

공학도를 위한

일단계 파이썬



목포대학교 전기및제어공학과

박장현

제 1 장 파이썬 개요

1.1 파이썬 개요

파이썬(python)은 컴퓨터 언어 중 하나로서 귀도 반 로섬(Guido V. Rossum)에 의해서 1990년에 초기 버전이 발표되었다. (2021년 6월 현재 버전이 3.10.x) 파이썬이라는 이름은 본인이 좋아하는 “Monty Python’s Flying Circus” 라는 코미디 쇼에서 따왔다고 한다. 파이썬의 사전적인 뜻은 큰 비단뱀인데 대부분의 파이썬 책 표지와 아이콘이 뱀 모양으로 그려져 있는 이유가 여기에 있다.



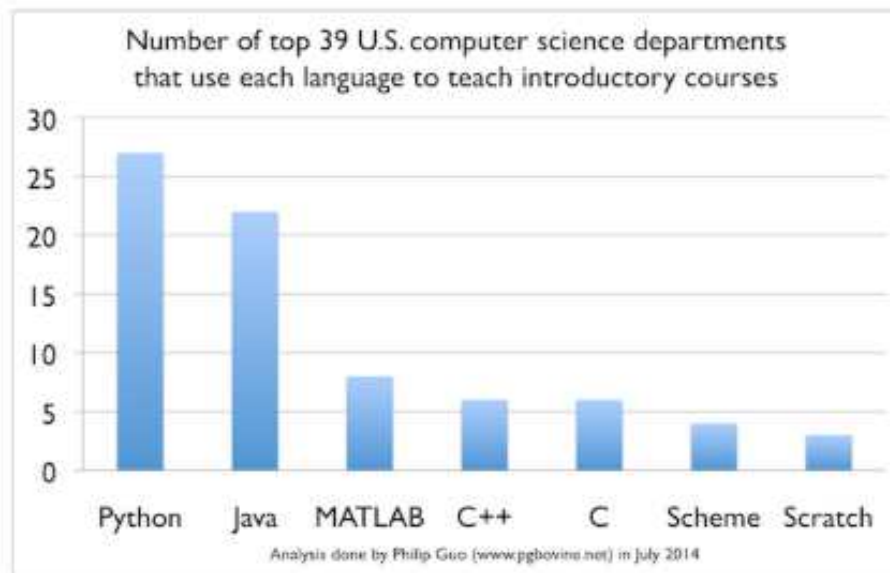
<그림 1.1.1> 파이썬 개발자와 파이썬 로고

파이썬 프로그램의 가장 큰 특징은 배우기 쉽고 직관적이라는 점이다. 인터프리터 언어이기 때문에 실행 결과를 즉시 확인해 볼 수 있으며 최근에는 실행 속도도 심지어 C/C++ 프로그램과 맞먹을 정도로 성능도 향상되고 있다. 또한 공동 작업과 유지 보수가 매우 쉽고 편하기 때문에 이미 다른 언어로 작성된 많은 프로그램과 모듈들이 파이썬으로 다시 재구성되고 있기도 하다. 전 세계적으로 그 가치를 인정받아 사용자 층이 넓어져 가고 있고, 파이썬을 이용한 프로그램을 개발하는 기업체들도 늘어가고 있는 추세이다.

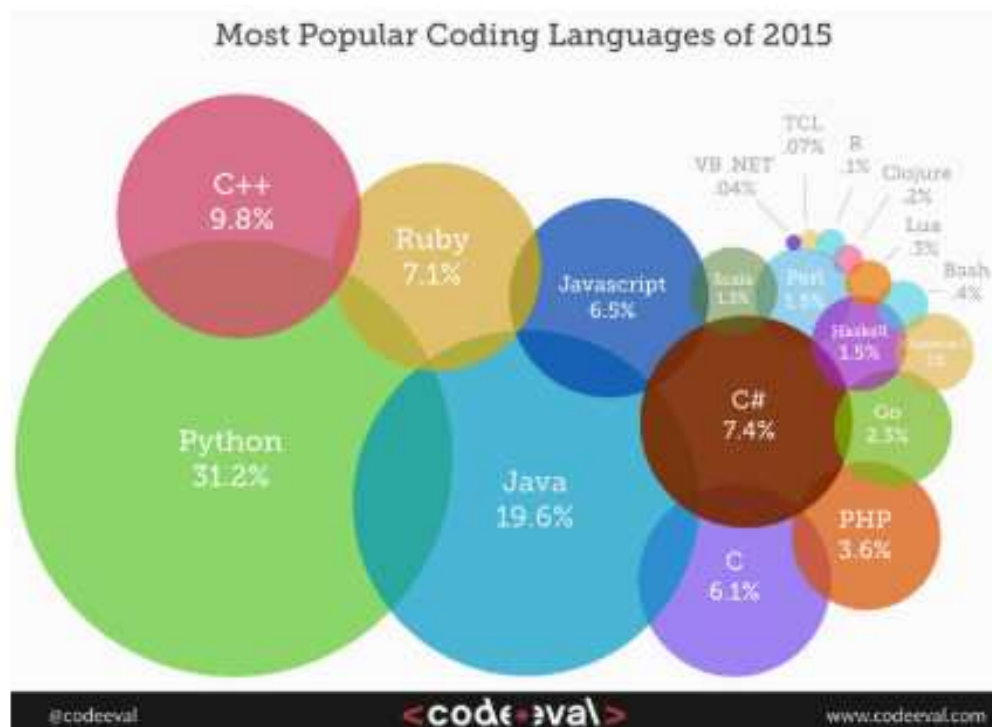
현재 파이썬은 교육의 목적뿐만 아니라 주요 산업체에서도 널리 사용되고 있는데 그 대표적인 예는 바로 구글(Google) 이다. 구글에서 만들어진 소프트웨어의 50% 이상이 파이썬으로 만들어졌다고 한다. 예를 들면 구글의 비디오 서비스인 Youtube는 프론트-엔드(front-end)와 API(application program interface)에 파이썬을 활용하고 있다. 이 외에도 유명한 것을 몇 가지 들어보면 Dropbox(파일 동기화 서비스), Django(파이썬 웹 프레임워크)등을 들 수 있다. 또한 빅데이터 분석(pandas) 이나 과학 계산 용도(numpy, scipy)로도 활발히 활용되고 있으며, 더욱 최근에는 딥러닝(deep learning)을 구현하는 용도의 다양한 파이썬 라이브러리(pytorch, tensorflow 등)도 주목받고 있다. 이와 같이 파이썬은 컴퓨터를 활용한 거의 모든 곳에 사용되고 있다고 해도 과언이

아닐 정도로 인기를 끌고 있다.

이러한 파이썬의 활용성 때문에 미국 대학들의 컴퓨터 관련 학과에서 프로그래밍 입문 교육에서 채택되는 언어들 중 파이썬이 가장 높은 비율을 차지하고 있다.



<그림 1.1.2> 미국 상위 39개의 컴퓨터 과학 학과의 입문용 프로그래밍 언어 순위



<그림 1.1.3> 가장 널리 사용되는 프로그래밍 언어들

파이썬의 특징을 정리하면 다음과 같다.

- 간결하고 쉬운 문법으로 빠르게 학습할 수 있다.
- 오픈 소스(open source)이며 무료이다.
- 강력한 성능을 가진다.
- 상대적으로 개발 속도가 빠르다.
- 다양한 분야에 적용할 수 있는 라이브러리가 풍부하다.
- 교육계, 연구기관, 산업체 등에서 널리 사용되고 있다.
- 스스로 학습할 수 있는 자료가 풍부하다.
- 어떠한 운영체제에서도 파이썬은 동일하게 구동된다. (즉, 윈도우즈, 맥, 리눅스 등 어떠한 기기에서 구동되든지 동일한 파이썬 프로그램은 동일한 동작을 수행한다.)

최근에 마이크로비트(micro:bit)와 같은 마이크로컨트롤러(micro-controller) 보드나 라즈베리파이(raspberry pi) 같은 원보드 컴퓨터(single board computer, SBC)이 인기를 끌고 있는데 이것들의 제어용 프로그래밍 언어로 파이썬이 사용된다. 전술한 바와 같이 파이썬은 리눅스에서도 동일하게 실행되고 최근에는 마이크로컨트롤러용 파이썬인 마이크로파이썬(micro-python)이라는 도구도 개발되어 파이썬의 활용폭을 넓히는데 기여하고 있다. 이러한 시스템에서도 파이썬을 이용하여 전통적인 언어인 C/C++ 등으로 개발하는 것보다 훨씬 더 쉽고 빠르게 응용 프로그램을 제작할 수 있다. 특히 라즈베리파이와 같은 SBC같은 경우 파이썬을 공식 개발 언어로 지정하고 지원하고 있을 정도로 코딩 교육용으로도 파이썬은 각광을 받고 있다.



<그림 1.1.4> 마이크로비트와 라즈베리파이

현재 파이썬은 버전이 2.x 대와 3.x 대로 나뉘어 두 가지 버전이 같이 사용되고 있다는 점이 처음 접하는 사용자가 선택하는데 혼동을 줄 여지가 있다. 특이하게도 3.x 버전의 문법이 2.x버전과는 달라서 100% 호환되지 않으므로, 같은 언어로 작성한 프

로그라인데도 불구하고 2.x 버전에서 잘 작동되는 것이 3.x 버전에서는 작동하지 않거나 반대의 경우도 발생한다. 하지만 2008년에 버전 3이 처음으로 공개된 이후 점차로 응용 분야를 넓혀가고 있고 버전 2에서 구동되는 주요한 모듈과 라이브러리들은 버전 3로 업그레이드되어 사용하는데 지장이 없는 경우가 대부분이다. 따라서 본 교재에서는 3.x 버전의 문법 위주로 설명한다.

1.2 파이썬 설치하기

본 교재에서는 초보자라도 손쉽게 구축할 수 있는 최소한의 환경에서 파이썬 실습을 진행하고자 하며 학습자는 윈도우즈 운영체제를 사용한다고 가정한다. 이를 위해서 python.org 사이트에서 윈도우즈용 설치 프로그램을 다운로드 후 설치한다.



<그림 1.2.1> python.org 페이지의 Downloads 메뉴

그러면 IDLE라는 프로그램을 이용하여 파이썬셸(python shell)을 이용하거나, 혹은 파이썬 프로그램을 작성한 후 실행시킬 수 있다.

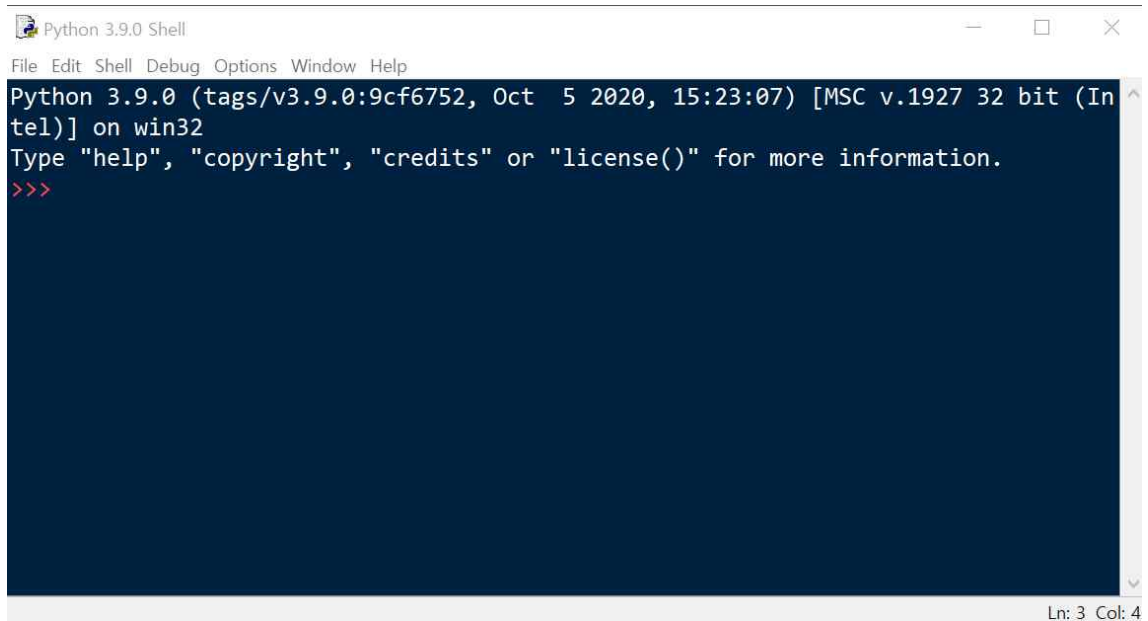
윈도우 커맨드창에서 파이썬을 실행시키려면 파이썬을 설치할 때 다음 그림과 같이 path를 반드시 등록해 주어야 실행을 할 수 있다. 설치가 끝나면 IDLE를 실행하면 (또는 윈도우 cmd 환경에서 python 이라는 명령을 수행하면) 파이썬셸(python shell)이 실행된다. 파이썬 명령어를 입력하고 그 실행 결과를 확인할 수 있다. 이렇게 파이썬 명령어를 실행할 수 있는 환경을 파이썬셸(python shell)이라고 한다.

파이썬셸에서 quit() 함수 혹은 exit() 함수를 호출하면 종료된다.

- 파이썬셸 (python shell)
 - 파이썬 명령을 입력/실행할 수 있는 명령창



<그림 1.2.2> 윈도우즈에서의 파이썬 설치창



<그림 1.2.3> 파이썬을 설치한 후 IDLE를 실행한 모습

1.1.3 파이썬을 계산기처럼 사용해 보기

이제 파이썬셸에서 다음과 같이 산술 연산을 수행할 수 있다. 예를 들어 곱셈을 수행하고 그 결과를 보려면 *기호를 이용하면 된다. 기본적인 산술 연산자는 다음과 같다.

산술 연산	연산자
덧셈	+
뺄셈/음부호	-
곱셈	*
나눗셈	/

<표 1.1> 파이썬의 사칙 연산자

단순한 산술연산과 숫자를 표기하는 것은 수학책에서의 표기법과 같다. 단, 곱셈은 x가 아니라 * 기호를 사용하고 나눗셈은 수학 기호가 아니라 / 이라는 것만 다르다. 여러가지 숫자들로 실습을 해자.

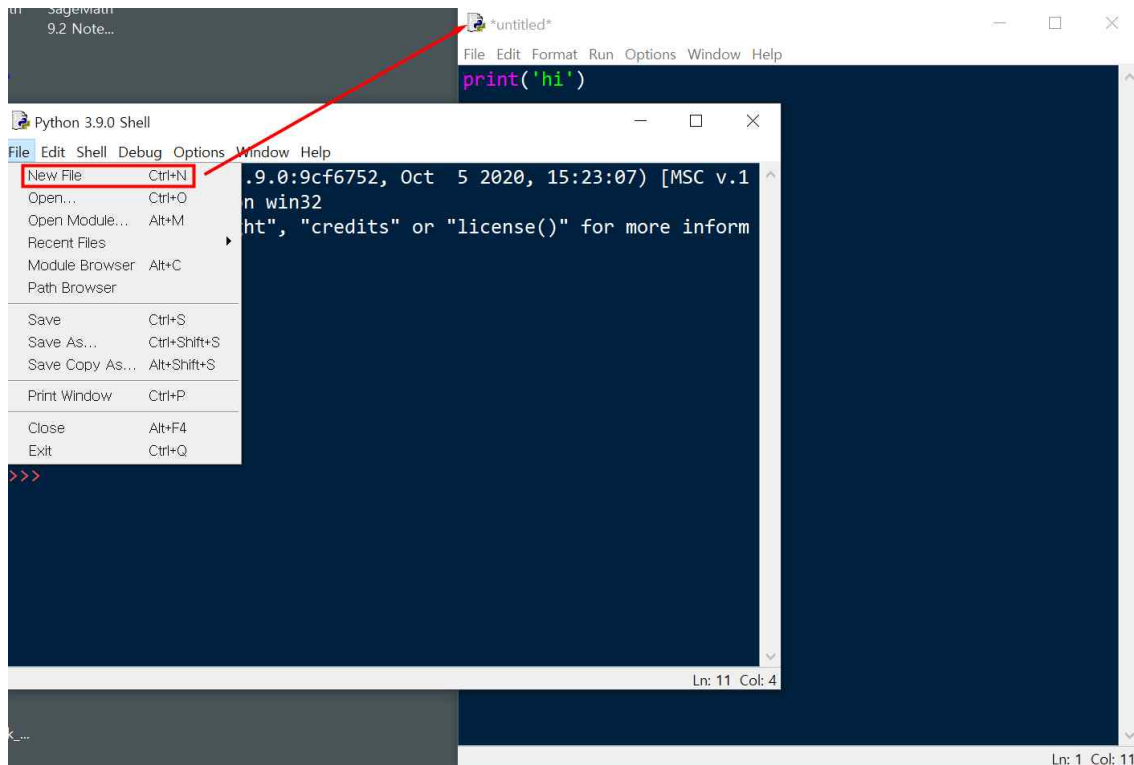
```
>>> 11+22
33
>>> 103-44.5
58.5
>>> -12.2*3.14156
-38.327032
>>> 45/32.1
1.4018691588785046
```

1.1.4 텍스트 에디터 사용하기

IDLE는 파이썬셸로 사용할 수 있을 뿐만 아니라 간단한 기능을 갖는 편집기를 포함하고 있다. 이 편집기에서 파이썬 프로그램을 작성한 후 파일로 저장할 수 있고, 그 실행 결과를 파이썬셸에서 확인할 수도 있다. IDLE의 file > New File 메뉴를 선택하면 편집창이 생성되고 여기에서 파이썬 프로그램을 입력하고 편집한 후 파일로 저장할 수 있다.

편집창에 다음과 같이 입력해 보자.

```
print('hello python')
```



<그림 1.2.4> IDLE에서 편집창 생성

이 것을 적당한 폴더에 입력한 프로그램을 파일로(파이썬 프로그램 파일의 확장자는 .py 이다) 저장한다. 파일 이름은 hello.py라고 설정한다. 파일로 저장한 후에는 **Enter** 키를 누르면 그 실행결과를 IDLE의 파이썬셸에서 확인할 수 있다.

- 파이썬 프로그램의 확장자는 .py 이다.

이것으로 파이썬을 입력하고 실행시킬 수 있는 실습 환경은 다 마련된 것이라고 할 수 있다.

제 2 장 변수와 숫자형

2.1 변수의 이름(식별자)

변수(variable)에 자료(data)를 저장할 수 있으며 고유한 이름을 가져야 한다. 변수의 이름(혹은 함수명, 클래스명 등)으로 쓸 수 있는 식별자(identifier)를 만드는 방법은 다른 언어들과 거의 동일하다. 식별자를 만들 때 보통 사용되는 문자들은 다음과 같다.

- 영문자 대소문자 (a,b ... z A B ... Z)
- 숫자 (0 1 ... 9)
- 언더바(_)

변수명을 만드는 규칙은 다음과 같다.

- 위 문자들을 조합한다.
- 숫자로 시작하면 안된다.

예를 들면 다음과 같다. (변수 혹은 함수의 이름이 여러 단어로 이루어진 경우 단어 사이를 언더바(_)로 구분한다.)

```
a, x, num, obj, count, is_prime, get_number_of_object
```

파이썬에서는 대문자와 소문자를 다른 문자로 구별한다. 즉, 다음과 같은 변수들은 모두 다른 것들로 간주된다.

```
aaa, Aaa, aAa, aaA, AAa, aAA, AaA, AAA
```

그리고 python3에서는 알파벳 외에도 유니코드 문자를 식별자로 사용할 수 있다. 즉, 한글로도 변수명을 만들 수 있지만 특별한 상황이 아니라면 권장되지는 않는다.

변수에 어떤 자료를 저장하기 위해서 = 기호 (대입연산자라고 한다)가 사용된다.

예를 들어 변수 a에 11이라는 숫자를 저장하려면 다음과 같다.

```
>>> a=11
```

이 명령을 내리는 순간 변수 a가 생성되고 그것에 11이라는 숫자가 저장된다. 이후에 변수 a에 저장된 내용을 보고 싶다면 파이썬셸에서 변수명만 치고 엔터키를 누르거나 print()함수를 이용하면 된다.

```
>>> a 
11
>>> print(a) 
11
```

파이썬셸 내에서 단순히 변수의 내용을 확인하고 싶은 경우에는 전자의 방법이 더 선호된다.

파이썬3에서는 한글로도 변수명을 짓는 것이 가능하다.

```
>>> 번호 = 3 
>>> 번호 
3
```

한국어 뿐만 아니라 일본어와 중국어 등도 변수명에 사용할 수는 있다. 재미있는 기능 이긴 하지만 보통 변수명에 알파벳 이외의 문자를 포함시키면 가독성이 현저하게 떨어지므로 권장되지 않는다.

2.2 파이썬의 기본 자료형의 종류

프로그램은 결국 자료(data)를 처리하는 일을 주로 하게 된다. 여기에서는 파이썬의 기본 자료형의 종류에 대해서 알아보자. 가장 중요하고 사용 빈도가 높은 파이썬 자료형은 <표 2.1>에 정리하였다.

〈표 2.2.1〉 사용 빈도가 높은 파이썬 자료형들

구분	내용	type()
None	아무 자료형도 아님을 나타냄	'NoneType'
숫자형	정수, 실수, 복소수 등	'int', 'float', 'complex'
문자열	문자들의 나열	'str'
진리값	참(True) 혹은 거짓(False)	'bool'
리스트	데이터의 묶음 (가변)	'list'
튜플	데이터의 묶음 (불가변)	'tuple'
딕셔너리	키(key)와 값(value)쌍들의 묶음	'dict'
집합	중복되지 않는 요소들의 묶음	'set'

이 표에 보면 자료형 중에서 None이라는 것이 있는데 이것은 ‘비어 있음’ 혹은 ‘아무것도 없음’ 을 표시하는 단일값으로 변수에 직접 대입할 수도 있다.

```
>>> a=None
>>> print(a)
None
>>> b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
>>>
```

어떤 변수가 아예 정의되어 있지 않은 경우(undefined)와, 변수의 값이 None인 경우는 구별해야 한다. 전자는 변수가 아예 존재하지 않는 것이고(위에서 변수 b의 경우 한 번도 사용된 적이 없기 때문에 undefined이다) 후자는 변수가 존재하되 거기에 저장된 값이 None인 경우이다.(위에서 변수 a의 경우)

2.3 숫자 자료형

파이썬에서 사용되는 숫자 자료를 세분하면 다음 표와 같다. 단, 이것은 내부적인 구분으로서 사용자 입장에서는 하나의 ‘숫자형’ 이라고 인식하고 사용하면 된다. 〈표 2.1〉에 분류된 것 말고 복소수(complex)도 있는데 그것은 이 장의 뒤에서 설명하


```

298
>>> f 
109
>>> g 
668

```

정수는 10진수/16진수/8진수/2진수로 표현할 수 있으며 실수형은 십진수 형식으로만 표현된다. 숫자에 소수점이 포함되어 있다면 실수, 그렇지 않다면 정수로 분류된다. 지수적 표기법을 사용하여 (즉 e/E 문자를 사용하여) 만든 숫자는 내부적으로 실수형으로 생성된다. 예를 들어 2e5의 값은 2×10^5 이므로 200000이니까 정수값이긴 한데 내부적으로 200000.0인 실수형으로 생성된다.

```

>>> a=2
>>> b=2.0
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(1e10)
<class 'float'>

```

파이썬에서는 정수의 범위에 제한이 없다는 것도 특이한 점이다. 이러한 점에서 파이썬이 매우 큰 숫자를 다뤄야 하는 분야나(예를 들어서 천문학, 경제학) 데이터 분석에서 타 언어보다 유리하다는 점을 짐작할 수 있다.

십진수의 이진수 표현을 알고 싶다면 bin()이라는 내장함수를 사용하면 된다. 팔진수는 oct(), 십육진수는 hex()함수를 이용한다.

```

>>> a=1234
>>> bin(a)
'0b10011010010'
>>> oct(a)
'0o2322'
>>> hex(a)
'0x4d2'

```

단, 이 함수들은 정수형에 대해서만 동작하며 반환값이 문자열이라는 것에 유의해야

한다.

한 줄에 여러 개의 변수에 숫자값을 저장해 주고 싶다면 다음과 같이 쉼표를 이용하면 된다.

```
>>> a, b, c = 11, -22.4, 3e10 Enter↵
```

위 예제는 변수 a에 11을, b에는 -22.4를 c에는 3e10 저장되므로, 다음과 같이 세 줄로 따로 입력한 것과 동작이 완전히 동일하다.

```
>>> a = 11 Enter↵
>>> b = -22.4 Enter↵
>>> c = 3e10 Enter↵
```

만약 어떤 변수의 값을 파이썬 셸에서 확인하고 싶다면 파이썬셸에서 입력하고 엔터를 치면 변수의 값을 보여준다.

```
>>> a Enter↵
11
>>> b, c Enter↵ #❶
(-22.4, 30000000000.0)
```

위에서 보인 바와 같이 두 개의 변수를 동시에 확인하고 싶다면 ❶과 같이 쉼표로 변수를 구분한 후 엔터키를 누르면 된다.

2.4 산술 연산자

〈표 2.4.1〉 파이썬의 산술연산자

연산자	기능	예제 (a, b는 변수)
+	덧셈	11+22, a+12, a+b
-	뺄셈	11-22, a-12, a-b
*	곱셈	11*22, a*b
/	나눗셈	11/22, a/b (결과는 항상 실수형)
//	자리내림 나눗셈	나눗셈의 결과값에서 소수점 아래는 버리고 정수만 취한다.
**	거듭제곱	2**10
%	나머지	3%4, -10%3, a%2,

파이썬의 산술 연산자는 앞에서 이미 설명한 네 개(+, -, *, /)를 포함해서 <표 1.2>에 나열된 것들이 있다. 여기서 나눗셈의 경우 정수들 간의 나눗셈의 결과는 나누어 떨어지는 경우 결과값이 정수형이라고 오해하기 쉬운데 나눗셈의 결과는 무조건 실수형이다. 즉, 1/2는 0.2, 6/3은 2.0이 된다. 다음 결과를 확인해 보자.

```
>>> 4/5 
0.8
>>> a,b=11,5 
>>> b/a 
0.45454545454545453
```

연산자 //는 정수몫을 구하는 연산자이다. 두 피연산자가 모두 정수형일 경우 결과값은 정수형이지만, 둘 중 하나라도 실수형이면 결과값도 실수형이다.

```
>>> 9//2  #결과는 4(정수)
4
>>> 9//2.0  #결과는 4.0(정수)
4.0
```

연산자 %는 나눗셈 수행 후 정수몫을 구하고 난 나머지를 구하는 연산자이다. 다음을 확인해보라.

```
>>> 3%2
1
>>> 11/5%0.2
5.551115123125783e-17
>>> 11.5%0.2
0.09999999999999937
>>> 11.5%0.3
0.10000000000000042
```

나머지 연산자(%)는 보통 두 개의 피연산자가 모두 정수인 경우에 사용한다. 실수에 대해서 %연산을 하는 경우는 결과값이 정확하지 않을 수도 있고 미세한 오차가 발생할 수도 있으므로 사용에 유의해야 한다.

연산자 **는 거듭제곱 연산자이다. 다음 연산의 예를 보자.

```
>>> 2**10
```



```
1024
>>> 3**20
3486784401
>>> 4.1**10
1342265.9310152389
```

지수는 정수뿐만 아니라 실수도 될 수 있으므로 거듭제곱 연산자를 이용하면 제곱근 값도 쉽게 구할 수 있다. 예를 들어 $\sqrt{2}$ 값은 2(밑)의 0.5(지수)승이므로 다음과 같이 하면 된다.

```
>>> 2**0.5
1.4142135623730951
```

<표 2.4.2> 산술연산자의 우선순위

연산 순위	연산자	비고
1	괄호(...)	
2	**	
3	-(음부호), +(양부호)	부호연산자
4	*, /, //, %	
5	+ (덧셈), - (뺄셈)	덧셈과 뺄셈

어떤 수식에서 수행되는 여러 연산은 어떤 것을 다른 것보다 먼저 수행해야 하는지가 정해져 있고 이를 연산 우선 순위라고 한다. 수식에서 괄호(..)로 둘러싸인 부분이 먼저 계산된다. 산술연산자는 우선순위가 정해져 있으며 일반적인 수학 연산의 경우와 동일하다. 즉, 가장 연산 순위가 높은 것은 거듭제곱(**)연산자이고 그 다음으로 부호 연산자 그리고 곱셈/나눗셈/나머지 연산자, 가장 연산 순위가 낮은 것은 덧셈과 뺄셈이다. 예를 들면 다음과 같다.

```
>>> 5/5 + 1*9 Enter↵
>>> 3 + -2*3 Enter↵
>>> -2.2*-1 - 40 Enter↵
>>> 2**-2 Enter↵
>>> 2+3**2-4 Enter↵
>>> 2*4/2%2 Enter↵
```

```
>>> -2**-2 Enter ↵ (결과값을 예상해 보자)
```

연산 순위가 같은 연산자들은 왼쪽에서부터 오른쪽으로 연산이 진행된다. 예를 들어 위의 예 중에서

$$2*4/2\%2$$

는 *과 / 그리고 %의 우선순위가 같으므로

$$((2*4)/2)\%2$$

와 동일하게 연산이 수행된다. 연산 우선 순위를 명확히 하기 위해서 의도적으로 괄호를 사용하는 것도 좋은 방법이다. 위의 예제에서

$$-2**-2$$

는 **가 앞의 -(음부호)보다 우선순위가 높기 때문에 결과값이 -0.25이다. 하지만 계산 의도를 명확히 하기 위해서

$$-(2**-2)$$

로 작성하는 것이 가독성을 더 높일 수도 있을 것이다.

2.5 복합 연산자

〈표 2.5.1〉 복합연산자

복합 연산자	동작	예제
+=	변수에 값을 더함	a+=1
-=	변수에서 값을 뺌	b-=10
=	변수에 값을 곱함	c=1.2
/=	변수에 값을 나눔	d/=-3
//=	변수에 자리내림 나눗셈 수행	x//=2
=	변수에 값을 거듭제곱 수행	y=3
%=	변수에 나머지를 저장	z%=2

변수의 값에 어떤 값을 더하거나 빼는 동작을 수행한 후 결과값을 다시 그 변수에

대입하는 역할을 하는 연산자를 복합 연산자라고 한다. 예를 들면 `a+=1` 이라고 입력하면 현재 `a`변수값에 1을 더한 결과값을 다시 변수 `a`에 저장한다. 결과적으로 `a`변수에 저장된 값이 1 증가시키는 동작을 수행한다. 여기서 `+=` 는 ‘덧셈’ 연산과 ‘대입’을 한 번에 수행하므로 복합 연산자라고 칭한다. 파이썬의 복합연산자는 다음 표에 기술된 것들이 있다. 다른 예를 들면 다음과 같다.

```
>>> a, b = 11, 3 Enter↵
>>> a+=1 Enter↵
>>> a-=b Enter↵
>>> b*=2 Enter↵
>>> b/=3 Enter↵
>>> a Enter↵
9
>>> b Enter↵
2.0
```

복합연산자를 사용할 때 한 가지 주의할 점은 좌변의 변수에 반드시 어떤 값이 미리 저장되어 있어야 한다는 점이다. 한 번도 사용한 적이 없는 변수에 복합연산자를 사용하려고 하면 오류가 발생한다.

```
>>> x-=1 Enter↵
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

위에서 변수 `x`는 한 번도 사용한 적이 없는 변수이므로 복합연산자 `--`를 사용하려고 하면 오류가 발생함을 알 수 있다.

다른 언어들, 예를 들어 C, C++, C#, JAVA, Javascript 등에 존재하는 증감연산자 (`++`, `--`)가 파이썬에는 없다는 사실도 유의해야 한다. 파이썬에서는 `a`변수를 1만큼 증가시키려면 `a+=1`, 그리고 1만큼 감소 시키려면 `a-=1` 이라고 복합연산자를 사용해야 한다.

2.6 복소수 (선택)

파이썬에서 복소수도 표현할 수 있다. 복소수는 일반적인 코딩에서 사용 빈도가 높지는 사용되지 않지만, 수학 관련 라이브러리를 사용해야 한다거나 혹은 공대생이라면 복소수를 다루는 법에 대해서도 알고 있는 것이 좋다. 복소수는 숫자 뒤에 영문자

‘j’ (혹은 ‘J’)를 붙여서 표시한다.

```
>>> d1=1+2j Enter↵
>>> d2=1j Enter↵
>>> d3=-3-1j Enter↵
```

복소수의 허수부를 입력할 때 j를 숫자 앞에 붙이면 안 된다. 즉, 위의 d1변수를 $d1=1+j2$ 라고 하면 안되는데 j2는 식별자로 인식되기 때문이다. 허수부로 입력하려면 반드시 숫자 뒤에 j를 붙여야 한다. 그리고 허수를 표기할 때는 (d2변수) 1j와 같이 입력해야 한다. 예를 들어서 $d2=j$ 라고 입력하면 j는 역시 식별자로 인식된다.

숫자에 허수부가 포함되어 있다면 복소수이다. 허수부가 0이라도 (0j, -0j, +0.0j, -0.0j 등) 내부적으로 'complex'로 간주된다. type()함수를 이용하여 이를 확인할 수 있다.

```
>>> a=3+4j
>>> type(a)
<class 'complex'>
>>> type(2+0j)
<class 'complex'>
```

복소수는 실수부, 허수부, 켈레복소수를 구할 수 있는 속성(attribute)이 있는데 각각 real, imag, conjugate() 이다. 다음을 실습해 보자.

```
>>> c=11+22j
>>> c.real
11.0
>>> c.imag
22.0
>>> c.conjugate()
(11-22j)
```

또한 내장함수인 abs()를 이용하면 복소수의 크기를 구할 수 있다. (내장함수는 곧바로 사용할 수 있는 함수로서 앞에서 나온 print(), type()과 같은 것들이다.)

```
>>> c=11+22j
>>> abs(c)
24.596747752497688
```

복소수에 대해서도 산술연산자를 사용할 수 있다, 단, //연산자와 %연산자는 복소수에 대해서는 사용할 수 없다. 또한 복합연산자 +=, -=, *=, /=, **= 도 복소수에 대해서 사용할 수 있다.

```
>>> a=3+4j
>>> b=1-1j
>>> a+b
(4+3j)
>>> a-b
(2+5j)
>>> a*b
(7+1j)
>>> b/a
(-0.04-0.28j)
```

위에서 b/a 의 경우 a 의 켤레복소수를 분자/분모에 곱한 결과값이라는 것을 쉽게 확인할 수 있다.

복소수가 밑인 거듭제곱도 간단하게 구할 수 있다.

```
>>> (3-4j)**3
(-117-44j)
>>> (1+2j)**10
(237-3116j)
>>> (4+3j)**0.5
(2.1213203435596424+0.7071067811865476j)
```

위에서 $(3-4j)**3$ 은 $(3-4j) \times (3-4j) \times (3-4j)$ 와 같다. 개념적으로는 간단하지만 손으로 계산하기에는 쉽지 않다. 승수가 높아질수록 더 그렇다. 복소수가 지수인 거듭제곱은 손으로 계산하려면 오일러공식을 이용해야 하지만 파이썬에서는 쉽게 구할 수 있다.

```
>>> 3**(4+5j)
(57.00434317675227-57.54567628403595j)
```

여기에서 $3**(4+5j)$ 는 왜 저런 결과가 나왔을까? 다음과 같이 오일러(Euler)공식을 이용하면 된다.

$$\begin{aligned}
 3^{4+j5} &= e^{\ln(3^{4+j5})} \\
 &= e^{(4+j5)\ln 3} \\
 &= e^{4\ln 3} e^{j5\ln 3} \\
 &= e^{4\ln 3} \cos(5\ln 3) + j e^{4\ln 3} \sin(5\ln 3)
 \end{aligned}$$

이 식의 결과와 위 예제의 결과는 동일하다. 이와 같이 꽤 복잡한 복소수 연산도 파이썬의 기본 연산자를 사용하면 쉽게 수행할 수 있다.

내장함수 `complex()`를 이용하여 복소수를 생성할 수도 있는데 인수로 실수부와 허수부를 입력하면 된다.

```

>>> a=complex(2,-3)
>>> a
(2-3j)

```

이 함수는 복소수를 생성하는 데에는 별로 사용하지는 않고, 다른 데이터형(문자열 등)을 복소수로 변환하는데 주로 사용된다.

2장 연습문제

2-1. 다음 중 식별자가 아닌 것을 모두 골라라.

arduino _123 300j your_name __new__ @gmail object.method

2-2. 2차원 평면의 두 점 (1,2)와 (4,7)의 좌표를 변수들 x1, y1, x2, y2에 각각 저장하라.

2-3. 위의 02번 문제의 두 점의 거리를 변수 dist에 저장하고 화면에 출력하라.

2-4. 1001을 29로 나눈 정수 몫과 나머지를 구하라

2-5. 어떤 학생의 국어, 영어, 수학 점수가 각각 70점, 87점, 16점일 경우 평균 점수를 파이썬으로 구하라.

2-6. 이원일차 연립방정식 $ax + by = n$ 과 $cx + dy = m$ 의 해는 다음과 같다.

$$x = \frac{dn - bm}{ad - bc}, \quad y = \frac{am - cn}{ad - bc}$$

만약 $a = 1, b = 2, c = 3, d = 4, n = 5, m = 6$ 일 경우 x 와 y 를 구하라.

제 3 장 문자열

코딩을 할 때 수치 데이터만큼 자주 사용되는 것이 바로 문자열(string) 데이터이다. 문자열(string)은 문자들의 나열로서 예를 들어 이름, 주소, 메시지 등을 데이터화 할 때 사용되는 자료형이다. 파이썬에서 문자열을 만들 때 큰따옴표(“...”) 혹은 작은따옴표 (‘...’)를 사용한다.

```
'Hello world.'  
"국립목포대학교"  
'12.3'
```

위의 예는 모두 문자열을 나타낸다. 12.3은 그냥 숫자이지만 이것을 따옴표로 묶은 ‘12.3’은 숫자데이터가 아니고 문자열이다. 즉, 문자 ‘1’, 문자 ‘2’, 문자 ‘.’, 문자 ‘3’의 연속된 나열이다. ‘12.3’은 숫자가 아니고 문자열이므로 이것으로 당연히 산술연산을 수행할 수 없다.

다른 몇몇 언어들과는 달리 파이썬에서 단문자형이라는 자료형은 별도로 없다. 또한, 다른 언어(C/C++, JAVA, C# 등)에서는 보통 작은따옴표는 단문자를 표현할 때 사용되지만 파이썬에서는 문자열을 입력할 때 사용한다는 것에 유의하자. 즉, 다음과 같이 단문자로 구성된 것도 문자열이다.

```
'x'  
'1'
```

숫자를 변수에 대입할 수 있는 것과 같이 문자열도 변수에 대입할 수 있다.

```
>>>name='홍길동'   
>>>my_school='국립목포대학교'   
>>>your_school = my_school   
>>>type(name)   
<class 'str'>
```

이렇게 변수에 대입된 문자열은 이후에 다른 변수에 대입된다든지 혹은 연산을 수행한다든지 하는 작업들을 할 수 있다. 그리고 문자열을 담은 변수에 type()함수를 적용하면 'str'이라는 문자열을 반환하는 것을 알 수 있다. 또한, 변수명이 여러 단어들로

이루어진 경우는 밑줄(_)로 이어주는 것이 파이썬 코딩의 관례이며 가독성을 높이는 데 좋다.

- 변수가 여러 단어일 경우 밑줄(_)로 연결한다.

파이썬 문자열은 큰따옴표와 작은따옴표를 사용하여 문자열을 입력하는 두 가지 방법이 있기 때문에 편한 쪽으로 사용하면 되는데 작은따옴표가 shift키를 안 눌러도 되기 때문에 타이핑하기 더 쉽다는 장점이 있다, 그리고 작은따옴표가 포함된 문자열 혹은 큰따옴표가 포함된 문자열을 쉽게 입력할 수도 있다. 다음 예들을 보자.

```
'He said "hi".  
"I'm your father."
```

파이썬 문자열에도 C언어의 printf()함수에서 사용하였던 특수 문자를 사용할 수 있다. 특수문자는 백슬래시(\ 는 여기서는 escape cahracter 라고 한다.)로 시작하며 특수한 용도로 사용된다. 다음 표에 주로 쓰이는 특수 문자를 정리하였다.

<표 3.1> 자주 사용되는 특수문자들

특수문자	기능
\n	줄바꿈
\t	수평탭(tab)
\\	백슬래시 '\' 문자 자체
\'	작은따옴표 문자
\“	큰따옴표 문자

예를 들면 다음과 같다.

```
"He says \"How are you?\""  
"Hi.\nHello."  
'He\'s finished.'
```

단 문자열의 중간에 줄바꿈 기호 '\n' 이 들어가면 가독성이 떨어지므로 파이썬에서는 줄바꿈 기호를 타이핑하지 않고 그대로 입력할 수 있는 방법으로 ““ ... ““ 과 ””

... ''' 를 제공한다.

```
>>> s='''hi
... hello'''
>>> s
'hi\nhello'
>>> print(s)
hi
hello
```

문자열 s에는 사용자가 엔터키로 입력한 줄바꿈 기호가 ‘\n’ 으로 자동으로 치환되었음을 알 수 있다.

만약 문자열 안의 ‘\’ 문자를 이스케이프 문자로 간주하지 않고 단순 문자로 사용하고 싶다면 ‘\\’ 와 같이 입력해도 되지만 한 문자열 안에 이런 경우가 많이 발생한다면 문자열 앞에 r을 붙이면 된다. 문자열 앞에 r이 붙으면 그 문자열 안의 모든 ‘\’ 는 단순 문자로 처리되며 결과 문자열에서는 ‘\’ 문자가 자동으로 ‘\\’ 로 변환된다.

```
>>> r'\user\salesiopark\hello.py'
'\\user\\salesiopark\\hello.py'
```

이 방법은 특히 파일 경로나 정규식(regular express)을 다룰 때 유용하다. 문자열의 길이를 구하고자 할 때는 len()함수를 이용하면 된다.

```
>>> msg = '안녕하세요 Mr. 박?'
>>> len(msg)
12
```

문자열의 길이에는 공백문자까지 모두 포함된다는 것에 유의하자. 위의 예에서 msg변수에 저장된 문자열은 공백문자 두 개를 포함해서 모두 12자인 것을 알 수 있다.

3.1 문자열의 인덱싱(indexing)과 슬라이싱(slicing)

파이썬3의 문자열은 내부적으로 (유니코드) 문자의 나열로 취급되며 문자들 각각에는 번호가 매겨진다. 예를 들어보자.

```
>>> s='Hello world' Enter↵
```

문자열	H	e	l	l	o		w	o	r	l	d
인덱스 (기본 방향)	0	1	2	3	4	5	6	7	8	9	10
인덱스 (역방향)	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

〈표 3.2〉 s문자열의 인덱스

문자열에 포함된 각각의 문자에 매겨진 이 번호를 인덱스(index)라고 한다. 이 예에서 문자열의 길이는 11이고 인덱스는 0부터 시작한다. 인덱스가 1부터 시작하지 않고 0부터 시작함에 주의해야 한다. (다른 프로그래밍 언어에서도 배열의 인덱싱은 보통 0부터 시작한다.)

```
>>> s[0] Enter↵
'H'
>>> s[6] Enter↵
'w'
>>> s[-1] Enter↵
'd'
```

마지막의 s[-1]과 같이 음수는 뒤에서부터 세는 것이다. 따라서 뒤에서 첫 번째 문자인 'd'가 된다.

```
>>> s[-2] Enter↵
'l'
>>> s[-6] Enter↵
' '
```

s[-2]는 s변수에 저장된 문자열의 뒤에서 두 번째 문자를 의미하고 s[-6]은 뒤에서 여섯 번째 문자를 의미한다.(〈표3.2〉 참조)

- s[n] 은 인덱싱(indexing)이라고 한다.
- 인덱싱은 여러 개의 요소 중 특정한 요소를 참조하는 것이다.
- 즉, s[n]은 변수 s의 앞에서부터 n번 요소를 의미한다.

- `s[-n]`은 변수 `s`의 뒤에서부터 `n` 번째 요소를 의미한다.

만일 다음과 같은 문자열에서

```
>>> a="python is the best."
```

맨 첫 단어를 뽑아내고 싶다면 아래와 같이 한다.

```
>>> b=a[0:6]
```

인덱스 '0:6' 이 뜻하는 것은 '0부터 5까지' 이다. 끝 번호 6은 포함하지 않는다는 것에 주의해야 한다. 이렇게 콜론(:)을 이용하여 연속적인 인덱스를 지정하는 것을 슬라이싱(slicing)이라고 한다.

문자열의 마지막까지 지정하려면 끝 번호를 생략하면 된다.

```
>>> c = a[7:] # "is the best." 가 c에 저장된다.
```

반대로 시작 번호가 생략되면 문자열의 처음부터 선택된다.

```
>>> d = a[:8] # "python is" 가 d에 저장된다.
```

그리고 시작 번호와 끝 번호가 모두 생략된다면, 즉 `e=a[:]` 이라고 하면 문자열 전체가 선택이 된다. 즉, `e`에는 `a` 문자열 전체가 저장된다.

슬라이싱에서도 인덱싱과 마찬가지로 음수를 사용할 수 있다.

```
>>> f = a[:-5]
```

결과를 확인해 보기 바란다. 이 경우에도 끝 번호는 포함되지 않으므로 첫 문자부터 -6번 문자까지 뽑아져서 `f`에 저장된다. 예를 들어서 만약 문자열 `h`를 5번째 문자를 기준으로 둘로 나눠서 `hl`, `hr`에 저장하고 싶다면 다음과 같이 하면 될 것이다.

```
>>> h1 = h[:5]
>>> hr = h[5:]
```

이러한 기능을 이용해서 문자열 자체를 바꿀 수는 없다는 것에 주의하자. 즉, 다음과 같이 문자열의 일부분을 바꾸는 것은 불가능하다.

```
>>>a[0] = 'x' #불가능하다
```

이는 문자열은 한 번 내용이 정해지면 내용을 읽는 수는 있지만 변경될 수는 없는 자료형이기 때문이다. 이러한 자료형을 불가역(immutable) 자료형이라고 한다.

또한, 슬라이싱에서 두 번째 콜론 뒤에 증분을 지정할 수 있다.

```
>>> n='국립목포대학교'
>>> m=n[::2]
>>> m
'국목대교'
```

위의 예에서 첫 인덱스와 끝 인덱스는 생략되었으므로 전체 문자열을 대상으로 하되 인덱스를 2씩 증가시키라고 지정한 것이다.

슬라이싱을 정리하면 다음과 같다.

- $a[m:n]$ 은 슬라이싱(slicing)이라고 한다.
- a 변수의 m 번 요소부터 $(n-1)$ 번 요소 전체를 의미한다
- n 과 m 은 음수일 수도 있다.
- $a[:n]$ 은 $a[0:n]$ 과 같다.
- $a[m:]$ 은 m 번째 요소부터 끝까지를 의미한다.
- 증분을 두 번째 콜론 뒤에 $a[m:n:s]$ 와 같이 줄 수 있다.

문자열의 인덱스/슬라이싱은 뒤에 나올 리스트(list)에도 똑같이 적용되고 자주 사용되는 기능이므로 잘 숙지해 두어야 한다.

3.2 문자열의 덧셈과 곱셈

문자열끼리 연결해서 하나의 문자열로 만들 때 덧셈 연산자(+)를 사용한다. 다음 예와 같이 두 개의 문자열을 더하면 문자열이 하나로 합해진 새로운 문자열이 만들어진다.

```
>>> 'My name is '+'Salesio.'
>>> q = 'ipython'
>>> r = ' is useful.'
>>> s = q+r
```

이 예제에서 원래의 문자열 q와 r은 더하기 연산으로 변경되지 않는다. 앞서서도 밝혔듯이 문자열은 한 번 내용이 정해지면 그 다음에는 변경할 수 없다. 다만 연산의 결과로 새로운 문자열이 생성되는 것이다. 만약

```
>>>s = 'Life is long.'
```

에서 long이라는 단어를 short로 바꾸고 싶다면 슬라이싱을 이용해서 다음과 같이 할 수 있다.

```
>>> s = s[:8] + 'short.'
```

이 경우 원래 문자열이 바뀌는 것이 아니라 기존의 문자열은 모두 삭제되고 새로 생성된 문자열이 다시 변수 s에 저장되는 것이다. (사실 변수 s는 문자열에 대한 참조가 저장된다. 이 경우 참조, 즉 주소가 새로 생성된 문자열의 그것으로 바뀌는 것이다.)

문자열끼리 연결하는데 덧셈 연산자를 사용하는 것은 다른 언어에서도 흔하지만 곱셈은 독특하다. 문자열에 정수를 곱하면 그 정수만큼 반복되는 새로운 문자열을 생성한다.

