2021년도
ICT 이노베이션스퀘어 확산 사업

[인공지능 : 기초부터 실전까지]

▸ 7차수 ~ 10차수 강의 교안

※ 본 교안은 강의 수강 용도로만 사용 가능합니다.
상업적 이용을 일절 금함.

사람투자 과학기술정보통신부 nipa 정보통신산업진흥원 National IT Industry Promotion Agency KSA 한국표준협회

6

입력과 출력

화면 출력

- print() 함수를 이용해 화면으로 출력
  - 기본 출력: 출력 형식을 지정하지 않음
  - 형식 지정 출력: 다양한 형식으로 출력 가능
- 기본 출력
  - 문자열을 print() 함수 안에 삽입

```
In: print("Hello Python!!")

Out: Hello Python!!
```

  - 문자열 여러 개를 연결해서 출력

```
In: print("Best", "python", "book")

Out: Best python book
```

  - 빈칸 대신 다른 문자열 삽입

```
In: print("Best", "python", "book", sep = "-:*:-")

Out: Best-:*:-python-:*:-book
```

화면 출력

- 빈칸 없이 두 문자열을 연결

```
In: print("abcd" +"efg")

Out: abcdefg
```

- 문자열을 여러 줄로 출력

```
In: print("James is my friend.₩nHe is Korean.")

Out: James is my friend.
     He is Korean.
```

- 문자열을 한 줄로 연결해서 출력

```
In: print("Welcome to ", end="")
    print("python!")

Out: Welcome to python!
```

화면 출력

- 형식 지정 출력
  - 나머지 연산자(%)를 이용한 형식 및 위치 지정

```
print("%type" % data)
print("%type %type" % (data1, data2))
```

  - %s로 문자열을 대입한 변수를 출력

```
In: name = "광재"
    print("%s는 나의 친구입니다." % name)

Out: 광재는 나의 친구입니다.
```

- 형식 지정 문자열에서 출력 위치 지정

```
print("{0} {1} {2} … {n}".format(data_0, data_1, data_2, …, data_n))
```

화면 출력

- 형식 지정 문자열의 출력 예

```
In: animal_0 = "cat"
    animal_1 = "dog"
    animal_2 = "fox"

     print("Animal: {0}".format(animal_0))
     print("Animal: {0},{1},{2}".format(animal_0, animal_1, animal_2))

Out: Animal: cat
     Animal: cat,dog,fox
```

- 형식 지정 문자열의 위치 지정

```
In: print("Animal: {1},{2},{0}".format(animal_0, animal_1, animal_2))

Out: Animal: dog,fox,cat
```

화면 출력

- 숫자의 출력 형식 지정

데이터(x)	출력 형식	출력 결과	설명
3	{N:2d}	↔3	정수를 공백 포함해 두 자리로 표시 (↔은 공백 한 칸을 의미함)
3	{N:05d}	00003	정수를 다섯 자리로 표시. 앞의 공백은 0으로 채움
12	{N:>5d}	↔↔↔12	정수를 다섯 자리로 표시. 숫자는 오른쪽으로 정렬
0.12345	{N:.3f}	0.123	실수를 소수점 셋째 자리까지 표시
7456000	{N:,}	7,456,000	통화 표시처럼 끝에서 셋째 자리마다 콤마(,)를 표시
0.3258	{N:.1%}	32.6%	소수를 퍼센트(%)로 표시. 퍼센트 표시에서 소수점 자리 수는 '.' 다음 숫자로 표시
92500000000	{N:.2e}	9.25e+10	숫자를 지수로 표시. 지수 표시에서 소수점 자리 수는 '.' 다음 숫자로 표시
16	{N:#x}	0x10	숫자를 16진수로 표시. #기호가 없으면 0x 없이 출력됨
8	{N:#o}	0o10	숫자를 8진수로 표시. #기호가 없으면 0o 없이 출력됨
2	{N:#b}	0b10	숫자를 2진수로 표시. #기호가 없으면 0b 없이 출력됨

키보드 입력

- 키보드로 데이터를 입력하기 위해서는 input() 함수를 이용
- 기본 구조

```
data = input("문자열")
```

- input() 함수로부터 입력받은 데이터를 print() 함수로 출력

```
In: yourName = input("당신의 이름은? ")
    print("당신은 {}이군요.".format(yourName))

Out: 당신의 이름은?  홍길동
     당신은  홍길동이군요.
```

- input() 함수로부터 입력받은 데이터를 숫자로 변환

```
In: a = input("정사각형 한 변의 길이는?: ")
    area = int(a) ** 2
    print("정사각형의 넓이: {}".format(area))

Out: 정사각형 한 변의 길이는?:  5
     정사각형의 넓이: 25
```

키보드 입력

- 입력하려는 숫자가 정수인지 실수인지 모를 경우
  - float() 함수를 사용

```
In: c = input("정사각형 한 변의 길이는?: ")
    area = float(c) ** 2
    print("정사각형의 넓이: {}".format(area))

Out: 정사각형 한 변의 길이는?:  3
     정사각형의 넓이: 9.0
```

파일 읽고 쓰기

- 파일 열기: 내장 함수 open() 사용

```
f = open('file_name', 'mode')
```

  - 파일 열기 속성

mode	의미
r	읽기 모드로 파일 열기(기본). 모드를 지정하지 않으면 기본적으로 읽기 모드로 지정됨
w	쓰기 모드로 파일 열기. 같은 이름의 파일이 있으면 기존 내용은 모두 삭제됨
x	쓰기 모드로 파일 열기. 같은 이름의 파일이 있을 경우 오류가 발생함
a	추가 모드로 파일 열기. 같은 이름의 파일이 없으면 w와 기능 같음
b	바이너리 파일 모드로 파일 열기
t	텍스트 파일 모드로 파일 열기(기본). 지정하지 않으면 기본적으로 텍스트 모드로 지정됨

  - mode는 혼합해서 사용 가능

파일 읽고 쓰기

- 파일 쓰기
  - 파일 쓰기를 하려면 파일을 쓰기 모드로 열어야
  - 파일을 열고 지정한 내용을 쓴 후에는 파일을 닫아야 함
- 파일 쓰기를 위한 코드 구조

```
f = open('file_name', 'w')
f.write(str)
f.close()
```

- 파일 쓰기 예제

```
In: cd C:₩myPyCode
Out: C:₩myPyCode
```

```
In: f = open('myFile.txt', 'w')          # (1)'myFile.txt' 파일 쓰기 모드로 열기
    f.write('This is my first file.')    # (2) 연 파일에 문자열 쓰기
    f.close()                            # (3) 파일 닫기
```

```
In: !type myFile.txt
Out: This is my first file.
```

파일 읽고 쓰기

- 파일 읽기
  - 파일을 읽으려면 파일을 읽기 모드로 열어야 함
  - 파일의 내용을 읽고 마지막으로 파일을 닫음
- 파일 읽기를 위한 코드 구조

```
f = open('file_name', 'r')   # f = open('file_name')도 가능
data = f.read()
f.close()
```

- 파일 읽기 예제

```
In: f = open('myFile.txt', 'r')     # (1)'myFile.txt' 파일 읽기 모드로 열기
    file_text = f.read()            # (2) 파일 내용 읽은 후에 변수에 저장
    f.close()                       # (3) 파일 닫기

    print(file_text)                # 변수에 저장된 내용 출력

Out: This is my first file.
```

반복문을 이용해 파일 읽고 쓰기

- 파일에 문자열 한 줄씩 쓰기

```
In: f = open('two_times_table.txt','w')                    # (1)파일을 쓰기 모드로 열기
    for num in range(1,6):                                 # (2) for문: num이 1~5까지 반복
        format_string = "2 x {0} = {1}₩n".format(num,2*num) # 저장할 문자열 생성
        f.write(format_string)                             # (3) 파일에 문자열 저장
    f.close()                                              #(4) 파일 닫기
```

- 파일에 저장된 내용 출력

```
In: !type two_times_table.txt

Out: 2 x 1 = 2
     2 x 2 = 4
     2 x 3 = 6
     2 x 4 = 8
     2 x 5 = 10
```

파일에서 문자열 한 줄씩 읽기

- readline()
  - 파일로부터 문자열 한 줄을 읽음
  - 마지막 한 줄을 읽고 나서 다시 readline()을 실행하면 빈 문자열을 반환

```
In: f = open("two_times_table.txt")     # 파일을 읽기 모드로 열기
    line1 = f.readline()                # 한 줄씩 문자열을 읽기
    line2 = f.readline()
    f.close()                           # 파일 닫기
    print(line1, end="")                # 한 줄씩 문자열 출력(줄 바꿈 안 함)
    print(line2, end="")

Out: 2 x 1 = 2
     2 x 2 = 4
```

파일에서 문자열 한 줄씩 읽기

- 파일 전체에서 한 줄씩 읽어 오는 예제

```
In: f = open("two_times_table.txt")        # 파일을 읽기 모드로 열기
    line = f.readline()                    # 문자열 한 줄 읽기
    while line:                            # line이 공백인지 검사해서 반복 여부 결정
        print(line, end = "")              # 문자열 한 줄 출력(줄 바꿈 안 함)
        line = f.readline()                # 문자열 한 줄 읽기
    f.close() # 파일 닫기

Out: 2 x 1 = 2
     2 x 2 = 4
     2 x 3 = 6
     2 x 4 = 8
     2 x 5 = 10
```

파일에서 문자열 한 줄씩 읽기

- readlines()
  - 파일 전체의 모든 줄을 읽어서 한 줄씩을 요소로 갖는 리스트 타입으로 반환

```
In: f = open("two_times_table.txt")        # (1) 파일을 읽기 모드로 열기
    lines = f.readlines()                  # (2) 파일 전체 읽기(리스트로 반환)
    f.close()                              # (3) 파일 닫기

    print(lines)                           # 리스트 변수 내용 출력

Out:  ['2 x 1 = 2₩n', '2 x 2 = 4₩n', '2 x 3 = 6₩n', '2 x 4 = 8₩n', '2 x 5 = 10₩n',
'2 x 6 = 12₩n', '2 x 7 = 14₩n', '2 x 8 = 16₩n', '2 x 9 = 18₩n']
```

파일에서 문자열 한 줄씩 읽기

- lines 리스트에 할당된 문자열 for 문을 이용해 항목을 하나씩 처리

```
In: f = open("two_times_table.txt")        # 파일을 읽기 모드로 열기
    lines = f.readlines()                  # 파일 전체 읽기(리스트로 반환)
    f.close()                              # 파일 닫기
    for line in lines:                     # 리스트를 <반복 범위>로 지정
        print(line, end="")                # 리스트 항목을 출력(줄 바꿈 안 함)

Out: 2 x 1 = 2
     2 x 2 = 4
     2 x 3 = 6
     2 x 4 = 8
     2 x 5 = 10
```

- for 문의 <반복 범위>에 lines 변수 대신 바로 f.readlines()를 사용

```
In: f = open("two_times_table.txt")        # 파일을 읽기 모드로 열기
    for line in f.readlines():             # 파일 전체를 읽고, 리스트 항목을 line에 할당
        print(line, end="")                # 리스트 항목을 출력(줄 바꿈 안 함)
    f.close()

Out: 2 x 1 = 2
     2 x 2 = 4
     2 x 3 = 6
     2 x 4 = 8
     2 x 5 = 10
```

파일에서 문자열 한 줄씩 읽기

- for 문의 <반복 범위>에 있는 f.readlines() 대신 f만 입력하는 예

```
In: f = open("two_times_table.txt")          # 파일을 읽기 모드로 열기
    for line in f:                        # 파일 전체를 읽고, 리스트 항목을 line에 할당
        print(line, end="")                  # line의 내용 출력(줄 바꿈 안 함)
    f.close()                             # 파일 닫기

Out: 2 x 1 = 2
     2 x 2 = 4
     2 x 3 = 6
     2 x 4 = 8
     2 x 5 = 10
```

with 문을 활용해 파일 읽고 쓰기

- with 문의 구조

```
with open('file_name', 'mode') as f:
    <코드 블록>
```

- with 문의 활용
  - 파일에 문자열을 쓰는 예

```
In: with open('C:/myPyCode/myTextFile2.txt', 'w') as f:      # (1) 파일 열기
        f.write('File read/write test2: line1₩n')            # (2) 파일 쓰기
        f.write('File read/write test2: line2₩n')
        f.write('File read/write test2: line3₩n')
```

  - with 문을 이용해 생성한 파일을 읽는 예

```
In: with open('C:/myPyCode/myTextFile2.txt') as f:   # (1) 파일 열기
        file_string = f.read()                       # (2) 파일 읽기
        print(file_string)

Out: File read/write test2: line1
     File read/write test2: line2
     File read/write test2: line3
```

with 문을 활용해 파일 읽고 쓰기

- with 문을 반복문과 함께 이용하는 예

```
In: with open('C:/myPyCode/myTextFile3.txt', 'w') as f:       # 파일을 쓰기 모드로 열기
        for num in range(1,6):                                  # for문에서 num이 1~5까지 반복
          format_string = "3 x {0} = {1}₩n".format(num,3*num) # 문자열 생성
          f.write(format_string)                                # 파일에 문자열 쓰기
```

- with 문과 for 문을 이용해 파일의 문자열을 한 줄씩 읽는 예

```
In: with open('C:/myPyCode/myTextFile3.txt', 'r') as f: # 파일을 읽기 모드로 열기
        for line in f:           # 파일 전체를 읽고 리스트 항목을 line에 할당
            print(line, end="")  # line에 할당된 문자열 출력(줄 바꿈 안 함)

Out: 3 x 1 = 3
     3 x 2 = 6
     3 x 3 = 9
     3 x 4 = 12
     3 x 5 = 15
```

7

함수

함수

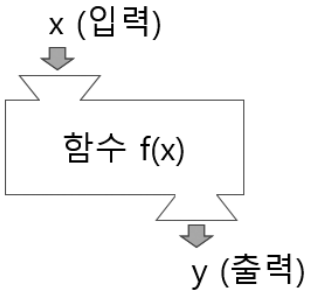
- 함수(function)는 특정 기능을 수행하는 코드의 묶음
- 함수를 이용하면 같은 기능을 수행하는 코드를 반복해서 작성할 필요가 없음
- 코드가 깔끔해지고 한번 만든 코드를 재사용할 수 있어서 코드를 작성하기가 편해짐
- 내장 함수의 예: print(), type()

함수 정의와 호출

- 수학에서의 함수

$$y=f(x)$$

- 함수에서 입력과 출력의 관계

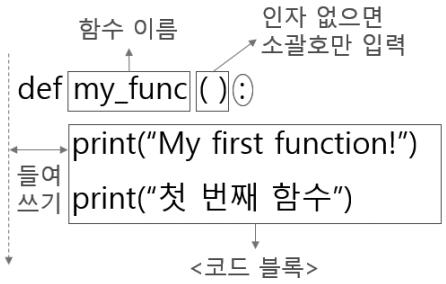


함수의 기본 구조

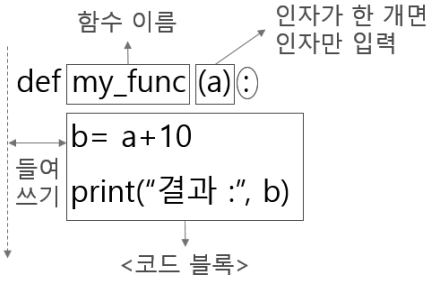
- 인자: 입력값
  - 인자를 통해 함수에 값을 전달
- 반환 값: 결괏값
- 함수 정의: 함수 만들기
- 함수 호출: 정의된 함수 부르기
- 함수의 구조

```
def 함수명([인자1, 인자2, … , 인자n]):
  <코드 블록>
  [return <반환 값>]
```

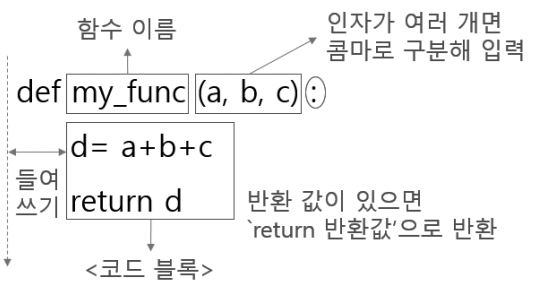
- 함수의 형태



인자도 반환 값도 없는 함수



인자는 있으나 반환 값이 없는 함수



인자도 있고 반환 값도 있는 함수

인자도 반환 값도 없는 함수

- 함수 정의

```
In: def my_func():
        print("My first function!")
        print("This is a function.")
```

- 함수 호출

```
In: my_func()

Out: My first function!
     This is a function.
```

인자는 있으나 반환 값이 없는 함수

- 함수 정의

```
In: def my_friend(friendName):
        print("{}는 나의 친구입니다.".format(friendName))
```

- 함수 호출

```
In: my_friend("철수")
    my_friend("영미")

Out: 철수는 나의 친구입니다.
     영미는 나의 친구입니다.
```

인자도 있고 반환 값도 있는 함수

- 함수 정의

```
In: def my_calc(x,y):
        z = x*y
        return z
```

- 함수 호출

```
In: my_calc(3,4)

Out: 12
```

변수의 유효 범위

- 함수 안에서 정의한(혹은 생성한) 변수는 함수 안에서만 사용할 수 있음
- 함수 안에서 생성한 변수는 함수를 호출해 실행되는 동안만 사용할 수 있고 함수 실행이 끝나면 더는 사용할 수 없음
- 지역 변수(local variable)
- 전역 변수(global variable)
- 이름 공간: 변수를 정의할 때 변수가 저장되는 공간
- 유효 범위(scope)
  - 지역 영역(local scope): 지역 변수를 저장하는 이름 공간
  - 전역 영역(global scope): 전역 변수를 저장하는 이름 공간
  - 내장 영역(built-in scope): 파이썬 자체에서 정의한 이름 공간
- 스코핑 룰(Scoping rule), LGB 룰(Local/Global/Built-in rule)

변수의 유효 범위

- 동일한 변수명을 지역 변수와 전역 변수에 모두 이용하면 스코핑 룰에 따라 변수가 선택됨
- 같은 이름의 변수를 지역 변수와 전역 변수로 모두 사용한 예

```
In: a = 5 # 전역 변수
    def func1():
        a = 1 # 지역 변수. func1()에서만 사용
        print("[func1] 지역 변수 a =", a)
    def func2():
        a = 2 # 지역 변수. func2()에서만 사용
        print("[func2] 지역 변수 a =", a)

    def func3():
        print("[func3] 전역 변수 a =", a)

    def func4():
        global a   # 함수 내에서 전역 변수를 변경하기 위해 선언
        a = 4      # 전역 변수의 값 변경
        print("[func4] 전역 변수 a =",a)
```

