

단원 02

인공지능소프트웨어학과

파이썬 기초와 자료구조

강환수 교수

DONGYANG MIRAE UNIVERSITY
Dept. of Artificial Intelligence



2.1 파이썬 언어 기초

-



주석과 상수

- 주석(comments)
 - # 이후부터 줄 마지막까지 주석
- 상수(literals)
 - 수(정수, 실수), 문자열 표시
- 오류 발생 표시

```
>>> 7L
File "<string>", line 1
  7L
  ^
SyntaxError: invalid decimal literal
```

콘솔 표시와 오류가 발생한 줄 번호

오류가 발생한 부분과 이 위치를 알리는 ^

오류 이름

오류 메시지

```
>>> -5 4.5
File "<string>", line 1
  -5 4.5
    ^^^
SyntaxError: invalid syntax
```

- 튜플(tuple)
 - 상수를 쉼표(또는 콤마) ,로 구분해 나열하면 괄호를 둘러싼 수의 나열이 표시

```
>>> 10, -3
(10, -3)
```

변수와 대입

- 다양하게 변하는 값을 저장하는 변수(variables) a에 값 3을 저장해 출력
 - 변수에 값을 저장하려면 대입 연산자 =을 사용

```
>>> # variable
>>> a = 3 # assign value to variable a
>>> a
3
```

- 다중 대입 지원

```
>>> m, n = 10, 20
>>> print(m, n)
10 20
```

- m과 n의 값을 서로 교환

```
>>> m, n = n, m
>>> print(m, n)
20 10
```

변수 이름 조건

- 식별자(identifiers)

- 프로그래밍 언어에서 프로그래머가 직접 만드는 변수나 함수 이름

- 식별자의 구성 조건

- 영어 알파벳, 숫자, _(밑줄)을 사용하며 알파벳 대문자와 소문자는 구별
- 첫 문자는 반드시 알파벳이나 밑줄(_)로 시작해야 하며 첫 문자로 숫자는 올 수 없음
- 마침표 .를 사용할 수 없음
 - 객체지향 언어인 파이썬은 마침표 .를 연산자로 사용하기 때문
- 문법에 사용되는 예약어인 키워드는 사용 불가

- 조건 정리

- 영어 알파벳, 숫자, _(밑줄)을 사용
- 첫 글자는 반드시 알파벳이나 밑줄(_)로 시작
- 다양한 특수 문자(% , \$, # 등)를 사용할 수 없음
- 문법에 사용하는 if while 등의 키워드는 사용할 수 없음

```
>>> # var name
>>> age = 19
>>> age
19
>>> my_age = 20
>>> my_age
20
>>> _height = 172.3
>>> _height
172.3
```

파이썬 기본 자료형

- 자료형(data types)
 - 프로그래밍 언어에서 자료의 종류

[표 2-1] 파이썬 다양한 기본 자료형

자료형	설명	예시	비고
float	실수	-3.89, 4.26	
int	정수	10, 15, -5	
complex	복소수	3 -4j, -10+7j	소수부에서 j 사용
string	문자 하나 또는 여러 개의 문자열	'Python', "VS code"	작은 따옴표와 큰 따옴표 모두 가능
bool	논리 자료	True, False	대소문자

자료형 코드

```
# int
a = 10
print(a)
type(a)
```

[4] ✓ 0.0s Python

... 10

... int


```
# float
x = 2.718
print(x)
type(x)
```

[2] ✓ 0.0s Python

... 2.718

... float


```
# complex
c = 3 - 4j
print(c)
type(c)
```

[3] ✓ 0.0s Python

... (3-4j)

... complex

```
# string
s = "VS code"
print(s)
type(s)
```

[5] ✓ 0.0s Python

... VS code

... str


```
# logical
b = True
print(b)
type(b)
```

[17] Python

... True

... bool

파이썬의 날짜 자료형

기본 자료형이 아닌 자료형도 매우 다양

- 파이썬에서 날짜 정보는 기본 자료형이 아님

- 코딩을 위해 먼저 모듈 datetime을 메모리에 로딩(load)
- 구문 import datetime as dt를 사용
 - 여기서 dt를 별명(alias)
 - 모듈의 이름 datetime을 짧게 dt로 만들어서 코드를 간결하게 만들기 위해서 사용

• 코드 dt.datetime.now(): 모듈 dt에 속해 있는 클래스 datetime의 메소드 now()를 호출

```
>>> import datetime as dt
>>> c = dt.datetime.now()
>>> print(c.year, c.month, c.day)
2025 5 3
>>> type(c)
<class 'datetime.datetime'>
```

메서드 datetime.now()는 현재 시스템 시간을 반환한다. 현재 날짜와 시간을 얻어서 c라는 변수에 저장한다.

변수 c에 저장된 현재 날짜와 시간에서 연도(year), 월(month), 일(day)을 출력한다.

- dt.datetime.now()

- 시스템의 현재 시간 정보를 생성해 정보를 출력
- 모듈 datetime의 datetime 클래스 사용
 - datetime.datetime
- 마침표 .
 - '~에 속해 있는'의 연산자로 사용
 - 식별자의 문자로 사용할 수 없음

지정한 날짜와 시간 객체 생성

- `dt.datetime(2026, 3, 1)` 생성자를 사용
 - 2026-03-01 날짜 정보

```
>>> import datetime as dt
>>> d = dt.datetime(2026, 3, 1)
>>> print(d, '요일:', d.weekday())
2026-03-01 00:00:00 요일: 6
```

- 변수 `d`에 저장된 날짜와 시간을 출력하고, 그 날짜의 요일을 `d.weekday()`로 출력
 - 메서드 `weekday()`는 날짜가 무슨 요일인지를 숫자로 반환
 - 0은 월요일, 1은 화요일, ..., 6은 일요일에 해당
 - 2026년 3월 1일이 일요일

다양한 연산자

[표 2-2] 파이썬 주요 연산자

분류	연산자	비고
사칙연산	+ - * /	R 언어와 일치
나머지 연산자(modulus)	%	R 언어, %%
정수 나누기, 몫 연산자(integer division)	//	R 언어, %/%
지수 연산자	**	R 언어, ** ^
관계 연산자	> < >= <= == !=	R 언어와 일치
논리 연산자	& and or not	R 언어, & && !
대입 연산자	=	파이썬과 자바 등의 대부분의 언어에서는 =을 사용하며, R은 대부분 <-을 사용하며, R 함수 선언에서 패러미터 초기 지정은 =을 사용