```
>>> 'blah '*5 Enter ↵
'blah blah blah blah blah '
>>> '='*50 Enter ↵
'====='
```

위에서 두 번째 예는 화면에 출력할 때 간단히 구분선을 만드는 용도로 사용된다.

3.3 f-문자열

파이썬 3.6버전 이후부터는 f-문자열을 지원하며 이를 이용하면 문자열 중간에 변수값을 쉽게 끼워넣을 수 있다. 문자열 앞에 'f'라는 접두사를 붙이면 문자열 안에 '{변수}' 는 그 변수의 값으로 통째로 치환된다.

```
>>> name='Salesio'
>>> f'My name is {name}.'
'My name is Salesio.'
```

위의 예에서 문자열 내의 {name}부분은 name변수의 'Salesio'라는 문자열로 대체되었다. 또한 단순히 변수값만을 참조할 수 있는 것이 아니라 수식도 대괄호 {...} 안에 올 수 있다.

```
>>> year=2021
>>> f'금년은 {year}년, 내년은 {year+1}이다.'
'금년은 2021년, 내년은 2022이다.'
```

위의 예에서 보면 {year}는 2021로 대체되었고 {year+1}은 2021에 1이 더해진 2022로 대체되었음을 알 수 있다. 만약 실수를 표시할 때 소수점 둘째 자리까지 표시하고 싶다면 다음과 같이 하면 된다.

```
>>> pi=3.14159265
>>> message=f'pi is {pi:.2f}'
>>> print(message)
pi is 3.14
```

여기서 {pi:.2f}의 의미는 pi 변수값을 소수점 둘째자리까지 표시하는 것이다. 다음 예를 보자.

```
>>> pi=3.14159265
>>> f'pi is [{pi:10.4f}]'
'pi is [ 3.1416]'
```

여기서 {pi:10.4f}의 의미는 소수점 4째자리까지 표시하되 전체 자리수는 10자리로 맞추라는 의미이다. 아래 그림을 보면 쉽게 이해가 갈 것이다. 3.1415가 아니라 3.1416으로 표시된 이유는 다섯째 자리에서 반올림이 되었기 때문이다.

자리수		1	2	3	4	5	6	7	8	9	10	
	[3	.	1	4	1	6]

변수값이 표시될 칸의 수를 미리 지정하고 그 안에서의 정렬방식도 설정할 수 있다.

```
>>> dong='청계면'
>>> f'[{dong:10}]' #10칸 차지 (왼쪽정렬)
'[청계면      ]'
>>> f'[{dong:>10}]' #10칸 우측정렬
'[      청계면]'
>>> f'[{dong:^10}]' #10칸 중앙정렬
'[  청계면  ]'
```

위 예제와 같이 {변수명:n}이라고 지정하면 변수까지 포함해서 총 n칸을 차지하고 정렬은 왼쪽에 하게 한다. {변수:>n}은 n칸의 우측에, {변수:^n}은 n칸위 중앙에 값을 각각 표시한다.

3.4 내장함수 str()

str()함수는 문자열 이외의 데이터를 문자열로 만들어주는 내장함수이다.

```
>>> str(12.34)
'12.34'
>>> str(5+6j)
'(5+6j)'
>>> a=1e-10
>>> b=str(a)
>>> b
'1e-10'
```

이 예와 같이 str()함수는 넘겨받은 데이터를 문자열로 바꾼 것을 반환한다. 반대로 숫자문자로 된 문자열을 정수 데이터로 바꾸고 싶다면 int(), 실수데이터로 만들고 싶다면

면 float(), 복소수로 만들고 싶다면 complex() 함수를 사용하면 된다.
input() 함수는 사용자로부터 입력(타이핑) 받은 내용을 문자열로 반환하는 기능을 한다.

```
>>> ans=input('반지름:') 
반지름:3 
>>> r=float(ans) 
>>> print(f'원의 면적은 {3.14*r**2}이다.') 
원의 면적은 28.26이다.
```

여기서 변수 ans에는 '3'이라는 문자열이 저장된다. 이것을 숫자 데이터로 만들기 위해서 float() 함수를 이용했다. 따라서 변수 r에는 3이라는 숫자가 저장된다. 이 숫자값을 이용하여 $3.14*r^2$ 을 계산한 후 화면에 계산 결과(원의 면적)를 출력하는 간단한 예제이다.

위 예제를 하나의 파일로 만들어 보자. 파이썬셸에서 한 줄씩 실행시키는 것 보다 파일로 작성한 후 실행시키는 것이 더 일반적이다. IDLE에서 편집기를 열어서 다음과 같이 작성한 후 'circle.py'로 저장한다. 적당한 폴더에 파일로 저장한 후에는 **F5**를 눌러서 실행할 수 있다. 전체가 한 번에 실행이 된다.

circle.py

```
ans=input('반지름:')
r=float(ans)
print(f'원의 면적은 {3.14*r**2}이다.')
```

실행결과

```
반지름:3 
원의 면적은 28.26이다.
```

문자열의 길이는 len() 내장 함수를 이용하면 구할 수 있다.

```
>>> a='파이썬은 쉽다'
>>> len(a)
7
```

앞에서 소개된 print(), type(), int(), float() 등과 같이 len()도 즉시 사용할 수 있는 파이썬 내장 함수인데 문자열의 길이를 구하는데 사용된다.

3.5 문자열의 메서드

메서드(method) 혹은 멤버함수(member function)는 뒤에 나올 클래스에 대한 이해가 있어야 의미를 정확하게 파악할 수 있다. 여기에서는 일단 문자열을 저장한 변수나 문자열 자체에 점(.)을 찍고 그 뒤에 이름을 써서 호출할 수 있는 함수 정도로 이해하자.

- 내장함수 - 독자적으로 동작한다.
- 메서드(method) - 데이터(객체)를 매개로 동작한다.

문자열의 메서드를 정리하면 다음 표와 같다. <표 4.1>에서 나열된 것들 말고도 몇 가지 메서드가 더 있지만 사용 빈도가 그리 높지는 않으므로 이것들만이라도 동작을 확실히 알아두면 문자열을 다루는데 큰 어려움이 없을 것이다.

<표 3.5.1> 문자열의 메서드와 기능

메서드	기능
lower()	대문자를 소문자로 바꾼 문자열을 반환
upper()	소문자를 대문자로 바꾼 문자열을 반환
swapcase()	대문자는 소문자로, 소문자는 대문자로 바꾼 문자열을 반환
title()	모든 단어의 첫 문자만 대문자로 나머지는 소문자로 바꾼 문자열을 반환
capitalize()	문자열의 첫 글자만 대문자로 나머지는 소문자로 바꾼 문자열을 반환
islower()	모든 문자가 소문자이면 True 반환
isupper()	모든 문자가 대문자이면 True 반환
isalnum()	모든 문자가 알파벳 혹은 숫자(alphanumeric)이면 True 반환
isalpha()	모든 문자가 알파벳이면 True 반환
isidentifier()	문자열이 식별자의 조건에 맞다면 True 반환
isdecimal()	문자열의 내용이 십진정수이면 True 반환
count(str)	str이 포함된 개수 반환
find(str)	str의 첫 위치를 반환(없다면 -1 반환)
index(str)	str의 첫 위치를 반환(없다면 예외 발생)
join(str)	str을 구성하는 각 문자 사이에 원 문자열을 끼워 넣는다.
lstrip()	좌측 공백 삭제
rstrip()	우측 공백 삭제
strip()	양쪽의 공백 삭제
replace(str1, str2)	원 문자열 안의 str1을 str2로 바꾼다.
split()	공백문자를 기준으로 나누어서 리스트에 저장
split(sep)	sep(문자열)을 기준으로 나누어서 리스트에 저장

3.5.1 lower(), upper() 메서드

전술한 바와 같이 <표 4.1>의 함수들은 모두 문자열 혹은 문자열 변수에 바로 이어서 점(.)을 찍은 다음 호출할 수 있다. 예를 들면 다음과 같다.

```
>>> s='Hello' Enter↵
>>> s2=s.lower() Enter↵
>>> s2 Enter↵
'hello'
>>> s3=s.upper() Enter↵
>>> s3 Enter↵
'HELLO'
>>> s4=s.swapcase() Enter↵
>>> s4 Enter↵
'hELLO'
```

예를 들어 s.lower()는 s라는 문자열에 lower()함수의 기능을 적용한 결과값을 반환한다. lower()는 s변수의 문자열 중 모든 대문자를 소문자로 만든 새로운 문자열을 만들어서 반환시켜주므로 s2변수에는 'hello'라는 문자열이 저장된다. upper()메서드는 모든 소문자를 대문자로, swapcase()는 대문자는 소문자로 그리고 소문자는 대문자로 바꾼 새로운 문자열을 반환한다. 문자열 메서드가 원래의 문자열을 변경시키지 못한다. 즉 s.lower()라고 변수 s를 통해서 lower()라는 메서드를 호출했다고 하더라도 s에 저장된 문자열은 변경되지 않는다. 문자열은 불가역(immutable) 자료형이기 때문이다.

```
>>> s='hello. nice to meet you.' Enter↵
>>> s.title() Enter↵
'Hello. Nice To Meet You.'
>>> s.capitalize() Enter↵
'Hello. nice to meet you.'
```

위 예와 같이 title()메서드는 모든 단어의 첫 자를 대문자로 변경한 새로운 문자열을 만들어 주고, capitalize()메서드는 첫 문자만을 대문자로 변경시킨 새로운 문자열을 반환한다.

3.5.2 is로 시작하는 메서드

is로 시작하는 메서드(혹은 함수)는 진리값을 반환하는 함수인 경우가 많다.

- is..로 시작하는 메서드/함수는 보통 진리값을 반환한다.

<표 4.1>에서 islower()메서드 모든 문자가 알파벳 소문자일 경우 True를 반환하고 isupper()는 모든 문자가 알파벳 대문자일 경우 True를 반환한다.

```
>>> a, b = 'hi', 'Hello'
>>> a.islower()
True
>>> b.islower()
False
>>> b.isupper()
False
>>> 'HELLO'.isupper()
True
```

isalpha()는 모든 문자가 알파벳이면 True를 반환하고, isalnum()은 모든 문자가 알파벳 이거나 숫자라면 True를 반환한다. isdecimal()메서드는 문자열을 십진 정수 숫자형으로 변경할 수 있다면 True를, 그렇지 않다면 false를 반환한다. isdecimal()이 True라면 int() 내장함수로 정수로 바꿀 수 있다.

3.5.3 count() 메서드

count()메서드는 주어진 문자열이 변수에 몇 번 포함되어 있는지를 세는 기능을 한다.

```
>>> s='python is easy to learn and easy to code.'
>>> s.count('e')
4
>>> s.count('to')
2
```

3.5.4 find()와 index()메서드

find()메서드는 어느 위치에 주어진 문자열이 처음 나오는 지를 검색해 준다. 만약 주어진 문자열이 없다면 -1을 반환한다. index()메서드의 기능을 find()와 같지만 만약 주어진 문자열이 변수 내에 없다면 오류를 발생시킨다.

```

>>> s='python is easy to learn and easy to code.'
>>> s.find('to')
15
>>> s.find('y')
1
>>> s.find('z')
-1
>>> s.index('to')
15
>>> s.index('hard')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found

```

find()메서드와 index()메서드는 다음과 같은 차이가 있다.

- find()메서드는 찾는 문자열이 없다면 -1을 반환
- 반면, index()메서드는 오류 발생

그냥 쉘에서 find()와 index()메서드는 별 차이가 없어 보이지만, 프로그램 내에서 동작할 경우에 index()메서드에서 오류가 발생한다면 별도의 예외처리를 하지 않는다면 그 지점에서 동작 중이던 프로그램의 실행이 멈추게 된다.

3.5.5 join() 메서드

join() 메서드의 경우 str1.join(str2) 라고 입력하면 str2를 구성하는 각 문자의 사이에 str1을 끼워 넣어서 새로운 문자열을 생성한다. (반대로 짐작하기 쉬우므로 순서에 유의해야 한다.) 다음 예제를 보면 쉽게 이해가 갈 것이다.

```

>>> ','.join('abc')
'a,b,c'
>>> '_and_'.join('JINX')
'J_and_I_and_N_and_X'

```

join() 메서드는 문자열뿐만 아니라 뒤에서 설명할 리스트(list)나 튜플(tuple)도 인수로 사용할 수 있다. join()메서드의 인수로 리스트를 사용하는 예는 다음과 같다.


```
>>> ';'.join(['if','for','while','else']) Enter↵
'if;for;while;else'
>>> ' and '.join(['park','kim','choi']) Enter↵
'park and kim and choi'
```

3.5.6 공백을 제거하는 메서드를

lstrip(), rstrip(), strip() 함수는 좌측, 우측, 양측의 공백을 제거하는 메서드이다.

```
>>> x=' <blah blah blah> ' Enter↵
>>> x2=x.lstrip() Enter↵
>>> x2
'<blah blah blah> ' Enter↵
>>> x3=x.rstrip() Enter↵
>>> x3 Enter↵
' <blah blah blah>'
>>> x4=x.strip() Enter↵
>>> x4 Enter↵
'<blah blah blah>'
```

이 함수들은 문자열 중간에 있는 공백문자는 없애지 않고, 오직 한 편에 있는 공백 문자들 만을 제거한다.

3.5.7 replace()메서드

문자열의 특정 내용을 다른 것으로 교체하고 싶을 때 replace()메서드를 이용하면 된다.

```
>>> a = 'Life is too short.' Enter↵
>>> b = a.replace('short', 'long') Enter↵
>>> b
'Life is too long.'
```

다시 한 번 언급하자면, 문자열의 메서드들은 원래의 문자열 자체를 바꾸지 않는다. 다음 예를 보면

```
>>> a = 'Life is too short.' Enter↵
>>> a.replace('short', 'long') Enter↵ # ❶
'Life is too long.' # ❷
>>> a Enter↵
'Life is too short.' # ❸
```

❶과 같이 `a.replace()`메서드는 ❷와 같은 결과를 반환하지만 여기에서는 그 결과값을 아무 변수에도 저장하지 않았으므로 그냥 소멸된다. 그리고 ❸에서 보듯이 변수 `a`에 저장된 문자열은 변하지 않았다. `replace()`메서드 결과로 `a`변수의 문자열에 대입하고 싶다면 다음과 같이 하면 된다.

```
>>> a = 'Life is too short.' Enter↵
>>> a=a.replace('short', 'long') # ❶
>>> a Enter↵
'Life is too long.' # ❷
```

이 예의 ❶을 보면 `a=a.replace(...)`명령으로 `replace()`메서드의 결과값을 다시 `a`변수에 저장했다. 따라서 이 경우는 `a`변수에 저장된 기존의 문자열은 소멸되고 새로운 것으로 치환된 것이다. 이를 ❷에서 확인할 수 있다.

3.5.8 split()메서드

`split()`메서드는 그 이름에서 알 수 있듯이 문자열을 쪼개서 각각의 파편을 리스트에 담아서 반환한다. 아무 인수도 넘겨주지 않으면 공백문자를 기준으로 문자열을 나눈다.

```
>>> a='Life is too short; you need python.' Enter↵
>>> a.split() Enter↵
['Life', 'is', 'too', 'short;', 'you', 'need', 'python.']
>>> a.split(';') Enter↵ # ❶
['Life is too short', ' you need python.']
```

위의 ❶을 보면 `a.split(';')`은 세미콜론을 기준으로 해서 `a`문자열을 나누라는 것이다. 나뉜 두 부분을 요소로하는 리스트가 반환됨을 알 수 있다.

3장 연습문제

- 3-1. 변수 `pn` 에 전화번호가 문자열로 “010-1234-5678” 와 같이 저장되어 있다고 가정하자. 변수 `a`에 두 번째 번호를 변수 `b`에 세 번째 번호를 각각 저장하라.
- 3-2. 위의 전화번호 문자열 `pn`에서 마지막 번호를 “xxxxx” 로 변경한 새로운 문자열을 변수 `pn2`에 저장하라.
- 3-3. 반지름을 입력받아서 구의 체적을 출력하는 프로그램을 작성하라. 반지름을 r 이라고 하면 구의 체적을 구하는 공식은 $\frac{4}{3}\pi r^3$ 이다.
- 3-4. 이메일 주소 “[abc@some.co.kr](#)” 에서 “@” 왼쪽은 아이디, 오른쪽은 서버의 주소이다. 이메일 주소를 입력받아서 문자 “@” 가 없다면 “잘못된 이메일 형식입니다.” 이라고 출력하고, 있다면 아이디와 서버의 주소를 각각 출력하라. 예를 들어서 다음과 같이 동작해야 한다.

```
>>> 이메일 주소 입력: abc#def.com
잘못된 이메일 형식입니다.
>>> 이메일 주소 입력: abc@def.com
아이디: abc
서버 주소: def.com
```

제 4 장 리스트

리스트(list)는 여러 개의 객체를 하나로 묶는 데이터로써 파이썬으로 코딩하는데 있어서 사용 빈도가 가장 높은 자료형 중 하나이므로 반드시 잘 숙지하고 있어야 한다. 이 장에서는 리스트의 생성하고 다루는 기본적인 내용을 설명하도록 하겠다.

4.1 리스트 생성

리스트를 생성하는데 두 가지 방법이 있는데 그 중 하나는 대괄호 [...]를 이용하여 묶고자 하는 데이터들을 콤마(,)로 구별하여 나열하는 것이다. 이렇게 콤마로 구분된 리스트 내의 개개의 데이터를 요소(element)라고 한다.

- 리스트 (list)
 - 대괄호 [...]안에 데이터들을 콤마로 구분하여 묶은 자료형
- 리스트의 요소(element)
 - 리스트의 콤마로 구분된 개개의 데이터

보통은 하나의 리스트 안에 같은 자료형의 데이터들을 묶어서 사용하지만 리스트의 요소는 서로 다른 데이터형일 수도 있다.

```
>>> a=[11,22,33] Enter↵ #리스트를 생성하여 a에 저장
>>> b=['life','is','short'] Enter↵ #문자열의 리스트
>>> c=[True, 'hi', 33.4, 2-3j] Enter↵
>>> d=[] #empty list를 생성하여 d에 저장
```

이 예제와 같이 리스트 안의 요소는 어떤 자료형도 될 수 있다. 지금까지 소개된 None, 숫자형, 진리값, 문자열, 그리고 앞으로 나올 튜플, 딕셔너리, 함수, 객체 어떤 것이든 리스트의 요소가 될 수 있다. 리스트에는 아무 요소도 존재하지 않을 수도 있는데 이를 empty list 라고 한다. 리스트 안에 리스트가 오는 것도 가능하다.

```
>>> e=[ [11,22], [33,44,55] ] Enter↵
```

위의 예에서 e변수에 담긴 리스트의 0번 요소는 리스트이고 1번 요소도 리스트이다. 이것을 중첩된 리스트(nested list)라고 하고 몇 단계로 중첩되든지 제한은 없다.

리스트를 생성하는 두 번째 방법은 list() 내장함수를 이용하는 것이다.

```
>>> a=list() Enter ↵
>>> a Enter ↵
[]
```

위와 같이 list()함수에 아무런 인수도 넘겨주지 않는 경우 empty list가 생성되어 반환되므로 변수 a에는 empty list가 생성된다. list()함수는 시퀀스(sequence)를 인수로 넘길 수 있는데 시퀀스란 문자열, 리스트, 튜플, range형 등이 있다.

- 시퀀스(sequence)
 - 집합자료형 중에서 요소에 순서가 있는 것들
 - 인덱싱과 슬라이싱이 가능
 - 문자열, 리스트, 튜플, range형 등이 시퀀스에 해당

튜플과 range는 아직 나오지 않았으므로 문자열을 한 번 list()함수에 넘겨보자.

```
>>> a=list('abcd') Enter ↵
>>> a Enter ↵
['a', 'b', 'c', 'd']
>>> len(a)
4
```

위에서 보인 바와 같이 'abcd'문자열의 각각의 단문자를 요소로 갖는 리스트를 생성한다는 것을 알 수 있다. 문자열에서와 마찬가지로 리스트의 요소의 개수를 얻고 싶다면 len()함수를 사용하면 된다.

보통 리스트는 대괄호[..]를 이용하여 생성되며 list() 함수는 특수한 경우에만 사용되는데 이에 대해서는 뒤에서 다시 자세히 설명하도록 하겠다.

4.2 리스트의 인덱싱과 슬라이싱

문자열과 마찬가지로 리스트의 각각의 요소에는 내부적으로 번호가 붙어서 구별하는데 이 번호를 인덱스(index)라고 한다. 첫 번째 요소의 인덱스는 0이고 두 번째 요소는 1이다. 만약 a=[11,22,33,-44]라고 정의된 리스트인 경우 인덱스는 아래의 그래프와 같다.

a=[11,	22,	33,	-44]
순방향 인덱스	0	1	2	3	
역방향 인덱스	-4	-3	-2	-1	

리스트도 문자열과 동일한 인덱싱과 슬라이싱이 가능하다. 문자열에서 설명했던 인덱싱과 슬라이싱을 다시 요약하면 다음과 같다. a변수에는 리스트가 저장되어 있다고 가정하자.

•인덱싱(indexing)

- a[n]은 n번 요소 자체이다.
- 인덱스는 0부터 시작하는 번호이다.
- n은 음수일 수도 있다. (역방향 인덱스)

•슬라이싱(slicing)

- a[m:n]은 m번 요소부터 (n-1)번 요소를 갖는 리스트이다.
- m과 n은 음수일 수도 있다.
- a[:n]은 a[0:n]과 같다.
- a[m:]은 m 번째 요소부터 끝까지의 리스트이다.
- 증분을 두 번째 콜론 뒤에 a[m:n:s]와 같이 줄 수 있다.

문자열의 차이점은 문자열의 인덱싱 결과는 문자열인데, 리스트의 인덱싱 결과는 그 요소 자체가 된다. 문자열의 슬라이싱이 문자열이라는 점과 리스트의 슬라이싱이 리스트라는 점은 유사하다.

리스트의 인덱싱의 예를 들면 다음과 같다.

```
>>> a=[11,-22,33,44,-55.1] Enter↵
>>> b=a[2] Enter↵
>>> b Enter↵
33
>>> c=a[-1] Enter↵
>>> c Enter↵
-55.1
>>> d=a[0]+a[-1] Enter↵
>>> d
-44.1
```

위와 같이 특정 요소를 뽑아내어 다른 변수에 저장하거나 수식의 피연산자로 사용될

수 있다. 그리고 문자열의 경우와는 다르게 리스트 내의 특정 요소를 교체할 수도 있다. 위에서 입력한 a리스트를 가지고 예를 들면 다음과 같다.

```
>>> a[0]=1 Enter↵
>>> a Enter↵
[1, -22, 33, 44, -55.1]
>>> a[1]+=10 Enter↵
>>> a Enter↵
[1, -12, 33, 44, -55.1]
>>> a[2]=a[1]**4 Enter↵
>>> a Enter↵
[1, -12, 20736, 44, -55.1]
>>> a[3]='hello' Enter↵
>>> a Enter↵
[1, -12, 20736, 'hello', -55.1]
```

위에서 a[0]=1은 0번 요소를 숫자 1로 바꾸라는 것이다. 문자열의 경우 특정 요소를 다른 것으로 바꿀 수 없지만 리스트는 그게 가능하다. 그리고 a[3]='hello'는 3번 요소를 문자열 'hello'로 교체하라는 명령이다. 명령을 실행한 후 a리스트를 확인하면 해당 요소가 바뀌어 있음을 알 수 있다.

슬라이싱도 문자열의 경우와 같이 복수 개의 연속된 요소를 지정한다.

```
>>> a=[11,-22,33,44,-55.1] Enter↵
>>> b=a[1:3] Enter↵
>>> b Enter↵
[-22, 33]
>>> c=a[:4] Enter↵
>>> c Enter↵
[11, -22, 33, 44]
>>> d=a[:-1] Enter↵
>>> d Enter↵
[11, -22, 33, 44]
```

a[1:3]은 1번과 2번 요소를 가지는 리스트를 의미한다.(3번 요소는 포함되지 않는다.) 따라서 b변수는 [-22, 33]이 된다. 슬라이싱의 결과는 리스트가 된다는 점은 문자열의 슬라이싱이 문자열이 되는 것과 유사하다. a[:-1]은 0번 요소부터 맨 마지막 직전의 요소까지이므로 [11, -22, 33, 44]가 되는 것이다. 따라서 d변수에 이 리스트가 저장된다.

- 리스트의 슬라이싱은 리스트이다.

a[2]와 a[2:3]의 차이점에 대해서도 숙지해 두어야 한다. a변수에 리스트가 저장된 경우 둘 다 2번 요소 하나만을 지정하는 것이지만 전자는 그 요소 자체를, 후자는 요소가 하나인 리스트를 생성한다.

```
>>> a=['python','is','easy'] Enter↵
>>> b=a[2] Enter↵
>>> c=a[2:3] Enter↵
>>> b Enter↵
'easy'
>>> c Enter↵
['easy']
>>> type(b) Enter↵
<class 'str'>
>>> type(c) Enter↵
<class 'list'>
```

위에서 보면 b변수에는 a의 2번 요소인 문자열이 그대로 대입되고 c변수에는 a의 2번 요소를 요소로 갖는 리스트가 저장됨을 알 수 있다. 즉, 슬라이싱은 요소가 단 한 개라도 리스트이다. 심지어 요소가 0개라도 리스트인데 a[2:2]라고 하면 empty list가 생성된다.

슬라이싱이 대입연산자의 좌변에 올 수도 있는데 처음으로 학습할 때에는 한 번에 이해하기 어려울 수도 있다. 슬라이싱이 대입연산자의 좌변에 올 때에는 지정하는 범위의 요소들을 한꺼번에 교체할 수 있는데 이 때 우변에는 반드시 시퀀스 데이터가 있어야 한다. 시퀀스는 문자열, 리스트, 튜플(tuple), range형 등이 있는데 튜플과 레인지형은 뒤에 나오므로 여기에서는 문자열과 리스트의 예만을 들겠다.

```
>>> a=[11,-22,33,44,-55,1] Enter↵
>>> a[1:4]=[True, False, None] Enter↵ #❶
>>> a Enter↵
[11, True, False, None, -55, 1]
>>> a[1:4]='abc' Enter↵ #❷
>>> a Enter↵
[11, 'a', 'b', 'c', -55, 1]
```


여기서 `a[1:4]` 1번, 2번, 3번 자리인데 그것들을 ❶에서는 `True`, `False`, `None`으로 교체하고 ❷에서는 `'a'`, `'b'`, `'c'`로 교체하는 것이다. 좌변의 요소의 개수와 우변의 개수가 반드시 같을 필요는 없다.

```
>>> a=[11,-22,33,44,-55,1] Enter↵
>>> a[:3]=[0] Enter↵ #❶
>>> a Enter↵
[0, 44, -55, 1]
>>> a[2:]=[100,200,300] Enter↵ #❷
>>> a Enter↵
[0, 44, 100, 200, 300]
```

❶에서는 `a`의 1번, 2번 요소 (두 개)자리에 0 하나를 집어넣는 것이다. 이때 `a[:3]=0`이라고 하면 오류가 발생한다. 좌변이 슬라이싱이면 우변은 반드시 시퀀스 데이터가 와야하기 때문이다. ❷에서는 `a`의 2번, 3번 요소 (두 개) 자리에 100, 200, 300 세 개를 집어 넣은 것이다. 결과를 보면 동작이 이해가 가리라 생각한다.

4.3 리스트의 덧셈과 곱셈

두 리스트를 더하면 두 리스트의 요소들을 병합하여 새로운 리스트가 만들어진다.

```
>>> a=[11,22] Enter↵
>>> b=[33,44,55] Enter↵
>>> c=a+b Enter↵
>>> c Enter↵
[11, 22, 33, 44, 55]
```

그리고 리스트에 정수를 곱하면 그 만큼 반복된 리스트가 만들어 진다.

```
>>> a = ['hi', 'there'] Enter↵
>>> b=a*3 Enter↵
>>> b Enter↵
['hi', 'there', 'hi', 'there', 'hi', 'there']
```

이와 같이 리스트의 덧셈/곱셈은 문자열의 덧셈/곱셈과 동작이 유사하다.

4.4 리스트의 메서드

리스트의 메서드들(method)들을 다음 <표 5.1>에 정리했다.

<표 4.4.1> 리스트 메서드의 종류와 동작

메서드	동작
append(x)	x를 리스트의 마지막 요소로 추가한다.
extend(list)	list의 요소로 원 리스트를 확장한다.
insert(i, x)	x를 i번째 위치로 끼워 넣는다.
remove(x)	x와 같은 첫 번째 요소를 삭제한다.
pop() pop(i)	마지막 요소를 삭제하고 반환한다. i번째 요소를 삭제하고 반환한다.
clear()	모든 요소를 삭제한다.
index(x)	x와 같은 첫 번째 요소의 인덱스를 반환한다.
count(x)	x와 같은 요소들의 개수를 구한다.
sort()	정렬
reverse()	역순으로 배열
copy()	얕은 복사본을 반환한다.

4.4.1 append()와 extend()메서드

append()메서드는 인수로 넘어온 데이터를 리스트의 마지막 요소로 추가시키는 기능을 한다.

```
>>> x=[1,2,3] Enter ↵
>>> x.append(4) Enter ↵
>>> x Enter ↵
[1, 2, 3, 4]
>>> x.append('five') Enter ↵
>>> x Enter ↵
[1, 2, 3, 4, 'five']
>>> x.append([6,7,8]) Enter ↵
>>> x Enter ↵
```

```
[1, 2, 3, 4, 'five', [6, 7, 8]]
```

위와 같이 `append()`메서드로 넘긴 어떠한 데이터라도 원래의 리스트의 맨 마지막 요소로 추가되므로 결국 리스트의 크기는 1이 증가하게 된다.

반면 `extend()`메서드는 시퀀스를 인수로 받아서 그 각각의 요소를 원래의 리스트의 요소로 마지막 위치에 병합시킨다.

```
>>> a=[1,2,3]
>>> a.extend([4,5,6])
>>> a
[1, 2, 3, 4, 5, 6]
>>> a.extend('789')
>>> a
[1, 2, 3, 4, 5, 6, '7', '8', '9']
```

위 예를 보면 `a.extend([4,5,6])`은 넘겨받은 리스트의 요소들 4, 5, 6을 a리스트의 마지막에 하나씩 차례로 추가시켰음을 알 수 있다. `a.extend('789')`는 문자열의 요소 문자들 '7', '8', '9'를 a의 마지막에 차례로 모두 추가시켰다. 특히 다음은 잘 구분해야 한다.

```
>>> a=[1,2,3]
>>> a.append([4,5,6])
>>> a
[1, 2, 3, [4, 5, 6]] #❶
>>> len(a)
4
>>> b=[1,2,3]
>>> b.extend([4,5,6])
>>> b
[1, 2, 3, 4, 5, 6] #❷
>>> len(b)
6
```

위에서 `append()`메서드를 사용한 결과인 ❶과 `extend()`메서드를 사용한 결과인 ❷를 눈여겨보기 바란다.

리스트의 메서드는 문자열의 메서드와 큰 차이가 있는데, 문자열의 메서드는 정해진 변경이 적용된 새로운 문자열을 반환하는 경우가 많았는데, 리스트의 메서드들은

원래의 리스트 자체에 변경을 가한다는 점이다. 예를 들어 s가 문자열이라면 s.upper()는 s의 모든 문자를 대문자로 변경한 새로운 문자열을 반환한다. 하지만 a.append()는 a 리스트 자체를 변경한다. 문자열은 불가역 자료형이고 리스트는 가역 자료형이기 때문이다.

4.4.2 pop()메서드

pop()메서드는 append()메서드와 반대 동작을 하는데 리스트의 맨 마지막 요소를 삭제하고 그것을 반환하는 동작을 한다.

```
>>> a=['first','middle','last']
>>> b=a.pop()
>>> b
'last'
>>> a
['first', 'middle']
>>> a.pop()
'middle'
>>> a
['first']
```

위에서 b=a.pop() 명령을 수행하면 a리스트의 맨 마지막 요소인 'last'는 삭제되고 그것에 변수 b에 저장된다. 그냥 a.pop()라고 하면 a리스트의 맨 마지막 요소가 사라지는 효과가 있다. pop(n)이라고 인덱스를 넘겨주면 그 인덱스의 요소를 삭제한 후 그것을 반환한다. 예를 들어 pop(0)이라고 호출하면 맨 첫 요소를 삭제하고 그것을 반환한다.

4.4.3 clear()와 remove() 메서드

clear()메서드는 모든 요소들을 삭제하고 empty list로 만드는 메서드이다. 그리고 remove(x)는 첫 번째로 나오는 x를 리스트에서 삭제하는 메서드이다.

```
>>> a=[11,True, -1.2, 'hi', 11, 22]
>>> a.remove(True)
>>> a
[11, -1.2, 'hi', 11, 22]
>>> a.remove(11)
>>> a
[-1.2, 'hi', 11, 22]
```

```
>>> a.clear()
>>> a
[]
```

만약 특정 인덱스를 가지는 요소를 삭제하고 싶다면 del 명령어를 사용하면 된다.

```
>>> a=[11,22,33,44,55]
>>> del a[1]
>>> a
[11, 33, 44, 55]
>>> del a[1:3]
>>> a
[11, 55]
```

위와 같이 a리스트의 1번 요소를 삭제하고 싶다면 del a[1] 이라고 하면 되고 1번,2번 요소를 한꺼번에 삭제하고 싶다면 del a[1:2]라고 하면 된다.

4.4.4 insert() 메서드

insert(n,x)메서드는 n번 자리에 x를 집어 넣는 메서드이다.

```
>>> a=[1,2,3]
>>> a.insert(1,101)
>>> a
[1, 101, 2, 3]
>>> a.insert(0,'start')
>>> a
['start', 1, 101, 2, 3]
```

4.4.5 sort()와 reverse() 메서드

sort()메서드는 정렬을 하는 메서드인데 정렬이란 크기 순서대로 나열하는 것을 의미한다.

```
>>> a=[11,-100,1,33,-2.5]
>>> a.sort()
>>> a
[-100, -2.5, 1, 11, 33]
```

위에서 보듯이 a.sort() 메서드를 실행하고 난 후에 a리스트의 요소들이 크기 순서로 재배열되었음을 알 수 있다.

reverse()메서드는 리스트 안의 요소들의 순서를 역순으로 바꾸는 것이다.

```
>>> b=['hi',True, 123, 45, 'park']
>>> b.reverse()
>>> b
['park', 45, 123, True, 'hi']
```

이 결과에서 알 수 있듯이 reverse()메서드를 호출하면 맨 마지막 요소가 맨 처음으로 오고, 맨 첫 요소가 맨 마지막으로 간 것을 알 수 있다.

4.5 스택과 큐 (선택)

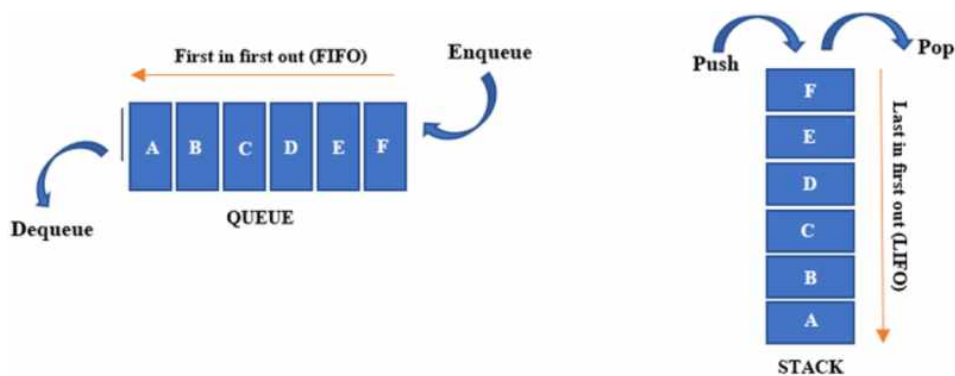
전 절에서 소개한 리스트의 append() 메서드와 pop() 메서드를 이용하면 리스트를 스택(stack)으로 운영할 수 있다. 다음 용례와 같이 리스트의 메서드를 이용하면 stack/queue 와 같은 고급 자료형을 쉽게 구현할 수 있다.

```
>>> a=['first','middle','last']
>>> b=a.pop(0)
>>> b
'first'
>>> a
['middle', 'last']
>>> a.pop(0)
'middle'
>>> a
['last']
```

위에서 a.pop(0)는 a리스트의 맨 첫 요소를 삭제한 후 그것을 반환한다.

전 절에서 나온 append()메서드를 pop()메서드와 같이 이용하면 스택(stack)이라는 자료구조를 쉽게 구현할 수 있다. 스택이란 자료는 순서대로 저장해 놓고 가장 나중에 저장된 자료를 가장 먼저 꺼낼 수 있는 자료구조인데, 저장하는 동작을 푸시(push), 꺼내는 동작을 팝(pop)이라고 한다. 즉, append()메서드가 스택의 푸시역할을 한다.

```
>>> stack=[]
>>> stack.append(3) # push
>>> stack.append(11) # push
>>> stack.pop() #pop
11
>>> stack
[3]
>>> stack.append(-22) #push
>>> stack
[3, -22]
>>> stack.pop() #pop
-22
>>> stack
[3]
```



<그림 4.1> 큐(queue)와 스택(stack)의 동작

위에서 stack이라는 변수에 빈리스트를 할당하여 저장공간을 만든다. 스택에 데이터를 저장할 필요가 있을 때는 stack.append()메서드를 이용하고, 스택에서 데이터를 빼낼 경우에는 stack.pop()메서드를 이용한다. stack.pop()메서드는 항상 가장 마지막에 저장된 자료가 반환된다.

큐(queue)는 스택과 다른 점이 데이터를 빼내 올 때 가장 먼저 저장된 데이터를 가져온다는 것이다. 큐에 데이터를 넣는 것을 인큐(enqueue), 데이터를 빼는 것을 디큐(dequeue)라고 한다. 인큐는 스택과 마찬가지로 append()메서드를 이용하고 디큐는 pop()메서드를 이용하면 된다.

```
>>> queue=[]
>>> queue.append('park')
>>> queue.append('kim')
>>> queue.append('choi')
>>> a=queue.pop(0)
>>> a
'park'
>>> queue
['kim', 'choi']
```

위에서와 같이 queue라는 리스트에 'park', 'kim', 'choi'를 차례로 인큐한 다음 디큐를 하려면 queue.pop(0)메서드를 호출한다. 그러면 가장 먼저 인큐된 데이터를 얻을 수 있다.

4장 연습문제

4-1. 자신의 성(문자열), 이름(문자열), 생년(숫자), 생월(숫자)를 요소로 갖는 리스트를 생성하여 my_info 변수에 저장하는 명령을 작성하라.

4-2. 다음 명령의 결과로 변수 d에 저장되는 값은 무엇인가?

```
>>> c=['kim','gildong','park','cheolsu','choi']
>>> d=c[2]+c[1]
```

4-3. 다음 명령의 결과로 변수 b에 저장되는 값은 무엇인가?

```
>>> a=[11,-22,33,-44,55]
>>> a.sort()
>>> b=a[1]
```

제 5 장 비교/논리 연산자와 if명령

5.1 진리값

앞에서도 설명한 바와 같이 파이썬에서 참, 거짓은 True 와 False 라는 키워드를 사용한다. (첫 글자가 대문자라는 것에 주의) 진리값은 자체로 변수의 값으로 사용될 수 있으며 논리 연산의 결과값을 표현하는데 사용된다. 이처럼 True 혹은 False 두 가지 상태만으로도 표시되는 데이터를 부울형(bool type)이라고도 한다.

5.2 비교 연산자

비교연산자는 두 개의 피연산자를 비교하여 참(True)이나 거짓(False)이나를 판별하는 기능을 한다. 산술연산자의 결과값이 어떤 숫자인 것에 비해서, 비교연산자의 결과값은 True 아니면 False이다. 파이썬에는 다음 <표 6.1>에 나열된 비교연산자들이 있다.

<표 5.2.1> 파이썬의 비교 연산자들

연산	동작
<code>x == y</code>	x와 y의 내용이 같다면 True
<code>x != y</code>	x와 y의 내용이 다르다면 True
<code>x > y</code>	x가 y보다 크면 True
<code>x >= y</code>	x가 y보다 크거나 같다면 True
<code>x < y</code>	x가 y보다 작다면 True
<code>x <= y</code>	x가 y보다 작거나 같다면 True
<code>x is y</code> <code>x is not y</code>	x와 y 가 같은 참조이면 True x와 y 가 다른 참조이면 True
<code>x in s</code> <code>x not in s</code>	x가 s(시퀀스)의 요소이면 True x가 s(시퀀스)의 요소가 아니면 True

연산자 `==` 와 `!=` 은 객체의 내용을 직접 비교하여 참/거짓을 판별한다. 객체의 형(type)이 다르면 두 객체는 항상 다르다. 단, 숫자형인 경우 int, float, complex 간에도 실제로 같은 숫자라면 같다. 즉, 1 (int형), 1.0 (float형), 1+0j (complex형), 1.0+0j (complex형), 1.0+0.0j (complex형) 는 모두 같은 숫자로 판별된다.

```
>>> a, b, c = 11, 21, 11.0
>>> a==b
False
>>> a==c
True
>>> b!=c
True
>>> a>b+c
False
>>> b<c*2
True
>>> b<=c+10
True
>>> a=='11'
False
```

문자열끼리도 비교연산을 할 수 있는데 주로 같은가(==) 다른가(!=)를 판별하게 된다. == 은 두 문자열의 내용이 같으면 True를 반환하고, != 은 한 자라도 틀리면 True를 반환한다.

```
>>> a=['kim','choi','park']
>>> my_name='park'
>>> my_name==a[0]
False
>>> my_name==a[1]
False
>>> my_name==a[2]
True
>>> his_name=a[0]
>>> my_name != his_name
True
```

비교연산자들 중 두 문자로 구성된 것들(==, !=, <=, >=)을 타이핑할 때 두 개의 문자를 반드시 붙여써야하고, <= 연산자를 =<와 같이 순서를 바꾸어서 타이핑하면 오류가 발생한다는 것도 유의하자.

5.3 논리연산자

논리연산자는 세 가지가 있으며 조건식들끼리 묶는 역할을 한다. <표 5.3.1>와 같이 파이썬의 논리 연산자는 자연어(and, or, not)를 사용한다.

<표 5.3.1> 파이썬의 논리 연산자들

연산자	기능
x and y	x와 y 둘 다 True 일 때 True 그 외는 False
x or y	x와 y 둘 다 False 일 때 False 그 외는 True
not x	x가 True 일 때 False

예를 들면 다음과 같다.

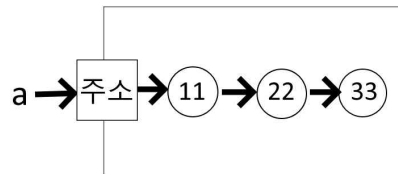
```
>>> a,b=11,-1
>>> c=a>0 and b<0
>>> c
True
>>> d=a>0 or b<0
>>> d
True
>>> e = not a==1
>>> e
True
```

조건식의 결과값(True/False)를 어떤 변수에 대입할 수도 있지만 보통은 조건식과 조합하여 사용한다.

5.4 리스트와 참조(reference)

비교연산자 중 is 연산자는 참조(reference)가 같으면 True를 반환하는데 ==연산자와는 그 동작이 완전히 다르다. is not 연산자는 참조가 다르면 True인데 이것도 != 연산자와는 다르게 동작한다. 이 연산자들의 동작을 이해하려면 참조가 무엇인지 개념을 파악하고 있어야 한다. 먼저 리스트를 가지고 설명하면 다음과 같다. 변수에 숫자를 저장하는 것과 리스트를 저장하는 것은 내부적으로 다르게 작동한다. 예를 들어 변수 a에 리스트 [11,22,33]을 저장한다고 가정하자. 이때 운영체제의 메모리 어딘가에 그 요소들이 모두 저장되어 리스트로 생성되면 그 리스트를 참조할 물리적인 주소(address)가 정해진다. 그리고 변수 a에는 그 주소가 저장되는데 이 주소를 참조(reference)라고 한다. 예를 들어 1번 요소의 리스트를 읽으려고 할 때 먼저 이 참조를 가지고 리스트가 저장된 곳을 뒤져 1번 요소를 찾게 된다. 즉, 숫자형을 변수에 저장

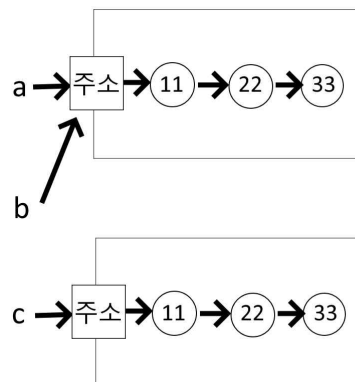
할 때는 그 숫자 자체가 변수에 저장되어 변수 자체가 그 숫자라고 생각해도 된다. 하지만, 리스트의 경우는 변수 자체가 리스트라고 생각하면 안 된다. 변수에는 리스트의 참조(주소)가 저장되는 것이라고 개념적으로 이해해야 된다. 참조는 내부적으로는 꽤 긴 정수로서 리스트마다 다른 숫자가 할당된다.



<그림 5.1> 변수 a의 리스트

이 개념을 알고 있다면 다음을 구별할 수 있다.

```
>>> a=[11,22,33] #❶
>>> b=a
>>> c=[11,22,33] #❷
```



<그림 5.2> 변수 b와 c

위에서 변수 b에는 a를 대입했다. a에는 ❶에서 생성한 리스트의 참조가 저장되어 있다는 사실을 알고 있다면 변수 b에 대입되는 것은 ❶에서 생성한 리스트의 참조라고 유추할 수 있다. 따라서 변수 a와 변수 b는 동일한 리스트를 가리키고 있다. 하지만 ❷에서는 새로운 리스트를 생성한 후 그 참조를 c변수에 저장한 것이다. 따라서 c변수의 리스트가 비록 a변수에 저장된 리스트와 비교해서 내용은 동일하지만 참조는 서로 다르게 된다. 이 참조는 내부적으로는 꽤 긴 정수인데 id()함수로 실제 값을 확인할 수 있다.

```
>>> a=[11,22,33]
>>> b=a
```

```
>>> c=[11,22,33]
>>> id(a)
27734504
>>> id(b) #❶
27734504
>>> id(c) #❷
28925480
```

❶에서 보면 a의 id는 b의 id와 같지만 c의 것과는 다르다는 것을 확인할 수 있다. 리스트끼리의 비교연산자 == 는 두 개의 리스트의 개개의 요소들이 모두 같은 것이라면 True를 반환하는 반면, is 연산자는 두 개의 리스트가 같은 참조(즉, 같은 id)를 가지고 있다면 True를 반환한다.

```
>>> a=[11,22,33] #❶
>>> b=a
>>> c=[11,22,33]
>>> a is b
True
>>> a is c
False
>>> a==b
True
>>> a==c
True
```

위에서 보면 a와 b는 같은 참조를 가지고 있으므로 a==b도 True, a is b도 True이다. a와 c는 내용은 같지만 다른 참조를 가지고 있으므로 a==c는 True, a is c는 False이다.

사실 리스트끼리 비교연산은 자주 사용되지는 않는다. 하지만, 어떤 리스트를 변경할 경우 같은 참조를 가지고 있는 변수 모두에게 영향을 미친다는 것은 반드시 이해해야 한다. 다음 예는 그것을 보여주고 있다.

```
>>> a=[11,22,33]
>>> b=a
>>> a[1]=0 #❶
>>> a
[11, 0, 33]
>>> b
```

```
[11, 0, 33] #②
```

위의 ❶에서 보면 a리스트의 1번 요소를 0으로 교체했으므로 a리스트의 내용이 바뀐 것은 당연한데 ❷에서 확인되듯이 b변수의 리스트도 역시 변경되었음을 알 수 있다. 왜냐면 a와 b는 같은 리스트를 참조하고 있기 때문이다.

리스트안에 다른 리스트가 들어가는 중첩리스트에도 같은 원리가 적용된다. 다음 예를 보면 b리스트의 0번 요소를 a리스트로 지정했다. 그러면 b[0]은 a리스트의 참조가 저장된 것이다. 따라서 a리스트를 변형하면 b[0]가 참조하는 리스트가 변경된 것이기 때문에 b리스트에도 영향을 미친다.

```
>>> a=[11,22]
>>> b=[a,33]
>>> b
[[11, 22], 33]
>>> a[0]=0
>>> a
[0, 22]
>>> b
[[0, 22], 33]
```

하지만 a리스트의 참조 자체가 변경되면 이제는 b[0]와는 다른 참조가 되기 때문에 a가 변형되어도 b에는 영향을 미치지 않는다.

```
>>> a=[11,22]
>>> b=[a,33]
>>> b
[[11, 22], 33]
>>> a=[-1,-2]
>>> a
[-1, -2]
>>> b
[[11, 22], 33]
```

이와 같이, 참조를 가지는 데이터를 하나의 변수를 통해서 변경했다면 같은 참조를 가지는 다른 변수들에도 동일한 효과를 가진다는 것은 반드시 숙지해야 한다.

어떤 리스트와 동일한 요소들을 가지는 별도의 리스트를 만들고 싶다면 copy()메서드를 이용하면 된다.

