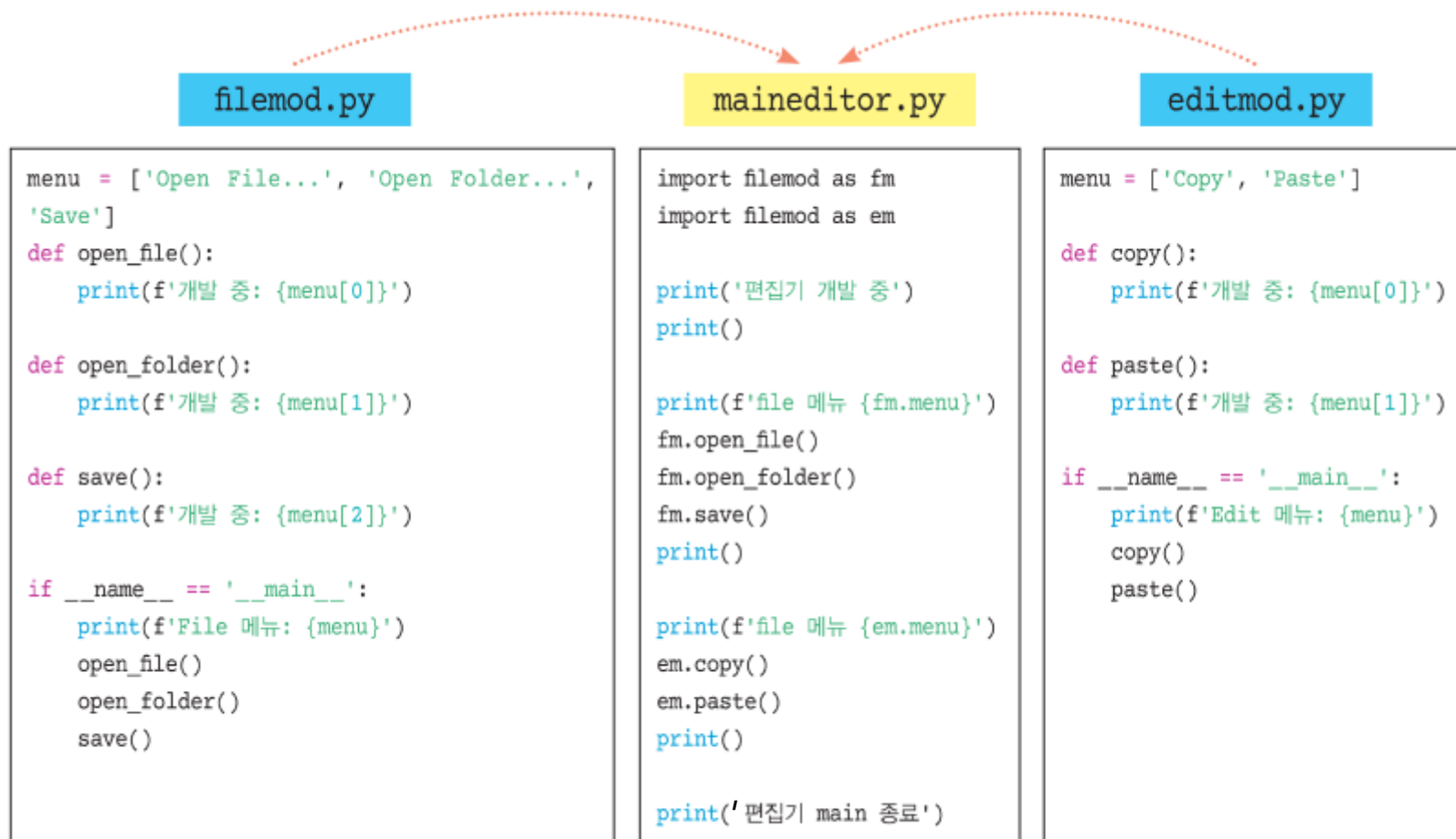
- 모듈화 장점

- 코드의 재사용성이 높아지고 유지 보수가 용이해짐
 - 단순성(simplicity):
 - 유지관리(maintainability):
 - 재사용(reusability)
 - 범위 지정(scoping)
 - 모듈은 일반적으로 별도의 네임스페이스를 정의하므로 프로그램의 여러 영역에 있는 식별자 간의 충돌을 피할 수 있음

