2. IPython 기초

파이썬 코드를 입력하고 Enter 키를 눌러서 실행시킬수 있다. 파이썬에 그냥 변수 이름만 입력하면 그 객체의 문자열 표현을 출력한다.

```
In [4]: data = {i : np.random.randn() for i in range(7) }

In [5]: data
Out[5]:
{0: -1.5804442179675395,
 1: -0.37078872808065644,
 2: 0.37889240710688427,
 3: 0.1998092914913129,
 4: -0.18685810574185185,
 5: 0.623615479087203,
 6: 0.4520943183327313}
```

첫 줄은 파이썬 코드인데, 두 번째 줄에서 data라는 이름의 변수를 생성하고 새로 생성한 파이썬 사전형(딕셔너리)을 참조하도록 했다. 마지막 줄은 data 변수의 값을 출력한다.

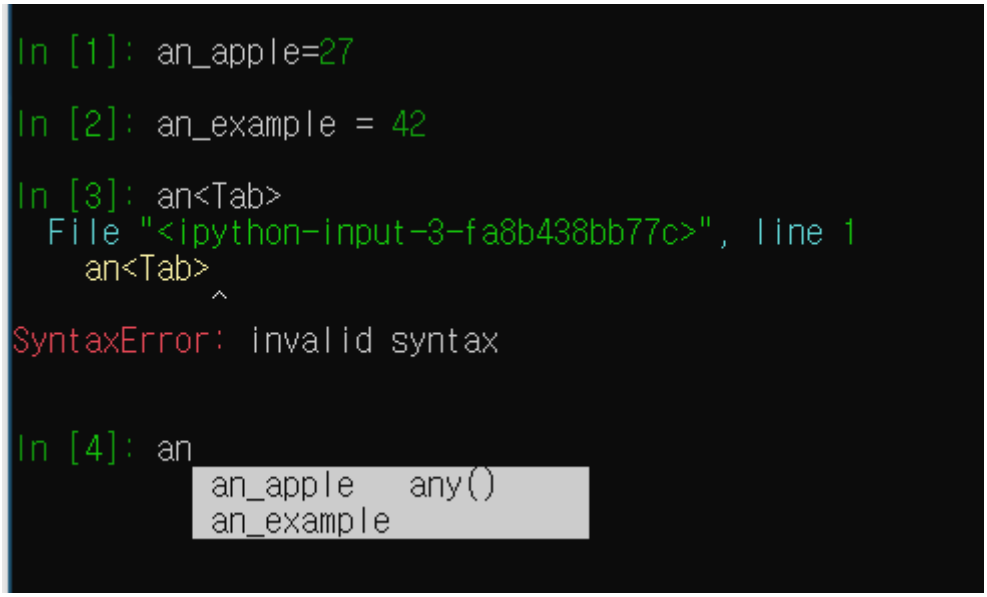
대부분의 파이썬 객체는 print를 이용한 보통의 출력 결과와는 달리 좀 더 읽기 편하거나 보기 좋은 형태로 출력된다. 위 예제를 표준 파이썬 인터프리터에서 보면 다소 읽기 불편한 형태로 출력된다.

```
>>> from numpy.random import randn
>>> data = {i : randn() for i in range(7)}
>>> print(data)
{0: 0.6123135870967825, 1: 0.9521743860753571, 2: 0.32769853402623367, 3:
1.169923116814086, 4: -0.14833565168973253, 5: -0.38331288921734147, 6:
-0.8618985537005338}
>>>
```