내장 함수

• 내장 함수

- 파이썬을 설치하고 추가 패키지 설치 없이 사용 가능한 함수
- input(), print(), range() 등 매우 다양

[표 2-3] 파이썬 주요 함수

분류	연산자
표준입력과 표준출력	input() print()
자료형 반환과 변환	type() int() float() str() bin() oct() hex()
자료구조 생성	list() tuple() set() dict()
시퀀스(문자열, 리스트, 튜플 등)의 길이를 반환	len()
숫자의 시퀀스 생성	range()
기초 통계 함수	min() max() abs() pow()
올림 반올림 함수	round()

도움말 함수

```
>>> help(print)
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
        string inserted between values, default a space.
    end
        string appended after the last value, default a newline.
    file
        a file-like object (stream); defaults to the current sys.stdout.
    flush
        whether to forcibly flush the stream.
```

함수 input() 활용

사용자로부터 키보드 입력을 받는 함수

- 입력된 값은 기본적으로 문자열 형태로 반환
 - 적절히 형 변환하여 다른 타입으로 사용

```
>>> year = input('당신이 태어난 년도는 ? ')
당신이 태어난 년도는 ? 2007
```

사용자가 직접 입력한 후 Enter를 누른다.

```
>>> print(year)
2007
>>> type(year)
<class 'str'>
```

```
>>> age = 2026 - year
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

```
>>> age = 2026 - int(year)
>>> print('나이', age)
나이 19
```

```
>>> year = int(input('당신이 태어난 년도는 ? '))
당신이 태어난 년도는 ? 2005
>>> print('나이', 2026 - year)
나이 21
```

다양한 함수

```
print(abs(-3))
print(min(10, 20, 1))
print(max(10, 100, 1))
print(min([10, 11, 12]))
print(max([10, 11, 12]))
```

[6] ✓ 0.0s Python

... 3
1
100
10
12

```
print(round(3.1415))
print(round(3.1415, 3))
```

[7] ✓ 0.0s Python

... 3
3.142

내장 모듈 math 활용

- 파이썬의 모듈
 - 프로그램의 기능을 확장하기 위해 만드는 파이썬 소스 파일
- 내장 모듈
 - 따로 설치 없이 사용할 수 있는 모듈
- 대표적인 파이썬 내장 모듈 **math**
 - 수학 관련 상수와 함수가 정의

[표 2-4] R 주요 연산자

분류	연산자
상수	pi e
제곱근과 지수, 로그 함수	sqrt() exp() log() log10()
천정과 바닥값, 내림	ceil() floor() trunc()
삼각 함수	sin() cos() tan()

패키지(package)

관련 여러 모듈(module)들의 모임

- 디렉토리 구조로 이루어져 있으며, 하위 디렉토리에는 모듈들이 위치
- 패키지 판다스(pandas)
 - 파이썬에서 테이블 형태의 데이터를 조작하고 분석하기 위한 강력한 라이브러리
 - 데이터프레임(dataframe)과 시리즈(series)라는 두 가지 핵심 데이터 구조를 제공
 - 시리즈
 - 1차원 배열과 유사
 - 데이터프레임
 - 2차원 테이블 형태의 자료구조
- 배포판인 아나콘다(anaconda)를 사용
 - 판다스는 이미 설치
- 표준 파이썬을 사용
 - 패키지 판다스를 다음 pip로 설치
- 먼저 표준 파이썬이 설치되어 있는 폴더 하부 Scripts에서 명령 프롬프트를 하나 실행
 - C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>

패키지(package) 설치

명령어 pip

- 명령어 pip.exe 있는 위치
 - 먼저 표준 파이썬이 설치되어 있는 폴더 하부 Scripts에서 명령 프롬프트를 하나 실행
 - C:\Users\[사용자계정]\AppData\Local\Programs\Python\Python312\Scripts>
- 명령 프롬프트에서 먼저 명령 pip show pandas로 판다스 설치 유무를 확인
 - > pip show pandas
- 다음 명령으로 판다스 설치
 - > pip install pandas

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>pip show pandas
WARNING: Package(s) not found: pandas

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>pip show pandas
WARNING: Package(s) not found: pandas

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>pip install pandas
Collecting pandas
  Downloading pandas-2.2.2-cp312-cp312-win_amd64.whl.metadata (19 kB)
Requirement already satisfied: numpy>=1.26.0 in c:\users\pc\appdata\local\programs\python\python312\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\pc\appdata\roaming\python\python312\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\pc\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\pc\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\pc\appdata\roaming\python\python312\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Downloading pandas-2.2.2-cp312-cp312-win_amd64.whl (11.5 MB)
----- 11.5/11.5 MB 43.5 MB/s eta 0:00:00
Installing collected packages: pandas
Successfully installed pandas-2.2.2

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>
```

패키지 numpy

- 넘파이 numpy

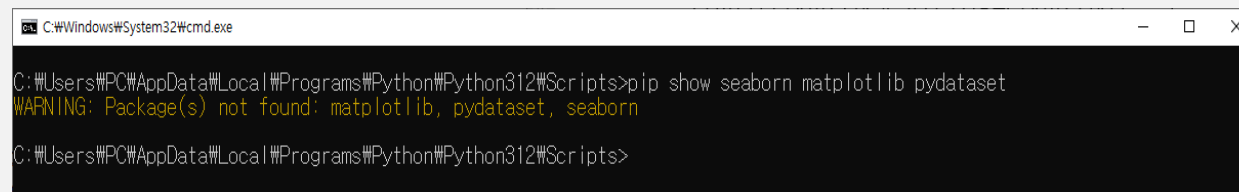
- 파이썬에서 수치 계산을 빠르게 수행하는 고성능 배열과 행렬 연산을 위한 과학용 컴퓨팅 라이브러리 (numpy.org)
- 판다스를 설치하면 넘파이는 함께 설치
 - 명령 프롬프트에서 먼저 명령 `pip show numpy`로 넘파이 설치 여부를 확인
 - > `pip show numpy`

matplotlib seaborn

- **맷플롯립으로 읽는 패키지 matplotlib**
 - 파이썬에서 2D 그래픽 그림을 생성하는 데 사용되는 데이터 시각화 라이브러리
 - 다양한 차트, 플롯, 히스토그램 등을 생성
- **패키지 시본 seaborn**
 - matplotlib에 기반한 고수준 인터페이스를 제공하는 시각화 라이브러리
 - 주로 통계적 그래픽 기능을 갖추고 있어 데이터 분석에 유용하게 활용