변수의 유효 범위

- 앞서 정의한 각 함수를 호출하는 예

```
In: func1() #함수 func1() 호출
    func2() #함수 func2() 호출
    print("전역 변수 a =", a) # 전역 변수 출력

Out: [func1] 지역 변수 a = 1
     [func2] 지역 변수 a = 2
     전역 변수 a = 5
```

```
In: func3() #함수 func3() 호출
    func4() #함수 func4() 호출
    func3() #함수 func3() 호출

Out: [func3] 전역 변수 a = 5
     [func4] 전역 변수 a = 4
     [func3] 전역 변수 a = 4
```

람다(lambda) 함수

- 한 줄로 함수를 표현
- 람다 함수의 기본 구조

```
lambda <인자> : <인자 활용 수행 코드>
```

- 람다 함수의 사용

```
(lambda <인자> : <인자 활용 수행 코드>) (<인자>)
```

```
lambda_function = lambda <인자> : <인자 활용 수행 코드>
lambda_function(<인자>)
```

- 람다 함수 호출

```
In: (lambda x : x**2) (3)

Out: 9
```

람다(lambda) 함수

- 람다 함수를 변수에 할당한 후에 인자를 입력해서 호출

```
In: mySquare = lambda x : x**2
    mySquare(2)

Out: 4
```

- 여러 개의 인자를 입력받아 연산 결과를 반환하는 람다 함수

```
In: mySimpleFunc = lambda x,y,z : 2*x + 3*y + z
    mySimpleFunc(1,2,3)

Out: 11
```

유용한 내장 함수

- 형 변환 함수
  - 정수형으로 변환: int()

```
In: [int(0.123), int(3.5123456), int(-1.312367)]

Out: [0, 3, -1]
```

  - 실수형으로 변환: float()

```
In: [float(0), float(123), float(-567)]

Out: [0.0, 123.0, -567.0]
```

  - 문자형으로 변환

```
In: [str(123), str(459678), str(-987)]

Out: ['123', '459678', '-987']
```

유용한 내장 함수

- 형 변환 함수
  - 리스트, 튜플, 세트형으로 변환

내장 함수	기능	사용 예
list()	튜플/세트 데이터를 리스트로 변환	list((1,2,3)), list({1,2,3})
tuple()	리스트/세트 데이터를 튜플로 변환	tuple([(1,2,3]), tuple({1,2,3})
set()	리스트/튜플 데이터를 세트로 변환	set([1,2,3]), set((1,2,3))

```
In: list_data = ['abc', 1, 2, 'def']
    tuple_data = ('abc', 1, 2, 'def')
    set_data = {'abc', 1, 2, 'def'}

# 각 데이터 타입 확인
In: [type(list_data), type(tuple_data), type(set_data)]
Out: [list, tuple, set]

# list() 함수를 이용해 리스트로 변환
In: print("리스트로 변환: ", list(tuple_data), list(set_data))
Out: 리스트로 변환:  ['abc', 1, 2, 'def'] ['abc', 2, 'def', 1]
```

유용한 내장 함수

- 형 변환 함수
  - 리스트, 튜플, 세트형으로 변환

```
# tuple() 함수로 튜플로 변환하겠습니다.
In: print("튜플로 변환:", tuple(list_data), tuple(set_data))
Out: 튜플로 변환: ('abc', 1, 2, 'def') ('abc', 2, 'def', 1)

# set() 함수를 이용해 세트로 변환
In: print("세트로 변환:", set(list_data), set(tuple_data))
Out: 세트로 변환: {1, 2, 'abc', 'def'} {1, 2, 'abc', 'def'}
```

유용한 내장 함수

- bool 함수
  - True 혹은 False의 결과를 반환
- 숫자를 인자로 bool 함수 호출
  - 숫자 0이면 False, 0 이외의 숫자이면 True를 반환

```
In: bool(0) # 인자: 숫자 0
Out: False

# 0 이외의 숫자를 인자로 삼아 bool() 함수를 호출
In: bool(1) # 인자: 양의 정수
Out: True

In: bool(-10) # 인자: 음의 정수
Out: True

In: bool(5.12) # 인자: 양의 실수
Out: True

In: bool(-3.26) # 인자: 음의 실수
Out: True
```

유용한 내장 함수

- 문자열을 인자로 bool 함수 호출
  - 문자열이 있으면 True를 반환, 없으면 False를 반환

```
In: bool('a') # 인자: 문자열 'a'
Out: True

In: bool(' ') # 인자: 빈 문자열(공백)
Out: True

In: bool('') # 인자: 문자열 없음
Out: False

In: bool(None) #인자: None
Out: False
```

유용한 내장 함수

- 리스트, 튜플, 세트를 인자로 bool 함수 호출
  - 항목이 있으면 True, 없으면 False를 반환

```
In: myFriends = []
    bool(myFriends) # 인자: 항목이 없는 빈 리스트
Out: False

# 항목이 있는 리스트를 인자로 bool() 함수를 호출
In: myFriends = ['James', 'Robert', 'Lisa', 'Mary']
    bool(myFriends) # 인자: 항목이 있는 리스트
Out: True

# 항목이 없는 튜플과 있는 튜플을 인자로 bool() 함수를 호출
In: myNum = ()
    bool(myNum) # 인자: 항목이 없는 빈 튜플
Out: False

In: myNum = (1,2,3)
    bool(myNum) # 인자: 항목이 있는 튜플
Out: True
```

유용한 내장 함수

- 리스트, 튜플, 세트를 인자로 bool 함수 호출
  - 항목이 있으면 True, 없으면 False를 반환

```
# 항목 유무에 따라 다음과 같이 각각 False와 True를 반환
In: mySetA = {}
    bool(mySetA) # 인자: 항목이 없는 빈 세트
Out: False

In: mySetA = {10,20,30}
    bool(mySetA) # 인자: 항목이 있는 세트
Out: True
```

유용한 내장 함수

- bool 함수의 활용
  - name 인자에 문자열이 있으면 이름을 출력하고, 없으면 입력된 문자열이 없다고 출력하는 예

```
In: def print_name(name):
        if bool(name):
            print("입력된 이름:", name)
        else:
            print("입력된 이름이 없습니다.")

# print_name() 함수 호출
In: print_name("James")
Out: 입력된 이름: James

In: print_name("")
Out: 입력된 이름이 없습니다.
```

유용한 내장 함수

- 최솟값과 최댓값을 구하는 함수
  - 내장 함수 min()과 max()를 이용

```
In: myNum = [10, 5, 12, 0, 3.5, 99.5, 42]
    [min(myNum), max(myNum)]
Out: [0, 99.5]

# 문자열에 대해서도 최솟값과 최댓값을 구하는 예
In: myStr = 'zxyabc'
    [min(myStr), max(myStr)]
Out: ['a', 'z']

# 각각 튜플과 세트에서 최솟값과 최댓값을 구하는 예
In: myNum = (10, 5, 12, 0, 3.5, 99.5, 42)
    [min(myNum), max(myNum)]
Out: [0, 99.5]

In: myNum = {"Abc", "abc", "bcd", "efg"}
    [min(myNum), max(myNum)]
Out: ['Abc', 'efg']
```

유용한 내장 함수

- 절댓값과 전체 합을 구하는 함수
  - 절댓값: 내장 함수 abs()를 이용

```
In: [abs(10), abs(-10)]
Out: [10, 10]

In: [abs(2.45), abs(-2.45)]
Out: [2.45, 2.45]
```

  - 합계: 내장 함수 sum()을 이용

```
In: sumList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    sum(sumList)

Out: 55
```

유용한 내장 함수

- 항목의 개수를 구하는 함수
  - 내장 함수 len()을 이용

```
In: len("ab cd") # 문자열
Out: 5

In: len([1, 2, 3, 4, 5, 6, 7, 8]) # 리스트
Out: 8

In: len((1, 2, 3, 4, 5)) # 튜플
Out: 5

In: len({'a', 'b', 'c', 'd'}) # 세트
Out: 4

In: len({1:"Thomas", 2:"Edward", 3:"Henry"}) # 딕셔너리
Out: 3
```

내장 함수의 활용

- 시험 점수가 입력된 리스트를 대상으로 sum()과 len()을 이용해 데이터 항목의 총합과 길이, 평균값 계산

```
In: scores = [90, 80, 95, 85] # 과목별 시험 점수
    print("총점:{0}, 평균:{1}".format(sum(scores), sum(scores)/len(scores)))
Out: 총점:350, 평균:87.5
```

- 시험 점수 중 최고점과 최하점 계산

```
In: print("최하 점수:{0}, 최고 점수:{1}".format(min(scores), max(scores)))
Out: 최하 점수:80, 최고 점수:95
```

8

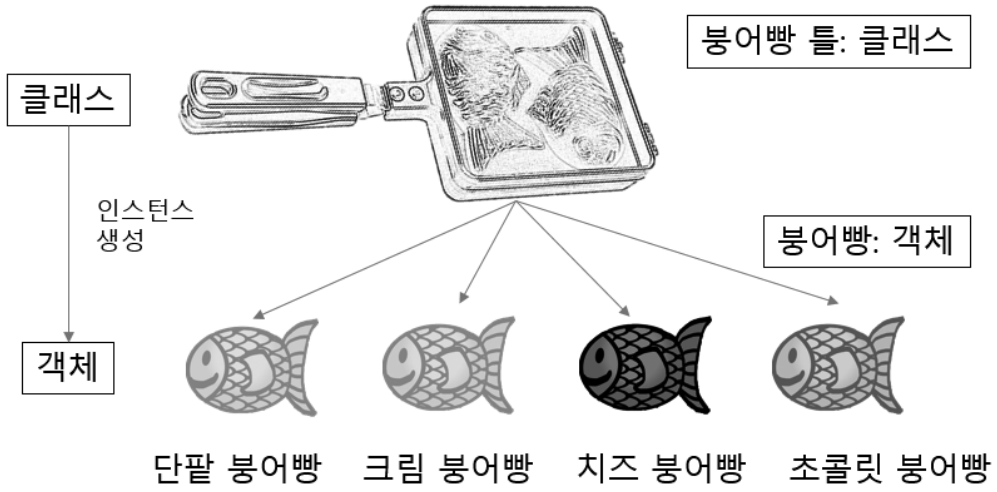
객체와 클래스

클래스 선언과 객체 선언

- 객체란?
  - 속성(상태, 특징)과 행위(행동, 동작, 기능)로 구성된 대상
  - 속성: 객체의 특징. 변수로 구현
  - 행동: 객체가 할 수 있는 일. 함수로 구현
  - 객체는 변수와 함수의 묶음
- 객체지향 프로그래밍 언어, 객체지향 언어
  - Object- Oriented Programming, OOP
  - 객체를 만들고 이용할 수 있는 기능을 제공하는 프로그래밍 언어

클래스 선언

- 객체를 만들려면 먼저 클래스를 선언해야 함
- 객체의 공통된 속성과 행위를 변수와 함수로 정의한 것
- 클래스는 객체를 만들기 위한 기본 틀, 객체는 기본 틀을 바탕으로 만들어진 결과
- 객체는 클래스에서 생성하므로 객체를 클래스의 인스턴스(Instance)라고 함
- 클래스와 객체의 관계



클래스 선언

- 클래스 선언을 위한 기본 구조

```
class 클래스명():
    [변수1]  # 클래스 변수
    [변수2]
    …
    def 함수1(self[, 인자1, 인자2, … , 인자n]): # 클래스 함수
        <코드 블록>
        …
    def 함수2(self[, 인자1, 인자2, … , 인자n]):
        <코드 블록>
        …
```

객체 생성 및 활용

- 자전거 클래스
  - 자전거의 속성: 바퀴 크기(wheel_size), 색상(color)
  - 자전거의 동작: 이동(move), 좌/우회전(turn), 정지(stop)
- 자전거 클래스 선언

```
In: class Bicycle(): # 클래스 선언
        pass
```

- 객체를 생성하는 방법

```
객체명 = 클래스명()
```

```
In: my_bicycle
Out: <__main__.Bicycle at 0x1dd5bdfc240>
```

- 객체에 속성을 설정

```
객체명.변수명 = 속성값
```

```
In: my_bicycle.wheel_size = 26
    my_bicycle.color = 'black'
```

객체 생성 및 활용

- 객체의 변수에 접근해서 객체의 속성 가져오기

```
객체명.변수명
```

```
In: print("바퀴 크기:", my_bicycle.wheel_size)  # 객체의 속성 출력
    print("색상:", my_bicycle.color)

Out: 바퀴 크기: 26
     색상: black
```

- 클래스에 함수 추가하기

```
In: class Bicycle():

        def move(self, speed):
            print("자전거: 시속 {0}킬로미터로 전진".format(speed))

        def turn(self, direction):
            print("자전거: {0}회전".format(direction))

        def stop(self):
            print("자전거({0}, {1}): 정지 ".format(self.wheel_size, self.color))
```

객체 생성 및 활용

- 객체의 메서드 호출

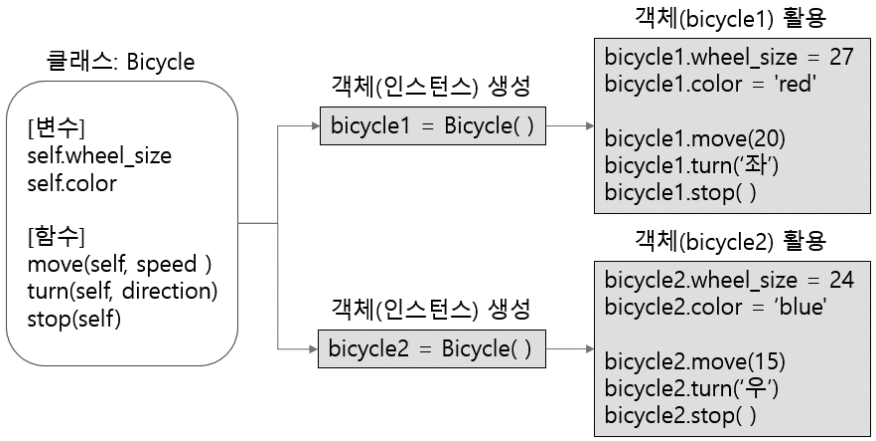
```
객체명.메서드명([인자1, 인자2, … , 인자n])
```

- Bicycle 클래스의 객체를 생성한 후 속성을 설정하고 메서드를 호출하는 예

```
In: my_bicycle = Bicycle() # Bicycle 클래스의 인스턴스인 my_bicycle 객체 생성
    my_bicycle.wheel_size = 26 # 객체의 속성 설정
    my_bicycle.color = 'black'
    my_bicycle.move(30)  # 객체의 메서드 호출
    my_bicycle.turn('좌')
    my_bicycle.stop()

Out: 자전거: 시속 30킬로미터로 전진
     자전거: 좌회전
     자전거(26, black): 정지
```

객체 생성 및 활용

- 클래스의 선언, 객체의 생성 및 활용 방법




```
In: bicycle1 = Bicycle() # Bicycle 클래스의 인스턴스인 bicycle1 객체 생성
    bicycle1.wheel_size = 27 # 객체의 속성 설정
    bicycle1.color = 'red'
    bicycle1.move(20)
    bicycle1.turn('좌')
    bicycle1.stop()

Out: 자전거: 시속 20킬로미터로 전진
     자전거: 좌회전
     자전거(27, red): 정지
```

객체 생성 및 활용

- 클래스의 선언, 객체의 생성 및 활용 방법

```
In: bicycle2 = Bicycle() # Bicycle 클래스의 인스턴스인 bicycle2 객체 생성
    bicycle2.wheel_size = 24 # 객체의 속성 설정
    bicycle2.color = 'blue'
    bicycle2.move(15)
    bicycle2.turn('우')
    bicycle2.stop()

Out: 자전거: 시속 15킬로미터로 전진
     자전거: 우회전
     자전거(24, blue): 정지
```

객체 초기화

- 초기화 함수 __init__()를 구현하면 객체를 생성하는 것과 동시에 속성값을 지정할 수 있음
- __init__() 함수는 클래스의 인스턴스가 생성될 때(즉, 객체가 생성될 때) 자동으로 실행됨

```
In: class Bicycle():

        def __init__(self, wheel_size, color):
            self.wheel_size = wheel_size
            self.color = color

        def move(self, speed):
            print("자전거: 시속 {0}킬로미터로 전진".format(speed))

        def turn(self, direction):
            print("자전거: {0}회전".format(direction))

        def stop(self):
            print("자전거({0}, {1}): 정지 ".format(self.wheel_size, self.color))
```

객체 초기화

- 객체를 생성할 때 속성값을 지정해서 초기화

```
객체명 = 클래스명(인자1, 인자2, 인자3, … , 인자n)
```

```
In: my_bicycle = Bicycle(26, 'black') # 객체 생성과 동시에 속성값을 지정
    my_bicycle.move(30)  # 객체 메서드 호출
    my_bicycle.turn('좌')
    my_bicycle.stop()

Out: 자전거: 시속 30킬로미터로 전진
     자전거: 좌회전
     자전거(26, black): 정지
```

클래스를 구성하는 변수와 함수

- 클래스에서 사용하는 변수
  - 위치에 따라 클래스 변수(class variable)와 인스턴스 변수(instance variable)로 구분
  - 클래스 변수: 클래스 내에 있지만 함수 밖에서 '변수명 = 데이터' 형식으로 정의한 변수
    - 클래스에서 생성한 모든 객체가 공통으로 사용 가능
    - '클래스명.변수명' 형식으로 접근
  - 인스턴스 변수: 클래스 내의 함수 안에서 'self.변수명= 데이터' 형식으로 정의한 변수
    - 클래스 내의 모든 함수에서 'self.변수명'으로 접근
    - 각 인스턴스(객체)에서 개별적으로 관리하며, 객체를 생성한 후에 '객체명.변수명' 형식으로 접근

