부록 A. 파이썬 프로그래밍 기초

A.1 토대 쌓기

A.2 클래스 활용

A.3 기본 자료구조: 리스트와 튜플, 딕셔너리, 셋

A.4 numpy 라이브러리

A.5 matplotlib 라이브러리

A.1 토대 쌓기

백문 불여일코딩이다. 바로 코딩을 시작한다.

짝수 판별 프로그램: if 선택문

[프로그램 A-1]은 사용자가 입력한 정수가 짝수인지 홀수인지 판별하는 프로그램이다.

[프로그램 A-1] 짝수 홀수 판정

```
01  num=input('정수를 입력하세요. ')
02  i=int(num)
03
04  if i%2==0:
05      print(i,'는 짝수입니다.')
06  else:
07      print(i,'는 홀수입니다.')
```

```
정수를 입력하세요. 35
35 는 홀수입니다.
```

01행의 input('정수를 입력하세요.')는 화면에 '정수를 입력하세요.'를 출력한 후 키보드 입력을 기다리는 명령어다. 사용자가 키보드에서 연속으로 문자를 눌러 문자열character string을 입력하고 <Enter> 키를 누르면 문자열을 반환한다. 01행은 input 명령어가 반환한 문자열을 num 변수에 저장한다. 파이썬은 입력된 모든 키를 문자로 간주하기 때문에 사용자가 '3' 키, '5' 키, <Enter> 키를 입력하면 숫자 35가 아니라 문자열 '35'를 반환한다. num은 문자열이기 때문에 num+2나 num/2와 같은 연산을 수행할 수 없다.

Tip 1: 파이썬에서 문자열을 표현하려면 'python'과 같이 따옴표로 묶는다. '...'의 작은 따옴표와 "..."의 큰 따옴표 중 어느 것을 사용해도 된다. 단 'python"처럼 섞어 쓰는 것은 안된다.

02행의 int 함수는 정수 형integer type으로 변환해주는 역할을 한다. 따라서 i=int(num) 명령어는 num을 정수로 변환하여 i 변수에 저장한다. 이제 i+2나 i/2와 같은 연산이 가능하다. int는 함수function, num은 인수argument라고 부른다. 01~02행은 아래와 같이 한 줄로 써도 된다.

```
i=int(input('정수를 입력하세요.'))
```

04~07행의 if~else~ 명령어는 조건을 검사하고 참일 때와 거짓일 때 서로 다른 일을 해준다. if 문을 선택문selection statement이라 부르는 이유이다. 04행에서 i%2==0이 조건이다. %은 나머지를 구해주는 연산자operator, ==은 같은 지 비교하는 연산자이기 때문에, i%2==0이 참이면 i는 짝수다. if 문의 끝에는 :을 덧붙여야 하는데, 뒤에 따르는 들여쓰기indentation 되어있는 명령어 블록이 if 문에 속한다는 뜻이다. if 문은 조건을 검사한 다음 조건이 참이면 소속된 명령어 블록을 수행한다. 이 경우에는 05행이 명령어 블록에 해당하므로 i%2==0이 참이면 05행을 수행하여 i가 짝수라는 메시지를 출력한다. 조건이 거짓이면 06행의 else에 딸린 명령어 블록을 수행한다. 이때는 07행에서 i는 홀수라는 메시지를 출력한다.

Tip 2: 파이썬은 들여쓰기를 통해 명령어 블록의 소속을 지정한다. 들여쓰기의 깊이는 자유인데, 같은 블록에 속하는 행은 모두 같은 깊이를 가져야 한다.

프로그램 실행 결과를 보면, 사용자가 35를 입력했는데 홀수라고 제대로 출력했다. 사용자의 입력을 파란색으로 표시해 프로그램의 출력 내용과 구분했다. 여러 번 실행하여 항상 옳은 판단을 하는지 테스트해보길 바란다.

화씨를 섭씨로 변환하는 프로그램: 수식 코딩

조금 실용적인 프로그램으로 한발 내딛어보자. [프로그램 A-2]는 화씨 온도를 섭씨로 변환해주는 프로그램이다. 01행은 키보드에서 화씨 온도를 입력 받아 num 변수에 저장한다. 02행은 float 함수를 사용하여 문자열을 실수로 변환하여 fahrenheit 변수에 저장한다.

Tip 3: 컴퓨터에서는 정수와 실수를 표현하고 계산하는 방법이 다르다. 프로그래밍에서도 둘을 잘 구분해야 한다. 파이썬에서는 52-30.27처럼 정수와 실수를 섞어 쓰면, 정수를 실수로 변환하여 52.0-30.27을 계산하고 결과를 실수로 표현한다.

식 (A.1)은 화씨 온도 f를 섭씨 온도 c로 변환하는 공식이다. 04행은 식 (A.1)을 적용하고 결과를 celsius 변수에 저장한다. 수식에 등장하는 모든 숫자에 소수점을 붙여 실수로 표현했다. 02행에서 fahrenheit 변수는 실수가 되었기 때문에, 소수점 없이 32, 5, 9로 표현해도 실수 연산을 수행해주지만 명확하게 소수점을 붙여주는 코딩 습관이 좋다.

$$c = \frac{5(f-32)}{9} \quad \text{(A.1)}$$

프로그램 실행 결과를 보면, 사용자가 87.35를 입력했을 때 30.75를 출력했음을 확인할 수 있다. 여러 번 실행하여 항상 옳게 출력하는지 테스트해보길 바란다.

[프로그램 A-2] 화씨를 섭씨로 변환

```
01  num=input('화씨를 입력하세요. ')
02  fahrenheit=float(num)
03
04  celsius=((fahrenheit-32.0)*5.0)/9.0  # 화씨를 섭씨로 변환하는 수식을 적용
05  print('화씨',fahrenheit,'도는 섭씨',celsius,'도 입니다.')
```

```
화씨를 입력하세요. 87.35
화씨 87.35 도는 섭씨 30.75 도 입니다.
```

온도 변환 표를 작성하는 프로그램: for 반복문

[프로그램 A-2]를 이용하면 언제든 원하는 값을 입력하여 온도 변환을 시도할 수 있다. 그런데 온도 변환을 자주 해야 하는 공장이라면 변환 표를 만들어 벽에 붙여 두면 편리할 것이다. [프로그램 A-3]은 이런 상황에 유용하다.

변환 표를 만드는 프로그램을 짜려면 몇 가지 사항을 신중하게 고려해야 한다. 먼저 시작 값, 끝 값, 증가 값이 필요하다. 여기서는 -30에서 시작하여 1만큼씩 증가하며 150까지 계산한다고 가정하자. 이제 화씨가 -30, -29, -28, ..., 149, 150으로 변하면서 섭씨를 계산하여 출력해야 한다. [프로그램 A-3]은 이 일을 한다.

[프로그램 A-3] 화씨를 섭씨로 변환하는 표 작성

```
01  print('화씨', '        섭씨')
02
03  for fahrenheit in range(-30,151,1):
04          celsius=((fahrenheit-32.0)*5.0)/9.0
05          print(fahrenheit,'-->',celsius)
```

```
화씨          섭씨
-30 --> -34.44444444444444
-29 --> -33.888888888888886
-28 --> -33.333333333333336
... ...
148 --> 64.44444444444444
149 --> 65.0
150 --> 65.55555555555556
```

01행은 변환 표의 헤딩을 출력한다. 적절한 수만큼 공백을 주어 열의 위치를 맞춘다. 03행의 for

명령어는 자신에 속한 04~05행의 명령어 블록을 반복한다. for 문은 보통 range 함수와 같이 사용하는데, range(a,b,c)는 a에서 시작하여 c만큼씩 증가시키며 b-1까지 발생시킨다. b보다 하나 작은 b-1임에 주의해야 한다. 따라서 03행의 range(-30,151,1)은 -30, -29, ..., 149, 150을 발생시킨다. for 문은 :으로 끝나는데 들여쓰기로 표시된 명령어 블록을 수행한다. 03~05행의 for 문은 fahrenheit=-30을 가지고 04~05행의 명령어 블록을 수행하고, fahrenheit=-29를 가지고 명령어 블록을 수행하고, fahrenheit=-28을 가지고 명령어 블록을 수행하고, ..., fahrenheit=150을 가지고 명령어 블록을 수행한다. 명령어 블록을 구성하는 04행은 fahrenheit에 있는 화씨 값을 섭씨로 변환하여 celsius에 저장하고, 05행은 celsius를 출력한다.

Tip 4: c를 생략한 range(a,b)는 c=1로 간주한다. 따라서 range(-30,151,1)과 range(-30,151)은 같다. range(10)은 a와 c를 생략한 형태인데, a=0과 c=1로 간주하여 0,1,...,9를 발생시킨다.

프로그램 실행 결과를 보면, -30부터 150까지 온도 변환을 하고 한 줄 씩 출력했음을 확인할 수 있다.
그런데 소수점 15자리까지 출력하여 지저분한 느낌이다. 반올림하는 자리를 지정할 수 있게 하여 융통성을 확보하려면 어떻게 해야 할까? [프로그램 A-4]는 round 함수를 사용하여 2자리까지 출력한다. [프로그램 A-3]에서 달라진 곳을 회색으로 표시했다. round(p,q)는 실수 p를 소수점 q자리에서 반올림한다. 실행 결과를 보면, 소수점 2자리까지 깔끔하게 출력되었음을 확인할 수 있다.

Tip 5: 인터넷에서 '반올림 자리를 지정하는 방법'과 같은 검색어를 입력하면 round 함수로 해결할 수 있다는 사실을 쉽게 알아낼 수 있다. 프로그래머는 검색을 통해 문제를 해결해가는 일이 몸에 배야 한다. 검색은 코딩에서 비단길이다.

[프로그램 A-4] 화씨를 섭씨로 변환하는 표 작성: round 함수로 소숫점 자리 지정

```
01  print('화씨', '      섭씨')
02
03  for fahrenheit in range(-30,151,1):
04      celsius=round(((fahrenheit-32.0)*5.0)/9.0,2)
05      print(fahrenheit,'-->',celsius)
```

```
화씨        섭씨
-30 --> -34.44
-29 --> -33.89
```

```
-28 --> -33.33
... ...
```

Tip 6: [프로그램 A-4]의 04행이 사용하는 round 함수는 파이썬이 제공하는 내장 함수built-in function 중의 하나다. [프로그램 A-1] 2행과 [프로그램 A-2] 2행에서 사용한 int와 float, [프로그램 A-4] 03행의 range도 내장 함수다. 또다른 내장 함수를 예로 들자면 최저와 최고를 구해주는 min과 max가 있는데, 예를 들어 max(12,34,22,17)은 34가 된다. https://docs.python.org/3/library/functions.html에 접속하면 내장 함수 목록과 기능을 확인할 수 있다.

온도 변환 표를 작성하는 프로그램: while 반복문

파이썬은 for 이외에 while 반복문을 제공한다. while 문은 조건을 검사하여 참이면 소속된 명령어 블록을 수행하는 일을 반복하다가 조건이 거짓이 될 때 끝낸다. [프로그램 A-5]는 [프로그램 A-4]를 while로 바꿔 쓴 것이다.

[프로그램 A-5] 화씨를 섭씨로 변환하는 표 작성: while 반복문

```
01  print('화씨', '      섭씨')
02
03  fahrenheit=-30
04  while fahrenheit<=150:
05      celsius=round(((fahrenheit-32.0)*5.0)/9.0,2)
06      print(fahrenheit,'-->',celsius)
07      fahrenheit=fahrenheit+1
```

[프로그램 A-4]와 같음

03행은 시작 값을 설정한다. 04행의 while 문은 조건 fahrenheit<=150이 참일 동안 명령어 블록 05~07을 반복한다. 05~06행은 현재 화씨 값을 섭씨로 변환하여 출력하고, 07행은 fahrenheit를 1만큼 증가시켜 다음 반복을 준비한다.

Tip 7: for와 while은 둘 다 반복에 쓰는데, 상황에 따라 적절하게 골라 쓰는 요령이 필요하다. 일정한 양만큼 씩 증가하며 같은 일을 반복하는 온도 변환 표 작성과 같은 일에는 for를 쓰면 훨씬 깔끔한 코드가 된다. 따라서 [프로그램 A-5]보다 [프로그램 A-4]가 좋다. 같은 일을 반복하다가 사용자가 지정된 키를 누르면 멈추는 일 같은 경우에는 for보다 while을 써야 한다.

코딩을 오래 하다 보면 자연스럽게 골라 쓰는 요령이 생긴다.

온도 변환 표를 작성하는 프로그램: numpy 모듈의 arange 함수로 실수 구간 지정하기

지금까지 파이썬이 기본으로 제공하는 range라는 내장 함수를 사용하여 표 작성에 필요한 구간을 생성했다. 그런데 range는 정수 구간만 지원하는 한계가 있다. 0.5도 간격으로 표를 작성하려면 다른 방법을 찾아야 한다. Tip 5에서 강조했듯이 코딩에서 검색은 비단길이다. '파이썬에서 실수 표현이 가능한 range'같은 검색어를 사용하면 쉽게 numpy 모듈이 제공하는 arange라는 함수를 알아낼 수 있다.

[프로그램 A-6]은 numpy 모듈의 arange 함수를 사용하여 0.5도 간격의 변환 표를 작성하는 프로그램이다.

[프로그램 A-6] 화씨를 섭씨로 변환하는 표 작성: arange 함수로 실수 구간 사용

```
01  import numpy as np
02
03  print('화씨', '      섭씨')
04
05  for fahrenheit in np.arange(-30,151,0.5):
06      celsius=round(((fahrenheit-32.0)*5.0)/9.0,2)
07      print(fahrenheit,'-->',celsius)
```

```
화씨        섭씨
-30.0 --> -34.44
-29.5 --> -34.17
-29.0 --> -33.89
-28.5 --> -33.61
... ...
```

arange 함수는 파이썬이 기본으로 제공하는 내장 함수가 아니다. 따라서 arange를 제공하는 모듈을 불러온 다음에 사용해야 한다. 01행은 arange를 제공하는 numpy 모듈을 불러와서 np라는 별명을 붙인다. 별명을 붙이는 이유는 긴 이름 대신 짧은 이름을 사용하여 간결함을 유지하려는 데 있다. numpy는 배열을 지원하는 모듈인데, 파이썬 모듈 중에 가장 유명하고 널리 쓰인다. A.4절에서 numpy 사용법을 자세히 설명한다. 모듈은 유용한 함수를 여럿 제공하는데, numpy는 지원하는 함수 개수에서도 다른 모듈을 크게 앞선다. 05행의 np.arange(-30,150,0.5)는 인수로 -30, 150, 0.5를 주고 np가 제공하는 arange 함수를 호출한다.

프로그램 실행 결과를 보면, 0.5도 간격으로 표를 출력했음을 확인할 수 있다.

Tip 8: 모듈이 지원하는 함수를 알아보려면 dir 명령어를 사용하면 된다. 아래는 numpy의 함수를 확인하는 사례다. 첫번째 dir에서는 함수가 너무 많아 앞 부분이 잘려 arange를 확인할 수가 없다. 두번째 dir처럼 하면, 앞 부분에서 100개까지 출력한다. arange뿐 아니라 이름만으로 기능을 추측할 수 있는 abs, add, append, arccos, arcsin 등을 확인할 수 있다. [0:100]은 리스트의 일부를 자르는 슬라이싱인데 리스트를 다루는 A.3절에서 자세히 설명한다.

```
In [1]: dir(np)
    ... ... 'array2string', 'array_equal', ... ..., 'where', 'who', 'zeros', 'zeros_like']
In [2]: dir(np)[0:100]
    ['ALLOW_THREADS', 'AxisError', 'BUFSIZE', ... ..., 'abs', 'absolute', 'add', 'add_docstring',
    'add_newdoc', 'add_newdoc_ufunc', 'alen', 'all', 'allclose', 'alltrue', 'amax', 'amin', 'angle', 'any',
    'append', 'apply_along_axis', 'apply_over_axes', 'arange', 'arccos', 'arccosh', 'arcsin', 'arcsinh']
```

import 사용법

파이썬은 유용한 모듈을 아주 많이 보유하고 있다. 이 책에서는 numpy, matplotlib, opencv, pyqt, os, winsound, mediapipe, pixellib, transformer 같은 다양한 모듈을 사용한다. 기능이 방대한 모듈의 경우 라이브러리라고 부르기도 한다.