패키지 pydataset 개요

- 파이썬에서 사용할 수 있는 데이터셋(dataset)을 쉽게 메모리에 저장하고 활용
 - 언어 R 패키지인 datasets에 포함된 여러 데이터셋을 파이썬에서 바로 사용할 수 있도록 제공
 - 테스트용 데이터셋을 포함하고 있어, 데이터 분석이나 시각화 연습 등 다양한 용도로 활용
 - 이러한 데이터 셋을 장난감 데이터(toy data)라고도 부름
- 명령 > `pip install seaborn pydataset`
 - 한 번에 패키지 seaborn matplotlib pydataset을 설치
 - 설치에 필요한 패키지의 종속성에 따라 설치
 - 패키지 seaborn은 matplotlib을 필요로 하므로 seaborn만 설치해도 알아서 matplotlib을 설치
 - seaborn과 pydataset을 명령에 기술
 - 물론 패키지 3개를 모두 기술해도 상관없음
- > `pip install seaborn pydataset`
- > `pip install seaborn matplotlib pydataset`



```
C:\Windows\System32\cmd.exe

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>pip show seaborn matplotlib pydataset
WARNING: Package(s) not found: matplotlib, pydataset, seaborn

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>
```

패키지 pydataset 설치

```
C:\Windows\System32\cmd.exe

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>pip install seaborn pydataset
Collecting seaborn
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Collecting pydataset
  Downloading pydataset-0.2.0.tar.gz (15.9 MB)
    15.9/15.9 MB 50.4 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from seaborn) (2.2.2)
Collecting matplotlib<3.6.1,>=3.4 (from seaborn)
  Downloading matplotlib-3.8.4-cp312-cp312-win_amd64.whl.metadata (5.9 kB)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (4.50.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in c:\users\pc\AppData\Roaming\Python\Python312\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (23.2)
Requirement already satisfied: pillow>=8 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\pc\AppData\Roaming\Python\Python312\site-packages (from matplotlib<3.6.1,>=3.4->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\pc\AppData\Local\Programs\Python\Python312\Lib\site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in c:\users\pc\AppData\Roaming\Python\Python312\site-packages (from python-dateutil>=2.7->matplotlib<3.6.1,>=3.4->seaborn) (1.16.0)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Downloading matplotlib-3.8.4-cp312-cp312-win_amd64.whl (7.7 MB)
    7.7/7.7 MB 61.2 MB/s eta 0:00:00
Building wheels for collected packages: pydataset
  Building wheel for pydataset (pyproject.toml) ... done
  Created wheel for pydataset: filename=pydataset-0.2.0-py3-none-any.whl size=15939425 sha256=1cbd931501ce517446e8f66de03e0e07b50b0356457f3f9a7ed6a8ff7950db09
  Stored in directory: c:\users\pc\AppData\Local\pip\Cache\wheels\4c\82\ad\ff04abc617222b10438b1285ab9b5cfaecd180c10a7c81cff54
Successfully built pydataset
Installing collected packages: matplotlib, seaborn, pydataset
Successfully installed matplotlib-3.8.4 pydataset-0.2.0 seaborn-0.13.2

C:\Users\PC\AppData\Local\Programs\Python\Python312\Scripts>
```

pydataset 활용

- 설치된 pydataset의 전체적인 정보를 얻기 위해 데이터셋 목록을 표시
- 패키지 pydataset에서 데이터를 불러와 data()로 생성
 - 데이터셋 ID(dataset_id)와 제목(title)이 포함된 데이터프레임을 반환
- 패키지 pydataset에는 757개의 샘플 데이터
 - 열 dataset_id은 데이터 셋의 식별자(id)이며 열 title은 데이터 셋 이름

```
>>> from pydataset import data
>>>
>>> all_data = data()
>>> all_data
```

	dataset_id	title
0	AirPassengers	Monthly Airline Passenger Numbers 1949-1960
1	BJsales	Sales Data with Leading Indicator
2	BOD	Biochemical Oxygen Demand
3	Formaldehyde	Determination of Formaldehyde
4	HairEyeColor	Hair and Eye Color of Statistics Students
..
752	VerbAgg	Verbal Aggression item responses
753	cake	Breakage Angle of Chocolate Cakes
754	cbpp	Contagious bovine pleuropneumonia
755	grouseticks	Data on red grouse ticks from Elston et al. 2001
756	sleepstudy	Reaction times in a sleep deprivation study

[757 rows x 2 columns]

pydataset 모듈에서 data 함수를 메모리에 올려 사용할 수 있도록 한다.

패키지 pydataset 내장 데이터 cars

- 데이터 셋 cars

- 자동차의 속도 speed와 정차 길이 dist가 있는 테이블 형태의 데이터
 - 첫 열은 각 행의 번호이며 첫 줄의 speed와 dist는 각 열 이름

```
>>> cars = data('cars')
>>> cars
```

	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10
7	10	18
...		
49	24	120
50	25	85

시속 7km인 차가 정차하는데 22미터가 측정된 하나의 관측 값을 말한다.

테이블 형식의 자료의 행 제목으로 기본적으로 번호를 표시한다.

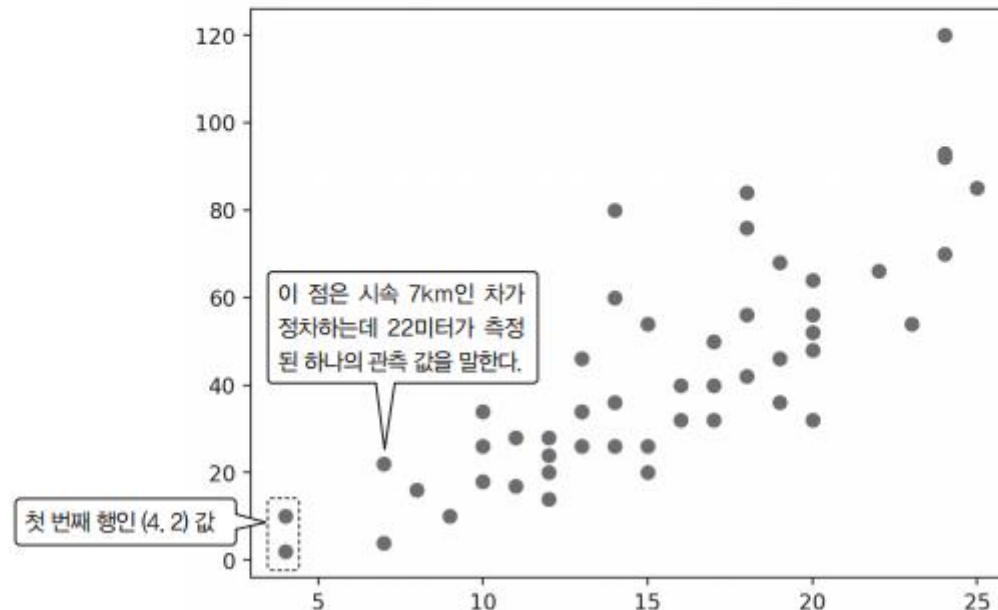
산점도(scatter plot)

- 데이터 시각화 matplotlib 사용

- plt.scatter(x = cars.speed, y = cars.dist)
 - 데이터 cars에서 50개 데이터 좌표를 작은 원으로 그린 그림