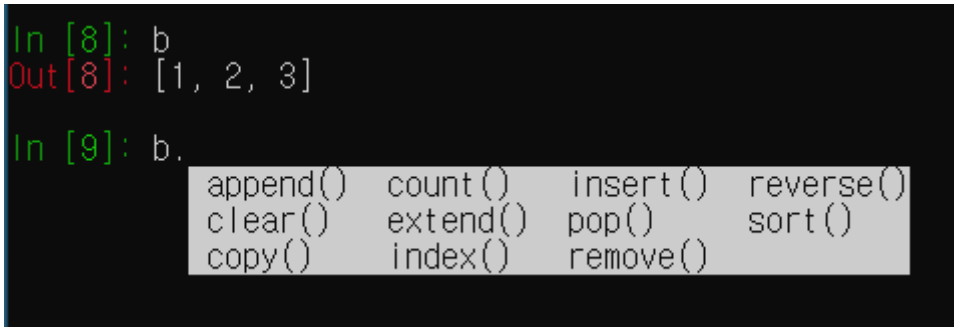
또한 IPython은 일부 코드(약간 개선된 복사/붙여넣기 기능을 통해)아니면 전체 파이썬 스크립트를 쉽게 실행할 수 있는 기능을 제공한다. 또한 주피터 노트북을 이용해서 많은 분량의 코드를 다룰수 있다.

2.2.3 탭 자동완성

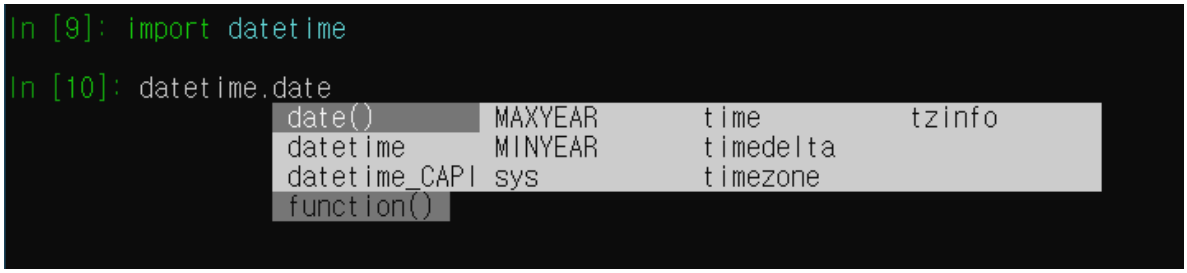
겉으로 보기에 IPython은 표준 파이썬 인터프리터와는 조금 다르게 생겼다. 표준 파이썬 셸에 비해 가장 두드러진 개선은 **탭을 통한 자동완성 기능**으로, 대부분의 통합 개발 환경이나 대화형 데이터 분석 환경에 구현되어 있는 기능이다. 셸에서 입력을 하는 동안 탭을 누르면 네임스페이스에서 그 시점까지 입력한 내용과 맞아떨어지는 변수(객체, 함수 등)를 자동으로 찾아준다.



이 예제에서 IPtython은 앞서 정의한 두 변수는 불론이고 파이썬 예약어인 and와 내장 함수인 any를 함께 보여주는 것을 확인할 수 있다. 물론 어떤 객체의 메서드나 속성 뒤에 마침표를 입력한 후 자동완성 기능을 활용할 수 도 있다.



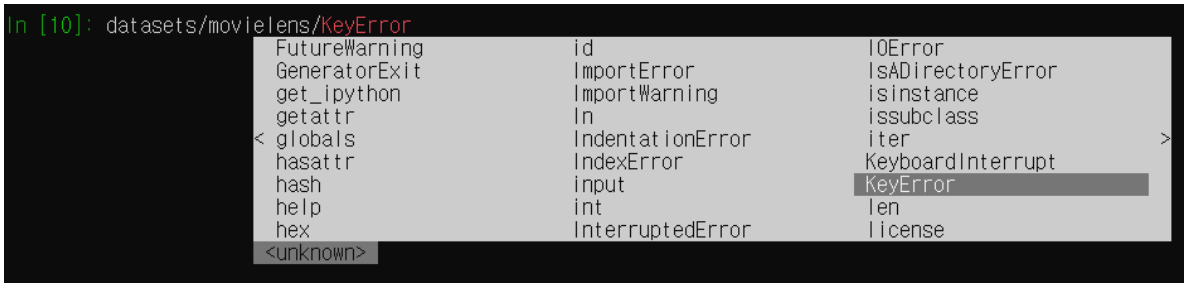
모듈도 똑같이 동작한다.