모듈은 import 명령어를 통해 프로그램 안으로 불러와야 쓸 수 있다. import를 사용하는 방식이 여럿이다. [프로그램 A-6] 1행은 표 A.1이 소개하는 방식1에 해당한다. 방식1을 가장 널리 쓴다. 방식2는 방식1과 같은데 단지 모듈 이름을 그대로 사용하는 차이만 있다. 방식1과 방식2는 모듈이 제공하는 모든 함수를 불러온다. 방식3을 사용하면 필요한 함수를 일부만 가져와 메모리 효율을 높일 수 있다. 표 A.1에서 방식3의 예시를 보면, numpy 모듈에서 arange 함수만 불러왔다. 방식3에서는 from numpy import arange,abs,append처럼 필요한 만큼 여러 함수를 불러 올 수 있다. 모든 함수를 가져오려면 from numpy import *처럼 하면 된다. 방식3으로 함수를 불러오면 함수를 호출할 때 모듈 이름 없이 함수 이름만 표시해야 한다.

표 A.1 import를 사용하는 방식

방식	코딩	[프로그램 A-6] 1행 예시	[프로그램 A-6] 5행 예시
1	import <모듈> as <별명>	import numpy as np	np.arange(-30,151,0.5)
2	import <모듈>	import numpy	numpy.arange(-30,151,0.5)

3	from <모듈> import <함수>	from numpy import arange	arange(-30,151,0.5)

온도 변환 표를 작성하는 프로그램: 함수 사용하기

프로그램이 수천~수만 행이 되었고 군데 군데에서 화씨를 섭씨로 변환하는 기능을 사용한다고 상상해보자. 이때는 04행의 수식이 여러 군데 나타날 것이고, 수식이 날것으로 표현된 셈이 된다. 표를 더 깔끔하게 하려고 소수점 한자리까지 표시한다면 04행이 celsius=round(((fahrenheit-32.0)*5.0)/9.0,1)로 바뀌어야 하는데, 수 만 행을 조사하여 일일이 해당하는 행을 찾아 바꾸는 일은 매우 번거로울 뿐 아니라 실수로 빠트리는 경우가 발생할 위험이 있다. 함수는 이런 경우에 매우 유용하다.

[프로그램 A-7]은 [프로그램 A-6]을 함수를 사용하여 개선한 버전이다. 03~05행은 fahrenheit_to_celsius라는 함수를 정의하고, 07~11행은 메인에 해당한다. 프로그램의 실행은 메인에서 시작하기 때문에 07행에서 시작한다. 07행은 변환 표의 헤딩을 출력하고, 09행은 arange 함수가 발생해주는 -30.0, -29.5, ..., 150.5를 가지고 10~11행의 명령어 블록을 반복한다. 10행의 fahrenheit_to_celsius(fahrenheit)는 함수를 호출$_{call}$한다. 괄호 속의 fahrenheit를 인수$_{argument}$라 부르는데, 인수는 함수에게 정보를 전달하는 역할을 한다. 다시 말해 10행은 fahrenheit 값을 가지고 fahrenheit_to_celsius 함수를 호출하고, 함수가 반환해주는 값을 celsius 변수에 저장한다.

03행은 함수의 머리다. 함수 머리는 def를 쓰고, 함수 이름을 쓰고, 괄호 속에 매개변수$_{parameter}$를 쓰고, :을 덧붙인다. 함수 머리에서 이름은 fahrenheit_to_celsius, 매개변수는 val이다. 매개변수는 10행의 함수 호출에서 오는 인수의 값을 받는다. 예를 들어 10의 인수 fahrenheit가 -30.0이라면 매개변수 val이 -30.0이 되어 함수에 딸린 04~05행의 명령어 블록을 실행한다. 04행은 val을 가지고 수식을 계산하고 결과인 -34.44444를 result에 저장한다. 05행은 return 문으로 result를 반환하고 함수를 종료한다. 함수가 끝나면 호출한 곳, 즉 10행으로 돌아간다. 10행은 함수가 반환한 값 -34.44444를 celsius에 저장한다.

[프로그램 A-7] 화씨를 섭씨로 변환하는 표 작성(함수 버전)

```
01  import numpy as np
02
03  def fahrenheit_to_celsius(val): # 화씨->섭씨 변환(소수점 2자리에서 반올림)
04      result=round(((val-32.0)*5.0)/9.0,2)
05      return result
06
07  print('화씨', '      섭씨')
```

```
08
09  for fahrenheit in np.arange(-30,151,0.5):
10      celsius=fahrenheit_to_celsius(fahrenheit)
11      print(fahrenheit,'-->',celsius)
```

[프로그램 A-6]과 같음

[프로그램 A-7]의 fahrenheit_to_celsius 함수는 반올림 자리가 2로 고정되어 있다. 반올림 자리를 지정하는 매개변수를 추가하면 융통성이 생겨 훨씬 유용한 함수가 된다. [프로그램 A-8]은 자리를 지정할 수 있게 확장한 프로그램이다. 달라진 곳을 음영 표시하여 쉽게 알아볼 수 있게 했다.

03행의 함수 머리를 보면, 화씨를 나타내는 val 매개변수와 반올림 자리를 나타내는 pos 매개변수가 있다. 04행에서는 이전의 고정된 값 2 대신 매개변수 pos로 반올림 자리를 지정한다. 10행의 함수 호출에서는 매개변수 pos에 해당하는 인수를 3으로 지정하여 3자리에서 반올림하라고 요청한다. 인수의 개수와 매개변수의 개수는 같아야 한다. 뒤에서는 기본값을 사용하여 개수가 달라도 되는 편리한 기능을 소개한다.

[프로그램 A-8] 화씨를 섭씨로 변환하는 표 작성(함수 버전, 반올림 자리 지정)

```
01  import numpy as np
02
03  def fahrenheit_to_celsius(val,pos): # 화씨->섭씨 변환(pos자리에서 반올림)
04      result=round(((val-32.0)*5.0)/9.0,pos)
05      return result
06
07  print('화씨', '      섭씨')
08
09  for fahrenheit in np.arange(-30,151,0.5):
10      celsius=fahrenheit_to_celsius(fahrenheit,3)
11      print(fahrenheit,'-->',celsius)
```

```
화씨        섭씨
-30.0 --> -34.444
-29.5 --> -34.167
-29.0 --> -33.889
… …
```

간결한 코딩을 좋아하는 프로그래머가 많다. 파이썬은 이런 목적으로 잘 발달되어 있다. [프로그램

A-9]는 [프로그램 A-8]을 간결하게 고쳐 쓴 코드다.

[프로그램 A-8]에서 04행은 변환된 온도를 celsius에 저장한 다음 05행에서 반환한다. 두 명령어를 [프로그램 A-9] 04행처럼 한 줄로 쓸 수 있다. 또한 [프로그램 A-8]에서 10행은 함수가 반환한 값을 celsius에 저장한 다음 11행에서 출력한다. 두 명령어를 [프로그램 A-9] 09행처럼 한 줄로 쓸 수 있다

[프로그램 A-9] 화씨를 섭씨로 변환하는 표 작성(간결한 버전)

```
01  import numpy as np
02
03  def fahrenheit_to_celsius(val,pos): # 화씨->섭씨 변환(pos자리에서 반올림)
04      return round(((val-32.0)*5.0)/9.0,pos)
05
06  print('화씨', '      섭씨')
07
08  for fahrenheit in np.arange(-30,151,0.5):
09      print(fahrenheit,'-->',fahrenheit_to_celsius(fahrenheit,3))
```

[프로그램 A-8]과 같음

함수에서 기본값 지정하기

파이썬은 매개변수에 기본값$_{default\ \ value}$을 지정하는 편리한 기능이 있다. [프로그램 A-10]은 [프로그램 A-9]를 기본값 지정 버전으로 확장한다. pos라는 매개변수를 pos=1로 바꾸면, 함수 호출할 때 해당 인수를 생략할 수 있다. 이렇게 하면 pos는 기본값으로 1을 가지게 되는데, pos를 생략하면 pos는 기본값으로 동작한다. 09행에서 pos에 해당하는 인수를 생략했기 때문에, 프로그램 실행 결과를 보면 1자리에서 반올림했음을 확인할 수 있다.

Tip 9: 커피점에서 특별히 지정하지 않으면 기본 크기의 커피를 제공한다. 기본 크기는 많은 사람이 선호하는 크기로 정해 놓는다. 함수를 만드는 프로그래머도 여러 사람이 선호하는 값을 기본값으로 지정한다. 예를 들어, Tip 4의 range(a,b,c)에서 c를 생략하여 range(a,b)로 쓸 수 있는데 기본값이 지정되어 있어 생략 가능하다. 보통 1씩 증가시키며 작업하는 경우가 많기 때문에 range 함수를 작성한 사람은 c의 기본값으로 1을 설정해 두었다.

[프로그램 A-10] 화씨를 섭씨로 변환하는 표 작성: 기본값 지정

```
01  import numpy as np
```

```
02
03  def fahrenheit_to_celsius(val,pos=1): # 화씨->섭씨 변환(pos자리에서 반올림, 기본은 1자리)
04      return round(((val-32.0)*5.0)/9.0,pos)
05
06  print('화씨', '      섭씨')
07
08  for fahrenheit in np.arange(-30,151,0.5):
09      print(fahrenheit,'-->',fahrenheit_to_celsius(fahrenheit))
```

```
화씨        섭씨
-30.0 -->  -34.4
-29.5 -->  -34.2
-29.0 -->  -33.9
… …
```

위치 인수와 키워드 인수

지금까지는 화씨 온도를 섭씨로 변환하는 기능은 fahrenheit_to_celsius 함수로 수행하고 표를 만드는 일은 메인에서 수행하는 방식으로 코딩했다. 실제 프로그래밍에서는 아주 많은 다양한 함수를 만들어 두고 호출해 사용하는 방식의 코딩을 주로 한다. 실제 상황에 조금 다가가는 의미에서, 표를 출력하는 기능을 함수로 작성해보자.

[프로그램 A-11]의 03~04행은 fahrenheit_to_celsius 함수, 06~09행은 변환 표를 출력하는 fahrenheit_to_celsius_table 함수를 정의하고, 11행이 메인에 해당한다. fahrenheit_to_celsius 함수는 [프로그램 A-10]과 같다. 06의 fahrenheit_to_celsius_table 함수는 3개 매개변수를 가지는데, 모두 기본값을 지정했다. 따라서 11행처럼 함수 호출할 때 인수를 생략하면 start는 0.0, end는 100.0, inc는 0.5를 가지고 동작한다.

함수 선언에서 매개변수를 지정할 때, 기본값을 가진 매개변수 오른쪽에는 기본값이 없는 매개변수를 둘 수 없다. 예를 들어, 06행에서 fahrenheit_to_celsius_table(start,end=100.0,inc=0.5)는 허용되지만, fahrenheit_to_celsius_table(start,end=100.0,inc)은 허용되지 않는다. 따라서 기본값이 없는 매개변수를 앞쪽에 배치하고 기본값이 있는 매개변수를 뒤쪽에 배치하는 것이 관행이다.

[프로그램 A-11] 화씨를 섭씨로 변환하는 표 작성: 기본값 지정

```
01  import numpy as np
02
03  def fahrenheit_to_celsius(val,pos=1): # 화씨->섭씨 변환(pos자리에서 반올림, 기본은 1자리)
04      return round(((val-32.0)*5.0)/9.0,pos)
```

```
05
06  def fahrenheit_to_celsius_table(start=0.0,end=100.0,inc=0.5): # 온도 변환 표 작성
07      print('화씨', '      섭씨')
08      for fahrenheit in np.arange(start,end,inc):
09          print(fahrenheit,'-->',fahrenheit_to_celsius(fahrenheit))
10
11  fahrenheit_to_celsius_table()
```

```
화씨        섭씨
0.0 --> -17.8
0.5 --> -17.5
1.0 --> -17.2
... ...
```

기본값을 가진 함수를 호출하는 방식은 다양하다. ①~④처럼 위치를 통해 매개변수와 대응을 결정하는 인수를 위치 인수$_{positional\ argument}$라 부른다. ②의 경우 인수 10.5를 첫번째 매개변수 start에 할당한 다음 생략된 두번째와 세번째 매개변수에는 기본값을 할당한다. ⑤~⑥과 같이 매개변수 이름을 명시적으로 표기하는 인수를 키워드 인수$_{keyword\ argument}$라 한다. 키워드 인수를 사용하면, 인수의 배치 순서가 매개변수 배치 순서와 달라도 된다. ⑥처럼 키워드 인수를 하나만 주는 경우, 나머지가 모두 기본값이 있다면 나머지는 기본값이 할당된다. ⑦~⑧은 위치 인수와 키워드 인수를 섞어 쓰는 경우다. ⑦은 제대로 작동한다. 하지만 ⑧처럼 키워드 인수 다음에 위치 인수가 나오면 오류다.

표 A.2 기본값을 가진 함수 호출 방식

함수 호출 명령어	매개변수 값 결정
①fahrenheit_to_celsius_table()	start=0.0,end=100.0,inc=0.5
②fahrenheit_to_celsius_table(10.5)	start=10.5,end=100.0,inc=0.5
③fahrenheit_to_celsius_table(10.5,200.0)	start=10.5,end=200.0,inc=0.5
④fahrenheit_to_celsius_table(10.5,200.0,2.0)	start=10.5,end=200.0,inc=2.0
⑤fahrenheit_to_celsius_table(inc=0.5,start=10.0,end=200.0)	start=10.5,end=200.0,inc=0.5
⑥fahrenheit_to_celsius_table(inc=0.5)	start=10.5,end=100.0,inc=0.5
⑦fahrenheit_to_celsius_table(0.0,inc=0.5,end=10.0)	start=0.0,end=100.0,inc=0.5
⑧fahrenheit_to_celsius_table(start=0.0,10.0,0.5)	오류

Tip 10: 파이썬이 인수를 매개변수에 대응할 때는 애매함 배제 원칙을 적용한다. 애매함이 없는

경우에는 규칙에 따라 쌍을 맺어주지만, 애매함이 있는 경우 오류를 발생시킨다.

A.2 클래스 활용

현대 프로그래밍 언어는 대부분 객체지향 언어object-oriented language다. 파이썬은 객체지향 언어다. C++는 객체지향이지만 C는 객체지향 언어가 아니다.

객체지향 언어는 클래스class를 지원한다. 클래스를 잘 활용하면 프로그램을 깔끔하게 설계할 수 있고 다른 프로그램에서 가져다 쓸 수 있는 재사용성reusability이 좋아진다. A.1절에서 연습해본 화씨-섭씨 변환 프로그램을 가지고 왜 클래스를 사용하는 지와 어떻게 사용하는 지를 익혀보자.

[프로그램 A-11]은 함수와 매개변수를 잘 설계하여 제법 융통성이 있지만, 기능을 확장하려고 함수를 늘려 나가면 프로그램이 파편화되는 한계가 있다. [프로그램 A-11]은 화씨를 섭씨로 변환하는 기능만 있는데 섭씨를 화씨로 변환하는 함수를 추가하면 프로그램의 완벽성이 좋아진다. 게다가 다양하게 단위 변환을 해주는 다목적 프로그램으로 확장하려면 온도 변환 뿐만 아니라, 길이와 무게, 넓이, 부피 등을 다룰 수 있어야 한다. 이때 온도 변환을 담당하는 함수들은 논리적으로 서로 연관되어 있지만, 자칫 무게나 부피 변환 함수들과 이리저리 섞여 혼란스러운 상황이 발생할 가능성이 높다. 이때 클래스를 이용하면 논리적으로 연관된 기능을 하나의 묶음으로 구성할 수 있다.

클래스 선언과 객체 만들기

[프로그램 A-12]는 [프로그램 A-11]을 클래스를 이용하여 개선한 프로그램이다. 03~11행은 TemperatureConversion이라는 클래스를 선언한다. 13~16행은 메인에 해당한다.

메인이 시작하는 13행은 TemperatureConversion 클래스로 t라는 변수를 만든다. 객체지향에서는 일반 변수와 구별하기 위해 클래스로 만든 변수를 객체object라고 부른다. 다시 말해 13행은 TemperatureConversion 클래스 형의 객체 t를 만든다.

클래스를 시작하는 03행에서는 class를 쓰고, 클래스 이름 TemperatureConversion을 쓰고, :을 덧붙인다. TemperatureConversion 클래스는 fahrenheit_to_celsius와 fahrenheit_to_celsius_table이라는 두 개의 함수를 가진다. 이들처럼 클래스에 소속된 함수를 멤버 함수member function 또는 메서드method라고 부른다.