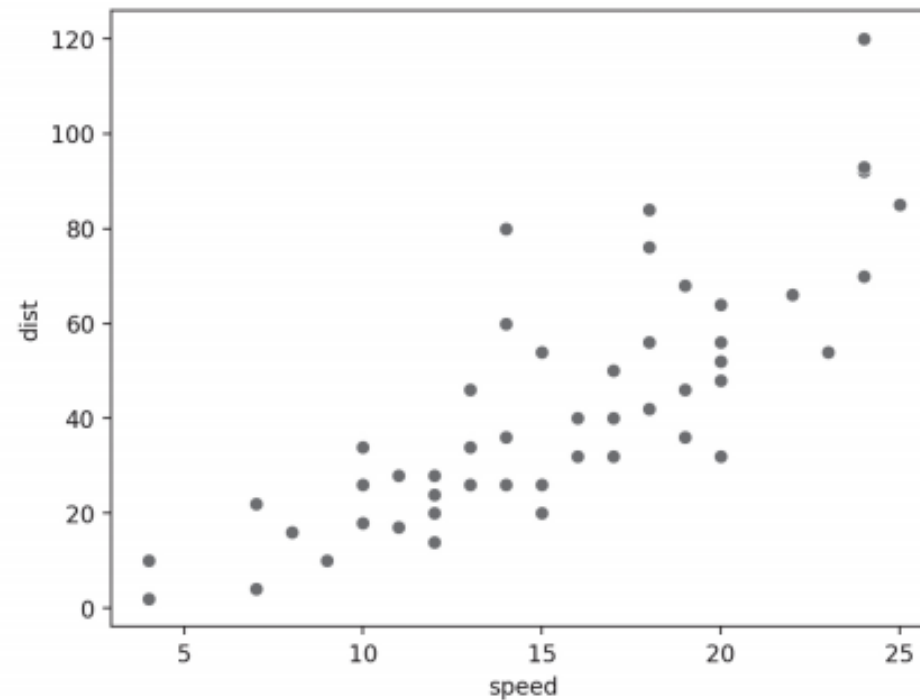
```
>>> import matplotlib.pyplot as plt
>>>
>>> plt.scatter(x = cars.speed, y = cars.dist)
>>> plt.show()
```

일반 프로그램 실행에서는 반드시 필요한 구문이나, 주피터 노트북 실행에서는 생략이 가능하다.



패키지 seaborn의 함수 scatterplot()으로 그린 산점도

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>>
>>> sns.scatterplot(x = cars.speed, y = cars.dist)
>>> plt.show()
```



주피터 노트북에서 그림을 선명하게 그리기와 한글 설정

TIP 주피터 노트북에서 그림을 선명하게 그리기와 한글 설정

주피터 노트북 또는 VS code의 반응형 윈도우(interactive window)에서 matplotlib이나 seaborn의 그림을 선명하게 그리려면 다음을 설정한 이후에 그리도록 한다.

```
%config InlineBackend.figure_format = 'retina'
```

다음은 matplotlib에서 한글을 처리하기 위한 코드이다. rc는 실행 시간 환경 설정(runtime configuration)을 의미한다. 인터프리터에서 실행 시간에 한 번만 실행되면 이후에 한글 지원이 가능하다.

```
import matplotlib.pyplot as plt
```

```
plt.rc('font', family='Malgun Gothic')
```

```
plt.rc('axes', unicode_minus=False)
```

위 문장은 다음 plt.rcParams[...] 설정 문장으로도 가능하다.

```
plt.rcParams['font.family'] = 'Malgun Gothic'
```

```
plt.rcParams['axes.unicode_minus'] = False
```

데이터프레임(dataframe)

판다스가 지원하는 테이블 형태의 행과 열로 구성된 자료구조

- 데이터 mpg(miles per gallon)의 데이터프레임 객체를 변수 mpg에 저장
 - mpg의 메소드 head(mpg)로 mpg의 첫 5개의 행을 표시

```
>>> mpg = data('mpg')
>>> mpg.head()
  manufacturer model  displ  year  cyl    trans  drv  cty  hwy  fl  class
1         audi   a4    1.8   1999    4  auto(l5)   f   18   29   p  compact
2         audi   a4    1.8   1999    4 manual(m5)   f   21   29   p  compact
3         audi   a4    2.0   2008    4 manual(m6)   f   20   31   p  compact
4         audi   a4    2.0   2008    4  auto(av)    f   21   30   p  compact
5         audi   a4    2.8   1999    6  auto(l5)    f   16   26   p  compact
```

메소드 mpg.info()

- mpg는 234개 자동차 모델의 정보
 - 11개의 열로 구성된 데이터프레임
 - 제조업체(열 manufacturer)와 모델 이름(열 model)
 - 도심 연비(열 cty)와 고속도로 연비(열 hwy) 등