클래스를 구성하는 변수와 함수

- 클래스 변수와 인스턴스 변수를 사용한 자동차 클래스

```
In: class Car():
        instance_count = 0 # 클래스 변수 생성 및 초기화

        def __init__(self, size, color):
            self.size = size     # 인스턴스 변수 생성 및 초기화
            self.color = color  # 인스턴스 변수 생성 및 초기화
            Car.instance_count = Car.instance_count + 1 # 클래스 변수 이용
            print("자동차 객체의 수: {0}".format(Car.instance_count))

        def move(self):
            print("자동차({0} & {1})가 움직입니다.".format(self.size, self.color))
```

```
In: car1 = Car('small', 'white')
    car2 = Car('big', 'black')

Out: 자동차 객체의 수: 1
     자동차 객체의 수: 2
```

클래스를 구성하는 변수와 함수

- 클래스 변수와 인스턴스 변수를 사용한 자동차 클래스

```
In: print("Car 클래스의 총 인스턴스 개수:{}".format(Car.instance_count))
Out: Car 클래스의 총 인스턴스 개수:2
```

```
In: print("Car 클래스의 총 인스턴스 개수:{}".format(car1.instance_count))
    print("Car 클래스의 총 인스턴스 개수:{}".format(car2.instance_count))
Out: Car 클래스의 총 인스턴스 개수:2
     Car 클래스의 총 인스턴스 개수:2
```

```
In: car1.move()
    car2.move()
Out: 자동차(small & white)가 움직입니다.
     자동차(big & black)가 움직입니다.
```

클래스를 구성하는 변수와 함수

- 이름이 같은 클래스 변수와 인스턴스 변수가 있는 클래스를 정의한 경우

```
In: class Car2():
        count = 0; # 클래스 변수 생성 및 초기화

        def __init__(self, size, num):
            self.size = size     # 인스턴스 변수 생성 및 초기화
            self.count = num  # 인스턴스 변수 생성 및 초기화
            Car2.count = Car2.count + 1 # 클래스 변수 이용
            print("자동차 객체의 수: Car2.count = {0}".format(Car2.count))
            print("인스턴스 변수 초기화: self.count = {0}".format(self.count))

        def move(self):
            print("자동차({0} & {1})가 움직입니다.".format(self.size, self.count))
```

클래스를 구성하는 변수와 함수

- 이름이 같은 클래스 변수와 인스턴스 변수가 있는 클래스를 정의한 경우

```
In: car1 = Car2("big", 20)
    car1 = Car2("small", 30)

Out: 자동차 객체의 수: Car2.count = 1
     인스턴스 변수 초기화: self.count = 20
     자동차 객체의 수: Car2.count = 2
     인스턴스 변수 초기화: self.count = 30
```

클래스를 구성하는 변수와 함수

- 클래스에서 사용하는 함수
  - 기능과 사용법에 따라 인스턴스 메서드(instance method), 정적 메서드(static method), 클래스 메서드(class method)로 구분
- 인스턴스 메서드
  - 각 객체에서 개별적으로 동작하는 함수를 만들고자 할 때 사용
  - 함수를 정의할 때 첫 인자로 self가 필요
  - 인스턴스 메서드 안에서는 'self.함수명()' 형식으로 클래스 내의 다른 함수를 호출
  - 인스턴스 메서드의 구조

```
class 클래스명():
    def 함수명(self[, 인자1, 인자2, … , 인자n]):
        self.변수명1 = 인자1
        self.변수명2 = 인자2
        self.변수명3 = 데이터
        …
        <코드 블록>
```

클래스를 구성하는 변수와 함수

- 인스턴스 메서드
  - 인스턴스 메서드의 사용

```
객체명 = 클래스명()
객체명.메서드명([인자1, 인자2, … , 인자n])
```

  - 인스턴스 메서드를 사용한 자동차 클래스

```
In: # Car 클래스 선언
    class Car():
        instance_count = 0 # 클래스 변수 생성 및 초기화

        # 초기화 함수(인스턴스 메서드)
        def __init__(self, size, color):
            self.size = size     # 인스턴스 변수 생성 및 초기화
            self.color = color  # 인스턴스 변수 생성 및 초기화
            Car.instance_count = Car.instance_count + 1 # 클래스 변수 이용
            print("자동차 객체의 수: {0}".format(Car.instance_count))
```

클래스를 구성하는 변수와 함수

- 인스턴스 메서드를 사용한 자동차 클래스

```
    # 인스턴스 메서드
    def move(self, speed):
        self.speed = speed  # 인스턴스 변수 생성
        print("자동차({0} & {1})가 ".format(self.size, self.color), end='')
        print("시속 {0}킬로미터로 전진".format(self.speed))

    # 인스턴스 메서드
    def auto_cruise(self):
        print("자율 주행 모드")
        self.move(self.speed) # move() 함수의 인자로 인스턴스 변수를 입력
```

클래스를 구성하는 변수와 함수

- 객체를 생성하고 인스턴스 메서드를 사용하는 예

```
In: car1 = Car("small", "red") # 객체 생성 (car1)
    car2 = Car("big", "green") # 객체 생성 (car2)

    car1.move(80) #객체(car1)의 move() 메서드 호출
    car2.move(100) #객체(car2)의 move() 메서드 호출

    car1.auto_cruise() #객체(car1)의 auto_cruise() 메서드 호출
    car2.auto_cruise() #객체(car2)의 auto_cruise() 메서드 호출

Out: 자동차 객체의 수: 1
     자동차 객체의 수: 2
     자동차(small & red)가 시속 80킬로미터로 전진
     자동차(big & green)가 시속 100킬로미터로 전진
     자율 주행 모드
     자동차(small & red)가 시속 80킬로미터로 전진
     자율 주행 모드
     자동차(big & green)가 시속 100킬로미터로 전진
```

클래스를 구성하는 변수와 함수

- 정적 메서드
  - 클래스와 관련이 있어서 클래스 안에 두기는 하지만 클래스나 클래스의 인스턴스(객체)와는 무관하게 독립적으로 동작하는 함수를 만들고 싶을 때 이용하는 함수
  - 함수를 정의할 때 인자로 self를 사용하지 않으며 정적 메서드 안에서는 클래스나 클래스 변수에 접근할 수 없음
  - 함수 앞에 데코레이터(Decorator)인 @staticmethod를 선언해 정적 메서드임을 표시
- 정적 메서드의 구조

```
class 클래스명():
    @staticmethod
    def 함수명([인자1, 인자2, … , 인자n]):
        <코드 블록>
```

- 정적 메서드 호출

```
클래스명.메서드명([인자1, 인자2, … , 인자n]):
```

클래스를 구성하는 변수와 함수

- 정적 메서드를 사용한 예

```
In: # Car 클래스 선언
    class Car():

        # def __init__(self, size, color): => 앞의 코드 활용
        # def move(self, speed): => 앞의 코드 활용
        # def auto_cruise(self): => 앞의 코드 활용

        # 정적 메서드
        @staticmethod
        def check_type(model_code):
            if(model_code >= 20):
                print("이 자동차는 전기차입니다.")
            elif(10 <= model_code < 20):
                print("이 자동차는 가솔린차입니다.")
            else:
                print("이 자동차는 디젤차입니다.")
```

클래스를 구성하는 변수와 함수

- 정적 메서드를 사용한 예

```
In: Car.check_type(25)
    Car.check_type(2)

Out: 이 자동차는 전기차입니다.
     이 자동차는 디젤차입니다.
```

클래스를 구성하는 변수와 함수

- 클래스 메서드
  - 클래스 변수를 사용하기 위한 함수
  - 함수를 정의할 때 첫 번째 인자로 클래스를 넘겨받는 cls가 필요
  - 함수 앞에 데코레이터인 @classmethod를 지정
- 클래스 메서드의 구조

```
class 클래스명():
    @classmethod
    def 함수명(cls[, 인자1, 인자2, … , 인자n]):
        <코드 블록>
```

- 클래스 메서드를 호출하는 방법

```
클래스명.메서드명([인자1, 인자2, … , 인자n]):
```

클래스를 구성하는 변수와 함수

- 클래스 메서드를 사용하는 예

```
In: # Car 클래스 선언
    class Car():
        instance_count = 0 # 클래스 변수

        # 초기화 함수(인스턴스 메서드)
        def __init__(self, size, color):
            self.size = size     # 인스턴스 변수
            self.color = color  # 인스턴스 변수
            Car.instance_count = Car.instance_count + 1

        # def move(self, speed): => 앞의 코드 활용
        # def auto_cruise(self): => 앞의 코드 활용
        # @staticmethod
        # def check_type(model_code): => 앞의 코드 활용

        # 클래스 메서드
        @classmethod
        def count_instance(cls):
            print("자동차 객체의 개수: {0}".format(cls.instance_count))
```

클래스를 구성하는 변수와 함수

- 클래스 메서드를 사용하는 예

```
In: Car.count_instance()          # 객체 생성 전에 클래스 메서드 호출
    car1 = Car("small", "red")    # 첫 번째 객체 생성
    Car.count_instance()          # 클래스 메서드 호출
    car2 = Car("big", "green")    # 두 번째 객체 생성
    Car.count_instance()          # 클래스 메서드 호출

Out: 자동차 객체의 개수: 0
     자동차 객체의 개수: 1
     자동차 객체의 개수: 2
```

객체와 클래스를 사용하는 이유

- 코드 작성과 관리가 편하기 때문
- 규모가 큰 프로그램을 만들 때 클래스와 객체를 많이 이용
- 유사한 객체가 많은 프로그램을 만들 때도 주로 클래스와 객체를 이용해 코드를 작성

객체와 클래스를 사용하는 이유

- 컴퓨터 게임의 로봇 예제
  - 로봇의 속성과 동작
    - 로봇의 속성: 이름, 위치
    - 로봇의 동작: 한 칸 이동
- 클래스와 객체를 사용하지 않는 코드

```
In: robot_name = 'R1'   # 로봇 이름
    robot_pos = 0     # 로봇의 초기 위치

    def robot_move():
        global robot_pos
        robot_pos = robot_pos + 1
        print("{0} position: {1}".format(robot_name, robot_pos))
```

```
In: robot_move()

Out: R1 position: 1
```

객체와 클래스를 사용하는 이유

- 클래스와 객체를 사용하지 않는 코드
  - 로봇을 추가해 두 대의 로봇을 구현

```
In: robot1_name = 'R1'    # 로봇 이름
    robot1_pos = 0        # 로봇의 초기 위치
    def robot1_move():
        global robot1_pos
        robot1_pos = robot1_pos + 1
        print("{0} position: {1}".format(robot1_name, robot1_pos))
    robot2_name = 'R2'    # 로봇 이름
    robot2_pos = 10       # 로봇의 초기 위치
    def robot2_move():
        global robot2_pos
        robot2_pos = robot2_pos + 1
        print("{0} position: {1}".format(robot2_name, robot2_pos))
```

```
In: robot1_move()
    robot2_move()

Out: R1 position: 1
     R2 position: 11
```

객체와 클래스를 사용하는 이유

- 클래스와 객체를 사용하는 코드

```
In: class Robot():
        def __init__(self, name, pos):
            self.name = name        # 로봇 객체의 이름
            self.pos = pos          # 로봇 객체의 위치

        def move(self):
            self.pos = self.pos + 1
            print("{0} position: {1}".format(self.name, self.pos))
```

```
In: robot1 = Robot('R1', 0)
    robot2 = Robot('R2', 10)
```

```
In: robot1.move()
    robot2.move()

Out: R1 position: 1
     R2 position: 11
```

객체와 클래스를 사용하는 이유

- 클래스와 객체를 사용하는 코드

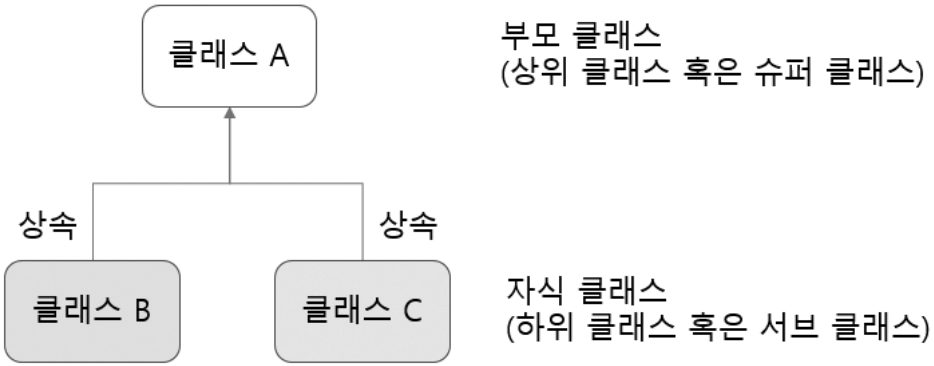
```
In: myRobot3 = Robot('R3', 30)
    myRobot4 = Robot('R4', 40)
    myRobot3.move()
    myRobot4.move()

Out: R3 position: 31
     R4 position: 41
```

클래스 상속

- 상속: 이미 만들어진 클래스의 변수와 함수를 그대로 이어받고 새로운 내용만 추가해서 클래스를 선언
- 상속관계에 있는 두 클래스는 자식이 부모의 유전적 형질을 이어받는 관계와 유사하기 때문에 흔히 부모 자식과의 관계로 표현
- 부모 클래스와 자식 클래스
  - 부모 클래스: 상위 클래스 혹은 슈퍼클래스
  - 자식 클래스: 하위 클래스 혹은 서브 클래스
- 자식 클래스가 부모 클래스로부터 상속을 받으면 자식 클래스는 부모 클래스의 속성(변수)과 행위(함수)를 그대로 이용 가능
- 상속 후에는 자식 클래스만 갖는 속성과 행위를 추가할 수 있음

클래스 상속

- 부모 클래스와 자식 클래스의 관계



※ 클래스 B와 클래스 C는 클래스 A의 자식 클래스로 클래스 A의 변수와 함수를 그대로 이용할 수 있으며 각각 필요한 변수와 함수를 추가할 수도 있음.

- 부모 클래스를 상속받는 자식 클래스를 선언하는 형식

```
class 자식 클래스 이름(부모 클래스 이름):
    <코드 블록>
```

클래스 상속

- Bicycle을 상속하는 FoldingBicycle 클래스

```
In: class FoldingBicycle(Bicycle):

        def __init__(self, wheel_size, color, state): # FoldingBicycle 초기화
            Bicycle.__init__(self, wheel_size, color) # Bicycle의 초기화 재사용
            #super().__init__(wheel_size, color) # super()도 사용 가능
            self.state = state  # 자식 클래스에서 새로 추가한 변수

        def fold(self):
            self.state = 'folding'
            print("자전거: 접기, state = {0}".format(self.state))

        def unfold(self):
            self.state = 'unfolding'
            print("자전거: 펴기, state = {0}".format(self.state))
```

클래스 상속

- FoldingBicycle 클래스의 인스턴스를 생성한 후 메서드 호출

```
In: folding_bicycle = FoldingBicycle(27, 'white', 'unfolding') # 객체 생성
    folding_bicycle.move(20)        # 부모 클래스의 함수(메서드) 호출
    folding_bicycle.fold()          # 자식 클래스에서 정의한 함수 호출
    folding_bicycle.unfold()

Out: 자전거: 시속 20킬로미터로 전진
     자전거: 접기, state = folding
     자전거: 펴기, state = unfolding
```

9 문자열과 텍스트 파일 데이터 다루기

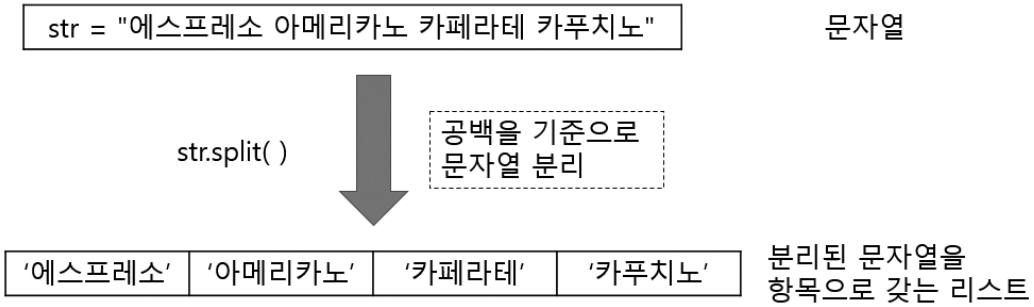
문자열 다루기

- 큰따옴표("")나 작은따옴표('') 안에 들어 있는 문자의 집합
- 텍스트 파일의 내용은 문자열이 됨
- 문자열 처리
  - 문자열 분리
  - 불필요한 문자열 제거
  - 문자열 연결 등

문자열 다루기

- 문자열 분리하기
  - split() 메서드 이용

```
str.split([sep])
```



  - split() 사용 예

```
In: coffee_menu_str = "에스프레소,아메리카노,카페라테,카푸치노"
    coffee_menu_str.split(',')
Out: ['에스프레소', '아메리카노', '카페라테', '카푸치노']
```

```
In: "에스프레소,아메리카노,카페라테,카푸치노".split(',')
Out: ['에스프레소', '아메리카노', '카페라테', '카푸치노']
```

문자열 다루기

- split() 사용 예

```
In: "에스프레소 아메리카노 카페라테 카푸치노".split(' ')
Out: ['에스프레소', '아메리카노', '카페라테', '카푸치노']
```

```
In: "에스프레소 아메리카노 카페라테 카푸치노".split()
Out: ['에스프레소', '아메리카노', '카페라테', '카푸치노']
```

```
In: "  에스프레소  ₩n₩n  아메리카노  ₩n  카페라테   카푸치노 ₩n₩n".split()
Out: ['에스프레소', '아메리카노', '카페라테', '카푸치노']
```

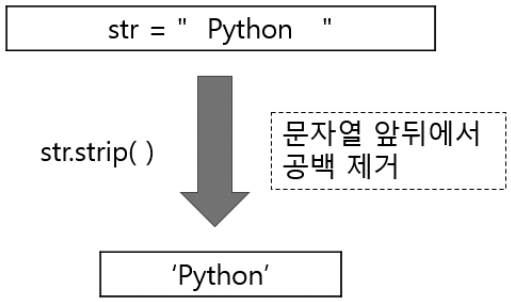
```
In: "에스프레소 아메리카노 카페라테 카푸치노".split(maxsplit=2)
Out: ['에스프레소', '아메리카노', '카페라테 카푸치노']
```

```
In: phone_number = "+82-01-2345-6789"   # 국가 번호가 포함된 전화번호
    split_num = phone_number.split("-", 1)   # 국가 번호와 나머지 번호 분리
    print(split_num)
    print("국내전화번호: {0}".format(split_num[1]))

Out: ['+82', '01-2345-6789']
     국내전화번호: 01-2345-6789
```