15행의 t.fahrenheit_to_celsius_table()은 객체 t가 가진 fahrenheit_to_celsius_table를 호출한다. 이처럼 객체가 멤버 함수를 호출할 때는 .으로 연결한다. 호출할 때 인수를 모두 생략했기 때문에 fahrenheit_to_celsius_table 멤버 함수는 자신이 가진 기본값을 가지고 동작하여 표를 출력해준다. 실행 결과에서 ①로 표시된 곳이 15행의 결과다. 16행은 100.0과 3의 인수를 가지고 fahrenheit_to_celsius 멤버 함수를 호출한다. fahrenheit_to_celsius 멤버 함수는 계산 결과로 37.778을 반환한다. 실행 결과에서 ②로 표시된 곳이 16행의 결과다.

이제 03~11의 TemperatureConversion 클래스를 자세히 살펴보자. 04~05행의 fahrenheit_to_celsius 멤버 함수와 07~11행의 fahrenheit_to_celsius_table 멤버 함수는 [프로그램 A-11]에 있는 함수와 비슷하다. 다른 점은 회색으로 표시한 self뿐이다. 이처럼 클래스의 멤버 함수는 첫번째 매개변수로 self를 가진다. self 매개변수는 특별한 역할을 한다. 15행에서

t.fahrenheit_to_celsius_table()을 호출하면, 멤버 함수를 호출한 객체 t가 self에 전달된다. 이런 이유로 보통 멤버 함수의 매개변수는 인수보다 하나 많다. 10행처럼 같은 클래스 내에 있는 다른 멤버 함수를 호출할 때는 self를 붙여 self.fahrenheit_to_celsius처럼 표기해야 한다. self를 붙이는 이유는 클래스 밖에 있는 함수와 구별하기 위해서다.

Tip 10: 멤버 함수의 self 매개변수는 다른 이름을 써도 된다. 예를 들어 self 대신 s라고 해도 아무런 문제 없이 잘 작동한다. 하지만 관행에 따라 self라고 쓰는 것이 좋다.

[프로그램 A-12] 화씨를 섭씨로 변환하는 표 작성: 클래스 사용

```
01  import numpy as np
02
03  class TemperatureConversion:
04      def fahrenheit_to_celsius(self,val,pos=1): # 화씨->섭씨 변환(pos자리에서 반올림, 기본은 1자리)
05          return round(((val-32.0)*5.0)/9.0,pos)
06
07      def fahrenheit_to_celsius_table(self,start=0.0,end=100.0,inc=0.5): # 온도 변환 표 작성
08          print('화씨', '      섭씨')
09          for fahrenheit in np.arange(start,end,inc):
10              celsius=self.fahrenheit_to_celsius(fahrenheit)
11              print(fahrenheit,'-->',celsius)
12
13  t=TemperatureConversion()
14
15  t.fahrenheit_to_celsius_table() ①
16  print('₩n화씨 100.0도=섭씨',t.fahrenheit_to_celsius(100.0,3)) ②
```

```
화씨      섭씨 ①
0.0 --> -17.8
0.5 --> -17.5
1.0 --> -17.2
... ...

화씨 100.0도=섭씨 37.778 ②
```

TemperatureConversion 클래스는 논리적으로 연결되어 있는 두 함수를 하나의 묶음으로 구성했다. 이제는 프로그램이 확장되어 넓이나 부피를 처리하는 함수들이 추가되어도 그들과

섞일 일이 없다. 다시 말해 프로그램이 모듈화되는 장점이 있다. 게다가 다른 프로그램에서 온도 변환하는 기능을 필요로 할 때 TemperatureConversion 클래스만 떼어 쉽게 가져다 쓸 수 있어 재사용성이 강화된다.

객체 내부 살펴보기

클래스로 만든 변수, 즉 객체는 내부에 풍부한 기능을 가진다. [프로그램 A-12] 13행으로 만든 객체 t를 아래와 같이 조사해보자. 뒤쪽에 fahrenheit_to_celsius와 fahrenheit_to_celsius_table 멤버 함수가 보인다. __init__처럼 앞과 뒤에 __이 붙은 것이 많은데, 특별한 일을 해주는 멤버 함수 또는 속성$_{attribute}$이다.

```
In [1]: dir(t)
    ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',... ..., '__init__', ... ..., 'fahrenheit_to_celsius',
    'fahrenheit_to_celsius_table']
```

예를 들어 __doc__은 클래스를 설명하는 속성이다. [프로그램 A-13]처럼 클래스 안에 """으로 묶은 문자열을 추가하면 그 내용이 __doc__에 기록된다. 이렇게 기록되고 나면, 아래와 같이 help 명령어를 사용하여 클래스가 무슨 일을 하는지 알아볼 수 있다. 또는 t.__doc__을 print로 출력하여 클래스의 기능을 확인할 수 있다.

```
In [1]: help(t)
    Help on class TemperatureConversion in module __main__:

    class TemperatureConversion(builtins.object)
     |  온도를 변환하는 클래스
     |  멤버 함수 fahrenheit_to_celsius: 화씨를 섭씨로 변환
     |  멤버 함수 fahrenheit_to_celsius_table: 화씨를 섭씨로 변환하는 표 작성
     |
     |  Methods defined here:
     |
     |  fahrenheit_to_celsius(self, val, pos=1)
     |
     |  fahrenheit_to_celsius_table(self, start=0.0, end=100.0, inc=0.5)
     |
     ... ...
```

In [2]: print(t.__doc__)

온도를 변환하는 클래스

멤버 함수 fahrenheit_to_celsius: 화씨를 섭씨로 변환

멤버 함수 fahrenheit_to_celsius_table: 화씨를 섭씨로 변환하는 표 작성

[프로그램 A-13] 클래스의 __doc__ 속성 사용하기

```
01  import numpy as np
02
03  class TemperatureConversion:
04      """
05      온도를 변환하는 클래스
06      멤버 함수 fahrenheit_to_celsius: 화씨를 섭씨로 변환
07      멤버 함수 fahrenheit_to_celsius_table: 화씨를 섭씨로 변환하는 표 작성
08      """
09
10      def fahrenheit_to_celsius(self,val,pos=1): # 화씨->섭씨 변환(pos자리에서 반올림, 기본은 1자리)
11          return round(((val-32.0)*5.0)/9.0,pos)
12
13      def fahrenheit_to_celsius_table(self,start=0.0,end=100.0,inc=0.5): # 온도 변환 표 작성
14          print('화씨', '      섭씨')
15          for fahrenheit in np.arange(start,end,inc):
16              celsius=self.fahrenheit_to_celsius(fahrenheit)
17              print(fahrenheit,'-->',celsius)
... ...
```

클래스의 확장성

클래스를 사용한 프로그램은 확장성이 뛰어나다. [프로그램 A-12]는 화씨를 섭씨로 변환하는 기능만 있는데 섭씨를 화씨로 변환하는 기능까지 확장하려면 멤버 함수를 추가하면 된다.

[프로그램 A-14] 03~20행은 섭씨를 화씨로 변환하는 celsius_to_fahrenheit 멤버 함수와 섭씨를 화씨로 변환하는 표를 작성해주는 celsius_to_fahrenheit_table 멤버 함수를 추가한 클래스 TemperatureConversion을 선언한다. 메인이 시작하는 22행은 TemperatureConversion 클래스형의 객체 t를 생성한다. 아래와 같이 dir로 객체 내부를 조사해 보면, 두 개의 멤버 함수가 추가되었음을 확인할 수 있다.

> dir(t)

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',... ..., '__init__', ..., 'celsius_to_fahrenheit', 'celsius_to_fahrenheit_table', 'fahrenheit_to_celsius', 'fahrenheit_to_celsius_table']

24행은 객체 t의 fahrenheit_to_celsius_table 멤버 함수를 호출하여 변환 표를 출력한다. 25행은 celsius_to_fahrenheit 멤버 함수로 섭씨 0.0도를 변환한 결과를 출력한다.

[프로그램 A-14] 다양한 멤버 함수를 가지는 클래스로 확장

```
01  import numpy as np
02
03  class TemperatureConversion:
04      def fahrenheit_to_celsius(self,val,pos=1): # 화씨->섭씨 변환(pos자리에서 반올림, 기본은 1자리)
05          return round(((val-32.0)*5.0)/9.0,pos)
06
07      def fahrenheit_to_celsius_table(self,start=0.0,end=100.0,inc=0.5): # 온도 변환 표 작성
08          print('화씨', '      섭씨')
09          for fahrenheit in np.arange(start,end,inc):
10              celsius=self.fahrenheit_to_celsius(fahrenheit)
11              print(fahrenheit,'-->',celsius)
12
13      def celsius_to_fahrenheit(self,val,pos=1): # 섭씨->화씨 변환(pos자리에서 반올림, 기본은 1자리)
14          return round((val*9.0)/5.0+32.0,pos)
15
16      def celsius_to_fahrenheit_table(self,start=0.0,end=100.0,inc=0.5):
17          print('섭씨', '      화씨')
18          for celsius in np.arange(start,end,inc):
19              fahrenheit=self.celsius_to_fahrenheit(celsius)
20              print(celsius,'-->',fahrenheit)
21
22  t=TemperatureConversion()
23
24  t.fahrenheit_to_celsius_table() # 화씨->섭씨 변환 표 출력 ①
25  print('₩n섭씨 0도=화씨 ',t.celsius_to_fahrenheit(0.0)) # 섭씨 0.0도에 해당하는 화씨 출력
```

```
화씨        섭씨 ①
0.0 --> -17.8
```

```
0.5 --> -17.5
1.0 --> -17.2
... ...

섭씨 0도=화씨  32.0 ②
```

생성자 __init__ 사용하기

앞에서 객체 내부를 살펴보았을 때 __init__이 있었다. __init__은 생성자$_{constructor}$라는 멤버 함수인데, 객체가 생성될 때 자동으로 실행되기 때문에 매우 유용하고 거의 모든 클래스가 활용한다. [프로그램 A-14]는 생성자를 사용하지 않았는데, 생성자를 사용하면 이점이 있다. 여러 가지 쓸모를 생각할 수 있는데, 여기서는 반올림 자리를 지정하는 데 활용한다.

[프로그램 A-15] 04~05행은 생성자 __init__을 정의한다. 04행에서 매개변수 position을 받아 05행에서 pos 변수에 저장한다. pos 변수는 self.pos로 표기되어 있는데, 이처럼 self가 붙은 변수는 클래스의 멤버 변수$_{member\ variable}$가 된다. 멤버 변수는 클래스 안에 있는 모든 멤버 함수가 접근할 수 있다. 또한 객체 t에 소속되어 있어 t를 통해 접근할 수 있다. pos는 반올림 자리를 나타내는데, fahrenheit_to_celsius 멤버 함수가 08행에서 사용하고 celsius_tofahrenheit 멤버 함수가 17행에서 사용하므로 self를 붙여 멤버 변수로 선언했다.

07~08행에 있는 fahrenheit_to_celsius 멤버 함수는 [프로그램 A-14] 04~05에서 약간 바뀌었다. 매개변수 pos가 사라졌고 대신 생성자가 설정해 놓은 self.pos를 사용한다. 16~17행의 celsius_to_fahrenheit 멤버 함수도 비슷하게 바뀌었다. 나머지 멤버 함수는 [프로그램 A-14]와 같다.

25~35행의 메인을 살펴보면 클래스 사용이 훨씬 편리해졌음을 알 수 있다. 25행의 t1=TemperatureConversion(3)이 실행되면, 객체 t1이 생성되면서 __init__ 생성자가 자동으로 실행된다. 이때 인수로 준 3이 03행의 __init__의 매개변수 position으로 전달되고 04행은 pos 멤버 변수에 position을 저장한다. 따라서 t1 객체로 온도를 변환하면 항상 셋째 자리에서 반올림을 한다. 26행의 t2=TemperatureConversion(1)은 첫째 자리에서 반올림하는 객체 t2를 생성한다. 이제부터는 첫째 자리 반올림을 원하면 t2를 사용하고 셋째 자리 반올림을 원하면 t1을 사용하면 된다. 아주 편리하다.

28행과 29행은 각각 t1과 t2를 사용하여 변환 표를 출력한다. 프로그램 실행 결과를 보면, t1을 사용한 경우에는 셋째 자리까지 출력하고 t2를 사용한 경우에는 첫째 자리까지 출력하는 것을 확인할 수 있다. 31~32행은 섭씨 -0.21도를 t1과 t2로 화씨로 변환하여 출력한다. 예상한 대로 셋째와 첫째 자리까지 출력되었음을 확인할 수 있다.

[프로그램 A-15] 생성자 활용하여 반올림 자리를 융통성 있게 지정하기

```
01  import numpy as np
```

```
02
03  class TemperatureConversion:
04      def __init__(self,position):
05          self.pos=position # 소수점 pos자리에서 반올림
06
07      def fahrenheit_to_celsius(self,val): # 화씨->섭씨 변환
08          return round(((val-32.0)*5.0)/9.0,self.pos)
09
10      def fahrenheit_to_celsius_table(self,start=0.0,end=100.0,inc=0.5):
11          print('화씨', '      섭씨')
12          for fahrenheit in np.arange(start,end,inc):
13              celsius=self.fahrenheit_to_celsius(fahrenheit)
14              print(fahrenheit,'-->',celsius)
15
16      def celsius_to_fahrenheit(self,val): # 섭씨->화씨 변환
17          return round((val*9.0)/5.0+32.0,self.pos)
18
19      def celsius_to_fahrenheit_table(self,start=0.0,end=100.0,inc=0.5):
20          print('섭씨', '      화씨')
21          for celsius in np.arange(start,end,inc):
22              fahrenheit=self.celsius_to_fahrenheit(celsius)
23              print(celsius,'-->',fahrenheit)
24
25  t1=TemperatureConversion(3) # 셋째 자리에서 반올림하는 객체
26  t2=TemperatureConversion(1) # 첫째 자리에서 반올림하는 객체
27
28  t1.fahrenheit_to_celsius_table(0,10,0.5) ①
29  t2.fahrenheit_to_celsius_table(0,10,0.5) ②
30
31  print('₩n섭씨 -0.21도=화씨 ',t1.celsius_to_fahrenheit(-0.21),'(셋째 자리 반올림)') ③
32  print('₩n섭씨 -0.21도=화씨 ',t2.celsius_to_fahrenheit(-0.21),'(첫째 자리 반올림)') ④
```

```
화씨       섭씨 ①
0.0 --> -17.778
0.5 --> -17.5
1.0 --> -17.222
... ...
```

화씨 섭씨 ②

0.0 --> -17.8

0.5 --> -17.5

1.0 --> -17.2

… …

섭씨 -0.21도=화씨 31.622 (셋째 자리 반올림) ③

섭씨 -0.21도=화씨 31.6 (첫째 자리 반올림) ④

A.3 기본 자료구조: 리스트와 튜플, 딕셔너리, 셋

무지개 색은 7개 요소로 구성된 리스트 ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']으로 표현할 수 있다. 스마트폰에 저장된 전화번호 목록은 [['홍길동', '010 1111 2222'],['김철수', '010 3333 4444'],…]와 같은데 각 요소가 이름과 전화번호를 가지기 때문에 리스트의 리스트다. 사람이 다루는 데이터는 리스트 형태로 표현하는 경우가 아주 많다.

소개

파이썬은 여러 개의 요소를 가진 데이터를 효율적으로 저장하고 처리하기 위한 자료구조로 리스트$_{list}$와 튜플$_{tuple}$, 딕셔너리$_{dictionary}$, 셋$_{set}$을 제공한다. 리스트는 프로그램을 실행하는 도중에 값을 변경할 수 있는데, 리스트와 달리 튜플은 생성하고 나면 값을 변경할 수 없다. 따라서 튜플은 한번 만들면 수정이 필요 없는 상황에 주로 사용한다. 수정 가능 여부를 빼고는 리스트와 튜플은 똑같다. 딕셔너리는 요소를 키$_{key}$와 값$_{value}$의 쌍으로 표현하는데, 키를 통해 값을 검색하는 응용에 사용한다. 셋은 집합을 표현하기 때문에 중복된 요소를 허용하지 않는다. 이들은 파이썬이 기본으로 제공하므로 모듈을 import하지 않고 바로 사용할 수 있다. 이들 자료구조는 파이썬 프로그래밍의 토대를 형성하기 때문에 익숙하게 쓸 수 있어야 한다.

Tip 11: 리스트와 튜플, 딕셔너리, 셋에 대한 공식적인 설명 문서는 https://docs.python.org/3/library/stdtypes.html에서 찾아볼 수 있다.