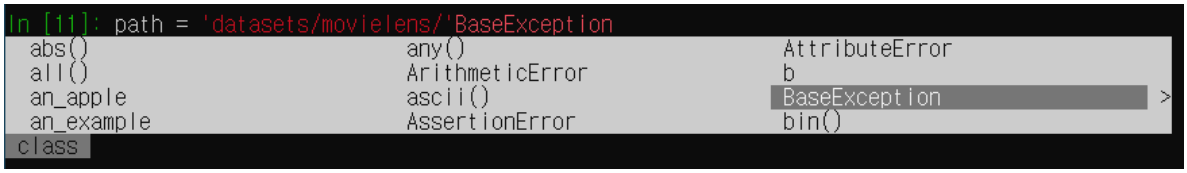


주피터 노트북과 IPython 5.0 이상 버전에서는 자동완성 목록이 일반 텍스트 출력이 아닌 드롭다운 형식이 나타낸다.

Note 탭을 눌렀을 때 화면에 출력 결과가 너무 많으면 초보자는 헷갈릴 수 있는데, IPython은 아예_로 시작하는 내부 메서드와 속성을 제외한 결과를 보여준다. 물론 먼저_를 입력하면 해당 메서드와 속성을 선택할 수 있다. 이런 메서드를 탭 자동완성 목록에 기본으로 넣고 싶다면 IPython 환경 설정에서 설정할 수 있다. 자세한 내용은 IPython 문서를 참고하자.

탭 자동완성은 대화형 네임스페이스 검색과 객체 및 모듈 생성의 자동완성 뿐만 아니라 파일 경로를 입력(파이썬 문자열 안에서도)한 후 탭을 누르면 입력한 문자열에 맞는 파일 경로를 컴퓨터의 파일 시스템 안에서 찾아서 보여준다.





나중에 살펴볼 %run 명령어와 조합하면 키 입력을 줄일 수 있다.

2.2.4 자기관찰

변수 이름 앞이나 뒤에 ?기호를 붙이면 그 객체에 대한 일반 정보를 출력한다.

```
In [16]: b=[1,2,3]

In [17]: b?
Type:        list
String form: [1, 2, 3]
Length:      3
Docstring:
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.
```

```
In [18]: print?
Docstring:
print(value, ..., sep=' ', end='₩n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

이 기능은 객체의 자기관찰(introspection)이라고 하는데, 만약 객체가 함수이거나 인스턴스 메서드라면 정의되어 있는 문서를 출력해 준다. IPython이나 주피터에서 사용 가능한 아래와 같은 함수를 작성했다고 하자.

```
In [20]: def add_numbers(a,b):
    ...:     return a+b
    ...:

In [21]: add_numbers?
Signature: add_numbers(a, b)
Docstring: <no docstring>
File:      c:₩users₩김대현₩<ipython-input-20-af18a5fccf65>
Type:      function
```

또한 ?는 표준 유닉스나 윈도우 명령행에서와 마찬가지로 IPython의 네임스페이스를 검색하는 데 사용할 수도 있다. 와일드카드(*) 기호와 함께 사용한 문자와 일치하는 모든 이름을 보여준다. 예를 들어 Numpy의 최상단 네임스페이스 안에서 load를 포함하는 모든 함수 목록을 가져올 수 있다.

```
In [24]: import numpy as np

In [25]: np.*load*?
np.__loader__
np.load
np.loads
np.loadtxt
```

2.2.5 %run 명령어

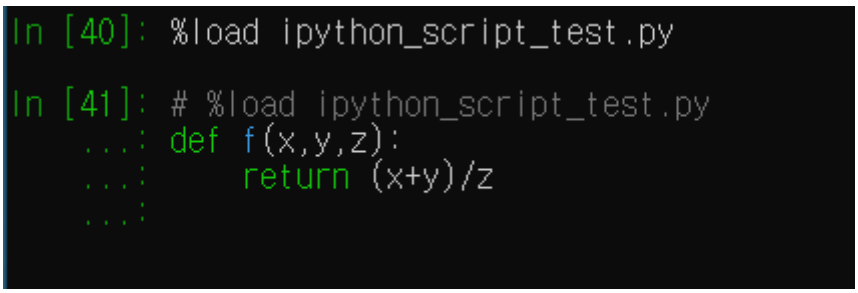
%run 명령어를 사용하면 IPython 세션 안에서 파이썬 프로그램 파일을 불러와서 실행할 수 있다. 다음과 같은 ipython_script_test.py 스크립트 파일이 있다고 치자