문자열 다루기

- 필요 없는 문자열 삭제하기
  - strip() 메서드 이용

```
str.strip([chars])
```



  - strip() 사용 예

```
In: "aaaaPythonaaa".strip('a')
Out: 'Python'
```

```
In: test_str = "aaabbPythonbbbaa"
    temp1 = test_str.strip('a') # 문자열 앞뒤의 'a' 제거
    temp1
Out: 'bbPythonbbb'
```

문자열 다루기

- strip() 사용 예

```
In: temp1.strip('b') # 문자열 앞뒤의 'b' 제거
Out: 'Python'
```

```
In: test_str.strip('ab') # 문자열 앞뒤의 'a'와 'b' 제거
Out: 'Python'
```

```
In: test_str.strip('ba')
Out: 'Python'
```

```
In: test_str_multi = "##***!!!##....  Python is powerful.!...  %%!#..   "
    test_str_multi.strip('*.#! %')
Out: 'Python is powerful'
```

```
In: "  Python   ".strip(' ')
Out: 'Python'
```

```
In: "₩n  Python  ₩n₩n".strip(' ₩n')
Out: 'Python'
```

```
In: "₩n  Python  ₩n₩n".strip()
Out: 'Python'
```

문자열 다루기

- strip() 사용 예

```
In: "aaaBallaaa".strip('a')
Out: 'Ball'
```

```
In: "₩n This is very ₩n fast. ₩n₩n".strip()
Out: 'This is very ₩n fast.'
```

```
In: str_lr = "000Python is easy to learn.000"
    print(str_lr.strip('0'))
    print(str_lr.lstrip('0'))
    print(str_lr.rstrip('0'))
Out: Python is easy to learn.
     Python is easy to learn.000
     000Python is easy to learn.
```

```
In: coffee_menu = "  에스프레소, 아메리카노,    카페라테   , 카푸치노  "
    coffee_menu_list = coffee_menu.split(',')
    coffee_menu_list
Out: ['  에스프레소', ' 아메리카노', '    카페라테   ', ' 카푸치노  ']
```

문자열 다루기

- strip() 사용 예

```
In: coffee_list = [] # 빈 리스트 생성
    for coffee in coffee_menu_list:
        temp = coffee.strip() # 문자열의 공백 제거
        coffee_list.append(temp) # 리스트 변수에 공백이 제거된 문자열 추가

    print(coffee_list) #최종 문자열 리스트 출력
Out: ['에스프레소', '아메리카노', '카페라테', '카푸치노']
```

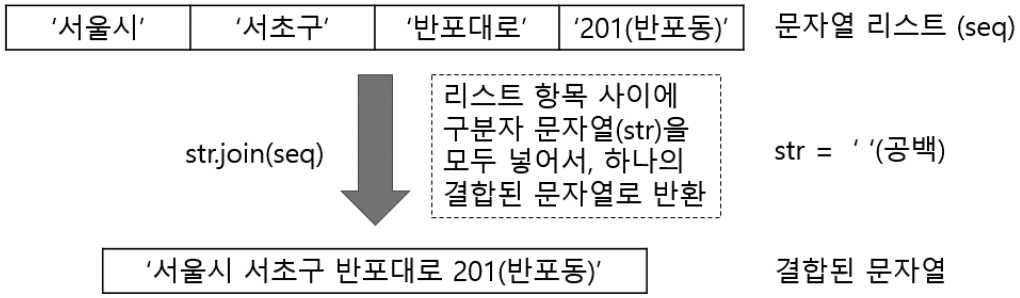
문자열 다루기

- 문자열 연결하기
  - 더하기 연산자(+)로 연결

```
In: name1 = "철수"
    name2 = "영미"
    hello = "님, 주소와 전화 번호를 입력해 주세요."
    print(name1 + hello)
    print(name2 + hello)
Out: 철수님, 주소와 전화 번호를 입력해 주세요.
     영미님, 주소와 전화 번호를 입력해 주세요.
```

  - join() 메서드 이용

```
str.join(seq)
```



문자열 다루기

- join() 메서드 이용 예

```
In: address_list = ["서울시", "서초구", "반포대로", "201(반포동)"]
    address_list
Out: ['서울시', '서초구', '반포대로', '201(반포동)']
```

```
In: a = " "
    a.join(address_list)
Out: '서울시 서초구 반포대로 201(반포동)'
```

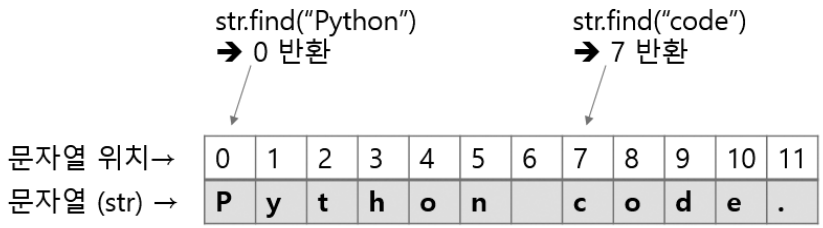
```
In: " ".join(address_list)
Out: '서울시 서초구 반포대로 201(반포동)'
```

```
In: "*^-^*".join(address_list)
Out: '서울시*^-^*서초구*^-^*반포대로*^-^*201(반포동)'
```

문자열 다루기

- 문자열 찾기
  - find() 메서드 이용

```
str.find(search_str)
```




```
In: str_f = "Python code."
    print("찾는 문자열의 위치:", str_f.find("Python"))
    print("찾는 문자열의 위치:", str_f.find("code"))
    print("찾는 문자열의 위치:", str_f.find("n"))
    print("찾는 문자열의 위치:", str_f.find("easy"))
Out: 찾는 문자열의 위치: 0
     찾는 문자열의 위치: 7
     찾는 문자열의 위치: 5
     찾는 문자열의 위치: -1
```

문자열 다루기

- find() 메서드에 시작 위치(start)와 끝 위치(end)를 지정해 검색 범위를 설정

```
str.find(search_str, start, end)
```

- find() 메서드에 시작 위치(start)만 지정해 검색 범위를 설정

```
str.find(search_str, start)
```

```
In: str_f_se = "Python is powerful. Python is easy to learn."
    print(str_f_se.find("Python", 10, 30)) # 시작 위치(start)와 끝 위치(end) 지정
    print(str_f_se.find("Python", 35)) # 찾기 위한 시작 위치(start) 지정
Out: 20
     -1
```

- count()로 문자열 일치 횟수 반환

```
str.count(search_str)
str.count(search_str, start)
str.count(search_str, start, end)
```

문자열 다루기

- count()로 문자열 일치 횟수 반환

```
In: str_c = "Python is powerful. Python is easy to learn. Python is open."
    print("Python의 개수는?:", str_c.count("Python"))
    print("powerful의 개수는?:", str_c.count("powerful"))
    print("IPython의 개수는?:", str_c.count("IPython"))
Out: Python의 개수는?: 3
     powerful의 개수는?: 1
     IPython의 개수는?: 0
```

문자열 다루기

- 문자열이 특정 문자열로 시작하는지 끝나는지 검사
  - startwith(), endwith() 메서드 이용

```
str.startswith(prefix)
str.startswith(prefix, start)
str.startswith(prefix, start, end)
```

```
str.endswith(suffix)
str.endswith(suffix, start)
str.endswith(suffix, start, end)
```

  - startwith(), endwith() 메서드 이용 예

```
In: str_se = "Python is powerful. Python is easy to learn."
    print("Python으로 시작?:", str_se.startswith("Python"))
    print("is로 시작?:", str_se.startswith("is"))
    print(".로 끝?:", str_se.endswith("."))
    print("learn으로 끝?:", str_se.endswith("learn"))
Out: Python으로 시작?: True
     is로 시작?: False
     .로 끝?: True
     learn으로 끝?: False
```

문자열 다루기

- 문자열 바꾸기
  - replace() 메서드 이용

```
str.replace(old, new[, count])
```

  - replace() 메서드 이용 예

```
In: str_a = 'Python is fast. Python is friendly. Python is open.'
    print(str_a.replace('Python', 'IPython'))
    print(str_a.replace('Python', 'IPython', 2))
Out: IPython is fast. IPython is friendly. IPython is open.
     IPython is fast. IPython is friendly. Python is open.
```

```
In: str_b = '[Python] [is] [fast]'
    str_b1 = str_b.replace('[', '') # 문자열에서 '['를 제거
    str_b2 = str_b1.replace(']', '') # 결과 문자열에서 다시 ']'를 제거
    print(str_b)
    print(str_b1)
    print(str_b2)
Out: [Python] [is] [fast]
     Python] is] fast]
     Python is fast
```

문자열 다루기

- 문자열의 구성 확인
  - 문자열이 숫자 또는 문자로만, 숫자와 문자가 모두 포함돼 있는지, 로마자 알파벳 대문자로만 이뤄졌는지, 소문자로만 이뤄졌는지 등을 확인

메서드	설명	사용 예
isalpha()	문자열이 숫자, 특수 문자, 공백이 아닌 문자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.isalpha()
isdigit()	문자열이 모두 숫자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.isdigit()
isalnum()	문자열이 특수 문자나 공백이 아닌 문자와 숫자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.isalnum()
isspace()	문자열이 모두 공백 문자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.isspace()
isupper()	문자열이 모두 로마자 대문자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.isupper()
islower()	문자열이 모두 로마자 소문자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.islower()

문자열 다루기

- 문자열의 구성 확인
  - 메서드 사용 예

```
In: print('Python'.isalpha())    # 문자열에 공백, 특수 문자, 숫자가 없음
    print('Ver. 3.x'.isalpha())  # 공백, 특수 문자, 숫자 중 하나가 있음
Out: True
     False
```

```
In: print('12345'.isdigit())      # 문자열이 모두 숫자로 구성됨
    print('12345abc'.isdigit())   # 문자열이 숫자로만 구성되지 않음
Out: True
     False
```

```
In: print('abc1234'.isalnum())    # 특수 문자나 공백이 아닌 문자와 숫자로 구성됨
    print('   abc1234'.isalnum())  # 문자열에 공백이 있음
Out: True
     False
```

```
In: print('   '.isspace())        # 문자열이 공백으로만 구성됨
    print(' 1 '.isspace())        # 문자열에 공백 외에 다른 문자가 있음
Out: True
     False
```

문자열 다루기

- 메서드 사용 예

```
In: print('PYTHON'.isupper())      # 문자열이 모두 대문자로 구성됨
    print('Python'.isupper())      # 문자열에 대문자와 소문자가 있음
    print('python'.islower())      # 문자열이 모두 소문자로 구성됨
    print('Python'.islower())      # 문자열에 대문자와 소문자가 있음
Out: True
     False
     True
     False
```

문자열 다루기

- 대소문자로 변경하기
  - lower()와 upper() 메서드 이용

```
str.lower()
str.upper()
```

  - lower()와 upper() 메서드 이용 예

```
In: string1 =  'Python is powerful. PYTHON IS EASY TO LEARN.'
    print(string1.lower())
    print(string1.upper())
Out: python is powerful. python is easy to learn.
     PYTHON IS POWERFUL. PYTHON IS EASY TO LEARN.
```

```
In: 'Python' == 'python'
Out: False
```

```
In: print('Python'.lower() == 'python'.lower())
    print('Python'.upper() == 'python'.upper())
Out: True
     True
```

텍스트 파일의 데이터를 읽고 처리하기

- 데이터 파일 준비 및 읽기
  - 데이터: 어느 커피 전문점에서 나흘 동안 기록한 메뉴별 커피 판매량
  - 작업: 나흘 동안 메뉴당 전체 판매량과 하루 평균 판매량 구하기
- 데이터 확인

```
In: !type c:₩myPyCode₩data₩coffeeShopSales.txt
Out: 날짜    에스프레소  아메리카노  카페라테  카푸치노
   10.15       10         50         45        20
   10.16       12         45         41        18
   10.17       11         53         32        25
   10.18       15         49         38        22
```

텍스트 파일의 데이터를 읽고 처리하기

- 데이터 읽기

```
In: # file_name = 'c:₩myPyCode₩data₩coffeeShopSales.txt'
    file_name = 'c:/myPyCode/data/coffeeShopSales.txt'
    f = open(file_name)          # 파일 열기
    for line in f:               # 한 줄씩 읽기
        print(line, end='')      # 한 줄씩 출력
    f.close()                    # 파일 닫기
Out: 날짜    에스프레소  아메리카노  카페라테  카푸치노
     10.15       10         50         45        20
     10.16       12         45         41        18
     10.17       11         53         32        25
     10.18       15         49         38        22
```

텍스트 파일의 데이터를 읽고 처리하기

- 문자열 데이터 처리

```
In: f = open(file_name)           # 파일 열기
    header = f.readline()         # 데이터의 첫 번째 줄을 읽음
    headerList = header.split()   # 첫 줄의 문자열을 분리한 후 리스트로 변환
    espresso = []                 # 커피 종류별로 빈 리스트 생성
    americano = []
    cafelatte = []
    cappucino = []
    for line in f:                # 두 번째 줄부터 데이터를 읽어서 반복적으로 처리
      dataList = line.split()     # 문자열에서 공백을 제거해서 문자열 리스트로 변환
      # 커피 종류별로 정수로 변환한 후, 리스트의 항목으로 추가
      espresso.append(int(dataList[1]))
      americano.append(int(dataList[2]))
      cafelatte.append(int(dataList[3]))
      cappucino.append(int(dataList[4]))

    f.close() # 파일 닫기

    print("{0}: {1}".format(headerList[1], espresso)) # 변수에 할당된 값을 출력
    print("{0}: {1}".format(headerList[2], americano))
    print("{0}: {1}".format(headerList[3], cafelatte))
    print("{0}: {1}".format(headerList[4], cappucino))
Out: 에스프레소: [10, 12, 11, 15]
     아메리카노: [50, 45, 53, 49]
     카페라테: [45, 41, 32, 38]
     카푸치노: [20, 18, 25, 22]
```

텍스트 파일의 데이터를 읽고 처리하기

– 나흘간 메뉴별 전체 판매량과 하루 평균 판매량 계산

```
In: total_sum = [sum(espresso), sum(americano), sum(cafelatte), sum(cappucino)]
    total_mean = [sum(espresso)/len(espresso), sum(americano)/len(americano),
                  sum(cafelatte)/len(cafelatte), sum(cappucino)/len(cappucino) ]
    for k in range(len(total_sum)):
        print('[{0}] 판매량'.format(headerList[k+1]))
        print('- 나흘 전체: {0}, 하루 평균: {1}'.format(total_sum[k], total_mean[k]))
Out: [에스프레소] 판매량
     - 나흘 전체: 48, 하루 평균: 12.0
     [아메리카노] 판매량
     - 나흘 전체: 197, 하루 평균: 49.25
     [카페라테] 판매량
     - 나흘 전체: 156, 하루 평균: 39.0
     [카푸치노] 판매량
     - 나흘 전체: 85, 하루 평균: 21.25
```

10 모듈

모듈

- 모듈(Module)
  - 코드가 저장된 파일
  - 다른 코드에서도 이 파일의 변수, 함수, 클래스를 불러와 이용할 수 있음
- 모듈을 사용하는 이유
  - 모듈로 나누면 코드 작성과 관리가 쉬워진다
  - 이미 작성된 코드를 재사용할 수 있다
  - 공동 작업이 편리해진다
- 모듈 생성 및 호출
  - 모듈 이름은 확장자(.py)를 제외한 파일 이름
  - 모듈이 저장된 위치(경로)에서 파이썬(혹은 IPython) 콘솔 혹은 주피터 노트북을 실행해 코드를 작성하거나 파이썬 코드 파일을 실행
- 모듈 만들기
  - 코드를 '모듈이름.py'로 저장
  - IPython의 내장 마술 명령어인'%%writefile'을 이용

모듈 생성 및 호출

– 모듈을 작성해 작업 폴더('C:₩myPyCode')에 저장하는 예

```
In: %%writefile C:₩myPyCode₩my_first_module.py
    # File name: my_first_module.py
    def my_function():
        print("This is my first module.")
Out: Writing C:₩myPyCode₩my_first_module.py
```

- 모듈 불러오기

```
import 모듈명
```

```
In: import my_first_module
   my_first_module.my_function()
Out: This is my first module.
```

모듈 생성 및 호출

- 모듈 생성

```
In: %%writefile C:₩myPyCode₩modules₩my_area.py
    # File name: my_area.py
    PI = 3.14
    def square_area(a): # 정사각형의 넓이 반환
        return a ** 2
    def circle_area(r): # 원의 넓이 반환
        return PI * r ** 2
Out: Writing C:₩myPyCode₩modules₩my_area.py
```

- 모듈의 변수와 함수 호출

```
In: import my_area                  # 모듈 불러오기

    print('pi =', my_area.PI)       # 모듈의 변수 이용
    print('square area =', my_area.square_area(5)) # 모듈의 함수 이용
    print('circle area =', my_area.circle_area(2))
Out: pi = 3.14
     square area = 25
     circle area = 12.56
```

모듈 생성 및 호출

- 모듈의 변수와 함수 호출
  - 모듈에서 사용 가능한 변수, 함수, 클래스 확인: dir(모듈명)

```
In: dir(my_area)
Out: ['PI',
      '__builtins__',
      '__cached__',
      '__doc__',
      '__file__',
      '__loader__',
      '__name__',
      '__package__',
      '__spec__',
      'circle_area',
      'square_area']
```

모듈 생성 및 호출

- 모듈을 불러오는 다른 형식
  - 모듈의 내용 바로 선언

```
from 모듈명 import 변수명
from 모듈명 import 함수명
from 모듈명 import 클래스명
```

```
In: from my_area import PI # 모듈의 변수 바로 불러오기
    print('pi =', PI) # 모듈의 변수 이용
Out: pi = 3.14
```

```
In: from my_area import square_area
    from my_area import circle_area
    print('square area =', square_area(5)) # 모듈의 함수 이용
    print('circle area =', circle_area(2))
Out: square area = 25
     circle area = 12.56
```

모듈 생성 및 호출

- 모듈의 내용 바로 선언

```
In: from my_area import PI, square_area, circle_area
    print('pi =', PI) # 모듈의 변수 이용
    print('square area =', square_area(5)) # 모듈의 함수 이용
    print('circle area =', circle_area(2))
Out: pi = 3.14
     square area = 25
     circle area = 12.56
```