생성과 인덱싱, 슬라이싱

리스트는 [ ], 튜플은 ( ), 딕셔너리는 { } 괄호를 사용하여 구별한다. 집합은 set 함수로 만든다. 아래 코드에서 [1]~[4]행은 각각 리스트, 튜플, 딕셔너리, 집합을 생성한다. [5]행에서 type 함수로 데이터형을 알아본 결과 순서대로 list 클래스, tuple 클래스, dict 클래스, set 클래스라는 사실을 확인할 수 있다. [6]행에서 객체 a, b, c, d를 출력한 결과, 집합을 표현하는 d에서는 두 번 나타난 2를 하나 제거하였음을 알 수 있다. print 문에서 sep='₩n'은 객체마다 줄을 바꿔 출력하라는 뜻이다. [7]행에서 튜플의 요소를 변경하려 시도하는데 TypeError가 발생하여 변경할 수 없음을 알 수 있다. [8]행은 len 함수로 길이, 즉 요소의 개수를 알아낼 수 있다는 사실을 보여준다. [9]행은 range의 결과에 list 함수를 적용하여 리스트를 만들 수 있음을 보여준다.

```
In [1]: a=[5,2,3,8,2]
In [2]: b=(5,2,3,8,2)
In [3]: c={1: 'book', 5: 'notebook', 3: 'pencil', -3: 'eraser', 2: 120, 12: 50}
In [4]: d=set([5,2,3,8,2])
In [5]: print(type(a),type(b),type(c),type(d))
  <class 'list'> <class 'tuple'> <class 'dict'> <class 'set'>
In [6]: print(a,b,c,d,sep='₩n')
  [5, 2, 3, 8, 2]
  (5, 2, 3, 8, 2)
  {1: 'book', 5: 'notebook', 3: 'pencil', -3: 'eraser', 2: 120, 12: 50}
  {8, 2, 3, 5}
In [7]: b[2]=-3
```

```
  TypeError: 'tuple' object does not support item assignment
In [8]: print(len(a),len(b),len(c),len(d))
  5 5 6 4
In [9]: e=list(range(3,8))
In [10]: print(e)
  [3, 4, 5, 6, 7]
```

리스트의 특정 요소에 접근하여 값을 읽거나 변경할 필요가 종종 있다. 이때 요소의 위치를 지정하는 것을 인덱싱$_{indexing}$이라 한다. 파이썬에서는 [그림 A-1]이 보여주는 바와 같이 리스트의 인덱스가 0, 1, 2, …처럼 0부터 len(a)-1까지이다.

	0	1	2	3	4	← 인덱스
a=	5	2	3	8	2	← 요소의 값

[그림 A-1] 리스트의 인덱싱

아래 코드는 리스트를 인덱싱하는 사례를 보여준다. [12]행에서 인덱스 2와 3에 있는 요소를 출력한 결과 예상대로 3과 8이 출력되었다. 인덱스 -1과 -3은 뒤에서 첫번째와 뒤에서 세번째를 가리킨다. 따라서 2 와 3이 출력된다. [13]행에서 인덱스 6은 유효한 인덱스 범위인 0~4를 벗어났기 때문에 IndexError가 발생하였다. [14]행에서는 인덱스 1인 요소에 -21을 대입한다. 출력 결과에서 바뀐 내용을 확인할 수 있다. 튜플의 인덱싱도 리스트와 같은 방식으로 수행한다.

```
In [11]: a=[5, 2, 3, 8, 2]
In [12]: print(a[2],a[3],a[-1],a[-3])
  3 8 2 3
In [13]: print(c[6])
  IndexError: list index out of range
In [14]: a[1]=-21
In [15]: print(a)
  [5, -21, 3, 8, 2]
```

리스트와 튜플의 인덱싱에서는 요소의 위치를 0,1,2,…의 숫자로 지정하는 반면, 딕셔너리의 인덱싱은 키를 통해 이루어진다. 다음 예시는 딕셔너리 c를 인덱싱한다. [17]행에서 -3과 5를 키로 사용해서 인덱싱한 결과 해당하는 값 eraser와 120이 출력되었다. [18]행에서 10을 키로 사용해서 c[10]과 같이 인덱싱했는데 10이라는 키가 없기 때문에 KeyError가 발생한다.

```
In [16]: c={1: 'book', 5: 'notebook', 3: 'pencil', -3: 'eraser', 2: 120, 12: 50}
In [17]: print(c[-3],c[2])
  eraser 120
In [18]: print(c[10])
  KeyError: 10
```

리스트와 튜플, 딕셔너리, 셋은 데이터형이 다른 요소를 허용한다. 다음 코드를 살펴보자. 리스트 aa는 5개 요소를 가지는데 앞의 3개는 정수고 네번째 요소는 길이가 4인 리스트이며 다섯 번째

요소는 문자열이다. 인덱스 2, 3, 4에 있는 요소를 출력하면 해당 요소 값이 제대로 출력된다. a[3][2]는 a의 3번 요소인 [‘pencil’,4,-2,2]의 2번 요소를 가리키므로 -2가 출력된다. a[4][3]은 a의 4번 요소인 'book'의 3번 요소를 가리키므로 문자 'k'가 출력된다. cc에서처럼 딕셔너리도 서로 다른 데이터형의 키와 값을 허용한다.

```
In [19]: aa=[5,2,3,[ 'pencil',4,-2,2], 'book']
In [20]: print(aa[2],aa[3],aa[4])
  3 ['pencil', 4, -2, 2] book
In [21]: print(aa[3][2],aa[4][3])
  -2 k
In [22]: cc={1: 'book', ‘com’: 'notebook', 30.1: 'pencil', -3: 'eraser', 2: 120, 12: 50}
In [23]: print(cc[‘com’], cc[30.1], cc[12])
  notebook pencil 50
```

슬라이싱slicing은 리스트의 일부분을 잘라내는 연산이다. 다음 코드는 몇가지 슬라이싱 예시를 보여준다. p:q는 p, p+1, …, q-1을 뜻한다. q를 포함하지 않는다는 사실을 눈 여겨 봐야 한다. p가 생략된 :q는 0:q를 뜻하고 q가 생략된 p:는 p부터 마지막 요소까지 포함한다.

```
In [24]: aa=[5, 2, 3, ['pencil', 4, -2, 2], 'book']
In [25]: print(aa[1:4],aa[:-1],aa[2:],sep='₩n')
  [2, 3, ['pencil', 4, -2, 2]]
  [5, 2, 3, ['pencil', 4, -2, 2]]
  [3, ['pencil', 4, -2, 2], 'book']
In [26]: print(aa[3][1:],aa[4][:3])
  [4, -2, 2] boo
```

대입 연산

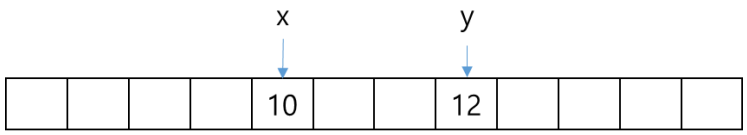
아래 코드는 값을 하나만 저장한 스칼라 변수 x를 y에 대입한 뒤 y를 변경한다. x와 y를 출력해보면 x는 원래 값 10을 가지고 있고 y는 변경된 값 12를 갖는다. [그림 A-2(a)]가 설명하는 바와 같이 x와 y는 서로 다른 메모리 주소에 저장되어 있기 때문이다.

```
In [27]: x=10
In [28]: y=x
In [29]: y=12
In [30]: print(x, y)
  10 12
```

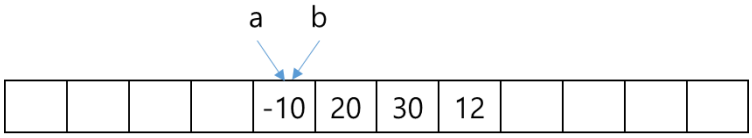
비슷한 과정을 리스트에 적용해 보자. 리스트 a를 만든 다음 a를 b에 대입한다. b의 요소 b[0]을 변경한 후 a와 b를 출력한다. 놀랍게도 b의 첫번째 요소를 바꾸었는데 a도 따라 바뀌었다. 이러한 사실과 그에 대한 이유를 꼭 이해해야 한다. 파이썬에서는 x, y와 같은 스칼라 변수를 다른 메모리 주소에 저장한다. y=x를 실행하면 x에 해당하는 메모리 주소에서 y에 해당하는 메모리 주소로 값을 복사한다. x와 y는 이름도 다르고 메모리 주소도 다르다. 이와 달리 리스트에서는 같은 메모리 주소를 공유한다. [그림 A-2(b)]는 이런 원리를 설명한다. b=a로 리스트 a를 리스트 b에 대입하면 b를 위해 새로운 메모리

주소를 확보하고 내용을 복사하는 것이 아니라, 단지 a가 차지하고 있는 메모리 주소를 b가 가리키게 하여 같은 메모리 주소를 공유한다.

```
In [31]: a=[10,20,30,40]
In [32]: b=a
In [33]: b[0]=-10
In [34]: print(a,b)
   [-10, 20, 30, 40] [-10, 20, 30, 40]
```



(a) 스칼라 변수에서는 다른 메모리 주소를 사용



(b) 리스트에서는 같은 메모리 주소를 공유

[그림 A-2] 대입 연산에 따른 효과

다른 메모리 주소에 저장하려면 copy 함수를 사용하면 된다. 아래 코드는 copy 함수를 사용하면 a와 c가 다른 메모리 주소를 차지하게 된다는 사실을 확인해준다.

```
In [35]: a=[10,20,30,40]
In [36]: c=a.copy()
In [37]: c[0]=-10
In [38]: print(a,c)
   [10, 20, 30, 40] [-10, 20, 30, 40]
```

연산: + 와 *, 추가와 삭제

리스트에서 +는 덧셈이 아니고 두 리스트를 결합하는 연산자다. *는 곱셈이 아니라 지정된 수만큼 반복하는 연산자다. 다음 코드를 살펴보자. + 와 * 연산자는 리스트와 튜플에만 적용할 수 있고 딕셔너리와 집합에는 적용할 수 없다.

```
In [39]: x=[1,2,'pen']
In [40]: y=[-2,1]
In [41]: print(3*x)
   [1, 2, 'pen', 1, 2, 'pen', 1, 2, 'pen']
In [42]: print(y+ x)
   [-2, 1, 1, 2, 'pen']
In [43]: print(x+ 2*y)
   [1, 2, 'pen', -2, 1, -2, 1]
```

리스트에 요소를 추가하거나 삭제하려면 함수를 사용한다. append 함수는 리스트 뒤에 요소를 추가하고, insert 함수는 지정한 위치에 요소를 추가한다. remove는 지정된 값을 가진 요소를 삭제하며 pop은 지정된 위치의 요소를 삭제한다. 아래 예시가 보여주듯이 remove는 삭제 대상 요소가 여럿일 때 맨 앞에 있는 한 개를 삭제한다. 추가와 삭제 연산 이외에도 정렬하는 sort, 뒤집는 reverse 등 유용한 함수를 많이 제공한다. 아래 예시 코드는 리스트 z에 append, insert, remove, pop 함수를 적용한 사례를 보여준다.

Tip 12: 리스트의 다양한 연산은 https://docs.python.org/3/tutorial/datastructures.html을 참조한다.

```
In [44]: z=[1,2,3,4]
In [45]: z.append(-7)
In [46]: print(z)
   [1, 2, 3, 4, -7]
In [47]: z.insert(1,-7)
In [48]: print(z)
   [1, -7, 2, 3, 4, -7]
In [49]: z.remove(-7)
In [50]: print(z)
   [1, 2, 3, 4, -7]
In [51]: v=z.pop(2)
In [52]: print(v,z)
  3 [1, 2, 4, -7]
```

리스트 내포

리스트 내포list comprehension를 이용하면 공식이나 조건을 만족하는 리스트를 만들 수 있다. 다음 코드는 i가 0,1,2,…,9로 변화할 때 i 제곱을 담은 리스트를 만들어준다.

```
In [53]: x2=[i*i for i in range(10)]
In [54]: print(x2)
    [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

위 코드는 아래 세 줄짜리 코드를 한 줄로 간결하게 쓴 셈이다. 이처럼 리스트 내포는 공식을 반복적으로 적용하여 리스트를 만드는 간결한 방법이다.

```
In [55]: x2=[ ]
In [56]: for i in range(10):
    ...:     x2.append(i*i)
```

다음 코드는 중첩된 리스트 내포를 예시하는데, 리스트 내포를 이용하여 리스트의 리스트를 만들 수 있다는 사실을 보여준다.

```
In [57]: x3=[[i,j*2] for i in [10,2,3,1] for j in [2,4]]
In [58]: print(x3)
    [[10, 4], [10, 8], [2, 4], [2, 8], [3, 4], [3, 8], [1, 4], [1, 8]]
```

다음 코드는 조건문을 적용하여 조건이 맞는 경우에만 값을 생성하게 할 수 있다는 사실을 보여준다.

```
In [59]: x4=[[i,j*2] for i in [10,2,3,1] for j in [2,4] if i!=j]
In [60]: print(x4)
    [[10, 4], [10, 8], [2, 8], [3, 4], [3, 8], [1, 4], [1, 8]]
```

리스트를 이용한 프로그래밍: 온도 변환 표 작성

A.2 절의 [프로그램 A-15]에서 온도 변환 표를 출력하는 fahrenheit_to_celsius_table 멤버 함수는 계산 결과를 함수 내에서 print 문을 이용하여 직접 출력한다. 이런 방식의 코딩은 품질이 낮다. 계산 결과를 리스트에 담아 반환하는 방식으로 수정하면 프로그램의 품질이 크게 향상된다. 왜냐하면 반환된 리스트를 받아 다양한 일을 할 수 있기 때문이다. 멋있게 장식하여 출력할 수도 있고, 리스트 슬라이싱을 이용하여 관심있는 특정 구간만 잘라낼 수도 있고, 특정 값을 검색할 수도 있고, 다른 데이터와 결합할 수도 있다.

[프로그램 A-16]은 fahrenheit_to_celsius_table 과 celsius_to_fahrenheit_table 멤버 함수가 계산 결과를 담은 리스트를 반환하도록 [프로그램 A-15]를 수정한 프로그램이다.

11 행은 표 헤딩을 가진 리스트 res 를 만든다. 14 행은 계산 결과를 리스트에 추가한다. 이때 숫자를 문자열로 바꿔주는 str 함수와 문자열을 접합하는 + 연산자로 하나의 문자열을 만든다. for 문이 끝나면 결과를 담은 res 를 return 문으로 반환한다. 메인에 해당하는 27~30 행을 살펴보자. 27 행은 TemperatureConversion 클래스 형의 객체 t 를 만든다. 28 행은 fahrenheit_to_celsius_table 멤버 함수가 반환한 결과를 ttable 에 저장한다.

[1]행은 ttable 의 내용을 조사한다. [2]행은 len(ttable)은 요소가 21 개라고 알려준다. [3]행은 ttable[:6]은 슬라이싱을 사용하여 앞에 있는 6 개 요소만 잘라낸다.