편집기 모의 개발을 위한 모듈 작성

모듈화 프로그래밍의 개념을 사용하여 코드를 구성하고 여러 모듈을 가져와서 사용하는 예제

- 편집기(editor)를 모의로 개발하기 위해 다음의 3개의 파일로 나누어 개발한다고 가정



▲ 그림 10 두 개의 모듈 filemode, editmod와 maineditor 스크립트 코드

Section 2. 패키지의 이해와 활용

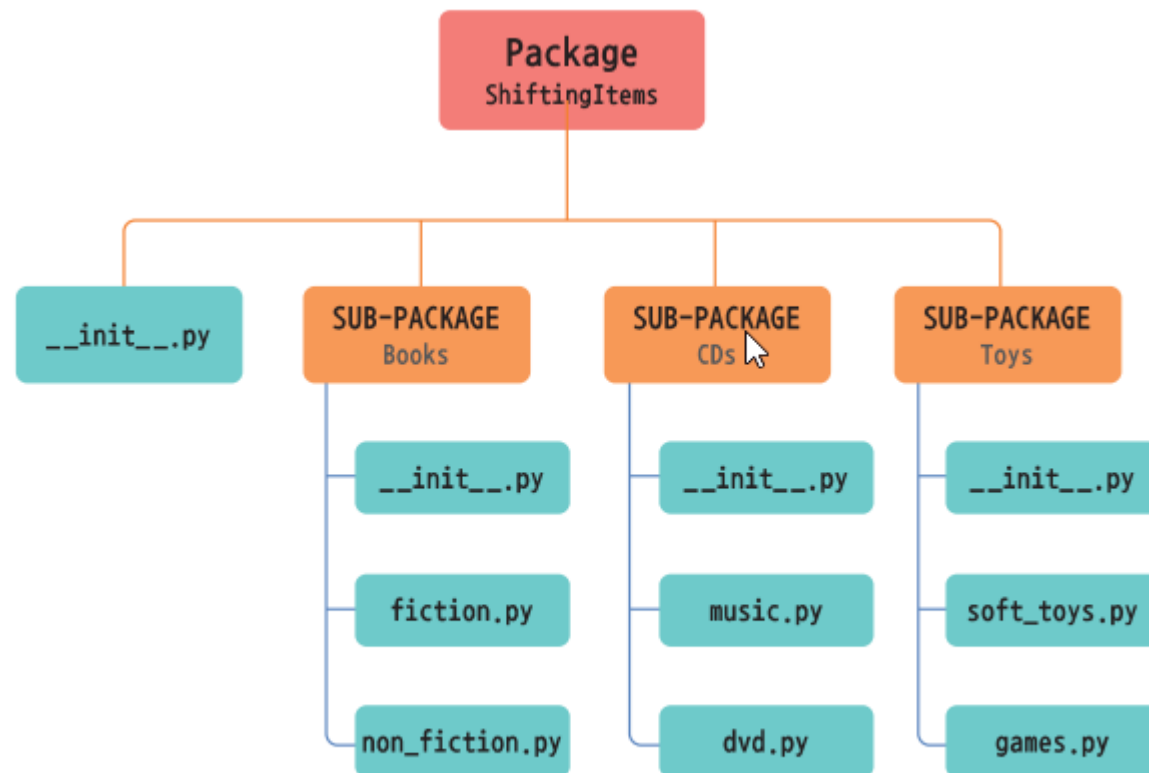
-



패키지(package)

- 패키지(package)