```
>>> mpg.info()
<class 'pandas.core.frame.DataFrame'>
Index: 234 entries, 1 to 234
Data columns (total 11 columns):
#   Column          Non-Null Count  Dtype
---  -
0   manufacturer    234 non-null   object
1   model           234 non-null   object
2   displ           234 non-null   float64
3   year            234 non-null   int64
4   cyl             234 non-null   int64
5   trans           234 non-null   object
6   drv             234 non-null   object
7   cty             234 non-null   int64
8   hwy             234 non-null   int64
9   fl              234 non-null   object
10  class           234 non-null   object
dtypes: float64(1), int64(4), object(6)
memory usage: 21.9+ KB
```

234개의 관측(행) 값이 1에서 234의 행 번호로 구성

결측값(null, NA) 없이 234개의 값이 있음

11개의 열 제목

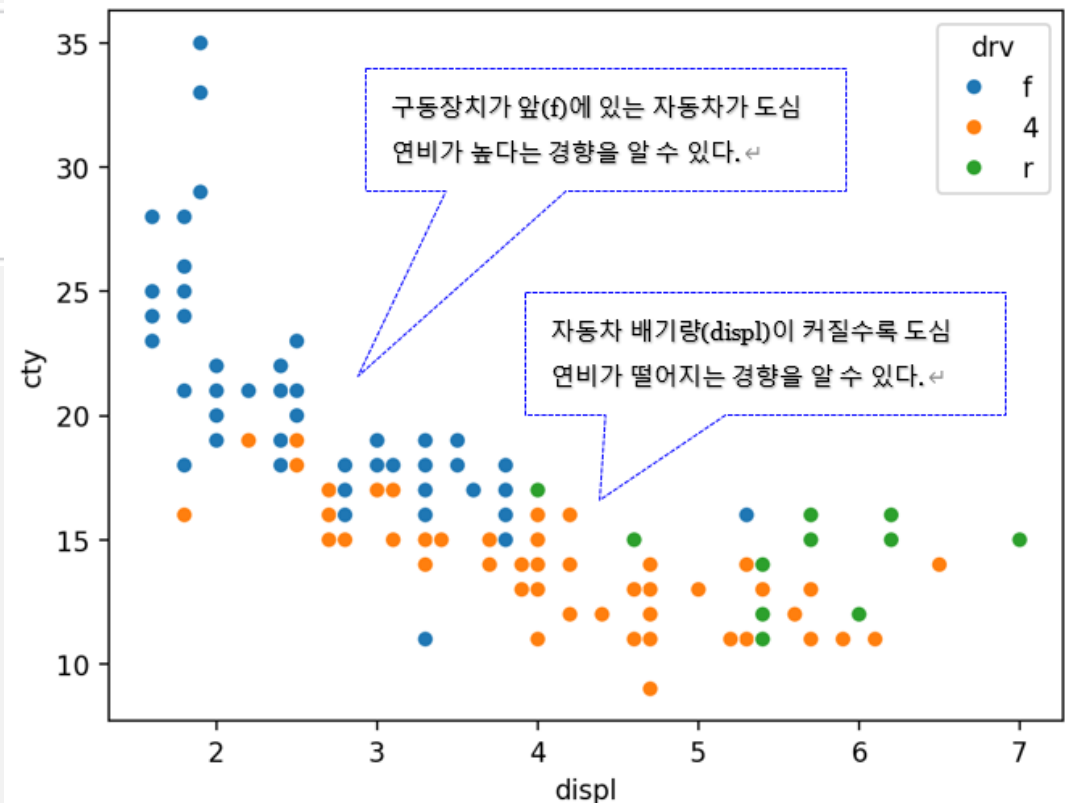
sns.scatterplot(x = mpg.displ, y = mpg.cty, hue = mpg.driv)

- 데이터프레임 mpg의 열 cty는 mpg.cty로 참조

- 가로 x 축인 자동차 배기량(displ: displacement)에 따른 세로 y 축 도심 연비(cty: city)를 나타내는 점
- 인자 hue = mpg.driv를 지정
 - 각각의 데이터를 구동방식(drv)에 따라 구분
 - 자동차 구동방식 driv
 - f(front-wheel drive, 전륜), r(rear wheel drive, 후륜), 4(4wd, 4륜) 등 3 종류

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>>
>>> sns.scatterplot(x = mpg.displ, y = mpg.cty, hue = mpg.driv)
>>> plt.show()
```

이미 실행했으면 다시 할 필요 없음



2.2 제어 흐름과 함수

-



조건 if

- if 조건: 블록 문장은 조건이 True이면 들여쓰기의 블록을 실행하는 조건 문장
 - 조건 이후에 콜론 :은 반드시 필요
 - 들여쓰기는 빈칸으로 2개 또는 4개를 일관되게 사용
 - 들여쓰기가 블록을 구성하는 문법으로 잘못 사용하면 오류가 발생하니 주의

```
>>> n = 20
>>> if n%2 == 0:
...     print('짝수')
...
짝수
```



다음 코드에서 n, 21은 홀수이므로 아무것도 출력되지 않는다.

```
>>> n = 21
>>> if n%2 == 0:
...     print('짝수')
...
```

조건 if else

- **else:**
 - else 이후에도 콜론 :이 반드시 필요

```
>>> age = 18
>>> if age >= 20:
...     print('성인')
... else:
...     print('미성년자')
...
미성년자
```

```
>>> age = 21
>>> if age >= 20:
...     print('성인')
... else:
...     print('미성년자')
...
성인
```

- **한 줄 조건 구문 if else**

```
>>> age = 18
>>> s = '성인' if age >= 20 else '미성년자'
>>> print(s)
미성년자
```

```
>>> age = 20
>>> s = '성인' if age >= 20 else '미성년자'
>>> print(s)
성인
```


반복된 조건 if elif

- else if가 아니라 elif로 쓰며
- 마지막에 else: 사용 가능

```
>>> point = 92
>>> if (90 <= point):
...     print('A')
... elif (80 <= point):
...     print('B')
... elif (70 <= point):
...     print('C')
... elif (60 <= point):
...     print('D')
... else:
...     print('F')
...
A
```

```
>>> point = 62
>>> if 90 <= point:
...     print('A')
... elif 80 <= point:
...     print('B')
... elif 70 <= point:
...     print('C')
... elif 60 <= point:
...     print('D')
... else:
...     print('F')
...
D
```

match 구문

- 파이썬 구문 match는 R과 자바, C 언어의 switch 구문과 비슷
- match와 case, default는 키워드
 - 연산식 또는 변수 exp의 값을 각 case의 value1, value2 등과 비교
 - 첫 번째로 일치하는 case를 찾으면 해당 valuei 이후의 블록인 statements를 실행
 - 어떤 case와도 일치하지 않으면 옵션인 case _: 이후의 블록을 실행

```
■ match exp:  
    case value1:  
        statements  
    case value2:  
        statements  
    ...  
    case _:  
        statements
```

- **exp**: match 구문에서 평가할 표현식(expression)이나 변수이며, 이 값이 어떤 case와 일치하는지 확인
- **value1, value2, ...**: 여러 개의 경우(case)를 정의하며, exp의 값과 각 경우를 비교
- **statements**: 각 경우에 해당하는 실행문장이 정의되는 블록이며, exp의 값이 해당하는 경우, 해당 블록이 실행

match 구문 사례

```
>>> value = "apple"
>>> match value:
...     case 'apple':
...         result = "사과"
...     case 'banana':
...         result = "바나나"
...
... print(result)
사과
```

```
value = "banana"

match value:
    case 'apple':
        result = "사과"
    case 'banana':
        result = "바나나"

print(result)
```

Python

바나나

```
value = "mango"

match value:
    case 'apple':
        result = "사과"
    case 'banana':
        result = "바나나"
    case _:
        result = None

print(result)
```

Python

None

반복 for 문

- for 구문

- in 이후의 리스트(list), 튜플(tuple), 사전(dictionary), 집합(set) 등의 항목이 있는 모임형으로 반복
- for 마지막 콜론 :은 반드시 필요하며 이후 반복 몸체는 들여쓰기로 블록을 구성

```
>>> for x in range(1, 6):  
...     print(x)  
1  
2  
3  
4  
5
```

```
>>> num = [10, 20, 30, 40]  
>>> for i in num:  
...     print(i)  
...  
10  
20  
30  
40
```

반복 while 문

- while 조건: 블록 구문으로 조건이 True이면 블록을 반복하는 구문
- 다음 코드로 1부터 10까지의 합을 sum에 구해 출력

```
>>> i = 1
>>> sum = 0
>>> while i <= 10:
...     sum += i
...     i += 1
...
>>> print(sum)
55
```