```
In [1]: print(ttable)

    ['화씨      섭씨', '0.0-->-17.778', '0.5-->-17.5', '1.0-->-17.222', '1.5-->-16.944', '2.0-->-16.667', '2.5-->-16.389', '3.0-->-16.111', '3.5-->-15.833', '4.0-->-15.556', '4.5-->-15.278', '5.0-->-15.0', '5.5-->-14.722', '6.0-->-14.444', '6.5-->-14.167', '7.0-->-13.889', '7.5-->-13.611', '8.0-->-13.333', '8.5-->-13.056', '9.0-->-12.778', '9.5-->-12.5']

In [2]: len(ttable)

    21

In [3]: ttable[:6]

    ['화씨      섭씨', '0.0-->-17.778', '0.5-->-17.5', '1.0-->-17.222', '1.5-->-16.944', '2.0-->-16.667']
```

29~30 행은 ttable 의 요소를 한 줄에 하나씩 출력한다. 29 행의 for 명령어는 ttable 리스트에 있는 요소를 차례로 element 에 대입한 다음 명령어 블록을 구성하는 30 행을 반복한다. 29~30 행은 아래와 같이 한 줄로 바꿔 쓸 수 있다.

print(*ttable,sep='₩n')

[프로그램 A-16] 리스트를 반환하는 멤버 함수

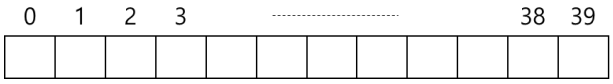
```
01  import numpy as np
02
03  class TemperatureConversion:
04      def __init__(self,position):
05          self.pos=position # 소수점 pos자리에서 반올림
06
07      def fahrenheit_to_celsius(self,val): # 화씨->섭씨 변환
08          return round(((val-32.0)*5.0)/9.0,self.pos)
09
10      def fahrenheit_to_celsius_table(self,start=0.0,end=100.0,inc=0.5):
11          res=['화씨        섭씨']
12          for fahrenheit in np.arange(start,end,inc):
13              celsius=self.fahrenheit_to_celsius(fahrenheit)
14              res.append(str(fahrenheit)+'-->'+str(celsius))
15          return res
16
17      def celsius_to_fahrenheit(self,val): # 섭씨->화씨 변환
18          return round((val*9.0)/5.0+32.0,self.pos)
19
20      def celsius_to_fahrenheit_table(self,start=0.0,end=100.0,inc=0.5):
21          res=['섭씨        화씨']
22          for celsius in np.arange(start,end,inc):
23              fahrenheit=self.celsius_to_fahrenheit(celsius)
24              res.append(str(fahrenheit)+'-->'+str(celsius))
25          return res
26
27  t=TemperatureConversion(3)
28  ttable=t.fahrenheit_to_celsius_table(0,10,0.5)
29  for element in ttable:
```

30 print(element)
화씨 섭씨 0.0-->-17.778 0.5-->-17.5 1.0-->-17.222 … …

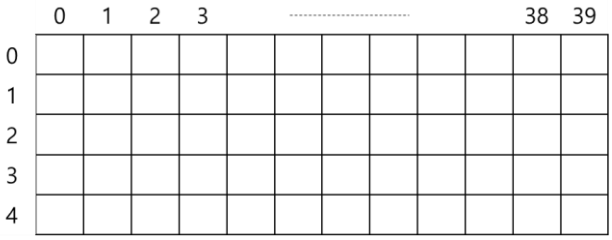
A.4 numpy 라이브러리

40명으로 구성된 분반의 중간고사 점수를 저장하려면 [그림 A-3(a)]와 같이 크기가 40인 1차원 배열이 필요하다. 중간, 기말, 퀴즈 등 5개 점수가 있다면 [그림 A-3(b)]와 같이 행이 5개이고 열이 40개인 2차원 배열을 써야 한다. 그런 분반이 3개라면 [그림 A-3(c)]에 있는 3차원 배열이 필요하다. 컴퓨터 비전에서는 다차원 배열을 텐서$_{tensor}$라 부른다. numpy 라이브러리는 ndarray라는 데이터형을 가지고 다차원 배열, 즉 텐서를 구현한다.

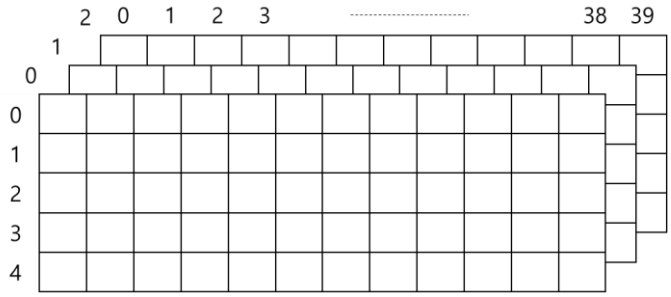
[TIP] numpy는 numerical python의 약어로서 넘파이라고 읽는다.



(a) 1차원 배열



(b) 2차원 배열



(c) 3차원 배열

[그림 A-3] 다차원 배열

데이터를 다루는 일은 목공 작업과 유사하다. 원시 데이터는 잡음도 있고, 손실 값도 있고, 아웃라이어도 많다. 이런 원시 데이터를 가공하여 핵심이 되는 패턴을 추출하고 패턴을 조합하고 변환하여 원하는 정보를 추출한다. [그림 A-4]가 보여주는 바와 같이 목공에서는 벌레 먹고, 파이고, 갈라진 원목을 자르고, 붙이고, 대패질하여 판재를 만들고, 판재를 조립하여 멋진 피크닉 테이블을 만든다. 목공을 잘 하려면 연장을 능숙하게 쓸 수 있어야 한다. 마찬가지로 데이터 분석을 잘 하려면 numpy라는 도구를 능숙하게 쓸 수 있어야 하다. 이 절에서는 numpy 라이브러리가 제공하는 멤버 함수를 사용하는 연습을 한다.

numpy를 보다 상세하게 공부하려면 공식 사이트인 https://numpy.org/에 접속하여

[Documentation]에서 제공하는 『Numpy User Guide』를 3장까지 공부하면 된다. 도구에 능숙해지는 유일한 길은 반복 학습이다. 이 절 또는 공식 문서에서 제공하는 예시를 변형하여 반복 학습을 충실히 하기 바란다.

[TIP]『Numpy User Guide』는 https://numpy.org/doc/stable/numpy-user.pdf에서 다운로드할 수 있다.

[그림 A-4] 데이터를 다루는 일과 비슷한 목공 작업

ndarray란

다차원 배열을 표현하는 ndarray 데이터형은 몇 가지 특성이 있다.

- 처음 생성될 때 메모리 크기가 정해진다. 배열이 커지면 원래 메모리를 반환하고 새로 생성한 큰 메모리를 사용한다.
- 모든 요소가 같은 데이터형을 가지며 같은 수의 바이트를 점유한다. 따라서 *i*번째 요소의 메모리 주소를 address(*i*)=*base_address*+*i***d*를 이용하여 아주 빨리 계산할 수 있다. *base_address*는 배열의 시작 주소이고 *d*은 요소 하나가 차지하는 바이트 수다. ndarray로 표현한 데이터를 처리하는 시간이 빠른 이유이다.
- 벡터와 행렬 연산을 포함하여 유용한 멤버 함수를 아주 많이 제공한다. 이들 함수는 계산 시간 측면에서 최적화되어 있어 컴퓨터 비전에 활용하기에 적합하다. 이런 이유 때문에 많은 라이브러리가 자신의 데이터를 ndarray로 변환하여 처리한다. 이 책에서 많이 사용한 tensorflow 라이브러리도 ndarray를 많이 사용하며, 자신에게 고유한 tensor 데이터형은 ndarray와 호환된다. OpenCV 라이브러리는 영상을 ndarray로 표현한다.

아래 예시 코드를 실행해보자. [1]행은 numpy 라이브러리를 불러온다. 지금부터 이 모듈이 임포트 된 상태라고 가정하고 실습을 진행한다.

[2]행은 5개 요소를 가진 ndarray를 만들어 객체 a에 저장한다. print문으로 출력한 결과를 보면 지정한 대로 값이 저장되었음을 확인할 수 있다. [3]행은 객체 a의 멤버 함수 sort를 호출한다. sort 함수는 배열에 있는 값을 오름차순으로 정렬하고 결과를 배열에 다시 저장한다. print문으로 출력한 결과를 보면 제대로 정렬되었음을 확인할 수 있다.

```
In [1]: import numpy as np
In [2]: a=np.array([12,8,20,17,15])
In [3]: print(a)
```

```
    [12 8 20 17 15]
In [4]: a.sort()
In [5]: print(a)
[ 8 12 15 17 20]
```

dir과 type, shape 명령어

파이썬 프로그래밍을 하면서 자주 쓰는 명령어로 dir과 type, shape이 있다. 앞의 예시에서 객체 a의 멤버 함수 sort를 사용했는데, 또다른 함수로 무엇이 있는지 궁금할 때가 있다. 이때 dir을 사용한다. 아래 코드를 실행하면 객체 a가 가진 멤버 함수의 목록을 알 수 있다.

```
In [6]: dir(a)
['T',
 '__abs__',
 '__add__',
 '__and__',
…
 'shape',
 'size',
 'sort',
…]
```

컴퓨터 비전 프로그래밍을 하다 보면 아주 많은 객체를 다루게 된다. 이런 상황에서는 객체가 어떤 데이터형을 가지는지 확인할 필요가 종종 생기는데 이때 쓰는 명령어가 type이다. 또한 객체의 모양을 확인할 필요가 발생하는데 이때 쓰는 명령어가 shape이다.

다음 코드에서 [1]행의 type 명령어는 객체 a가 ndarray형임을 알려주고 [2]행의 shape 명령어는 a가 5개 요소로 구성된 1차원 구조임을 알려준다.

```
In [7]: type(a)
numpy.ndarray
In [8]: a.shape
(5,)
```

ndarray 배열 만들기

새 배열을 만드는 방법은 여러 가지가 있다. 앞의 코드에서 [2]행은 array 함수를 사용하여 1차원 구조의 배열을 만들었다. 다음 코드에서 [9]행도 array 함수로 배열을 만드는데 앞의 [2]행과 다른 점이 두 가지 있다. 첫째는 2차원 구조의 배열을 만든다는 점이고 둘째는 요소 중에 4.0이라는 실수가 들어있다는 점이다. 배열은 생성될 때 요소를

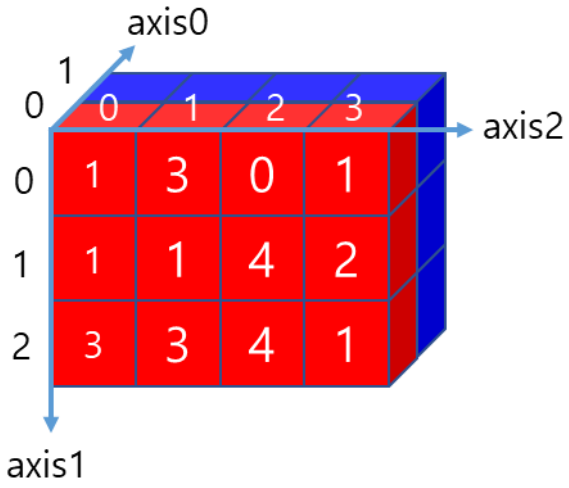
보고 데이터형을 결정하는데, b는 요소 중에 실수가 있기 때문에 64비트 실수인 float64형이 된다. ndarray 객체는 배열이 몇 차원의 구조인지 알려주는 ndim, 요소의 데이터형을 알려주는 dtype, 모양을 알려주는 shape 등의 멤버 변수를 가진다. [11], [12], [13]행은 이들 멤버 변수를 출력하여 객체 b의 정보를 알아낸다.

```
In [9]: b=np.array([[12,3,4.0],[1,4,5]])
In [10]: print(b)
[[12.   3.   4.]
 [ 1.   4.   5.]]
In [11]: b.ndim
2
In [12]: b.shape
(2,3)
In [13]: b.dtype
dtype('float64')
```

[14]행은 3차원 구조의 배열 c을 생성한다. numpy에서는 n차원 구조에서 각각의 차원을 축$_{axis}$이라 부른다. c는 3개의 축을 가지는데 axis0은 두 요소를 가진다. 두 요소를 구분하기 위해 빨간색과 파란색으로 표시하였다.

axis1은 3개의 요소, axis2는 4개의 요소를 가진다. 요소의 개수를 길이라고 부르기도 하는데 axis0, axis1, axis2의 길이는 각각 2, 3, 4이다. [그림 A-5]는 이런 구조를 설명한다. 뒤에서 축을 이용하여 배열을 다른 형태로 변환하는 방법을 공부한다. [그림 A-5]를 보고 c의 구조를 잘 이해해 두기 바란다.

```
In [14]: c=np.array([[[1,3,0,1],[1,1,4,2],[3,3,4,1]], [[2,1,2,1] ,[1,0,1,0],[1,5,6,2]]])
In [15]: print(c)
  [[[1 3 0 1]
    [1 1 4 2]
    [3 3 4 1]]
   [[2 1 2 1]
    [1 0 1 0]
    [1 5 6 2]]]
In [16]: c.shape
  (2, 3, 4)
```



[그림 A-5] 세 개의 축을 가진 3차원 배열([13]행이 생성하는 배열 c)

zeros 함수는 0으로 채운 배열을 만들고 ones 함수는 1로 채운 배열을 만든다. [17]행은 두 개의 축을 가진 2차원 배열 d를 생성한다. axis0은 길이가 2, axis1은 길이가 3이다. d.dtype으로 확인한 결과는 float64이다. numpy는 기본으로 64비트 실수형을 사용하기 때문이다. 마지막 명령어가 보여주듯이 random 함수를 사용하여 난수로 구성된 배열을 만들 수 있다. print문의 결과에서 볼 수 있듯이 random 함수는 [0,1] 범위에서 실수 난수를 생성해준다.

```
In [17]: d=np.zeros([2,3])
In [18]: print(d)
[[0. 0. 0.]
 [0. 0. 0.]]
In [19]: d.dtype
dtype('float64')
In [20]: e=np.random.random([2,5])
In [21]: print(e)
[[0.86997882 0.32528632 0.13021243 0.48668632 0.24810257]
 [0.13114461 0.28725987 0.49369103 0.01422569 0.2509103 ]]
```

배열을 만드는 또 다른 방법은 일정한 간격의 수를 만들어주는 arange 함수를 쓰는 것이다. 다음 코드의 [22]행은 1에서 시작하여 2.5씩 증가시키면 20미만까지 등간격 수를 만들어 배열에 담아준다.

```
In [22]: f=np.arange(1,20,2.5)
In[23]: print(f)
[ 1.    3.5   6.    8.5 11.   13.5 16.   18.5]
```

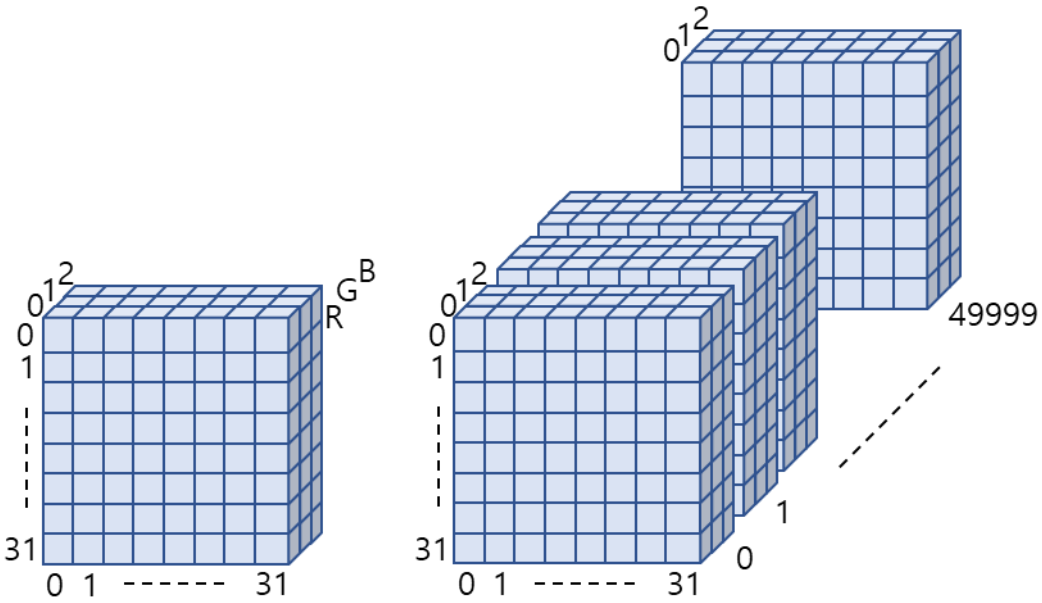
공개 데이터로 ndarray 만들기

컴퓨터 비전에서는 공개된 데이터를 읽어 ndarray 배열을 생성하는 경우가 많다. 이 책이 제시한 프로그램 대부분은 이렇게 배열을 생성한다. 다음 코드는 자연 영상으로 구성된 CIFAR-10 데이터를 읽어 cifar10_data라는 ndarray 객체를 생성하고 특성을 확인하는 것이다. 한 장의 영상은 32*32 크기이고 RGB의 3개 채널로 표현된다.