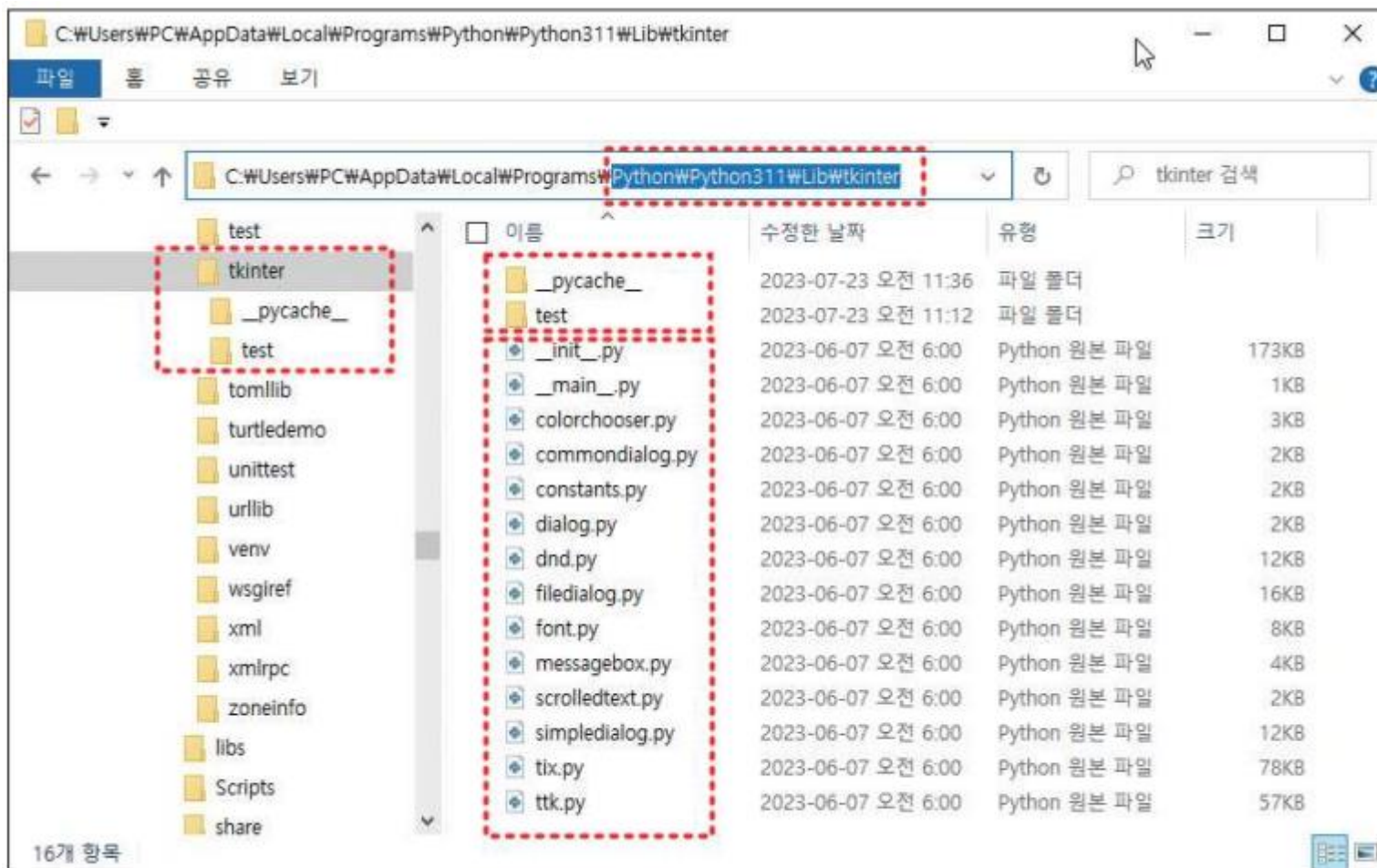
- 패키지 이름의 폴더에 여러 모듈을 저장하고 다시 하부 폴더로 나누어 관련 모듈을 저장하는 계층적 파일 구조를 사용하는 파이썬 라이브러리



▲ 그림 11 파이썬의 패키지 개념

패키지 tkinter

- 패키지 이름인 tkinter가 바로 모듈을 저장된 폴더 이름
 - 하부의 colorchooser.py 등 여러 모듈과 하부 폴더가 있음



▲ 그림 12 파이썬 패키지 tkinter

패키지 myai

직접 만들어보는 패키지 myai

- 기준 폴더 D:/pycode/ch07 하부에 폴더 myai를 만들고 하부에 다시 폴더와 모듈을 구성

패키지 이름: myai		패키지 폴더: D:/pycode/ch07/myai
패키지 계층 구조	모듈	소스 코드
myai	ai.py	<pre># %% 모듈 파일 myai\ai.py def getAI(): print(msg) msg = '저는 인공지능 모듈입니다.'</pre>
	gpt.py	<pre># %% 모듈 파일 myai\gpt.py def getGPT(): print(msg) msg = '저는 생성형 모델 인공지능 모듈입니다.'</pre>
myai/ml	machine.py	<pre># %% 모듈 파일 myai\ml\machine.py def getML(): print(msg) msg = '저는 머신러닝 모듈입니다.'</pre>
myai/nn	neural.py	<pre># %% 모듈 파일 myai\nn\neural.py def getANN(): print(msg) msg = '저는 뉴럴 네트워크 모듈입니다.'</pre>