- 모듈의 모든 변수, 함수, 클래스를 바로 모듈명 없이 바로 이용할 경우

```
from 모듈명 import *
```

```
In: from my_area import *
    print('pi =', PI) # 모듈의 변수 이용
    print('square area =', square_area(5)) # 모듈의 함수 이용
    print('circle area =', circle_area(2))
Out: pi = 3.14
     square area = 25
     circle area = 12.56
```

모듈 생성 및 호출

- 모듈명을 별명으로 선언

```
import 모듈명 as 별명
```

```
from 모듈명 import 변수명 as 별명
from 모듈명 import 함수명 as 별명
from 모듈명 import 클래스명 as 별명
```

```
In: import my_area as area # 모듈명(my_area)에 별명(area)을 붙임

    print('pi =', area.PI) # 모듈명 대신 별명 이용
    print('square area =', area.square_area(5))
    print('circle area =', area.circle_area(2))
Out: pi = 3.14
     square area = 25
     circle area = 12.56
```

모듈 생성 및 호출

- 모듈명을 별명으로 선언

```
In: from my_area import PI as pi
    from my_area import square_area as square
    from my_area import circle_area as circle

    print('pi =', pi) # 모듈 변수의 별명 이용
    print('square area =', square(5)) # 모듈 함수의 별명 이용
    print('circle area =', circle(2))
Out: pi = 3.14
     square area = 25
     circle area = 12.56
```

모듈 실행 방법

- 모듈을 직접 실행하는 경우

```
In: %%writefile C:₩myPyCode₩modules₩my_module_test1.py
    # File name: my_module_test1.py
    def func(a):
        print("입력 숫자:", a)

    func(3)
Out: Writing C:₩myPyCode₩modules₩my_module_test1.py
```

```
In: %run C:₩myPyCode₩modules₩my_module_test1.py
Out: 입력 숫자: 3
```

```
In: import my_module_test1
Out: 입력 숫자: 3
```

모듈 실행 방법

- 모듈을 직접 실행하는 경우

```
In: %%writefile C:₩myPyCode₩modules₩my_module_test1.py
    # File name: my_module_test1.py
    def func(a):
        print("입력 숫자:", a)

    func(3)
Out: Writing C:₩myPyCode₩modules₩my_module_test1.py
```

```
In: %run C:₩myPyCode₩modules₩my_module_test1.py
Out: 입력 숫자: 3
```

```
In: import my_module_test1
Out: 입력 숫자: 3
```

- 모듈을 직접 수행하는 경우와 임포트해서 이용하는 경우를 구분할 수 있는 코드의 구조

```
if __name__ == "__main__":
    <직접 수행할 때만 실행되는 코드>
```

모듈 실행 방법

- 모듈을 직접 수행하는 경우와 임포트해서 이용하는 경우를 구분할 수 있는 코드의 구조

```
In: %%writefile C:₩myPyCode₩modules₩my_module_test2.py
    # File name: my_module_test2.py
    def func(a):
        print("입력 숫자:",a)
    if __name__ == "__main__":
        print("모듈을 직접 실행")
        func(3)
        func(4)
Out: Writing C:₩myPyCode₩modules₩my_module_test2.py
    'my_module_test2.py'
```

```
In: %run C:₩myPyCode₩modules₩my_module_test2.py
Out: 모듈을 직접 실행
     입력 숫자: 3
     입력 숫자: 4
```

```
In: import my_module_test2
```

모듈 실행 방법

- 모듈을 직접 수행하는 경우와 임포트해서 이용하는 경우를 구분할 수 있는 코드의 구조

```
if __name__ == "__main__":
    <직접 수행할 때만 실행되는 코드>
else:
    <임포트됐을 때만 실행되는 코드>
```

```
In: %%writefile C:₩myPyCode₩modules₩my_module_test3.py
    # File name: my_module_test3.py
    def func(a):
        print("입력 숫자:",a)
    if __name__ == "__main__":
        print("모듈을 직접 실행")
        func(3)
        func(4)
    else:
        print("모듈을 임포트해서 실행")
Out: Writing C:₩myPyCode₩modules₩my_module_test3.py
```

모듈 실행 방법

- 모듈을 직접 수행하는 경우

```
In: %run C:₩myPyCode₩modules₩my_module_test3.py
Out: 모듈을 직접 실행
     입력 숫자: 3
     입력 숫자: 4
```

- 모듈을 임포트하는 경우

```
In: import my_module_test3
Out: 모듈을 임포트해서 실행
```

내장 모듈

- 파이썬에는 개발 환경을 설치할 때 내장 모듈과 다양한 공개 모듈이 함께 설치됨
- 자신이 원하는 코드를 쉽고 간편하게 작성할 수 있음

내장 모듈

- 난수 발생 모듈
  - 난수(random number): 임의의 숫자
  - random 모듈을 이용해 난수를 생성
- random 모듈 사용법

```
import random
random.random모듈함수()
```

```
In: import random
    random.random()
Out: 0.479891168214428
```

내장 모듈

- random 모듈의 함수와 사용 예

함수	설명	사용 예
random()	0.0 <= 실수 < 1.0 범위의 임의의 실수를 반환	random.random()
randint(a,b)	a <= 정수 <= b의 범위의 임의의 정수 반환	random.randint(1,6)
randrange([start,] stop [,step])	range([start,] stop [,step])에서 임의의 정수를 반환	random.randrange(0, 10, 2)
choice(seq)	공백이 아닌 시퀀스(seq)에서 임의의 항목을 반환	random.choice([1,2,3])
sample(population, k)	시퀀스로 이뤄진 모집단(population)에서 중복되지 않는 k개의 인자를 반환	random.sample([1,2,3,4,5], 2)

- random 모듈 함수의 사용 예

```
In: import random

    dice1 = random.randint(1,6) # 임의의 정수가 생성됨
    dice2 = random.randint(1,6) # 임의의 정수가 생성됨
    print('주사위 두 개의 숫자: {0}, {1}'.format(dice1, dice2))
Out: 주사위 두 개의 숫자: 2, 5
```

내장 모듈

- random 모듈 함수의 사용 예

```
In: import random

    random.randrange(0,11,2)
Out: 4
```

```
In: import random

    num1 = random.randrange(1, 10, 2) # 1 ~ 9(10-1) 중 임의의 홀수 선택
    num2 = random.randrange(0,100,10) # 0 ~ 99(100-1) 중 임의의 10의 단위 숫자 선택
    print('num1: {0}, num2: {1}'.format(num1,num2))
Out: num1: 3, num2: 80
```

```
In: import random

    menu = ['비빔밥', '된장찌개', '볶음밥', '불고기', '스파게티', '피자', '탕수육']
    random.choice(menu)
Out: '탕수육'
```

```
In: import random
    random.sample([1, 2, 3, 4, 5], 2) # 모집단에서 두 개의 인자 선택
Out: [5, 2]
```

내장 모듈

- 날짜 및 시간 관련 처리 모듈
  - datetime 모듈
    - date 클래스: 날짜를 표현
    - time 클래스: 시간을 표현
    - datetime 클래스: 날짜와 시간을 표현
  - datetime 모듈의 각 클래스의 객체를 생성해 이용

```
import datatime

date_obj = datetime.date(year, month, day)
time_obj = datetime.time(hour, minute, second)
datetime_obj = datetime.datatime(year, month, day, hour, minute, second)
```

  - 객체를 생성하지 않고 각 클래스의 클래스 메서드를 이용

```
import datatime

date_var = datetime.date.date_classmethod()
time_var = datetime.time.time_classmethod()
datetime_var = datetime.datetime.datetime_classmethod()
```

내장 모듈

– datetime 모듈 예제

```
In: import datetime

    set_day = datetime.date(2019, 3, 1)
    print(set_day)
Out: 2019-03-01
```

```
In: print('{0}/{1}/{2}'.format(set_day.year,set_day.month,set_day.day ))
Out: 2019/3/1
```

```
In: import datetime
    day1 = datetime.date(2019, 4, 1)
    day2 = datetime.date(2019, 7, 10)
---------------------- Page 199-----------------------
    diff_day = day2 - day1
    print(diff_day)
Out: 100 days, 0:00:00
```

```
In: type(day1)
Out: datetime.date
```

```
In: type(diff_day)
Out: datetime.timedelta
```

내장 모듈

– datetime 모듈 예제

```
In: print("** 지정된 두 날짜의 차이는 {}일입니다. **".format(diff_day.days))
Out: ** 지정된 두 날짜의 차이는 100일입니다. **
```

```
In: import datetime

    print(datetime.date.today())
Out: 2018-05-13
```

```
In: import datetime

    today = datetime.date.today()
    special_day = datetime.date(2018, 12, 31)
    print(special_day - today)
 Out: 232 days, 0:00:00
```

```
In: import datetime
    set_time = datetime.time(15, 30, 45)
    print(set_time)
Out: 15:30:45
```

내장 모듈

– datetime 모듈 예제

```
In: print('{0}:{1}:{2}'.format(set_time.hour,set_time.minute,set_time.second ))
Out: 15:30:45
```

```
In: import datetime
    set_dt = datetime.datetime(2018, 10, 9, 10, 20, 0)
    print(set_dt)
Out: 2018-10-09 10:20:00
```

```
In: print('날짜 {0}/{1}/{2}'.format(set_dt.year, set_dt.month, set_dt.day))
    print('시각 {0}:{1}:{2}'.format(set_dt.hour, set_dt.minute, set_dt.second))
Out: 날짜 2018/10/9
     시각 10:20:0
```

```
In: import datetime
    now = datetime.datetime.now()
    print(now)
Out: 2018-05-13 23:30:49.649207
```

```
In: print("Date & Time: {:%Y-%m-%d, %H:%M:%S}".format(now))
Out: Date & Time: 2018-05-13, 23:30:49
```

내장 모듈

– datetime 모듈 예제

```
In: print("Date: {:%Y, %m, %d}".format(now))
    print("Time: {:%H/%M/%S}".format(now))
Out: Date: 2018, 05, 13
     Time: 23/30/49
```

```
In: now = datetime.datetime.now()
    set_dt = datetime.datetime(2017, 12, 1, 12, 30, 45)

    print("현재 날짜 및 시각:", now)
    print("차이:", set_dt - now)
Out: 현재 날짜 및 시각: 2018-05-13 23:43:02.340587
     차이: -164 days, 12:47:42.659413
```

```
In: from datetime import date, time, datetime
    print(date(2019, 7, 1))
Out: 2019-07-01
```

```
In: print(date.today())
Out: 2018-05-13
```

내장 모듈

- datetime 모듈 예제

```
In: print(time(15, 30, 45))
Out: 15:30:45
```

```
In: print(datetime(2020, 2, 14, 18, 10, 50))
Out: 2020-02-14 18:10:50
```

```
In: print(datetime.now())
Out: 2018-05-13 23:44:37.975887
```

내장 모듈

- 달력 생성 및 처리 모듈
  - calendar 모듈

함수	설명	사용 예
calendar(year [,m=3])	지정된 연도(year)의 전체 달력을 문자열로 반환 (기본 형식은 3개의 열)	calendar.calendar(2017)
month(year, month)	지정된 연도(year)와 월(month)의 달력을 문자열로 반환	calendar.month(2019,1)
monthrange(year, month)	지정된 연도(year)와 월(month)의 시작 요일과 일수 반환. 요일의 경우 0(월요일) ~ 6(일요일) 사이의 숫자로 반환	calendar.monthrange(2020,1)
firstweekday()	달력에 표시되는 주의 첫 번째 요일값을 반환. 기본값으로는 월요일(0)로 지정됨	calendar.firstweekday()
setfirstweekday(weekday)	달력에 표시되는 주의 첫 번째 요일을 지정	calendar.setfirstweekday(6)
weekday(year,month, day)	지정된 날짜[연도(year), 월(month), 일(day)]의 요일을 반환	calendar. weekday(year,month,day)
isleap(year)	지정된 연도(year)가 윤년인지를 판단해 윤년이면 True를, 아니면 False를 반환	calendar.isleap(2020)

내장 모듈

- calendar 모듈의 요일 지정 상수

요일	요일 지정 상수	숫자로 표시
월	calendar.MONDAY	0
화	calendar.TUESDAY	1
수	calendar.WEDNESDAY	2
목	calendar.THURSDAY	3
금	calendar.FRIDAY	4
토	calendar.SATURDAY	5
일	calendar.SUNDAY	6

내장 모듈

- calendar 모듈의 예

```
In: import calendar

    print(calendar.calendar(2018))
Out:                                  2018

      January                   February                  March
  Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
   1  2  3  4  5  6  7            1  2  3  4               1  2  3  4
   8  9 10 11 12 13 14       5  6  7  8  9 10 11       5  6  7  8  9 10 11
  15 16 17 18 19 20 21      12 13 14 15 16 17 18      12 13 14 15 16 17 18
  22 23 24 25 26 27 28      19 20 21 22 23 24 25      19 20 21 22 23 24 25
  29 30 31                  26 27 28                  26 27 28 29 30 31

       April                     May                       June
  Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1           1  2  3  4  5  6               1  2  3
   2  3  4  5  6  7  8       7  8  9 10 11 12 13       4  5  6  7  8  9 10
   9 10 11 12 13 14 15      14 15 16 17 18 19 20      11 12 13 14 15 16 17
  16 17 18 19 20 21 22      21 22 23 24 25 26 27      18 19 20 21 22 23 24
  23 24 25 26 27 28 29      28 29 30 31               25 26 27 28 29 30
  30
(이하 생략)
```

내장 모듈

- calendar 모듈의 예

```
In: print(calendar.calendar(2019, m=4))
Out:                                  2019

      January               February              March                 April
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6            1  2  3            1  2  3       1  2  3  4  5  6  7
 7  8  9 10 11 12 13   4  5  6  7  8  9 10   4  5  6  7  8  9 10   8  9 10 11 12 13 14
14 15 16 17 18 19 20  11 12 13 14 15 16 17  11 12 13 14 15 16 17  15 16 17 18 19 20 21
21 22 23 24 25 26 27  18 19 20 21 22 23 24  18 19 20 21 22 23 24  22 23 24 25 26 27 28
28 29 30 31           25 26 27 28           25 26 27 28 29 30 31  29 30

       May                   June                  July                  August
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
       1  2  3  4  5            1  2       1  2  3  4  5  6  7            1  2  3  4
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   8  9 10 11 12 13 14   5  6  7  8  9 10 11
13 14 15 16 17 18 19  10 11 12 13 14 15 16  15 16 17 18 19 20 21  12 13 14 15 16 17 18
20 21 22 23 24 25 26  17 18 19 20 21 22 23  22 23 24 25 26 27 28  19 20 21 22 23 24 25
27 28 29 30 31        24 25 26 27 28 29 30  29 30 31              26 27 28 29 30 31

    September             October              November              December
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
                   1            1  2  3  4  5  6            1  2  3                  1
 2  3  4  5  6  7  8   7  8  9 10 11 12 13   4  5  6  7  8  9 10   2  3  4  5  6  7  8
 9 10 11 12 13 14 15  14 15 16 17 18 19 20  11 12 13 14 15 16 17   9 10 11 12 13 14 15
16 17 18 19 20 21 22  21 22 23 24 25 26 27  18 19 20 21 22 23 24  16 17 18 19 20 21 22
23 24 25 26 27 28 29  28 29 30 31           25 26 27 28 29 30     23 24 25 26 27 28 29
30                                                               30 31
```

내장 모듈

- calendar 모듈의 예

```
In: print(calendar.month(2020,9))
Out:    September 2020
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
```

```
In: calendar.monthrange(2020,2)
Out: (5, 29)
```

```
In: calendar.firstweekday()
Out: 0
```

```
In: calendar.setfirstweekday(calendar.SUNDAY)
    print(calendar.month(2020,9))
Out:    September 2020
Su Mo Tu We Th Fr Sa
      1  2  3  4  5
6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

내장 모듈

- calendar 모듈의 예

```
In: print(calendar.weekday(2018, 10, 14))
Out: 6
```

```
In: print(calendar.isleap(2018))
    print(calendar.isleap(2020))
Out: False
     True
```

패키지

- 패키지(Package)란?
  - 여러 모듈을 체계적으로 모아서 관리하기 위한 꾸러미
  - 모듈을 폴더로 묶어서 계층적으로 관리
- 패키지의 구조
  - 폴더 구조. 각 폴더에는 '__init__.py'라는 특별한 파일이 존재

```
₩---image
   |   __init__.py
   |
   +---effect
   |       rotate.py
   |       zoomInOut.py
   |       __init__.py
   |
   +---filter
   |       blur.py
   |       sharpen.py
   |       __init__.py
   |
   ₩---io_file
           imgread.py
           imgwrite.py
           __init__.py
```

패키지

- 패키지 만들기
  - PYTHONPATH 환경 변수에 패키지가 위치한 폴더를 지정

```
₩---image
    |   __init__.py
    |
    ₩---io_file
            __init__.py
            imgread.py
```

```
In: mkdir C:₩myPyCode₩packages₩image; C:₩myPyCode₩packages₩image₩io_file
```

```
In: %%writefile C:₩myPyCode₩packages₩image₩__init__.py
    # File name: __init__.py
Out: Writing C:₩myPyCode₩packages₩image₩__init__.py
```

```
In: %%writefile C:₩myPyCode₩packages₩image₩io_file₩__init__.py
    # File name: __init__.py
Out: Writing C:₩myPyCode₩packages₩image₩io_file₩__init__.py
```

패키지

- 패키지 만들기

```
In: %%writefile C:₩myPyCode₩packages₩image₩io_file₩imgread.py
    # File name: imgread.py

    def pngread():
        print("pngread in imgread module")

    def jpgread():
        print("jpgread in imgread module")

Out: Writing C:₩myPyCode₩packages₩image₩io_file₩imgread.py
```

```
In: !tree /F  c:₩myPyCode₩packages

Out: System 볼륨에 대한 폴더 경로의 목록입니다.
     볼륨 일련 번호가 00000028 A8C5:B1DE입니다.
     C:₩MYPYCODE₩PACKAGES
      └─image
          |   __init__.py
          |
          └─io_file
                imgread.py
                __init__.py
```

패키지

- 패키지 사용하기
  - 'import 패키지 내 모듈명'으로 선언