```
In [24]: import tensorflow.keras.datasets as ds
In [25]: (x_train, y_train), (x_test, y_test) = ds.cifar10.load_data()
In [26]: print(x_train)
[[[[ 59   62   63]
    [ 43   46   45]
    [ 50   48   43]
    ...
    [158 132 108]
    [152 125 102]
    [148 124 103]]

   [[ 16   20   20]
    [  0    0    0]
    [ 18    8    0]
    ...
In[27]: x_train.shape
(50000, 32, 32, 3)
In [28]: x_train.dtype
dtype('uint8')
In [29]: x_train.nbytes
153600000
```

[24]행은 tensorflow 라이브러리가 설치되어 있어야 실행이 가능하다. tensorflow 설치에 대해서는 책 2.2절을 참조한다. [25]행은 tensorflow가 기본으로 제공하는 CIFAR-10 데이터를 읽어 훈련 집합과 테스트 집합으로 나누어 저장한다. 여기서는 훈련 집합인 x_train을 살펴본다. x_train을 print문으로 출력하면 [[[[…로 표시된 것으로 보아 [그림 A-6]과 같은 4차원 구조의 배열임을 알 수 있다. x_train.shape으로 확인해보니 50000*32*32*3 구조의 배열이라는 사실을 확인할 수 있다. 32*32 해상도의 RGB 영상이 50,000장 담겨 있기 때문이다. x_train.dtype으로 요소의 데이터형을 확인해보니 부호 없는 1바이트 정수형을 뜻하는 uint8이다. 데이터의 메모리 용량을 알아보려면 nbytes 속성을 출력하면 된다. x_train의 경우 15,360,000바이트임을 확인했다. 대략 15메가 바이트 정도이다. nbytes는 요소의 수를 알려주는 size와 개별 요소가 차지하는 바이트 수를 알려주는 itemsize를 곱한 것과 같다. 따라서 x_train.nbytes는 x_train.size*x_train.itemsize 또는 np.prod(x_train.shape)*x_train.itemsize와 같다.



(a) 컬러 영상 한 장(3차원 구조 텐서) (b) 50,000장의 컬러 영상(4차원 구조 텐서)

[그림 A-6] 4차원 구조 배열로 표현되는 CIFAR-10 데이터셋

대입 연산

[그림 A-2(b)]는 리스트 변수 a와 b가 있을 때 b=a 명령어로 a를 b에 대입하면 a와 b가 메모리 주소를 공유한다는 사실을 설명한다. ndarray로 만든 배열의 대입에서도 메모리 주소를 공유한다.

아래 코드는 배열 a를 만든 다음, a를 b에 대입한다. b의 첫번째 요소 b[0]를 -10으로 변경한 후 a와 b를 출력해보면, b의 첫번째 요소를 바꾸었는데 a도 따라서 바뀐다는 사실을 확인할 수 있다. a를 b에 대입하면 b를 위해 새로운 메모리 주소를 확보하고 내용을 복사하는 것이 아니라, 단지 a가 차지하고 있는 메모리 주소를 b가 가리키게 하여 메모리 주소를 공유한다.

```
In [30]: a=np.array([10,20,30,40])
In [31]: b=a
In [32]: b[0]=-10
In [33]: print(a,b)
[-10   20   30   40] [-10   20   30   40]
```

리스트에서와 마찬가지로, 다른 메모리 공간에 실제 복사를 하려면 [34]행과 같이 copy 함수를 사용해야 된다. 아래 코드는 copy 함수를 사용하면 a와 c가 다른 메모리 공간을 차지하게 됨을 확인해준다.

```
In [34]: c=a.copy()
```

```
In [35]: c[1]=-20
In [36]: print(a,c)
[-10   20   30   40] [-10 -20   30   40]
```

모양 바꾸기

컴퓨터 비전에서는 배열을 다른 모양으로 바꾸어야 하는 경우가 많다. 예를 들어 2차원 구조의 배열로 표현되는 영상을 다층 퍼셉트론에 입력하려면 1차원 구조로 바꾸어야 한다. 반대로 1차원 구조를 2차원으로 변환해야 할 경우도 있다. 이때 쓰는 함수가 reshape이다. [38]행은 1차원 배열에 reshape(2,3) 함수를 적용하여 2*3 모양으로 바꾼다. 원래 배열 a와 새로운 배열 b를 출력한 결과, a는 그대로 1차원을 유지하고 b는 2차원이 되었음을 확인할 수 있다. 유념할 점은 배열 a는 원래 모양을 그대로 유지한다는 사실이다.

```
In [37]: a=np.array([1,2,3,4,5,6])
In [38]: b=a.reshape([2,3])
In [39]: print(a)
[1 2 3 4 5 6]
In [40]: print(b)
[[1 2 3]
 [4 5 6]]
```

또 하나 유념할 점이 있다. 배열 a와 b가 모양이 달라 서로 다른 메모리 공간을 차지할 것이라 생각할 수 있는데 그렇지 않다. [그림 A-2(b)]의 상황을 그대로 유지한다. 단지 a와 b는 모양만 달라진다. numpy에서는 a와 b의 뷰$_{view}$는 다르고 내용은 같다라고 말한다. a와 b는 뷰 정보는 각자 갖지만 내용은 공유한다. 다음 코드는 a와 b가 메모리 공간을 여전히 공유한다는 사실을 확인한다.

```
In [41]: a[4]=-5
In [42]: print(b)
[[ 1   2   3]
 [ 4 -5   6]]
```

때때로 데이터의 축의 위치를 교환할 필요가 있다. 축의 위치를 교환하는 T, swapaxes, transpose를 실험해보자. [44]행은 2*3 배열 a에 T를 적용하여 3*2 배열 b에 저장한다. 즉 axis0과 axis1을 서로 교환한 셈이다. 축에 대해서는 [그림 A-5]를 참조한다. T는 전치 행렬을 구해주는데 그 이상의 기능을 바로 이어 확인해 본다.

```
In [43]: a=np.array([[1,2,3],[4,5,6]])
In [44]: b=a.T
In [45]: print(b)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

이제 좀더 복잡한 CIFAR-10 데이터에 대해 축을 교환하는 연산을 적용해 보자. x_train은 [그림 A-6]의 50000*32*32*3 구조이다. x_train.T를 적용하면 축의 순서가 뒤집어져 3*32*32*50000이 된다. 즉 axis0, axis1, axis2, axis3이 axis3, axis2, axis1, axis0이 된다. [49]행에서는 이전과 다르게 x_train.T.shape으로 간결하게 썼는데, 헷갈리면 이전처럼 x1=x_train.T와 x1.shape의 두 줄로 실행하면 된다. swapaxes 함수는 교환하려는 두 축을 매개변수로 주면 된다. [50]행은 axis0과 axis3을 교환한다. transpose 함수는 모든 축에 대해 새로운 위치를 지정하면 그에 따라 축의 순서가 바뀐다. [51]행의 x_train.transpose(1,2,3,0)은 원래 0,1,2,3이었던 축을 1,2,3,0으로 바꾸라는 지시이다. 다시 한번 유념할 점은 원래 배열인 x_train은 원래대로 유지되며, 변환된 배열은 뷰만 다르고 내용은 원래 배열과 같다는 사실이다.

```
In [46]: import tensorflow.keras.datasets as ds
In [47]: (x_train, y_train), (x_test, y_test) = ds.cifar10.load_data()
In [48]: x_train.shape
(50000, 32, 32, 3)
In [49]: x_train.T.shape
(3, 32, 32, 50000)
In [50]: x_train.swapaxes(2,3).shape
(50000, 32, 3, 32)
In [51]: x_train.transpose(1,2,3,0).shape
(32, 32, 3, 50000)
```

인덱싱과 슬라이싱

배열의 인덱싱은 A.3절에서 설명한 리스트의 인덱싱과 비슷하다. 아래 코드는 3*5 크기의 a를 가지고 인덱싱을 설명한다. a[1,2]는 axis0은 1, axis1은 2 위치의 요소를 가리킨다. 리스트와 마찬가지로 배열의 인덱스를 0, 1, 2, …처럼 0부터 시작한다. 따라서 a[1,2]는 print문의 결과에서 빨간색으로 표시한 요소를 가리킨다. 인덱스 -1은 축의 마지막 인덱스를 가리킨다. 축 0의 마지막 인덱스는 2이므로 a[-1,3]은 a[2,3]과 같아 주황색의 요소를 가리킨다. 인덱스 -2는 뒤에서 두번째 인덱스이므로 a[0,-2]는 a[0,3]과 같아 파란색으로 표시된 요소를 가리킨다.

```
In [52]: a=np.array([[3,2,-2,0,1],[2,-3,4,5,2],[1,1,-2,-3,2]])
In [53]: print(a)
[[ 3  2 -2  0  1]
 [ 2 -3  4  5  2]
 [ 1  1 -2 -3  2]]
```

```
In [54]: print(a[1,2],a[-1,3],a[0,-2])
4 -3 0
```

이제 배열의 일부를 잘라내는 슬라이싱slicing을 실험해본다. 리스트의 슬라이싱과 비슷하다. 바로 이전 코드의 배열 a를 그대로 사용한다. 아래 코드는 축을 생략하는 방법을 예시한다. [55]행의 a[2]는 axis1을 생략한다. 이 경우에 axis0의 2번 인덱스만 잘라내므로 [0 1 -2 -3 2]가 출력된다. [56]행의 a[-2]는 axis0에 대해 뒤에서 두번째를 가리키므로 a[1]과 같다. axis0을 생략하려면, [57]행처럼 axis0에 :을 지정하면 된다. a[:,3]은 axis0을 생략하고 axis1의 위치 3을 가리키므로 [0 5 -3]이 된다.

```
In [55]: print(a[2])
[ 1   1 -2 -3   2]
In [56]: print(a[-2])
[ 2 -3   4   5   2]
In [57]: print(a[:,3])
[ 0   5 -3]
```

특정 축에서 일부 범위를 지정하여 슬라이싱할 수 있다. 아래 코드의 [58]행은 a[1,1:3]으로 슬라이싱하는데, axis0은 인덱스 1만 취하고 axis1의 1:3은 인덱스 1과 2를 취한다. 리스트에서처럼 p:q는 p, p+1, …, q-1을 뜻한다. q는 빠진다는 사실을 눈 여겨 봐야 한다. [59]행은 a[:,1:]으로 슬라이싱한다. axis0이 지정한 :은 처음부터 끝까지 취하라는 뜻이다. axis1의 1:은 q가 생략되어 있는 셈인데, 1부터 끝까지 취하라는 뜻이다. 따라서 axis0은 전체를, axis1은 1:부터 마지막 인덱스까지 취하라는 뜻이어서 axis1에서 1,2,3,4를 잘라낸다. [60]행은 p가 생략된 :3 형태인데 0:3과 같다.

```
In [58]: print(a[1,1:3])
[-3   4]
In [59]: print(a[:,1:])
[[ 2 -2   0   1]
 [-3   4   5   2]
 [ 1 -2 -3   2]]
In [60]: print(a[:,:3])
[[ 3   2 -2]
 [ 2 -3   4]
 [ 0   1 -2]]
```

지금까지 인덱싱하고 슬라이싱하는 방법을 설명하였는데, 인덱싱하는 또다른 방법을 소개한다. 앞에서 a[1,3]과 같이 인덱싱하였는데 a[1][3]처럼 해도 동작한다. 다음 코드의 [61]행은 둘이 같은 곳을 가리킨다는 사실을 보여준다. 하지만 둘의 내부적인 동작은 크게 다르다. a[1,3]은 인덱스 값 1과 3을 가지고 메모리 주소를 계산하여 바로 접근하는 반면, a[1][3]은 a[1]을 실행하여 슬라이싱한 다음 그 결과에 [3]을 다시 적용한다.

다시 말해 a[*slice*1][*slice*2]는 a를 *slice*1로 슬라이싱한 다음 그 결과에 *slice*2를 적용한다. 아래 코드의 [62]행과 [63]행은 a[1,3]과 a[1][3]을 실행하는 시간을 측정한다. timeit a[1,3]은 a[1,3]을 처리하는데 걸리는 시간을 측정해주는데, 실행 결과를 보면 1천만 번 반복하는 일을 7번 반복한 다음 평균을 구하여 110나노초 가량 걸린다는 사실을 알려준다. [63]행의 결과를 보면 a[1][3]을 계산하는데 224나노초가 걸려 대략 두 배의 시간이 걸린다는 사실을 확인할 수 있다. [64]행은 두 가지 인덱싱 방식이 서로 다른 결과를 출력하는 보다 극적인 사례이다. a[:,2]는 axis0은 다 취하고 axis1은 인덱스 2만 취하여 결과가 [-2 4 -2]가 된다. 하지만 a[:][2]는 먼저 a[:]을 실행하여 axis0을 기준으로 다 취하니 a와 같게 되고 그 결과에 [2]를 적용하니 axis0을 기준으로 인덱스 2를 취하여 [1 1 -2 -3 2]가 된다. 앞으로 배열에서 한 요소를 지정할 때는 a[1,3]과 같은 표현을 쓰기 바란다.

```
In [61]: print(a[1,3],a[1][3])
5 5
In [62]: timeit a[1,3]
110 ns ± 1.12 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
In [63]: timeit a[1][3]
224 ns ± 1.15 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
In [64]: print(a[:,2],a[:][2])
[-2  4 -2] [ 1  1 -2 -3  2]
```

벡터화를 통한 사칙 연산과 관계 연산

파이썬은 벡터화vectorization되어 있다고 말한다. 벡터화의 장점은 표기의 간결성과 효율적인 계산에 있다. 먼저 표기에 대해 살펴보자. 아래 코드가 보여주듯이 두 개의 벡터 a와 b를 요소별로 더할 때 파이썬은 for i…와 같은 반복 표현을 사용하지 않는다. 단지 a+b와 같이 코딩하면, 파이썬은 a와 b가 벡터라는 사실을 알고 있으므로 실제 연산을 반복문으로 변환하여 요소 하나씩 순차적으로 계산을 수행한다. 이처럼 +, -, *, /의 사칙 연산자는 모두 요소별로 연산을 적용한다. **는 요소별 제곱을 하는 연산이다. [72]행의 나눗셈 연산에서는 첫번째 요소가 0이라 Inf가 출력되고 경고 메시지가 출력되었다. [73]행의 b<0은 요소별로 0보다 작은지 검사하여 참이면 True, 거짓이면 False가 된다. 벡터화는 병렬 처리를 지원하여 GPU와 같은 병렬 처리 하드웨어를 사용하는 경우 모든 요소를 동시에 계산하여 계산 속도를 높여준다.

```
In [65]: a=np.array([1,2,3,4,5])
In [66]: b=np.array([0,-1,2,6,1])
In [67]: print(3*a)
[ 3  6  9 12 15]
In [68]: print(a**2)
[ 1 4 9 16 25]
```

```
In [69]: print(a+b)
[ 1    1    5 10    6]
In [70]: print(a-b)
[ 1    3    1 -2    4]
In [71]: print(a*b)
[ 0 -2    6 24    5]
In [72]: print(a/b)
[              inf -2.                1.5               0.66666667    5.              ]
__main__:1: RuntimeWarning: divide by zero encountered in true_divide
In [73]: print(b<0)
[False    True False False False]
```

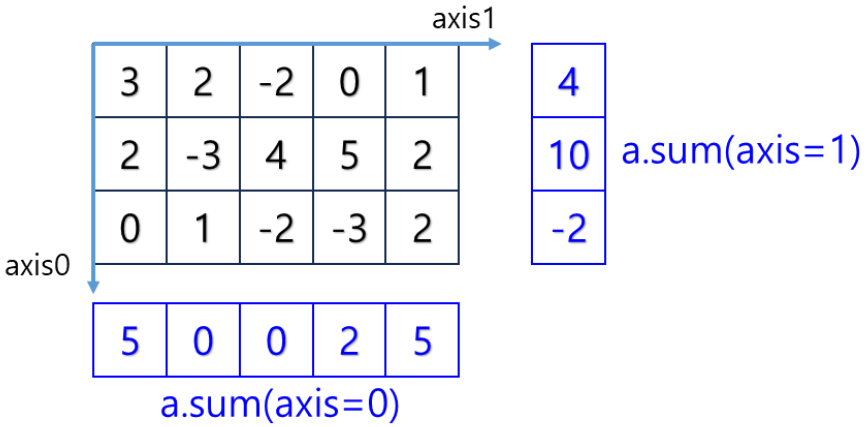
유용한 함수들

numpy 라이브러리는 유용한 함수를 많이 제공한다. 아래 코드는 몇가지 활용 예시를 보여준다. a.sum()은 배열 a의 요소의 합을 구한다. a.sum(axis=0)은 axis0을 기준으로 합을 구하며, a.sum(axis=1)은 axis1을 기준으로 합을 구한다. 축을 기준으로 합을 구한다는 말의 의미를 [그림 A-7]이 설명한다.

a.cumsum(axis=0)은 axis0을 기준으로 누적 합을 계산한다. a.max(axis=0)는 axis0을 기준으로 최대값을 계산하며, a.argmax(axis=0)는 axis0을 기준으로 최대값을 가진 인덱스를 찾아준다. 예시 코드에서는 sum, cumsum, max, argmax를 사용했는데, 추가로 평균을 계산하는 mean, 표준편차를 계산하는 std, 정렬해주는 sort 등이 있다.

```
In [74]: a=np.array([[3,2,-2,0,1],[2,-3,4,5,2],[0,1,-2,-3,2]])
In [75]: print(a)
[[ 3    2 -2    0    1]
 [ 2 -3    4    5    2]
 [ 0    1 -2 -3    2]]
In [76]: print(a.sum())
12
In [77]: print(a.sum(axis=0))
[5 0 0 2 5]
In [78]: print(a.sum(axis=1))
[ 4 10 -2]
In [79]: print(a.cumsum(axis=0))
[[ 3    2 -2    0    1]
 [ 5 -1    2    5    3]
 [ 5    0    0    2    5]]
In [80]: print(a.max(axis=0))
[3 2 4 5 2]
```

```
In [81]: print(a.argmax(axis=0))
[0 0 1 1 1]
```



[그림 A-7] a.sum(axis=0)과 a.sum(axis=1)

때때로 어떤 조건을 만족하는 요소만 골라 연산을 적용할 필요가 있다. 다음 예시는 0보다 큰 요소만 골라 합을 구하는 코드이다.

```
In [82]: positive=a>0
In [83]: print(positive)
[[ True  True False False  True]
 [ True False  True  True  True]
 [False  True False False  True]]
In [84]: b=a[positive]
In [85]: print(b)
[3 2 1 2 4 5 2 1 2]
In [86]: b.sum()
22
```

앞의 코드는 다음과 같이 하나의 명령어로 간결하게 쓸 수 있다.

```
In [87]: a[a>0].sum()
22
```

배열 붙이기

hstack과 vstack은 각각 배열을 가로 방향과 세로 방향으로 이어 붙이는데 쓰는 함수이다. [91]행의 명령어는 a와 b를 수직 방향으로 쌓고 [92]행도 수직 방향으로 쌓는다. [95]행은 수평 방향으로 이어 붙인다. [97]행은 1차원 구조의 배열과 2차원 구조의 배열을 수평

방향으로 이어 붙일 수 없어 오류가 발생한다.

```
In [88]: a=np.array([1,2,3])
In [89]: b=np.array([4,5,6])
In [90]: c=np.array([[7,8,9],[1,4,7]])
In [91]: x=np.vstack([a,b])
In [92]: print(x)
[[1 2 3]
 [4 5 6]]
In [93]: y=np.vstack([a,c])
In [94]: print(y)
[[1 2 3]
 [7 8 9]
 [1 4 7]]
In [95]: z=np.hstack([a,b])
In [96]: print(z)
[1 2 3 4 5 6]
In [97]: zz=np.hstack([a,c])
ValueError: all the input arrays must have same number of dimensions, but the array at index 0 has 1 dimension(s) and the array at index 1 has 2 dimension(s)
```

유니버설 함수와 브로드캐스팅

요소별 연산을 지원하는 함수를 유니버설 함수라 부른다. 앞에서 수행한 사칙 연산과 관계 연산은 요소별로 계산하므로 유니버설 함수에 해당한다. 유니버설 함수는 사칙, 관계, 로그, 제곱근, 삼각함수, 비트, 최대와 최소 등의 다양한 연산을 제공한다. 다음 코드는 이들 연산을 예시한다.