```
In [27]: %run ipython_script_test.py

In [28]: a=5

In [29]: b=6

In [30]: c=7.5

In [31]: result = f(a,b,c)

In [32]: c
Out[32]: 7.5

In [33]: result
Out[33]: 1.4666666666666666
```

만약 파이썬 스크립트에 명령행 인자 (sys.argv에 저장되는)를 넘겨야 한다면 명령행에서 실행하는 것처럼 파알 경로 다음에 필요한 인자를 넘겨주면 된다.

Note 실행하려는 스크립트 파일에서 대화형 IPython 네임스페이스에 따라 선언된 변수에 접근해야 한다면 %run -i 명령을 사용한다.

주피터 노트북에서는 스크립트 파일을 코드 셀로 불러오는 %load 매직함수를 사용할 수도 있다.



실행중인 코드 중지하기

%ru을 통해 스크립트가 실행되고 있거나 오랜 실행 시간이 필요한 코드가 실행되고 있는 중간에 Ctrl+c를 누르면 keyboardInterrupt예외를 발생시킨다. 이 예외는 몇몇 특수한 경우를 제외한 거의 모든 파이썬 프로그램을 즉시 멈추도록 한다.

CAUTION 싱행 중인 파이썬 코드가 컴파일된 확장 모듈에서 호출된 경우에는 Ctrl + C 를 눌러도 프로그램이 즉각 멈추지 않는데, 이런 경우에는 프로그램의 제어권이 파이썬 인터프리터로 되돌아올 때 까지 기다리거나 심각한 경우에는 OS의 작업 관리자 메뉴를 이용해 파이썬 프로세스를 강제로 종료해야 한다.

2.2.7 키보드 단축기

커서 이동

ctrl + a : 커서 맨 앞으로

ctrl + e : 커서 맨 뒤로

ctrl + b : 커서 한 칸 뒤로

ctrl + f : 커서 한 칸 앞으로

명령 지우기

ctrl + u : 명령어 전체 지우기

2.2.8 매직 명령어

IPython은 파이썬 자체에는 존재하지 않은 '매직'명령어라고 하는 여러 가지 특수한 명령어를 포함하고 있다. 이 매직 명령어는 일반적인 작업이나 IPython 시스템의 동작을 쉽게 제어할 수 있도록 설계된 특수한 명령어이다. 매직 명령어는 앞에 % 기호를 붙이는 형식인데, 예를 들어 IPython에서 행렬 곱셈 같은 코드가 실행된 시간을 측정하고 싶을 때는 %timeit 매직함수를 이용해서 값을 얻어올 수 있다.

```
In [12]: a = np.random.randn(100,100)

In [13]: %timeit np.dot(a,a)
46.5 µs ± 1.71 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

매직명령어는 IPython 시스템 안에서 실행되는 명령행 프로그램으로 볼 수 있는데, 많은 매직 명령어는 주기적인 옵션을 필요로 하며 ?를 이용해서 전체 옵션을 살펴볼 수 있다.

```
In [14]: %debug?
Docstring:
::

  %debug [--breakpoint FILE:LINE] [statement [statement ...]]

Activate the interactive debugger.
```

```
This magic command support two ways of activating debugger.
One is to activate debugger before executing code.  This way, you
can set a break point, to step through the code from the point.
You can use this mode by giving statements to execute and optionally
a breakpoint.

The other one is to activate debugger in post-mortem mode.  You can
activate this mode simply running %debug without any argument.
If an exception has just occurred, this lets you inspect its stack
frames interactively.  Note that this will always work only on the last
traceback that occurred, so you must call this quickly after an
exception that you wish to inspect has fired, because if another one
occurs, it clobbers the previous one.

If you want IPython to automatically do this on every exception, see
the %pdb magic for more details.

.. versionchanged:: 7.3
    When running code, user variables are no longer expanded,
    the magic line is always left unmodified.

positional arguments:
---Return to continue, q to quit---
```