```
In: import image.io_file.imgread # image 패키지 io_file 폴더의 imgread 모듈 임포트

    image.io_file.imgread.pngread() # imgread 모듈 내의 pngread() 함수 호출
    image.io_file.imgread.jpgread() # imgread 모듈 내의 jpgread() 함수 호출

Out: pngread in imgread module
     jpgread in imgread module
```

```
In: from image.io_file import imgread

    imgread.pngread()
    imgread.jpgread()

Out: pngread in imgread module
     jpgread in imgread module
```

```
In: from image.io_file.imgread import pngread

    pngread()

Out: pngread in imgread module
```

패키지

- 패키지 사용하기

```
In: from image.io_file.imgread import *

    pngread()
    jpgread()

Out: pngread in imgread module
     jpgread in imgread module
```

```
In: from image.io_file.imgread import pngread, jpgread

    pngread()
    jpgread()

Out: pngread in imgread module
     jpgread in imgread module
```

```
In: from image.io_file import imgread as img
    img.pngread()

    img.jpgread()

Out: pngread in imgread module
     jpgread in imgread module
```

패키지

- 패키지 사용하기

```
In: from image.io_file.imgread import pngread as pread
    from image.io_file.imgread import jpgread as jread

    pread()
    jread()
Out: pngread in imgread module
     jpgread in imgread module
```

11

데이터 분석을 위한 패키지

배열 데이터를 효과적으로 다루는 NumPy

- 파이썬으로 과학 연산을 쉽고 빠르게 할 수 있게 만든 패키지
- NumPy 홈페이지: http://www.numpy.org
- 아나콘다 배포판에는 NumPy가 포함돼 있음

배열 데이터를 효과적으로 다루는 NumPy

- 배열 생성하기
  - 배열(Array)이란 순서가 있는 같은 종류의 데이터가 저장된 집합
  - NumPy 임포트

```
In: import numpy as np
```

  - 시퀀스 데이터로부터 배열 생성

```
arr_obj = np.array(seq_data)
```

  - 리스트로부터 NumPy의 1차원 배열을 생성하는 예

```
In: import numpy as np
    data1 = [0, 1, 2, 3, 4, 5]
    a1 = np.array(data1)
    a1
Out: array([0, 1, 2, 3, 4, 5])
```

```
In: data2 = [0.1, 5, 4, 12, 0.5]
    a2 = np.array(data2)
    a2
Out: array([ 0.1,  5. ,  4. , 12. ,  0.5])
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열 객체의 타입 확인

```
In: a1.dtype
Out: dtype('int32')
```

```
In: a2.dtype
Out: dtype('float64')
```

- 다차원 배열 생성

```
In: np.array([[1,2,3], [4,5,6], [7,8,9]])
Out: array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
```

- 범위를 지정해 배열 생성

```
arr_obj = np.arange([start,] stop[, step])
```

```
In: np.arange(0, 10, 2)
Out: array([0, 2, 4, 6, 8])
```

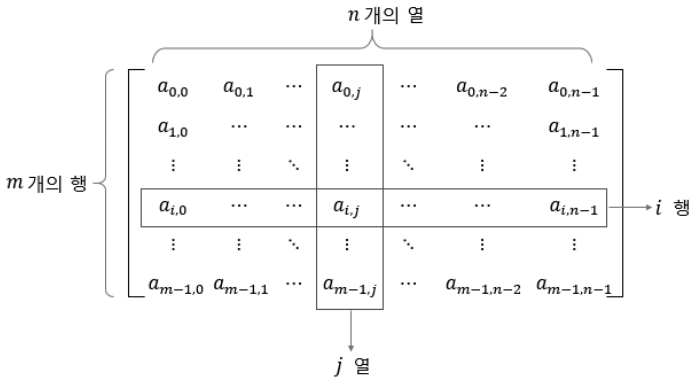
배열 데이터를 효과적으로 다루는 NumPy

- 범위를 지정해 배열 생성

```
In: np.arange(1, 10)
Out: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In: np.arange(5)
Out: array([0, 1, 2, 3, 4])
```

- m x n 행렬: .reshape(m, n) 이용




```
In: np.arange(12).reshape(4,3)
Out: array([[ 0,  1,  2],
            [ 3,  4,  5],
            [ 6,  7,  8],
            [ 9, 10, 11]])
```

배열 데이터를 효과적으로 다루는 NumPy

- m x n 행렬의 형태 확인

```
In: b1 = np.arange(12).reshape(4,3)
    b1.shape
Out: (4, 3)
```

```
In: b2 = np.arange(5)
    b2.shape
Out: (5,)
```

- 범위의 시작과 끝, 데이터의 개수를 지정해 배열 생성

```
arr_obj = np.linspace(start, stop[, num])
```

```
In: np.linspace(1, 10, 10)
Out: array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

```
In: np.linspace(0, np.pi, 20 )
Out: array([0.        , 0.16534698, 0.33069396, 0.49604095, 0.66138793,
                    0.82673491, 0.99208189, 1.15742887, 1.32277585, 1.48812284,
                    1.65346982, 1.8188168 , 1.98416378, 2.14951076, 2.31485774,
                    2.48020473, 2.64555171, 2.81089869, 2.97624567, 3.14159265])
```

배열 데이터를 효과적으로 다루는 NumPy

- 특별한 형태의 배열 생성

```
arr_zero_n = np.zeros(n)
arr_zero_mxn = np.zeros((m,n))
arr_one_n = np.ones(n)
arr_one_mxn = np.ones((m,n))
```

```
In: np.zeros(10)
Out: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
In: np.zeros((3,4))
Out: array([[0., 0., 0., 0.],
            [0., 0., 0., 0.],
            [0., 0., 0., 0.]])
```

```
In: np.ones(5)
Out: array([ 1.,  1.,  1.,  1.,  1.])
```

```
In: np.ones((3,5))
Out: array([[ 1.,  1.,  1.,  1.,  1.],
            [ 1.,  1.,  1.,  1.,  1.],
            [ 1.,  1.,  1.,  1.,  1.]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 단위 행렬 생성

```
arr_I = np.eye(n)
```

```
In: np.eye(3)
Out: array([[1., 0., 0.],
            [0., 1., 0.],
            [0., 0., 1.]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 데이터 타입 변환

```
In: np.array(['1.5', '0.62', '2', '3.14', '3.141592'])
Out: array(['1.5', '0.62', '2', '3.14', '3.141592'], dtype='<U8')
```

  - NumPy 데이터의 형식

기호	설명
'b'	불, bool
'i'	기호가 있는 정수, (signed) integer
'u'	기호가 없는 정수, unsigned integer
'f'	실수, floating-point
'c'	복소수, complex-floating point
'M'	날짜, datetime
'O'	파이썬 객체, (Python) objects
'S' 혹은 'a'	바이트 문자열, (byte) string
'U'	유니코드, Unicode

  - NumPy 배열의 형 변환

```
num_arr = str_arr.astype(dtype)
```

배열 데이터를 효과적으로 다루는 NumPy

- NumPy 배열의 형 변환

```
In: str_a1 = np.array(['1.567', '0.123', '5.123', '9', '8'])
    num_a1 = str_a1.astype(float)
    num_a1
Out: array([1.567, 0.123, 5.123, 9.   , 8.   ])
```

```
In: str_a1.dtype
Out: dtype('<U5')
```

```
In: num_a1.dtype
Out: dtype('float64')
```

배열 데이터를 효과적으로 다루는 NumPy

- 난수 배열의 생성

```
rand_num = np.random.rand([d0, d1, ..., dn])
rand_num = np.random.randint([low,] high [,size])
```

```
In: np.random.rand(2,3)
Out: array([[0.65311939, 0.89752463, 0.63411962],
            [0.1345534 , 0.27230463, 0.02711115]])
```

```
In: np.random.rand()
Out: 0.8324172369983784
```

```
In: np.random.rand(2,3,4)
Out: array([[[ 0.06256587,  0.48831201,  0.57252114,  0.78417988],
             [ 0.62835321,  0.13173961,  0.46895454,  0.00443031],
             [ 0.76377121,  0.71765738,  0.0828908 ,  0.57340376]],
            [[ 0.97789304,  0.94486134,  0.86353152,  0.2843577 ],
             [ 0.1634681 ,  0.39515681,  0.21691386,  0.19066458],
             [ 0.38078663,  0.35489043,  0.60452622,  0.91283752]]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 난수 배열의 생성

```
In: np.random.randint(10, size=(3, 4))
Out: array([[4, 1, 8, 7],
            [2, 9, 3, 2],
            [2, 9, 3, 8]])
```

```
In: np.random.randint(1, 30)
Out: 12
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 연산
  - 기본 연산

```
In: arr1 = np.array([10, 20, 30, 40])
    arr2 = np.array([1, 2, 3, 4])
```

```
In: arr1 + arr2
Out: array([11, 22, 33, 44])
```

```
In: arr1 - arr2
Out: array([ 9, 18, 27, 36])
```

```
In: arr2 * 2
Out: array([2, 4, 6, 8])
```

```
In: arr2 ** 2
Out: array([ 1,  4,  9, 16], dtype=int32)
```

```
In: arr1 * arr2
Out: array([ 10,  40,  90, 160])
```

```
In: arr1 / arr2
Out: array([ 10.,  10.,  10.,  10.])
```

배열 데이터를 효과적으로 다루는 NumPy

- 기본 연산

```
In: arr1 / (arr2 ** 2)
Out: array([10.        ,  5.        ,  3.33333333,  2.5       ])
```

```
In: arr1 > 20
Out: array([False, False,  True,  True])
```

- 통계를 위한 연산
  - sum(), mean(), std(), var(), min(), max(), cumsum(), cumprod() 등

```
In: arr3 = np.arange(5)
    arr3
Out: array([0, 1, 2, 3, 4])
```

```
In: [arr3.sum(), arr3.mean()]
Out: [10, 2.0]
```

```
In: [arr3.std(), arr3.var()]
Out: [1.4142135623730951, 2.0]
```

```
In: [arr3.min(), arr3.max()]
Out: [0, 4]
```

배열 데이터를 효과적으로 다루는 NumPy

– 통계를 위한 연산

```
In: arr4 = np.arange(1,5)
    arr4
Out: array([1, 2, 3, 4])
```

```
In: arr4.cumsum()
Out: array([ 1,  3,  6, 10], dtype=int32)
```

```
In: arr4.cumprod()
Out: array([ 1,  2,  6, 24], dtype=int32)
```

– 행렬 연산

행렬 연산	사용 예
행렬곱(matrix product)	A.dot(B), 혹은 np.dot(A,B)
전치행렬(transpose matrix)	A.transpose(), 혹은 np.transpose(A)
역행렬(inverse matrix)	np.linalg.inv(A)
행렬식(determinant)	np.linalg.det(A)

배열 데이터를 효과적으로 다루는 NumPy

– 행렬 연산

```
In: A = np.array([0, 1, 2, 3]).reshape(2,2)
    A
Out: array([[0, 1],
            [2, 3]])
```

```
In: B = np.array([3, 2, 0, 1]).reshape(2,2)
    B
Out: array([[3, 2],
            [0, 1]])
```

```
In: A.dot(B)
Out: array([[0, 1],
            [6, 7]])
```

```
In: np.dot(A,B)
Out: array([[0, 1],
            [6, 7]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 행렬 연산

```
In: np.transpose(A)
Out: array([[0, 2],
            [1, 3]])
```

```
In: A.transpose()
Out: array([[0, 2],
            [1, 3]])
```

```
In: np.linalg.inv(A)
Out: array([[-1.5,  0.5],
            [ 1. ,  0. ]])
```

```
In: np.linalg.det(A)
Out: -2.0
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 인덱싱과 슬라이싱
  - 배열에서 선택된 원소는 값을 가져오거나 변경할 수 있음
  - 인덱싱(Indexing): 배열의 위치나 조건을 지정해 배열의 원소를 선택
  - 슬라이싱(Slicing):범위를 지정해 배열의 원소를 선택
  - 배열의 인덱싱

```
배열명[위치]
```

```
In: a1 = np.array([0, 10, 20, 30, 40, 50])
    a1
Out: array([ 0, 10, 20, 30, 40, 50])
```

```
In: a1[0]
Out: 0
```

```
In: a1[4]
Out: 40
```

```
In: a1[5] = 70
    a1
Out: array([ 0, 10, 20, 30, 40, 70])
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열에서 여러 개의 원소를 선택

```
배열명[[위치1, 위치2, …, 위치n]]
```

```
In: a1[[1,3,4]]
Out: array([10, 30, 40])
```

- 2차원 배열에서 특정 위치의 원소를 선택

```
배열명[행_위치, 열_위치]
```

```
In: a2 = np.arange(10, 100, 10).reshape(3,3)
    a2
Out: array([[10, 20, 30],
            [40, 50, 60],
            [70, 80, 90]])
In: a2[0, 2]
Out: 30
```

배열 데이터를 효과적으로 다루는 NumPy

– 2차원 배열에서 특정 위치의 원소를 선택

```
In: a2[2, 2] = 95
    a2
Out: array([[10, 20, 30],
            [40, 50, 60],
            [70, 80, 95]])
```

```
In: a2[1]
Out: array([40, 50, 60])
```

```
In: a2[1] = np.array([45, 55, 65])
    a2
Out: array([[10, 20, 30],
            [45, 55, 65],
            [70, 80, 95]])
```

```
In: a2[1] = [47, 57, 67]
    a2
Out: array([[10, 20, 30],
            [47, 57, 67],
            [70, 80, 95]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 2차원 배열에서 여러 원소를 선택

```
배열명[[행_위치1, 행_위치2, …, 행_위치n], [열_위치1, 열_위치2, …, 열_위치n]]
```

```
In: a2[[0, 2], [0, 1]]
Out: array([10, 80])
```

- 배열에 조건을 지정해 조건에 맞는 배열을 선택

```
배열명[조건]
```

```
In: a = np.array([1, 2, 3, 4, 5, 6])
    a[a > 3]
Out: array([4, 5, 6])
```

```
In: a[(a % 2) == 0]
Out: array([2, 4, 6])
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 슬라이싱

```
배열[시작_위치:끝_위치]
```

```
In: b1 = np.array([0, 10, 20, 30, 40, 50])
    b1[1:4]
Out: array([10, 20, 30])
```

```
In: b1[:3]
Out: array([ 0, 10, 20])
```

```
In: b1[2:]
Out: array([20, 30, 40, 50])
```

```
In: b1[2:5] = np.array([25, 35, 45])
    b1
Out: array([ 0, 10, 25, 35, 45, 50])
```

```
In: b1[3:6] = 60
    b1
Out: array([ 0, 10, 25, 60, 60, 60])
```

배열 데이터를 효과적으로 다루는 NumPy

- 2차원 배열의 슬라이싱

```
배열[행_시작_위치:행_끝_위치, 열_시작_위치:열_끝_위치]
```

- 특정 행을 선택한 후 열을 슬라이싱

```
배열[행_위치][열_시작_위치:열_끝_위치]
```

- 슬라이싱 예

```
In: b2 = np.arange(10, 100, 10).reshape(3,3)
    b2
Out: array([[10, 20, 30],
            [40, 50, 60],
            [70, 80, 90]])
```

```
In: b2[1:3, 1:3]
Out: array([[50, 60],
            [80, 90]])
```

```
In: b2[:3, 1:]
Out: array([[20, 30],
            [50, 60],
            [80, 90]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 슬라이싱 예

```
In: b2[1][0:2]
Out: array([40, 50])
```

```
In: b2[0:2, 1:3] = np.array([[25, 35], [55, 65]])
    b2
Out: array([[10, 25, 35],
            [40, 55, 65],
            [70, 80, 90]])
```

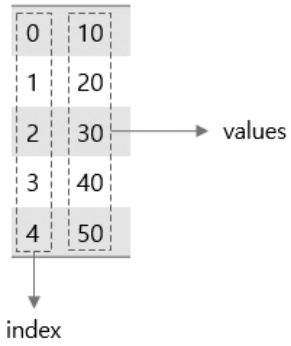
구조적 데이터 표시와 처리에 강한 pandas

- 데이터 분석과 처리를 쉽게 할 수 있게 도와주는 라이브러리
- NumPy를 기반으로 하지만 좀 더 복잡한 데이터 분석에 특화
- 아나콘다 배포판에 포함돼 있음
- pandas 홈페이지: http://pandas.pydata.org

구조적 데이터 표시와 처리에 강한 pandas

- 구조적 데이터 생성하기
  - Series를 활용한 데이터 생성

```
In: import pandas as pd
s = pd.Series(seq_data)
```




```
In: s1 = pd.Series([10, 20, 30, 40, 50])
    s1
Out: 0    10
     1    20
     2    30
     3    40
     4    50
     dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- Series를 활용한 데이터 생성

```
In: s1.index
    print(s1.index)
Out: RangeIndex(start=0, stop=5, step=1)
```

```
In: s1.values
Out: array([10, 20, 30, 40, 50], dtype=int64)
```

```
In: s2 = pd.Series(['a', 'b', 'c', 1, 2, 3])
    s2
Out: 0    a
    1    b
    2    c
    3    1
    4    2
    5    3
    dtype: object
```

구조적 데이터 표시와 처리에 강한 pandas

- Series를 활용한 데이터 생성

```
In: import numpy as np
    s3 = pd.Series([np.nan,10,30])
    s3
Out: 0     NaN
     1    10.0
     2    30.0
     dtype: float64
```

```
s = pd.Series(seq_data, index = index_seq)
```

```
In: index_date = ['2018-10-07','2018-10-08','2018-10-09','2018-10-10']
    s4 = pd.Series([200, 195, np.nan, 205], index = index_date)
    s4
Out: 2018-10-07    200.0
     2018-10-08    195.0
     2018-10-09      NaN
     2018-10-10    205.0
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

– Series를 활용한 데이터 생성

```
s = pd.Series(dict_data)
```

```
In: s5 = pd.Series({'국어': 100, '영어': 95, '수학': 90})
    s5
Out: 국어   100
     수학    90
     영어    95
     dtype: int64
```

– 날짜 자동 생성: date_range

```
pd.date_range(start=None, end=None, periods=None, freq='D')
```

```
In: import pandas as pd
    pd.date_range(start='2019-01-01',end='2019-01-07')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range

```
In: pd.date_range(start='2019/01/01',end='2019.01.07')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

```
In: pd.date_range(start='01-01-2019',end='01/07/2019')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

```
In: pd.date_range(start='2019-01-01',end='01.07.2019')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