```

>>> a=['ab','bc','cd']
>>> b=a.copy()
>>> b
['ab', 'bc', 'cd']
>>> a[1]='ef'
>>> a
['ab', 'ef', 'cd']
>>> b
['ab', 'bc', 'cd']
>>> id(a)
32649704
>>> id(b)
33836584
>>>

```

위에서 b는 a리스트와 동일한 요소들을 가지는 새로운 리스트의 참조가 대입된다. 따라서 b의 내용을 보면 a리스트의 내용과 동일한 것을 알 수 있다. 그리고 a리스트와 b리스트는 별도의 리스트이기 때문에 a를 변경시킨다고 b가 변화되지 않는다. a와 b 변수의 참조(id)가 서로 다르기 때문이다.

5.5 if 명령어

if 명령은 그 뒤에 오는 조건식의 참/거짓 여부에 따라 속한 블록을 수행할지 말지를 결정하는 명령이다. 가장 기본 문법은 다음과 같다.

```

if 조건식:
    실행문1
    실행문2
    ...
    실행문n
실행문m

```

여기에서 조건식이 참(True)이면 실행문1, 실행문2 ... 실행문n이 순차적으로 수행된 후 실행문m이 실행된다. 만약 조건식이 False이면 실행문1부터 실행문n까지는 실행되지 않고 실행문m만 실행된다. 주의할 점은 if 문에 속한 모든 실행문은 들여쓰기가 같아야 한다는 것이다. 편의상 if문에 속한 실행문1부터 실행문n까지는 if블럭(block)이

라고 칭하겠다. 같은 블록에 속한 모든 명령어들은 동일한 칸수의 들여쓰기가 되어 있어야 한다. 한 칸이라도 틀리면 실행 시 오류를 발생하게 된다. 파이썬에는 관례적으로 한 수준의 들여쓰기는 공백문자 4칸으로 한다. 또한 조건식 뒤의 콜론(:)도 처음에는 빠뜨리기 쉬우니 조심하자.

- 같은 블록의 명령어들은 들여쓰기 칸 수가 같아야 한다.
- 파이썬에서 들여쓰기는 관례적으로 공백문자 4개이다.

파이썬은 들여쓰기가 문법에 포함된 거의 유일한 언어이다. 처음에 이러한 문법에 익숙치 않다면 tab과 공백문자들을 섞어쓰는 오류를 범하기 쉬운데 주의해야 한다. 탭과 공백문자들이 보기에 같은 크기만큼 들어갔다고 하더라도 서로 다른 들여쓰기로 해석되므로 오류를 발생시킨다.

if 명령의 예를 들면 다음과 같다.

iseven.py
<pre>n = int(input("integer:")) if n%2==0: print(f'{n}을 입력했다.') #❶ print("even number!") #❷</pre>
실행결과
<pre>integer:12 even number!</pre>

이 예는 입력 받은 정수가 짝수일 경우 화면에 메시지를 출력하고 홀수일 경우는 아무런 일도 하지 않는 것이다. ❶과 ❷이 if 블록에 속한 명령어들이고 `n%2==0`이라는 조건식이 참이라면 실행되고 거짓이면 실행되지 않는다. 예를 적당한 폴더에 'iseven.py'로 저장한 후 `Enter` 키를 눌러서 실행하면 파이썬셸에서 실행결과를 확인할 수 있다.

아래와 같이 if 명령은 else 와 짝을 이룰 수도 있다.

```
if 조건식:
    실행문1
    실행문2
    ...
else:
    실행문3
```

실행문4

...

이 경우 조건식이 거짓이면 else 블록(실행문 3, 실행문4...)을 수행하게 된다. else 명령 뒤에 콜론(:)을 빠뜨리면 안되므로 유의하자. 그리고 else블록에 포함된 실행문3, 실행문4 등도 반드시 같은 칸 수 만큼 들여쓰기가 되어야 한다.

myage.py
<pre> age = int(input("나이:")) # ❶ print('당신은') # ❷ if age<30: print("청년") else: print("중년") print('입니다.') # ❸ </pre>
실행결과
<pre> 나이:44 중년 </pre>

위는 사용자의 나이를 입력을 받아서 나이가 30 미만이면 ‘청년’, 30 이상이면 ‘중년’ 이라고 출력하는 간단한 예제이다. 위에서 ❸번 줄은 else블록에 포함되지 않는다는 점을 알아야 한다. 왜냐면 들여쓰기가 되어있지 않기 때문이다. 실행순서는 다음과 같다.

❶ -> ❷ -> if-else 실행 -> ❸

따라서 age변수와 조건과 상관없이 ❸은 반드시 실행이 되는 것이다.

가장 일반적인 if 명령의 구조는 다음과 같다. 키워드 elif 는 else if 를 줄인 단어이다. 아래와 같은 구조에서 조건식1이 참이라면 실행문1을 수행하고 if 블록을 완전히 빠져나간다. 만약 조건문1이 거짓이라면 조건식2를 판별한다. 그래서 조건식2가 참이면 실행문2를 실행하고 if 블록을 빠져나간다. 모든 조건이 거짓일 경우 else 문에 속한 실행문n이 실행된다.

if 조건식1:

```

    실행문1
    ...
elif 조건식2:
    실행문2
    ...
elif 조건식3:
    실행문3
    ...
else:
    실행문n
    ...

```

elif.py

```

s = 'hello'
if 'a' in s: #❶
    print("'a' is contained")
elif 'b' in s:
    print("'b' is contained")
else:
    print("both 'a' and 'b' are not contained")

```

실행결과

both 'a' and 'b' are not contained

위의 ❶에서 in 연산자가 사용되었는데 `x in s` 연산은 `x`가 `s`의 요소로 존재한다면 True를 반환한다. 여기서 `s`는 시퀀스(문자열, 리스트, 튜플, range형 등)여야 한다. 'hello'라는 문자열에는 'a'도 'b'도 포함되어 있지 않으므로 실행결과가 그렇게 나오는 것이다. in 연산자는 조건식에 사용되었다면 다음과 같은 동작을 한다.

- `x in s` : (`s`는 시퀀스) `x`가 `s`에 있다면 True
- `x not in s` : (`s`는 시퀀스) `x`가 `s`에 없다면 True

이와 같이 if-elif-else 명령은 많은 수의 다중 조건을 판단해야 할 경우에 사용된다.

가끔 조건문을 판단하고 참 거짓에 따라 행동을 정의 할 때 아무런 일도 하지 않도록 설정을 하고 싶을 때가 생기게 된다. 다음의 예를 보자.

"집에 돈이 있으면 가만히 있고 집에 돈이 없으면 노트북을 팔아라"

위의 예를 pass를 적용해서 구현해 보자.

```
>>> home = ['money', 'TV', 'radio', 'notebook']
>>> if 'money' in home:
...     pass
... else:
...     print("sell notebook.")
```

home이라는 리스트 안에 'money'라는 문자열이 있기 때문에 if 문 다음 문장인 pass가 수행되었고 아무 일도 수행하지 않는 것을 확인할 수 있다. 아무 것도 수행하지 않으려고 할 때 사용되는 명령어가 pass 이다. 위의 예제는 다음과 동작이 동일하다.

```
>>> home = ['money', 'TV', 'radio', 'notebook']
>>> if 'money' not in home:
...     print("sell notebook.")
```

어떤 block 대신에 pass명령을 사용하는 이유는 실제로 수행해야 할 일이 없는 경우에 사용하기 보다는, 전체적인 논리 구조를 일단 구현하고 세세한 동작을 하는 코드는 나중에 작성할 용도로 사용하는 경우가 많다. 세부적인 동작 없이 프로그램이 생각한 순서대로 바깥에서 동작하는지 먼저 확인한 다음, 나중에 pass 명령어 실제 동작하는 코드로 교체하면 된다.

5.6 단순한 조건 실행문

만약 if, elif, else 명령에 포함된 실행문이 단순하거나 한 줄로 (짧게) 표현된다면 다음과 같이 작성해도 된다.

```
if 조건식: 명령어
elif 조건식: 명령어
else: 명령어
```

즉, 특정 조건에 포함되는 명령어가 여러 줄이면 반드시 내려 써야 하지만, 단순한 경우는 위와 같이 한 줄에 쓸 수도 있다.

```
>>> if a==1: b=22
```

다른 예를 들어보면 다음과 같다.

```
a = 1
if a==0:
    print('a is zero.')
    b=22
else: b=33
```

그리고 여러 줄로 이루어진 명령어 블록을 콜론 바로 뒤에 적는 것도 명령어들을 세미콜로(;)으로 구분한다면 가능하다. 하지만 보통은 콜론(:) 이후에는 한 줄을 내려써서 가독성을 더 높이는 것이 일반적이다.

5.7 후위 if ~ else 명령

아주 단순한 if ~ else 코드를 예를 들어 보자

```
if a == 2: b=22
else: b=11
```

위와 같이 단순한 if~else 구문은 다음과 같은 후위 조건문으로 대체할 수 있다.

```
b = 22 if a == 2 else 11
```

즉, 다음 표현식은 C 조건이 참이면 이것을 A가, 거짓이면 B가 선택(실행)된다.

```
A if C else B
```

이 때 뒤의 else 문은 생략할 수 없다는 점에 유의하자. 즉, 만약 조건식 C가 참일 때만 b 변수의 값을 변하게 하려면 다음과 같이 할 수는 없다는 의미다.

```
b = 22 if a == 2
```

위 명령은 문법 오류가 난다. 다음과 같이 수정해야 한다.

```
if a==2: b=22
혹은
b = 22 if a == 2 else b
```

이렇게 하면 b 변수는 조건이 참일 경우만 22가 지정되고 거짓이면 기존의 값이 유지가 된다. 또 다른 예를 들면 다음과 같다.

```
>>> print('hi') if tom=='here' else print('pass')
```

이와 같이 후위 if~else 명령을 이용하면 코드를 좀 더 간결하게 작성할 수 있다.

5.8 False로 간주되는 것들

파이썬은 False 뿐만 아니라 None, 0 (0.0, 0j, 0.0j, 0+0j 도 포함), 빈 문자열(“ ” 혹은 ‘ ’), 비어 있는 시퀀스 ((), {}, [])도 False 로 간주한다 어떤 데이터가 True로 간주되는 지 혹은 False로 간주되는 지는 bool() 내장함수에 그 데이터를 인수로 넘겨 보면 확인할 수 있다.

```
>>> a=0
>>> bool(a)
False
>>> b=[]
>>> bool(b)
False
>>> c='abc'
>>> bool(c)
True
>>> bool('')
False
```

보통은 if명령어니 elif명령 뒤에 오는 조건식은 True나 False가 엄격하게 발생하도록 코딩하는 것이 더 바람직하다. 즉, 조건식의 결과가 숫자나 문자열 같은 것을 발생시키는 것은 별로 바람직하지 않다.

True는 내부적으로 숫자 1로, False는 숫자 0으로 간주된다. 따라서 True/False를 산술연산에도 이용할 수 있으나 바람직하지는 않다. 단, 주의할 것은 리스트에 숫자

1/0과 True/False가 섞여 있을 때 이를 구분하지 않는다는 점이다.

```
>>> a=[1,0,1,True,False]
>>> a.count(1)
3
>>> a.count(0)
2
>>> a.index(True)
0
```

위와 같이 count()나 index()메서드에서 1과 True는 같은 것으로 (0과 False도 마찬가지다) 취급하므로 의도와는 다른 결과를 반환한다.

5장 연습문제

5-1 생년을 입력받아서 현재 7살 이하면 ‘아동’, 8살 이상, 13살 이하면 ‘어린이’ 14살 이상, 19살 이하면 ‘청소년’, 20살 이상이면 ‘성인’ 이라고 출력하는 프로그램을 작성하라.

5-2 전체 이름을 입력받아서 다음과 같이 출력하는 프로그램을 작성하라. 성이 ‘박’ 씨인 경우에는 출력이 다르다는 것에 유의하라.

```
>>> 이름을 입력하세요:홍길동
홍선생님. 안녕하세요.
>>> 이름을 입력하세요:박철수
박회장님. 환영합니다.
>>> 이름을 입력하세요:김영희
김선생님. 안녕하세요.
>>> 이름을 입력하세요:최민수
최선생님. 안녕하세요.
```

5-3 년도를 입력 받아서 윤년인지 아닌지를 출력하는 프로그램을 작성하라. 윤년을 정하는 기준은 다음과 같다.

(1) 연수가 4로 나누어 떨어지는 해는 윤년

(2) 단, 연수가 100으로 나누어 떨어지지만 400으로 나누어 떨어지지 않는 해는 평년 (예:1900년, 2100년)

예를 들어 다음과 같이 동작해야 한다.

```
>>> 년도를 입력하세요:2020
2020년은 윤년입니다.
>>> 년도를 입력하세요:1900
1900년은 윤년이 아닙니다.
```


제 6 장 튜플, 딕셔너리, 집합

6.1 튜플

튜플(tuple)은 객체들의 묶음이라는 점에서 리스트와 유사하다. 리스트와의 가장 차이점은 튜플은 그 크기나 개별 요소를 변경시킬 수 없다는 점이다. 즉, 한 번 생성되고 나면 변경시킬 수 없다. (불가역 시퀀스) 튜플도 인덱싱과 슬라이싱이 가능하고 I 문자열과 리스트와 마찬가지로 덧셈과 곱셈이 가능하다. 튜플은 정의할 때 대괄호 대신 괄호 (..)를 사용한다.

```
>>> a=(11,22,33)
>>> b=('abc',11,[22,33])
>>> c=((11,22),('hello','world'))
>>> d=(11,) #❶
>>> e=() # empty tuple
```

위와 같이 튜플의 요소는 파이썬의 어떤 데이터라도 올 수 있다. 그리고 변수 e와 같이 요소가 하나도 없는 튜플을 empty tuple이라고 한다. 한가지 주의할 점은 튜플의 요소가 단 하나일 때 ❶과 같이 첫 번째 요소 뒤에 반드시 쉼표(,)를 붙여야 한다는 점이다. 튜플의 특이한 점은 요소가 있는 튜플을 정의할 때 괄호를 생략할 수 있다는 점이다. 아래의 변수 a, b, c, d는 위에서 정의한 튜플과 동일하다.

```
>>> a=11,22,33
>>> b='abc',11,[22,33]
>>> c=(11,22),('hello','world')
>>> d=11, #❶
```

단, 이경우에도 주의할 점은 튜플의 요소가 단 하나일 때는 ❶과 같이 쉼표를 뒤에 붙여야 한다는 점이다. ❶을 언뜻 보면 변수 d에 숫자 11을 대입한 것 같지만 요소가 한 개인 튜플 (11,)이라는 것을 알아야 한다.

이제 변수를 여러 개 동시에 대입하는 예를 다시 한 번 살펴보자.

```
>>> a,b,c = 11, 22, 33
```

사실 이 대입식의 좌변과 우변은 튜플임을 알 수 있다. 즉, 다음과 동일한 동작을 한다. 단지 괄호를 쓰지 않는 것이 더 편리하기 때문에 주로 위와 같이 사용하는 것이다.

```
>>> (a,b,c) = (11, 22, 33)
```

따라서 다음과 같이 정의할 수도 있다.

```
>>> t=11,22,33
>>> a,b,c=t
>>> a
11
>>> b
22
>>> c
33
```

이렇게 명령을 내리면 우변 t의 첫 번째 요소는 a에, 두 번째 요소는 b에 그리고 세 번째 요소는 c에 각각 대입이 되는 것이다.

튜플도 문자열/리스트와 같이 인덱싱, 슬라이싱, 곱셈, 덧셈이 가능하다. 리스트를 슬라이싱하면 새로운 리스트가 생성되듯이, 튜플을 슬라이싱하면 그 결과로 새로운 튜플이 생성된다. 문자열이 요소를 바꿀 수 없는 것과 마찬가지로 튜플도 한 번 정의 되면 그 요소를 바꿀 수 없다.

```
>>> a='park','kim','choi','jang'
>>> b=a[0]
>>> b
'park'
>>> c=a[1:3]
>>> c
('kim', 'choi')
>>> a[3]='lee' # 오류 발생
```

튜플끼리 더하면 각각의 요소들을 통합한 새로운 튜플이 생성되고, 튜플과 정수를 곱하면 그 정수만큼 요소가 반복된 새로운 튜플이 생성된다. 이것은 문자열이나 리스트의 경우와 같다.

```
>>> a,b =(11,22), (33,44)
>>> c=a+b
>>> c
(11, 22, 33, 44)
>>> d=a*3
>>> d
(11, 22, 11, 22, 11, 22)
```

튜플의 전체 요소의 개수는 len() 내장함수를 이용하면 얻을 수 있다. 또한 특정한 요소의 개수를 구하고 싶다면 count()메서드를 이용하면 된다. 그리고 index()메서드는 주어진 요소가 처음으로 위치하는 인덱스를 반환하고 없다면 오류를 발생한다.

```
>>> a=True,False,True,True
>>> len(a)
4
>>> a.count(True)
3
>>> a.index(False)
1
```

튜플의 메서드는 count()와 index() 두 개뿐이다.

6.2 딕셔너리

딕셔너리(dictionary)는 키(key)와 그것에 딸린 값(value)의 쌍들의 집합으로 이루어진 자료형이다. 어떤 사람과 관련된 데이터의 예를 들면 'name'은 '홍길동', 'age'가 22 등이 있을 것이다. 여기서 'name', 'age' 등이 키가 되고 '홍길동'은 'name'이라는 키에 해당되는 값, 22는 'age'라는 키에 해당되는 값이다. 딕셔너리는 대괄호 {...}기호로 생성된다.

```
>>> man1={'name':'홍길동', 'age':22}
```

이렇게 입력하면 man1이라는 딕셔너리가 생성된다. 이와 같이 딕셔너리는 키-값 쌍들의 집합이다. 키와 그 값은 콜론(:)으로 각 쌍은 콤마(,)로 구분한다.

```
{키1:값1, 키2:값2, 키3:값3, ...}
```

키로 쓸 수 있는 객체는 문자열, 숫자, 튜플 등 불가역 자료형이라면 무엇이든지 사용할 수 있다. 반면, 값에 해당하는 것은 어떤 파이썬 객체라도 올 수 있다.

만약 앞의 예에서 man1의 이름에 해당하는 값을 얻고 싶다면 다음과 같이 접근할 수 있다.

```
>>> man1['name']
'홍길동'
```

리스트의 인덱싱과 유사하게 하듯이 이름 뒤에 대괄호를 사용했지만 대괄호 안에는 키가 와야 한다. 즉, man1['name']은 man1 딕셔너리의 'name' 키에 해당하는 값을 반환한다.

딕셔너리를 입력할 때 다음과 같이 여러 줄에 입력하는 것도 가능하다.

```
capital={
    'korea':'seoul',
    'japan':'tokyo',
    'china':'beijing'
}
```

이 예에서 키가 문자열이고 값도 문자열이다. 딕셔너리는 데이터의 저장 ‘순서’라는 개념이 없다. 따라서 내부에 순서(인덱스)는 없으며 오직 ‘키’에 할당된 ‘값’에 접근할 수 있을 뿐이다.

```
color={0:'red',1:'yellow', 2:'white', 3:'black'}
```

위는 키로 정수를 갖는다. 따라서 아래와 같이 키를 가지고 해당하는 값을 접근할 수 있다.

```
>>> color={0:'red',1:'yellow', 2:'white', 3:'black'}
>>> color[1]
'yellow'
```

```
>>> x=color[1]
>>> x
'yellow'
>>> color[3]='grey' #❶
>>> color
{0: 'red', 1: 'yellow', 2: 'white', 3: 'grey'}
>>> color[4]='blue' #❷
>>> color
{0: 'red', 1: 'yellow', 2: 'white', 3: 'grey', 4: 'blue'}
```

전술한 바와 같이 딕셔너리는 인덱스가 없다. 이 예제에서 color[1] 과 같은 용례는 인덱싱이 아니라 1이라는 '키'를 지정해 준 것이므로 혼동하지 않아야 한다. 딕셔너리는 인덱싱도 할 수 없고 슬라이싱도 불가능하다. 키-값들 쌍들에는 순서라는 개념이 없기 때문이다. 그리고 ❶에서 보면 color 딕셔너리의 키3은 이미 존재하므로 기존의 값인 'black'은 'grey'로 교체되었다. 하지만 ❷에서는 color에 4를 키로 갖는 기존의 키-값의 쌍이 없으므로 새로운 키-값 쌍이 추가됨을 알 수 있다.

딕셔너리의 키는 중복이 허용되지 않는다.

```
>>> a={0:'a', 1:'b', 0:'c'}
>>> a
{0: 'c', 1: 'b'}
```

위 예에서 딕셔너리 a에 0이라는 키가 중복으로 지정되었는데 결과를 보면 하나는 무시되었다. 딕셔너리는 키-값 쌍들을 오직 키로만 구별하기 때문에 중복된 키는 허용하지 않는 것이다.

- 딕셔너리의 키(key)는 불가역 자료형만 사용할 수 있다.
- 따라서 문자열, 숫자, 튜플 등이 키로 사용된다.
- 딕셔너리의 키는 중복될 수 없다.

딕셔너리의 키로 리스트는 사용 불가능지만 튜플은 가능하다. 리스트는 가역 자료형이고 튜플은 불가역 자료형이기 때문이다. 아래 예는 키가 (위도, 경도) 인 튜플이고 값이 도시의 이름이다.

```
>>> xys={
```

```
(37.541,126.986):'seoul',
(35.137,129.055):'pusan',
(35.159,126.853):'gwangju'
}
```

값에는 어떠한 파이썬 객체도 올 수 있으며, 딕셔너리 안에 값으로 딕셔너리가 다시 올 수도 있는 등 중첩도 얼마든지 가능하다.

6.3 딕셔너리의 내장 메서드들

딕셔너리에 대해서 사용할 수 있는 메서드는 다음과 같은 것들이 있다. 여기서 d는 딕셔너리 객체를 나타낸다.

<표 6.3.1> 딕셔너리의 내장 메서드

소속 함수	기능
d.keys() d.values() d.items()	키들을 모아서 dict_keys 객체로 반환한다. 값들을 모아서 dict_values 객체로 반환한다. (키,값) 튜플을 모아서 dict_items 객체로 반환한다.
d.clear()	모든 키:값 쌍을 제거하고 빈 딕셔너리로 만든다.
d.get(key)	key에 해당하는 값을 가져온다. d[key]와의 차이점은 해당 키가 없을 경우 에러를 발생시킨다는 것이다. (d[key]는 None을 반환함)
d.update(dict) d.update(**kwargs)	주어진 dict 혹은 kwargs 로 딕셔너리를 확장한다.

여기에서 dict_keys, dict_values, dict_items 객체는 모두 시퀀스이다. 딕셔너리를 list() 내장함수에 넘기면 키를 추출하여 그것들을 요소로 하는 리스트가 만들어진다.

```
>>> dic={11:'a', 22:'b', 'c':'c'}
>>> a=list(dic)
>>> a
[11, 22, 'c']
```

반면 값들로 리스트를 만들고 싶다면 dic.values()메서드를 이용해야 한다.

```
>>> dic={11:'a', 22:'b', 'c':'c'}
>>> b=list(dic.values())
>>> b
['a', 'b', 'c']
```

딕셔너리를 for 반복문의 반복자료형으로 사용할 경우에는 반복변수에 키들이 실려서 수행되게 된다.

```
>>> city={
... 'korea':'seoul',
... 'usa':'washington',
... 'japan':'tokyo'
}
>>> for k in city:print(k)
...
korea
usa
japan
```

반면에, 어떤 딕셔너리의 값들을 for명령어의 시퀀스로 사용하고 싶다면 다음과 같이 values() 메서드를 이용하면 된다.

```
>>> city={
... 'korea':'seoul',
... 'usa':'washington',
... 'japan':'tokyo'
}
>>> for v in city.values(): print(v)
...
seoul
washington
tokyo
```

딕셔너리의 값들로 리스트를 만들고 싶다면 list() 내장함수에 city.values()메서드의 결과값을 넘기면 된다.

```
>>> list(city.values())
['seoul', 'washington', 'tokyo']
```

만약 딕셔너리에 어떤 키가 있는지 조사하려면 in 연산자를 이용한다.

```
>>> 'korea' in city
True
>>> 'china' in city
False
```

값들 중에서 특정 데이터가 포함되어있는지의 여부를 판별하려면 다음과 같이 in 연산자에 city.values() 메서드를 이용해야 한다.

```
>>> 'seoul' in city.values()
True
```

특정 키:값 쌍을 삭제하려면 파이썬 명령어인 del 을 이용하면 된다.

```
>>> del city['japan']
>>> city
{'korea': 'seoul', 'usa': 'washington'}
```

items()메서드는 키-값 쌍을 튜플로 만든 다음 그것들을 요소로 갖는 dict_items형 (시퀀스) 반환한다. 이것을 리스트로 만들려면 list() 내장함수를 거쳐야 한다.

```
>>> list(city.items())
[('korea', 'seoul'), ('usa', 'washington'),
 ('japan', 'tokyo')]
```

clear()메서드는 딕셔너리의 모든 키-값 쌍들을 삭제하고 empty dictionary로 만든다.

```
>>> city.clear()
>>> city
{}
```

update()메서드는 주어진 딕셔너리나 키워드인수로 원래의 딕셔너리를 확장한다.


```

>>> a={'a':11}
>>> a.update(b=22,c='hi') #❶
>>> a
{'a': 11, 'b': 22, 'c': 'hi'}
>>> b={'d':True, 'e':[-1,-2]}
>>> a.update(b) #❷
>>> a
{'a': 11, 'b': 22, 'c': 'hi', 'd': True, 'e': [-1, -2]}

```

update()메서드의 인수로는 ❶과 같이 다수의 키=값 형식을 줄 수도 있고, ❷과 같이 딕셔너리를 통째로 넘길 수도 있다.

6.4 집합 (선택)

집합(set)도 시퀀스 중 하나이다. 즉, 리스트와 유사하게 여러 요소를 보관할 수 있지만 차이점은 요소들의 중복이 허용되지 않는다는 점이다. 집합은 중괄호{...}로 생성한다.

```

>>> dice={1,2,3,4,5,6}
>>> coin={'앞', '뒤'}
>>> yut={'도', '개', '걸', '윷', '모'}

```

만약 요소들중에 중복된 것들이 있다면 하나만 포함된다.

```

>>> s={1,1,2,3,4,4,4,5,6,6,6,6}
>>> s
{1, 2, 3, 4, 5, 6}

```

위와 같이 집합에서 중복된 요소는 하나만 남고 모두 배제된다.

- 집합(set)은 중괄호{...}로 생성한다.
- 집합의 요소는 중복되지 않는다.

또한, 집합에는 가역 자료형(리스트나 딕셔너리)을 요소로 집어넣을 수는 없다. 숫자, 문자열, 튜플 등 불가역 자료형만이 집합의 요소가 될 수 있다.

집합에 대해서 다음과 같은 연산을 수행할 수 있다.

<표 6.4.1> 집합(set)의 연산

연산자	동작
$a \mid b$	합집합
$a - b$	차집합
$a \& b$	교집합
$a \wedge b$	대상차집합 ($a \setminus b - a \& b$)
$a < b$	a 가 b 의 부분집합이면 True
$a \leq b$	a 가 b 의 부분집합이거나 서로 같다면 True
$a > b$	b 가 a 의 부분집합이면 True
$a \geq b$	b 가 a 의 부분집합이거나 서로 같다면 True

집합 간 연산의 용례는 다음과 같다.

```
>>> a|b
{1, 2, 3, 4, 5, 6}
>>> a-b
{1, 2}
>>> b-a
{5, 6}
>>> a&b
{3, 4}
>>> a^b
{1, 2, 5, 6}
>>> c={3,4}
>>> c<a
True
>>> c<b
True
```

집합은 리스트와 같이 내장함수 `len()`, `max()`, `min()`, `sum()` 등을 사용할 수 있고 `for` 반복문의 시퀀스로도 사용할 수 있다. 또한 `set()` 내장함수를 이용하여 문자열, 리스트, 튜플, 딕셔너리(키) 등을 집합으로 변환할 수도 있다. 단, 문자열, 리스트, 튜플 등을 집합으로 변환할 때 중복되는 요소는 하나만 남기고 제거된다.

집합의 메서드는 다음 표와 같다.

<표 6.4.2> 집합(set)의 메서드

메서드 (s는 집합 객체)	설명
s.add(객체)	집합 s의 요소를 한 개 추가
s.update(시퀀스)	집합 s에 시퀀스의 요소들을 추가
s.remove(객체)	집합 s에서 객체를 제거
s.clear()	집합 s의 모든 요소 제거
s.pop()	집합 s의 임의의 요소를 삭제하고 그것을 반환

```
>>> s={11,20,0,-40,1.5}
>>> s.add(100)
>>> s
{0, 1.5, 20, 100, -40, 11}
>>> s.update([200,300])
>>> s
{0, 1.5, 20, 100, -40, 11, 300, 200}
>>> s.remove(1.5)
>>> s
{0, 20, 100, -40, 11, 300, 200}
>>> s.clear()
>>> s
set()
```

이 중 pop()메서드는 리스트와 다르게 동작하는데, 리스트의 요소는 순서(인덱스)가 있어서 항상 마지막 요소가 정해져 있다. 집합의 요소도 생성된 순서는 있겠지만 내부적으로는 순서가 정해져 있지 않다. 때문에 리스트의 pop()메서드는 항상 마지막 요소를 삭제할 수 있지만, 집합의 pop()메서드는 임의의 요소를 삭제한다.

6.5 *tuple, *list, **dict 의 용례 (선택)

파이썬 3.5버전 이상부터 리스트/튜플의 앞에 *가 붙거나 딕셔너리의 앞에 **가 붙으면 unpack 동작을 하게 된다. 이것의 의미는 *(튜플)은 단일 튜플이 아니라 개별 요소들로 취급된다는 의미이다. *[리스트]도 마찬가지이다. 예를 들어서 설명하겠다.

```
>>> a=[11,(22,33),44]
>>> a
[11, (22, 33), 44]
>>> len(a)
3
```

위에서 a리스트의 1번 요소는 튜플이므로 전체 길이는 3이 된다.

```
>>> b=22,33
>>> b=(22,33)
>>> c=[11,*b,44]
>>> c
[11, 22, 33, 44]
>>> len(c)
4
>>> d=[0,*c] #❶
>>> d
[0, 11, 22, 33, 44]
```

하지만 여기에서는 튜플 b앞에 *가 붙아서 단일 튜플이 아니라 그 안의 요소를 unpack하여 b 내부의 개개의 요소가 변수 c의 리스트의 개별 요소가 된다. 이것이 unpack동작이다. ❶은 d=[0] 이후에 d.extend(c) 메서드를 호출한 것과 동일한 동작을 한다.

딕셔너리의 경우에는 **{딕셔너리}처럼 앞에 **가 붙으면 키-값 쌍들이 unpack 된다.

```
>>> a={'name':'gildong'}
>>> a={'이름':'길동'}
>>> b={'성':'홍','나이':26}
>>> b={'성':'홍','나이':26, **a} #❶
>>> b
{'성': '홍', '나이': 26, '이름': '길동'}
```

❶에서 보면 b딕셔너리 안에 **a가 포함되었다. 그러면 a 딕셔너리의 키-값 쌍들이 개개의 요소로 b 딕셔너리에 끼어 들어오게 되는 것이다. 두 개의 딕셔너리의 키-값 쌍들을 하나의 딕셔너리에 포함시키는 것도 다음과 같이 할 수 있다.

```
>>> info1={'성':'홍','이름':'길동'}
>>> info2={'나이':26, '성별':'남'}
>>> info12 = {**info1, **info2}
>>> info12
{'성': '홍', '이름': '길동', '나이': 26, '성별': '남'}
```

함수의 파라미터 앞에 *와 **가 붙는 것은 또 다른 기능을 하는데 이는 함수를 설명하는 데서 자세히 알아보도록 하자.

```
>>> info1={'성':'홍','이름':'길동'}
>>> info2={'나이':26, '성별':'남'}
>>> info12 = {**info1, **info2}
>>> info12
{'성': '홍', '이름': '길동', '나이': 26, '성별': '남'}
```

집합의 요소로 가역자료형은 될 수 없지만, 튜플은 가역자료형이 요소가 될 수 있다.

```
>>> a=[11,22]
>>> b='py','java',a,33
>>> b
('py', 'java', [11, 22], 33)
>>> b[2][1]=44 #❶
>>> b
('py', 'java', [11, 44], 33)
>>> b[2]=[55,66] #❷
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

그리고 ❶과 같이 튜플의 요소로 들어간 리스트를 변경하는 것도 가능하다. 단, ❷와 같이 리스트 자체를 다른 리스트로 교체하는 것은 불가능하다.

6장 연습문제

- 6-1. 양의 정수들 중 10이하의 짝수들을 요소로 갖는 튜플을 작성하여 변수 a에 저장하는 명령을 작성하라.
- 6-2. 튜플을 리스트로 자료형을 변경시키는 내장함수는 list()이다. 즉 a=(11,22,33)일 때 b=list(a)라고 명령하면 b 변수에는 [11,22,33] 리스트가 저장된다. 반대로 리스트를 튜플로 변환시키는 함수는 tuple()이다. 만약 변수 a의 1번 요소만 0으로 변경된 튜플을 변수 a에 다시 저장하고 싶다면 어떻게 해야 할까?
- 6-3. 어떤 PC의 운영체제(OS)는 윈도우10이고, CPU는 인텔(intel)사 제푸이고, 램(RAM)이 4G, 하드디스크(HDD) 용량이 100G이다. 이 정보들로 딕셔너리에 생성한 후 변수 pc1에 저장하라. 또 다른 PC의 운영체제는 macOS이고 CPU는 AMD사 제품, 램이 8G, 하드디스크 용량이 500G이다. 이 정보들로도 딕셔너리를 생성한 후 변수 pc2에 저장하라.

제 7 장 반복문

프로그램을 이용하면 특정한 블록을 반복해서 실행시킬 수 있는데 이때 사용되는 것이 for와 while이다. 조건문과 더불어 반복문은 프로그램의 흐름을 제어하는 명령어로써 사용 빈도가 매우 높다.

7.1 for 명령어와 range형

파이썬의 for 명령어는 해당 블록을 반복해서 실행토록 해준다. 기본적인 문법은 다음과 같다.

```
for 반복변수 in 집합자료형:
    반복실행문
```

반복변수는 집합자료형에서 요소를 하나씩 끄집어 내어 그 값을 저장한 후 반복실행문에서 사용된다. 그리고 집합자료형은 문자열, 리스트, 튜플, 딕셔너리, 집합, range형 등의 자료이다. for 구문과 자주 같이 사용되는 내장함수 중에서 range() 함수가 있는데 이 함수는 range형 데이터를 생성한다. for 반복문과 조합되어서 자주 사용되므로 여기에서 자세히 설명하도록 하겠다.

range() 함수는 세 가지 용법이 있다. 다음에서 m, n, s는 정수이다.

```
range(n) # 0 부터 n-1 까지의 요소를 가지는 range형 반환
range(m,n) # m 부터 n-1 까지의 요소를 가지는 range형 반환
range(m,n,s) # m 부터 n-1 까지의 요소(s는 step)를 가지는 range형 반환
```

파이썬3는 range()함수는 range형을 반환한다. (반면 파이썬 2.x에서는 리스트를 반환한다.) 따라서 파이썬3를 사용할 때 range형을 리스트로 만들기 위해서는 list()함수를 반드시 명시적으로 사용해야 한다. 동일한 간격을 가지는 정수들을 요소로 가지는 리스트를 만들고자 할 때 range()함수와 list()함수를 조합하면 된다.

```
>>> range(5)
range(0,5)
>>> list(range(5))
[0,1,2,3,4]
```

```
>>> list(range(1,11))
[1,2,3,4,5,6,7,8,9,10]
>>> list(range(0,30,5)) # 5가 증분
[0,5,10,15,20,25]
>>> list(range(0,-5,-1)) # -1이 증분
[0,-1,-2,-3,-4]
```

하지만 for문 안에서 range 함수를 사용할 경우 굳이 리스트로 바꿀 필요는 없다. 왜냐하면 for 문에는 시퀀스가 사용되며 range형은 자체로 시퀀스이기 때문이다. 예를 들어서 'hi.' 라는 문자열을 다섯 번 출력하는 프로그램을 작성해 보자.

```
for _ in range(5):
    print('hi.')
```

이 반복문은 변수 _ 가 0,1,2,3,4 값을 가지고 각각 반복을 수행하게 된다. 실제 이 변수가 사용되지는 않으므로 그냥 _ 로 지정하였다. (dummy variable 의 이름은 보통 _ 로 지정한다.)

구구단의 2단을 출력하는 프로그램을 작성해 보자.

gugu2.py

```
for n in range(2,10):
    print(f'2 x {n} = {2*n}')
```

실행결과

```
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
```

여기서 변수 n은 2부터 9까지의 값을 가지고 반복문이 수행된다. range(2,10)은 2부터 9까지의 숫자를 요소로 갖는 range형이기 때문이다.

for 반복문은 중첩해서도 얼마든지 사용할 수 있다. 구구단의 2단부터 9단까지 한꺼번에 출력하려면 다음과 같이 작성하면 될 것이다.

gugu.py

```
for m in range(2,10):
    for n in range(2,10):
        print(f'{m} x {n} = {m*n}')
    print('-'*10)
```

여기에서 print('-' *10)은 '-----' 을 출력한다.

다음 예는 시퀀스로 리스트가 사용된 것이다. 어떤 리스트의 요소들 중 짝수의 개수를 세는 프로그램이다.

counteven.py

```
nums = [11, 44, 21, 55, 101]
count = 0
for n in nums :
    if n%2==0:
        count += 1
print(count)
```

실행결과

1

이 예제는 리스트 nums 의 각 요소가 순서대로 n 변수에 대입되고 반복문이 수행된다. for 반복문의 in 지정어 다음에 딕셔너리가 오는 경우는 딕셔너리의 키만 추출된다.

```
name = {'first':'salesio', 'last':'park'}
for k in name:
    ...print(k) 

first
last
```

이 예제의 출력은 딕셔너리 name의 키값들만 표시된다. 키와 값 두 개를 차례로 취하고 싶으면 딕셔너리의 items() 메서드를 이용하면 된다.

```
>>> num_cases = {'coin': 2, 'dice': 6, 'yut':5}
```

```
>>> for k, v in num_cases.items():
...     print(k, v)
coin 2
dice 6
yut 5
```

위에서 반복변수 k에는 키가, v에는 값이 대입되어 반복문이 수행된다.

만약 리스트의 위치까지 반복 문자로 사용하려면 내장함수 enumerate() 내장함수를 이용하면 된다.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

이 예제에서 변수 i가 리스트 요소의 인덱스를 가지고, v는 리스트의 요소 자체를 가지고 반복문이 수행된다.

7.2 리스트 표현식(list comprehension)

앞 절의 range() 내장함수를 이용하면 일정한 간격을 가지는 연속된 값들을 갖는 리스트를 쉽게 생성할 수 있다.

```
>>> a=list(range(1,11))
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> b=list(range(1,20,3))
>>> b
[1, 4, 7, 10, 13, 16, 19]
```

그런데 파이썬 리스트의 특이한 점은 리스트를 정의할 때 대괄호 안에 for 반복문과 if 조건문을 사용할 수 있다는 것인데, 이것을 리스트 표현식(list comprehension) 이라고 한다. 리스트 표현식을 이용하면 일정한 패턴을 갖는 요소들로 이루어진 리스트를 쉽게 생성할 수 있다. 예를 들어서 2의 거듭제곱 수의 리스트를 만든다고 가정하자.

다음과 같이 for 문을 이용할 수도 있을 것이다.

```
>>> sq=[]
>>> for x in range(1,11):
...     sq.append(2**x)
...
>>> sq
[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

이것과 완전히 동일한 방법이 바로 다음과 같은 리스트 표현식이고 훨씬 더 간결하고 가독성이 높다.

```
>>> sq=[2**x for x in range(1,11)]
>>> sq
[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

일반적으로 리스트 표현식은 다음과 같이 구성된다.

[수식 for 변수 in 집합자료형 (for 반복문 혹은 if 조건문)*]

예를 들면 다음과 같다.

```
>>> a=[x**2 for x in range(2,10) if x%2==1]
>>> a
[9, 25, 49, 81]
>>> b=[x+'자' for x in '홍길동']
>>> b
['홍자', '길자', '동자']
```

리스트 표현식에서 for 반복문은 중첩해서 사용할 수 있다. 예를 들어서 두 리스트 요소 중 서로 다른 것들로만 튜플로 조합하는 리스트를 만드는 방법은 다음과 같다.

```
>>> [(x,y) for x in [1,2,3] for y in [3,1,4] if x!=y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

이것은 다음 프로그램과 동일한 동작을 수행한다.

```

>>> grid=[]
>>> for x in [1,2,3]:
    for y in [3,1,4]:
        if x!=y:
            grid.append((x,y))

>>> grid
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

```

두 가지 경우에서 for 문과 if 문의 순서가 같다는 것을 알 수 있다.

리스트 표현식의 수식이 튜플일 경우 반드시 괄호로 묶어야 하며 괄호를 생략할 수 없다.

```

[(x,x**2) for x in range(1,11)]
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100)]

```

만약 다음과 같이 괄호를 생략하면 에러가 발생한다.

```

>>> [x,x**2 for x in range(1,11)]
SyntaxError: invalid syntax

```

이와 같이 리스트 표현식을 이용하면 일정한 패턴을 갖는 리스트를 손쉽게 생성할 수 있다.

7.3 while 반복문

파이썬의 while 명령은 그 뒤에 오는 조건식이 참이면 해당 실행문들을 반복해서 실행하는 단순한 동작을 수행한다.

```

while 조건식:
    실행문
    ....

```

여기서 조건식이 참이면 실행문을 수행한 후 다시 조건을 검사하여 역시 조건이 참이면 다시 실행문을 수행한다. 즉, 조건문이 거짓이 될 때까지 실행문을 반복 수행하는

것이다.

```
>>> m,p = 1,1
>>> while m<=10:
    p *= m
    m += 1
>>> p
3628800
```

이 예는 10! 을 구하는 프로그램이다. 10!=3628800 이라는 것을 알 수 있다. 다른 예를 들어보자.

```
>>> a=['*'*x for x in range(1,6)]
>>> while a:
    print(a.pop())
*****
****
***
**
*
```

위의 예는 a 리스트가 빈 리스트(즉, 요소가 하나도 없는 리스트)가 될 때까지 반복문을 수행하는 것이다. 전에 설명한 바와 같이 빈 리스트는 거짓으로 해석된다. 위의 예에서 a.pop() 대신 a.pop(0)이라고 하면 어떻게 화면에 출력되는지 실행해서 확인해 보자.

7.4 break 명령과 for~else 구문

파이썬의 for/while 반복문 안에서 break 명령을 사용할 수 있는데 이 명령은 즉시 그것이 포함된 가장 안쪽의 반복문을 빠져나온다. 다음 반복문을 보자.

```
for n in lst:
    if n==0:
```

```
break
print(n)
```

이것은 lst 안의 요소들을 차례대로 프린트하다가 0이 발견되면 바로 반복을 멈추는 프로그램이다. 만약 lst=[1,2,3,0,4,5] 라면 1,2,3만 출력되고 반복문은 종료될 것이다. 만약 반복문의 중첩되어 있다면 가장 안쪽의 반복문만 빠져나온다는 점은 유의해야 한다.

파이썬의 for 반복문은 else 명령과 짝을 이룰 수도 있다.

```
for 반복변수 in 집합자료형:
    수행문들1
else:
    수행문들2
```

else 블록에 포함된 ‘수행문들2’는 for 반복문이 집합자료형의 마지막 요소까지 모두 반복한 경우에만 수행된다. 즉, 모든 반복이 성공적으로 수행된 경우에 한 번 수행된다. 하지만 break문을 만나면 else 구문은 수행되지 않고 for 블록을 완전히 빠져나간다.

```
for n in lst:
    if n==0: break
    print(n)
else:
    print('There are no zeros.') # break를 안 만났다면 실행
```

이 예제의 경우 lst 안에 0이 있다면 break를 만나게 되고, 따라서 else 블록은 수행되지 않고 for 반복문을 빠져나오게 된다. 따라서 for 반복문이 종료된 시점에서 이 종료가 모든 반복을 다 수행한 후의 정상적인 종료인지, 아니면 break 명령에 의한 강제 종료인지에 따라서 수행할 일을 구분할 필요가 있는 경우에 for~else 구문을 사용하면 된다.

7.5 while-else 구문과 break 명령

for문과 마찬가지로 while 문도 else 절이 붙을 수 있다.

```
while 조건식:
    실행문들1
else:
    실행문들2
```

조건식이 거짓으로 판정되어서 ‘실행문들1’ 이 수행되지 않을 때 else 절의 ‘실행문들2’ 가 수행된다. 만약 break 문에 의해서 반복이 끝난다면 for 반복문과 마찬가지로 else절은 수행되지 않고 그 바깥으로 빠져 나가게 된다.

```
n=3
while n>=0:
    m = input("Give a nonzero integer :")
    if int(m)==0: break
    n -= 1
else:
    print('You gave me four nonzero integers.')
```

이 예제는 4개의 0이 아닌 정수를 입력 받으면 else 절이 수행된다.

```
#실행 결과 1
Give a nonzero integer :1
Give a nonzero integer :2
Give a nonzero integer :3
Give a nonzero integer :4
You gave me four nonzero integers.
```

만약 그 전에 0이 입력된다면 else 절이 수행되지 않고 while 반복문을 완전히 종료하게 된다.

```
# 실행 결과 2
Give a nonzero integer :0
```

for 반복문과 마찬가지로 break 문에 의한 반복 종료인지 아니면 조건문이 False 가 되어서 반복문을 종료하는 것인지를 구별하여 다른 실행문을 수행할 경우에 while ~ else 절을 사용하면 된다.

```
inp = input('정수를 입력하시오:')
m=int(inp)
```

```
n=2
while n<=m//2:
    if m%n==0: break
    n += 1
else:
    print(f'{m}은 소수')
```

이 예는 사용자가 입력한 정수가 (prime number)인지 아닌지 판별하는 함수이다. 2부터 $m//2$ 까지 차례로 나누어서 나머지가 한 번이라도 0이 된다면 break 명령에 걸리게 된다. 만약 한 번도 0이 아니어서 반복문이 끝까지 돌았다면 else 절이 수행되어 m이 소수임을 출력한다.

다음과 같이 while 다음의 조건식이 True인 경우를 무한반복문이라고 한다.

```
while True:
    실행문들
```

이 경우 반복문을 종료할 수 있는 방법은 실행문들 안에서 break가 수행되는 것 뿐이다. 무한 반복문은 특수한 경우에 사용되면 실행문들 안에 반드시 특정 조건이 만족되는 break를 실행될 수 있도록 해야 한다.

infloop.py

```
while True:
    n=int(input('Give positive integer:'))
    if n>0:
        break
    print(f'Your input is {n}.')
```

위의 예에서 사용자가 양의 정수를 입력하면 무한반복문을 빠져나오게 된다. 무한반복문에서는 while 뒤에 else-블록이 오는 것이 의미가 없는데, 그 이유는 반복이 정상적으로 종료되지 않고 오직 break문에 의해서만 반복문이 종료될 수 있기 때문이다.

7.6 continue 명령

반복문 안에서 continue 명령은 그 이후의 반복문은 수행하지 않고 즉시로 다음 반복 실행으로 넘어가는 동작을 수행한다.


```
for n in lst:
    if n==0: continue
    print(n)
```

이 예제에서 lst의 요소가 만약 0 이라면 continue 명령을 만나게 되고 그 이후의 print(n)은 수행되지 않고 다음으로 바로 넘어가게 된다.

이 예를 while 문으로 작성하면 다음과 같다.

```
k = 0
while k<len(lst) :
    if lst[k]==0:
        k +=1
        continue
    print(lst[k])
    k +=1
```

이와 같이, continue 명령은 반복문 안에서 사용되며 그것을 둘러싼 가장 안쪽의 반복문의 다음 단계로 즉시 넘어가는 동작을 수행한다.

7.7 제어 블록에서 변수 범위

앞에서 살펴본 바와 같이 if, for, while 문은 동일한 들여쓰기가 적용된 블록을 동반한다. 파이썬은 이 블록 내부에서 사용된 변수들을 실행이 끝난 다음에도 그 내용을 가지고 남아있게 된다. 다음과 같은 간단한 예를 보자.

```
for k in range(10):
    a = k+1
    print(k, a) # 9, 10 이 출력된다.
```

이 for 명령이 수행되는 동안 변수 k가 생성되고 for-블록 안에서 a 변수가 새로 만들어져 사용된다. 여기서 for 반복이 종료된 이후에도 이 변수들은 남아있다는 점을 염두에 두어야 한다. 다른 예를 보자

```
b = 0
for k in range(101):
    b += k
print(b) # 5050
```

이 코드는 1부터 100까지 더한 결과를 변수 b에 저장하는 것으로 잘 수행된다. for-블럭에서 변수 b를 참조할 때 외부에서 0으로 초기화된 변수 b를 참조하기 때문이다. 하지만 외부에서 b를 생성하지 않고 참조하려면 예외가 발생된다.