패키지 myai 활용

- 먼저 상위 폴더인 r'D:\pycode\ch07'를 path에 추가

```
>>> import sys
>>> sys.path.append(r'D:\pycode\ch07')
>>> sys.path
['', 'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311\\python311.
zip', 'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311\\DLLs', <중략> ...
'C:\\Users\\PC\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages',
'D:\\pycode\\ch07']
```

- import 구문

```
>>> import myai.ai
>>> myai.ai.getAI()
저는 인공지능 모듈입니다.
```

다양한 import 구문

형식과 예문 1	<code>import module1[, modules2, ...]</code>	<code>import math, random</code>
설명	<ul style="list-style-type: none"> 모듈 사용 구문으로, 콤마로 나열하여 여러 모듈을 한 번에 불러오기가 가능 모듈은 모듈 자체이거나 패키지에 속한 모듈일 수 있음. <code>module:= package.module</code> 참조는 <code>module1.obj</code>로 가능하며 바로 소속 객체인 <code>obj</code>만으로는 참조 불가능 	
형식과 예문 2	<code>import module as alias_name</code>	<code>import math as m</code> <code>import random as rd</code>
설명	<ul style="list-style-type: none"> 모듈을 별칭인 <code>alias_name</code> 이름으로 지정 참조는 <code>alias_name.obj</code>로 가능 	
형식과 예문 3	<code>from module import mod_obj1[, mod_obj2, ...]</code> <code>from module import *</code>	<code>from math import gcd, trunc</code> <code>from math import *</code>
설명	<ul style="list-style-type: none"> 모듈에서 지정된 이름을 모듈 이름 없이 사용 가능. 즉, <code>mod_obj1</code>을 바로 사용 가능 <code>import *</code>는 모듈에 속한 모든 객체나 변수 등의 이름 사용 가능 <code>import *</code>는 많은 타이핑 없이 모듈이 제공되므로 편리하지만, 실제 프로젝트 코드에서 권장되지 않음. <code>import *</code>는 함수 내부에서는 사용 불가능 	
형식과 예문 4	<code>from module import mod_obj1 as alias_name1[, mod_obj2 as alias_name2, ...]</code>	<code>from math import sqrt as sr, truck as tc</code>
설명	<ul style="list-style-type: none"> 모듈에서 지정된 이름인 <code>mod_obj</code>를 다른 이름인 <code>alias_name</code>으로 사용 이전에 존재하는 이름과의 충돌을 피할 수 있음. 	

패키지 myai의 하부 ml 폴더의 모듈 machine를 불러오기

- `from myai.ml import machine` 구문
 - `machine.getML()`로 호출이 가능

```
>>> from myai.ml import machine
>>> machine.getML()
저는 머신러닝 모듈입니다.
```

- myai 폴더의 하부 폴더 nn에 있는 모듈 neural도 `myai.nn.neural`로 표현

```
>>> from myai.nn.neural import getANN
>>> getANN()
저는 뉴럴 네트워크 모듈입니다.
```


파이썬 표준 패키지 collections

여러 특별한 컨테이너 데이터 유형을 제공

- **defaultdict, namedtuple, counter, ordereddict 등의 클래스를 제공**
 - 표준 자료형인 dict, list, str, tuple 등과 유사하게 동작하면서 문제를 해결할 수 있는 효율적인 기능
- **패키지 collections 폴더와 파일**
 - 파일은 `__init__.py`와 `abc.py` 두 개 뿐
 - 대부분 코드가 `__init__.py`에 저장
 - 예전에는 폴더가 패키지임을 나타내는 파일이 `__init__.py`
 - 파이썬 버전 3.3 이후 현재, 패키지 폴더에 파일 `__init__.py`의 존재는 선택적
 - 그럼에도 이전부터 제공하던 패키지는 대부분 `__init__.py`이 존재



Collections in Python

클래스 Counter 활용

전체 목록에서 원소의 개수를 세어 딕셔너리 형태로 저장하는 컨테이너

- **collections 패키지의 Counter 클래스**
 - 리스트, 문자열 등의 시퀀스에 포함된 요소들의 개수를 쉽게 세고 관리

```
>>> from collections import Counter
>>> lst = [1, 2, 3, 1, 2, 1, 1, 3, 4, 5]
>>> dgt_counter = Counter(lst)
>>> dgt_counter
Counter({1: 4, 2: 2, 3: 2, 4: 1, 5: 1})
>>> # 요소 1의 개수 확인
>>>
>>> count_of_ones = dgt_counter[1]
>>> print(f"요소 1의 개수: {count_of_ones}")
요소 1의 개수: 4
>>> # 가장 많이 등장하는 요소와 개수
>>>
>>> most_common_element = dgt_counter.most_common(1)
>>> print(f"가장 많이 등장하는 요소와 개수: {most_common_element}")
가장 많이 등장하는 요소와 개수: [(1, 4)]
>>> # 상위 2개 요소와 개수
>>>
>>> top_two_elements = dgt_counter.most_common(2)
>>> print(f"상위 2개 요소와 개수: {top_two_elements}")
상위 2개 요소와 개수: [(1, 4), (2, 2)]
□
```