```
In: pd.date_range(start='2019-01-01', periods = 7)
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range
  - date_rage() 함수의 freq 옵션

약어	설명	부가 설명 및 사용 예
D	달력 날짜 기준 하루 주기	하루 주기: freq = 'D', 이틀 주기: freq = '2D'
B	업무 날짜 기준 하루 주기	업무일(월요일 ~ 금요일) 기준으로 생성. freq = 'B', freq = '3B'
W	일요일 시작 기준 일주일 주기	월요일: W-MON, 화요일: W-TUE. freq = 'W', freq = 'W-MON'
M	월말 날짜 기준 주기	한 달 주기: freq = 'M', 네 달 주기: freq = '4M'
BM	업무 월말 날짜 기준 주기	freq = 'BM', freq = '2BM'
MS	월초 날짜 기준 주기	freq = 'MS', freq = '3MS'
BMS	업무 월초 날짜 기준 주기	freq = 'BMS', freq = '3BMS'
Q	분기 끝 날짜 기준 주기	freq = 'Q', freq = '2Q'
BQ	업무 분기 끝 날짜 기준 주기	freq = 'BQ', freq = '2BQ'
QS	분기 시작 날짜 기준 주기	freq = 'QS', freq = '2QS'
BQS	업무 분기 시작 날짜 기준 주기	freq = 'BQS', freq = '2BQS'
A	일년 끝 날짜 기준 주기	freq = 'A', freq = '5A'
BA	업무 일년 끝 날짜 기준 주기	freq = 'BA', freq = '3BA'
AS	일년 시작 날짜 기준 주기	freq = 'AS', freq = '2AS'
BAS	업무 일년 시작 날짜 기준 주기	freq = 'BAS', freq = '2BAS'
H	시간 기준 주기	1시간 주기: freq = 'H', 2시간 주기: freq = '2H'
BH	업무 시간 기준 주기	업무 시간 (09:00 ~ 17:00) 기준으로 생성
T, min	분 주기	10분 주기: freq = '10T', 30분 주기: freq = '30min'
S	초 주기	1초 주기: freq = 'S', 10초 주기: freq = '10S'

구조적 데이터 표시와 처리에 강한 pandas

– 날짜 자동 생성: date_range

```
In: pd.date_range(start='2019-01-01', periods = 4, freq = '2D')
Out: DatetimeIndex(['2019-01-01', '2019-01-03', '2019-01-05', '2019-01-07'],
      dtype='datetime64[ns]', freq='2D')
```

```
In: pd.date_range(start='2019-01-01', periods = 4, freq = 'W')
Out: DatetimeIndex(['2019-01-06', '2019-01-13', '2019-01-20', '2019-01-27'],
      dtype='datetime64[ns]', freq='W-SUN')
```

```
In: pd.date_range(start='2019-01-01', periods = 12, freq = '2BM')
Out: DatetimeIndex(['2019-01-31', '2019-03-29', '2019-05-31', '2019-07-31',
                    '2019-09-30', '2019-11-29', '2020-01-31', '2020-03-31',
                    '2020-05-29', '2020-07-31', '2020-09-30', '2020-11-30'],
                    dtype='datetime64[ns]', freq='2BM')
```

```
In: pd.date_range(start='2019-01-01', periods = 4, freq = 'QS')
Out: DatetimeIndex(['2019-01-01', '2019-04-01', '2019-07-01', '2019-10-01'],
      dtype='datetime64[ns]', freq='QS-JAN')
```

```
In: pd.date_range(start='2019-01-01', periods = 3, freq = 'AS')
Out: DatetimeIndex(['2019-01-01', '2020-01-01', '2021-01-01'], dtype='datetime64[ns]',
      freq='AS-JAN')
```

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range

```
In: pd.date_range(start = '2019-01-01 08:00', periods = 10, freq='H')
Out: DatetimeIndex(['2019-01-01 08:00:00', '2019-01-01 09:00:00',
                    '2019-01-01 10:00:00', '2019-01-01 11:00:00',
                    '2019-01-01 12:00:00', '2019-01-01 13:00:00',
                    '2019-01-01 14:00:00', '2019-01-01 15:00:00',
                    '2019-01-01 16:00:00', '2019-01-01 17:00:00'],
                    dtype='datetime64[ns]', freq='H')
```

```
In: pd.date_range(start = '2019-01-01 08:00', periods = 10, freq='BH')
Out: DatetimeIndex(['2019-01-01 09:00:00', '2019-01-01 10:00:00',
                    '2019-01-01 11:00:00', '2019-01-01 12:00:00',
                    '2019-01-01 13:00:00', '2019-01-01 14:00:00',
                    '2019-01-01 15:00:00', '2019-01-01 16:00:00',
                    '2019-01-02 09:00:00', '2019-01-02 10:00:00'],
                    dtype='datetime64[ns]', freq='BH')
```

```
In: pd.date_range(start = '2019-01-01 10:00', periods = 4, freq='30min')
Out: DatetimeIndex(['2019-01-01 10:00:00', '2019-01-01 10:30:00',
                    '2019-01-01 11:00:00', '2019-01-01 11:30:00'],
                    dtype='datetime64[ns]', freq='30T')
```

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range

```
In: pd.date_range(start = '2019-01-01 10:00', periods = 4, freq='30T')
Out: DatetimeIndex(['2019-01-01 10:00:00', '2019-01-01 10:30:00',
                    '2019-01-01 11:00:00', '2019-01-01 11:30:00'],
                    dtype='datetime64[ns]', freq='30T')
```

```
In: pd.date_range(start = '2019-01-01 10:00:00', periods = 4, freq='10S')
Out: DatetimeIndex(['2019-01-01 10:00:00', '2019-01-01 10:00:10',
                    '2019-01-01 10:00:20', '2019-01-01 10:00:30'],
                    dtype='datetime64[ns]', freq='10S')
```

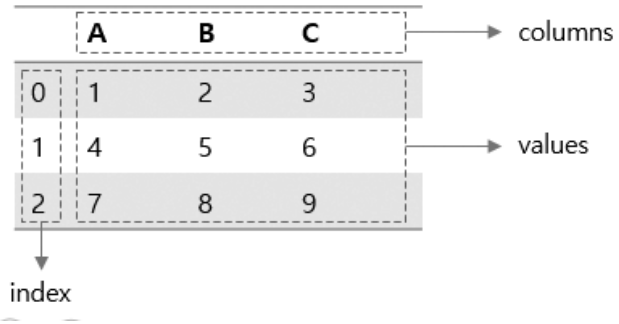
```
In: index_date = pd.date_range(start = '2019-03-01', periods = 5, freq='D')
    pd.Series([51, 62, 55, 49, 58], index = index_date )
Out: 2019-03-01    51
     2019-03-02    62
     2019-03-03    55
     2019-03-04    49
     2019-03-05    58
     Freq: D, dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame을 활용한 데이터 생성
  - DataFrame: 표(Table)와 같은 2차원 데이터 처리를 위한 형식

```
df = pd.DataFrame(data [, index = index_data, columns = columns_data])
```

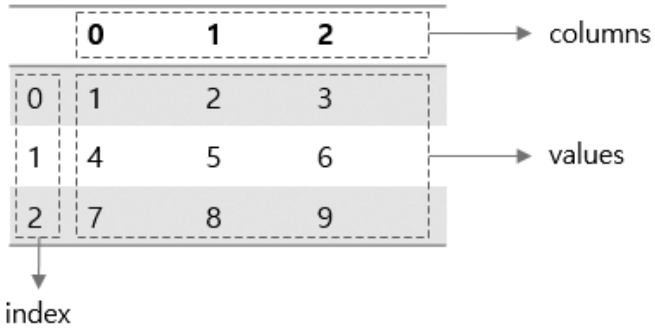
  - DataFrame의 구조




```
In: import pandas as pd
    pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
Out:
       0  1  2
    0  1  2  3
    1  4  5  6
    2  7  8  9
```

구조적 데이터 표시와 처리에 강한 pandas

– 자동으로 생성된 index와 columns를 갖는 DataFrame 데이터




```
In: import numpy as np
    import pandas as pd
    data_list = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
    pd.DataFrame(data_list)
Out:
        0   1   2
    0  10  20  30
    1  40  50  60
    2  70  80  90
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 데이터 예제

```
In: import numpy as np
    import pandas as pd

    data = np.array([[1, 2, 3], [4, 5, 6], [7, 8 ,9], [10, 11, 12]])
    index_date = pd.date_range('2019-09-01', periods=4)
    columns_list = ['A', 'B', 'C']
    pd.DataFrame(data, index=index_date, columns=columns_list)
Out:
                    A     B     C
        2019-09-01      1     2     3
        2019-09-02      4     5     6
        2019-09-03      7     8     9
        2019-09-04      10    11    12
```

구조적 데이터 표시와 처리에 강한 pandas

– DataFrame 데이터 예제

```
In: table_data = {'연도': [2015, 2016, 2016, 2017, 2017],
                  '지사': ['한국', '한국', '미국', '한국','미국'],
                  '고객 수': [200, 250, 450, 300, 500]}
    table_data
Out: {'고객 수': [200, 250, 450, 300, 500],
      '연도': [2015, 2016, 2016, 2017, 2017],
      '지사': ['한국', '한국', '미국', '한국', '미국']}
```

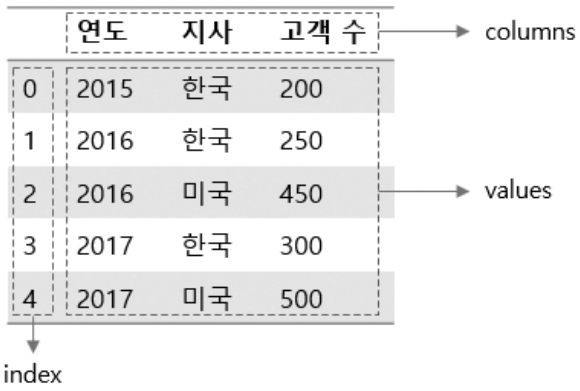
```
In: pd.DataFrame(table_data)
Out:
```

	고객 수	연도	지사
0	200	2015	한국
1	250	2016	한국
2	450	2016	미국
3	300	2017	한국
4	500	2017	미국

구조적 데이터 표시와 처리에 강한 pandas

– DataFrame 데이터 예제

```
In: df = pd.DataFrame(table_data, columns=['연도', '지사', '고객 수'])
    df
Out:
        연도    지사    고객 수
    0   2015    한국    200
    1   2016    한국    250
    2   2016    미국    450
    3   2017    한국    300
    4   2017    미국    500
```



구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 데이터 예제

```
In: df.index
Out: RangeIndex(start=0, stop=5, step=1)
```

```
In: df.columns
Out: Index(['연도', '지사', '고객 수'], dtype='object')
```

```
In: df.values
Out: array([[2015, '한국', 200],
            [2016, '한국', 250],
            [2016, '미국', 450],
            [2017, '한국', 300],
            [2017, '미국', 500]], dtype=object)
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 연산

```
In: s1 = pd.Series([1, 2, 3, 4, 5])
    s2 = pd.Series([10, 20, 30, 40, 50])
    s1 + s2
Out: 0    11
    1    22
    2    33
    3    44
    4    55
    dtype: int64
```

```
In: s2 - s1
Out: 0     9
     1    18
     2    27
     3    36
     4    45
     dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 연산

```
In: s1 * s2
Out: 0     10
     1     40
     2     90
     3    160
     4    250
     dtype: int64
```

```
In: s2 / s1
Out: 0    10.0
     1    10.0
     2    10.0
     3    10.0
     4    10.0
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 연산
  - 데이터 크기가 다른 경우

```
In: s3 = pd.Series([1, 2, 3, 4])
    s4 = pd.Series([10, 20, 30, 40, 50])
    s3 + s4
Out: 0    11.0
     1    22.0
     2    33.0
     3    44.0
     4     NaN
     dtype: float64
```

```
In: s4 - s3
Out: 0     9.0
     1    18.0
     2    27.0
     3    36.0
     4     NaN
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 크기가 다른 경우

```
In: s3 * s4
Out: 0    10.0
     1    40.0
     2    90.0
     3   160.0
     4     NaN
     dtype: float64
```

```
In: s4/s3
Out: 0    10.0
     1    10.0
     2    10.0
     3    10.0
     4     NaN
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 간의 사칙 연산

```
In: table_data1 = {'A': [1, 2, 3, 4, 5],
                   'B': [10, 20, 30, 40, 50],
                   'C': [100, 200, 300, 400, 500]}
    df1 = pd.DataFrame(table_data1)
    df1
Out:
        A   B    C
    0   1   10   100
    1   2   20   200
    2   3   30   300
    3   4   40   400
    4   5   50   500
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 간의 사칙 연산

```
In: table_data2 = {'A': [6, 7, 8],
                   'B': [60, 70, 80],
                   'C': [600, 700, 800]}
    df2 = pd.DataFrame(table_data2)
    df2
Out:
        A    B     C
    0   6    60    600
    1   7    70    700
    2   8    80    800
```

```
In: df1 + df2
Out:
        A       B        C
    0   7.0     70.0     700.0
    1   9.0     90.0     900.0
    2   11.0    110.0    1100.0
    3   NaN     NaN      NaN
    4   NaN     NaN      NaN
```

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

연도	봄	여름	가을	겨울
2012	256.5	770.6	363.5	139.3
2013	264.3	567.5	231.2	59.9
2014	215.9	599.8	293.1	76.9
2015	223.2	387.1	247.7	109.1
2016	312.8	446.2	381.6	108.1

2012년부터 2016년까지 우리나라의 계절별 강수량

```
In: table_data3 = {'봄':  [256.5, 264.3, 215.9, 223.2, 312.8],
                   '여름': [770.6, 567.5, 599.8, 387.1, 446.2],
                   '가을': [363.5, 231.2, 293.1, 247.7, 381.6],
                   '겨울': [139.3, 59.9, 76.9, 109.1, 108.1]}
    columns_list = ['봄', '여름', '가을', '겨울']
    index_list = ['2012', '2013', '2014', '2015', '2016']
    df3 = pd.DataFrame(table_data3, columns = columns_list, index = index_list)
    df3
Out:
```

	봄	여름	가을	겨울
2012	256.5	770.6	363.5	139.3
2013	264.3	567.5	231.2	59.9
2014	215.9	599.8	293.1	76.9
2015	223.2	387.1	247.7	109.1
2016	312.8	446.2	381.6	108.1

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

```
In: df3.mean()
Out: 봄      254.54
     여름    554.24
     가을    303.42
     겨울     98.66
     dtype: float64
```

```
In: df3.std()
Out: 봄      38.628267
     여름   148.888895
     가을    67.358496
     겨울    30.925523
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

```
In: df3.mean(axis=1)
Out: 2012    382.475
     2013    280.725
     2014    296.425
     2015    241.775
     2016    312.175
     dtype: float64
```

```
In: df3.std(axis=1)
Out: 2012    274.472128
     2013    211.128782
     2014    221.150739
     2015    114.166760
     2016    146.548658
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

```
In: df3.describe()
Out:
              봄          여름          가을          겨울
count    5.000000     5.000000     5.000000     5.000000
mean   254.540000   554.240000   303.420000    98.660000
std     38.628267   148.888895    67.358496    30.925523
min    215.900000   387.100000   231.200000    59.900000
25%    223.200000   446.200000   247.700000    76.900000
50%    256.500000   567.500000   293.100000   108.100000
75%    264.300000   599.800000   363.500000   109.100000
max    312.800000   770.600000   381.600000   139.300000
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기
  - 2010년부터 2017년까지 노선별 KTX 이용자 수

연도	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	-
2012	39896	6967	4168	1771	-
2013	42005	6873	4088	1954	-
2014	43621	6626	4424	2244	-
2015	41702	8675	4606	3146	2395
2016	41266	10622	4984	3945	3786
2017	32427	9228	5570	5766	6667

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기

```
In: import pandas as pd
    import numpy as np

    KTX_data = {'경부선 KTX': [39060, 39896, 42005, 43621, 41702, 41266, 32427],
                '호남선 KTX': [7313, 6967, 6873, 6626, 8675, 10622, 9228],
                '경전선 KTX': [3627, 4168, 4088, 4424, 4606, 4984, 5570],
                '전라선 KTX': [309, 1771, 1954, 2244, 3146, 3945, 5766],
                '동해선 KTX': [np.nan,np.nan, np.nan, np.nan, 2395, 3786, 6667]}
    col_list = ['경부선 KTX','호남선 KTX','경전선 KTX','전라선 KTX','동해선 KTX']
    index_list = ['2011', '2012', '2013', '2014', '2015', '2016', '2017']
    df_KTX = pd.DataFrame(KTX_data, columns = col_list, index = index_list)
    df_KTX
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	NaN
2012	39896	6967	4168	1771	NaN
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0
2016	41266	10622	4984	3945	3786.0
2017	32427	9228	5570	5766	6667.0

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기

```
In: df_KTX.index
Out: Index(['2011', '2012', '2013', '2014', '2015', '2016', '2017'], dtype='object')
```

```
In: df_KTX.columns
Out: Index(['경부선 KTX', '호남선 KTX', '경전선 KTX', '전라선 KTX', '동해선 KTX'], dtype='object')
```

```
In: df_KTX.values
Out: array([[39060.,  7313.,  3627.,   309.,    nan],
            [39896.,  6967.,  4168.,  1771.,    nan],
            [42005.,  6873.,  4088.,  1954.,    nan],
            [43621.,  6626.,  4424.,  2244.,    nan],
            [41702.,  8675.,  4606.,  3146.,  2395.],
            [41266., 10622.,  4984.,  3945.,  3786.],
            [32427.,  9228.,  5570.,  5766.,  6667.]])
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기
  - 처음 일부분과 끝 일부분만 선택

```
DataFrame_data.head([n])
DataFrame_data.tail([n])
```

```
In: df_KTX.head()
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	NaN
2012	39896	6967	4168	1771	NaN
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0

```
In: df_KTX.tail()
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0
2016	41266	10622	4984	3945	3786.0
2017	32427	9228	5570	5766	6667.0

구조적 데이터 표시와 처리에 강한 pandas

- 처음 일부분과 끝 일부분만 선택

```
In: df_KTX.head(3)
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	NaN
2012	39896	6967	4168	1771	NaN
2013	42005	6873	4088	1954	NaN

```
In: df_KTX.tail(2)
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2016	41266	10622	4984	3945	3786.0
2017	32427	9228	5570	5766	6667.0

구조적 데이터 표시와 처리에 강한 pandas

- 연속된 구간의 행 데이터 선택