```
for k in range(101):
    b += k # 예외 발생
```

만약 내부 블럭에서 생성되지 않은 변수라면 외부에서 찾아보기 때문이다. 외부에도 없다면 예외가 발생된다.

즉, 제어 블럭의 경우는 변수 공간이 외부와 분리되지 않는다고 생각하면 된다. 이것은 다른 언어들과는 매우 다른 특성이다. 다른 언어에서는 보통 블럭 안에서 생성된 변수는 그 블럭의 실행이 끝나면 소멸되기 때문이다. 이 예에서는 for-블럭만 예를 들었지만 if-블럭, while-블럭도 마찬가지로 동작한다. 변수의 범위에 대해서는 함수를 설명하는 부분에서도 다시 한 번 다루게 될 것이다.

7장 연습문제

7-1 sum()이라는 내장함수는 시퀀스 요소들의 총합을 구하는 함수이다. 이것을 이용하여 다음과 같이 시험 성적이 저장된 튜플의 평균 점수를 구하는 프로그램을 작성하라.

```
(70, 60, 55, 98, 91, 40, 49, 75, 39, 32)
```

7-2: 1000 이하의 자연수 중 7의 배수의 총합을 출력하는 프로그램을 작성하라.

7-3: 1000이하의 자연수 중 소수(prime number)를 요소로 가지는 리스트를 생성한 후 그것을 출력하는 프로그램을 작성하라.

7-4: 다음과 같이 names라는 리스트에 인명이 저장되어 있다. 이들 중에서 성이 '박'씨인 사람이 몇 명인지 계산하는 프로그램을 작성하라.

```
names=['김유신','홍길동','박혁거세','정중부','서희','이순신','박문수','한석봉','김삿갓','정약용','안중근','윤동주','안창호']
```

7-5: 위의 names라는 리스트 안의 요소를 성은 그대로 놔두고 이름만 모두 '아무개'로 교체하는 프로그램을 작성하라. (names 리스트 자체를 수정한다.)

7-6: 문제 7-4번의 names라는 리스트 안의 이름을 ('성', '이름') 튜플로 바꾸어 names2라는 변수에 리스트로 저장하는 프로그램을 작성하라. 즉, names2에는 다음과 같이 저장되어야 한다.

```
names2=[('김', '유신'), ('홍', '길동'), ... ('안', '창호')]
```

7-7: 7-4에서의 names 리스트를 고려한다. num_sung 이라는 변수에 성을 키(key)로 하고 그 성씨를 가지는 사람이 몇 명인지를 값(value)으로 가지는 딕셔너리를 저장하라.

7-8 사용자가 입력한 두 정수의 최대공약수(greatest common factor)을 구하는 프로그램을 작성하라.

7-9 사용자가 입력한 세 개의 정수들의 최소공배수(least common multiple)을 구하는 프로그램을 작성하라.

7-10 이차 방정식 $x^2 + ax + b = 0$ 의 계수 a, b를 사용자가 입력하면 이 방정식의 근을 출력하는 프로그램을 작성하라.

7-11 f변수에 문자열을 요소로 갖는 리스트가 있다고 가정하자. f리스트 각 요소(문자열)의 길이를 요소로 갖는 리스트를 생성하여 g리스트에 저장하는 명령을 작성하라. 예를 들어서 $f = ['kim' , 'gildong' , 'park' , 'cheolsu' , 'choi']$ 라면 g는 $[3,7,4,7,4]$ 가 저장되어야 한다. (단, 리스트표현식을 이용해야 한다.)

7-12 모든 요소가 숫자들인 리스트에서 100 이상인 숫자들만 골라서 다른 리스트에 저장하는 명령을 작성하라.

7-13 계산식 $2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^{10}$ 의 값을 구하는 프로그램을 작성하라.

제 8 장 함수

8.1 함수의 정의와 호출

함수(function)란 실행문들을 묶어서 하나의 블록으로 만든 후 이름을 붙인 것을 말한다. 보통은 입력을 받아서 정해진 동작을 수행한 후에 출력을 내는 구조로 되어 있다. 프로그램에서는 자주 실행되는 정해진 형태의 처리가 있는데 이와 같은 처리를 언제든지 사용할 수 있는 도구와 같은 형태로 정의해 두고 이들을 필요한 곳에서 실행하여 프로그램 전체를 보다 효율적으로 만드는 데 함수가 사용된다.

파이썬 함수는 다음과 같이 정의된다.

```
def 함수명(매개변수1, 매개변수2, ...):
    함수 본체
```

함수의 정의는 항상 def 키워드로 시작한다. 함수명은 일반적인 식별자를 사용하며 변수명과 마찬가지로 관례적으로 소문자로 시작한다. 그다음에 호출하는 쪽에서 넘겨 받을 인수들을 저장할 매개변수(parameter)들을 함수명 뒤의 괄호(...)안에 콤마로 구별하여 지정해 주고 콜론(:) 뒤에 함수의 본체를 작성해 주면 된다. 함수의 본체는 반드시 def 첫 글자의 시작 위치보다 들여써야 한다.

간단한 예를 들어보자

```
>>> def say_hi():
...     print('Hi')
```

이 함수는 'hi' 라는 문자열을 출력하는 함수이며 함수의 이름은 say_hi()이다. 매개변수는 없으며 반환값도 없다. 이 함수를 실행시키는 것을 '함수를 호출(call)한다'고 한다. 위의 say_hi() 함수를 호출하려면 다음과 같이 하면 된다.

```
>>> say_hi()
Hi
```

단수를 입력받아서 구구단을 출력하는 함수를 예로 들어보자. 함수를 파이썬 셸에서 입력하는 것은 불편한 점이 있다. 따라서 IDLE에서 New File 메뉴를 실행하여 다음과 같은 함수를 편집기에서 작성해 보자.

gugudan.py

```
def gugu(n):
    for x in range(2,10):
        print(f'{n} x {x} = {n*x}')
```

이것을 적당한 폴더에 gugudan.py 라는 파일명으로 저장한 후 **F5**키를 눌러 실행하면 파이썬셸에서 gugu()함수를 호출할 수 있다. 이 함수의 정의부에는 def gugu(n): 이라고 작성되어 있는데, 정의부에 명시되어 있는 변수 n을 매개변수(parameter)라고 한다. 위의 gugu()는 매개변수를 한 개 가진 함수이다. 따라서 이 함수는 호출하는 쪽에서 하나의 인수(argument)를 넘겨줄 수 있다. 예를 들어 파이썬셸에서 다음과 같이 실행할 수 있다.

```
>>> gugu(4)
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
```

위의 예는 gugu()함수의 인수로 4를 넘겨준 것이다. 그러면 gugu()함수의 매개변수 n에 넘겨받은 인수값 4가 매개변수 n에 저장되어 함수 본체가 실행된다. 본 교재에서는 인수와 매개변수를 다음과 같이 구별하여 사용한다.

- **인수(argument)**: 함수를 호출할 때 넘겨주는 값
- **매개변수(parameter)**: 함수 정의부에서 인수를 저장하는 변수

gugu()함수에 8이라는 인수를 넘겨주면 매개변수 n에 8값이 저장되어 함수가 실행되며, 그 결과로 8단이 출력된다,

```
>>> gugu(8)
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
```

```
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
```

함수명은 파이썬 객체이며 다른 객체와 유사하게 대입이 가능하다. 예를 들어 앞에서 정의한 함수 say_hi()와 gugu()를 하나의 리스트로 묶을 수 있다.

```
>>> fn = [say_hi, gugu]
```

여기서 fn 리스트의 첫 번째 요소는 함수 say_hi 이고 두 번째 요소는 gugu 이다. 따라서 다음과 같은 함수 호출이 가능하다.

```
>>> fn[0]()
Hi.
>>> fn[1](9)
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
```

또는 함수를 다른 변수에 대입할 수도 있다.

```
>>> kuku = gugu
```

이제 kuku 는 함수이며 gugu()함수와 같은 함수이다. 따라서 다음과 같이 동일하게 호출할 수도 있다.

```
>>> kuku(7)
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
```

```
7 x 8 = 56
7 x 9 = 63
```

가끔 함수의 본체를 구현하지 않고 꺾쇠기만 작성하는 경우도 있다. 이 경우에는 앞에서 나왔던 `pass` 명령을 이용하면 된다.

```
>>> def nop(): pass
```

이 함수는 호출은 할 수 있으나 아무런 일도 수행하지 않는다.

함수명은 그 기능을 표현하는 영어 단어를 사용하는 것이 무난하다. 함수의 기능을 뜻하는 영어단어의 동사 하나 또는 동사+명사의 조합하면 된다. 두 단어 이상 조합되어 있으면 밑줄(_)로 연결하는 것이 파이썬 코딩의 관례이다. 예를 들면 `remove()`, `create()`, `print_name()`, `get_string()`, `set_value()` 등이다. 변수나 함수를 명명하는 것은 의외로 어려운 일이며 능숙한 프로그래머도 이 명명에 고민을 많이 하는 편이다.

8.2 함수의 표준인수와 반환값

함수의 정의부에는 매개변수가 없을 수도 있고 여러 개가 있을 수도 있다. 반환값(`return value`)이란 함수 내에서 얻어진 계산 결과 중에서 함수를 호출한 쪽으로 전달되는 값을 의미하는데 반환 값의 개수도 매개변수와 마찬가지로 없을 수도 있고 한 개 혹은 그 이상일 수도 있다. 전 절에서 작성한 `say_hi()` 함수는 매개변수도 없고 반환값도 없는 경우이고 `gugu()` 함수는 매개변수가 하나 그리고 반환값은 없는 함수이다.

- 반환값(`return value`) : 함수에서 호출한 곳으로 전달되는 값

함수를 호출하는 쪽에 반환값을 되돌려주기 위해선 `return` 이라는 키워드를 이용하여 그 뒤에 반환할 값을 써 주면 된다. 함수를 호출한 쪽에서는 대입 연산자를 이용하여 반환값을 변수에 저장할 수 있다. 이것을 설명하기 위해서 계승값 `n!`을 계산해서 반환하는 `factorial(n)` 함수를 다음과 같이 작성해보자.

```
fact.py
def factorial(n):
    x=2
    for k in range(3,n+1):
        x*=k
```



```
return x #①
```

이 파일을 적당한 폴더에 저장한 후 **F5**키를 눌러 실행시킨다. 그리고 파이썬셸에서 다음과 같이 함수를 호출할 수 있다.

```
>>> a=factorial(5) #②
>>> a
120
```

fact.py파일의 ①에서 보면 return x 라고 되어 있는데 x변수에 저장된 값을 반환하라는 의미이다. 이 x변수에는 n!값이 계산되어 저장되어 있다. 그리고 호출하는 쪽인 ②에서는 a=factorial(5) 라고 실행하고 있는데 대입연산자의 우변이 함수의 반환값으로 치환된다고 생각하면 이해하기 쉬운 것이다. 변수 a에는 5!값이 저장된다. factorial()함수에 인수로 5를 넘겼고 매개변수 n에 5값이 저장되어 함수가 실행되었기 때문이다.

매개변수가 두 개 이상일 경우에는 함수를 호출하는 쪽에서는 매개변수 순서대로 값을 입력해야 한다.

```
>>> def mod(x,y):
    return x%y

>>> mod(3,2) #매개변수 x에 3이, 매개변수 y에 2가 저장된다.
1
```

이 예는 x를 y로 나눈 나머지를 반환하는 함수인데 인수는 순서대로 정의된 모든 매개변수에 넘겨줘야 한다. 다음과 같은 호출은 오류를 발생시킨다.

```
>>> mod()
>>> mod(3)
>>> mod(3,2,1)
```

즉, mod()함수의 매개변수는 두 개 이므로 인수도 두 개가 정확하게 넘겨져야 한다. 이와 같이, 함수의 정의부에 매개변수의 변수명만 있는 것을 표준 매개변수(standard paramter)라고 하며, 호출하는 쪽에서는 반드시 순서와 개수를 맞추어서 인수를 넘겨주어야 한다. 표준 매개변수에 넘겨주는 값들을 표준 인수(standard argument)라고 한다. 표준 인수의 개수는 반드시 표준 매개변수의 그것과 일치해야 한다.

반환 값이 두 개 이상일 경우에는 return 명령 뒤에 반환 값들을 콤마(,)로 구분해야 한다.

```
>>> def cal(x,y):
...     return x+y, x-y # 하나의 튜플을 반환
...

>>> cal(11,22) # 반환값은 튜플이다.
(33, -11)

>>> a,b=cal(33,44)
>>> a
77
>>> b
-11
```

이 간단한 예제에서 cal() 함수는 두 수의 합과 차 두 개를 반환하는데 사실은 하나의 튜플을 반환하는 것이다. 튜플을 정의할 때 괄호를 생략할 수 있다는 점을 상기하면 이해할 수 있을 것이다. 호출하는 쪽에서는 결과값들로 이루어진 튜플을 받게 된다. 다른 변수로 각각 받으려면 위와 같이 하면 된다.

```
>>> a, b = cal(33,44)
```

앞에서 설명한 바와 마찬가지로 cal(33,44)는 튜플인 (33+44, 33-44)로 치환된다고 생각하면 이해하기 쉽다. 즉, a변수에는 77이, b변수에는 -11이 저장된다.

반환값이 명시적으로 없는 함수의 경우 내부적으로 None이 반환된다. 전 절에서 예로 든 say_hi() 함수는 반환값이 없는 함수인데 내부적으로는 함수의 실행이 끝나고 None이 반환된다.

```
>>> a=say_hi()
Hi
>>> print(a)
None
```

보통은 반환값이 없는 함수에서 반환값을 위와 같이 변수에 저장하지 않는다. 이 예는 반환값이 없는 함수라도 내부적으로는 None을 반환한다는 사실을 알려준다. 그리고 return 문이 실행되면 그 위치가 함수의 어디이든지 간에 그 즉시 함수를 종료시키는 역할을 하기도 한다.

- 함수의 내부에서 return 명령을 만나면 그 즉시 함수는 종료된다.

다음과 같은 count()함수를 보자.

```
count.py

def count(n):
    if type(n) is not int or n<=0: #❶
        return #❷
    else:
        for k in range(n,0,-1):
            print(k)
```

위의 ❷를 보면 ❶조건이 만족되면 return 명령을 실행하는데, 이는 넘어온 인수가 정수형이 아니거나 0이하의 정수라면 return 명령이 실행되어 그 즉시로 함수가 종료된다.

```
>>> count(3.1) #㉠
>>> count('hi') #㉢
>>> count(4) #㉡
4
3
2
1
```

위에서 ㉠, ㉢, ㉡ 중 ❶조건이 참이 되는 것은 ㉡뿐이다. 즉, 함수 내부에서 특정 조건이 만족될 경우에 함수를 종료하고 싶을 때에도 return명령이 사용되며, 함수의 내부에서 return 명령을 만나면 함수는 그 즉시 종료된다.

8.3 매개변수의 기본값

매개변수의 기본값(default value)이란 인수를 받지 못했을 경우 자동으로 매개변수에 저장되는 값을 의미한다. 기본값은 함수를 정의할 때 지정할 수 있다. 다음과 같은 예를 들어보자.

```
>>> def foo(a=10):
        print(f'a={a}')
```

함수의 정의부에서 매개변수를 a=10이라고 지정했는데 10은 매개변수 a의 기본값이

다. 이것의 의미는 매개변수 `a`에 저장될 인수를 넘겨받지 못한 경우 10값을 저장한 후 함수를 실행하고, 인수를 받았다면 기본값은 무시하고 넘겨받은 인수를 매개변수 `a`에 저장하라는 것이다. 즉, 호출하는 쪽에서 `foo()`라고 인수를 주지 않고 호출하면 매개변수 `a`에 자동으로 10이 저장된 후 함수가 실행된다. 따라서, 이 함수는 다음과 같이 두 가지 방법으로 호출할 수 있다.

```
>>> foo() # 매개변수 a에 기본값인 10이 자동으로 저장된다.
a=10
>>> foo(20) # 매개변수 a에 인수 20이 전달된다.
a=20
```

위에서 보듯이 `foo()`와 같이 인수를 주지 않으면 '`a=10`'이 출력되고 `foo(20)`과 같이 인수를 넘기면 기본값은 무시되고 넘겨받은 인수를 매개변수 `a`에 저장하게 된다. 따라서 '`a=20`'이 출력되었다. 기본값이 지정된 매개변수를 기본값 매개변수(default parameter)라고 칭한다. 다른 예를 들어보자.

```
>>> def print_name(name, sung='김'):
    print(f'내 이름은 {sung}{name}이다.')
```

이 함수도 다음과 같이 두 가지 방법으로 호출이 가능하다.

```
>>> print_name('길동') #❶
내 이름은 김길동이다.
>>> print_name('길동', '홍') #❷
내이름은 홍길동이다.
```

기본값 매개변수는 인수를 생략해서 호출할 수 있지만, 표준 매개변수는 반드시 값을 지정해 주어야 한다. 따라서 `print_name()` 함수를 호출할 때 첫 번째 인수는 반드시 넘겨주어 표준 매개변수 `name`에 전달해야 한다. 여기에서 ❶과 같이 `print_name()` 함수의 두 번째 인수를 입력하지 않으면 기본값인 '`김`'이 `sung` 매개변수에 저장된다. 하지만 ❷와 같이 두 번째 인수를 명시적으로 지정하면 그것이 매개변수 `sung`에 저장된다.

한 가지 주의할 점은 위의 `print_name()` 함수와 같이 표준 매개변수와 기본값 매개변수가 혼재할 경우 반드시 기본값 매개변수는 표준 매개변수 다음에 위치해야 한다는 점이다. 즉, 아래와 같이 정의하면 오류가 발생한다.

```
>>> def print_name(sung='Kim', name):
```

기본값 매개변수가 두 개 이상일 때에서도 항상 표준 매개변수 우측에 와야 한다.

```
>>> def add(a, b=0, c=0):
...     return a+b+c
...
>>> add(1) # b와 c에 기본값이 저장된다.
1
>>> add(1,2) # b에 2가, c에 기본값이 저장된다.
3
>>> add(1,2,3) # b에 2, c에 3이 저장된다.
6
```

이 예에서 보듯이 함수를 호출할 때 넘겨진 인수의 순서대로 기본값 매개변수에 값이 저장된다.

8.4 가변개수 매개변수

어떤 경우에는 함수를 호출하는 쪽에서 몇 개의 인수가 넘어오는지 모르는 경우가 있을 수 있다. 이 경우에 사용되는 것이 가변개수 매개변수(variable number parameter)으로써, 함수는 넘겨진 인수들을 묶어서 하나의 튜플로 받는다.

예를 들어 넘어온 인수들 중 짝수의 개수를 구하는 함수를 작성해야 하는데 몇 개의 인수가 넘겨질 지 모른다고 가정하자. 이런 경우의 해법은 가변개수 매개변수를 지정하는 것인데, 함수의 정의부에서 매개변수 앞에 별표(*)를 붙인다. 가변개수 매개변수의 이름은 관례적으로 args를 사용하는데(arguments를 줄인말) 반드시 args로 사용할 필요는 없다.

```
countargeven.py
```

```
def count_even(*args):
    count = 0
    for v in args:
        if v%2==0:
            count +=1
    return count
```

이 파일을 실행하면 파이썬셸에서 다음과 같이 실행할 수 있다.

```
>>> count_even(11,22,33)
1
>>> count_even(1,2,3,4,5,6,7)
3
```

이와 같이, 가변개수 매개변수를 이용하면 함수를 호출할 때 콤마로 구분된 인수를 몇 개라도 입력할 수 있고, 함수 내부에서는 그 인수들을 요소로 갖는 튜플이 매개변수 args에 저장된다.

한 가지 주의할 점은 표준 매개변수와 가변개수 매개변수가 혼재할 경우 반드시 가변개수 매개변수는 표준 매개변수 다음에 와야 한다는 점이다. 또한, 가변개수 매개변수는 함수를 정의할 때 단 한 번만 사용할 수 있다는 점도 알아야 한다.

```
>>> def f1(a, *args): # 정상
>>> def f2(a, *args, c): # 오류 - 가변개수 매개변수 뒤에 표
                        준 매개변수가 올 수 없다.
>>> def f3(*args, *args): #오류 - 가변개수 매개변수는 하나만
                        사용 가능
>>> def f4(a, b, *args) # 정상
```

이러한 사항에 주의해서 함수를 작성해야 한다.

8.5 키워드 인수와 키워드 매개변수

키워드 인수(keyword argument)는 함수를 호출할 때 값뿐만 아니라 매개변수의 이름까지 명시적으로 지정해서 전달하는 것이다. 예를 들어 기본값 매개변수가 세 개인 다음과 같은 간단한 함수가 있다고 가정하자.

```
def add(a=0, b=0, c=0):
    return a+b+c
```

이 함수를 호출하는 방법은 앞서서도 설명했듯이 다음과 같은 것들이 있다.

```
>>> add()
0
>>> add(11) # a에 11이 대입됨
```

```

11
>>> add(11,22) # a에 11, b에 22가 대입됨
33
>>> add(11,22,33) # a에 11, b에 22, c에 33이 대입됨
66

```

위에서 보듯이 인수의 순서대로 기본값 매개변수에 저장될 값들이 결정된다. 좌측에 있는 a만, 혹은 a와 b에만 원하는 값을 대입하는 것은 문제가 없다. 그런데 만약 a와 b는 기본값을 사용하고 c에만 다른 값을 넣어주고 싶을 때는 어떻게 해야 할까? 이런 경우 호출하는 쪽에서 매개변수명을 지정하면 된다.

```

>> add(c=33) # c에만 33이 대입되고 나머지는 기본값 사용
>> add(c=33, b=22) #c에 33, b에 22가 대입, a는 기본값

```

위와 같이 호출하는 쪽에서 매개변수 이름을 지정하는 인수를 키워드 인수(keyword argument)라고 칭한다.

- 키워드 인수 (keyword argument)
 - 함수를 호출하는 쪽에서 매개변수의 이름까지 지정한 인수

여러 개의 키워드 인수를 사용할 경우 함수 정의부에 명시된 매개변수의 순서와 상관 없이 나열할 수 있다. 다른 예를 보자.

```

def foo(a, b=-1, c=-2, d=-3):
    pass

```

이 함수는 표준 매개변수(a) 하나와 세 개의 기본값 매개변수(b, c, d)를 가지고 있다. 이 함수는 다음과 같이 다양한 방식으로 호출될 수 있다.

```

foo(1)
foo(a=1) #표준 매개변수도 키워드 인수로 지정 가능
foo(a=1, b=2)
foo(b=2, a=1)
foo(1, 2, 3)
foo(1, d=4)

```

위의 호출방식의 공통점은 표준 매개변수에는 어떤 식으로든 값을 넘겨준다는 점이

다. 위와 같이, 사용빈도는 낮지만 표준매개변수도 키워드인수로 값을 넘겨줄 수 있다. 하지만 다음과 같이 호출할 수는 없다.

```
foo()           # 표준 인수가 없음
foo(a=1, 2)     # 키워드 인수 뒤에 표준 인수를 주지 못함
foo(1, a=2)     # 하나의 매개변수에 두 개의 인수를 주지 못함
foo(e=6)        # e라는 매개변수가 없음
```

함수를 호출할 때에도 키워드 인수는 표준 인수 우측에 와야 한다. 또한 키워드 인수로 넘겨지는 것은 반드시 함수의 정의에서 매칭되는 매개변수가 있어야 하며 이때 매개변수의 순서는 중요하지 않다.

• 함수를 호출할 때에도 키워드 인수는 표준 인수 다음에 위치해야 한다.

키워드 매개변수(keyword parameter)는 함수의 정의부에 `**kwarg` 와 같은 형태로 정의되고, `kwarg` 매개변수는 키워드 인수들 중에서 표준 인수가 아닌 것들을 딕셔너리로 넘겨 받는다. 관례상 키워드 매개변수의 이름은 `kwarg`로 사용하지만, 반드시 이 이름을 사용해야만 하는 것은 아니다.

```
def foo(**kwarg):
    print(kwarg)
```

이 함수를 호출할 경우에는 반드시 인수의 매개변수 이름까지(즉, 키워드 인수들로만) 명시해 주어야 한다.

```
>>> foo(x=11, y=22)
{'x':11, 'y':22}
```

따라서 `foo()` 함수의 내부에서는 어떤 키워드 인수들이 넘어왔는지 `kwarg` 딕셔너리를 검사하여 파악할 수 있다. 키워드 매개변수 `**kwarg` 는 반드시 표준 매개변수나 기본값 매개변수 뒤에 와야 한다. 또한, 만약 가변개수 매개변수 `*args` 와 혼용하는 경우에는 반드시 이것의 우측에 와야 한다.

```
def f(a, *args, **kwarg):
    print('a=', a)
    print('args=', args)
    print('kwarg=', kwarg)
```


이 함수는 다음과 같이 호출될 수 있다.

```
>>> f(11)
a= 11
args= ()
kwarg= {}
>>> f(11,22,33)
a= 11
args= (22, 33)
kwarg= {}
>>> f(11,22,33,b=44,c=55)
a= 11
args= (22, 33)
kwarg= {'c': 55, 'b': 44}
```

이 예에서 보듯이 반드시 표준 매개변수(a), 가변개수 매개변수(*args), 키워드 매개변수(**kwarg)의 순으로 나열되어야 함을 주의하자.

정리하면 매개변수의 종류는 다음과 같다.

- 표준 매개변수
 - 반드시 인수를 받아야만 하는 매개변수
- 기본값 매개변수
 - 인수가 넘어오지 않았을 때 저장될 값이 정해진 매개변수
- 가변개수 매개변수 (*args)
 - 표준/기본값 인수를 제외한 나머지 인수들 전체를 요소로 가지는 튜플이 저장된 매개변수
- 키워드 매개변수 (**kwarg)
 - 키워드 인수들을 딕셔너리로 묶은 매개변수

이와 같이, 함수에 인수를 전달하는 다양한 방식은 반드시 숙지하고 있어야 한다.

8.6 지역변수와 전역변수

매개변수는 함수 내에서 일반 변수와 같이 사용할 수 있는데 함수 내부에서만 사용할 수 있다. 함수 내부에서 정의한 변수도 함수에서만 사용할 수 있다. 이처럼 정해진 블록에서만 쓸 수 있는 변수를 지역변수(local variable)라고 한다. 파이썬에서는 함

수 블록의 안과 밖을 별개로 취급한다. 함수의 매개변수, 함수 블록 안에서 생성된 변수는 함수 내부에서만 사용 가능한 지역변수가 된다. 함수의 실행이 종료되면 지역변수도 함께 소멸되고 사용할 수가 없다.

```
>>> a=-1 #❶
>>> def foo(b):
...     a=b+1 #❷
...
>>> foo(10) # ❸
>>> a #❹
-1
```

위의 예를 보면 함수 밖인 ❶에서 변수 a가 사용(생성)되었다. 그리고 ❸에서 foo()함수가 실행되면 foo()함수 블록 안의 ❷에서 변수 a가 새로 생성되었다. 이때 ❷에서는 변수 a가 foo()함수의 지역변수로 생성된 것이고 함수 바깥의 ❶에서 생성된 변수 a와는 이름이 같지만 전혀 별개의 것이다. 따라서 ❹에서와 같이 foo()함수가 종료된 다음 a변수 값을 출력하면 -1이 보이는 것이다. 그리고 ❷에서 생성된 지역변수 a는 매개변수 b와 함께 foo()함수가 종료되면 소멸된다. 파이썬에서 지역변수가 정의되는 것은 기본적으로 함수 내부뿐이다. 다른 프로그래밍 언어에서는 if블록이나 for블록 안에서 생성된 변수도 지역변수로 다뤄지는 경우가 있다. 하지만 파이썬의 지역변수 규칙은 매우 간단한데 오직 함수(그리고 뒤에서 나올 메서드)안에서만 지역변수가 만들어진다는 것이다.

한 가지 혼동의 여지가 있는 것은 함수 블록 내에서는 함수 바깥의 변수를 읽을 수는 있다는 것이다. 그리고 정해진 방법을 사용하면 함수 내부에서 함수 바깥의 변수의 값을 변경할 수도 있지만 특별한 이유가 없는 한 그러한 동작은 바람직하지 않다.

8.7 익명함수

익명함수(lambda function)란 이름이 없는 함수를 의미하며 lambda 명령어로 정의한다. 주로 어떤 함수의 인수로 간단한 기능의 함수를 즉석으로 넘겨줄 필요가 있을 때 사용된다. 익명함수를 정의하는 문법은 다음과 같다.

```
lambda 매개변수1,매개변수2, ... : (반환될) 표현식
```

익명함수는 보통 한 줄로 정의되고 return문도 없으며 단지 매개변수들과 반환값들의 관계식으로만 표현된다. 예를 들어 두 수의 합을 반환하는 익명함수는 다음과 같이 정의할 수 있다.

```
>>> add = lambda a, b : a+b
>>> add(1,2)
3
```

이 익명함수는 다음과 같이 일반 함수를 정의하는 것과 유사하다.

```
>>> def add(a, b) :
...     return a+b
>>> add(1,2)
3
```

익명함수 자체가 파이썬 객체이므로 리스트의 요소가 될 수도 있다.

```
>>> f1 = [
...     lambda x,y:x+y,
...     lambda x,y:x*y,
... ]
>>> f1[0](1,2)
3
>>> f1[1](3,4)
12
```

그렇다면 일반 함수와 익명함수와의 차이점은 무엇인가? 익명함수의 기능은 일반 함수보다도 훨씬 제한적이고 익명함수로 할 수 있는 것은 일반 함수로도 모두 할 수 있다. 그렇다면 왜 익명함수를 사용할까? 어떤 경우에는 굳이 번거롭게 일반 함수를 정의할 필요가 없이 간단한 기능만을 구현해도 되는 곳이 있다. 익명함수의 전형적인 사용처가 내장함수 `map()`과 `filter()`이다. `map()`과 `filter()`는 첫 번째 인수로 함수를 두 번째 인수로 시퀀스를 받는다.

```
map(함수, 시퀀스)
filter(함수, 시퀀스)
```

`map()`함수는 시퀀스의 개별 요소에 함수를 적용한 값들을 요소로 하는 시퀀스를 반환한다. 엄밀히 말하면 `iterator`를 반환하는 데 `list()`, `tuple()`등의 함수를 적용하여 리스트나 튜플로 변환할 수 있다.

```
>>> def sq(x):
...     return x**2
...
>>> list( map(sq, [2,3,4,5]) ) #❶
[4, 9, 16, 25]
>>> b=(6,7,8,9)
>>> c=list(map(sq,b))
>>> c
[36, 49, 64, 81]
```

❶에서 보면 [2,3,4,5]의 개별 요소에 sq()함수를 취한 리스트, 즉, [sq(2), sq(3), sq(4), sq(5)]가 결과이다. ❶을 익명함수를 이용하면 다음과 같이 간단히 작성할 수 있다.

```
>>> list( map(lambda x:x**2, [2,3,4,5]) )
[4, 9, 16, 25]
```

반면 filter()함수는 주어진 시퀀스의 개별 요소에 함수 결과값을 구하여 그것이 True 인 것만 취하는 내장함수이다. 예를 들어서 인수가 0보다 크면 True를 반환하는 함수를 생각해 보자.

```
>>> def pos(x):
...     return x>0
>>> list(filter(pos, range(-5,6)))
[1,2,3,4,5]
```

이것을 익명함수를 이용하면 다음과 같이 간단하게 처리할 수 있다.

```
>>> list( filter(lambda x:x>0, range(-5,6)) )
[1,2,3,4,5]
```

이와 같이, 익명함수의 용도는 보다 간결하게 원하는 기능을 수행할 수 있도록 하는 것이다.

8.8 함수의 인수가 가변 자료형일 때

함수의 매개변수가 예를 들어 리스트를 인수로 받는다고 가정해보자. 아래의 foo() 함수는 넘겨받은 리스트의 0번 요소를 -1로 바꾸는 코드밖에 없다.

```
>>> def foo(arr):
...     arr[0]=-1
...
>>> a = [11,22,33]
>>> foo(a)
```

그 다음 위와 같이 foo(a)와 같이 함수를 호출하면 호출된 쪽의 a 리스트 원본에 영향을 미칠까?

```
>>> a
[-1, 22, 33]
```

답은 그렇다이다. 왜냐면 함수 foo()의 매개변수 arr은 인수 a에 저장된 리스트의 참조가 복사되기 때문이다. 따라서 함수 내에서 arr을 통해 리스트를 변경하면 원래의 리스트가 변경이 된다. 같은 참조를 가진 하나의 리스트이기 때문이다. 이것은 아래 예와 같이 arr변수에 a를 대입하고 arr[0]=-1로 변경하면 a리스트도 변경하는 것과 근본적으로 동일한 원리이다. 왜냐면 인수로 리스트를 넘겨주면 매개변수에 그 리스트의 참조가 저장되기 때문이다. 이것은 내부적으로 굉장히 효율적인 방식인데, 왜냐면 시퀀스의 요소들이 굉장히 많은 경우에 그것을 함수로 넘겨주려고 할 때, 그 많은 요소들을 전부 복사해서 넘기지 않고 참조만을 전달하기 때문에 메모리 사용 측면에서 유리한 방식이다.

```
>>> a = [11,22,33]
>>> arr=a
>>> arr[0]=-1
>>> a
[-1, 22, 33]
```

만약 별개의 리스트를 함수로 넘기고 싶다면 복사해서 넘기면 된다. 리스트뿐만 아니라 가역 자료형(딕셔너리, 집합 등)에 대해서도 모두 동일한 원리로 동작한다. 즉, 딕셔너리를 인수로 넘기고 함수에게 그것을 받아 변경시키면 호출된 쪽의 원래 딕셔너리도 동일하게 변경된다.

8.9 기타 사항들 (선택)

튜플이나 리스트의 개별 요소들 모두를 순서대로 함수의 표준 인수로 넘겨주기 위

해서는 *를 붙이면 된다.

```
>>> arg=[11,22]
>>> def add(a,b):
...     return a+b
...
>>> add(*arg) #❶
33
```

위에서 ❶은 add(11,22)와 같이 호출하는 것과 동일하다. 딕셔너리의 키-값 쌍들을 키워드 인수로 넘겨주려는 경우에는 ** 지정자를 이용하면 된다.

```
>>> def mul(x=0,y=0):
...     return x*y
...
>>> d={'x':11, 'y':22}
>>> mul(**d)
242
```

위 예와 같이 키워드 인수의 변수명이 문자열 키로 저장된 딕셔너리를 mul()함수의 인수로 mul(**d)와 같이 넘겨주면 함수를 다음과 같이 호출하는 것과 동일하다.

```
>>> mul(x=11, y=22)
```

이와 같이, 딕셔너리를 어떤 함수의 키워드 인수들로 풀어서 넘겨줄 수도 있다. 단, 이 경우 딕셔너리의 키는 식별자 요건을 만족하는 문자열이어야 한다.

주의할 점은 리스트나 딕셔너리와 같은 가변(mutable) 객체를 기본값 인수로 사용할 때 최초의 호출 시에만 지정된 값으로 초기화되고 이후의 호출에서는 그렇지 않다는 점이다.

```
>>> def f(a, L=[]):
...     L.append(a)
...     return L
...
>>> f(1)
[1]
>>> f(2)
```

```
[1, 2]
>>> f(3)
[1, 2, 3]
```

이 예제를 보면 기본값 인수 L은 최초의 호출에서만 빈 리스트로 초기화되고 그 이후의 호출에서는 그 내용물은 유지된다. 따라서 리스트의 요소가 축적되는 것이다.

후속 호출에도 mutable 객체를 초기화하려면 다음과 같은 방법으로 코딩하면 된다.

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

이 경우 실행 결과는 다음과 같다.

```
>>> f(1)
[1]
>>> f(2)
[2]
>>> f(3)
[3]
```

8장 연습문제

8.1 정수를 하나 받아서 그 수 만큼 ‘hi’를 화면에 출력하는 `repeat_hi(n)` 함수를 작성하라. 단 정수가 음수라면 아무런 출력도 하지 않는다.

8.2 양의 정수를 받아서 그 수에 해당하는 삼각형을 출력하는 함수 `print_triangle(n)` 함수를 작성하라. 예를 들면 다음 표와 같다.

n==3	n==5
*	*
**	**
***	***

8.3 리스트를 받아서 그 요소들 중 음수만 삭제해서 반환하는 함수 `remove_negative(s)`를 작성하라.

8.4 두 개의 양의 정수를 받아서 직사각형을 출력하는 함수 `draw_rectangle(width,height)` 를 작성하라. 단 인수를 하나만 주었을 때는 정사각형이 출력되어야 한다.

width==2 height==3	width==4	width==4 height==2
**	****	****
**	****	****
**	****	

8.5 양의 정수를 받아서 그것이 소수(prime number)라면 `True`를 반환하고 아니면 `False`를 반환하는 `is_prime_number(n)` 함수를 작성하라.

8.6 위에서 작성한 `is_prime_number()` 함수를 이용하여 리스트를 인수로 받아서 그 요소들 중 소수만으로 구성된 리스트를 반환하는 함수 `select_prime(s)`를 작성하라.

8.7 정수들을 받아서 그것들의 최대공약수를 반환하는 함수 `get_gcd()`를 작성하라. 단 인수가 몇 개가 넘어올지는 모른다고 가정한다.(즉, 두 개가 될 수도 있고 네 개가 될 수도 있다.) 만약 인수가 없거나 하나라도 양의 정수가 아니면 `None`을 반환한다. 인수가 딱 하나라면 넘어온 인수를 그대로 반환한다. 두 개 이상인 경우에 최

대공약수를 계산하여 반환한다.

8-8 18. 삼각형의 두 내각을 인수로 받아서 나머지 하나의 내각을 반환하는 함수 `get_3rd_angle()` 함수를 작성하라. 단 넘겨받은 두 내각의 합이 180을 초과한다면 `None`을 반환해야 한다.

8-9 양의 정수를 받아서 그 수에 해당하는 역삼각형을 출력하는 함수 `print_inv_triangle(n)` 함수를 작성하라. 예를 들면 다음 표와 같다.

n==3	n==5
***	*****
**	****
*	***
	**
	*

8-10 입력을 정수 `n`으로 받았을 때, `n` 이하까지의 피보나치 수열을 출력하는 함수를 작성하라. 피보나치 수열은 다음과 같은 순서로 결과값을 리턴한다.

`fib(0)` → 0 리턴

`fib(1)` → 1 리턴

`fib(2)` → `fib(0) + fib(1)` → `0 + 1` → 1 리턴

`fib(3)` → `fib(1) + fib(2)` → `1 + 1` → 2 리턴

`fib(4)` → `fib(2) + fib(3)` → `1 + 2` → 3 리턴

제 9 장 내장함수

내장함수는 `print()`, `type()` 등과 같이 (`import` 하지 않고) 즉시로 사용할 수 있는 함수들을 칭한다. 프로그램 작성자가 어떤 기능을 갖는 함수를 작성하려고 할 때 이미 같은 기능을 가지는 함수나 모듈이 있는지를 먼저 조사해야 한다. 내장함수명은 예약어(keyword)로 간주하여야 하며 사용자의 식별자, 즉, 변수명이나 함수명으로 사용하는 것은 피하여야 한다. 여기에서는 이미 소개된 내장함수들과, 앞에서는 나오지 않았지만 사용빈도가 높은 내장함수들에 대해서 설명하겠다.

9.1 수학 관련 내장함수들

9.1.1 `abs()`함수 - 절대값

`abs()`는 절대값(absolute value)을 구하는 함수이다. 만약 복소수가 인수로 넘어온다면 복소수의 크기를 계산해준다.

```
>>> abs(-1.2)
1.2
>>> abs(3-4j)
5.0
>>>
```

9.1.2 `round()`함수 - 반올림

반올림을 하고자 할 때는 `round()`함수를 사용하면 된다.

```
>>> round(3.14159265)
3
>>> round(3.14159265, 2)
3.14
>>> round(137.1987, -1)
140.0
```

두 번째 인수를 주지 않으면 소수점 첫째 자리에서 반올림하여 정수값을 구한다. 두 번째 인수가 음수이면, 즉, $-n$ ($n > 0$)이면 $10^{(n-1)}$ 자리에서 반올림한다. 위에서

137.1987을 1의 자리에서 반올림하여 140이라는 결과가 구해진 것이다. 사사오입의 원칙을 따르는데 이는 반올림 수가 5인 경우 그 윗수가 짝수이면 내리고 홀수이면 올리는 것이다.

9.1.3 min(), max() 함수

min()함수는 최소값(minimum)을, max()함수는 최대값(maximum)을 구한다. 인수로는 시퀀스 하나 혹은 여러 개의 숫자형을 넘겨주면 된다.

```
>>> min(11,-22,100,-1)
-22
>>> a=[1,2,-3,0]
>>> max(a)
2
>>> max('hello')
'o'
```

9.1.4 all()과 any()

내장함수 all()은 시퀀스 하나를 인수로 넘겨받아서 그것의 모든 요소가 True이면 True를 반환한다. 데이터 중에서 False로 간주되는 것은 False뿐만 아니라 None, 숫자 0 (0.0, 0+0j), 빈 문자열 "", 그리고 비어있는 시퀀스([], {}, empty tuple) 등이다. 반면 any()는 인수로 넘겨받은 시퀀스의 요소가 하나라도 True이면 True를 반환한다.

```
>>> all([1,2,3])
True
>>> all([0,1,2])
False
>>> any([0,0+0j,None,{}])
False
>>> any([None, None, 1, 0])
True
```

만약 빈 시퀀스가 넘어온다면 all()은 True, any()는 False를 반환한다.

```
>>> all([])
True
```

```
>>> any([])
False
```

9.1.5 hex(), oct(), bin()

hex()은 넘겨받은 정수 데이터를 16진수로 변환하여 ‘문자열’로 반환한다. oct(), bin()은 각각 8진수, 2진수로 반환한다. 이 함수들의 인수는 정수여야 하며 반환값은 문자열임에 유의하자.

```
>>> hex(123)
'0x7b'
>>> oct(123)
'0o173'
>>> bin(123)
'0b1111011'
```

위와 같이 16진수는 접두사가 '0x'가 붙고, 8진수는 '0o', 그리고 2진수는 '0b'가 숫자 앞에 붙는다.

9.1.6 sum()

sum()함수는 시퀀스를 받아서 모든 요소의 합을 구하는 함수이다.

```
>>> sum((11,22,33))
66
>>> sum(range(1,101)) #1부터 100까지의 합
5050
>>> sum([1,2+3j,4])
(7+3j)
```

sum함수는 두 번째 인수를 줄 수 있는데 이 경우 첫 번째 시퀀스 요소들의 총합에 두 번째 인수를 더한 값을 반환한다.

9.2 기본 입출력과 관계된 함수들

기본 입출력과 관련된 함수로 대표적인 것인 print()함수와 input()함수가 있다. input()은 사용자 입력을 문자열로 반환하는 함수이고, print()함수는 문자열을 출력하

는 함수이다. 이 두 함수는 이전 장들의 많이 사용되었으므로 예제는 생략한다.

9.2.1 globals(), locals(), vars() 함수

globals()함수는 전역변수(global variable)의 이름을 리스트로 만들어서 반환하는 함수이고, locals()함수는 지역변수(local variable)의 이름을 리스트로 만들어서 반환하는 함수이다. 파이썬셸에서 전역변수와 지역변수를 확인하는 용도로 주로 사용되고 프로그램 내에서는 별로 사용되지 않는다.

vars()함수는 객체의 필드를 딕셔너리로 반환해주는 함수이다. 이것은 다음 장에서 클래스를 설명할 때 좀 더 자세히 알아보도록 하겠다.

9.2.2 eval()과 exec() 함수

eval()함수는 문자열로 된 표현식(expression)을 실행해서 결과값을 얻는 함수이다. 주로 입력받은 문자열로 파이썬 함수를 실행시키거나 클래스의 객체를 생성하고자 할 때 사용된다.

```
>>> x = 1
>>> eval('x+1')
2
```

반면 exec()함수는 파이썬 프로그램 조각을 입력받아서 실행시키는 함수이다. 파이썬 코드를 문자열로 넘겨줄 수도 있고 파일 객체를 넘겨줄 수도 있다.

```
>> a=10
>>> exec('b=a+10')
>>> b
20
```

9.2.3 open()함수 (선택)

open()함수는 존재하는 파일을 열거나 새로 파일을 생성하여 파일 객체를 반환해주는 함수이다.

```
>>> f = open('test.txt') # 존재하는 test.txt 파일을 연
다.
```

첫 번째 인수로 파일 이름(문자열)을 받고 두 번째 인수로 모드(문자열)을 받는다. 위의 예와 같이 두 번째 mode 인수가 생략되면 읽기 모드인 'r'로 설정된다. 모드는

다음 <표 9.2.1>과 같은 것이 있다.

<표 9.2.1> open()의 mode 인수

모드	설명	모드	설명
'w'	쓰기	'wb'	바이너리모드 쓰기
'r'	읽기	'rb'	바이너리모드 읽기
'a'	추가	'ab'	바이너리모드 추가

open()함수에서 얻은 파일객체를 이용하여 파일의 내용을 읽거나, 파일을 생성하여 내용을 집어 넣거나, 혹은 기존 파일에 내용을 추가할 수 있다.

9.3 자료형의 생성과 변환

9.3.1 숫자형 함수들

숫자형과 관련된 함수들은 int(), float(), 그리고 complex()함수가 있다. 이들 함수를 이용하면 문자열을 숫자형으로 변환하거나 정수형을 실수형으로 변환하거나 혹은 복소수를 생성하는 등의 작업을 할 수 있다.

9.3.2 부울함수 bool()

부울함수 bool()은 인수로 넘어온 데이터가 True인지 False인지를 판별한다.

```
>>> bool('abc')
True
>>> bool('')
False
>>> bool(0+0.0j)
False
>>> bool([False]) #❶
True
>>> bool([])
False
>>> bool( () ) # empty tuple
False
```

❶에서 보듯이 리스트는 비어있지만 않으면 (즉, 요소가 하나라도 있으면) True로 간주된다.

9.3.3 시퀀스 생성 함수들

시퀀스를 생성하거나 시퀀스로 변환하는 함수들은 다음 표에 정리하였다.

<표 9.3.1> 배열형 생성 함수들

내장함수	설명
str(객체)	객체를 출력할 수 있는 문자열로 변환
llist(시퀀스)	시퀀스를 받아서 같은 순서의 리스트 생성
tuple(시퀀스)	시퀀스를 받아서 같은 순서의 튜플 생성
range(시작값, 끝값 [, 증분])	0부터 끝값-1까지의 range형 생성
set(시퀀스)	시퀀스를 받아서 같은 순서의 집합 생성 (중복 불가)
frozenset(시퀀스)	집합(set)의 불가역형
bytearray()	1바이트 요소를 갖는 리스트 (가역 시퀀스)
bytes()	bytearray의 불가역형

이 표에서 전 장에서 소개한 적이 없는 자료형들이 있는데 frozenset()과 bytearray() 그리고 bytes() 등이다. 다른 자료형보다 사용빈도는 낮으므로 여기에서는 간략히 설명만 하고 넘어가도록 하겠다.

먼저 frozenset()은 집합과 동일한 성질의 배열형인데 변경이 불가하다. 마치 튜플이 리스트의 불가역형인 것처럼 frozenset도 집합의 불가역형이다. 따라서 집합에서 요소를 변경하는데 사용하는 메서드인 add(), clear(), discard(), pop(), remove() 등을 사용할 수 없다. 또한 집합과 달리 디렉터리의 키로 사용할 수 있다.

bytearray()는 요소가 1바이트 숫자(0~255)인 리스트로 간주하면 된다. bytes()는 bytearray의 불가역 자료형이다. 보통 문자열을 ASCII코드로 처리하고 싶을 때 bytearray나 bytes 객체를 사용한다.

9.4 시퀀스를 다루는 내장함수들

9.4.1 len() 함수

len()함수는 시퀀스의 요소의 개수를 반환하는 함수이다.

```
>>> len('abcd')
4
```

```
>>> a=[11, 0, -10, -22]
>>> len([x for x in a if x!=0]) #❶
3
```

❶은 a의 요소 중 0이 아닌 것의 개수를 구하는 것이다.

9.4.2 enumerate() 함수

enumerate()은 시퀀스로부터 enumerate형을 반환하는데 이것은 (인덱스, 요소) 튜플을 요소로 갖는 시퀀스이다.

```
>>> seasons = ['spring','summer','fall','winter']
>>> list(enumerate(seasons))
[(0, 'spring'), (1, 'summer'), (2, 'fall'), (3, 'winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'spring'), (2, 'summer'), (3, 'fall'), (4, 'winter')]
```

보통은 for 문과 같이 사용되어서 반복문 안에서 시퀀스의 요소뿐만 아니라 그 인덱스 정보도 필요할 경우 사용한다.

```
>>> seasons=['spring','summer','fall','winter']
>>> for i,e in enumerate(seasons):
...:     print(f'({i},{e})')

0:spring
1:summer
2:fall
3:winter
```

위에서 보인 바와 같이 for반복문에서 반복변수 v에는 seasons의 요소가, 반복변수 i에는 그 요소의 인덱스가 대입된다.

9.4.3 filter()와 map()함수

이 함수들은 전 장의 익명함수를 설명할 때 예로 든 적이 있다. 다시 설명하면 filter()는 함수와 시퀀스를 입력으로 받아서 시퀀스의 요소가 하나씩 함수에 인수로 전달될 때, 참을 반환시키는 것들만 따로 모아서 반환하는 함수다. 이 결과를 리스트

나 튜플로 변환하려면 `list()`, `tuple()` 함수를 이용해야 한다. 다음 예를 살펴보자.