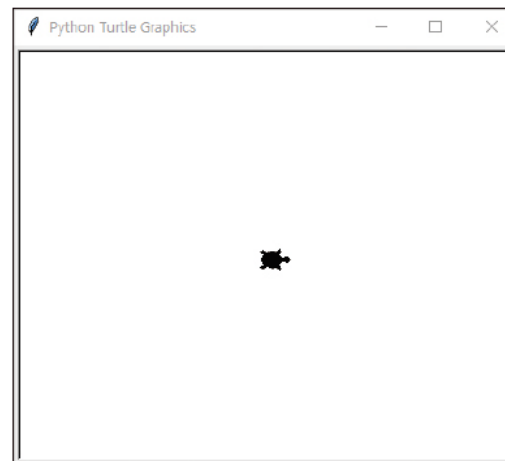
Section 3. 표준 모듈 터틀과 서드 파티 패키지 numpy, matplotlib



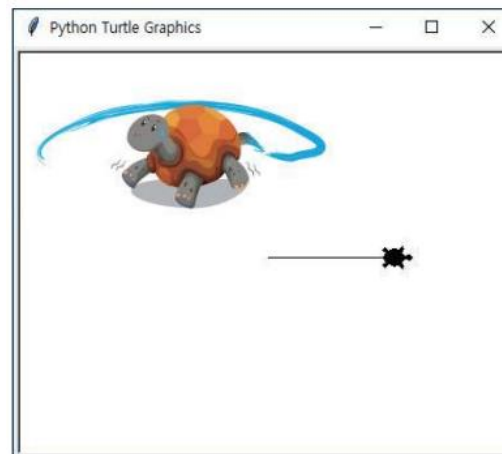
모듈 turtle 개요와 기본 명령

- 1967년 아동 교육용으로 개발된 로고(logo)라는 프로그래밍 언어의 일부

```
>>> import turtle as t  
>>> t.shape('turtle')
```



```
>>> t.forward(100)
```