[100]행에서 pi는 원주율 3.141592…이다. [102]행의 add는 유니버설 함수로서 덧셈을 수행한다. [103]행의 log10은 밑이 10인 로그, [104]행의 sin은 사인 함수, [105]행의 left_shift는 왼쪽 시프트, [106]행의 greater는 큰지 비교, [107]행의 maximum은 최대값, [108]행의 round는 반올림을 계산해 준다. round(2)는 소수점 2자리에서 반올림하라는 뜻이다.

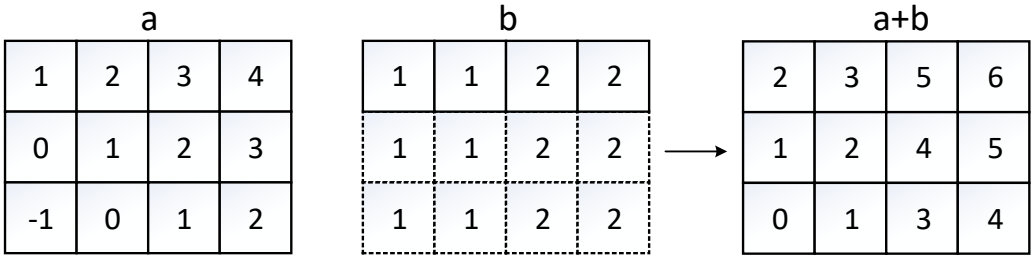
[TIP] 유니버설 함수 목록을 보려면 https://numpy.org/doc/stable/reference/ufuncs.html을 참조한다.

```
In [98]: a=np.array([1,2,3,4,5])
In [99]: b=np.array([0,-1,2,6,1])
In [100]: c=np.array([np.pi/2,np.pi,np.pi*2])
In [101]: print(c)
[1.57079633 3.14159265 6.28318531]
In [102]: print(np.add(a,b))
```

```
[1 1 5 10 6]
In [103]: print(np.log10(a))
[0.    0.30103    0.47712125 0.60205999    0.69897]
In [104]: print(np.sin(c))
[ 1.0000000e+00    1.2246468e-16 -2.4492936e-16]
In [105]: print(np.left_shift(a,1))
[ 2    4    6    8 10]
In [106]: print(np.greater(a,b))
[ True    True    True False    True]
In [107]: print(np.maximum(a,b))
[1 2 3 6 5]
In [108]: print(c.round(2))
[1.05 3.14 6.28]
```

유니버설 함수는 모양이 다른 배열 사이의 연산을 가능하게 해주는 브로드캐스팅broadcasting 기능과 결합하여 더욱 강력하다. [그림 A-8]은 브로드캐스팅 예를 보여준다. 브로드캐스팅은 해당하는 축의 길이가 같거나 둘 중 하나가 1이어야 적용이 가능하다. 아래 코드에서 axis0에 대해서 a는 길이가 3이고 b는 1이다. axis1에 대해서는 a와 b는 길이가 4이다. 따라서 [그림 A-8]이 설명하는 바와 같이 b를 확장하여 모양을 맞춘 다음에 연산을 실행한다.

```
In [109]: a=np.array([[1,2,3,4],[0,1,2,3],[-1,0,1,2]])    # shape: (3,4)
In [110]: b=np.array([[1,1,2,2]])                          # shape: (1,4)
In [111]: print(a+b)                                       # shape: (3,4)
[[2 3 5 6]
 [1 2 4 5]
 [0 1 3 4]]
```



[그림 A-8] 브로드캐스팅

아래 예시를 추가로 살펴보자. 첫번째 예시에서 a의 axis0과 b의 axis0은 각각 길이가 6과 2인데 길이가 다르고 둘 중 하나가 1이 아니므로 적용이 불가능하다. 나머지 예시는 모두 브로드캐스팅이 가능하다. 두번째 예시는 [그림 A-8]에 해당한다. 세번째 예시에서

b(0)은 b는 스칼라임을 뜻하는데, 이 예시는 np.array([1,2,3,4])+5와 같이 1차원 배열에 스칼라를 더하는 경우이다. 이때 [1,2,3,4]+[5,5,5,5]로 브로드캐스팅 되어 [6,7,8,9]가 된다. 네번째는 브로드캐스팅 규칙에 따라 길이가 1인 축이 더 큰 길이로 확장된다. 다섯 번째 예시에서 b(1)은 b=[5]와 같은 경우이다. 이 예시는 축의 개수가 다른 경우인데 이 경우 손실된 축을 길이가 1인 축으로 확장한 다음에 브로드캐스팅을 적용하여 모양을 맞춘다.

a(6,4), b(2,4)	→ 불가능
a(3,4), b(1,4)	→ c(3,4)
a(4), b(0)	→ c(4)
a(5,1,4,1), b(1,6,4,5)	→ c(5,6,4,5)
a(3,4,2), b(1)	→ c(3,4,2)

A.5 matplotlib 라이브러리

컴퓨터 비전은 데이터를 다루는 학문이다. 대부분 데이터는 아주 많은 양의 숫자와 기호로 구성되므로 그 자체를 보고 특성을 파악하거나 패턴을 발견하는 일은 거의 불가능하다. 하지만 그래프를 이용하여 시각화하면 데이터의 특성을 한 눈에 파악할 가능성이 생긴다. matplotlib은 데이터 시각화data visualization에 가장 널리 쓰이는 라이브러리다.

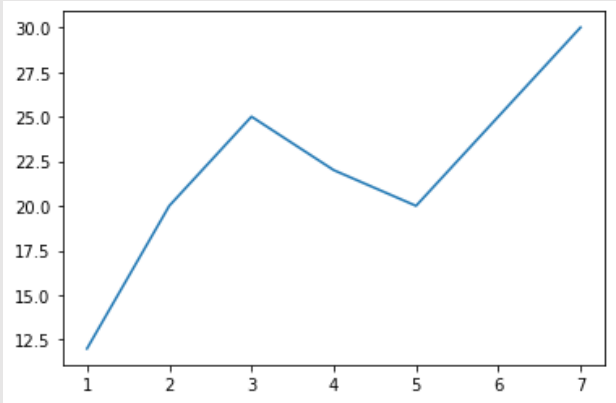
[TIP] matplotlib은 맷플롯립이라고 부른다.

그래프 그리기

아래 코드에서 첫번째 행은 matplotlib.pyplot과 numpy 모듈을 불러온다. 지금부터 이들 모듈이 임포트된 상태라고 가정하고 실습을 진행한다.

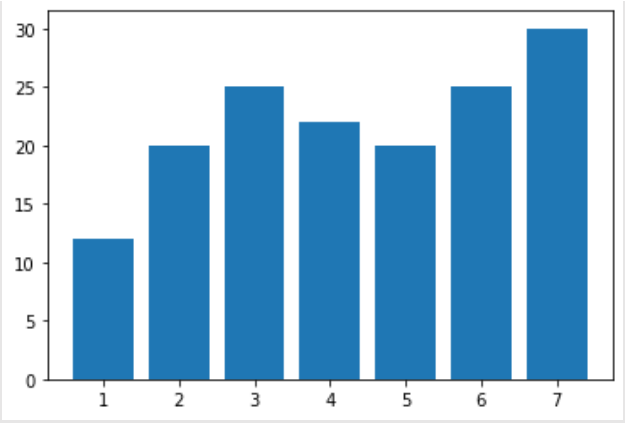
아주 단순한 그래프를 그려보자. 예를 들어 일주일의 과일 판매량을 그래프로 그린다고 가정한다. 7일을 1, 2, 3, …으로 표현하여 x에 저장하고 판매량을 y 변수에 저장한다. 그런 다음 plt.plot(x,y)를 실행하면 멋진 꺾은선 그래프가 생성된다. x는 가로축을 위한 값으로 쓰이고 y는 세로축의 값으로 쓰인다.

```
In [1]: import matplotlib.pyplot as plt, numpy as np
In [2]: x=[1,2,3,4,5,6,7]                # 일주일
In [3]: y=[12,20,25,22,20,25,30]    # 과일 판매량
In [4]: plt.plot(x,y)
```

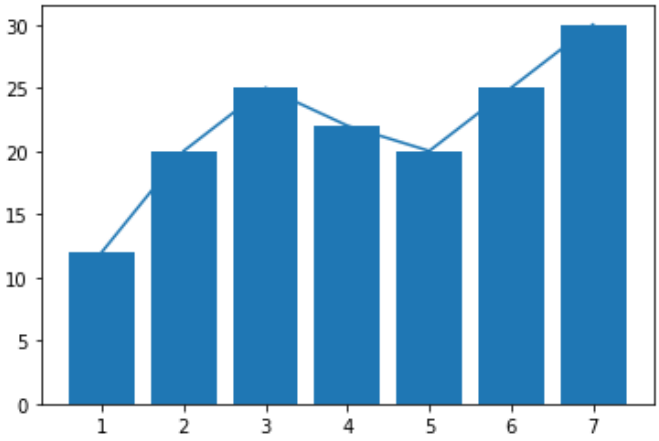


막대 그래프를 그리고 싶으면 plt.bar(x,y)를 실행하면 된다.

```
In [5]: plt.bar(x,y)
```



앞의 예시처럼 plt.plot()과 plt.bar()를 따로 실행하는 경우 별개의 그림이 생성된다. 하지만 스파이더의 스크립트 창에 [1]~[5]행의 코드를 입력하고 실행 버튼을 눌러 한꺼번에 실행하면 아래처럼 하나의 그림에 두 그래프가 겹쳐서 나타난다. matplotlib은 이어서 등장하는 명령어를 하나의 그림에 계속 적용하기 때문이다.



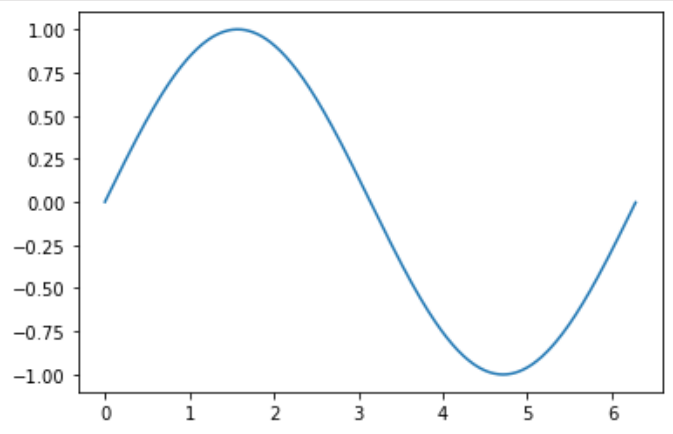
show() 함수가 나타나면 그때 하나의 그림을 완성한 다음, 이후 명령어는 새로운 그림에 적용한다. 예를 들어 아래 코드와 같이 plt.plot() 명령어 다음에 plt.show() 함수를 두면 이 코드를 한꺼번에 실행하더라도 두 개의 그래프를 따로따로 생성해준다.

```
…
plt.plot(x,y)
plt.show()
plt.bar(x,y)
plt.show()
…
```

이제 약간 복잡한 sin 그래프를 그려보자. 가로축은 0도부터 2π도까지 0.05만큼씩

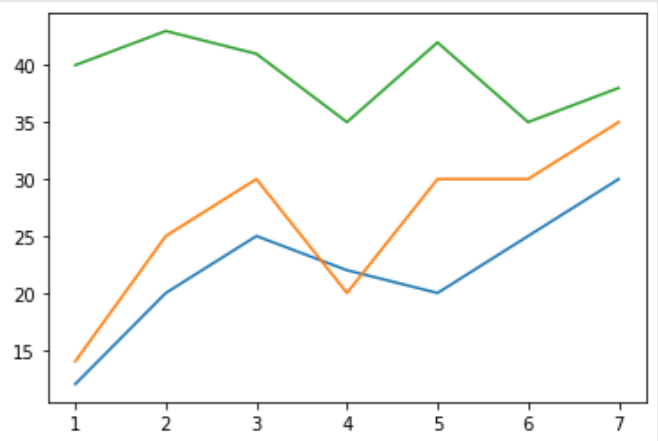
증가시킨다. 아래 코드로 쉽게 멋진 sin 그래프를 그렸다.

```
In [6]: x=np.arange(0,2*np.pi,0.05)
In [7]: y=np.sin(x)
In [8]: plt.plot(x,y)
```



앞에서는 한 개 품목의 판매량을 시각화했는데, 이제 세 개 품목으로 확장해보자. [10]행은 판매량을 저장하는 y를 2차원 구조로 확장한다. 그래프를 그리는 명령어는 앞에서와 똑같이 plt.plot(x,y)이다. 단지 y 변수만 1차원 구조가 2차원 구조로 바뀌었다. matplotlib은 세 개 품목의 데이터를 색깔로 구분하여 보기 좋게 그려준다.

```
In [9]: x=np.array([1,2,3,4,5,6,7])
In [10]: y=np.array([[12,14,40],[20,25,43],[25,30,41],[22,20,35],[20,30,42],[25,30,35],[30,35,38]])
In [11]: plt.plot(x,y)
```

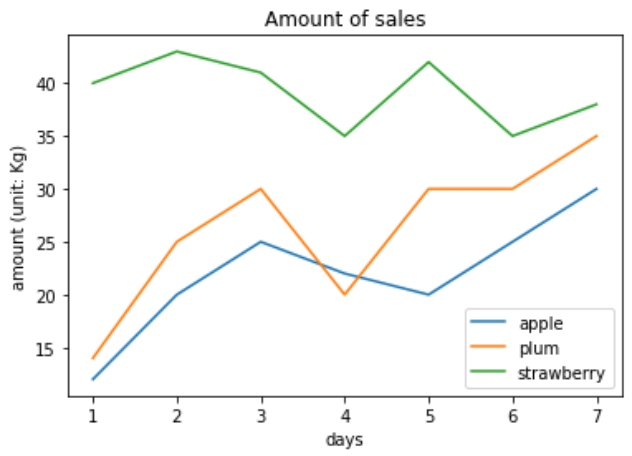


그래프 꾸미기

앞 절에서는 간단한 코드로 멋진 그래프를 얻었다. 그런데 이들 그래프는 여러 측면에서 부족함이 있다. 축이 무엇을 뜻하는지 설명이 없고 세 품목이 무엇인지 표시되어 있지 않다. 선의 색깔이나 굵기도 기본값을 사용하였기 때문에 마음에 들지 않을 수 있다. 이제 내 맘대로 그래프를 꾸미는 프로그래밍 실습을 해본다.