```
>>> def pos(x):
...     return x > 0
>>> list(filter(pos, [1,-3,2,0,-5,6]))
[1, 2, 6]
```

`filter()` 함수는 첫 번째 인수로 함수를, 두 번째 인수로는 시퀀스를 받는다. 두 번째 인수인 시퀀스의 각 요소들이 함수에 들어갔을 때 리턴값이 참인 것만을 묶어서 돌려준다. 위의 예에서는 1, 2, 6 만이 양수로 $x > 0$ 이라는 문장이 참이 되므로 [1, 2, 6]이라는 리스트가 생성되었다. 전 장에서 설명한 바와 같이 익명함수(`lambda`)를 쓰면 더욱 간단하게 사용할 수 있다.

```
>>> list(filter(lambda x: x > 0, [1,-3,2,0,-5,6]))
[1, 2, 6]
```

`map()` 함수는 `filter()`와 유사한 면이 있다. `map()`도 함수와 시퀀스를 입력으로 받아서 각각의 요소가 함수의 입력으로 들어간 다음 나오는 출력값을 묶어서 이터레이터로 돌려주는 함수이다. 만약 어떤 리스트의 모든 요소의 제곱값을 갖는 새로운 리스트를 만들고 싶다면 다음과 같이 하면 된다.

```
>>> a = [2, 3, 4, 7, 8, 10]
>>> a2 = list(map(lambda x:x**2, a))
[4,9,16,49,64,100]
```

파이썬3에서는 이들 함수의 반환값이 리스트가 아닌 반복형(`iterable`)이므로 이것을 리스트로 만들기 위해서는 `list()` 함수를 명시적으로 사용해야 한다. 튜플로 만들고 싶다면 `tuple()` 내장함수를 사용하면 된다.

```
>>> tuple(map(lambda x:2**x, range(1,11)))
(2, 4, 8, 16, 32, 64, 128, 256, 512, 1024)
```

위에서 `range(1,11)`은 1부터 10까지의 시퀀스이므로 (2^{**1} , 2^{**2} , ..., 2^{**10}) 튜플이 생성된 것이다.

9.4.4 `sorted()`와 `reversed()`

`sorted()` 함수는 오름차순으로 정렬된 리스트를 얻는 함수이다. 반면, `reversed()`는 내림차순으로 정렬된 시퀀스를 반환하므로 이것을 리스트로 변환하려면 `list()` 함수를

거쳐야 한다.

```
>>> a=[-10,-5,1,0]
>>> sorted(a)
[-10, -5, 0, 1]
>>> b=sorted(a)
>>> b
[-10, -5, 0, 1]
>>> c=list(reversed(a))
>>> c
[0, 1, -5, -10]
```

리스트에는 sort()메서드가 있다. 만약 a.sort()라고 메서드를 호출하면 a리스트 자체가 정렬되어 변경된다. 반면 sorted(a)는 a리스트의 요소를 정렬하되 별도의 리스트를 만들어서 반환한다. 따라서 위에서 a리스트와 b리스트는 별개의 리스트이다.

9.5 객체의 정보를 얻는 내장함수들

9.5.1 type()함수

객체의 데이터형을 알 수 있는 함수로 type()이 있는데 전 장에서도 몇 번 소개된 바가 있다. 이것을 이용하여 예를 들면 변수에 담긴 객체가 문자열인가를 판단할 때 is 연산자와 조합해서 사용할 수 있다.

```
>>> a='hello'
>>> type(a) is str
True
>>> b=['hello','world']
>>> type(b) is list
True
>>> c=11.0
>>> type(c) is float
True
```

type()함수는 뒤에서 소개될 클래스의 객체인지도 확인할 수 있다. 단 상속된 클래스 까지도 고려하려면 isinstance()함수를 사용해야 하는데 이는 클래스의 상속을 설명할 때 더 자세히 설명토록 하겠다.

9.5.2. id()함수

id()함수는 객체의 고유주소(양의 정수값)를 반환하는 데 전장에서 참조를 설명할 때 그 개념을 자세히 설명한 바가 있다. 두 객체를 a와 b라고 할 때 a is b는 id(a)==id(b)와 같은 결과를 발생시킨다.

9.5.3 dir()함수

dir()함수는 객체의 메서드와 변수들의 목록을 리스트의 형태로 반환해준다.

```
>>> a={1,2,3,4,5,6}
>>> dir(a)
['__and__', ..., '__xor__', 'add', 'clear', 'copy',
'difference', 'difference_update', 'discard',
'intersection', 'intersection_update', 'isdisjoint',
'issubset', 'issuperset', 'pop', 'remove',
'symmetric_difference',
'symmetric_difference_update', 'union', 'update']
```

위의 예에서 a는 집합객체인데 이 객체에 대해서 add나 clear같은 메서드가 있다는 것을 확인할 수 있다. 객체의 메서드 중 __로 시작되고 __로 끝나는 것들은 (magic method라 불리는 특수한 메서드들 보통 직접 호출할 수 없는 것들이고 영문자로 시작하는 메서드들은 직접 실행할 수 있는 것들이다.

9.6 기타 함수들

ord()함수는 유니코드 문자의 코드값(양의 정수)을 반환한다. chr()함수는 반대로 유니코드에 해당하는 문자를 반환한다.

```
>>> ord('박')
48149
>>> chr(48149)
'박'
```

여기서 유니코드(uni-code)는 다국어 문자들에 부여되어 있는 고유번호 정도로만 이해하고 넘어가도록 하겠다.

9.7 모듈

특정한 기능을 갖는 이미 작성된 함수들의 집합을 모듈(module)이라고 하고, 그것들을 필요한 때 읽어 들여서 사용할 수 있다. 이 모듈을 이용하면 다양한 기능을 추가하여 복잡한 일도 손쉽게 수행할 수 있다. 파이썬에 기본적으로 포함된 모듈들을 표준 라이브러리(standard library)라고 한다. 파이썬은 표준 라이브러리 외에도 여러 가지 용도로 활용할 수 있는 편리한 모듈이 매우 많이 존재하며 이는 파이썬의 강력한 장점 중 하나이다.

- **표준 라이브러리** : 파이썬에 기본으로 포함되는 모듈들의 집합

파이썬에 기본으로 포함된다는 것의 의미는 모듈을 사용할 때 설치(install)하는 과정이 필요하지 않고 바로 읽어들이 수 있다는 의미이다. 사실 현재 표준 라이브러리 외에도 파이썬의 라이브러리는 방대하다고 할 수 있을 만큼 숫자가 굉장히 많다. 따라서 표준 라이브러리에 원하는 기능을 갖는 모듈이 없다고 할지라도 구글링(googleing)을 통해서 검색하는 과정이 반드시 필요하다.

모듈을 사용하여 파이썬의 기능을 확장하려면 import 명령을 이용하면 된다.

```
import 모듈명
```

표준 라이브러리 중 하나의 calendar을 읽어들이어 사용하려면 다음과 같이 하면 된다.

```
import calendar
```

이 모듈에는 month()라는 함수가 미리 정의되어 있는데 년도와 월을 차례로 인수로 넘겨주면 달력을 출력하는 기능을 가지고 있다.

```
>>> import calendar
>>> print(calendar.month(2021,6))
    June 2021
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
```

이와 같이 만약 프로그램 작성자가 달력을 출력하는 기능이 필요로 하다면 이미 작성된 모듈을 불러서 사용하면 된다.

표준 라이브러리에 random이라는 모듈이 있는데 난수를 발생시키는 기능을 가지고 있다. random모듈에는 다음과 같이 편리하게 사용할 수 있는 함수들이 미리 작성되어 있다.

〈표 9.7.1〉 random모듈의 함수

random모듈의 함수	기능
randint(a, b)	a이상 b이하의 임의의 정수 반환(b도 포함된다).
random()	0이상, 1미만의 임의의 실수를 반환
randrange(a,b[,c])	a이상 b미만 (증분 c)의 임의의 정수 반환
choice(시퀀스)	시퀀스의 요소 중 임의의 요소 반환
shuffle(시퀀스)	시퀀스 요소의 순서를 임의로 섞는다 (시퀀스 자체를 변경시킴)

이 모듈을 이용하면 다음과 같이 주사위를 구현할 수 있다.

```
>>> from random import randint
>>> dice='□□□□□□'
>>> dice[randint(0,5)]
'□'
>>> dice[randint(0,5)]
'□'
>>> dice[randint(0,5)]
'□'
```

위의 첫줄에 from random import randint 라는 명령이 있는데 이것은 random이라는 모듈로부터(from) randint라는 함수 하나만 읽어들이라(import)는 명령이다. 이렇게 import하면 random모듈 내의 여러 함수 중에서 randint함수만을 바로 사용할 수 있다. randint(0,5)함수는 0,1,2,3,4,5 중 임의로 하나를 선택해서 반환하므로 dice튜플의 임의의 요소가 화면에 출력되는 것이다. 다음 예제 파일은 주사위를 5번 굴리는 효과를 낸다.

```
dice.py
```

```

from random import randint
dice='□□□□□□'
for _ in range(5):
    i = randint(0,5)
    print(dice[i])

```

실행결과

```

□□
□
□
□□
□□

```

이 프로그램을 dice.py로 저장한 후 **F5**를 눌러서 실행시켜 보자.
import 명령을 정리하면 다음과 같다.

- import 모듈명
 - 모듈 전체를 읽어들인다.
 - 모듈 안의 함수는 모듈명.함수명() 으로 호출한다.
- from 모듈명 import 함수명1, 함수명2, ...
 - 모듈 내의 특정함수(들)만 읽어 들인다.
 - 모듈명을 거치지 않고 함수명1() 등과 같이 바로 호출해서 사용한다.

예를 들어 random 모듈에서 choice와 shuffle 함수 두 개만 읽어들이려면 다음과 같이 하면 된다.

```

>>> from random import choice, shuffle
>>> dice=(1,2,3,4,5,6)
>>> choice(dice)
5
>>> cards=['heartAce','heart2','diamondAce','diamond2']
>>> shuffle(cards)
>>> cards
['diamond2', 'heartA', 'heart2', 'diamondA']

```

모듈에는 함수뿐만 아니라 뒤에서 소개될 클래스가 정의되어 포함될 수도 있으며 상수가 정의될 수도 있다. 이것에 대해서는 뒤에서 더 자세히 다루도록 하겠다.

9장 연습문제

9-1. 트럼프 카드는 네 가지 무늬 ♠, ♥, ♦, ♣ 와 이들 뒤에 붙는 숫자가 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A 가 있어서 이들을 모두 조합하면 52장의 카드이고 여기에 JOKER 카드 두 장이 더해져 총 54장의 카드가 된다. 이들 54장의 카드를 요소로 갖는 리스트를 생성하여 trump_card 변수에 저장하라.

9-2 위에서 생성한 trump_cards 리스트를 이용하여 임의의 카드를 다섯 장씩 세 사람에게 나눠주는 프로그램을 작성하라. 즉, player1, player2, player3 변수에 각각 다섯 개의 요소가 저장된 리스트를 생성하면 된다.

제 10 장 클래스

클래스(class)는 파이썬의 핵심 기능 중 하나이다. 파이썬의 클래스 기능으로 인해 객체 지향적인 프로그래밍을 할 수 있는 표준 기능들을 제공한다. 클래스는 데이터와 그 데이터를 다루는 함수를 함께 묶는 방법을 제공한다. 클래스는 새로운 자료형을 정의하는 것이고 객체(object)는 클래스의 인스턴스(instance, 클래스가 구체화된 것)를 의미한다. 즉, 클래스를 통하여 객체를 생성할 수 있는데 지금까지 다뤄온 숫자, 문자열, 리스트, 딕셔너리 같은 기본 자료형도 사실은 모두 이 객체이다.

- 클래스(class) - 새로운 자료형을 정의하는 설계도
- 객체(object) - 클래스를 이용하여 구체화된 데이터

다음 그림은 클래스를 개념적으로 잘 설명해 준다. 예를 들어서 손바닥 모양의 무늬가 많이 필요할 때 그 모양을 찍을 수 있는 도장을 만들어 놓으면 여러 개의 무늬를 손쉽게 생성할 수 있다.



<그림 11.1> 도장을 이용한 무늬를 찍는 모습

여기서 도장을 클래스에 비유할 수 있고 찍힌 모양 무늬는 바로 객체라고 할 수 있다. 클래스란 동일한 기능을 하는 무엇인가를 여러 개를 만들어낼 수 있는 설계도 같은 것이고 객체는 이 클래스에 의해서 만들어진 실체이다. 객체에는 수치나 문자열 같은 데이터가 들어 있을뿐만 아니라 그 데이터에 대해서 여러 가지 처리를 할 수 있는 메서드도 갖고 있다. 객체는 단순히 데이터만을 가지는 것이 아니라 어떻게 동작하면 되

는지를 알고 있다고 할 수 있다.

10.1 클래스의 정의 및 객체 생성

클래스는 다음과 같이 정의한다.

```
class 클래스명:
    클래스 본체
```

관례적으로 클래스의 이름은 대문자로 시작한다. 클래스 본체는 반드시 들여쓰기를 해야 한다. 클래스명의 예를 들면 다음과 같다.

Person, Animal, Robot, AutoCar, MyPersonalObject

이와 같이 각 단어를 붙여 쓰는 대신 단어의 첫 문자를 대문자로 시작하는 표기법을 camel-case 표기법이라고 한다.

- 클래스명은 대문자로 시작한다.
- 클래스명이 두 단어 이상일 경우 관례적으로 camel-case 표기법에 따른다.

반면, 변수명이나 함수명은 관례적으로 소문자로 시작하고 각 단어를 밑줄(_)로 구분하는데 이를 snake-case 표기법이라고 한다.

클래스 본체는 이 클래스에 속하는 변수와 함수를 정의하게 된다. 특별히 클래스에 속한 변수들을 필드(field), 클래스에 속한 함수들을 일반 함수들과 구분하기 위해서 메서드(method)라고 부르며 이 둘을 통칭하여 속성(attribute)라고 한다. 이후에 본 교재에서는 이 용어들을 일관적으로 사용하도록 하겠다.

- 필드(field) : 객체에 내장된 변수
- 메서드(method) : 객체를 통해서 호출하고 객체 필드를 이용하는 함수
- 속성(attribute) : 필드와 메소드를 통칭해서 속성이라 한다.

가장 간단한 형태의 클래스를 다음과 같이 정의해 보자.

```
class Point:
```

```
pass
```

이 클래스는 보면 알겠지만 본체가 없다. 전술한 바와 같이 클래스의 구현체를 객체라고 하는데 예를 들어 Point 클래스의 객체를 생성하려면 다음과 같이 하면 된다.

```
>>> pa = Point()
```

이제 pa 는 Point 클래스의 객체가 되었다. 다른 객체들도 얼마든지 생성할 수 있다.

```
>>> pb = Point()
>>> pc = Point()
>>> pd = Point()
```

이와 같이 어떤 클래스의 객체를 생성하여 변수에 저장하려면 다음과 같이 한다.

```
변수명 = 클래스명()
```

객체를 생성할 때 클래스명 뒤에 괄호()가 오는 것이 마치 함수를 호출하는 것과 유사하지만 클래스명은 보통 대문자로 시작하고 함수는 소문자로 시작하므로 함수를 호출하는 것과 객체를 생성하는 것은 이것으로 구별할 수 있다.

10.2 __init__ 메서드와 self 매개변수

전 절에서 객체를 생성하는 것이 외형상으로 함수를 호출하는 것과 유사하다고 했는데, 사실 객체를 생성할 때 클래스의 __init__() 메서드를 호출하게 된다. 메서드 이름의 앞과 뒤에 __ 이 붙어있는 것들을 매직메서드(magic method)라 칭한다. 타이핑할 때 언더바(_)가 두 개 연속인 것이 주의하자. 이 메서드는 다음과 같이 클래스 본체 내에 사용자가 정의할 수 있다.

```
class 클래스명:

    def __init__(self, 매개변수들...):
        메소드 본체
```

이 예에서 __init__(self) 메서드는 객체를 생성할 때 자동으로 호출되는 특수한 메서드이고 반드시 첫 번째 매개변수는 self 이어야 한다. 객체가 생성될 때 호출되는 함수이기 때문에 생성자(constructor)라고도 불리고, 보통 클래스를 작성할 때 반드시 정의를 해주어야 하는 메서드이다.

- 생성자 : 객체가 생성될 때 호출되는 `__init__()` 메서드
 - 첫 번째 매개변수는 `self` 이고 이것은 객체 자체이다.
 - 사용자가 명시적으로 (None 이외의) 어떤 값을 반환할 수 없다.

생성자 내부에서 객체의 필드를 `self` 매개변수를 이용하여 생성할 수 있다.

```
class Point:

    def __init__(self,x,y):
        self.x = x
        self.y = y
```

전술한 바와 같이 생성자는 사용자가 명시적으로 어떤 값을 반환할 수 없으며 위의 예제에서도 아무런 값도 반환되지 않았고 오직 `self` 매개변수를 이용하여 필드를 생성하는 작업만을 하고 있다. 일단 생성자는 내부적으로 (자동으로) `self`를 반환한다고 개념적으로 이해하면 된다.

이제 아래와 같이

```
>>> p = Point(2, 3)
```

`Point`객체 `p`를 생성하면 `x` 와 `y` 필드가 생성되고 넘겨준 인수값 2와 3으로 초기화된 다. 전 절에서 객체를 생성하는 것이 함수와 모양이 유사하다고 했는데 사실 이렇게 호출하면 `__init__()` 함수가 호출되는 것이다. 이 때 `x`로는 2가 `y`로는 3이 넘어간다. 그리고 호출하는 쪽에서는 `self` 매개변수는 고려할 필요가 없는데 바로 `Point` 객체가 하나 생성되어 `self` 파라미터로 자동으로 넘어가기 때문이다.

```
Point객체 ⇨ self,
2 ⇨ x,
3 ⇨ y
```

그리고 생성자는 자동으로 `self`객체를 반환한다. 따라서 `p`변수에는 생성자가 인수로 건네받아서 필요한 작업을 거친 후 반환되는 `self`객체가 저장되는 것이다. 객체의 필드는 객체변수 뒤에 점을 찍은 다음 필드명을 써서 접근한다.

```
>>> p.x
2
```

```
>>> p.y
2
```

객체의 필드와 그 값을 확인하고 싶다면 내장함수 vars()를 이용하면 된다.

```
>>> vars(p)
{'x': 2, 'y': 2}
```

(뒤에서 더 자세히 설명하겠지만 이 내장 함수는 객체의 __dict__ 필드를 반환한다. __dict__ 는 객체의 속성을 요소로 갖는 딕셔너리이다. 즉, vars(p)는 p.__dict__ 를 반환한다.)

만약 아무런 값도 인수로 넘어오지 않았을 때 x, y필드를 둘 다 0으로 초기화하고 싶다면 다음과 같이 기본값 매개변수를 사용하면 된다.

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

이제 객체를 생성할 때 인수를 넘기지 않아도 x, y필드가 기본값으로 초기화된다.

```
>>> p0=Point()
>>> vars(p0)
{'x': 0, 'y': 0}
```

이와 같이 __init__() 생성자도 일반 함수처럼 표준 매개변수뿐만 아니라 기본값 매개변수, 가변개수 매개변수, 키워드 매개변수 등을 적용할 수 있다. 단, __init__() 메서드의 정의가 일반적인 함수와 다른 점은 첫 번째 인수가 반드시 self이어야 한다는 점인데 다음 절에서 설명하듯이 다른 사용자 메서드들도 첫 번째 인수는 반드시 self이어야 한다.

10.3 사용자 메서드 정의

클래스 정의부에서 사용자의 필요에 의해 작성되는 메서드는 일반 함수를 정의하는 것과 동일하지만 생성자와 마찬가지로 '첫 번째 인수는 반드시 self가 되어야 한다'는 점이 다르다.

- 메서드 정의에서 첫 번째 인수는 self 이다.

관례적으로 메서드 정의들 사이에는 한 줄 공백을 준다.

```
class Point:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def abs(self):
        xs = self.x**2
        ys = self.y**2
        return (xs + ys)**0.5
```

self라는 매개변수를 클래스 메서드의 첫 번째 매개변수로 두어야 하는 것은 파이썬만의 특징이다. 언어의 규칙이 이렇게 규정되어 있으니 사용자는 거기에 맞추어 작성해 주어야만 한다. 메서드 내에서는 이 self 매개변수를 통해서 기정의된 필드에 접근하거나 다른 메서드를 호출할 수 있으며 또한 새로운 필드를 생성하는 것도 가능하다. abs() 메서드 내부에서 x 라는 필드를 self 매개변수를 통해서 self.x 와 같이 사용했음을 알 수 있다. 이와 같이, 객체의 필드는 메서드 내에서 반드시 self 매개변수를 통해서만 접근할 수 있다.

객체의 모든 필드와 메서드 목록(숨겨진 것들 까지 포함한)을 확인하고 싶으면 내장함수 dir()를 이용하면 된다.

```
>>> p=Point(3,4)
>>> dir(p)
['__class__', '__delattr__', '__dict__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__gt__', '__hash__', '__init__',
 '__init_subclass__', '__le__', '__lt__', '__module__',
 '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', '__weakref__', 'abs', 'x', 'y']
```

위에서 보면 매직속성들(‘__’로 시작하고 ‘__’로 끝나는 속성들) 목록 다음에 abs, x, y 등 필드와 메서드의 이름을 모두 확인할 수 있다. vars()내장함수는 필드의 이름과 저장된 값을 알 수 있는 반면, dir() 내장함수는 모든 속성(필드와 메서드)의 이

를만을 확인할 수 있다.

이제 p라는 Point 클래스 객체를 통해서 메서드를 호출할 수 있다. 필드를 접근할 때와 마찬가지로 객체명 뒤에 점(.)을 찍고 호출할 메서드명을 명시하면 된다.

```
>>> p=Point(3,4)
>>> p.abs()
5.0
```

메서드를 호출하는 쪽에서는 메서드 정의부의 self 매개변수는 없는 것으로 간주하고 인수를 넘겨주면 된다. 메서드를 호출할 때는 첫 번째 인수로 객체 자체가 자동으로 넘겨지기 때문이다. 따라서 p.abs() 메서드는 인수가 없이 호출하면 된다.

생성자와 마찬가지로 self 매개변수 뒤에 표준 매개변수를 둘 수도 있다. 다음 예는 각 좌표에 어떤 값을 더하는 기능을 하는 메서드들을 추가한 것이다.

```
point.py
class Point:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def abs(self):
        xs = self.x**2
        ys = self.y**2
        return (xs + ys)**0.5

    def addx(self, dx):
        self.x += dx

    def addy(self, dy):
        self.y += dy

    def add(self, dx, dy):
        self.x += dx
        self.y += dy
```

이렇게 정의하면 addx(), addy() 메서드는 하나의 표준 인수를 가지고 호출해야 하고 add()함수는 두 개를 주어야 한다.

```

>>> p=Point()
>>> p.addx(11)
>>> vars(p)
{'x': 11, 'y': 0}
>>> p.addy(22)
>>> vars(p)
{'x': 11, 'y': 22}
>>> p.add(33,44)
>>> vars(p)
{'x': 44, 'y': 66}

```

이와 같이 메서드는 첫 번째 매개변수가 self 라는 점만 제외하면 일반 함수와 정의하는 방법이 동일하다.

- 메서드 정의부에서 첫 번째 매개변수는 self다
- 생성자와 달리 어떤 값이라도 반환할 수 있다.
- 메서드를 호출할 때에는 self매개변수는 없는 것으로 간주하고 인수를 지정한다.

메서드 안에서 다른 메서드를 호출하는 것도 가능하다. 위의 예에서 add()함수를 다음과 같이 변경할 수 있다.

```

point.py

class Point:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def abs(self):
        xs = self.x**2
        ys = self.y**2
        return (xs + ys)**0.5

    def addx(self, dx):
        self.x += dx

    def addy(self, dy):


```

```

        self.y += dy

    def add(self, dx, dy):
        self.addx(dx)
        self.addy(dy)

```

이 예에서 보면 add()메서드 안에서 addx(), addy()메서드를 호출한 것을 알 수 있다. 이 파일을 point.py라고 저장한 후 키를 눌러 실행시키면 파이썬셸에서 Point클래스를 사용할 수 있다.

10.4 클래스 예제

여기에서는 사각형(rectangle)의 정보를 다루는 새로운 자료형을 클래스로 구현해 보도록 하겠다. 클래스의 이름은 Rect로 하고 생성자에서는 이 사각형의 너비와 높이 그리고 중심점의 xy좌표를 Point클래스의 객체로 받도록 한다. 아래의 rect.py와 전 절의 point.py는 같은 폴더에 저장해야 한다.

rect.py


```

from point import Point #❶

class Rect:

    def __init__(self, width, height, point=Point()):
        self.width = width
        self.height = height
        self.center = point

```

❶은 point모듈(point.py파일)에서 Point클래스를 임포트해서 현재 파일에서 사용할 수 있도록 하는 것이다. 이렇게 해두면 기존에 작성해둔 Point 클래스를 간단하게 재사용할 수 있다. 생성자는 width, height, point 세 개를 받도록 했는데 point는 기본값으로 원점좌표를 가지는 Point객체가 설정되어 있다. 이 프로그램을 를 눌러서 실행시키면 파이썬셸에서 Rect 클래스의 객체를 생성할 수 있다.

```

>>> a=Rect(10,20)
>>> print(a.point.x, a.point.y)
0 0
>>> b=Rect(30,30,Point(1,2))

```



```
>>> print(b.point.x, b.point.y)
1 2
```

위의 예에서 a 객체는 중심점이 1이고 너비가 10, 높이가 20인 직사각형을, b 객체는 너비와 높이가 30이고 중심점이 (1,2)인 사각형의 정보를 가지고 있다.

이제 사각형이 정사각형인지를 판별하는 is_square() 메서드와 너비를 계산하는 get_area() 메서드를 추가해 보자.

```
def is_square(self):
    return self.width == self.height

def get_area(self):
    area = self.width*self.height
    return area
```

이제 코드를 다시 저장한 후 **F5**키를 눌러 실행하면 파이썬셸에서 수정된 Rect 클래스를 사용할 수 있다.

```
>>> a=Rect(15,15,Point(20,30))
>>> a.is_square()
True
>>> a.get_area()
225
>>> b=Rect(20,30)
>>> b.is_square()
False
>>> b.get_area()
600
```



만약 사각형의 중심점과 원점 간의 거리를 구하는 get_distance_from_origin()이라는 메서드를 추가하고자 한다면 다음과 같이 Point 객체의 abs()메서드를 이용하면 된다.


```
def get_distance_from_origin(self):
    return self.center.abs()
```

왜냐면 Point 객체의 abs()메서드가 원점과 객체의 (x,y)좌표점과의 거리를 반환해 주기 때문이다. 이제 코드를 다시 저장한 후 **F5**키를 눌러 실행하면 파이썬셸에서 수정된

Rect 클래스를 사용할 수 있다.

```
>>> pt=Point(3,4)
>>> a=Rect(10,10,pt)
>>> a.get_distance_from_origin()
5.0
```

IDLE를 사용하고 있다면 메서드의 이름이 긴 경우에는 앞의 몇 문자만 입력한 후  키를 누르면 나머지가 자동으로 입력되는 기능을 사용할 수 있다. 예를 들어 a.get_d까지만 입력하고  키를 누른다.

```
>>> a.get_d
```

만약 입력한 문자로 시작하는 속성들이 여러 개라면 그것들이 팝업창에 나열되고 하나를 선택할 수 있다. 전체 프로그램은 아래와 같다.

rect.py

```
from point import Point

class Rect:

    def __init__(self, width, height, point=Point()):
        self.width = width
        self.height = height
        self.center = point

    def is_square(self):
        return self.width == self.height

    def get_area(self):
        area = self.width*self.height
        return area

    def get_distance_from_origin(self):
        return self.center.abs()
```

클래스를 처음 코딩을 할 때 메서드의 첫 번째 매개변수가 self라는 것을 빠뜨리기 쉬우니 주의하자.

10.5 매직메서드

매직메서드는 전술한 바와 같이 `__` 로 시작하고 `__` 로 끝나는 특별한 메서드로서 앞에서도 설명한 생성자(`__init__`), +연산자 오버로딩(`__add__`) 같은 것들이 있다. 이외에도 다양한 매직메서드들이 있는데 여기에서는 이것들에 관해서 살펴보도록 하자.

10.5.1 `__str__()`메서드와 `__repr__()` 메서드

`__str__()`메서드는 객체가 내장함수 `str()`의 인수로 넘겨졌을 때 호출되는 함수이다. 혹은 `print()`함수의 인수로 넘겨졌을 경우에도 이 `__str__()` 매직메서드가 호출된다. 반면 `__repr__()`메서드는 내장함수 `repr()`의 인수로 넘겨진 경우 혹은 파이썬셸에서 객체를 문자열로 표현해야 할 경우에 (즉, `>>>` 뒤에 객체명만 적고 엔터키를 누르는 경우 출력되는 문자열) 호출되는 메서드이다. 두 메서드가 비슷한 역할을 하고 있지만 굳이 기능적인 측면에서 구별하자면 `__str__()`은 원래 객체를 복구할 수 있는 파이썬 코드를 문자열로 생성하면 되고, `__repr__()`은 객체의 내용을 사람이 보기에 이해하기 쉬운 문자열을 담으면 된다.

다음은 필드가 `x`와 `y` 두 개인 `Vector`클래스의 예이다.

vector.py

```
class Vector:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'Vector({self.x},{self.y})'

    def __repr__(self):
        return f'<{self.x},{self.y}>'
```

이렇게 정의된 `Vector`클래스에 대해서 예를 들어서

```
>>> a = Vector(11,22)
```

라고 객체를 생성한 다음에 다음과 같이 파이썬셸에서 명령을 내리면 어떻게 출력이 되는지를 관찰해 보자.

```
>>> str(a)
>>> repr(a)
>>> a #❶
>>> print(a) #❷
```

❶의 경우에는 `__repr__()` 메서드가, ❷의 경우는 `__str__()` 매직메서드가 호출됨을 알 수 있다.

10.5.2 산술연산자 오버로딩

연산자 오버로딩(operator overloading)이란 객체간 연산에 파이썬 연산자를 사용할 수 있도록 하는 기능을 말한다. 예를 들면 객체간 덧셈이나 뺄셈에서 `+` 연산자, `-` 연산자를 사용할 때 프로그래머가 원하는 결과값을 반환하도록 직접 작성할 수 있다.

- 연산자 오버로딩
 - 객체 간 연산에 파이썬 연산자를 사용할 수 있도록 한다.
 - 매직메서드(`__add__`, `__sub__` 등)를 이용하면 된다.

각종 산술연산자를 오버로딩하는 매직메서드가 <표 10.5.1>에 설명되어 있다.

<표 10.5.1> 산술연산자 오버로딩 매직메서드

매직메서드	동작
<code>__add__(self, other)</code>	<code>self + other</code> 실행 시 호출
<code>__sub__(self, other)</code>	<code>self - other</code> 실행 시 호출
<code>__mul__(self, other)</code>	<code>self * other</code> 실행 시 호출
<code>__truediv__(self, other)</code>	<code>self / other</code> 실행 시 호출
<code>__floordiv__(self, other)</code>	<code>self // other</code> 실행 시 호출
<code>__mod__(self, other)</code>	<code>self % other</code> 실행 시 호출
<code>__divmod__(self, other)</code>	내장함수 <code>divmod(self, other)</code> 실행 시 호출
<code>__pow__(self, other)</code>	<code>self ** other</code> 실행 시 호출

이것을 설명하기 위해서 전 절의 `Vector` 클래스를 계속 예제로 사용하도록 하겠다. `Vector` 클래스는 `x`, `y` 두 개의 필드를 갖는데 만약 두 객체를 더하려고 한다면 `x`는 `x`끼리, `y`는 `y`끼리 더해야 한다. 이를 위해서 다음과 같이 `add()` 메서드를 추가할 수 있다.

```
def add(self, vec):
    x = self.x + vec.x
    y = self.y + vec.y
    return Vector(x,y)
```

add()메서드를 이용하면 두 벡터의 덧셈을 수행하여 그 결과를 저장한 새로운 Vector 객체를 얻을 수 있다.

```
>>> v1=Vector(1,2)
>>> v2=Vector(2,3)
>>> v3 = v1.add(v2)
>>> v3
<3, 5>
```

하지만 add() 메서드를 이용하는 것 보다 다음과 같이

```
>>> v3 = v1 + v2
```

덧셈연산자를 이용할 수 있다면 코드가 더 직관적이 되고 사용하기에도 편리할 것이다. 이렇게 객체 간 덧셈연산자를 이용할 수 있도록 해 주는 것인 매직메서드 __add__()이다. 위의 예에서 add()메서드의 이름을 __add__()로 바꿔보자.

```
def __add__(self, vec):
    x = self.x + vec.x
    y = self.y + vec.y
    return Vector(x,y)
```

이렇게 작성하면 덧셈연산자(+)를 사용할 수 있다.

```
>>> v1=Vector(11,22)
>>> v2=Vector(33,44)
>>> v3=v1+v2 #❶ v1.__add__(v2)가 자동 호출
>>> v3
<44, 66>
>>> v4 = v1+v2+v3 #❷ v1.__add__(v2).__add__(v2)
>>> v4
```

```
<88, 132>
```

위의 ❶에서 $v3=v1+v2$ 가 실행될 때 내부적으로는 `v1.__add__(v2)` 와 같이 매직메서드가 호출이 되는 것이다. ❷의 경우는 내부적으로 `v1.__add__(v2).__add__(v3)`와 같이 실행될 것이다. 내부적으로 어떠한 방식으로 실행이 되든지 사용자는 덧셈(+)연산자를 이용하여 간단하게 사용할 수 있다.

뺄셈연산자를 오버로딩하려면 `__sub__()`메서드를 작성하면 된다.

```
def __sub__(self, vec):
    x = self.x - vec.x
    y = self.y - vec.y
    return Vector(x,y)
```

그러면 다음과 같이 Vector 객체 간 뺄셈을 수행할 수 있다.

```
>>> v1=Vector(1,2)
>>> v2=Vector(-3,4)
>>> v2-v1
<-4, 2>
```

만약 두 Vector 객체간 곱셈을 수행할 때 내적(inner product)을 계산하고 싶다면 다음과 같이 `__mul__()` 매직메서드를 작성하면 된다.

```
def __mul__(self, vec):
    return self.x*vec.x + self.y*vec.y
```

그러면 Vector 객체 간의 곱셈은 내적이 계산된다.

```
>>> v1=Vector(2,1)
>>> v2=Vector(-1,2)
>>> v1*v2
0
```

전체 파일은 아래와 같다.

```
vector.py
```

```

class Vector:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return f'Vector({self.x},{self.y})'

    def __repr__(self):
        return f'<{self.x}, {self.y}>'

    def __add__(self, vec):
        x = self.x + vec.x
        y = self.y + vec.y
        return Vector(x,y)

    def __sub__(self, vec):
        x = self.x - vec.x
        y = self.y - vec.y
        return Vector(x,y)

    def __mul__(self, vec):
        return self.x*vec.x + self.y*vec.y

```

10.5.3 비교 연산자 오버로딩

산술연산자뿐만 아니라 비교연산자도 오버로딩할 수 있는데 이를 <표 10.5.2>에 나열했다. 이 매직메서드를 이용하면 사용자가 비교연산의 결과값을 정해서 작성할 수 있다.

전 절의 Vector 클래스를 가지고 설명하면, 두 Vector 객체가 같다고 판별하려면 x 값들 끼리 그리고 y 값들 끼리 서로 같아야 한다. 이를 위해서 `__eq__()` 매직메서드를 작성해야 하는데 이 메서드에서는 x끼리 같고 y끼리 같다면 True를 반환하도록 작성해야 한다. 하지만 `__eq__()` 매직메서드를 작성하지 않았더라도 객체간 비교연산은 가능하지만 사용자가 원하는 결과는 발생하지 않는다.

```

>>> v1=Vector(1,2)
>>> v2=Vector(1,2)

```

```
>>> v1==v2
False
```

<표 10.5.2> 비교연산자 오버로딩 매직메서드

매직메서드	동작
<code>__eq__(self, other)</code>	<code>self == other</code> 실행 시 호출
<code>__ne__(self, other)</code>	<code>self != other</code> 실행 시 호출
<code>__lt__(self, other)</code>	<code>self < other</code> 실행 시 호출
<code>__le__(self, other)</code>	<code>self <= other</code> 실행 시 호출
<code>__gt__(self, other)</code>	<code>self > other</code> 실행 시 호출
<code>__ge__(self, other)</code>	<code>self >= other</code> 실행 시 호출
<code>__bool__(self)</code>	내장함수 <code>bool(self)</code> 실행 시 호출

이제 다음과 같이 `__eq__()` 매직메서드를 `Vector` 클래스에 추가해보자.

```
def __eq__(self, vec):
    return self.x==vec.x and self.y==vec.y
```

그러면 다음과 같이 `v1==v2` 연산을 수행할 때 `v1.__eq__(v2)` 메서드가 호출되어 사용자가 작성한 논리값을 반환한다.

```
>>> v1=Vector(1,2)
>>> v2=Vector(1,2)
>>> v1 == v2
True
```

두 `Vector` 객체가 같지 않다는 조건을 판별하는 `__ne__()` 매직메서드는 다음과 같이 작성할 수 있을 것이다.

```
def __ne__(self, vec):
    return self.x!=vec.x or self.y!=vec.y
```

그러면 이 메서드는 `!=` 연산자를 사용할 때 호출된다.

```
>>> v1=Vector(1,2)
```



```
>>> v2=Vector(1,3)
>>> v3=Vector(1,2)
>>> v1 != v2
True
>>> v1 != v3
False
```

크기를 비교하는 연산자도 필요하다면 오버로딩할 수 있다. 만약 두 Vector 객체의 원점에서부터의 길이를 이용해서 $v1 < v2$ 비교식의 결과가 $v1$ 의 길이가 $v2$ 의 길이보다 작다면 True를 반환토록 하고자 한다면 어떻게 작성해야 할지 생각해 보자.

10.6 클래스변수와 정적메서드

10.6.1 클래스변수

객체의 필드는 각각의 객체마다 고유의 값을 갖는다. 다음과 같이 작성된 Robot 클래스를 살펴보자.

robot.py

```
class Robot:

    def __init__(self, name = 'dummy', legs=0):
        self.name = name
        self.legs=legs

    def talk(self):
        print(f'Hi. I am {self.name} with {self.legs} legs')

    def walk(self, step):
        if self.legs >= 2:
            print('done.')
        else:
            print('Sorry. Not enough legs.')

    def run(self, dist, speed = 10):
        print(f'Must go {dist}m at {speed}m/s speed.')
```

```

if self.legs >= 2:
    print('done.')
else:
    print('Sorry. Not enough legs.')

```

사용 예는 다음과 같다.

```

In [12]: my_robot = Robot('Tom')
In [13]: your_robot = Robot('Jane')
In [14]: my_robot.name
Out[14]: 'Tom'
In [15]: your_robot.name
Out[15]: 'Jane'

```

여기서 my_robot 과 your_robot은 Robot 클래스의 객체이며 각각 name 필드를 갖는다. 두 객체의 name필드는 완전히 독립적인 변수이며 서로 다른 참조를 갖는다. 즉, 필드는 각 객체에 개별적으로 할당되는 변수이다.

이에 반해서 같은 클래스의 모든 객체에서 공통적으로 참조할 수 있는 클래스 필드를 생성할 수 있는데 이를 클래스 변수(혹은 정적 필드라고도 함)라고 한다. 클래스 필드는 __init__() 함수 내부에서 생성하는 것이 아니라 메서드의 외부에서 정의되어야 한다.

robot.py

```

class Robot:

    count=0 #❶

    def __init__(self, name = 'dummy', legs=0):
        self.name = name
        self.legs=legs
        self.id = Robot.count #❷
        Robot.count += 1 #❸

    def talk(self):
        print(f'Hi. I am {self.name} with {self.legs}
legs')
        print(f'My model number is {self.id}')

```

```

def walk(self, step):
    if self.legs >= 2:
        print('done.')
    else:
        print('Sorry. Not enough legs.')

def run(self, dist, speed = 10):
    print(f'Must go {dist}m at {speed}m/s speed.')
    if self.legs >= 2:
        print('done.')
    else:
        print('Sorry. Not enough legs.')

```

이 예에서 ❶에서 선언된 변수 count는 클래스 변수이며 메서드 외부에서 정의되었다는 것을 알 수 있다. 클래스 필드는 ❷와 ❸과 같이 클래스명을 통해서 접근한다.

```

>>> a=Robot('hubo')
>>> b=Robot('asimo')
>>> a.talk()
Hi. I am hubo with 0 legs
My model number is 0
>>> b.talk()
Hi. I am asimo with 0 legs
My model number is 1
>>> Robot.count
2

```

위 예에서 Robot.count 가 2인 것은 객체가 모두 두 개가 생성되었기 때문이다.

이와 같이, 클래스변수는 객체의 필드와 달리 모든 객체에서 공통적으로 접근하여 읽거나 쓸 수 있는 변수이다. 클래스변수와 같은 이름을 갖는 객체 필드는 정의하지 않는 것이 불필요한 혼동을 피할 수 있다.

10.6.2 정적 메서드

클래스변수와 특성이 유사한 메서드로 정적 메서드(static method)가 있다. 정적 메서드는 개별 객체의 필드를 사용하는 메서드가 아니라 일반적인 함수의 소속을 클래스로 지정한다는 개념으로 이해하면 된다. 정적 메서드는 함수 정의부 바로 윗줄에 @staticmethod (이것을 decorator라고 한다.)라고 지정하여 정의하며 self 매개변수를 첫 번째 매개변수로 두지 않는다.

robot.py

```

class Robot:

    count=0

    def __init__(self, name = 'dummy', legs=0):
        self.name = name
        self.legs=legs
        self.id = Robot.count
        Robot.count += 1

    def talk(self):
        print(f'Hi. I am {self.name} with {self.legs}
legs')
        print(f'My model number is {self.id}')

    def walk(self, step):
        if self.legs >= 2:
            print('done.')
        else:
            print('Sorry. Not enough legs.')

    def run(self, dist, speed = 10):
        print(f'Must go {dist}m at {speed}m/s speed.')
        if self.legs >= 2:
            print('done.')
        else:
            print('Sorry. Not enough legs.')

    @staticmethod
    def info():
        print(f'total number of robots:{Robot.count}')

```

이 예에서 info()함수가 정적 메서드이다. 이 함수는 self 인수를 받지 않기 때문에 개별 객체의 필드를 사용할 수 없지만 개별 객체와는 관련이 없는 클래스 변수(이 예에서는 Robot.count)는 사용할 수 있다. 정적 메서드를 호출하고자 할 때는 객체를 통해서 할 수도 있고 클래스명을 이용할 수도 있으나 후자를 사용하는 것이 정적 메서드임을 명시적으로 나타내므로 더 권장된다.

```
>>> a=Robot()
>>> a.info() # 가능하지만 바람직하지 않음
total number of robots:1
>>> Robot.info() # 이 방법이 바람직함
total number of robots:1
```

일반적인 메서드는 첫 번째 인수가 반드시 self 매개변수로 저장되지만 정적 메서드는 그렇지 않다는 사실에 유의해야 한다. 정적 메서드는 객체와 상관없는 메서드이기 때문이다.

10.7 캡슐화

객체지향 프로그래밍 언어가 지원해야 하는 가장 중요한 특성에 다음 세 가지가 있다.

- 캡슐화 (encapsulation)
- 상속 (inheritance)
- 다형성 (polymorphism)

캡슐화는 객체를 통해서(또는 객체의 ‘외부’에서라고도 한다) 객체의 필드를 직접 접근하여 조작하는 것을 막는 기능을 의미한다. 작은 규모의 프로그램은 별로 중요치 않으나 프로그램의 덩치가 커지고 개발자(사용자)가 많아질수록 객체의 내부를 보호하여 의도치 않은 오동작을 막는 기능은 매우 중요하다. 메서드도 외부 사용자에게 필수적인 것만 개방시키고 그렇지 않고 내부적으로 사용되는 것들은 접근을 막는 것이 바람직하다.

객체를 통해서 접근할 수 없는 속성을 private 속성이라고 하는데, 파이썬에는 접근 지정자 (예를 들면 JAVA의 public, private, protected 등)가 없다. 그 대신 이름 규칙에 의해서 접근 지정을 할 수 있다. 만약 밑줄(_) 하나로 시작하는 속성은 비공개 모드로 객체를 통해서 접근을 하지 않는 것이 바람직하다. 하지만 물리적으로 접근이 막혀있는 것은 아니며 상속도 되므로 객체 외부나 상속된 클래스 내외부에서 여전히 접근이 가능하다. 클래스의 ‘내부’는 클래스 정의부를 의미한다. 만약 밑줄 두 개(__)로 시작하는 속성은 보호 모드로서 외부의 접근과 상속을 받은 클래스 객체의 내부에서도 접근이 금지된다. 이 경우는 객체 내부에서 필드의 이름을 다른 것으로 바꾸는 방식으로 물리적으로 외부의 접근을 막는다.

- 밑줄(_)로 시작하는 속성은 객체를 통해서 접근할 수 있으나 사용하지 않도록 한다.
- 밑줄 두 개로 시작하는 속성은 아예 객체를 통해서 접근할 수 없으며 클래스 정의부(클래스 내부)에서만 사용 가능하다.

간단한 예제를 들어 보자.

```
class Car:
    def __init__(self, name):
        self.name = name
        self._brake = True
        self.__accel = 'on'
```

이 예에서 Car 클래스의 필드는 세 개로 각각 name, _brake, __accel 이다. name과 _brake는 객체 외부에서 자유롭게 접근할 수 있다.

```
>>> my_car=Car('carnival')
>>> vars(my_car)
{'name': 'carnival', '_brake': True, '_Car__accel': 'on'}
>>> my_car._brake
True
>>> my_car.__accel
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    my_car.__accel
AttributeError: 'Car' object has no attribute '__accel'
```

하지만 밑줄 한 개(_)로 시작하는 _brake 필드는 비록 외부에서 접근할 수는 있지만 의도적으로 사용하지 않아야 한다. 하지만 __accel은 내부적으로 _Car__accel 으로 변경되어 mycar.__accel 과 같이 참조할 수 없다. 이렇게 밑줄이 두 개로 시작하는 속성(필드와 메서드)은 외부에서의 직접적인 접근을 막는다.

10.8 상속

10.8.1 상속의 개념

상속(inheritance)은 어떤 클래스에서 작성된 모든 속성을 새로운 클래스에서 다시 작성하지 않고 복사해서 쓸 수 있도록 하는 기능으로서 객체지향 프로그래밍에서 매우 중요한 기능이다. 파이썬은 상속을 위한 문법을 제공하는 데 클래스 정의부에서 상속받을 클래스를 클래스명 뒤의 괄호 안에 명시해 주면 된다.

```
class NameOfClass(BaseClass):
    body_of_class
```

예를 들어서 전 절의 Point 클래스를 상속받는 Point3d 클래스를 정의하고 싶다면 다음과 같이 작성하면 된다.

```
point3d.py

from point import Point

class Point3d(Point):

    def __init__(self,x=0,y=0,z=0):
        super().__init__(x,y) #❶
        self.z=z
```

여기에서 Point3d 클래스는 Point 클래스를 상속받았다. 따라서 Point 클래스의 모든 속성(x, y 필드와 메서드들)을 그대로 복사되어 사용할 수 있으며 추가되는 z 속성만 별도로 작성해주면 되는 것이다. Point를 Point3d의 부모클래스(parent class), Point3d를 Point의 자식클래스(child class)라고 한다. ❶에서 보면 생성자 안에서 super().__init__(x,y) 라고 작성되어 있는데 이는 부모클래스(이 경우 Point 클래스)의 생성자를 호출하는 것이다. Point클래스의 생성자는 x와 y필드를 처리할 수 있으므로 인수로 받은 x,y값을 Point 클래스의 생성자에 넘겨주어 처리한 후 z필드만 Point3d 클래스의 생성자에서 추가로 저장한다.