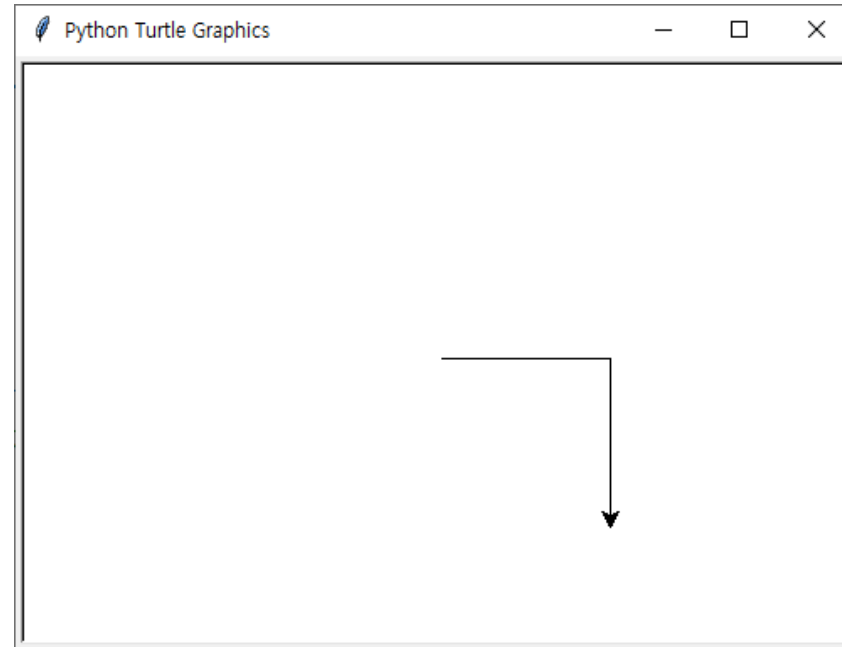


기초 코드

- 중앙 좌표가 0, 0

```
import turtle as t
t.shape('turtle')

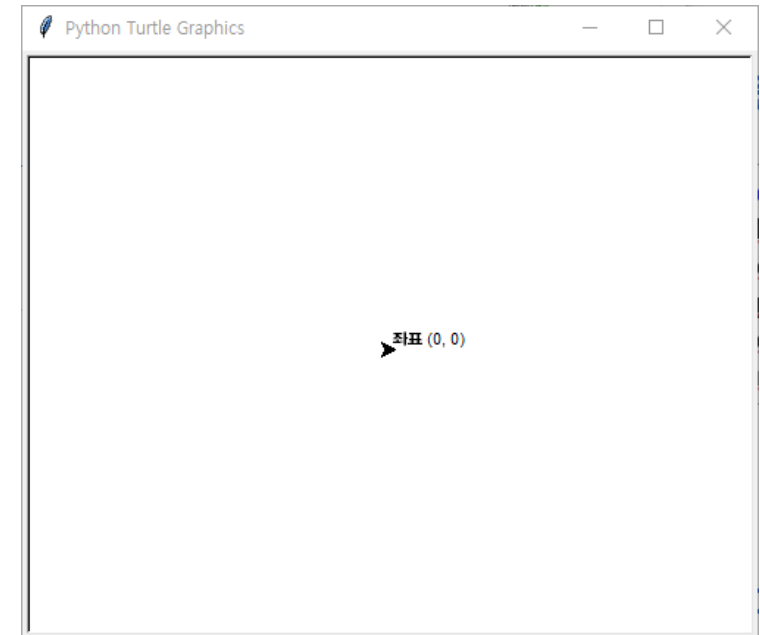
t.forward(100)
t.right(90)
t.goto(100, -100)
```



윈도 크기와 선 그리기

- 터틀 윈도 크기와 터틀 모양 수정

```
import turtle as t
t.shape('classic')
t.setup(500, 400) #초기 윈도의 크기 조정
t.speed(1) #1에서 10까지 거북이 속도 증가, 0이면 최고속
t.home() #기본(classic 모양) 모양으로 (0,0)에 위치
t.write('좌표 (0, 0)') #글씨 쓰기
```