매직함수와 같은 이름의 변수가 선언되지 있지 않다면 기본적으로 %기호 없이도 매직함수를 사용할 수 있다. 이를 **오토매직**이라고 하는데, %automagic을 사용해서 이 기능을 켜거나 끌 수 있다.

일부 매직함수는 파이썬 함수처럼 동작하며 결과를 변수에 대입할 수도 있다.

2.2.9 matplotlib 통합

IPython이 분석 컴퓨팅 환경에서 널리 사용되고 있는 이유는 데이터 시각화를 위한 matplotlib 이나 다른 사용자 인터페이스 라이브러리와 잘 통합되어 있기 때문이다. matplotlib은 나중에 더 설명하겠다. %matplotlib 매직함수는 IPython 셸이나 주피터 노트북과 matplotlib 통합을 설정한다. 이 기능을 실행하지 않으면 노트북의 경우 그래프가 화면에 나타나지 않거나 셸의 경우 세션의 제어권을 뺏기게 되므로 중요하다.

IPython 셸에서 %matplotlib을 실행하면 콘솔 세션을 방해받지 않고 여러 플롯 윈도우를 생성할 수 있다.

```
In [16]: %matplotlib
Using matplotlib backend: Qt5Agg
```

2.3 파이썬 기초

2.3.1 시맨틱

파이썬은 가독성과 명료성 그리고 명백함을 강조한다. 어떤 사람은 파이썬을 '실행 가능한 의사코드'라고 표현하기도 한다.

들여쓰기

파이썬은 R, C++, 자바, 펄 같은 다른 많은 언어와는 다르게 중괄호 대신 공백 문자를 사용해서 코드를 구조화한다.

```
for x in array:
    if x < pivot:
        less.append(x)
    else:
        greater.append(x)
```

콜론은 코드 블록의 시작을 의미하며, 블록이 끝날 때까지 블록 안에 있는 코드는 모두 같은 크기만큼 들여써야 한다.

좋든 싫든 공백 문자를 가진다는 것은 파이썬 개발자에게는 일상과 같은 일이다.

모든 것은 객체

파이썬 언어의 중요한 특징 중 하나는 객체 모델의 연관성이다. 모든 숫자, 문자열, 자료구조, 함수, 클래스, 모듈 등은 파이썬 인터프리터에서 파이썬 객체라고 하는 어떤 '상자'안에 저장된다. 각 객체는 연관된 자료형(예를 들면 문자열이나 함수)과 내부 데이터를 갖고 있다. 실제로 이 특징은 심지어는 함수마저도 하나의 객체로 간주함으로써 파이썬을 매우 유연한 언어로 만든다.

cf) 주석은 # 처리한다.

함수와 객체 메서드 호출

함수는 괄호와 0개 이상의 인자를 전달해서 호출할 수 있다. 변환되는 값은 선택적으로 변수에 대입할 수 있다.

변수와 인자 전달

파이썬에서 변수(혹은 이름)에 값을 대입하면 대입 연산자 오른쪽에 있는 객체에 대한 참조를 생성하게 된다. 정수가 담긴 리스트를 생각해보자.

```
In [19]: a = [1,2,3]
In [20]: b=a
In [21]: a.append(4)

In [22]: b
Out[22]: [1, 2, 3, 4]
```

Note 변수에 값을 할당하는 것은 이름을 객체에 연결하는 것이므로 바인딩이라고 부른다. 값이 할당된 변수이름을 때때로 바운드 변수라고 부르기도 한다.