함수 선언과 호출

- 특정한 기능을 수행하는 함수를 직접 정의
 - 사용자 정의 함수를 선언
 - 키워드 **def**를 사용한 다음 구문을 활용

```
def function_name(parameters):  
    function_body  
    return value
```

- **function_name**: 이는 함수 정의 후 해당 함수를 호출하는 데 사용되는 함수 개체의 이름
- **parameters**: 매개변수(parameters)라 부르며 함수를 호출할 때마다 괄호 안에 배치되고 쉼표로 구분된 함수 정의의 변수로, 인수(arguments)라고 불리는 실제 값으로 설정
- **function_body**: 함수 구현 부분으로 함수의 기능을 수행, 함수 몸체라 부름
- **return value**: 옵션으로 함수를 수행한 후 반환되는 값

- 함수이름과 매개변수 이후에는 콜론 :이 반드시 표기
 - 다음은 들여쓰기로 사용해 함수 몸체를 구현

함수 선언과 호출

- 함수 이름 이후 괄호가 없으면

- 단지 메모리에 저장된 함수 객체를 참조한다. 다음 16진수 값은 객체가 저장된 메모리 주소

```
def hello():  
    print('Hello, Python!')  
  
hello()
```

Python

Hello, Python!

```
hello
```

Python

```
<function __main__.hello()>
```

```
type(hello)
```

Python

```
function
```

함수 매개변수 기본 값

- 함수 선언에서 매개변수에 값을 대입하면 인자 없이 함수를 호출 가능
 - = 연산자를 사용
- 인자없이 다음 함수 hi() 호출로 기본 인자 값으로 함수가 실행
- 함수 hi('철수') 호출로 기본 인자 값은 무시되고 인자로 지정된 철수로 함수가 실행

```
def hi(one = '친구들'):
    print('안녕,', one)
```

```
hi()
hi('철수')
```

Python

```
안녕, 친구들
안녕, 철수
```


함수 circle_area()

- 매개변수 기본 값으로 반지름 r을 1로 지정해 원의 면적을 구하는 함수
 - 인자 없이 함수 circle_area() 호출은 반지름 기본 값 1인 원의 면적을 구하기
 - 인자 r을 직접 지정하는 키워드 인자 지정 방식인 r = 5.6을 사용 가능
- return 문장이 없는 함수를 호출해 그 결과를 출력하면 None이 출력

```
import math as m
def circle_area(r = 1):
    print(m.pi * r**2)

circle_area()
circle_area(4.5)
```

Python

```
3.141592653589793
63.61725123519331
```

```
circle_area(r = 5.6)
```

Python

```
98.5203456165759
```

⌵ ▶ ⏪ ⏩ ... 🗑

```
print(circle_area(r = 7.2))
```

↺

Python

```
162.8601631620949
None
```

return 구문과 함수 매개변수 지정

- 함수 몸체에 return 구문으로 함수 결과를 반환
 - 할 수 있다. 코드 `print(circle_area(r = 7.2))` 호출로 반지름이 7.2인 원의 면적을 반환 받아 출력

```
def sum_sub(x, y):  
    hap = x + y  
    cha = x - y  
    return hap, cha
```

```
sum_sub(10, 3)
```

Python

```
(13, 7)
```

```
sum_sub(x = 10, y = 3)
```

Python

```
(13, 7)
```

```
sum_sub(y = 3, x = 10)
```

Python

```
(13, 7)
```

lambda 익명 함수

- 함수 이름을 지정하지 않은 익명 함수(anonymous function)
 - 일반적으로 다시 사용할 수 없음
- 람다 함수를 선언하면서 이후에 인자를 (3, 4)를 적용해 호출

```
import math as m  
(lambda a, b: m.sqrt(a**2 + b**2))(3, 4)
```

Python

5.0

2.3 표준 파이썬 기본 자료 구조



리스트(list)

- 여러 요소(elements)를 담을 수 있는 가변(mutable)한 데이터 구조
 - 리스트는 목록이라고도 부르며 리스트의 구성 요소는 항목 또는 원소라고도 부름
 - 원소는 순서를 가지고 있음
 - 리스트는 대괄호 []로 표현되며, 요소들은 쉼표 , 로 구분해 기술
- 대괄호를 사용하여 리스트를 생성
- 리스트의 자료형은 class list

```
# %% list tuple  
lst = [10, 20, 30]  
print(lst, type(lst))
```

Python

```
[10, 20, 30] <class 'list'>
```

```
list(['java', 'python', 'R'])
```

Python

```
['java', 'python', 'R']
```

```
print(list(range(5)))  
print(list(range(1, 10, 2)))
```

Python

```
[0, 1, 2, 3, 4]  
[1, 3, 5, 7, 9]
```

리스트 참조

```
lst = [1, 3.92, 3 > 4, 'list']  
print(lst[0], lst[2])
```

Python

1 False

```
lst[2:4]
```

Python

[False, 'list']

```
print(lst[2:])  
print(lst[:4])  
print(lst[-1:-4:-1])
```

Python

[False, 'list']
[1, 3.92, False, 'list']
['list', False, 3.92]

메소드 append() insert()

- 리스트는 어떠한 자료형도 항목으로 구성 가능
- 리스트의 메소드 append()
 - 리스트에 새로운 요소를 마지막에 추가

```
import math as m
a = [m.pi, 30, 10 > 4, 'list', lambda x: x**3]
a.append('py')
a
```

Python

```
[3.141592653589793, 30, True, 'list', <function __main__.<lambda>(x)>, 'py']
```

```
def add2(x, y):
    return x + y

a.insert(1, add2)
a[1](10, 20)
```

Python

30

a

Python

```
[3.141592653589793,
 <function __main__.add2(x, y)>,
 30,
 True,
 'list',
 <function __main__.<lambda>(x)>,
 'py']
```

메소드 remove() pop()

a

```
[3.141592653589793,  
<function __main__.add2(x, y)>,  
30,  
True,  
'list',  
<function __main__.<lambda>(x)>,  
'py']
```

a.remove(30)

a

```
[3.141592653589793,  
<function __main__.add2(x, y)>,  
True,  
'list',  
<function __main__.<lambda>(x)>,  
'py']
```

a.pop()

'py'

a

```
[3.141592653589793,  
<function __main__.add2(x, y)>,  
True,  
'list',  
<function __main__.<lambda>(x)>]
```