```
>>> a = Point3d(11,22,33)
>>>a.x
11
>>>a.y
22
>>>a.z
33
```

자식클래스에서는 부모클래스의 모든 메서드를 사용할 수 있으며 이 예의 경우 Point

클래스의 `addx()`, `addy()`, `add()`, `abs()` 메서드들을 모두 사용할 수 있다. 그리고 필요에 따라 새로운 메서드를 추가로 정의하여 사용할 수 있다. `Point` 클래스에는 `addx()`, `addy()` 메서드들이 있으나 `addz()` 메서드는 없으므로 이것을 추가해 보자.

point3d.py

```
from point import Point

class Point3d(Point):

    def __init__(self, x=0, y=0, z=0):
        super().__init__(x,y)
        self.z=z

    def addz(self, dz):
        self.z += dz
```

이제 `Point3d` 클래스 객체는 `addx()`, `addy()`, `addz()` 메서드를 모두 사용할 수 있다. `addx()`, `addy()` 메서드는 `Point` 클래스에서 상속받았고 `addz()` 함수는 새로 추가해서 정의했기 때문이다.

어떤 메서드는 그대로 상속 받아서 사용하는데 문제가 있는 경우가 있다. `Point` 클래스의 `abs()` 함수는 `x`와 `y`만을 고려하여 원점에서 `(x,y)`점까지의 거리를 계산하여 반환하는 것이므로 `Point3d` 클래스와는 맞지 않다. 왜냐면 `Point3d` 객체는 원점에서 `(x,y,z)`점까지의 거리를 계산해야 되기 때문이다. 이 경우 `abs()` 메서드를 재정의해야 한다.

point3d.py

```
from point import Point

class Point3d(Point):

    def __init__(self, x=0, y=0, z=0):
        super().__init__(x,y)
        self.z=z

    def addz(self, dz): #❶
        self.z += dz

    def abs(self): #❷
```



```

        xs = self.x**2
        ys = self.y**2
        zs = self.z**2
        return (xs+ys+zs)**0.5

    def add(self, dx, dy, dz): #❸
        super().add(dx,dy) #❹
        self.z += dz

```

위의 ❷에서 abs()메서드를 Point3d 클래스에서 재정의하였는데 이것은 부모클래스인 Point 클래스의 abs()메서드와 이름이 같다. Point3d 객체가 abs()메서드를 호출한다면 Point3d 클래스에서 재정의된 ❷가 호출된다. 즉, ❷는 Point클래스에서 상속받은 abs()메서드를 새로 덮어썼다고 할 수 있다. 이것을 메서드 오버라이딩(overriding)이라고 한다. (우리말로 표현하면 ‘덮어쓰기’ 정도가 되겠지만 보통은 오버라이딩이라고 그대로 칭한다.) Point클래스의 add()메서드도 x,y필드만을 갱신하므로 Point3d에서는 그대로 사용할 수 없어서 ❸과 같이 오버라이딩했다. ❹에서 보면 super().add(dx,dy) 라고 작성했는데 이는 부모클래스(이 경우 Point 클래스)의 add()메서드를 호출하는 것이다. dx, dy값은 Point클래스에서 처리할 수 있으므로 오버라이딩된 add()메서드에서는 dz값만을 self.z에 더했다. 이런 식으로 오버라이딩하는 메서드 내에서 상속받은 원래의 메서드를 활용할 수도 있다.

```

>>> a=Point3d(11,22,33)
>>> a.add(1,2,3)
>>> a.x
12
>>> a.y
24
>>> a.z
36

```

한 가지 주의할 점은, 클래스 속성 중에 ‘_’로 시작하는 것과 ‘__’로 시작하는 것은 private 속성으로 간주한다고 설명한 바 있는데, 자식클래스 내부에서 전자는 접근이 가능하지만 후자는 가능하지 않다. 즉, “__”로 시작하는 내부 속성은 자식클래스로 상속되지 않는다.

결국 상속은 기존의 클래스를 재활용하는 것이다. 기존의 클래스를 기반으로 새로운 클래스를 작성하는데 새로이 필요한 기능은 추가하고, 변경이 필요한 상속받은 기능은 오버라이딩하여 기능을 확장하는 것이다. 파이썬에서는 두 개 이상의 클래스를

동시에 상속받는 것도 가능하다. 이것을 다중 상속이라고 하는데 이 경우 혼동의 여지가 있으므로 잘 사용하지 않으므로 여기에서는 설명을 생략한다.

10.8.2 상속의 예

여기에서는 상속의 예를 들기 위해서 먼저 Animal이라는 클래스를 다음과 같이 정의한다.

```
class Animal:

    def __init__(self, name, legs=0):
        self.name = name
        self.legs = legs

    def move(self, distance):
        print(f'{self.name} walked {distance} meters')

    def sleep(self, time):
        print(f'{self.name} slept {time} hours.')
```

Animal 클래스의 생성자는 이름과 다리의 개수를 각각 name, legs 매개변수로 받아서 필드에 저장한다. 그리고 move(), sleep()메서드가 있다. 이것을 상속받아서 다음과 같이 Dog, Snake 클래스를 작성한다.

```
class Dog(Animal):

    def __init__(self):
        super().__init__('dog', 4)

    def bark(self, count=1): #❶
        print('bark'*count)

class Snake(Animal):

    def __init__(self):
        super().__init__('snake')

    def move(self, distance): #❷
```

```
print(f'{self.name} crawled {distance} meters')
```

Dog, Snake클래스는 둘 다 Animal 클래스를 상속받았기 때문에 move(), sleep() 메서드를 바로 사용할 수 있다. Dog 클래스는 ❶과 같이 bark()메서드를 새로 추가했고, Snake클래스는 ❷과 같이 move()메서드를 오버라이딩했다. 따라서 Dog의 move()와 Snake의 move()는 서로 다르다.

```
>>> john.fly(10)
dove flew 10 meters
>>> jindo=Dog()
>>> jindo.move(10)
dog walked 10 meters
>>> doksa=Snake()
>>> doksa.move(10)
snake crawled 10 meters
```

위의 실행 예에서 보면 jindo.move() 메서드는 Animal 클래스의 상속받은 원래 메서드이고 doksa.move()메서드는 ❷에서 오버라이딩된(즉, 새로 정의된) 메서드임을 알 수 있다.

이제 Animal을 상속받아 Bird 클래스를 작성하고, Bird 클래스를 상속받아 Dove 클래스를 작성해 보자.

```
class Bird(Animal):

    def __init__(self, name):
        super().__init__(name, 2) #❶

    def fly(self, distance):
        print(f'{self.name} flew {distance} meters')

class Dove(Bird):

    def __init__(self):
        super().__init__('dove')
```

Bird클래스는 ❶에서와 같이 legs필드를 2로 고정시킨다. 그리고 fly()메서드를 추가했다. 따라서 Dove 클래스는 Bird 클래스를 상속했으므로 그 객체는 Bird클래스의 모든 메서드들, 즉, move(), sleep(), fly()를 사용할 수 있으며 legs필드는 자동으로 2가 저장

된다. 이 사실은 vars()내장함수와 dir()내장함수를 이용하면 확인할 수 있다.

```
>>> tom=Dove()
>>> vars(tom)
{'name': 'dove', 'legs': 2}
>>> dir(tom)
[... , 'fly', 'legs', 'move', 'name', 'sleep']
```

전체 코드를 하나로 묶으면 다음과 같다.

animal.py

```
class Animal:

    def __init__(self, name, legs=0):
        self.name = name
        self.legs = legs

    def move(self, distance):
        print(f'{self.name} walked {distance} meters')

    def sleep(self, time):
        print(f'{self.name} slept {time} hours.')

class Dog(Animal):

    def __init__(self):
        super().__init__('dog', 4)

    def bark(self, count=1):
        print('bark'*count)

class Snake(Animal):

    def __init__(self):
        super().__init__('snake')

    def move(self, distance):
        print(f'{self.name} crawled {distance} meters')
```

```

class Bird(Animal):

    def __init__(self, name):
        super().__init__(name, 2)

    def fly(self, distance):
        print(f'{self.name} flew {distance} meters')

class Dove(Bird):

    def __init__(self):
        super().__init__('dove')

```

10.8.3 isinstance() 내장함수

isinstance()는 객체가 어떤 클래스의 인스턴스(instance)인지를 판별하는 내장함수이다. 첫 번째 인수로 객체를, 두 번째 인수로 클래스명을 받는다. 아래 예제들은 전 절의 animal.py 파일이 미리 실행된 후에 입력한다.

```

>>> bulldog = Dog()
>>> isinstance(bulldog, Dog)
True
>>> isinstance(bulldog, Snake)
False

```

bulldog 객체는 Dog 객체이고 Snake 객체가 아닌 것은 매우 명확하다, 그리고 Dog 클래스는 Animal 클래스를 상속받았다. 그렇다면 bulldog은 Animal 객체일까?

```

>>> isinstance(bulldog, Animal)
True

```

정답은 ' 그렇다 '이다.

- 자식클래스의 객체는 부모클래스의 객체로도 간주된다.

Dove 클래스의 경우 Bird를 상속받았고 Bird는 Animal을 상속받았다. 따라서 Dove

객체는 Bird객체이기도 하고 Animal 객체이기도 하다.

```
>>> tom=Dove()
>>> isinstance(tom, Dove)
True
>>> isinstance(tom, Bird)
True
>>> isinstance(tom,Animal)
True
```

이와 같이 자식클래스의 객체는 부모클래스의 객체로도 간주된다는 사실은 잘 숙지하자.

10.9 클래스 예제 : 2x2 행렬 (선택)

독자에 따라서는 행렬(matrix)에 대한 지식이 없을 수도 있으니 그런 경우에는 이 절의 내용은 건너뛰면 된다. 본 절에서는 '2x2 크기의 행렬'이라는 새로운 자료형을 클래스를 이용하여 구현하는 예제를 작성해 보도록 하겠다. 즉, 기본 자료형에 2x2 크기의 행렬을 다루는 것이 없으니 사용자가 직접 클래스 기능을 이용하여 만들어야 한다. 클래스의 이름은 Mat22라고 하겠다.

10.9.1 생성자 작성

먼저 필드는 네 개가 필요한데 2x2 행렬의 요소가 네 개이기 때문이다. 따라서 생성자는 네 개의 인수를 받는 것으로 정하고 다음과 같이 코딩할 수 있을 것이다.

```
mat22.py

class Mat22:

    def __init__(self, a, b, c, d):
        self.e11 = a
        self.e12 = b
        self.e21 = c
        self.e22 = d
```

이것을 mat22.py 파일명으로 적당한 폴더에 저장한 후 **F5**키를 눌러 실행시키면 IDLE 파이썬셸에서 다음과 같이 객체를 생성할 수 있다.

```
>>> a=Mat22(1,2,3,4)
>>> vars(a)
{'e11': 1, 'e12': 2, 'e21': 3, 'e22': 4}
```

이제 2x2 행렬의 네 요소를 저장하는 가장 간단한 자료형인 Mat22를 만들어서 사용할 수 있게 된 것이다.

10.9.2 __repr__() 메서드 작성

두 번째로 저장된 요소들을 보기 위해서 vars() 함수를 사용하지 않고 화면에 행렬 모양으로 직접 표시하는 메서드를 추가하도록 하겠다. 메서드의 이름은 show()로 정하고 이 메서드를 호출하면 화면에 행렬 모양으로 표시되도록 작성한다.

mat22.py

```
class Mat22:

    def __init__(self, a, b, c, d):
        self.e11 = a
        self.e12 = b
        self.e21 = c
        self.e22 = d

    def show(self):
        print(f'[{self.e11}\t{self.e12}]')
        print(f'[{self.e21}\t{self.e22}]')
```

이것을 실행한 후 파이썬셸에서 다음과 같이 실행해 볼 수 있다.

```
>>> a=Mat22(1,2,3,4)
>>> a.show()
[1    2]
[3    4]
>>> b=Mat22(11,22,-33.3,4.5)
>>> b.show()
[11      22]
[-33.3    4.5]
```

이런 방식도 나름대로 괜찮지만 약간 번거롭다. 기왕이면 print()함수를 이용하거나 셸

어서 객체의 이름만 치고 엔터키를 누르면 그 객체의 내용이 show()메서드를 실행했을 때와 동일하게 바로 표시되도록 하면 더 편리하게 사용할 수 있을 것이다. 이럴 때 정의하는 것이 `__repr__()` 매직메서드이다. 이 메서드에서는 `print()`함수에 객체가 이수로 넘어갔을 때 혹은 셸에서 객체의 이름만 입력하고 엔터를 눌렀을 때 표시될 문자열을 반환하면 된다.

mat22.py

```
class Mat22:

    def __init__(self, a, b, c, d):
        self.e11 = a
        self.e12 = b
        self.e21 = c
        self.e22 = d

    def __repr__(self):
        s1= f'[{self.e11}\t{self.e12}]'
        s2= f'[{self.e21}\t{self.e22}]'
        return f'{s1}\n{s2}'
```

이렇게 `__repr__()` 메서드를 정의해 놓으면 다음과 같이 사용할 수 있다.

```
>>> a=Mat22(1,2,3,4)
>>> a
[1  2]
[3  4]
>>> b=Mat22(11,22,-33.3,4.5)
>>> b
[11  22]
[-33.3  4.5]
```

10.9.3 행렬식 값을 계산하는 메서드 작성

여기에서는 행렬의 행렬식(determinant)을 구하는 메서드를 작성해 보자. 2x2행렬의 행렬식을 구하는 공식은 다음과 같다.

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

이 수식을 이용하면 메서드를 쉽게 작성할 수 있다. 메서드 이름은 `det()`로 한다.

```
class Mat22:

    # 다른 메서드들은 이전과 동일

    def det(self):
        return self.e11*self.e22 - self.e12*self.e21
```

이제 `Mat22` 객체의 행렬식 값을 구할 수 있다.

```
>>> m=Mat22(1,2,3,4)
>>> m
[1  2]
[3  4]
>>> m.det()
-2
```

메서드 안에서 다른 메서드를 호출하는 것도 얼마든지 가능하다. 예를 들어 행렬식 값이 0인 행렬을 특이행렬(singular matrix)라고 하는 데 현재 행렬이 특이행렬이면 `True`를 반환하는 메서드 `is_singular()`를 작성하면 다음과 같다.

```
class Mat22:

    # 다른 메서드들은 위와 동일

    def det(self):
        return self.e11*self.e22 - self.e12*self.e21

    def is_singular(self):
        return 0==self.det()
```

실행 예는 다음과 같다.

```
>>> m=Mat22(1,2,1,2)
>>> m.det()
0
```

```
>>> m.is_singular()
True
```

10.9.4 역행렬을 반환하는 메서드 작성

이제 행렬이 특이행렬이면 None을 반환하고, 그렇지 않다면 역행렬을 계산하여 반환하는 메서드를 추가해 보자. 어떤 행렬의 역행렬은 행렬식이 0이라면 존재하지 않고, 그렇지 않다면 다음 공식으로 구할 수 있다.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{D} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

여기서 D는 행렬의 행렬식 값이다. 특이행렬인지 아닌지는 is_singular() 메서드를, 역행렬을 계산할 때 필요한 특성값은 det()메서드를 이용하여 구하면 된다. 예를 들면 다음과 같이 inv()메서드를 추가한다.

```
class Mat22:

    # 다른 메서드들은 위와 동일

    def inv(self):
        if self.is_singular():
            return #❶
        else:
            d = self.det() #❷
            e11, e12 = self.e22/d, -self.e12/d
            e21, e22 = -self.e21/d, self.e11/d
            return Mat22(e11, e12, e21, e22) #❸
```

이 예제에서 눈여겨볼 부분은 ❸번 줄이다. 여기에서 새로운 Mat22객체를 생성한 후 반환했다는 것에 유의하자. 여기에서 알 수 있는 것은 Mat 객체 내부의 메서드에서 새로운 Mat객체를 생성하여 사용하는 것에 전혀 지장이 없다는 점이다. 따라서 이 메서드의 반환값(이 경우 Mat22객체)을 새로운 변수에 저장하는 것도 가능하다.

```
>>> m=Mat22(1,2,3,4)
>>> n=m.inv()
>>> n
```

```

[-2.0, 1.0]
[1.5, -0.5]
>>> a = Mat22(1,2,1,2)
>>> a.inv()
None

```

이 사용 예를 보면 변수 `n`에 `m`객체의 역행렬을 저장하였다. 특히행렬 `a`는 `inv()`메서드의 반환값이 `None`임을 확인할 수 있다.

10.9.5 행렬의 덧셈과 뺄셈

본 절에서는 `Mat22` 객체들끼리의 덧셈을 `+`연산자를 이용하여 구현하는 방법을 알아보겠다. 앞에서 설명한 바와 같이 `__add__()` 매직메서드를 다음과 같이 정의하면 된다.

```

class Mat22:

    # 다른 메서드들은 위와 동일

    def __add__(self, other):
        e11 = self.e11 + other.e11
        e12 = self.e12 + other.e12
        e21 = self.e21 + other.e21
        e22 = self.e22 + other.e22
        return Mat22(e11, e12, e21, e22)

```

이제 덧셈 연산자를 `Mat22`객체에 사용할 수 있다.

```

>>> m1=Mat22(1,2,3,4)
>>> m2=Mat22(5,6,7,8)
>>> n=m1+m2 #❶
>>> n
[6  8]
[10 12]

```

위의 ❶에서 보면 `n=m1+m2`라고 실행되는 데 이 때 `m1`객체의 `__add__()` 매직메서드가 호출된다. `self`는 `m1`이 되고 `other` 매개변수에는 `m2`가 저장되어 메서드가 실행된다. 그리고 `__add__()`메서드에서 반환된 객체가 `m2`에 저장된다.

이 절에서 작성한 전체 프로그램은 다음과 같다.

mat22.py

```
class Mat22:

    def __init__(self, a, b, c, d):
        self.e11 = a
        self.e12 = b
        self.e21 = c
        self.e22 = d

    def __repr__(self):
        s1= f'[{self.e11}\t{self.e12}]'
        s2= f'[{self.e21}\t{self.e22}]'
        return f'{s1}\n{s2}'

    def det(self):
        return self.e11*self.e22 - self.e12*self.e21

    def is_singular(self):
        return 0==self.det()

    def inv(self):
        if self.is_singular(): return #(1)
        else:
            d = self.det() #(2)
            e11, e12 = self.e22/d, -self.e12/d
            e21, e22 = -self.e21/d, self.e11/d
            return Mat22(e11, e12, e21, e22) #(3)

    def __add__(self, other):
        e11 = self.e11 + other.e11
        e12 = self.e12 + other.e12
        e21 = self.e21 + other.e21
        e22 = self.e22 + other.e22
        return Mat22(e11, e12, e21, e22)
```

이 파일을 실행한 이후에 Mat22클래스를 파이썬셸에서 사용할 수 있다.

10.10 클래스 심화(선택)

전 장들에서와 같이 (선택)으로 표시된 이하의 내용은 초보자가 반드시 학습해야 할 필요는 없다고 생각된다. 기본 문법에 어느 정도 숙달되고 더 깊은 내용을 학습하고 싶을 때 한 번 읽어보기를 바란다.

10.10.1 __call__() 매직메서드

객체를 함수처럼 호출이 가능하도록 만들어 주는 매직메서드가 __call__()이다. 다음과 같은 단순한 Dice클래스 예를 보자

```
cdice.py

from random import randint

class Dice:

    def __init__(self, start=1, end=6):
        self._start=start
        self._end=end

    def roll(self):
        return randint(self._start, self._end)
```

이 파일을 실행한 후 파이썬 셸에서 Dice클래스를 이용할 수 있다.

```
>>> dice = Dice()
>>> dice.roll()
1
```

하지만 이것을 조금 수정하여 아예 객체를 함수처럼 호출할 때, 즉 dice()라고 명령을 내릴 때 roll()메서드가 실행되도록 하고 싶다면 roll()메서드를 __call__() 매직메서드로 이름을 바꾸면 된다.

```
cdice2.py

from random import randint
```

```

class Dice:

    def __init__(self, start=1, end=6):
        self._start=start
        self._end=end

    def __call__(self):
        return randint(self._start, self._end)

if __name__ == '__main__': # 테스트 실행 코드
    dice = Dice()
    for k in range(2):
        print(dice()) #❶

```

위에서 테스트 코드 내의 ❶을 보면 dice()라고 객체를 마치 함수처럼 호출했는데 이때 매직메서드 __call__()이 실행되고 설정된 범위의 정수 중 임의로 하나가 반환된다. 이와 같이, __call__()메서드는 객체를 함수처럼 호출했을 경우에 실행되는 메서드이다.

10.10.2 객체를 딕셔너리처럼 동작하게 만드는 매직메서드

여기에서 소개할 매직메서드들은 객체 자체를 딕셔너리처럼 동작하게 만드는 매직메서드들로서 <표 10.11.1>에 정리하였다.

<표 10.11.1> 객체의 딕셔너리처럼 동작하게 만드는 매직메서드들

매직메서드	동작
__getitem__(self, key)	self[key] 실행 시 호출
__setitem__(self, key, value)	self[key]=value 실행 시 호출
__delitem__(self, key)	del self[key] 실행 시 호출

10.10.3 getter와 setter

객체의 필드를 읽거나 설정하는 데 사용되는 메서드를 보통 getter, setter 라고 한다. getter 와 setter 를 사용하는 이유는 객체의 필드를 외부에서 직접 변경하는 것을 제한하기 위한 것으로 캡슐화의 한 방법이다. 외부에서는 객체의 필드를 접근하는 것처럼 보이지만 내부에서는 메서드를 통해서 접근하도록 하는 방식을 지원하는데 이를

속성(property)라고 한다. 간단한 예를 들어보자.

```
point.py

class Point:

    def __init__(self, x, y):
        self._x = x
        self._y = y

    @property
    def x(self): #❶
        return self._x

    @x.setter
    def x(self, v): #❷
        self._x = v

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, v):
        self._y = v
```

이 예를 보면 Point 클래스의 x값과 y값을 내부 속성인 `_x` 와 `_y` 에 저장하였다. 내부 속성은 외부에서는 가능한 접근하지 말아야 하므로 `_x` 값에 접근하여 읽고 쓰는 기능을 수행하는 두 개의 메서드를 정의한 후 각각 `@property` 와 `@x.setter` 라는 데코레이터를 붙여서 정의하였다.

```
>>> a=Point(11, 22)
>>> a.x
11
>>> a.x = 12
>>> a.x
12
```

여기서 `a.x` 와 같이 값을 읽는 경우에는 ❶함수가, `a.x=12` 와 같이 값을 변경하는 경우에는 ❷함수가 호출된다. 필드 `y`에 대해서도 같은 방식으로 동작된다.

만약 x나 y에 음수가 들어오는 경우 그 값 대신 0을 대입하고자 한다면 setter에 적당한 코드를 작성하면 된다.

```
point.py

class Point:

    def __init__(self, x, y):
        self._x = x
        self._y = y

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, x):
        self._x = 0 if x<0 else x

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, y):
        self._y = 0 if y<0 else y
```

이렇게 하면 x나 y에 음수를 대입하려고 하면 0이 대입된다.

```
>>> a = Point(11,22)
>>> a.x=-1
>>> a.x
0
```

이런 식으로 속성(property)을 이용하면 유효하지 않은 값으로 설정하려고 하는 경우에 적절한 처리를 수행할 수 있다.

10.10.4 추상클래스

자식 클래스가 부모 클래스를 상속하는 경우 부모 클래스에서는 함수 원형만 선언

해 놓고 실제 구현은 자식 클래스에서 해야되는 경우가 있다. 상속받은 메서드의 본체를 자식 클래스가 반드시 구현하도록 강제하는 간단한 방법은 부모클래스의 메서드 원형에서 단순히 `NotImplementedError` 를 발생시키도록 하면 된다.

```
from math import pi

class Shape:
    def get_area(self):
        raise NotImplementedError

class Circle(Shape):
    def __init__(self, r):
        self.r = r

    def get_area(self):
        return pi*self.r**2

class Square(Shape):
    def __init__(self, side):
        self.side=side
```

이 예제에서 `Square` 클래스의 객체가 `get_area()` 메서드를 호출하면 부모클래스인 `Shape`의 그것이 호출되어 예외가 발생하게 된다. 즉, 실행할 때 비로서 예외가 발생하게 된다. 이 방식은 자식클래스가 `get_area()` 메서드를 구현하지 않으면 부모 클래스의 그것을 호출하므로 (비록 그 안에서 예외가 발생하긴 하지만) 자식클래스를 강제하는 방식은 아니다.

추상 클래스(`abstract class`)란 이것을 상속 받은 자식 클래스에서 반드시 구현시킬 메서드들의 선언이 있는 클래스를 말한다. 파이썬에서는 추상클래스의 기능을 제공하는 `abc`모듈이 있다. 추상클래스는 그것을 상속받을 클래스의 기본적인 뼈대를 제공하는 역할을 한다.

shape.py

```
from abc import ABCMeta, abstractmethod

class Shape(metaclass=ABCMeta):
    @abstractmethod
    def get_area(self): pass
```

```

class Circle(Shape):
    def __init__(self, r):
        self.r = r

    def get_area(self):
        return 3.14*self.r**2

class Square(Shape):
    def __init__(self, side):
        self.side=side

```

이 예제에서 Square 클래스는 get_area() 메서드가 구현되지 않았으므로 Square 객체를 생성하려고 하면 다음과 같이 예외(오류)가 발생한다.

```

>>> s=Square(3)
-----
TypeError                                Traceback (most recent
call last)
<ipython-input-5-9195b78d6c71> in <module>()
----> 1 s=Square(3)

TypeError: Can't instantiate abstract class Square with
abstract methods get_area

```

따라서 Shape 클래스를 상속받는 모든 자식 클래스는 반드시 get_area() 메서드를 구현하도록 강제할 수 있다.

제 11 장 모듈

파이썬은 표준 라이브러리에 들어 있는 모듈을 사용할 수도 있고 자신이 직접 만드는 것도 가능하다. 자신이 만든 함수나 클래스를 모듈로 만들어 놓으면 다른 파일에서 사용하기가 용이해진다. 또 모듈들을 패키지라는 자료형으로 모아서 코드를 보다 쉽게 재사용할 수 있다. 파이썬셸에서 작성한 명령들은 셸을 종료하면 사라지지만 이 명령들을 파일(script file)로 작성해서 저장해 두면 몇 번이고 반복해서 사용할 수 있다. 파이썬에서는 이 스크립트 파일을 그대로 모듈로 이용할 수 있다. 자주 사용하는 함수 등을 파일로 작성해 두고, 필요에 따라 import하여 그 안의 함수를 이용할 수 있다.

11.1 사용자 모듈

파일명에서 '.py' 확장자 바로 앞부분이 모듈명이 된다. 아래와 같은 파일을 작성해서 파일명을 'mymodule.py'으로 작성해 보자.

```
mymodule.py

import random

phi=1.618

def foo():
    print('hello foo()')

class Car:
    pass

print('This is mymodule.py') #❶

if __name__ == '__main__': #❷
    print('This is the code block')
    print(f'dice roll:{random.randint(1,6)}')
```

모듈로 사용할 스크립트 파일의 파일명은 다음과 같은 규칙을 지켜야한다.

- 숫자로 시작하지 않고 중간에 점(.)을 포함하지 않는다.

- 예약어, 내장 함수, 표준라이브러리의 모듈명은 피한다.
- 알파벳은 소문자만 사용한다.

이제 이 파일을 import하려면 파일명을 이용하면 된다.

```
>>> import mymodule
This is mymodule.py
```

만약 파이썬셸의 폴더가 mymodule.py 파일이 저장된 곳이 아니라면 import하기 전에 os.chdir()함수를 이용하여 mymodule.py 파일이 위치한 폴더로 이동해야 한다. 예를 들어 mymodule.py 파일의 전체 경로가 'e:/coding/mymodul.py'라면 다음과 같이 하면 된다.

```
>>> import os
>>> os.chdir('e:/coding')
>>> import mymodule
This is mymodule.py
```

그러면 mymodule.py 파일을 읽어오게 되는데 이때 최상위의 블록, 즉 들여쓰기되어 있지 않은 위치에 정의되어 있는 명령을 실행한다. 실행문이 있다면 실행하고 변수, 함수, 클래스가 정의되어 있다면 그것들을 생성한다. 이때 변수, 함수, 클래스 등은 그 모듈에 속한 객체로 정의된다. 따라서 위의 경우 import한 직후에 문자열 'This is mymodule.py'가 화면에 표시되는데 ❶이 실행되었기 때문이다. 그리고 mymodule이 import될 때 그 안의 변수 phi, 함수 foo(), 클래스 Car가 생성되어 이 모듈의 부속 객체로 존재하게 된다.

```
>>> import mymodule
This is mymodule.py
>>> mymodule.phi
1.618
>>> mymodule.foo()
hello foo()
>>> carnival = mymodule.Car()
>>> mymodule.random.random()
0.057485625194610424
```

위와 같이 변수 phi를 참조할 수 있고, foo()도 호출할 수 있으며 Car클래스의 객체도

생성할 수 있다. 그리고 mymodule.py 안에서 random모듈을 import하는데, 모듈 안에서 import한 외부 모듈도 역시 그 모듈에 속한 하위 객체가 된다. 위에서 mymodule에서 import한 random모듈을 사용하였다.

만약 mymodule.py에서 다른 것은 필요 없고 Car 클래스만 사용하고 싶다면 from명령어로 그것만 읽어들이면 된다

```
>>> from mymodule import Car
>>> carnival = Car()
```

그러면 위와 같이 Car클래스를 mymodule이라는 이름도 필요 없이 바로 사용할 수 있다. 변수 phi와 foo()함수 두 개만 읽어들이려면 다음과 같이 한다.

```
>>> from mymodule import phi, foo
>>> a=phi
>>> a
1.618
>>> foo()
hello foo()
```

이렇게 from으로 읽어들이는 객체들은 모듈명도 필요 없이 바로 현재의 작업 공간의 객체가 된다.

mymodule.py의 마지막에는 if 블록 ❷가 있는데 이것은 최상위 명령어인데 이 블록은 실행되지 않는다. 왜냐면 mymodule.py가 import 명령에 의해서 실행될 때에는 `__name__ == '__main__'` 조건문이 거짓이 되기 때문이다. 반면 mymodule.py가 실행이 될 때는 (IDLE편집기 상에서는 **Enter**키를 누르면 실행된다) 이 조건문이 참이 되어 if 블록 ❷가 실행된다. (만약 import 되었다면 `__name__` 변수는 모듈의 이름을 문자열로 갖는다.)

- `__name__` 변수
 - 파일이 실행되었다면 `'__main__'` 문자열을 갖는다.
 - 파일이 import 되었다면 모듈의 이름을 문자열로 갖는다.
- `__file__` 변수
 - 실행 혹은 import된 파일의 전체 경로를 문자열로 갖는다.

IDLE편집기 밖에서 실행시키려면 mymodule.py 파일의 폴더로 찾아가서 다음과 같이 실행하면 된다. 다음은 윈도우의 커맨드창에서 실행한 결과이다.

```
PS C:\Users\sales> cd e:/coding
PS E:\coding> python mymodule.py
This is mymodule.py
This is the code block
dice roll:4
PS E:\coding>
```

이와 같이 if블록 ❷에는 모듈의 기능을 테스트하기 위한 테스트 코드를 작성해 두는 용도로도 사용할 수 있다. 모듈이 제대로 동작하는지를 확인하고 싶을 때는 모듈을 import하는 것이 아니라 파일을 직접 실행해 본 후 오류가 없다면 모듈을 수정하지 않고 그대로 import해서 사용하면 된다.

11.2 os 모듈

이전 장에서 calendar 모듈과 random모듈은 설명한 바가 있으므로 여기에서는 이들 외에 사용 빈도가 높은 모듈들과 포함된 객체들에 대해서 설명하도록 하겠다. 표준 라이브러리에 포함된 모듈들은 외부에서 설치하는 과정 없이 바로 import 하여 사용할 수 있다.

파이썬은 운영체제 위에서 돌아간다. os는 운영체제(operating system)의 준말로 os모듈은 운영체제에서 제공하는 정보를 제공하거나 운영체제의 명령을 수행할 수 있는 방법을 제공한다.

<표 11.2.1> os모듈의 함수와 변수

os모듈 변수/함수	설명
os.getcwd()	현재 폴더(경로)를 구한다.
os.chdir()	폴더를 변경한다.
os.name	운영체제 'nt', 'posix', 'ce', 'java' 중 하나
os.system('command')	운영체제의 command를 실행한다.
os.listdir('path')	path내의 파일과 폴더를 리스트로 반환한다. 인수가 생략되면 현재 경로에 대해서 수행한다.
os.mkdir()	폴더를 생성한다.
os.remove()	파일을 삭제한다.
os.rmdir()	폴더를 삭제한다.

몇 가지 용례를 들면 다음과 같다.

```
>>> import os
>>> os.name
```

```
'nt'
>>> os.getcwd()
'C:\\Users\\sales\\AppData\\Local\\Programs\\Python\\P
ython39-32'
>>> os.chdir('e:\\coding')
>>> os.getcwd()
'e:\\coding'
>>> os.listdir()
['hello.py', 'mu1.py', 'mymodule.py', 'py',
'pyautogui', '__pycache__']
```

여기서 os,name 변수는 윈도우의 경우 'nt' 문자열을 갖는다. 그리고 윈도우에서 폴더의 구분을 '\' 문자로 한다는 사실도 알아두어야 한다.(리눅스나 맥은 '/' 문자로 폴더를 구분한다.) 따라서 E:드라이브 밑의 coding 폴더는 'e:\\coding'이라고 해야 한다.

11.3 sys 모듈

sys는 파이썬과 관련된 정보를 얻고 설정을 조작하기 위한 함수를 포함하고 있는 모듈이다.

<표 11.3.1> os모듈의 함수와 변수

sys모듈 변수/함수	설명
sys.argv	스크립트 실행시 주어진 명령행 인수
sys.version	파이썬 버전
sys.path	시스템의 경로 정보를 담은 리스트

sys모듈에서 가장 사용빈도가 높은 변수는 sys.argv인데 이것은 스크립트 실행시 주어진 명령행 인수들을 문자열로 요소로된 리스트이다.

```
sysargv.py

import sys

print(sys.argv)

if len(sys.argv)>1:
    if sys.argv[1]=='-f':
        print('force option enabled.')
        print(f'for {sys.argv[2]}')
```

```
elif sys.argv[1]=='-r':
    print(f'remove {sys.argv[2]}'.')
else:
    print('no options.')
```

그리고 윈도우의 커맨드창에서 다음과 같이 실행시켰다고 가정하자.

```
PS E:\coding> python sysargv.py -r all
['sysargv.py', '-r', 'all'] #❶
remove all.
PS E:\coding>
```

위의 ❶이 `print(sys.argv)` 실행문의 결과이다. `sys.argv` 리스트의 0번 요소가 실행되는 파일명, 1번 요소가 ' -r', 2번 요소가 'all' 임을 알 수 있다. 이와 같이 스크립트파일을 실행할 때 같이 주어지는 공백문자로 구분된 여분의 옵션들을 `sys.argv` 리스트에서 확인할 수 있고, 실행파일 안에서는 그 옵션에 따라서 정해진 동작을 하도록 코딩할 수 있다.

11.4 datetime 모듈

`datetime`은 날짜와 시간을 데이터로 표현하고 싶을 때 사용하는 모듈이다. 뒤에 나올 `time`모듈에 비해서 취급할 수 있는 날짜와 시간의 범위가 훨씬 더 크고 날짜와 시간을 사용한 연산 및 비교를 더 쉽게 할 수 있다. 이 절 안의 예제들은 모두 다음과 같이 `datetime`모듈이 `import` 되었다고 가정한다.

```
>>> import datetime
```

11.4.1 datetime.date 클래스

`datetime`에는 `date`라는 클래스가 있는데 이것의 `today()`라는 정적함수를 이용하면 오늘 날짜를 객체로 생성할 수 있다.

```
>>> this_day=datetime.date.today()
>>> this_day
datetime.date(2021, 6, 29)
```



```
>>> this_day.year
2021
>>> this_day.month
6
>>> this_day.day
29
```

위에서 보듯이 date객체는 year, month, day라는 필드가 있어서 각각 년, 월, 일에 해당하는 정보를 가지고 있다. 특정한 날짜를 date클래스 생성자의 인수로 주어서 객체를 생성할 수도 있다. 이때 인수는 년, 월, 일 순으로 주어야 한다.

```
>>> that_day=datetime.date(2021,1,1)
>>> that_day
datetime.date(2021, 1, 1)
```

date객체의 weekday()메서드를 이용하면 요일의 번호(월요일이 0, 화요일이 1, ... 일요일이 6)를 얻을 수 있다.

```
>>> that_day=datetime.date(2021,1,1)
>>> that_day.weekday()
4
```

또한 isoweekday()메서드는 월요일이 1이고 일요일이 7인 요일 번호를 얻을 수 있다. date객체끼리는 뺄셈과 비교연산이 가능한데 더 미래의 날짜일수록 더 크다고 판별된다. 뺄셈의 결과는 timedelta클래스의 객체가 되는데 이는 뒤에서 설명한다.

11.4.2 datetime.time 클래스

datetime에는 time클래스도 있는데 이것은 시간을 취급하는 클래스이다. 이것의 객체는 생성자에 시간(hour), 분(minute), 초(second), 마이크로초(microsecond) 순서로 인수들을 입력해야 하는 초와 밀리초는 생략할 수 있다.

```
>>> now = datetime.time(9,50)
>>> now.hour, now.minute, now.second
(9, 50, 0)
```

time클래스의 객체는 위와 같이 hour, minute, second 필드가 있어서 각각 설정된 값

을 참조할 수 있다. date 객체와 마찬가지로 time 객체 간 비교연산은 가능하지만 뺄셈은 불가능하다.

11.4.3 datetime.datetime클래스

datetime.datetime클래스는 날짜와 시간을 동시에 취급할 수 있는 클래스이다. 현재 날짜와 시간을 얻는 now() 정적메서드가 있다.

```
>>> now = datetime.datetime.now()
>>> now
datetime.datetime(2021, 6, 29, 10, 23, 9, 628636)
>>> now.year, now.month, now.day
(2021, 6, 29)
>>> now.hour, now.minute, now.second, now.microsecond
(10, 23, 9, 628636)
```

위와 같이 year, month, day, hour, minute, second, microsecond 필드를 가지고 있어서 해당하는 숫자 정보를 저장하고 있다. 또는 생성자 인수로 연, 월, 일, 시간, 분, 초, 밀리초를 순서대로 (시간 이후는 생략 가능) 넘겨주면 datetime.datetime객체가 생성된다.

```
>>> past = datetime.datetime(2003,3,1,13)
>>> past
datetime.datetime(2003, 3, 1, 13, 0)
```

datetime.datetime객체의 date()메서드를 이용하면 날짜 정보만 갖는 date객체를 얻을 수 있고, time()메서드를 이용하면 시간 정보만 갖는 time객체를 얻을 수 있다.

11.4.4 datetime.timedelta클래스

timedelta클래스는 날짜와 시간의 차를 취급하기 위한 특수한 클래스이다. date객체끼리 뺄셈을 하면 그 간격(일 수)을 확인할 수 있다.

```
>>> date1=datetime.date.today()
>>> date2=datetime.date(2021,1,1)
>>> dt1 = date1-date2 #❶
>>> print(dt1)
179 days, 0:00:00
```

위의 ❶에서 date 객체끼리 뺄셈을 수행했는데 그 결과는 timedelta객체이다. 따라서 변수 dt1에는 timedelta객체가 저장된다. datetime.datetime 객체들끼리의 뺄셈을 수행해도 timedelta객체가 얻어진다.

```
>>> time1=datetime.datetime.now()
>>> time2=datetime.datetime(2021,1,1)
>>> dt2=time1-time2
>>> print(dt2)
179 days, 17:15:23.716422 #❶
>>> dt2.days
179
>>> dt2.seconds
62123
>>> dt2.microseconds
716422 #❷
>>> dt2.total_seconds()
15527723.716422
```

위의 예는 2021년 1월1일 0시부터 현재까지의 시간 간격을 구하는 것이다. 위에서 dt2의 days, seconds, microseconds 멤버변수들은 시간 간격에 대한 각각의 정보를 가지고 있다. 즉, 위의 ❶에서 17시간15분23초는 716422초(❷)가 되는 것이다. 전체 시간 간격은 days+seconds+microseconds와 같다. 전체 시간 간격이 모두 몇 초인지를 계산하려면 위와 같이 total_seconds()메서드를 호출하면 된다.

<표 11.4.1> timedelta객체의 멤버변수와 메서드

timedelta객체의 필드	설명
days	일의 수
seconds	초의 수
microseconds	마이크로초의 수
total_seconds()	전체 시간간격(days+seconds+mocroseconds)을 초 수로 변환하여 반환
date()	날짜 정보로 date클래스의 객체 생성
time()	시간 정보로 time클래스의 객체 생성

date 혹은 datetime 객체와 deltatime 객체와의 덧셈도 가능하다. 만약 timedelta 객체를 생성자로부터 바로 생성하려면 days, seconds, microseconds 키워드 인수를 넘겨주면 된다.

```
>>> d1=datetime.date(2021,6,29)
>>> td=datetime.timedelta(days=100)
>>> d2=d1+td #❶
>>> print(d2)
2021-10-07
```

❶에서 보면 date객체와 timedelta객체의 덧셈이 이루어 졌는데 그 결과값은 date객체이다. 유사하게 datetime객체와 timedelta객체의 덧셈의 결과는 datetime이 된다.

```
>>> now=datetime.datetime.now()
>>> td=datetime.timedelta(seconds=1000)
>>> after=now+td
>>> print(after)
2021-06-29 17:04:29.971013
```

timedelta객체들끼리는 덧셈과 뺄셈이 가능하고 timedelta객체에 정수를 곱하는 연산도 가능하다.

```
>>> td1 = datetime.timedelta(microseconds=200)
>>> td2= datetime.timedelta(days=1, seconds=20)
>>> td3 = td1+td2
>>> print(td3)
1 day, 0:00:20.000200
>>> td4=td2-td1
>>> print(td4)
1 day, 0:00:19.999800
>>> td5=td2*3
>>> print(td5)
3 days, 0:01:00
```

그리고 timedelta객체에 //연산을 취하는 것도 가능하다.

11.4.5 datetime 클래스의 strftime()메서드

datetime객체의 strftime()메서드를 이용하면 날짜와 시간을 원하는 내용의 문자열로 만들 수 있다.

dt.strftime(형식문자열)

여기에서 dt는 datetime 클래스(혹은 date나 time클래스도 가능하다)의 객체이고 형식 문자열에 포함될 수 있는 지정자는 다음 <표 11.4.2>와 같다.

<표 11.4.2> strftime()메서드의 형식 지정자

구분	지정자	설명	예
연도	%y	연도에서 끝 두 문자	2021년이라면 '21'
	%Y	모든 연도	'2021'
월	%b	월의 축약명	6월이면 'Jun'
	%B	월의 영문 문자열	6월이면 'June'
	%m	월을 두 자리 숫자로	6월이면 '06'
일	%d	두 자리의 일	9일이면 '09'
시간	%H	24시간 기준 시('00' ~ '23')	오후 1시면 '13'
	%I	12시간 기준 시('00' ~ '12')	오후 1시면 '01'
	%p	오전이면 'AM' 오후면 'PM'	
분	%M	분을 두 자리 숫자로	
초	%S	초를 두 자리 숫자로	
요일	%a	축약된 요일명	수요일은 'Wed'
	%A	축약되지 않는 요일명	수요일은 'Wednesday'
	%w	월요일을 1로 한 요일 번호	수요일은 '3'
종합	%x	'mm/dd/yy'형식의 날짜	
정보	%X	'HH:MM:SS'형식의 시간	

몇 가지 사용 예를 들면 다음과 같다.

```
>>> from datetime import *
>>> now = datetime.now()
>>> now.strftime('%Y년 %m월 %d일(%A)')
'2021년 06월 30일(wednesday)'
>>> d1 = date(now.year,1,1)
>>> days = (now.date()-d1).days+1 #❶
>>> now.strftime(f'%Y/%B/%d(%a) 는 %Y년의 {days}번째 날이
다')
'2021/06/30(wed) 는 2021년의 181번째 날이다'
>>> now.strftime('지금은 %p %I시 %M분이다')
'지금은 AM 07시 16분이다'
```

date객체는 date객체와 뺄셈을 수행해야 한다. 따라서 ❶에서 dtm.date()메서드를 사용해서 datetime객체인 dtm에서 date객체를 얻어낸 후 d1객체와 뺄셈을 수행했다.

11.5 time 모듈

time모듈의 중요한 기능은 datetime모듈에서도 쉽게 할 수 있다. 단, time모듈의 sleep()이라는 함수는 주어진 초시간 동안 아무 일도 하지 않고 대기하는 기능을 하는데 이것은 datetime모듈에는 없는 것이다.

countdown.py

```
import time
count_down=10
for _ in range(11):
    if count_down == 0:
        print('Fire')
    else:
        print(count_down)
    time.sleep(1)
    count_down-=1
```

실행결과

```
10
9
8
7
6
5
4
3
2
1
Fire
```

위 countdown.py예제는 10부터 1까지 1초 간격으로 카운트다운을 수행하고 마지막 0이 되면 숫자 대신 'Fire'라는 문자열을 출력한다.

11.6 math 모듈

math모듈은 수학 계산과 관련 함수와 상수를 제공하는 모듈이다. 원주율같은 수학 상수들이 정의되어 있으며 삼각함수와 같은 수학 관련 함수들을 가지고 있다.

〈표 11.6.1〉 math모듈의 변수와 함수

변수/함수	설명
pi	원주율
e	자연상수
inf	무한대
ceil(x) floor(x)	x보다 크거나 같은 가장 작은 정수값 (천장값, 올림) x보다 작거나 같은 가장 큰 정수값 (바닥값, 내림)
gcd(*ints) lcm(*ints)	최대공약수 최소공배수
sin(x) cos(x) tan(x)	삼각함수 (x의 단위는 라디안)
sinh(x) cosh(x) tanh(x)	쌍곡선함수
asin(x) acos(x) atan(x) atan2(y,x)	역삼각함수 (arcsin 함수) 역삼각함수 (arccos 함수) 역삼각함수 (arctan 함수) y/x의 arctan 함수 (x가 0인 경우에도 계산 가능)
asinh(x) acosh(x) atanh(x)	역쌍곡선함수
degrees(x) radians(x)	라디안 x를 도(degree)로 변환 도(degree) x를 라디안으로 변환
log(x[,base]) log2(x) log10(x)	x의 자연로그. (두 번째 인수로 밑을 줄 수 있다.) 밑이 2인 로그 (log(x,2)보다 더 정확함) 밑이 10인 로그 (log(x,10)보다 더 정확함)
exp(x)	자연상수의 x거듭제곱
pow(x,y)	x의 y거듭제곱
sqrt(x)	x의 제곱근
fabs(x)	절대값

참고로 반올림을 하기 위해서는 파이썬 내장함수인 round()를 이용하면 된다.

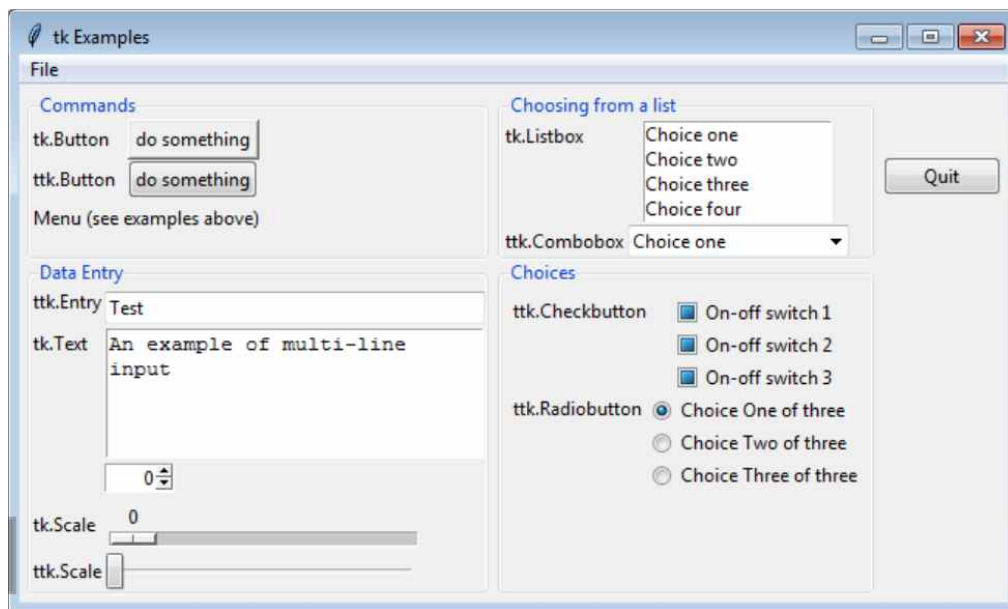
11.7 패키지 (선택)

11장 연습문제

pb11-1: $x = \frac{3\pi}{4}$ 일 때 $\sin^2 x + \cos x$ 값을 구하라.

제 12 장 Tkinter로 GUI 코딩

파이썬 표준 라이브러리에 tkinter 패키지가 포함되어 있는데 이것을 이용하면 그래픽 사용자 인터페이스(graphical user interface, 이하 GUI) 프로그래밍을 쉽게 할 수 있다. tkinter는 표준 라이브러리에 포함되기 때문에 별도로 설치하는 과정이 필요 없다. <그림 12.1>은 tkinter를 이용하여 만든 GUI와 다양한 위젯(widget)의 예시를 보여주고 있다.



<그림 12.1> tkinter를 이용한 GUI와 다양한 위젯(widget) 예시

12.1 첫 번째 예제와 동작 원리

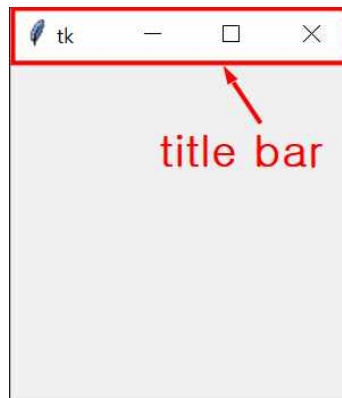
첫 번째 프로그램으로 다음을 편집기에서 작성한 후 실행시켜 보자.