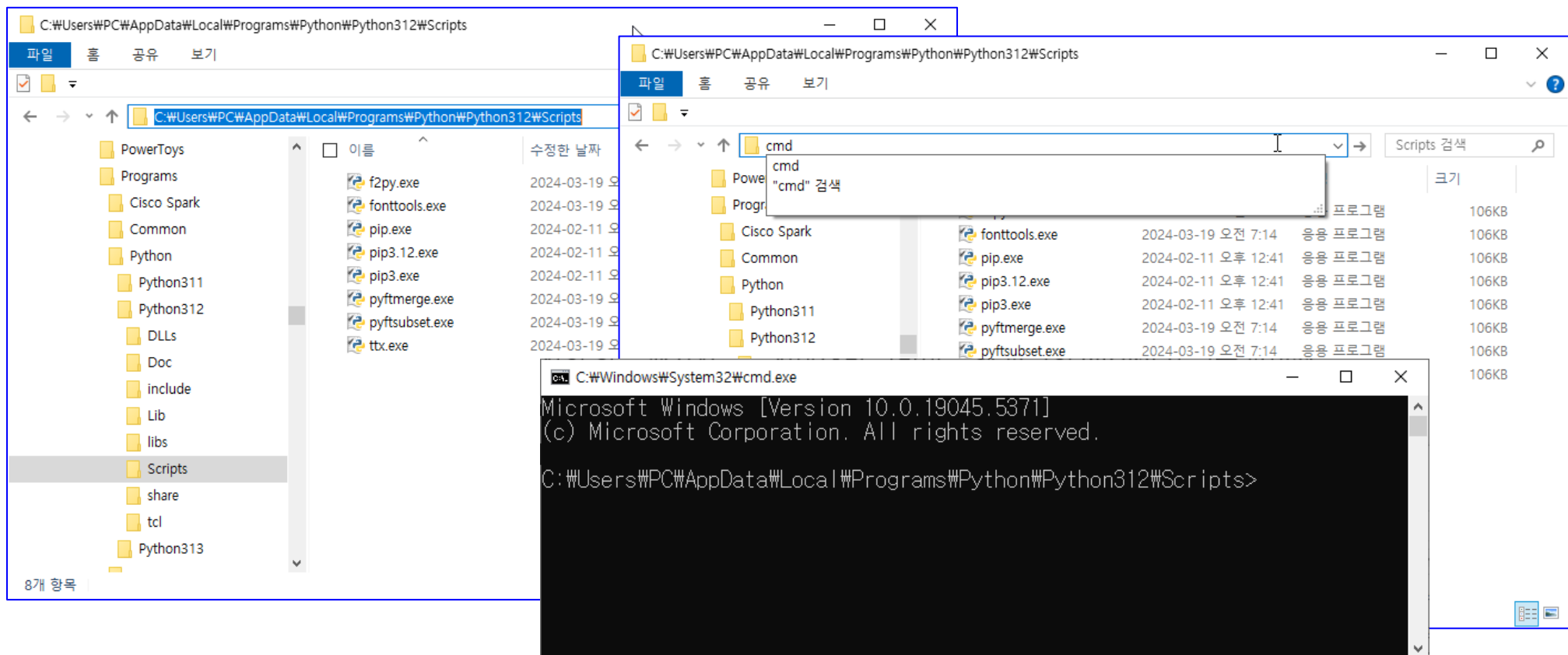


- 선을 그리지 않는 penup()과 선을 그리는 pendown()

명령어 입력을 위한 셸이나 cmd 열기

• 준비

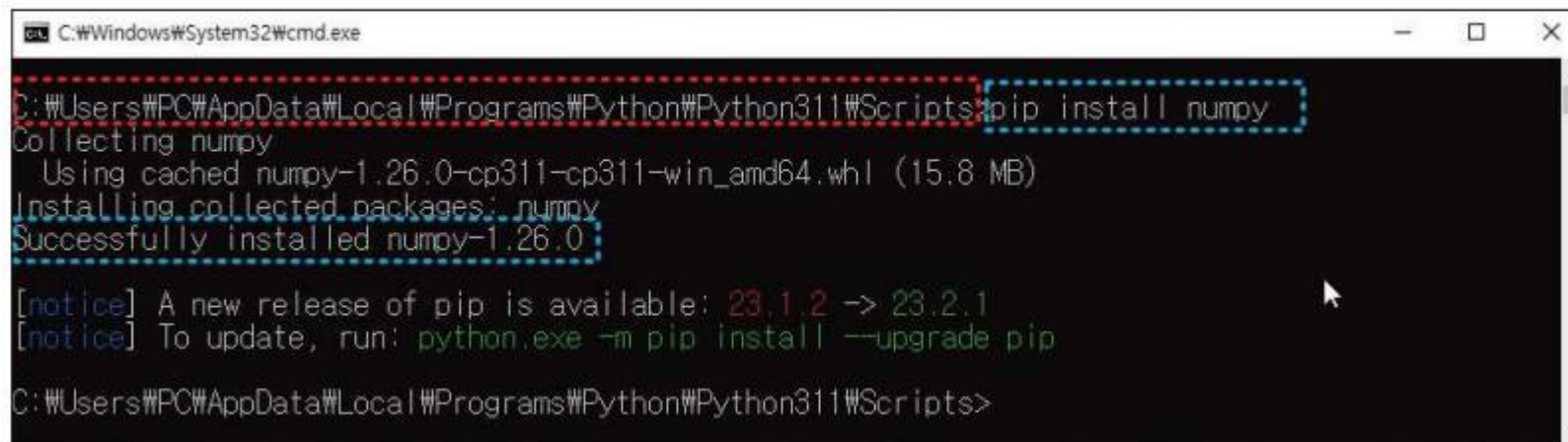
- 여러 파이썬 버전이 설치되어 있다면
 - **현재의 윈도우 path 설정에 유의**
- 설치하려는 버전의 pip.exe가 있는 폴더에서 명령 프롬프트를 열어 설치하면 가장 확실



서드 파티 패키지 설치

- 서드 파티(third party, 외부) 패키지를 설치해 사용
 - 파이썬 패키지 색인(PyPI: Python Package Index) 페이지(<https://pypi.org>)
 - 공식적인 서드 파티 패키지 라이브러리를 제공
 - PyPI는 구글플레이나 앱스토어와 비슷하나 무료
- pip(python install package) 명령어
 - 한 줄로 쉽게 원하는 패키지의 설치가 가능

```
pip install numpy
```



```
C:\Windows\System32\cmd.exe
C:\Users\PC\AppData\Local\Programs\Python\Python311\Scripts>pip install numpy
Collecting numpy
  Using cached numpy-1.26.0-cp311-cp311-win_amd64.whl (15.8 MB)
Installing collected packages: numpy
Successfully installed numpy-1.26.0

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
C:\Users\PC\AppData\Local\Programs\Python\Python311\Scripts>
```