```
DataFrame_data[행_시작_위치:행_끝_위치]
```

```
In: df_KTX[1:2]
Out:
      경부선 KTX  호남선 KTX  경전선 KTX  전라선 KTX  동해선 KTX
2012     39896       6967       4168       1771        NaN
```

```
In: df_KTX[2:5]
Out:
      경부선 KTX  호남선 KTX  경전선 KTX  전라선 KTX  동해선 KTX
2013     42005       6873       4088       1954        NaN
2014     43621       6626       4424       2244        NaN
2015     41702       8675       4606       3146       2395.0
```

구조적 데이터 표시와 처리에 강한 pandas

- index 항목 이름을 지정해 행을 선택

```
DataFrame_data.loc[index_name]
```

```
In: df_KTX.loc['2011']
Out: 경부선 KTX    39060.0
     호남선 KTX     7313.0
     경전선 KTX     3627.0
     전라선 KTX      309.0
     동해선 KTX        NaN
     Name: 2011, dtype: float64
```

- index 항목 이름으로 구간을 지정해 연속된 구간의 행을 선택

```
DataFrame_data.loc[start_index_name:end_index_name]
```

```
In: df_KTX.loc['2013':'2016']
Out:
          경부선 KTX    호남선 KTX    경전선 KTX    전라선 KTX    동해선 KTX
2013        42005         6873         4088         1954          NaN
2014        43621         6626         4424         2244          NaN
2015        41702         8675         4606         3146         2395.0
2016        41266        10622         4984         3945         3786.0
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터에서 하나의 열만 선택

```
DataFrame_data[column_name]
```

```
In: df_KTX['경부선 KTX']
Out: 2011    39060
     2012    39896
     2013    42005
     2014    43621
     2015    41702
     2016    41266
     2017    32427
     Name: 경부선 KTX, dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- 하나의 열을 선택한 후 index의 범위를 지정해 선택

```
DataFrame_data[column_name][start_index_name:end_index_name]
DataFrame_data[column_name][start_index_pos:end_index_pos]
```

```
In: df_KTX['경부선 KTX']['2012':'2014']
Out: 2012    39896
     2013    42005
     2014    43621
     Name: 경부선 KTX, dtype: int64
```

```
In: df_KTX['경부선 KTX'][2:5]
Out: 2013    42005
     2014    43621
     2015    41702
     Name: 경부선 KTX, dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- 하나의 원소만 선택

```
DataFrame_data.loc[index_name][column_name]
DataFrame_data.loc[index_name, column_name]
DataFrame_data[column_name][index_name]
DataFrame_data[column_name][index_pos]
DataFrame_data[column_name].loc[index_name]
```

```
In: df_KTX.loc['2016']['호남선 KTX']
Out: 10622.0
```

```
In: df_KTX.loc['2016','호남선 KTX']
Out: 10622
```

```
In: df_KTX['호남선 KTX']['2016']
Out: 10622
```

```
In: df_KTX['호남선 KTX'][5]
Out: 10622
```

```
In: df_KTX['호남선 KTX'].loc['2016']
Out: 10622
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame의 행과 열을 바꾸는 방법(전치)

```
DataFrame_data.T
```

```
In: df_KTX.T
Out:
```

	2011	2012	2013	2014	2015	2016	2017
경부선 KTX	39060.0	39896.0	42005.0	43621.0	41702.0	41266.0	32427.0
호남선 KTX	7313.0	6967.0	6873.0	6626.0	8675.0	10622.0	9228.0
경전선 KTX	3627.0	4168.0	4088.0	4424.0	4606.0	4984.0	5570.0
전라선 KTX	309.0	1771.0	1954.0	2244.0	3146.0	3945.0	5766.0
동해선 KTX	NaN	NaN	NaN	NaN	2395.0	3786.0	6667.0

```
In: df_KTX
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	NaN
2012	39896	6967	4168	1771	NaN
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0
2016	41266	10622	4984	3945	3786.0
2017	32427	9228	5570	5766	6667.0

구조적 데이터 표시와 처리에 강한 pandas

- 열의 항목을 지정해 열의 순서를 지정

```
In: df_KTX[['동해선 KTX', '전라선 KTX', '경전선 KTX', '호남선 KTX', '경부선 KTX']]
Out:
```

	동해선 KTX	전라선 KTX	경전선 KTX	호남선 KTX	경부선 KTX
2011	NaN	309	3627	7313	39060
2012	NaN	1771	4168	6967	39896
2013	NaN	1954	4088	6873	42005
2014	NaN	2244	4424	6626	43621
2015	2395.0	3146	4606	8675	41702
2016	3786.0	3945	4984	10622	41266
2017	6667.0	5766	5570	9228	32427

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 통합하기
  - 세로로 증가하는 방향으로 통합하기
  - 가로로 증가하는 방향으로 통합하기
  - 특정 열을 기준으로 통합하기
- 세로 방향으로 통합하기

```
DataFrame_data1.append(DataFrame_data2 [,ignore_index=True])
```

```
In: import pandas as pd
    import numpy as np
    df1 = pd.DataFrame({'Class1': [95, 92, 98, 100],
                        'Class2': [91, 93, 97, 99]})
    df1
Out:
        Class1    Class2
    0   95        91
    1   92        93
    2   98        97
    3   100       99
```

구조적 데이터 표시와 처리에 강한 pandas

- 세로 방향으로 통합하기

```
In: df2 = pd.DataFrame({'Class1': [87, 89],
                        'Class2': [85, 90]})
    df2
Out:
          Class1    Class2
       0  87        85
       1  89        90
```

  - append()로 데이터 추가

```
In: df1.append(df2)
Out:
          Class1    Class2
       0  95        91
       1  92        93
       2  98        97
       3  100       99
       0  87        85
       1  89        90
```

구조적 데이터 표시와 처리에 강한 pandas

- 세로 방향으로 통합하기

```
In: df1.append(df2, ignore_index=True)
Out:
      Class1   Class2
   0   95       91
   1   92       93
   2   98       97
   3   100      99
   4   87       85
   5   89       90
```

  - 열이 하나만 있는 DataFrame 생성

```
In: df3 = pd.DataFrame({'Class1': [96, 83]})
    df3
Out:
      Class1
   0   96
   1   83
```

구조적 데이터 표시와 처리에 강한 pandas

– 열이 두 개인 데이터(df2)에 열이 하나인 DataFrame 데이터(df3)를 추가

```
In: df2.append(df3, ignore_index=True)
Out:
      Class1   Class2
   0  87       85.0
   1  89       90.0
   2  96       NaN
   3  83       NaN
```

- 가로 방향으로 통합하기

```
DataFrame_data1.join(DataFrame_data2)
```

```
In: df4 = pd.DataFrame({'Class3': [93, 91, 95, 98]})
    df4
Out:
      Class3
   0  93
   1  91
   2  95
   3  98
```

구조적 데이터 표시와 처리에 강한 pandas

- 가로 방향으로 통합하기

```
In: df1.join(df4)
Out:
        Class1    Class2    Class3
    0    95        91        93
    1    92        93        91
    2    98        97        95
    3    100       99        98
```

  - index 라벨을 지정한 경우

```
In: index_label = ['a','b','c','d']
    df1a = pd.DataFrame({'Class1': [95, 92, 98, 100],
                         'Class2': [91, 93, 97, 99]}, index= index_label)
    df4a = pd.DataFrame({'Class3': [93, 91, 95, 98]}, index=index_label)
    df1a.join(df4a)
Out:
        Class1    Class2    Class3
    a    95        91        93
    b    92        93        91
    c    98        97        95
    d    100       99        98
```

구조적 데이터 표시와 처리에 강한 pandas

- index의 크기가 다른 경우

```
In: df5 = pd.DataFrame({'Class4': [82, 92]})
    df5
Out:
        Class4
    0   82
    1   92
```

```
In: df1.join(df5)
Out:
        Class1   Class2   Class4
    0   95       91       82.0
    1   92       93       92.0
    2   98       97       NaN
    3   100      99       NaN
```

구조적 데이터 표시와 처리에 강한 pandas

- 특정 열을 기준으로 통합하기

```
DataFrame_left_data.merge(DataFrame_right_data)
```

```
In: df_A_B = pd.DataFrame({'판매월': ['1월', '2월', '3월', '4월'],
                           '제품A': [100, 150, 200, 130],
                           '제품B': [90, 110, 140, 170]})
    df_A_B
Out:
        제품A    제품B    판매월
    0   100      90       1월
    1   150      110      2월
    2   200      140      3월
    3   130      170      4월
```

```
In: df_C_D = pd.DataFrame({'판매월': ['1월', '2월', '3월', '4월'],
                           '제품C': [112, 141, 203, 134],
                           '제품D': [90, 110, 140, 170]})
    df_C_D
Out:
        제품C    제품D    판매월
    0   112      90       1월
    1   141      110      2월
    2   203      140      3월
    3   134      170      4월
```

구조적 데이터 표시와 처리에 강한 pandas

- 특정 열을 기준으로 통합하기

```
In: df_A_B.merge(df_C_D)
Out:
      제품A    제품B    판매월    제품C    제품D
0     100      90       1월       112      90
1     150      110      2월       141      110
2     200      140      3월       203      140
3     130      170      4월       134      170
```

  - 두 개의 DataFrame이 특정 열을 기준으로 일부만 공통된 값을 갖는 경우

```
DataFrame_left_data.merge(DataFrame_right_data, how=merge_method, on=key_label)
```

    - merge() 함수의 how 선택 인자에 따른 통합 방법

how 선택 인자	설명
left	왼쪽 데이터는 모두 선택하고 지정된 열(key)에 값이 있는 오른쪽 데이터를 선택
right	오른쪽 데이터는 모두 선택하고 지정된 열(key)에 값이 있는 왼쪽 데이터를 선택
outer	지정된 열(key)을 기준으로 왼쪽과 오른쪽 데이터를 모두 선택
inner	지정된 열(key)을 기준으로 왼쪽과 오른쪽 데이터 중 공통 항목만 선택(기본값)

구조적 데이터 표시와 처리에 강한 pandas

– 두 개의 DataFrame이 특정 열을 기준으로 일부만 공통된 값을 갖는 경우

```
In: df_left = pd.DataFrame({'key':['A','B','C'], 'left': [1, 2, 3]})
    df_left
Out:
        key     left
    0   A       1
    1   B       2
    2   C       3
```

```
In: df_right = pd.DataFrame({'key':['A','B','D'], 'right': [4, 5, 6]})
    df_right
Out:
        key     right
    0   A       4
    1   B       5
    2   D       6
```

```
In: df_left.merge(df_right, how='left', on = 'key')
Out:
          key     left    right
    0     A       1       4.0
    1     B       2       5.0
    2     C       3       NaN
```

구조적 데이터 표시와 처리에 강한 pandas

– 두 개의 DataFrame이 특정 열을 기준으로 일부만 공통된 값을 갖는 경우

```
In: df_left.merge(df_right, how='right', on = 'key')
Out:
    key   left   right
0   A     1.0    4
1   B     2.0    5
2   D     NaN    6
```

```
In: df_left.merge(df_right, how='outer', on = 'key')
Out:
    key   left   right
0   A     1.0    4.0
1   B     2.0    5.0
2   C     3.0    NaN
3   D     NaN    6.0
```

```
In: df_left.merge(df_right, how='inner', on = 'key')
Out:
    key   left   right
0   A     1      4
1   B     2      5
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 파일을 읽고 쓰기
  - 표 형식의 데이터 파일을 읽기

```
DataFrame_data = pd.read_csv(file_name [, options])
```

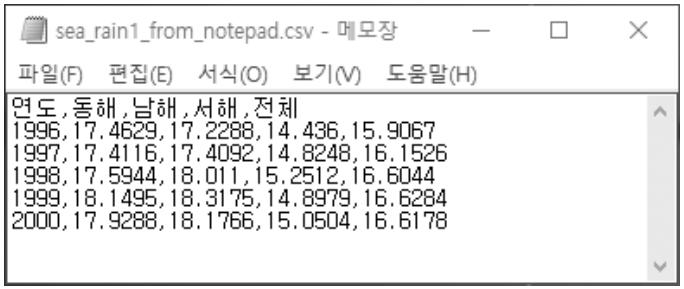
```
In: %%writefile C:₩myPyCode₩data₩sea_rain1.csv
    연도,동해,남해,서해,전체
    1996,17.4629,17.2288,14.436,15.9067
    1997,17.4116,17.4092,14.8248,16.1526
    1998,17.5944,18.011,15.2512,16.6044
    1999,18.1495,18.3175,14.8979,16.6284
    2000,17.9288,18.1766,15.0504,16.6178
Out: Writing C:₩myPyCode₩data₩sea_rain1.csv
```

```
In: import pandas as pd
    pd.read_csv('C:/myPyCode/data/sea_rain1.csv')
Out:
```

	연도	동해	남해	서해	전체
0	1996	17.4629	17.2288	14.4360	15.9067
1	1997	17.4116	17.4092	14.8248	16.1526
2	1998	17.5944	18.0110	15.2512	16.6044
3	1999	18.1495	18.3175	14.8979	16.6284
4	2000	17.9288	18.1766	15.0504	16.6178

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터 파일을 읽기




```
In: pd.read_csv('C:/myPyCode/data/sea_rain1_from_notepad.csv', encoding = "cp949")
Out:
```

	연도	동해	남해	서해	전체
0	1996	17.4629	17.2288	14.4360	15.9067
1	1997	17.4116	17.4092	14.8248	16.1526
2	1998	17.5944	18.0110	15.2512	16.6044
3	1999	18.1495	18.3175	14.8979	16.6284
4	2000	17.9288	18.1766	15.0504	16.6178

```
In: %%writefile C:₩myPyCode₩data₩sea_rain1_space.txt
    연도 동해 남해 서해 전체
    1996 17.4629 17.2288 14.436 15.9067
    1997 17.4116 17.4092 14.8248 16.1526
    1998 17.5944 18.011 15.2512 16.6044
    1999 18.1495 18.3175 14.8979 16.6284
    2000 17.9288 18.1766 15.0504 16.6178
Out: Writing C:₩myPyCode₩data₩sea_rain1_space.txt
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터 파일을 읽기

```
In: pd.read_csv('C:/myPyCode/data/sea_rain1_space.txt', sep=" ")
Out:
```

	연도	동해	남해	서해	전체
0	1996	17.4629	17.2288	14.4360	15.9067
1	1997	17.4116	17.4092	14.8248	16.1526
2	1998	17.5944	18.0110	15.2512	16.6044
3	1999	18.1495	18.3175	14.8979	16.6284
4	2000	17.9288	18.1766	15.0504	16.6178

```
In: pd.read_csv('C:/myPyCode/data/sea_rain1.csv', index_col="연도" )
Out:
```

연도	동해	남해	서해	전체
1996	17.4629	17.2288	14.4360	15.9067
1997	17.4116	17.4092	14.8248	16.1526
1998	17.5944	18.0110	15.2512	16.6044
1999	18.1495	18.3175	14.8979	16.6284
2000	17.9288	18.1766	15.0504	16.6178

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
DataFrame_data = pd.to_csv(file_name [, options])
```

```
In: df_WH = pd.DataFrame({'Weight':[62, 67, 55, 74],
                          'Height':[165, 177, 160, 180]},
                           index=['ID_1', 'ID_2', 'ID_3', 'ID_4'])
    df_WH.index.name = 'User'
    df_WH
Out:
            Height      Weight
      User
      ID_1    165         62
      ID_2    177         67
      ID_3    160         55
      ID_4    180         74
```

```
In: bmi = df_WH['Weight']/(df_WH['Height']/100)**2
    bmi
Out: User
     ID_1    22.773186
     ID_2    21.385936
     ID_3    21.484375
     ID_4    22.839506
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
In: df_WH['BMI'] = bmi
    df_WH
Out:
              Height      Weight       BMI
       User
       ID_1     165          62          22.773186
       ID_2     177          67          21.385936
       ID_3     160           55          21.484375
       ID_4     180          74          22.839506
```

```
In: df_WH.to_csv('C:/myPyCode/data/save_DataFrame.csv')
```

```
In: !type C:₩myPyCode₩data₩save_DataFrame.csv
Out: User,Height,Weight,BMI
    ID_1,165,62,22.77318640955005
    ID_2,177,67,21.38593635289987
    ID_3,160,55,21.484374999999996
    ID_4,180,74,22.839506172839506
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
In: df_pr = pd.DataFrame({'판매가격':[2000, 3000, 5000, 10000],
                          '판매량':[32, 53, 40, 25]},
                          index=['P1001', 'P1002', 'P1003', 'P1004'])
    df_pr.index.name = '제품번호'
    df_pr
Out:
                판매가격     판매량
      제품번호
      P1001       2000        32
      P1002       3000        53
      P1003       5000        40
      P1004       10000        25
```

```
In: file_name = 'C:/myPyCode/data/save_DataFrame_cp949.txt'
    df_pr.to_csv(file_name, sep=" ", encoding = "cp949")
```

```
In: !type C:₩myPyCode₩data₩save_DataFrame_cp949.txt
Out: 제품번호 판매가격 판매량
     P1001 2000 32
     P1002 3000 53
     P1003 5000 40
     P1004 10000 25
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
In: df_pr = pd.DataFrame({'판매가격':[2000, 3000, 5000, 10000],
                          '판매량':[32, 53, 40, 25]},
                          index=['P1001', 'P1002', 'P1003', 'P1004'])
    df_pr.index.name = '제품번호'
    df_pr
Out:
                판매가격     판매량
      제품번호
      P1001       2000        32
      P1002       3000        53
      P1003       5000        40
      P1004       10000        25
```

```
In: file_name = 'C:/myPyCode/data/save_DataFrame_cp949.txt'
    df_pr.to_csv(file_name, sep=" ", encoding = "cp949")
```

```
In: !type C:₩myPyCode₩data₩save_DataFrame_cp949.txt
Out: 제품번호 판매가격 판매량
     P1001 2000 32
     P1002 3000 53
     P1003 5000 40
     P1004 10000 25
```

감사합니다.

※ 본 교안은 강의 수강 용도로만 사용 가능합니다.
상업적 이용을 일절 금함.

사람투자 과학기술정보통신부 nipa 정보통신산업진흥원 National IT Industry Promotion Agency KSA 한국표준협회