```
gui01.py
import tkinter as tk #❶

gui=tk.Tk() #❷

gui.mainloop() #❸
```

그러면 다음과 같은 GUI가 생성될 것이다.



〈그림 12.2〉 생성된 GUI

생성된 GUI은 기본적인 기능을 다 가지고 있다. 크기가 조절되고 축소, 확대, 종료 세 개의 버튼도 타이틀바(title bar)에 가지고 있다.

gui01.py 파일의 ❶번 줄은 tkinter 패키지를 import하는데 tk라는 이름으로 사용하겠다는 것이다. tk라는 별칭을 사용하므로 tkinter라는 긴 이름을 매번 사용할 필요가 없는데 이러한 축약은 파이썬 프로그램에서 자주 이용하게 된다. ❷번 줄에서는 tkinter패키지의 Tk클래스 객체를 생성해서 gui라는 변수에 저장하는 것이다. ❸번 줄에서는 Tk 객체의 mainloop() 메서드를 호출해서 생성된 창(window)의 이벤트 순환문을 시작한다. 이 mainloop()메서드가 호출되기 전에는 GUI가 화면에 표시되지 않는다.

GUI에서 이름(title)과 축소/확대/종료 버튼이 있는 맨 윗부분을 타이틀바(title bar)라고 칭한다. (〈그림 12.2〉 참조) 타이틀바에 표시되는 GUI의 문자열을 바꾸려면 title() 메서드를 호출하고 인수로 원하는 문자열을 넘기면 된다.

gui01b.py

```
import tkinter as tk

gui=tk.Tk()
gui.title('tkinter GUI') #❶

gui.mainloop()
```

그러면 타이틀바에 주어진 문자열이 표시된다.

만약 생성 시 GUI의 크기를 설정하고 싶다면 geometry()메서드를 호출하면 된다. 예를 들어 400x200 픽셀(pixel)크기로 생성하고 싶다면

```
gui.geometry('400x200')
```

이라고 문자열 인수를 넘겨주면 된다.



<그림 12.3> gui01b.py (타이틀 변경)

그리고 가로나 세로방향의 크기조절을 불가능하게 하려면 `resizable()`메서드를 호출하면 된다.

```
gui.resizable(True,False) # 가로 방향만 크기 조절
gui.resizable(False,True) # 세로 방향만 크기 조절
gui.resizable(False,False) # 가로/세로 모두 크기 조절 불가
```

이렇게 하면 GUI의 크기를 원하는 방향만 조절할 수 있게끔 한다던가 혹은 양방향 모두 크기 조절이 불가능하도록 할 수 있다.

12.2 위젯

위젯(widget)이란 GUI 안에 배치할 수 있는 제어 요소를 의미한다. 예를 들면 버튼(button), 라벨(label), 입력창, 콤보박스, 라디오버튼 등등 여러 가지가 있다.

- 위젯(widget)
 - GUI에 배치할 수 있는 제어 요소
 - 버튼, 라벨, 입력창, 콤보박스, 라디오버튼 등

각각의 위젯은 클래스로 제공이 되는데 이 클래스들은 tkinter 패키지 안의 ttk라는 모듈에 미리 작성되어 있다. 따라서 위젯을 생성하기 위해서는 tkinter 패키지에서 ttk모듈을 import해야 한다. tkinter 패키지 자체도 GUI를 생성하기 위해서는 여전히 임포트해야되고 추가로 위젯을 생성하기 위해서는 ttk모듈도 임포트해야 한다.

12.2.1 라벨

본 절에서는 그 중 라벨을 추가하는 방법에 대해서 알아보도록 하자. 라벨은 원하는 문자열을 표시해주는 위젯이다.

```
gui02.py

import tkinter as tk
from tkinter import ttk #❶

gui=tk.Tk()
gui.title('tkinter GUI')

label = ttk.Label(gui, text='Hello tkinter') #❷
label.pack() #❸

win.mainloop()
```

❶에서 ttk모듈을 import하고 ❷번에서 ttk.Label 클래스의 객체를 생성하여 label이라는 변수에 대입했다. 이 때 ttk.Label생성자의 첫 번째 표준 매개변수에 라벨을 추가할 Tk객체 (여기에서는 gui변수)를 넘기고 text 키워드 인수에는 그 라벨에 표시할 문자열을 지정해 주면 된다. 라벨 객체를 생성했다고 그것이 바로 GUI에 표시되지는 않고 배치를 잡아주어야만 실제로 표시된다. 그 역할을 해주는 것이 ❸의 pack()메서드이다. 일단 이 프로그램을 실행하면 다음 그림과 같은 결과를 보인다.



<그림 12.4>gui02.py

GUI의 크기가 더 작아졌는데 이는 위젯을 추가했기 때문이다. 위젯을 추가하면 그것을 표시하기 위해서 필요한 크기를 사용하도록 GUI의 크기가 최적화된단. 하지만 GUI의 크기조절은 여전히 가능하며, 크기 조절을 불가능하게 하려면 앞에서 설명한 바 대로 resizable()메서드를 호출해야 한다.

라벨은 얼마든지 추가할 수 있다.

```
gui2b.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
```

```

gui.title('tkinter GUI')

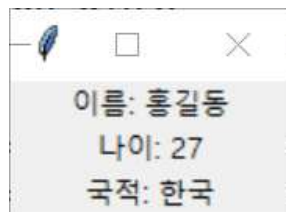
label1 = ttk.Label(gui, text='이름: 홍길동')
label1.pack()

label2 = ttk.Label(gui, text='나이: 27')
label2.pack()

label3 = ttk.Label(gui, text='국적: 한국')
label3.pack()

gui.mainloop()

```



〈그림 12.5〉 gui02b

이 예에서 보듯이 각각의 라벨 객체는 서로 다른 변수(label1, label2, label3)에 저장하여 구별해야 한다.

ttk.Label 생성자의 키워드 매개변수를 <표 12.1>에 설명하였다. 이 표에 나열된 옵션은 label객체를 생성한 이후에도 configure()메서드의 키워드 인수로 넘겨서 재설정할 수 있다.

〈표 12.1〉 ttk.Label 생성자/configure()메서드의 키워드 매개변수

키워드 인수	설명	값
text	표시할 문자열	
width	라벨의 너비(문자 수)	
anchor	width가 문자 수보다 클 경우의 문자열 정렬 방식	tk.LEFT, tk.RIGHT, tk.CENTER
foreground	문자의 색을 문자열로 지정()	#rrggbb형식의 문자열 혹은 white, black, red, green, blue, cyan, yellow, magenta
background	배경의 색을 문자열로 지정	

12.2.2 버튼

GUI에 버튼(button)을 추가하려면 ttk.Button클래스의 객체를 생성하면 된다.

gui03.py

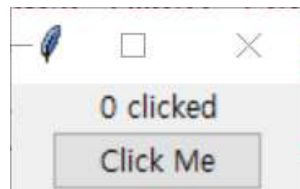
```
import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

label = ttk.Label(gui, text='Click button')
label.pack()

button = ttk.Button(gui, text='Click Me') #❶
button.pack()

gui.mainloop()
```



<그림 12.6> gui03

gui02.py를 저장하고 실행하면 라벨 밑에 버튼이 생성되었음을 알 수 있다. ❶에서 보면 ttk.Button 클래스의 생성자도 ttk.Label의 생성자와 마찬가지로 gui변수와 text 키워드 인수를 넘기면 된다는 것을 알 수 있다. text 인수에는 버튼에 표시할 문자열을 지정해 주면 된다.

하지만 이렇게만 해서는 버튼을 클릭해도 아무일도 일어나지 않는다. 버튼을 클릭했을 때 어떤 동작이 일어나게 하려면 ttk.Button 클래스 생성자의 command 인수를 이용하면 된다. 이 command에 버튼을 클릭했을 때 실행되는 함수를 지정해 줄 수 있다.

gui03b.py

```
import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')
```

```

label = ttk.Label(gui, text='Click button')
label.pack()

def on_click(): #❶
    label.configure(text='Button clicked!')

button = ttk.Button(
    gui,
    text='Click Me',
    command=on_click #❷
)

button.pack()

gui.mainloop()

```

gui03b.py의 ❶를 보면 ttk.Button 객체를 생성하기 전에 on_click()이라는 함수를 정의하고 있는데 이 안에서 label.configure()메서드를 호출하여 label객체의 문자열을 바꾸도록 하고 있다. 그리고 ❷에서 Button 객체 생성자의 command 키워드 인수에 이 함수를 지정해 두면 버튼이 클릭될 때마다 on_click()함수가 호출되게 된다.

만약 클릭하는 횟수를 세고 싶다면 다음과 같이 프로그램을 변경하면 된다.

```

gui03c.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

label = ttk.Label(gui, text='Click button')
label.pack()

count=[0] #❶
def on_click():
    count[0]+=1 #❷
    label.configure(text=f'{count[0]} clicked') #❸

button = ttk.Button(
    gui,

```

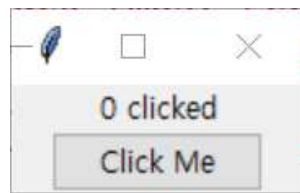
```

        text='Click Me',
        command=on_click
    )

    button.pack()

    gui.mainloop()

```



<그림 12.7> gui03c

이 프로그램을 실행한 후 버튼을 클릭하면 클릭할 때마다 라벨의 숫자(카운트)가 증가하는 것을 확인할 수 있다. ❶에서 보면 on_click()함수 바깥에 count=[0] 라고 리스트를 생성하였다. 그리고 ❷에서 count리스트의 0번 요소를 1 증가 시키고 ❸에서 count[0]을 라벨에 표시한다. 결과적으로 버튼이 클릭될 때마다 count[0]의 숫자가 1씩 증가하고 그것이 라벨이 표시되는 것이다. 이 프로그램의 on_click()함수를 다음과 같이 작성하면 안 되는 이유를 생각해 보자.

```

count=0 #❹
def on_click():
    count+=1 #❺
    label.configure(text=f'{count} clicked') #❻

```

위의 ❹와 같이 count를 숫자형으로 생성하면 ❺에서 count변수 값을 변경시키려고 할 때 오류가 발생하게 된다. 외부 변수는 읽을 수는 있되 쓸 수는 없기 때문이다. 만약에 쓰고 싶다면 다음의 ❷처럼 count를 global 변수로 선언해야 한다.

```

count=0
def on_click():
    global count #❷
    count+=1
    label.configure(text=f'{count} clicked')

```


하지만 global변수를 사용하는 것은 그리 바람직 하지 않으므로 gui03c.py처럼 count를 리스트로 선언하는 방법을 사용해도 된다. 그렇다면 ❶처럼 count를 리스트로 생성하면 왜 ❷에서는 오류가 발생하지 않을까. 이 경우도 count는 외부 변수이기 때문에 count 변수값 자체(리스트의 참조)를 변경할 수는 없지만, 리스트 내부의 요소를 변경하는 데는 아무런 지장이 없다.

12.2.3 엔트리

tkinter에서 한 줄짜리 텍스트 박스 위젯을 엔트리(entry)라고 하고 이것 역시 ttk모듈의 Entry 클래스를 이용하면 된다.

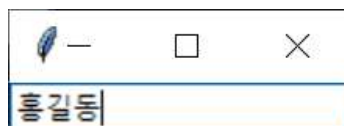
```
gui04.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

entry = ttk.Entry(gui)
entry.pack()

gui.mainloop()
```



<그림 12.8> gui04.py

이 프로그램을 실행하면 사용자가 텍스트를 입력할 수 있는 작은 입력창이 GUI안에 위치하게 되지만 이것만으로는 아무 일도 할 수 없다. Entry의 목적은 사용자 입력을 문자열로 얻어내어 이것을 목적에 맞게 이용하는 것에 있다. 이것을 위해서 Entry 객체의 생성자에 textvariable 이라는 키워드 인수에 tk.StringVar 객체를 지정해 주어야 한다.

```
gua04a.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
```

```

gui.title('tkinter GUI')

label = ttk.Label(gui, text='이름을 입력하세요.')
label.pack()

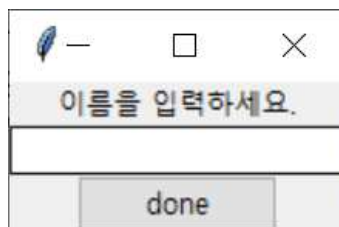
name = tk.StringVar() #❶
entry = ttk.Entry(
    gui,
    width=20,
    textvariable=name #❷
)
entry.pack()

def on_click():
    label.configure(text=f'{name.get()}님. 안녕하세요')#❸

button = ttk.Button(gui, text='done', command =
on_click)
button.pack()

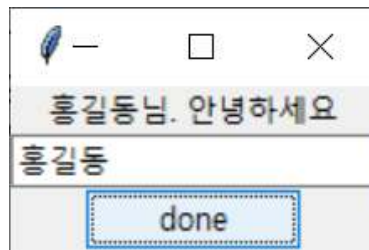
gui.mainloop()

```



〈그림 12.9〉 gui04a.py

❶에서 보면 `name=tk.StringVar()` 이라고 `tk.StringVar` 클래스의 객체를 생성하여 `name`이라는 외부 변수에 저장하였다. 그리고 ❷에서 `ttk.Entry` 객체 생성자의 `textvariable`이라는 키워드 인수에 `name`변수(즉, `tk.StringVar` 객체)를 지정하여 넘겨주었다. 이렇게 하면 `name`의 `get()`메서드를 이용하여 Entry에 사용자가 입력한 문자열을 얻어낼 수 있다. 따라서 ❸에서와 같이 사용자가 버튼을 클릭한 시점에서 Entry에 입력한 문자열을 label에 표시할 수 있는 것이다. Entry 객체 생성자에서 `width` 키워드 인수로 위젯의 너비를 설정할 수 있다는 것도 알아두자.



〈그림 12.10〉 gui04a.py 예

〈표 12.2〉 ttk.Entry 생성자/configure()메서드의 키워드 매개변수

키워드 인수	설명	지정값 예시
show	입력된 문자 대신 표시할 문자열 지정	show='*'으로 지정하면 모든 입력문자가 *로 표시됨
width	라벨의 너비(문자 수)	
justify	입력된 문자열을 정렬하는 방식 지정	tk.LEFT, tk.RIGHT, tk.CENTER

위의 〈표 12.2〉에서 show키워드는 Entry가 암호를 입력받는 창으로 사용하면 유용하다. 예를 들어 show='*'로 설정해 두면 입력 영역에 사용자가 “sesame” 라고 입력해도 “*****” 으로 표시된다.

12.2.4 콤보박스

콤보박스(combo-box)는 여러 개의 미리 정해진 선택지 중에서 하나를 고를 수 있도록 하는 기능의 위젯이다. ttk.Combobox 클래스를 이용해서 만들 수 있다.

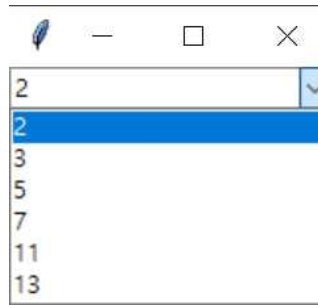
```
gui05.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

cb = ttk.Combobox(gui, width=20) #❶
cb['values']=(2,3,5,7,11,13) #❷
cb.current(0) #❸
cb.pack()

gui.mainloop()
```



<그림 12.11>

❶에서와 같이 `ttk.Combobox` 객체의 생성자도 첫 번째 인수로 `Tk` 객체(이 경우 `gui` 변수)를 받는다. ❷에서와 같이 드롭박스(drop box)에 나타날 각 항목들을 튜플로 지정해 주면 된다. 그리고 ❸과 같이 `current()` 메서드를 이용하면 맨 처음에 미리 선택되어진 항목을 튜플의 인덱스로 지정해 줄 수 있다. 이렇게 생성된 콤보박스는 마치 `Entry` 위젯과 같이 사용자가 어떤 내용을 타이핑할 수 있으며, `Combobox` 객체의 `get()` 메서드를 호출하면 입력된 내용을 문자열로 얻어낼 수 있다.

하지만 보통 콤보박스는 주어진 항목 중 하나를 선택하는 기능을 하며 사용자가 어떤 값을 입력하는 용도로는 사용하지 않는다. 이를 위해서 생성자의 `state` 키워드 인수를 `'readonly'`로 만들면 사용자 입력은 더이상 불가능하고 정해진 항목들 중에서만 선택할 수 있다. 또한 `get()` 메서드를 이용하면 현재 선택되어 있는 값을 알아낼 수 있다.

```
gui05a.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

label = ttk.Label(gui, text='Chosen : ')
label.pack()

def on_click():
    label.configure(text=f'Chosen:{cb.get()}') #❶

button=ttk.Button(
    gui,
    text='select',
    command=on_click
```

```

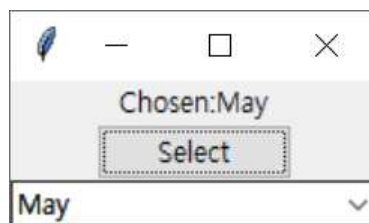
)
button.pack()

cb = ttk.Combobox(
    gui,
    width=20,
    state='readonly' #❷
)
cb['values']=('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun') #❸
cb.current(0)
cb.pack()

gui.mainloop()

```

❷에서 보면 state='readonly' 로 설정하여 콤보박스에 입력이 불가하도록 설정한다. 그리고 ❸처럼 튜플로 콤보박스에 표시될 각 항목을 지정해 줄 수 있다. ❶에서처럼 cb.get()메서드를 호출하면 현재 콤보박스에 선택되어 있는 항목(튜플의 한 요소)가 얻어지고 이게 라벨에 표시된다.



<그림 12.12>

이와 같이 콤보박스의 객체변수가 cb라고 할 때 cb['values']값에 각 항목들을 요소로 하는 튜플을 지정하고, cb.get()메서드를 이용해서 현재 선택되어 있는 항목을 읽어낼 수 있다.

12.2.5 체크버튼

체크버튼(check button)이란 체크박스가 달려서 체크와 언체크를 할 수 있는 위젯을 말한다. ttk.Checkbutton 클래스를 이용하면 생성할 수 있다. 다음 예제는 체크버튼 두 개를 생성하는 예제이다.

```

gui06.py

import tkinter as tk

```

```

from tkinter import ttk

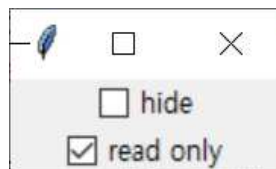
gui=tk.Tk()
gui.title('tkinter GUI')

var1 = tk.BooleanVar() #❶
check1=ttk.Checkbutton(gui,
    text='hide',
    variable = var1 #❷
)
check1.pack()

var2 = tk.BooleanVar()
var2.set(True) #❸
check2=ttk.Checkbutton(gui,
    text='read only',
    variable=var2
)
check2.pack()

gui.mainloop()

```



<그림 12.13>

체크박스의 상태는 체크가 되었는지 혹은 않았는지 두 가지 경우밖에 없으므로 진리 값으로 그 상태를 표시할 수 있다. 따라서 ❶에서처럼 tk.BooleanVar 객체를 생성하여 ttk.Checkbutton 객체의 생성자에서 variable 키워드 인수로 넘겨주면 된다. 초기값을 False이고 이 값이 반영되어 체크버튼도 초기에는 체크되어 있지 않다. 하지만 ❸처럼 BooleanVar 객체의 set()메서드를 이용하여 True로 설정해 준 다음에 생성자에서 variable 키워드 인수로 넘겨주면 초기에 체크가 된 상태로 시작된다. (<그림 12.13> 참조)

tk.Checkbutton 클래스의 생성자에 command 키워드 인수로 함수를 지정해 주면 체크를 하거나 해제할 경우에 그 함수가 호출되어 특정한 기능을 수행하게 할 수도 있다.

gui06b.py

```

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

def on_check1():
    if var1.get() :
        print('checked')
    else:
        print('unchecked')

var1 = tk.BooleanVar()
cb1 = ttk.Checkbutton(gui,
                      text='hide',
                      variable=var1,
                      command=on_check1
                      )

cb1.pack()

gui.mainloop()

```

다음 예는 버튼을 눌렀을 때 체크버튼들의 상태를 읽어 들어서 라벨에 적절한 메시지를 표시하도록 하는 예이다.

gui06c.py

```

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

label = ttk.Label(gui, text='파일 상태를 선택하세요')
label.pack()

def on_click():
    s1, s2 = var1.get(), var2.get()
    if s1 and s2:

```

```

        label.configure(text='hide and read only')
    elif s1 and not s2:
        label.configure(text='hide and read/write')
    elif not s1 and s2:
        label.configure(text='show and read only')
    else:
        label.configure(text='show and read/write')

var1 = tk.BooleanVar() # tk.StringVar tk.IntVar
cb1 = ttk.Checkbutton(gui,
    text='hide',
    variable=var1
)
cb1.pack()

var2 = tk.BooleanVar() # tk.StringVar tk.IntVar
cb2 = ttk.Checkbutton(gui,
    text='read only',
    variable=var2
)
cb2.pack()

btn = ttk.Button(gui, text='적용', command=on_click)
btn.pack()

gui.mainloop()

```

12.2.6 라디오버튼

라디오버튼(radio button)이란 여러 개의 옵션 중 하나만을 선택할 수 있는 위젯을 말한다. ttk모듈의 Radiobutton 클래스의 객체를 생성하면 GUI에 라디오버튼을 추가할 수 있다.

```
gui07.py
```

```

import tkinter as tk
from tkinter import ttk

```



```

rb2=ttk.Radiobutton(gui, text='green',
    variable=var, value=2,
    command = on_select) #❷
rb2.pack()

rb3=ttk.Radiobutton(gui, text='blue',
    variable=var, value=3,
    command = on_select) #❸
rb3.pack()

gui.mainloop()

```

위의 ❶, ❷, 그리고 ❸을 보면 ttk.Radiobutton 의 command 키워드인수에 on_select 함수를 지정했다. 이렇게 하면 라디오버튼이 선택된 직후 on_select()함수가 호출된다. 이 함수 내에서는 var.get()메서드를 이용하여 현재 어떤 라디오버튼이 선택되어 있는지를 읽어낸 후 각각의 값에 따라서 GUI의 배경색을 gui.configure()함수를 이용하여 변경시키도록 하고 있다. 만약 현재 'red'가 선택되어 있다면 var.get()메서드는 1을 반환하고, 'green'은 2가, 'blue'는 3이 반환되므로 if-elif 명령에 의해서 따로 구분해서 각각에 대해서 기능을 수행토록 한다.

12.2.7 ttk.LabelFrame 위젯

ttk.LabelFrame 위젯은 여러 개의 위젯을 하나의 그룹으로 묶고 테두리 선과 라벨로 구분해 주는 역할을 한다.

```

gui08.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

frame1 = ttk.LabelFrame(gui, text='options') #❶
frame1.pack()

var1 = tk.BooleanVar()

```

```

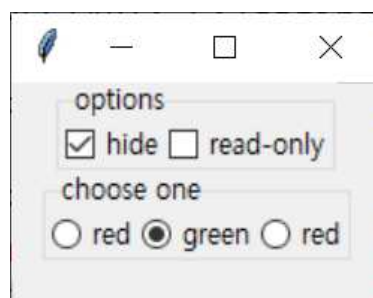
var1.set(True)
check1=ttk.Checkbutton(
    frame1, #❷
    text='hide',
    variable=var1
)
check1.grid(row=0, column=0) #❸
var2 = tk.BooleanVar()
check2=ttk.Checkbutton(frame1,text='read-only',variable=var2)
check2.grid(row=0, column=1)

frame2 = ttk.LabelFrame(gui,text='choose one')
frame2.pack()

var3 = tk.IntVar()
var3.set(2)
rb1=ttk.Radiobutton(frame2,text='red',variable=var3,value=1)
rb1.grid(row=0, column=0) #❹
rb2=ttk.Radiobutton(frame2,text='green',variable=var3,value=2)
rb2.grid(row=0, column=1) #❺
rb3=ttk.Radiobutton(frame2,text='red',variable=var3,value=3)
rb3.grid(row=0,column=2) #❻

gui.mainloop()

```



<그림 12.16>

이 예제에서 보면 두 개의 `ttk.LabelFrame` 객체가 있는데 ❶에서 첫 번째 것을 생성하고 변수 `frame1`에 저장하였다. 그리고 두 개의 `ttk.Checkbutton` 객체를 이 `frame1`에 포함시키기 위해서는 ❷와 같이 `ttk.Checkbutton` 생성자의 첫 번째 인수를 `gui`가 아니

라 frame1으로 지정해야 한다. 두 번째 ttk.Checkbutton객체 생성자의 첫 번째 인수도 frame1으로 지정한다. 이렇게 하면 두 개의 체크버튼이 frame1에 속하게 된다. (<그림 12.16> ❸에서 pack()메서드 대신에 grid()메서드를 사용했는데 이것은 frame1 구획을 격자(grid) 모양으로 나눈 특정 구획에 위젯을 위치시켜주는 기능을 한다. 예를 들어서 2x3 차원의 격자를 나눈다면 <그림 12.17>과 같다.

row=0 column=0	row=0 column=1	row=0 column=2
row=1 column=0	row=1 column=1	row=1 column=1

<그림 12.17> 2x3 차원의 격자 예

gui08.py의 frame2에 속한 세 개의 라디오버튼은 한 행에 모두 배치했으므로 ❹, ❺, ❻과 같이 grid()메서드에 인수를 넘겨주어서 각각의 위치를 잡아주면 된다.

12.2.8 스케일 위젯

스케일(scale) 위젯은 슬라이더(slider)를 마우스나 키보드로 조작하여 어떤 범위의 숫자를 사용자가 선택할 수 있도록 하는 위젯이다. ttk.Scale 클래스의 객체를 생성하면 된다.

scale.py

```
import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

scale = ttk.Scale(gui)
scale.pack()

gui.mainloop()
```



실행하면 위 그림처럼 슬라이더 막대가 생기고 그것을 마우스로 드래그(drag)하면 원

하는 위치에 맞출 수 있다. 하지만 이것만으로는 소용이 없이 슬라이더의 위치에 따른 값을 읽을 수 있어야 한다. 이를 위해서 생성자의 `command` 키워드인수를 이용하면 함수를 지정해 주면 슬라이더 막대의 위치가 변할 때마다 그 함수가 호출된다.

```
scale1.py

import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

frm = ttk.LabelFrame(gui)
frm.pack()

label = ttk.Label(frm, text='0.00')
label.grid(row=0, column=0)

def on_slide(v):
    value = float(v) #❶
    label.configure(text=f'{value:.2f}')

slider = ttk.Scale(frm, command = on_slide)
slider.grid(row=0, column=1)

gui.mainloop()
```



위의 `on_slide()` 함수는 스케일 객체의 막대가 움직일 때마다 호출되는데 첫 번째 인수로 막대의 위치가 (0, 1) 범위의 실수로 환산되어 문자열로 넘어온다. 이 숫자의 범위는 `Scale` 생성자에 `from_`, `to` 매개변수들로 조절할 수 있는데 `from_`의 기본값은 0, `to` 매개변수의 기본값은 1이다. 예를 들어서 숫자의 범위를 -10부터 10까지로 하고 싶다면 생성자에서 다음과 같이 지정하면 된다

```
slider = ttk.Scale(frm,
    command = on_slide,
    from_ = -10,
```

```
to = 10
)
```

위젯을 가로방향이 아니라 세로방향으로 만들고 싶다면 생성자의 orient 인수를 tk.VERTICAL로 지정하면 된다.

```
slider = ttk.Scale(frm,
    command = on_slide,
    from_ = -10,
    to = 10,
    orient = tk.VERTICAL
)
```

orient 매개변수의 기본값을 tk.HORIZONTAL이다. 그리고 스케일의 길이는 length 인수를 설정하면 되는데 기본값은 100픽셀(pixel)이다. 만약 현재 슬라이더의 값을 변수로 받고 싶다면 생성자의 variable 인수에 tk.DoubleVar 객체를 지정해 두면 된다.

scale2.py

```
import tkinter as tk
from tkinter import ttk

gui=tk.Tk()
gui.title('tkinter GUI')

frm = ttk.LabelFrame(gui)
frm.pack()

label = ttk.Label(frm, text='0')
label.grid(row=0, column=0)

val = tk.DoubleVar() #❶

def on_slide(_):
    value = val.get() #❷
    label.configure(text=f'{value:.2f}')

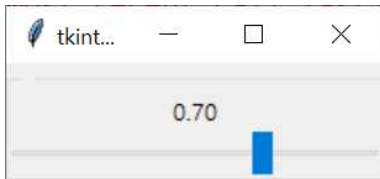
slider = ttk.Scale(frm,
    command = on_slide,
```

```

        length=200,
        variable=val #❸
    )
    slider.grid(row=1,column=0)

    gui.mainloop()

```



위의 ❸과 같이 `variable` 키워드인수를 `tk.DoubleVar` 객체인 `val`을 지정해 두면 `on_slide()`함수의 ❷에서 `val.get()` 메서드를 이용하여 슬라이더 위치 값을 직접 실수값으로 받을 수 있다. 이 경우 `on_slide()`함수의 첫 번째 인수는 사용하지 않게 되므로 매개변수명을 그냥 '_'으로 지정하였다.

12.3 경고창

이 절에서는 전 절에서 소개한 위젯을 조합하여 경고 메시지를 담은 창을 화면에 띄우고 [OK]버튼을 누르면 사라지는 단순한 기능을 하는 GUI를 만들어보자. 재사용할 수 있도록 하기 위해서 이것을 클래스로 작성하도록 하겠다. 클래스명은 `Alert`로 한다.

alert.py

```

import tkinter as tk
from tkinter import ttk

class Alert:

    def __init__(self,
                  message,
                  ok_text='OK'
                  ):
        self._gui = tk.Tk()

        label = ttk.Label(self._gui, text=message)
        label.grid(row=0,column=0,padx=10,pady=10)

```



```

        ok_button = ttk.Button(self._gui,
                                text= ok_text,
                                command=self.on_ok
                                )
        ok_button.grid(row=1,column=0,padx=10,pady=10)

        self._gui.mainloop()

    def on_ok(self):
        self._gui.destroy()

if __name__=='__main__':

    gui = Alert('설치가 성공적으로 수행되었습니다.')

```

Alert클래스의 _gui필드에 tk.Tk객체를 생성하여 저장하였는데, 이 필드는 외부에서 접근할 필요가 없으므로 필드명을 '_'로 시작하였다. 그리고 _gui필드는 on_ok()메서드에서 사용되었는데 destroy()메서드는 생성된 GUI를 소멸시키는 역할을 한다. 따라서 Alert클래스의 객체를 생성하면 화면에 첫 번째 인수로 넘긴 문자열을 표시하는 GUI가 생성되고, [OK]버튼을 누르면 GUI가 사라지는 단순한 기능을 한다.

Alert클래스를 tk.Tk클래스를 상속하여 작성할 수도 있다. 이렇게 하면 Alert객체가 tk.Tk객체 자체가 된다.

alert2.py

```

import tkinter as tk
from tkinter import ttk

class Alert(tk.Tk): #❶

    def __init__(self, message, ok_text='OK'):
        super().__init__() #❷

        label = ttk.Label(self, text=message) #❸
        label.grid(row=0,column=0,padx=10,pady=10)

        ok_button = ttk.Button(self,
                                text= ok_text,

```

```

        command=self.on_ok
    )
    ok_button.grid(row=1,column=0,padx=10,pady=10)

    self.mainloop() #④

    def on_ok(self):
        self.destroy() #⑤

if __name__=='__main__':

    gui = Alert('설치가 성공적으로 수행되었습니다.')
```

위의 ❶에서 보면 class Alert(tk.Tk): 라고 클래스를 정의하면 Alert클래스는 tk.Tk 클래스를 상속받게 된다. 이 경우 ❷와 같이 tk.Tk의 생성자를 먼저 호출해야 한다. 이후에는 self객체가 tk.Tk객체 자체가 되며, 더불어 사용자 필드나 메서드를 추가할 수도 있다. 이제 ❸과 같이 ttk.Label의 첫 번째 인수로 self를 바로 넘길 수도 있고 ❹,❺와 같이 mainloop()메서드와 destroy()메서드와 같이 tk.Tk 클래스의 메서드도 self를 통해서 직접 호출할 수 있다. alert.py보다 alert2.py가 좀 더 클래스(상속)의 장점을 활용하여 효율적인 방식이라고 할 수 있다.

12.4 메시지창 예제

이 절에서는 경고창을 좀더 확장하여 메시지를 표시하고 두 가지 선택지 중에서 하나를 고르는 기능을 하는 메시지창을 클래스로 작성하도록 하겠다. 클래스명은 Confirm으로 하고 전 절의 alter2.py와 같이 tk.Tk클래스를 상속받아서 작성하도록 한다.

```

confirm.py

import tkinter as tk
from tkinter import ttk

class Confirm(tk.Tk):

    def __init__(self,
                  message,
                  ok_text='OK',
```

```

        cancel_text='CANCEL'
    ):
    super().__init__()

    label = ttk.Label(self, text=message)
    label.grid(row=0,column=0,padx=10,pady=10)

    ok_button = ttk.Button(self,
        text= ok_text,
        command=self.on_ok
    )
    ok_button.grid(row=1,column=0,padx=10,pady=10)

    cancel_button = ttk.Button(self,
        text= cancel_text,
        command=self.on_cancel
    )
    cancel_button.grid(row=1,column=2,
                        padx=10, pady=10)

    self.is_ok = None
    self.mainloop()

    def on_ok(self):
        self.is_ok = True
        self.destroy()

    def on_cancel(self):
        self.is_ok = False
        self.destroy()

if __name__=='__main__':

    gui = Confirm('설치를 진행합니다.\n 계속할까요?')
    if gui.is_ok:
        print('설치가 진행된다')
    else:
        print('설치가 취소되었다')

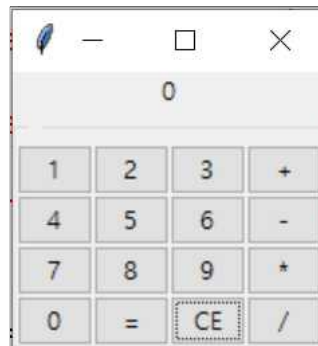
```

destroy()메서드가 호출되어 GUI가 사라졌다고 할 지라도 이 Confirm 객체는 아직 남

아있다. 따라서 그 객체의 `is_ok` 필드를 검사하면 사용자가 어떤 버튼을 눌렀는지를 판단할 수 있다.

12.5 계산기 예제

전 절에서 설명한 위젯을 이용하여 간단한 계산기 앱을 작성해 보도록 하자. 외형은 다음 그림과 같다.



〈그림12.??〉 계산기 앱

계산기 앱의 맨 상단에는 입력한 숫자를 표시할 라벨이 위치하고 그 아래에 버튼들이 자리하고 있다. 버튼들은 모두 `ttk.LabelFrame` 클래스를 사용하여 하나의 프레임으로 묶는데 숫자 버튼들 [1], [2], ... [9], [0], 그리고 사칙연산 버튼([+], [-], [*], [/]), 결과를 계산하는 [=] 버튼, 계산기를 초기화하는 [CE] 버튼이 있다. 이 계산기 정수 계산만 수행할 수 있는 간단한 앱이다.

앱이 동작하는 순서를 먼저 다음과 같이 설계해보자.

- ① 초기에는 숫자 0만 표기되어 있다.
- ② 숫자키를 누르면 그 숫자가 현재 숫자에 추가된다.
- ③ 계산키를 누르면 현재 표시된 숫자(a)와 연산자(b)를 저장한다.
 - ③-1 만약 이전 단계에서 저장된 숫자(c)와 이전에 저장된 연산자(d)가 있다면 그것과 (a)를 (d)연산하여 display에 표시하고 (c)를 계산된 결과로, (d)를 지금 누른 연산자로 갱신한다.
 - ③-2 이후에 숫자키가 눌러지면 display를 초기화한 후 표시한다.
- ④ [=]버튼이 눌러졌다면 현재 표시된 숫자(a)와 이전에 저장된 숫자(c)와 이전에 저장된 연산자(d)를 적용한 결과를 표시한다. 그리고 (c)를 지금 계산된 결과로 갱신한다.
- ⑤ [CE]버튼이 눌러지면 모든 상태를 초기화한다.

이 알고리즘을 구현하면 예를 들어 다음과 같이 작성할 수 있다.

calculator.py

```
import tkinter as tk
from tkinter import ttk

gui = tk.Tk()
gui.title('계산기')

g = { #❶
    'disp':None, #현재 disp(Label)에 표시되는 문자열
    'op':None,
    'num':None #직전에 입력된 숫
}

disp = ttk.Label(gui, text='0')
disp.pack()

key_frame = ttk.LabelFrame(gui)
key_frame.pack()

def on_key_num(num_str): #❷
    if g['disp'] == None: #None이라면 새로운 숫자를 시작
        g['disp'] = num_str
    else:
        g['disp'] += num_str

    disp.configure(text=g['disp'])

def on_key_cal(op):
    if g['disp']==None:
        g['op']=op
        return

    if g['num']==None:
        g['num'] = g['disp']
    else: #직전에 지정된 피연산자가 있는 경우
        ans = eval(g['num']+g['op']+g['disp'])
```

```
        str_ans = str(ans)
        disp.configure(text=str_ans)
        g['num']=str_ans

    g['disp']=None
    g['op']=op

def on_key_equal():
    if g['num'] and g['op']:
        ans = eval(g['num']+g['op']+g['disp'])
        str_ans = str(ans)
        disp.configure(text=str_ans)
        g['disp']=None
        g['num']=str_ans
        g['op']=None

def on_key_ce():
    g['dict']=None
    g['num']=None
    g['op']=None
    disp.configure(text='0')

w = 4
key1=ttk.Button(key_frame, text='1',width=w,
                command = lambda:on_key_num('1') )
key1.grid(row=0,column=0)

key2=ttk.Button(key_frame, text='2',width=w,
                command = lambda:on_key_num('2') )
key2.grid(row=0,column=1)

key3=ttk.Button(key_frame, text='3',width=w,
                command = lambda:on_key_num('3') )
key3.grid(row=0,column=2)

key4=ttk.Button(key_frame, text='4',width=w,
                command = lambda:on_key_num('4') )
```

```
key4.grid(row=1,column=0)

key5=ttk.Button(key_frame, text='5',width=w,
                command = lambda:on_key_num('5') )
key5.grid(row=1,column=1)

key6=ttk.Button(key_frame, text='6',width=w,
                command = lambda:on_key_num('6') )
key6.grid(row=1,column=2)

key7=ttk.Button(key_frame, text='7',width=w,
                command = lambda:on_key_num('7') )
key7.grid(row=2,column=0)

key8=ttk.Button(key_frame, text='8',width=w,
                command = lambda:on_key_num('8') )
key8.grid(row=2,column=1)

key9=ttk.Button(key_frame, text='9',width=w,
                command = lambda:on_key_num('9') )
key9.grid(row=2,column=2)

key0=ttk.Button(key_frame, text='0',width=w,
                command = lambda:on_key_num('0') )
key0.grid(row=3,column=0)


key_add=ttk.Button(key_frame, text='+',width=w,
                  command = lambda:on_key_cal('+') )
key_add.grid(row=0,column=3)

key_sub=ttk.Button(key_frame, text='-',width=w,
                  command = lambda:on_key_cal('-') )
key_sub.grid(row=1,column=3)

key_mul=ttk.Button(key_frame, text='x',width=w,
                  command = lambda:on_key_cal('*') )
key_mul.grid(row=2,column=3)
```

```

key_div=ttk.Button(key_frame, text='/',width=w,
                    command = lambda:on_key_cal('/') )
key_div.grid(row=3,column=3)

key_equal=ttk.Button(key_frame, text='=',width=w,
                     command = on_key_equal )
key_equal.grid(row=3,column=1)

key_ce=ttk.Button(key_frame, text='CE',width=w,
                  command = on_key_ce )
key_ce.grid(row=3,column=2)

gui.mainloop()

```

이 예제의 ❶에서 보면 변수 g에 딕셔너리를 생성해서 대입했는데 세 개의 키-값 쌍이 있다. 이 g딕셔너리는 이후에 나오는 함수들 내부에서 그 키-값 쌍을 읽거나 새로운 값을 저장하면서 흐름을 제어하는데 사용되는 변수이다.

먼저 g['disp']는 라벨에 표시될 문자열을 저장하는 역할을 한다. ❷의 on_key_num(num_str)함수는 숫자 버튼이 눌렸을 때 그 숫자의 문자열(즉, '0', '1', '2', ... , '9' 중 하나)을 num_str 매개변수로 받는데 받은 숫자를 현재 문자열 뒤에 추가하는 역할을 한다. 단, g['disp']가 None일 경우 이전의 값을 무시하고 처음부터 다시 숫자를 시작한다.

g['num']과 g['op']는 직전에 저장된 숫자와 연산자를 저장하는 요소이다. 예를 들어서 사용자가 다음 <표12.2>와 같이 순차적으로 버튼을 눌렀다고 가정하고 g딕셔너리의 각각의 키-값 쌍들의 변화를 보도록 하자. 실제 함수들의 동작과 이 표에서의 g딕셔너리의 변화값을 확인하면 전체적인 동작의 흐름이 이해가 가리라고 생각된다. 이 표에서 설명하지는 않았지만 실제로 연속 계산도 가능하다. 예를 들어 [1] [+] [2] [-] [3] [*] [4] 와 같이 수식을 연속으로 입력해도 해당되는 계산이 수행이 된다. 이 경우도 g딕셔너리의 키-값 쌍들의 값들의 변화를 표로 만들어서 보면 어떻게 동작하는 지 이해할 수 있을 것이다.

〈표 12.7〉 계산기 앱 동작 예

눌려진 버튼	호출되는 함수	결과			
		disp (표시)	g['disp']	g['num']	g['op']
초기값		'0'	None	None	None
[1]	on_key_num('1')	'1'	'1'	None	None
[2]	on_key_num('2')	'12'	'12'	None	None
[+]	on_key_calc('+')	'12'	None	'12'	'+'
[3]	on_key_num('3')	'3'	'3'	'12'	'+'
[0]	on_key_num('0')	'30'	'30'	'12'	'+'
[=]	on_key_equal()	'42'	None	'42'	None
[CE]	on_key_ce()	'0'	None	None	None

제 13 장 sympy로 수학적 풀기

컴퓨터는 원래 수치 계산(numerical calculation)을 빠르고 정확하게 수행할 목적으로 개발되었다. 컴퓨터를 이용하면 아무리 복잡하고 어려운 계산이라도 단시간에 정확히 수행하여 결과값을 얻어낼 수 있다. 파이썬에서도 산술 연산자와 math모듈을 이용하여 다양한 수치계산을 수행할 수 있다. 반면, 수치 계산과는 다르게 기호식 계산(symbolic calculation)은 대수 기호(symbol)가 들어간 수식을 다루는 것이다. 예를 들어서 이차방정식 $x^2 + 3x + 2 = 0$ 의 근을 구하는 것은 수치 계산에 해당된다. 근의 공식을 이용하여 수치 해를 구할 수 있기 때문이다. 하지만 $x^2 + ax + 2 = 0$ 이라는 이차방정식의 계수에는 대수 기호 a 가 포함되어 있기 때문에 이것의 근을 구하는 것은 수치 계산과 다른 문제이다. 이외에도 인수분해를 한다든가 또는 미분방정식을 푼다든가 하는 것들이 다 기호식 계산의 범주에 들어간다. 사람이 이러한 개념을 사용하기는 어렵지 않으나 컴퓨터에게 이러한 일을 수행토록 하는 것은 고전적인 수치 계산 알고리즘보다 훨씬 더 어려운 일이다.

컴퓨터를 이용하여 이러한 기호식 계산을 수행할 수 있는 도구는 많이 있는데 대표적인 예를 들면 maxima, mathematica, sagemath 등이다. 본 교재에서는 파이썬 패키지인 sympy를 사용하여 기호식 계산을 수행해 보도록 하겠다. sympy는 symbolic python을 조합한 단어로써 대수방정식, 인수분해, 미적분, 미분방정식, 선형대수, 미분기하 등 다양한 분야에 활용될 수 있는 파이썬 패키지이다. 이것은 기본 라이브러리가 아니고 외부 라이브러리이기 때문에 사용하려면 설치하는 과정을 거쳐야 한다. 윈도우 실행창(cmd)에서 다음과 같이 명령을 내리면 쉽게 설치할 수 있다. 패키지를 다운로드해야 하므로 인터넷에 연결이 되어 있어야 정상적으로 설치된다.

```
> pip3 install sympy
```

만약 sympy가 이미 설치되어 있다면 다음과 같이 업그레이드 한다.

```
> pip3 install --upgrade sympy
```

이후에 파이썬셸을 실행하면 sympy 패키지를 임포트할 수 있다.

실습을 위해서 jupyter 환경을 사용해도 된다. PC에 별도로 설치 과정 없이 웹브라우저 상에서 동작하는 jupyterLab(jupyter.org/try)을 사용하는 것을 권장한다. 이 환경에서는 sympy를 설치하지 않아도 바로 임포트할 수 있고 latex 기반의 수식이나 그래프도 바로 확인할 수 있다.

13.1 대수식 만들기

pip3명령을 이용하여 sympy 패키지를 설치한 후에는 파이썬셸에서 sympy를 임포트할 수 있다. 본 장에서는 다른 모듈은 임포트하지 않고 오직 sympy만을 사용한다고 가정하고 다음과 같이 임포트하여 모든 sympy내의 구성요소를 곧바로 사용할 수 있도록 한다.

```
>>> from sympy import *
>>> init_printing() #❶
```

❶은 확장된 문자 집합을 사용하여 표현식을 좀 더 수학적과 유사하게 화면에 표시하도록 하는 기능을 한다. winpython을 이용한다면 미려한 수식으로 결과를 확인할 수 있다. 이제 Symbols 클래스를 사용하여 다음과 같이 대수기호 x 를 생성할 수 있다.

```
>>> x = Symbol('x')
```

이제 x 는 일반적인 변수가 아니라 sympy에서 사용되는 대수 기호 (객체)로 정의되었으며 이 대수기호를 이용하여 다항식을 만들 수 있다. 생성자에 넘겨준 인수 ' x '는 이 객체를 화면에 표시할 때 사용할 문자열을 지정해 준 것이다. 여러개의 대수기호를 동시에 생성하고 싶다면 symbols()함수를 이용하면 된다.

```
>>> x, a = symbols('x a')
```

위와 같이 하면 x 와 a 두 개의 Symbol 객체가 동시에 생성되었다.

이제 수식 x^2+x+1 을 변수 $y1$ 에 저장해보자.

```
>>> y1=x**3+x**2+x+1
>>> y1
 $x^3+x^2+x+1$ 
```

그리고 $(2x+1)^5$ 을 변수 $y2$ 에 저장해보자.

```
>>> y2=(2*x+1)**5
>>> y2
```

$$(2x+1)^5$$

산술연산자를 이용하면 기존의 수식으로부터 새로운 수식도 만들 수 있다.

```
>>> y3=y1/y2
>>> y3

$$\frac{(2x+1)^5}{x^3+x^2+x+1}$$

>>> y4=y1**3
>>> y4

$$(x^3+x^2+x+1)^3$$

```

다항식과 유리식뿐만 아니라 상수들과 초등함수(elementary functions)도 sympy에 마련되어 있는데 <표 13.1.1>에 기술하였다.

<표 13.1.1> sympy의 상수와 초등함수들

상수/초등함수 (x는 symbol)	설명
<code>Rational(n,m)</code> 여기서 n과 m은 정수	분수 $\frac{n}{m}$ 객체 생성
<code>pi</code> , <code>E</code>	π , e (자연상수)
<code>oo</code>	무한대(infinite, ∞)
<code>exp(x)</code>	e^x
<code>ln(x)</code> 혹은 <code>log(x)</code>	밑이 e 인 로그함수
<code>sin(x)</code> , <code>cos(x)</code> , <code>tan(x)</code>	삼각함수
<code>asin(x)</code> , <code>acos(x)</code> , <code>atan(x)</code>	역삼각함수
<code>sinh(x)</code> , <code>cosh(x)</code> , <code>tanh(x)</code>	초월함수
<code>asinh(x)</code> , <code>acosh(x)</code> , <code>atanh(x)</code>	역초월함수
<code>sqrt(x)</code>	제곱근

예를 들어 변수 y5 에 $\frac{\sin x}{x}$ 를, y6에 $\frac{1}{\sqrt{x^2+1}}$ 을 대입해보자.

```
>>> y5=sin(x)/x
>>> y5

$$\frac{\sin(x)}{x}$$

>>> y6=1/sqrt(x**2+1)
>>> y6
```

$$\frac{1}{\sqrt{x^2+1}}$$

이와 같이 sympy의 대수기호 객체와 초등함수들을 이용하여 다양한 수식을 만들 수 있고 수식들 간 산술 연산을 수행할 수 있다.

대수기호 대신 실제 값(수치)를 대입하여 그 결과값을 얻고자 할 때 subs()메서드를 이용하면 된다. 예를 들어 y1식의 x대신 2를 대입하여 그 값을 알고 싶다면 subs()메서드를 이용한다.