함수에 객체를 인자로 넘기면 새로운 지역 변수가 생성되고 원래 객체를 복사하지 않고 참조만 하게 된다. 만일 함수 안에 있는 어떤 변수에 새로운 객체를 연결한다면 함수 바깥에는 영향을 끼치지 않는다.

```
In [24]: def append_element(some_list, element):
    ...:     some_list.append(element)
    ...:

In [25]: data = [1,2,3]

In [26]: append_element(data,4)

In [27]: data
Out[27]: [1, 2, 3, 4]
```

동적 참조와 강한 타입

자바나 c++ 같은 컴파일 언와는 달리 파이썬에서는 객체 참조에 타입이 관여하지 않는다. 아래 코드는 전혀 문제되지 않는다.

```
In [28]: a =5
In [29]: type(a)

Out[29]: int
In [30]: a ='foo'

In [31]: type(a)
Out[31]: str
In [32]: '5'+5
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-e84694abbbbf> in <module>
----> 1 '5'+5

TypeError: can only concatenate str (not "int") to str
```

In [32]: '5'+5 에서와 같이 문자형과 숫자형을 같이 연산할 수 없다.

isinstance 함수로 객체의 타입을 알 수 있다.

```
In [34]: isinstance(a,int)
Out[34]: False

In [35]: a
Out[35]: 'foo'
```

속성과 메서드

파이썬에서 객체는 일반적으로 속성(객체 내부에 저장되는 다른 파이썬 객체)과 메서드 (객체의 내부 데이터에 접근할 수 있는 함수)를 가진다. 속성과 메서드는 obj.attribute_name 문법으로 접근할 수 있다.

속성과 메서드는 getattr 함수를 통해 이름으로 접근하는 것도 가능하다.

```
In [36]: getattr(a,'split')
Out[36]: <function str.split(sep=None, maxsplit=-1)>
```

다른 언어에서는 이름으로 객체에 접근하는 것을 '리플렉션'이라고 한다. 이 책에서는 getattr함수와 이와 관련있는 hasattr와 setattr 함수를 그다지 사용하진 않지만 이 함수들은 범용적이고 재사용이 가능한 코드를 작성할 때 아주 유용하다.

덕 타이핑

객체의 자료형에는 관심이 없고 그 객체가 어떤 메서드나 행동을 지원하는지만 알고 싶은 경우가 있다. 이를 '덕 타이핑'이라고 부르는데, '만일 오리처럼 걷고 오리처럼 꽤꽥 운다면 그것은 오리다'라는 의미로 지어진 이름이다. 예를 들어, 어떤 객체가 **이터레이터**를 구현했다면 순회가능한 가능한 객체인지 검증할 수 있다. 이는 대부분의 객체의 경우 *iter* 라는 '매직 메서드'를 가지고 있는지 확인하면 된다. 좀 더 나은 방법은 iter 함수를 이용해서 검사하는 것이다.

```
In [39]: def isiterable(obj):
    ...:     try:
    ...:         iter(obj)
    ...:         return True
    ...:     except TypeError: #iterable 객체 아님
    ...:         return False
    ...:
    ...:
In [40]: isiterable('a string')
Out[40]: True

In [41]: isiterable([1,2,3])
Out[41]: True

In [42]: isiterable(5)
Out[42]: False
```

여러 개의 입력을 처리해야 하는 함수를 작성할 때 이 기능을 사용하면 좋다. 보통은 리스트나 튜플, ndarray 처럼 순차적인 자료구조를 다루는 함수를 작성할 때 주로 사용하며 이터레이터를 처리할 때도 사용한다. 먼저 객체가 리스트 인지 검사해서 그렇지 않을 경우 인자를 변환해 줄 수 있다!!