리스트 컴프리헨션

- 파이썬에서 리스트를 간결하게 생성하는 방법 중 하나
 - 반복문과 조건문을 사용하여 리스트를 생성하는 표현식
 - 리스트 컴프리헨션 구조
 - `[expression for item in iterable if condition]`
 - expression: 각 요소를 생성하는 표현식
 - item: iterable에서 가져온 요소
 - iterable: 반복 가능한 객체 (예: 리스트, 튜플, 범위 등)
 - condition (선택사항): 요소를 선택하는 조건
- 리스트 컴프리헨션 장점
 - 간결함
 - 반복문을 사용하는 일반적인 방법보다 훨씬 간결하고 가독성이 좋은 코드를 작성
 - 속도
 - 내부적으로 최적화되어 있어, 일반적으로 반복문보다 더 빠르게 실행
 - 효율성
 - 리스트 컴프리헨션은 한 줄에 표현되기 때문에 코드를 작성하고 이해하기가 훨씬 용이
 - 한 줄에 모든 로직을 포함하기 때문에 변수의 스코프와 관련된 오류를 줄일 수 있음

리스트 컴프리헨션 활용

```
my_list = [x for x in range(10)]  
my_list
```

Python

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[x for x in range(10) if x%2 == 1]
```

Python

```
[1, 3, 5, 7, 9]
```

```
['even' if i%2 == 0 else 'odd' for i in range(1, 11)]
```

Python

```
['odd', 'even', 'odd', 'even', 'odd', 'even', 'odd', 'even', 'odd', 'even']
```

```
[[i for j in range(5)] for i in range(3)]
```

Python

```
[[0, 0, 0, 0, 0], [1, 1, 1, 1, 1], [2, 2, 2, 2, 2]]
```

```
[[j for j in range(i, i+6)] for i in range(3)]
```

Python

```
[[0, 1, 2, 3, 4, 5], [1, 2, 3, 4, 5, 6], [2, 3, 4, 5, 6, 7]]
```

사전(dictionary) 개요와 생성

키(key)와 값(value)의 쌍을 저장하는 자료형

- **쉽표로 구분된 키:값 쌍 목록을 중괄호 { }로 묶어 사전을 정의**
 - 중괄호 { }로 둘러싸여 있으며, 각 쌍은 쉽표 ,로 구분
 - **콜론 :은 각 키를 연관된 값을 구분**
- **{ key1: value1, key2: value2, ..., keyn: valuen }**
 - 키: 값 형태로 키와 값은 콜론 :으로 구분하며 항목들은 쉽표 ,로 구분
 - **메소드 items()로 (키, 값) 항목의 목록 형태인 dict_items를 반환**
 - key1, key2, ..., keyn: 키는 수정불가능(또는 불변, immutable, hashable)한 자료형만 가능
 - 메소드 keys()로 키의 목록 형태인 dict_keys를 반환
 - **value1, value2, ..., valuen: 값은 모든 자료형이 가능**
 - 메소드 values()로 값의 목록 형태인 dict_values를 반환
- **사전의 특성**
 - 데이터 조직화
 - **키와 값의 쌍을 사용하여 데이터를 구조화하고 관리 가능**
 - 빠른 검색
 - **키를 사용하여 값을 빠르게 찾을 수 있음**
 - 중복 제거
 - **사전은 키가 고유하므로 중복된 데이터를 쉽게 제거**

사전 코드

- 사전은 키를 사용하여 값을 참조

```
# dict  
mart = {'라면':1200, '음료수':1500, '과자':800}  
print(mart, type(mart))
```

Python

```
{'라면': 1200, '음료수': 1500, '과자': 800} <class 'dict'>
```

```
print(mart['라면'])  
print(mart['과자'])  
print(mart['음료수'])
```

Python

```
1200  
800  
1500
```

```
data = {1: '일', 3.14: 2.71, (1, 2): '튜플', 'list': [1, 2, 3]}  
data
```

Python

```
{1: '일', 3.14: 2.71, (1, 2): '튜플', 'list': [1, 2, 3]}
```

사전 코드

- Unhashable type
 - 키로 사용 불가능

```
data = {1: '일', 3.14: 2.71, [1, 2]: '리스트', 'list': [1, 2, 3]}
```

Python

```
-----  
TypeError                                Traceback (most recent call last)  
Cell In[4], line 1  
----> 1 data = {1: '일', 3.14: 2.71, [1, 2]: '리스트', 'list': [1, 2, 3]}
```

TypeError: unhashable type: 'list'

```
f_cnt = dict (('apple', 2), ('banana', 5), ('orange', 3))  
f_cnt
```

Python

```
{'apple': 2, 'banana': 5, 'orange': 3}
```

```
f_price = dict (apple=4000, banana=300, orange=2000)  
f_price
```

Python

```
{'apple': 4000, 'banana': 300, 'orange': 2000}
```

항목 추가와 메소드 items()

- 사전에 없는 항목을 참조해 값을 대입
 - 새로운 키: 값의 항목이 추가
 - `mart['삼각김밥'] = 1700`
 - 키 '삼각김밥'의 값 1700을 저장
- 이미 있는 항목을 참조해 값을 저장
 - 이전 값이 수정
- 사전의 메소드 `keys()`
 - 사전의 모든 키를 자료형 `dict_keys`로 반환
- 사전의 메소드 `values()`
 - 사전의 모든 값을 자료형 `dict_values`로 반환
- 메소드 `items()`
 - 사전의 모든 키와 값의 쌍을 자료형 `dict_items`로 반환

```
mart = {'라면': 1200, '음료수': 1500, '과자': 800}  
mart['삼각김밥'] = 1700  
mart
```

Python

```
{'라면': 1200, '음료수': 1500, '과자': 800, '삼각김밥': 1700}
```

```
mart['과자'] = 900  
mart
```

Python

```
{'라면': 1200, '음료수': 1500, '과자': 900, '삼각김밥': 1700}
```

```
mart.keys()
```

Python

```
dict_keys(['라면', '음료수', '과자', '삼각김밥'])
```

```
mart.values()
```

Python

```
dict_values([1200, 1500, 900, 1700])
```

```
mart.items()
```

Python

```
dict_items([('라면', 1200), ('음료수', 1500), ('과자', 900), ('삼각김밥', 1700)])
```

튜플 개요

- 튜플
 - 변경할 수 없는(immutable, 불변의) 시퀀스 자료형
 - 여러 항목을 담을 수 있으며, 대괄호 대신 소괄호로 둘러싸 표현
- 튜플의 필요성
 - 불변성(immutable)
 - 한 번 생성된 튜플은 수정할 수 없기 때문에, 데이터 무결성을 보장하고 의도치 않은 변경을 방지
 - 성능
 - 리스트보다 메모리 사용이 적고, 읽기 속도가 빠름
 - 해시 가능성(hashable)
 - 튜플은 딕셔너리의 키나 집합의 원소로 사용될 수 있음

튜플 활용

```
# %% list tuple
tp1 = 1, 2, 3
tp2 = (10, 20, 30)
print(tp1, tp2)
```

Python

(1, 2, 3) (10, 20, 30)