다음 코드는 그래프에 여러 가지 유용한 장식을 덧붙인다. [15]행의 title 함수는 그래프에 이름을 붙여주고, [16]~[17]행의 xlabel과 ylabel은 가로축과 세로축에 이름을 붙여준다. [18]행의 legend 함수의 첫번째 매개변수는 품목 세 개의 설명문이며 두번째 매개변수 loc은 설명문이 나타날 위치를 지정한다. loc 매개변수를 생략하면 기본값인 'best'로 설정되어 네 구석 중 가장 여유로운 곳을 찾아 자동으로 위치를 잡아준다. [19]행의 show 함수는 앞에서 설명한 바와 같이 호출되는 순간 그래프를 완성한다. 이후에 나오는 명령어는 새로운 그림을 시작한다.

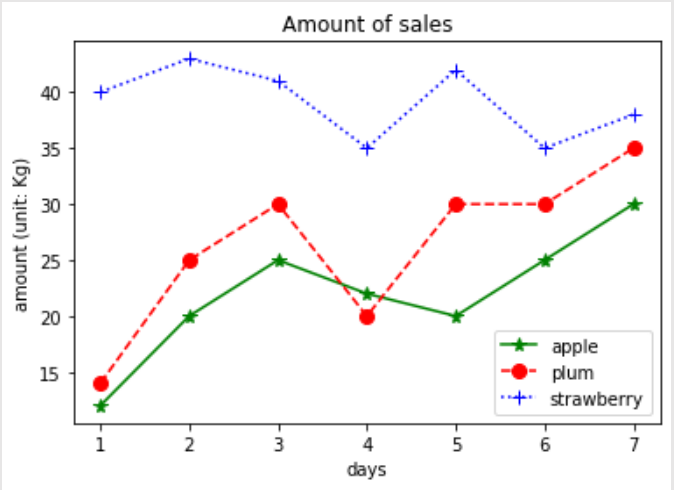
```
In [12]: x=np.array([1,2,3,4,5,6,7])
In [13]: y=np.array([[12,14,40],[20,25,43],[25,30,41],[22,20,35],[20,30,42],[25,30,35],[30,35,38]])
In [14]: plt.plot(x,y)
In [15]: plt.title('Amount of sales')
In [16]: plt.xlabel('days')
In [17]: plt.ylabel('amount (unit: Kg)')
In [18]: plt.legend(['apple','plum','strawberry'],loc='lower right')
In [19]: plt.show()
```



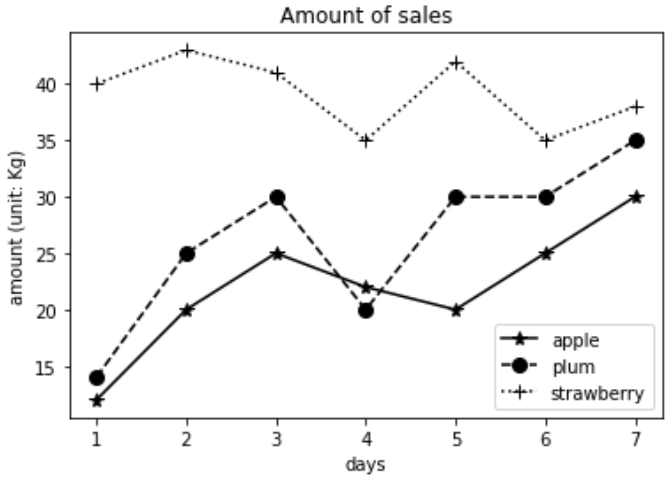
이제 품목을 나타내는 세 개의 선분을 꾸며보자. 위에 있는 코드에서 [14]행을 아래와 같이 세 행으로 바꾸면 된다. 이들 행은 세 개의 품목을 저장한 y를 슬라이싱하여 각각 첫번째 품목 y[:,0], 두번째 품목 y[:,1], 세번째 품목 y[:,2]를 사용한다. 'g*-', 'ro—', 'b+:'에서 g(녹색), r(빨간색), b(파란색)는 색깔을 나타내며, *, o, +는 점을 강조할

마커를 나타내며, - (실선), -- (파선), : (점선)은 선분의 스타일을 나타낸다. linewidth 매개변수는 선분의 굵기, markersize는 마커의 크기를 나타낸다.

```
plt.plot(x,y[:,0],'g*-',linewidth=1.5,markersize=8)
plt.plot(x,y[:,1],'ro--',linewidth=1.5,markersize=8)
plt.plot(x,y[:,2],'b+:',linewidth=1.5,markersize=8)
```



컬러를 쓸 수 없는 상황에서는 마커 또는 선분 모양이 매우 중요하다. 아래 그림은 위의 코드에서 색상을 나타내는 g, r, b를 모두 k로 바꾸고 실행한 결과이다.



격자를 넣어 값을 쉽게 알아볼 수 있게 하려면 plt.grid()라는 명령을 추가하면 된다. 위의 코드에서 plt.plot()과 plt.show() 사이에 아무 곳에나 추가하면 된다. 다음은 격자를 추가한 그림으로 점의 위치를 보다 쉽게 알아볼 수 있다. grid 함수는 매개변수를 통해 격자 간격과 모양을 내 마음대로 꾸밀 수 있다. 스스로 찾아 실험해보기 바란다.

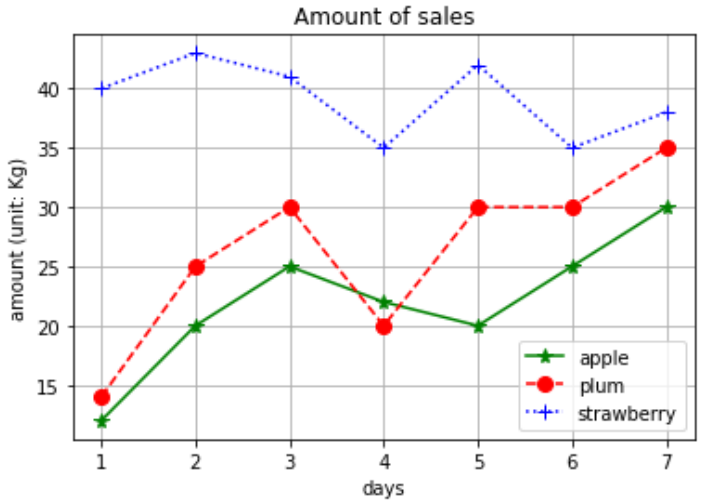


그림 하나에 여러 그래프 배치하기

때때로 여러 개의 그래프를 하나의 그림에 담을 필요가 있다. 예를 들어 세 품목을 따로 그린 세 개의 그래프를 하나의 그림에 그리는 프로그래밍을 해보자. 이런 용도로 사용하는 함수가 subplot이다.

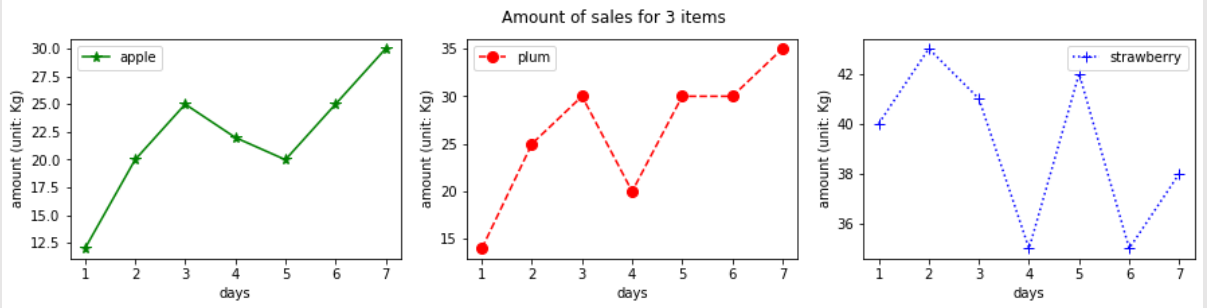
다음 코드는 subplot 함수를 이용하여 세 개의 그래프를 한 줄에 그린다. [22]행은 그림의 크기를 지정한다. figsize=(15,3)은 가로 방향 길이를 15, 세로 방향 길이를 3으로 설정한다. 이 매개변수를 다르게 설정하여 그림 크기가 어떻게 달라지는지 실험해 보기 바란다. [23]행의 suptitle 함수는 그래프 전체의 제목을 위쪽에 붙인다.

이후에 나타나는 [24]~[28]행, [29]~[33]행, [34]~[38]행은 파란색으로 표시한 부분만 다르고 같은 코드를 반복한다. [24]행의 subplot(1,3,1)은 1*3 배치(한 행에 3열 배치)에서 1번 위치에 그리라는 뜻이고, 그 뒤에 따라오는 plot, xlabel, ylabel, legend 함수는 1번 위치에 적용된다.

[29]행의 subplot(1,3,2)는 1*3 배치에서 2번 위치에 그리라는 뜻이고, 그 뒤에 따라오는 plot, xlabel, ylabel, legend 함수는 2번 위치에 적용된다. 같은 과정이 [34]행에도 적용된다. 마지막 명령어인 show 함수는 그래프를 완성하고 생성해준다. legend 함수에서 loc 매개변수를 생략하여 기본값 'best'를 사용하기 때문에 세 개 그래프의 설명문이 자동으로 가장 여유로운 곳에 배치되었다.

```
In [20]: x=np.array([1,2,3,4,5,6,7])
In [21]: y=np.array([[12,14,40],[20,25,43],[25,30,41],[22,20,35],[20,30,42],[25,30,35],[30,35,38]])
In [22]: plt.figure(figsize=(15,3))
In [23]: plt.suptitle('Amount of sales for 3 items')
In [24]: plt.subplot(1,3,1)
In [25]: plt.plot(x,y[:,0],'g*-',linewidth=1.5,markersize=8)
```

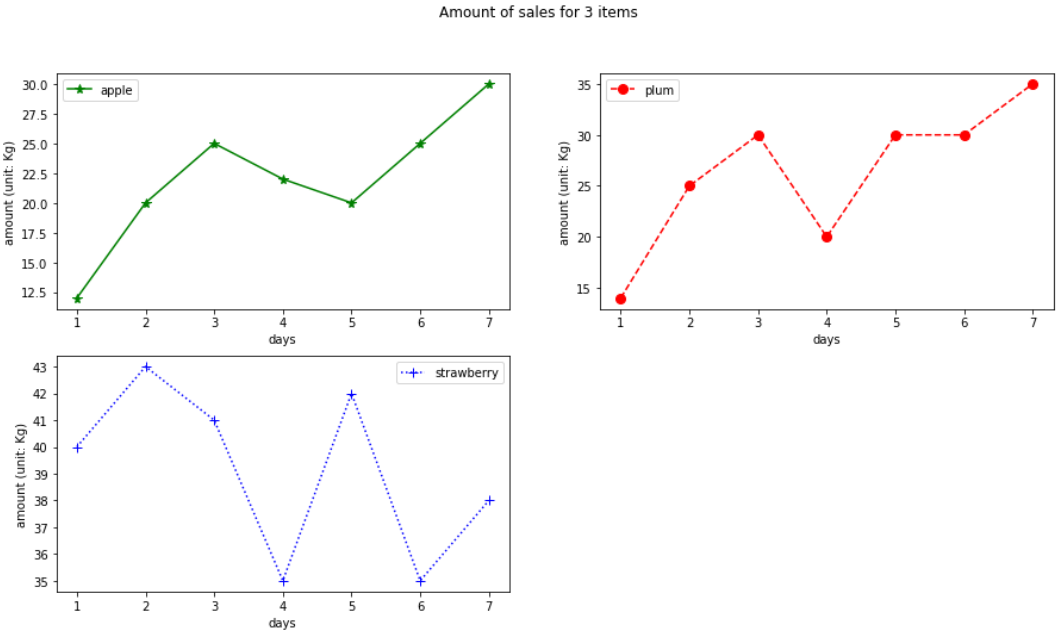
```
In [26]: plt.xlabel('days')
In [27]: plt.ylabel('amount (unit: Kg)')
In [28]: plt.legend(['apple'])
In [29]: plt.subplot(1,3,2)
In [30]: plt.plot(x,y[:,1],'ro--',linewidth=1.5,markersize=8)
In [31]: plt.xlabel('days')
In [32]: plt.ylabel('amount (unit: Kg)')
In [33]: plt.legend(['plum'])
In [34]: plt.subplot(1,3,3)
In [35]: plt.plot(x,y[:,2],'b+:',linewidth=1.5,markersize=8)
In [36]: plt.xlabel('days')
In [37]: plt.ylabel('amount (unit: Kg)')
In [38]: plt.legend(['strawberry'])
In [39]: plt.show()
```



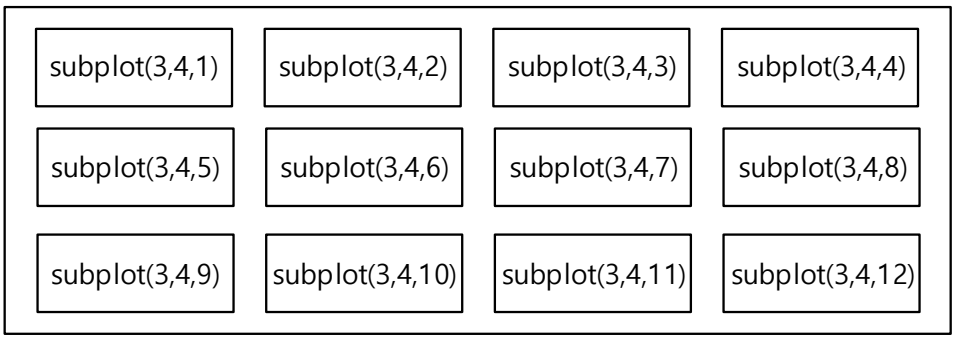
위의 프로그램은 같은 코드 패턴을 세 번 반복한다. 이런 프로그래밍은 바람직하지 않다. 아래 코드는 중복성을 제거한 깔끔한 코드이다. 위의 코드에서 파란색으로 표시한 부분, 즉 반복할 때마다 바뀌는 부분만 반복문의 인덱스 i로 적절히 대치한 프로그램이다.

```
In [40]: x=np.array([1,2,3,4,5,6,7])
In [41]: y=np.array([[12,14,40],[20,25,43],[25,30,41],[22,20,35],[20,30,42],[25,30,35],[30,35,38]])
In [42]: plt.figure(figsize=(15,8))
In [43]: plt.suptitle('Amount of sales for 3 items')
In [44]: decoration=['g*-','ro--','b+:']          # 선분 스타일
In [45]: legends=['apple','plum','strawberry'] # 세 품목의 설명문
In [46]: for i in range(3):
    ...         plt.subplot(1,3,i+1)
    ...         plt.plot(x,y[:,i],decoration[i],linewidth=1.5,markersize=8)
    ...         plt.xlabel('days')
    ...         plt.ylabel('amount (unit: Kg)')
    ...         plt.legend([legends[i]])
In [47]: plt.show()
```

앞의 코드는 한 줄에 세 개, 즉 1*3 형태로 그래프를 배치하였다. 한 줄에 두 개씩 두 줄에 걸쳐 그리고 싶으면, 위 코드에서 plt.subplot(1,3,i+1)을 plt.subplot(2,2,i+1)로 바꾸기만 하면 된다. 위 코드의 [42]행은 figure 함수로 그림의 크기를 지정하는데, [22]행의 figure(figsize(15,3))을 그대로 두면 위아래로 찌그러져 보이기 때문에 figure(figsize(15,8))로 바꾸었다. 아래 그림은 이렇게 얻은 그래프다.



[그림 A-9]는 subplot(r,c,i)가 그래프의 위치를 지정하는 방법을 설명한다. 앞의 두 매개변수 r과 c는 배치 모양을 나타내는데, [그림 A-9]에서는 r=3이고 c=4이어서 3*4 모양으로 배치한다. 모든 subplot 함수 호출에서 앞의 두 매개변수의 값은 3과 4이어야 한다. 세번째 매개변수 i는 3*4 배치의 12곳 위치를 지정하는데 [그림 A-9]가 보여주는 바와 같이 행 우선으로 순서를 매긴다. 즉 첫번째 행은 1,2,3,4이고 두번째 행은 5,6,7,8이고 세번째 행은 9,10,11,12이다.



[그림 A-9] 여러 그래프를 한꺼번에 그리는 subplot(3,4,i) 함수 사용 예시