▲ 그림 16 패키지 NumPy 설치

수학용 라이브러리 numpy 활용

- NumPy의 자료형 ndarray

- 수학의 행렬(matrix)과 같은 다차원 자료의 나열을 지원하는 자료형

```
>>> import numpy as np
>>> data = np.array([1, 2, 3, 4, 5])
>>> data
array([1, 2, 3, 4, 5])
```

```
>>> print(data)
[1 2 3 4 5]
>>> data.shape
(5,)
>>> data.ndim
1
```

속성 shape의 결과는 튜플로, 원소가 한 개이면 1차원을 의미하며, 원소 값 5는 원소의 수를 말한다.

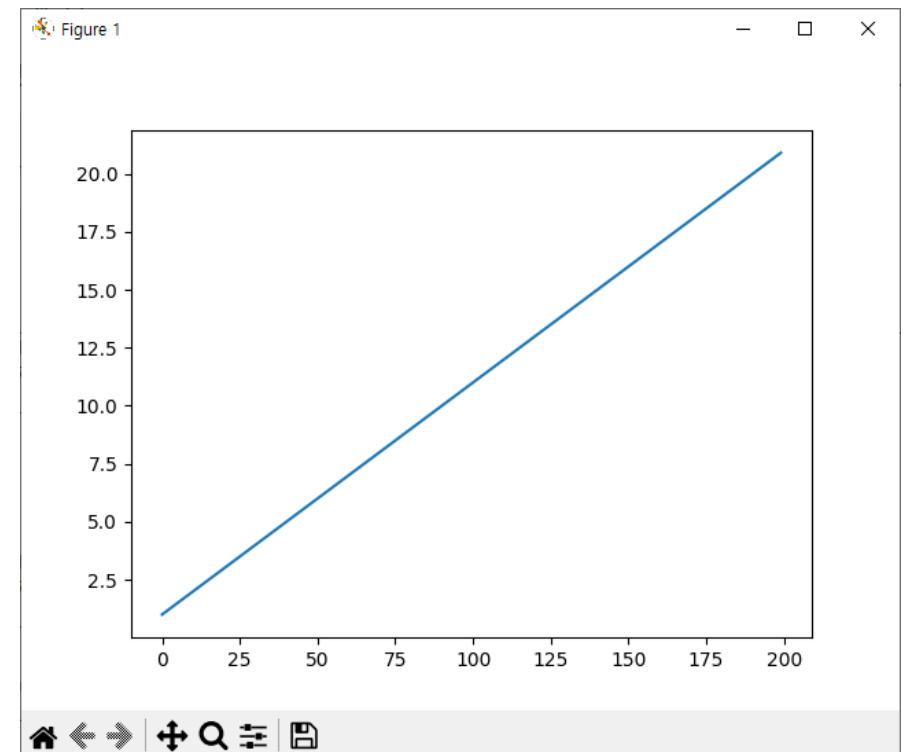
데이터 시각화 Matplotlib 활용

설치 `pip install matplotlib` 필요

• 데이터 시각화 Matplotlib 개요

- 데이터 시각화에 많이 활용되는 패키지 중의 하나가 Matplotlib
- 라인 플롯, 바 차트, 파이 차트, 히스토그램, 박스 플롯, 스캐터 플롯 등을 비롯하여 다양한 차트와 플롯 스타일을 지원

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> y = np.arange(1, 21, 0.1)
>>> plt.plot(y)
[<matplotlib.lines.Line2D object at 0x000002390AAEB390>]
>>> plt.show()
```



NumPy의 난수를 활용

- `np.random.randn(5)`
 - 표준 정규분포(평균: 0, 표준편차: 1)의 난수 5개 반환

```
>>> import numpy as np
>>> arr = np.random.randn(5) # 표준 정규분포 난수 5개
>>> arr
```

```
array([ 1.7478767 ,  0.6387504 ,  0.83625316, -0.170917 , -1.64782038])
```

또한 메서드 `cumsum(arr)`으로 처음부터 합산된 누적 값 5개를 얻을 수 있다.

```
>>> arr_cs = np.cumsum(arr) # 누적 값
>>> arr_cs
```

```
array([1.7478767 , 2.38662711, 3.22288027, 3.05196327, 1.40414289])
```

이전 3개의 수의 합이 3.22288027