```
tp3 = tuple((10, 20, 30))
tp4 = tuple(['py', 'R'])
print(tp3, tp4)
```

Python

(10, 20, 30) ('py', 'R')

```
tpl = 1, 3.92, 3 > 4, 'tuple'
print(tpl, type(tpl))

print(tpl[0])
print(tpl[3])
```

Python

(1, 3.92, False, 'tuple') <class 'tuple'>
1
tuple

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = tuple1 + tuple2
concatenated_tuple
```

Python

(1, 2, 3, 4, 5, 6)

```
# tpl[1] = 100
# tpl.append('py') # 오류발생
```

Python

집합 개요

- **중복된 원소가 없는 순서가 없는 자료형**
 - 중괄호 { } 안에 쉼표 ,로 구분된 값을 넣어 생성
 - 수정 불가능한 집합 원소는 추가나 삭제가 가능
 - 집합 자체는 수정 가능
 - 집합의 원소는 중복을 불허
 - 멤버십 검사와 중복 제거에 주로 사용될 수 있음
 - 중복된 데이터를 제거하는 데 유용
- **{ element0, element1, element2, ..., elementn }**
 - 원소인 elementi는 고유한(unique) 값으로 중복을 허용하지 않으며
 - 원소의 순서는 의미가 없으며
 - 원소는 int, float, str, tuple 등 수정 불가능(immutable, hashable)한 것이어야 함

집합 활용

```
# %% set  
basket = {'apple', 'orange', 'apple', 'pear'}  
print(basket)
```

Python

```
{'orange', 'apple', 'pear'}
```

```
len(basket)
```

Python

```
3
```

```
print('grape' in basket)  
print('apple' in basket)
```

Python

```
False  
True
```

다양한 집합 연산

합집합, 교집합, 차집합, 여집합

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6, 7}
print(A.union(B))      # 합집합
print(A.intersection(B)) # 교집합
print(A.difference(B))  # 차집합
print(B.difference(A))  # 차집합
print(A.symmetric_difference(B)) # 여집합
```

Python

```
{1, 2, 3, 4, 5, 6, 7}
{3, 4}
{1, 2}
{5, 6, 7}
{1, 2, 5, 6, 7}
```

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6, 7}
print(A | B) # 합집합
print(A & B) # 교집합
print(A - B) # 차집합
print(B - A) # 차집합
print(A ^ B) # 여집합
```

Python

```
{1, 2, 3, 4, 5, 6, 7}
{3, 4}
{1, 2}
{5, 6, 7}
{1, 2, 5, 6, 7}
```

집합에 추가할 원소: 수정불가능한 객체

[1, 2]와 같은 수정가능한 리스트는 집합의 원소가 될 수 없음

- 집합 메소드 `add(item)`
 - 튜플, 삽입 가능
 - (1, 2)와 같은 수정불가능한 튜플은 집합의 원소가 될 수 있음
- 집합 메소드 `remove(item)`
 - item을 삭제

```
basket = {'apple', 'orange', 'apple', 'pear'}  
basket.add('banana')  
basket
```

Python

```
{'apple', 'banana', 'orange', 'pear'}
```

```
myset = {0}  
print(myset)  
myset.add([1, 2]) # 오류
```

Python

```
{0}
```

```
-----  
TypeError                                Traceback (most recent call last)  
Cell In[9], line 3  
      1 myset = {0}  
      2 print(myset)  
----> 3 myset.add([1, 2]) # 오류  
  
TypeError: unhashable type: 'list'
```

```
myset = {0}  
myset.add((1, 2))  
myset
```

Python

```
{(1, 2), 0}
```

```
myset.remove(0)  
myset
```

Python

```
{(1, 2)}
```

수정불가능 immutable

수정 불가능하다는 immutable(불변)은 변경할 수 없는 객체를 가리킴

- immutable 객체

- 생성된 후에 그 값을 변경 불가능

- 정수(int), 부동소수점(float), 문자열(str), 튜플(tuple) 등은 immutable한 객체

- 한 번 생성되면 값을 변경할 수 없으며, 변경하려고 하면 새로운 객체가 생성

- immutable의 특성

- 객체의 안전성과 불변성을 보장하여 예상치 못한 변경으로부터 객체를 보호

- 해시 가능한 객체만

- 딕셔너리(dict)의 키가 가능

- 집합(set)의 원소로 가능

내장 함수 hash(obj)

- 주어진 객체 obj의 해시 값을 반환
 - 해시 값은 정수
 - 객체의 내용에 기반하여 생성
 - 동일한 입력에 대해서 항상 동일한 해시 값을 반환
 - 동일한 값이면 항상 동일한 해시 값
 - 해시 값의 고유성은 데이터 구조에서 빠른 검색 및 비교를 위한 핵심 요소 중 하나
- 사전이나 리스트, 집합
 - 수정 가능한 객체
 - hash() 값을 가질 수 없음
- 기본 자료형과 튜플
 - 수정 불가능한 객체
 - 고유한 hash() 값

```
hash((1, 2))
```

✓ 0.0s

Python

```
-3550055125485641917
```

```
hash((1, 2))
```

✓ 0.0s

Python

```
-3550055125485641917
```

```
hash([1, 2])
```

⊗ 0.4s

Python

TypeError

Traceback (most recent call last)

Cell In[3], line 1

----> 1 hash([1, 2])

TypeError: unhashable type: 'list'

```
print(hash('py'), hash((1, 2)))
```

```
-1857589116887210153 -3550055125485641917
```

수정 불가능한 immutable 객체와 함수 id()

- 파이썬에서 객체의 고유 식별자(unique identifier)를 이해하는 데 중요한 역할
- immutable 객체는 변경할 수 없는 객체
 - 이러한 객체는 생성된 후에 그 값을 변경할 수 없으며,
 - 변경하려고 하면 새로운 객체가 생성
 - 결국 id() 값이 달라짐
- 함수 id()
 - 모든 객체의 고유 식별자(unique identifier)를 반환
 - 식별자는 객체가 메모리 상에서의 위치를 나타내는 정수 값
 - id() 함수를 사용하면
 - 두 개의 객체가 동일한 메모리 위치를 가리키는지 여부를 확인 가능

```
help(id)
Help on built-in function id in module builtins:

id(obj, /)
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
```

두 객체의 id()가 동일

- 두 객체는 동일한 메모리 위치를 가리키고 있음을 의미
 - 따라서 두 객체는 동일한 것

```
x, y = 10, 10  
print(id(x), id(y))
```

Python

```
140704255284296 140704255284296
```

```
x = 10  
print(id(x))  
x = 9 + 1  
print(id(x))
```

Python

```
140704255284296  
140704255284296
```

```
lst = [1, 2]  
print(id(lst))  
lst.append(3)  
print(id(lst))
```

Python

```
1408107247680  
1408107247680
```