```
>>> y1.subs(x,2)
15 #❶
>>> y1.subs(x, x**2+1) #❷
 $x^2 + (x^2 + 1)^3 + (x^2 + 1)^2 + 2$ 
```

그러면 $2^3 + 2^2 + 2 + 1$ 값을 계산하여 반환된 것을 알 수 있다. ❷와 같이 숫자뿐만 아니라 다른 표현식으로 대치할 수도 있는데 y1 수식의 x대신 $x^2 + 1$ 을 대입하라는 것이다.

sympy는 무한소수(예를 들어 $\frac{1}{3}$ 같은 순환소수나 무리수) 그 자체로 표현된다. 예를 들어 π 는 무리수이므로 이것을 정확하게 표현하는 유한소수 표기는 없다. 3.14159265 라는 실수는 π 의 근사값일 뿐이지 실제 정확한 π 값은 아니다. $\frac{1}{3}$ 도 0.33333333 이라는 유한소수와 다른 수이므로 $\frac{1}{3}$ 은 이 자체로 표기가 되어야 한다는 것이다. sympy에서 $\frac{1}{3}$ 과 같이 나누어 떨어지지 않는 분수를 정확히 입력하기 위해서는 Rational 클래스나 S 클래스의 객체를 생성하면 된다.

```
>>> n1=Rational(1,3)
>>> n1
 $\frac{1}{3}$ 
>>> n1+S(3)/4      #  $\frac{1}{3} + \frac{3}{4}$  계산
 $\frac{13}{12}$ 
```

```
>>> f=x+S(1)/3
>>> f

$$x + \frac{1}{3}$$

```

이러한 sympy의 특성은 파이썬이 기본적으로 수를 다루는 방식과 구별되는 차이점이다. 만약 이러한 무한소수의 ‘근사값’을 알고 싶다면 N()함수를 이용하던가 혹은 수식의 evalf()메서드를 호출하면 된다.

```
>>> g=pi/x
>>> v=g.subs(x,3)
>>> v

$$\frac{\pi}{3}$$

>>> N(v,4)
1.047
>>> v.evalf()
1.04719755119660

>>> h=1/sqrt(x**2+1)
>>> h.subs(x,2)

$$\frac{\sqrt{5}}{5}$$

>>> N(h.subs(x,2))
0.447213595499958
```

즉, N()함수 혹은 evalf()메서드는 수식에 무한소수가 포함된 경우 그것의 근사값을 (유한)실수로 구해주는 메서드이다. N()함수의 두 번째 인수 혹은 evalf()의 첫 번째 인수에 유효자리의 개수를 넘겨줄 수 있다.

13.2 인수분해와 전개

sympy의 factor()함수를 이용하여 다항식을 인수분해할 수 있다.

```
>>> x,a=symbols('x a')
>>> factor(x**2-1)
(x - 1) * (x + 1)
```

```
>>> a=symbols('a') #❶
>>> factor(x**2-3*a*x+2*a**2) #❷
(-2 · a + x) · (-a + x)
>>> factor(y1) #❸
(x + 1)(x2 + 1)
```

❶에서 또 다른 대수기호 객체 a 를 생성한 후 ❷에서 $x^2 - 3ax + 2a^2$ 을 인수분해를 수행했다. 결과로 $(-2a+x)(-a+x)$ 가 얻어졌다. ❸과 같이 기존에 정의된 표현식을 `sp.factor()`함수의 인수로 넘겨도 된다.

`expand()`함수는 반대로 수식을 전개하는 기능을 한다.

```
>>> expand((x+2)**3)
x3 + 6x2 + 12x + 8
>>> expand((2*x-a)**3)
a3 + 6a2x + 12ax2 + 8x3
>>> y=expand((sp.sin(x)+sp.cos(x))**2) #❶
sin2(x) + 2sin(x)cos(x) + cos2(x)
>>> y.simplify() #❷
sin(2x) + 1
```

위에서 ❶과 같이 `expand()`함수의 결과식을 y 변수에 저장할 수도 있다. 이후에 ❷와 같이 `simplify()` 메서드를 호출하면 저장된 수식에서 간략화할 수 있는 요소들을 찾아내어 더 간단한 수식을 반환해 준다. 이 경우 $\sin^2 x + \cos^2 x = 1$ 과 $2\sin x \cos x = \sin 2x$ 두 사실이 이용된 것이다.

13.3 방정식의 근 구하기

`solve()`함수를 이용하면 대수방정식의 해를 구할 수 있다. `solve()`함수의 첫 번째 인수는 표현식인데 방정식의 우변을 0으로 만들었을 때의 좌변식을 입력하면 된다. 예를 들어 $x^3 = 1$ 이라는 방정식의 우변을 0으로 만들면 $x^3 - 1 = 0$ 이 되는데 이것의 좌변을 첫 번째 인수로 넘기면 된다. 두 번째 인수는 해를 구할 대수기호이다.

```
>>> solve(x**3-1,x)
```

```


$$\left[ 1, -\frac{1}{2} - \frac{\sqrt{3}i}{2}, -\frac{1}{2} + \frac{\sqrt{3}i}{2} \right]$$

>>> solve(sin(x)-cos(x),x)

$$\left[ -\frac{3\pi}{4}, \frac{\pi}{4} \right]$$


```

방정식에 대수기호가 포함되어 있어도 해를 구할 수 있다.

```

>>> solve(x**2+a*x+1,x)

$$\left[ -\frac{a}{2} - \frac{\sqrt{a^2-4}}{2}, -\frac{a}{2} + \frac{\sqrt{a^2-4}}{2} \right]$$


```

연립방정식의 경우 첫 번째 인수로 표현식들을 리스트로 묶어서 넘겨주면 된다.

```

>>> x, y=Symbol('x y')
>>> solve([2*x+3*y-1, 3*x+2*y+2],x,y)

$$\left\{ x: -\frac{8}{5}, y: \frac{7}{5} \right\}$$

>>> solve([x**2-y, 2*x+y+1],x,y)
[(-1,1)]
>>> a,b=symbols('a b')
>>> solve([a*x+y+1, x+b*y+2],x,y)

$$\left\{ x: \frac{2-b}{ab-1}, y: \frac{1-2a}{ab-1} \right\}$$


```

단, 방정식이 너무 복잡하면 해를 구하는데 실패하는 경우도 있다.

13.4 도함수 구하기

sympy의 diff()함수를 이용하면 도함수를 구할 수 있다. 첫 번째 인수로 수식을, 두 번째 인수로 미분변수를 넘겨주면 된다.

```

>>> y1 = x**3+x**2+x+1
>>> diff(y1,x)

```



```

3x2 + 2x + 1
>>> diff(y1/(a*x+1),x)

$$-\frac{a(x^3+x^2+x+1)}{(ax+1)^2} + \frac{3x^2+2x+1}{ax+1}$$

>>> diff(sin(x)**2*log(x),x)

$$2\log(x)\sin(x)\cos(x) + \frac{\sin^2(x)}{x}$$


```

미리 정의된 표현식들 y1, y2를 이용하여 유리식 y1/y2 의 x에 대한 도함수를 구하면 다음과 같다.

```

>>> y=diff(y1/y2,x)

$$\frac{3x^2+2x+1}{(2x+1)^5} - \frac{10(x^3+x^2+x+1)}{(2x+1)^6}$$


```

이 결과로부터 $\frac{x^3+x^2+x+1}{(2x+1)^5}$ 의 x에 대한 도함수는 $\frac{3x^2+2x+1}{(2x+1)^5} - \frac{10(x^3+x^2+x+1)}{(2x+1)^6}$ 이라는 것을 알 수 있다.

```

>>> diff(ln(x)/x,x) #❶

$$-\frac{\log(x)}{x^2} + \frac{1}{x^2}$$

>>> diff(tanh(a*x),x) #❷

$$a(1 - \tanh^2(ax))$$


```

위와 같이 초등함수의 미분도 쉽게 구할 수 있다. ❶은 $\frac{\ln x}{x}$ 의 미분을 구한 것이고, ❷은 $\tanh(ax)$ 의 미분을 구한 것이다. 여기서 a도 sympy의 기호객체이다. 만약 $x\sqrt{x+3}$ 의 도함수를 구하고 싶다면 다음과 같이 입력하면 된다.

```

>>> diff(x*sqrt(x+3),x)

$$\frac{x}{2\sqrt{x+3}} + \sqrt{x+3}$$


```

이 예제와 같이 제공된 함수를 입력할 때는 sqrt()를 이용한다.

13.5 적분 수행하기

sympy를 이용하면 기호식의 적분도 쉽게 구할 수 있는데 integrate() 함수를 이용하면 된다. 이 함수는 다음과 같이 두 가지 용법이 있다.

integrate(f, x) : 부정적분식 $\int f dx$ 를 반환
 integrate(f,(x,a,b)) : 정적분 $\int_a^b f dx$ 값을 반환

정적분의 예를 들어보자.

```
>>> integrate(x**2 + x + 1, x)

$$\frac{x^3}{3} + \frac{x^2}{2} + x$$

>>> integrate(x/(x**2+2*x+1), x)

$$\log(x + 1) + \frac{1}{x + 1}$$

>>> integrate(x**2 * exp(x) * cos(x), x)

$$\frac{x^2 e^x \sin(x)}{2} + \frac{x^2 e^x \cos(x)}{2} - x e^x \sin(x) + \frac{e^x \sin(x)}{2} - \frac{e^x \cos(x)}{2}$$

```

정적분 결과식에서 적분상수는 표기가 생략되어 있다. 부정적분은 다음과 같이 구할 수 있다.

```
>>> integrate(1/x, (x,1,2))
log(2)
>>> integrate(sin(x)**2,(x,0,pi)) #❶

$$\frac{\pi}{2}$$

>>> integrate(x**2*exp(x)*cos(x), (x,0,2*pi))

$$-\frac{e^{2\pi}}{2} + \frac{1}{2} + 2\pi^2 e^{2\pi}$$

>>> integrate(exp(-x),(x,0,oo)) #❷
1
```

```
>>> a,b=symbols('a,b')
>>> integrate(x**2,(x,0,a)) #❸

$$\frac{a^3}{3}$$

>>> integrate(a*x**2+b,(x,0,1)) #❹

$$\frac{a}{3}+b$$

```

위에서 ❶은 $\int_0^{\pi} \sin^2 x dx$ 값을 구하는 것이고 ❷은 $\int_0^{\infty} e^{-x} dx$ 를 계산하는 것이다. sympy에서 oo(소문자 o 두 개)는 무한대를 의미한다. 그리고 ❸과 ❹의 경우와 같이 적분 구간이나 표현식에 기호객체가 포함된 경우에도 답을 구할 수 있다.

13.6 극한값 구하기

표현식의 극한값을 구하려면 limit()함수를 이용하면 된다. 세 개의 인수를 넘겨야 하는데 순서대로 함수식, 변수, 극한값이다.

```
>>> limit(sin(x)/x,x,0)
1
>>> limit(x**2/exp(x),x,oo)
0
```

좌극한이나 우극한을 구하려면 limit()함수의 네 번째 인수로 '+' 혹은 '-' 를 넘겨 주면 된다.

```
>>> limit(1/x,x,0,'+')

$$\infty$$

>>> limit(1/x,x,0,'-')

$$-\infty$$

```

위는 $\lim_{x \rightarrow 0^+} \frac{1}{x}$ 와 $\lim_{x \rightarrow 0^-} \frac{1}{x}$ 를 구한 것이다.

13.7 그래프 그리기

sympy에는 수학식의 그래프를 그리기 위한 plot()함수가 있는데 이것은

sympy.plotting 패키지에서 읽어들여야 한다.

```
>>> from sympy.plotting import plot
```

기본적인 사용법은 다음과 같다.

```
plot(함수, (변수, 최대값, 최소값))
```

예를 들면 다음과 같다.

plot1.py

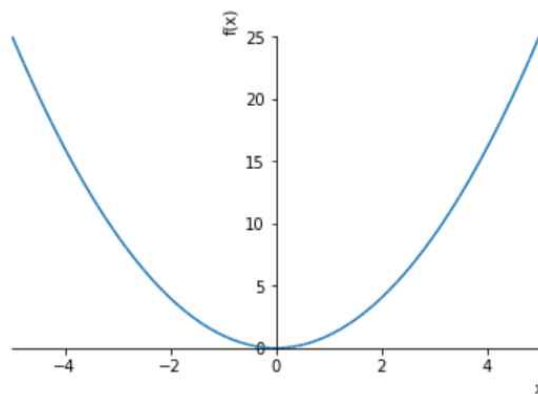
```
from sympy import *
from sympy.plotting import plot
x=symbols('x')

plot(sin(x)/x,(x,-4*pi,4*pi)) #❶

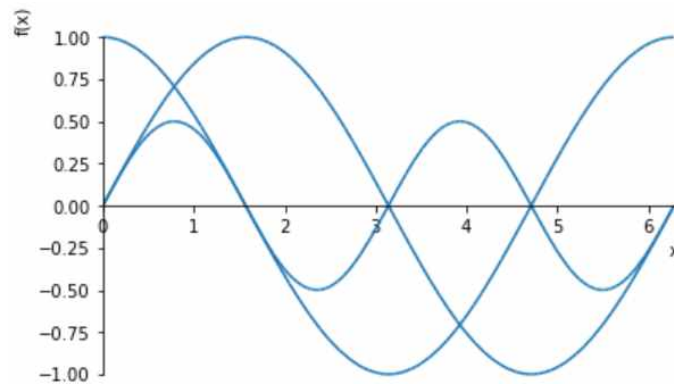
plot(sin(x),cos(x),sin(x)*cos(x),(x,0,2*pi)) #❷

plot((x**2,(x,-6,6)),(x,(x,-5,5))) #❸
```

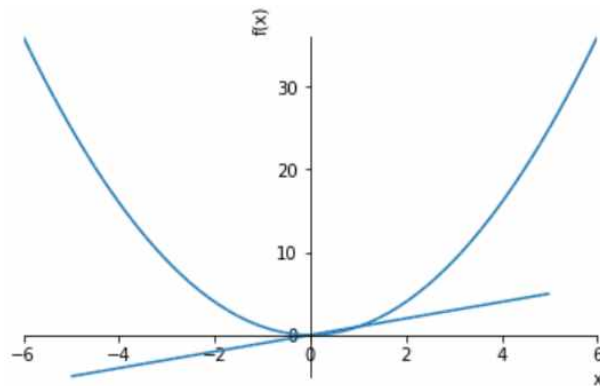
이것을 저장한 후 실행하면 ❶번 줄에서 다음과 같은 그래프를 얻을 수 있다.



같은 범위의 다수의 함수를 그릴려면 ❷와 같이 수식을 먼저 나열한 후 맨 마지막에 범위정보를 담은 튜플을 넘기면 된다.



서로 다른 범위의 다른 함수를 도시하려면 ❸과 같이 함수-범위 쌍을 튜플로 묶고 이것을 콤마로 구분하여 여러 개를 나열하면 된다.



13.8 행렬 계산(선택)

13.8.1 행렬의 생성

sympy를 이용하여 행렬 계산을 할 수 있는데 이 때 사용되는 것이 Matrix 클래스이다. 행렬은 Matrix의 객체인데 생성자에 행렬의 요소를 리스트로 넘기면 된다.

```
>>> m1=Matrix([[1,2],[3,4]]) #❶
>>> m1

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

>>> m2=Matrix([[5,6,7],[8,9,10]]) #❷
>>> m2
```

```


$$\begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$$

>>> m3=Matrix([11,22,33]) #❸
>>> m3

$$\begin{bmatrix} 11 \\ 22 \\ 33 \end{bmatrix}$$

>>> m4=Matrix([[11,22,33]]) #❹
>>> m4

$$\begin{bmatrix} 11 & 22 & 33 \end{bmatrix}$$


```

위의 ❶, ❷와 같이 행렬을 생성할 때 각 행을 이루는 리스트들을 다시 리스트로 묶어서 중첩 리스트를 생성자의 인수로 넘겨준다. ❸과 같이 단일 리스트를 넘겨주면 열벡터가 생성되고 ❹와 같이 중첩 리스트 안에 리스트가 하나일 때는 행벡터가 생성된다.

다른 방법으로 Matrix 생성자에 행의 개수와 열의 개수를 먼저 표준 인수로 넘겨주는 방법도 있다. 세 번째 인수로는 정해진 개수만큼 요소를 갖는 시퀀스이다.

```

>>> m5=Matrix(3,2,[2,3,5,7,11,13])
>>> m5

$$\begin{bmatrix} 2 & 3 \\ 5 & 7 \\ 11 & 13 \end{bmatrix}$$

>>> m6=Matrix(4,4,range(16))
>>> m6

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$


```

첫 번째와 두 번째 인수로 행수와 열수를 넘기는 경우 세 번째 인수로 함수를 줄 수도 있다. 이 함수는 행과 열을 받아서 하나의 값을 반환하는 함수여야 한다.

```

>>> m7=Matrix(4,4,lambda i,j: 0 if i==j else 1)
>>> m7

```

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

```
>>> m8=Matrix(4, 3, lambda i,j:i+j)
```

```
>>> m8
```

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

다음 <표 13.8.1>에는 특수한 행렬을 생성하는 함수들을 나열한 것이다.

<표 13.8.1> 특수행렬을 생성하는 sympy 함수들

sympy 함수	
eye(n)	nxn 크기의 항등행렬 생성
zeros(n)	nxn 크기의 모든 요소가 0인 정방행렬 생성
zeros(n,m)	nxm 크기의 모든 요소가 0인 행렬 생성
ones(n,m)	nxm 크기의 모든 요소가 1인 행렬 생성
diag(e1,e2,...)	주어진 인수들을 대각요소로 하는 정방행렬 생성
randMatrix(n)	nxn의 로 채워진 행렬 생성
randMatrix(n, symmetric=True)	nxn의 난수로 채워진 대칭 행렬 생성
randMatrix(n,m)	nxm의 난수로 채워진 행렬 생성 (난수는 0에서 99사이의 정수)

아래는 그 예제들이다.

```
>>> m9=eye(3)
```

```
>>> m9
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
>>> ma=zeros(2)
```

```
>>> ma
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
>>> mb=zeros(2,4)
```

```
>>> mb
```

```


$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

>>> mc=ones(2,5)
>>> mc

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

>>> md=diag(11,22,33)
>>> md

$$\begin{bmatrix} 11 & 0 & 0 \\ 0 & 22 & 0 \\ 0 & 0 & 33 \end{bmatrix}$$

>>> me=diag(ones(2),2*ones(2),3*ones(2))
>>> me

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 3 & 3 \end{bmatrix}$$

>>> mf=randMatrix(3)
>>> mf

$$\begin{bmatrix} 60 & 39 & 1 \\ 20 & 82 & 61 \\ 8 & 64 & 22 \end{bmatrix}$$


```

Matrix 객체는 리스트나 딕셔너리와 같이 가변 객체이다. 따라서 불가변성이 요구되는 곳(예를 들어 딕셔너리의 키)에는 사용할 수 없다. 불가변 행렬이 필요하다면 ImmutableMatrix 클래스를 이용해야 한다.

13.8.2 행렬의 연산

파이썬의 산술연산자를 행렬의 연산에 그대로 사용할 수 있다. 단, 덧셈과 뺄셈은 두 행렬의 크기가 동일해야 하고 곱셈의 경우에는 앞 행렬의 열수와 뒷 행렬의 행수가 같아야 한다. 숫자로만 이루어진 행렬 뿐만 아니라 대수기호가 포함된 행렬의 연산도 가능하다.

〈표 13.8.2〉 sympy 행렬의 연산

연산	설명 (A,B는 행렬 객체)
$A+B$	행렬간 덧셈
$A-B$	행렬간 뺄셈
$k*A, A*k$	(k는 상수) 행렬과 스칼라 곱셈
$A*B$	행렬간 곱셈
$A**k$	(k는 상수) 행렬의 거듭제곱
$v1.dot(v2)$	(v1, v2는 벡터) 벡터의 내적

행렬간 곱셈에 교환법칙은 성립하지 않는다.

```
>>> A=randMatrix(3,4)
>>> A

$$\begin{bmatrix} 89 & 97 & 70 & 40 \\ 60 & 95 & 23 & 29 \\ 20 & 69 & 2 & 21 \end{bmatrix}$$

>>> B=randMatrix(4,2)
>>> B

$$\begin{bmatrix} 41 & 19 \\ 34 & 28 \\ 16 & 89 \\ 40 & 81 \end{bmatrix}$$

>>> A*B

$$\begin{bmatrix} 12633 & 10621 \\ 9567 & 10295 \\ 17494 & 15112 \end{bmatrix}$$

>>> B*A
ShapeError: Matrix size mismatch: (4, 2) * (3, 4).
```

파이썬에서 **가 거듭제곱 연산자이므로 심파이에서도 행렬의 거듭제곱은 ** 연산자로 수행한다.

```
>>> C=Matrix([[1,2],[2,3]])
>>> C**10
```

$$\begin{bmatrix} 514229 & 832040 \\ 832040 & 1346269 \end{bmatrix}$$

거듭제곱을 수행하려면 행렬이 정방행렬이어야 한다.

13.8.3 행렬의 선형대수 연산

<표 13.8.3>에 sympy 행렬 객체를 이용하여 수행할 수 있는 기본적인 선형대수 연산을 나열하였다.

<표 12.8.3> sympy 행렬의 선형대수 연산

연산(A는 Matrix객체)	
A.T	전치행렬(transposition)
A.H	복소전치행렬(hermite conjugation)
A.rank()	행렬의 랭크(rank)
A.det()	행렬식(determinant)
A.inv()	역행렬(inverse matrix)
A.LUsolve(b)	행렬방정식 $Ax=b$ 를 푼다
A.norm()	노름(norm)을 구한다
A.eigenvals(**flags)	행렬의 고유값을 구한다.
A.eigenvects(**flags)	행렬의 고유값과 고유벡터를 구한다.
A.applyfunc(f)	행렬 각 요소에 함수 f를 적용한다.
A.evalf()	행렬 각 요소의 실수 근사값을 구한다.

```
>>> J=randMatrix(3)
```

```
>>> J
```

$$\begin{bmatrix} 24 & 74 & 32 \\ 91 & 57 & 18 \\ 24 & 13 & 78 \end{bmatrix}$$

```
>>> J.T
```

$$\begin{bmatrix} 81 & 36 & 18 \\ 10 & 79 & 50 \\ 2 & 96 & 77 \end{bmatrix}$$

```
>>> J.det()
```

```
432756
```

```
>>> J.inv()
```

$$\begin{bmatrix} \frac{3641}{265998} & -\frac{1481}{265998} & -\frac{472}{132999} \\ -\frac{212}{44333} & \frac{1}{44333} & \frac{749}{44333} \\ -\frac{311}{132999} & \frac{2720}{132999} & -\frac{2038}{132999} \end{bmatrix}$$

선형대수 연산은 숫자로만 이루어진 행렬뿐만 아니라 대수기호가 포함된 행렬에 대한 연산도 가능하다.

```
>>> F=Matrix([[x,2],[1,x]])
>>> F

$$\begin{bmatrix} x & 2 \\ 1 & x \end{bmatrix}$$

>>> F.det()
 $x^2 - 2$ 
>>> F.inv()

$$\begin{bmatrix} -\frac{x}{2-x^2} & \frac{2}{2-x^2} \\ \frac{1}{2-x^2} & -\frac{x}{2-x^2} \end{bmatrix}$$

>>> G=ones(2)*x
>>> G

$$\begin{bmatrix} x & x \\ x & x \end{bmatrix}$$

>>> F*G

$$\begin{bmatrix} x^2 + 2x & x^2 + 2x \\ x^2 + x & x^2 + x \end{bmatrix}$$

>>> G.applyfunc(lambda x:x**3)

$$\begin{bmatrix} x^3 & x^3 \\ x^3 & x^3 \end{bmatrix}$$

```

이제 다음과 같은 행렬의 행렬식이 0이 되기 위한 x 값을 구하는 문제를 풀어보자.

$$\begin{bmatrix} x & x^2 & 1 \\ 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}$$

```
>>> H=Matrix([[x,x**2,1],[2,3,4],[5,6,7]])
>>> solve(H.det(),x)

$$\left[-\frac{1}{2}, 1\right]$$

```

위와 같이 sympy의 solve()함수를 이용하면 방정식 $\det(H)=0$ 을 만족시키는 해들을 구할 수 있다. 이 결과로부터 해는 1 또는 $-\frac{1}{2}$ 임을 알 수 있다. 이와 같이 손계산으로 수행하기 복잡한 문제라도 sympy를 이용하면 쉽게 해결할 수 있다.

13.9 미분방정식의 해 구하기 (선택)

어떤 수 x 가 있는데 이 수가 무엇인지는 모르지만 예를 들어 $x^2+x+1=0$ 이라는 방정식을 만족하는 것을 알고 있다. 이 방정식으로부터 x 가 실제 무슨 수인지를 구하는 과정을 ‘(대수)방정식을 푼다’ 라고 한다. 미분방정식이란 함수에 대한 방정식에도 함수가 포함된 것을 의미한다. 구하려는 대상은 ‘함수’이다. 독립변수 x 의 어떤 함수 $y(x)$ 가 있는데 그 함수가 실제로 어떤 모양인지를 알 수가 없다고 하자. 함수의 수식은 모르지만 이 함수에 대해서 $y'+y=0$ 이라는 도함수가 포함된 방정식 (미분방정식)을 만족한다는 사실을 알고 있다. 그러면 이 미분방정식으로부터 이것을 만족하는 미지의 함수 $y(x)$ 을 구하는 것을 ‘미분방정식을 푼다’고 말한다.

sympy를 이용하면 간단한 미분방정식의 해를 구할 수 있다. 먼저 미지의 함수 객체를 생성해야 하는데 이를 위해서 Function 클래스를 이용한다.

```
>>> x=Symbol('x')
>>> y=Function('y')(x) #❶
>>> y
y(x)
```

Function객체를 생성할 때 ❶과 같이 독립변수를 지정해 줄 수 있다. 이렇게 하면 y 는 x 의 (미지의) 함수로 사용할 수 있다. 함수 y 의 독립변수에 대한 미분은 diff()메서드를 호출하면 구할 수 있다. 즉, $y.diff()$ 는 $\frac{dy}{dx}$ 객체를 반환한다.

```
>>> y.diff()
```

$$\frac{d}{dx}y(x)$$

```
>>> y.diff(x,2) #❶
```

$$\frac{d^2}{dx^2}y(x)$$

❶은 $y(x)$ 의 x 에 대한 2계 도함수를 반환하는데 `diff()`메서드의 두 번째 인수에 미분 차수를 지정함으로써 함수의 고계 도함수를 얻을 수 있다. 이것으로부터 미분방정식 $y' + y = 0$ 의 우변은 `y.diff()+y` 라고 입력할 수 있다. 이제 sympy의 `dsolve()` 함수를 이용하면 미분방정식 $y' + y = 0$ 을 만족하는 함수 y 를 구할 수 있다.

```
>>> dsolve(y.diff()+y,y)
```

$$y(x) = C_1 e^{-x}$$

여기서 C_1 은 임의의 상수이다. (엄밀히 말하면 `dsolve()`함수는 `Eq` 클래스의 객체를 반환한다.) 다른 예로 다음과 같은 미분방정식의 해를 구해보자.

$$y' = e^{2x-1} y^2$$

이것을 위해서 우변을 0으로 만들면 $y' - e^{2x-1} y^2 = 0$ 이 되므로 다음과 같이 입력하면 된다.

```
>>> dsolve(y.diff()-exp(2*x-1)*y**2,y)
```

$$y(x) = -\frac{2e}{C_1 + e^{2x}}$$

다른 예로 다음과 같은 선형 2차 미분방정식의 해를 구해보자.

$$y'' + y' - 2y = 0$$

```
>>> dy = y.diff()
```

```
>>> d2y = y.diff(x,2)
```

```
>>> dsolve(d2y+dy-2*y, y)
```

$$y(x) = C_1 e^{-2x} + C_2 e^x$$

여기서 C_1, C_2 는 임의의 상수이다.

제 14 장 micro:bit

14.1 마이크로파이썬

지금까지는 작성한 파이썬 코드가 모두 PC 위에서 실행되었다. 근래에는 파이썬은 피지컬 컴퓨팅(physical computing)을 위한 마이크로컨트롤러의 프로그래밍 언어로도 널리 사용되고 있다. 피지컬 컴퓨팅이란 실제 동작하는 물리적인 장치(센서, 모터 등)를 제어하는 것을 의미하며 아두이노(arduino)와 같은 마이크로컨트롤러 기반의 제어 보드가 널리 사용되고 있다. 초기의 아두이노는 C++로만 코딩할 수 있었는데 근래에는 마이크로파이썬(micro-python)을 이용할 수 있는 보드들이 많이 출시되어 있다. 마이크로컨트롤러는 PC에 비해서 성능이나 메모리에 제약이 많기 때문에 PC에서 돌아가는 파이썬 인터프리터를 그대로 사용할 수는 없다. 마이크로파이썬은 마이크로컨트롤러 기반의 보드들을 제어하는데 사용되는 파이썬 인터프리터인데 성능과 자원이 제한된 환경에서 돌아갈 수 있도록 설계된 파이썬 인터프리터이다. 본 교재에서 설명한 파이썬3 문법과 기본 모듈을 거의 그대로 사용할 수 있다.

이것을 실습하기 위해서 본 장에서는 micro:bit v2 보드에 대해서 알아보고 실습해 보도록 한다. 이 보드의 외양은 <그림 4.1.1>과 같다.



<그림 4.1.1> micro:bit v2 보드의 외양

영국의 BBC에서 개발하여 배포하고 있는 micro:bit 보드는 <그림 4.1.1>과 같은 외양을 가지고 있다. 이 보드는 LED를 비롯하여 온도센서, 기울기센서, 마이크로폰, 스피커등 여러 가지 부품을 내장하고 있으며 이들을 파이썬 코드로 손쉽게 제어할 수 있다. micro:bit 보드를 제어하기 위해서는 microbit 패키지를 임포트해야 한다.

```
>>> from microbit import *
```

microbit 패키지에는 micro:bit 보드의 LED를 제어하는 display 모듈, 버튼의 상태를

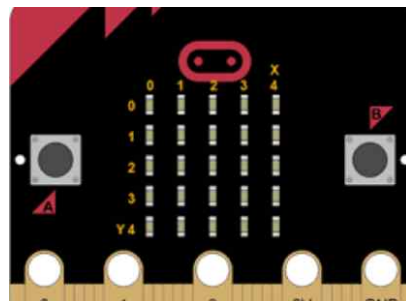
읽어들이는 함수 등 여러 가지 모듈이 포함되어 있다.

14.2 microbit.display 모듈

micro:bit에서는 5행 5열로 배열된 LED들이 있는데 이것을 이용하여 문자나 간단한 이미지를 표시할 수 있다. 이 LED배열을 제어하기 위해서 microbit 패키지지의 display 모듈을 이용하면 된다. 다음 <표 13.2.1>에 display모듈의 함수들과 그 기능을 나열하였다.

<표 14.2.1> microbit.display 모듈의 함수

함수	설명
get_pixe(x,y)	(x,y)좌표의 LED의 상태값을 읽는다 (0에서 9까지의 정수)
set_pixel(x,y,val)	(x,y)좌표의 LED를 밝기 설정 val은 0이상 9이하의 정수
show(image)	image를 표시
scroll(문자열)	
clear()	모든 표시 정보를 삭제하고 초기화
on()	LED배열을 켜다
off()	LED배열을 끈다(표시 내용 보존)
is_on()	LED가 동작중이면 True반환



(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)

<그림 13.2.2> LED의 좌표

off()함수는 LED배열 전체를 끄는 함수이고 on()함수는 LED배열 전체를 켜는 함수이다. off()함수는 현재 LED표시 정보를 보존하지만 clear()함수는 모든 정보를 삭제한 후 초기화한다. set_pixel(x,y,val)함수는 (x,y)좌표의 LED의 밝기를 0부터 9까지 열 단 계로 설정한다. 0은 끄는 것이고 9는 가장 밝게 켜는 것이다. get_pixel(x,y)함수는 (x,y)좌표의 LED의 밝기를 반환하는데 0부터 9사이의 정수값을 얻는다. display.show()함수를 이용하면 미리 정해진 이미지도 표시할 수 있다.

```
>>> display.show(Image.HEART)
```


Image.HEART는 Image클래스의 클래스변수로써 미리 정의된 것이다. 기정의된 변수들은 다음과 같다.

Image.

HEART	HEART_SMALL	HAPPY	SMILE
SAD	CONFUSED	ANGRY	ASLEEP
SURPRISED	SILLY	FABULOUS	MEH
YES	NO	CLOCK12	CLOCK11
CLOCK10	CLOCK9	CLOCK8	CLOCK7
CLOCK6	CLOCK5	CLOCK4	CLOCK3
CLOCK2	CLOCK1	ARROW_N	ARROW_NE
ARROW_E	ARROW_SE	ARROW_S	ARROW_SW
ARROW_W	ARROW_NW	TRIANGLE	TRIANGLE_LEFT
CHESSBOARD	DIAMOND	DIAMOND_SMALL	SQUARE
SQUARE_SMALL	RABBIT	COW	MUSIC_CROCHET
MUSIC_QUAVER	MUSIC_QUAVERS,	.PITCHFORK,	XMAS
PACMAN	TARGET	ROLLERSKATE	DUCK
TSHIRT	HOUSE	TORTOISE	BUTTERFLY
STICKFIGURE	GHOST	SWORD	GIRAFFE
SKULL	UMBRELLA	SNAKE	

Image 객체를 여러 개 묶어서 하나의 리스트로 만들어서 display.show()함수의 첫 번째 인수로 넘겨줄 수 있다. 이 경우 순차적으로 리스트 안의 Image객체를 표시하는데 한 장의 이미지를 유지할 시간(ms)을 delay 키워드 인수로 넘겨주면 된다.

heartbeat.py

```
from microbit import *

hearts=[Image.HEART_SMALL, Image.HEART]
display.show(hearts,delay=500,loop=True)
```

위에 보면 Image.CLOCK1부터 Image.CLOCK12까지의 변수가 있는데 이것을 리스트로 묶어놓은 것이 Image.ALL_CLOCKS이다. 이를 이용하여 다음과 같이 간단한 초시계도 구현할 수 있다.

clock.py

```
from microbit import *

display.show(Image.ALL_CLOCKS, delay=1000, loop=True)
```

ARRROW로 시작하는 모든 Image 객체를 모아놓은 Image.ALL_ARROWS 리스트도 있

다.

정해진 모양만 출력할 수 있는 것은 아니고 직접 원하는 모양을 만들 수도 있다. show()함수를 이용하면 Image 객체나 문자열을 표기할 수 있다.

```
# Add your Python code here. E.g.
from microbit import *
boat = Image('05050:05050:05050:99999:09990')#❶
display.show(boat)
```

❶에서 Image 생성자에 문자열을 넘겼는데 스물 다섯 개의 숫자로 구성되어 있는데 다섯 개씩 콜론(:)으로 구분되어 있다. 각 숫자들은 LED의 밝기(0~9)를 지정한 것이다.

display.show()함수로 문자열도 표시할 수 있으나 display.scroll()함수를 이용하는 것이 좀 더 자연스러운 움직임을 보여준다.

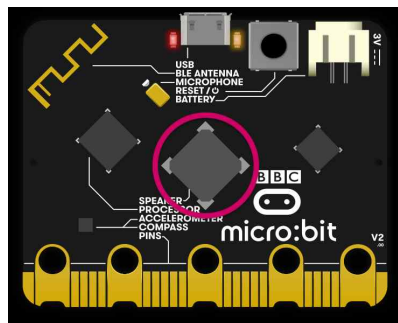
```
>>> display.scroll('hello')
```

이렇게 하면 문자열이 왼쪽으로 스크롤되면서 차례로 LED배열에 표시된다. display.scroll()함수는 키워드 매개변수를 가지고 있는데 loop, clear, wait, delay 등이다. 만약 문자표시를 계속 반복하고 싶다면 다음과 같이 하면 된다.

```
>>> display.scroll('micro:bit',wait=False,loop=True)
```

loop=True 키워드 인수는 문자를 무한히 반복해서 표시토록 하는 것이고, wait=False 키워드 인수는 곧바로 그 다음 명령을 수행할 수 있는 상태로 만들어 준다.

14.3 스피커 출력



마이크로비트 V2에는 스피커가 내장되어 있는데 이 스피커를 이용하여 다양한 소리를 발생시킬 수 있다.

14.3.1 microbit 의 스피커 관련 모듈

사용되는 모듈은 audio, speaker가 있고 Sound, Music 클래스를 이용하면 된다.

<표 14.3.1> micro:bit v2의 스피커 관련 함수와 클래스

메서드/객체	기능
speaker.on() speaker.off()	스피커를 켜다. 스피커를 끈다.
audio.play() audio.is_playing() audio.stop() audio.AudioFrame	Sound 객체를 플레이한다. 현재 플레이중이면 True반환. 플레이를 중단한다. AudioFrame 클래스
Sound	미리 샘플링된 오디오 데이터가 클래스변수로 저장돼 있다.(GIGGLE, HAPPY, HELLO, MYSTERIOUS, SAD, SLIDE, SOARING, SPRING, TWINKLE, YAWN)

먼저 speaker 모듈의 on(), off() 메서드를 이용하여 스피커를 켜거나 끌 수 있다. 그리고 audio.play()메서드를 이용하면 Sound 클래스에 저장된 소리 데이터를 재생시킬 수 있다.

```
from microbit import *

audio.play(Sound.HELLO)
```

14.3.2 마이크로파이썬의 music 모듈

마이크로파이썬에는 music 모듈이 있는데 이것을 이용해서 음을 발생시킬 수도 있다. 이것은 microbit 패키지의 모듈이 아니라 마이크로파이썬의 기본모듈이므로 별도로 임포트해야 한다. music 모듈에는 멜로디를 연주하기 위한 music.play() 함수와 특정한 주파수의 음을 발생시키기 위한 music.pitch() 함수가 있다. 또한 다양한 멜로디가 객체로 미리 저장되어 있다. 멜로디의 전체적인 빠르기는 ticks와 bpm 값으로 조절할 수 있는데 music.set_tempo()함수를 이용하여 이 값들을 조절할 수 있다.

〈표 14.3.2〉 micro:bit v2의 스피커 관련 메서드와 객체

메서드/객체	기능
music.play(music, wait=True, loop=False)	음악을 재생한다.
music.pitch(freq, duration=-1, wait=True)	주파수 음을 발생. duration은 ms단위
music.stop()	play, pitch를 멈춘다.
music.set_tempo(ticks=4,bpm=120)	연주 속도를 변경한다.
music.reset()	내부의 상태값을 아래의 값들로 초기화시킨다. ticks=4, bpm=120, octave=4, duration=4

아래와 같은 간단한 코드로 미리 저장된 멜로디를 연주할 수 있다.

```
import music

music.play(music.NYAN)
```

music 모듈에 미리 저장된 멜로디 객체는 다음과 같다.

〈표 14.3.2〉 music모듈에 저장된 멜로디 객체들

DADADADUM	ENTERTAINER	PRELUDE	ODE
NYAN	RINGTONE	FUNK	BLUES
BIRTHDAY	WEDDING	FUNERAL	PUNCHLINE
PYTHON	BADDY	CHASE	BA_DING
WAWAWAWAA	JUMP_UP	JUMP_DOWN	POWER_UP
POWER_DOWN			

멜로디를 임의로 작성할 수도 있는데 음표 정보를 리스트에 문자열로 저장해 주는 방식이다. 우리가 흔히 아는 음과 기호의 관계는 아래의 표와 같다.

〈표 13.3.2〉 음이름과 기호의 관계표

도		레		미	파		솔		라		시
c	c#	d	d#	e	f	f#	g	g#	a	a#	b

예를 들어서 'a1:4'는 a1(1옥타브 라음) 4분음표를 의미한다. (음기호 대신 'r'을 쓰면 쉼표를 의미한다.) 콜론 뒤의 숫자는 음길이를 말하는데 4는 4분음표, 8은 2분음표를 의미한다. 즉, 콜론 뒤에 오는 숫자가 클수록 음길이가 길어진다. 플랫과 �arp은 b와 #

기호를 사용한다. 예를 들면 'ab', 'c#' 등이다. 옥타브의 기본값은 4이고, 음길이의 기본값도 4이다.

```
import music

tune = [
    'c', 'd', 'e', 'c', 'c', 'd', 'e', 'c', 'e', 'f',
    'g:8', 'e:4', 'f', 'g:8'
]
music.play(tune)
```

위 예에서 옥타브와 음길이의 초기값은 모두 4이므로(octave=4, duration=4) 맨 처음 'c'는 'c4:4'와 같다. 두 번째 음부터는 만약 옥타브나 음길이가 생략되면 기본값이 아니라 직전에 연주된 음의 옥타브와 음길이를 연주한다는 점에 주의하자. 따라서 'g:8' 다음에 그냥 'e'라고 해버리면 'e:4'가 아니라 'e:8'로 연주되므로 주의해야 한다.

music.pitch()메서드는 주어진 주파수의 음을 발생시키는 기능을 하며 효과음을 간단하게 생성하는데 사용된다. 첫 번째 인수는 주파수, 두 번째 인수는 음길이(duration)인데 ms 단위의 숫자로 넘겨주면 된다.

```
import music

for freq in range(880, 1760, 16):
    music.pitch(freq, 6)
for freq in range(1760, 880, -16):
    music.pitch(freq, 6)
```

두 번째 인수인 음길이를 지정하지 않으면 계속 연주된다.

14.3.3 speech 모듈

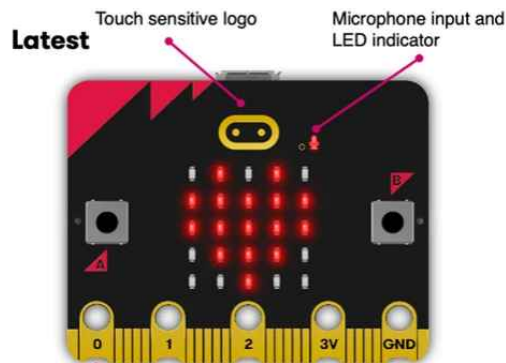
마이크로비트 v2 보드의 스피커에서 음성을 출력할 수 있는데 speech모듈의 say() 함수를 이용하면 된다.

```
import speech

speech.say('Hello, world')
```

하지만 하드웨어의 제약 때문에 그리 음성이 그리 또렷하지는 않다.

14.4 버튼 입력



〈그림 14.4.1〉 마이크로비트 v2 보드의 버튼 터치로고

마이크로비트 보드에는 두 개의 누름버튼(push button)이 있는데 각각 버튼a, 버튼b로 구분되어 있다. 누름버튼을 제어하기 위해서 microbit 모듈을 импорт하면 button_a 와 button_b 객체가 있는데 다음과 같은 메서드들이 있다.

〈표 14.4.1〉 Button클래스 객체의 메서드들

메서드	기능
is_pressed()	현재 눌러져 있다면 True 반환
was_pressed()	직전에 was_pressed()메서드를 호출한 이후(호출된 적이 없다면 전원이 켜진 이후)에 버튼이 눌린 적이 있다면 True 반환
get_pressed()	get_pressed()메서드를 호출한 이후(호출된 적이 없다면 전원이 켜진 이후)부터 현재까지 눌린 횟수 반환

예를 들면 다음과 같다.

```
button1.py

from microbit import *

while True:
    if button_a.is_pressed() and button_b.is_pressed():
        display.scroll("AB")
        break
```

```
elif button_a.is_pressed():
    display.scroll("A")
elif button_b.is_pressed():
    display.scroll("B")
sleep(100)
```

이 예제는 버튼a가 눌렸을 때, 버튼b가 눌렸을 때, 그리고 둘 다 동시에 눌렸을 때 특정한 문자열을 표시하는 동작을 한다.

마이크로비트 v2 보드의 금속 로고는 터치센서가 연결되어 있다. 이것의 상태를 알기 위해서는 pin_logo 객체의 is_touched() 메서드를 이용하면 된다. 만약 로고가 터치된 상태라면 True를 반환한다.

touchlogo.py

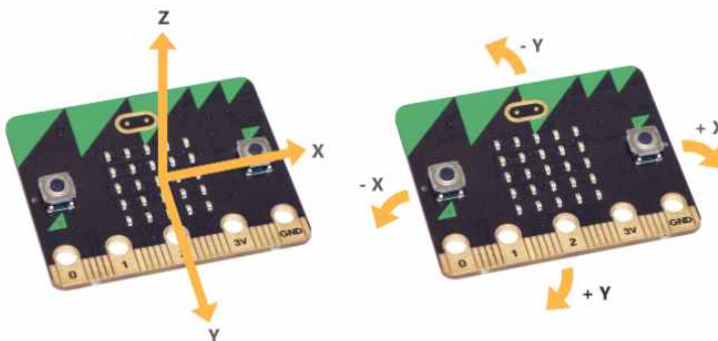
```
from microbit import *

while True:
    if pin_logo.is_touched():
        display.show(Image.SURPRISED)
```

위 프로그램은 로고를 터치하면 Image.SURPRISED 를 LED 배열에 출력하는 예제이다.

14.5 움직임 감지

마이크로비트 보드에는 가속도계(accelerometer)가 내장되어 있다. 가속도계를 이용하면 x/y/z축 세 방향의 가속도값을 읽어낼 수 있다. 이 값들을 이용하여 현재 보드가 어느 방향으로 얼마만큼 기울어져 있는지를 판단할 수 있으며 흔들림이나 자유낙하 등 간단한 동작도 감지할 수 있다.



<그림 14.5.1> 가속도계의 방향

위의 <그림 14.5.1>에 도시되어 있듯이 x/y축은 그 방향으로의 기울기 정도를 그리고 z축은 위/아래 방향으로의 움직임을 의미한다. microbit.accelerometer 모듈에 가속도계 관련 함수들이 제공되며 <표 14.5.1>에 관련 함수들과 그 기능들을 정리하였다.

<표 14.5.1> accelerometer 모듈 함수와 기능

accelerometer 모듈 함수	기능
get_x() get_y() get_z()	x축, y축, z축의 가속도 값을 읽는다. 단위는 mg(milli-g)이고 범위는 -2000~2000 사이의 정수이다.
get_values()	x축, y축, z축의 가속도 값을 튜플로 한 번에 반환
current_gesture()	현재 동작("up", "down", "left", "right", "face up", "face down", "freefall", "3g", "6g", "8g", "shake")을 판독하여 반환
is_gesture(name)	현재 동작이 name(문자열)과 같으면 True 반환
was_gesture(name)	직전의 was_gesture()함수를 호출한 이후에 name(문자열)과 같은 동작이 발생했다면 True 반환
get_gestures()	직전의 get_gestures()함수 호출 이후에 발생한 모든 동작을 튜플로 반환

다음 예제는 가속도계의 x/y/z축 값을 읽어서 표시하는 것이다.

accel.py

```
from microbit import *

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    z = accelerometer.get_z()
    print("x, y, z:", x, y, z)
    sleep(500)
```

<표 14.5.1>에 보면 current_gesture(), is_gesture(), was_gesture(), get_gestures() 함수들은 특정한 동작을 의미하는 문자열을 사용한다. 이들 중에서 'left' / 'right' / 'up' / 'down'은 각 방향으로의 움직임을, 'face up'/'face down'은 LED 배열이 위로 향하고 있는지 혹은 아래로 향하고 있는지를, 'freefall', 'shake'는 자유낙하와 흔들림을, 그리고 '3g', '6g', '9g'는 가속력을 의미한다. 다음 예제는 이 중 'face up'을 검출하여 LED 배열에 표시되는 이미지를 변경시키는 것이다.


```

accel2.py

from microbit import *

while True:
    gesture = accelerometer.current_gesture()
    if gesture == "face up":
        display.show(Image.HAPPY)
    else:
        display.show(Image.ANGRY)

```

이와 같이 가속도계를 이용하면 마이크로비트 보드의 위치와 움직임에 관련된 정보를 검출할 수 있다.

14.6 전자 나침판

마이크로비트 보드는 전자 나침판(electronic compass)도 내장되어 있는데 compass 모듈에 관련 함수가 마련되어 있다.

<표 14.6.1> microbit.compass 모듈 함수와 기능

compass 모듈 함수	기능
calibrate()	초기 조정을 수행한다.
is_calibrated	초기 조정이 성공적으로 수행되었다면 True 반환
clear_calibration()	조정을 취소한다.
get_x() get_y() get_z()	해당 축의 자기장 강도를 반환한다.
heading()	방향값을 반환한다. 북쪽을 (0, 360)범위의 정수값이고 북쪽이 0이고 시계방향으로 값이 증가한다.
get_field_strength()	자기장 강도(nano tesla)를 정수값을 반환

나침판을 구현한 프로그램의 아래와 같이 작성할 수 있다.

```

compass.py

from microbit import *

compass.calibrate()

```

```
while True:
    sleep(100)
    needle = ((15 - compass.heading()) // 30) % 12
    display.show(Image.ALL_CLOCKS[needle])
```

이 프로그램에서는 내장된 CLOCKS 이미지들을 이용하여 항상 북쪽을 가리키게 된다.

부록

부록.1 내장함수

부록.2 매직메서드

부록.3 기본모듈