이항 연산자와 비교문

두 참조 변수가 같은 객체를 가리키고 있는지 검사하려면 is 예약어를 사용한다. is not를 사용하면 두 객체가 같지 않은지 검사할 수 있다.

```
In [46]: a = [1,2,3,]

In [47]: b=a

In [49]: c=list(a)

In [50]: a is b
Out[50]: True

In [51]: a is not c
Out[51]: True

In [52]: a is c
Out[52]: False
```

list는 항상 새로운 파이썬 리스트를 생성하므로 c는 a와 구별된다는 점을 명심하자.

is로 비교하는 것과 ==연산자로 비교하는 것은 같지 않다. 이 경우 ==연산자의 결과는 다음과 같다.

```
In [53]: a == c
Out[53]: True
```

is와 is not은 변수가 None인지 검사하기 위해 흔히 사용하는데, None인스턴스는 하나만 존재하기 때문이다.

<이항 연산자>

연산자	설명
a//b	a를 b로 나눈 몫을 취한다.
a**b	a의 b승을 구한다.
a==b	a와 b가 같은 경우 True
a is not b	a와 b가 다른 파이썬 객체를 참조할 경우 True
a is b	a와 b가 같은 파이썬 객체를 참조할 경우 True

뮤터블, 이뮤터블 객체

파이썬에서 리스트, 사전, Numpy 배열 또는 사용자 정의 클래스 같은 대부분의 객체는 변경 가능하다(뮤터블). 이 말은 객체나 값의 내용을 바꿀 수 있다는 뜻이다.

```
In [54]: a_list = ['foo', 2,[4,5]]

In [55]: a_list[2] = (3,4)

In [56]: a_list
Out[56]: ['foo', 2, (3, 4)]
```

문자열이나 튜플은 변경 불가능하다(이뮤터블).

```
In [57]: a_tuple = (3,5,(4,5))

In [58]: a_tuple[1] = 'four'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-23fe12da1ba6> in <module>
----> 1 a_tuple[1] = 'four'

TypeError: 'tuple' object does not support item assignment
```

객체는 변경할 수 있으니 언제든지 변경해도 된다는 의미는 아님을 기억하자. 그런 방식의 사용은 프로그래밍에서 부작용을 유발한다. 어떤 함수를 작성할 때 발생할 수 있는 부작용에 대해 함수의 문서나 주석으로 명시적 설명을 남겨야 한다. 가능하면 변경 가능한 객체를 많이 사용 하더라도 부작용을 피하고 불역성을 잘 활용하기 바란다.

2.3.2 스칼라형

파이썬은 숫자 데이터와 문자열, 불리언값 그리고 날짜와 시간을 다룰 수 있는 몇몇 내장 자료형을 제공한다. 이런 '단일 값'을 담는 자료형을 스칼라 타입이라고 하며 이 책에서도 스칼라 타입으로 부르도록 하겠다. 스칼라형의 목록을 정리하도록 하겠다. 날짜와 시간을 다루는 방법은 표준 라이브러리의 datetime 모듈에서 제공하므로 따로 설명하겠다.

자료형	설명
None	파이썬의 'null' 값(하나의 유일한 None 인스턴스만 존재한다.)
str	문자열 자료형, 유니코드 (UTF-8 인코딩) 문자열
bytes	Raw ASCII 바이트 (또는 바이트로 인코딩된 유니코드)
bool	참(True) 또는 거짓(False)
int	부호가 있는 (음수 표현이 가능한) 정수. 값의 범위는 플랫폼에 의존적이다.
float	배정밀도(64비트) 부동소수점수, double형이 따로 존재하지 않는다는 점을 기억하자.

숫자 자료형

숫자를 위한 파이썬의 주요 자료형은 int와 float이다. int는 임의의 숫자를 저장할 수 있다.

```
In [59]: ival = 17239871

In [60]: ival ** 6
Out[60]: 26254519291092456596965462913230729701102721
```

문자열

많은 사람은 파이썬을 강력하고 유연한 문자열 처리 기능 때문에 애용한다.

문자열은 작은따옴표나 큰따옴표로 둘러싸서 표현할 수 있다.

```
a = '문자열을 작은 따음표로 둘러싼다'
b = "문자열을 처리하는 다른 방법"
```

개행 문자가 포함된 여러 줄에 걸친 문자열은 세 개의 작은 따음표나 큰 따옴표로 둘러싼다.

```
c = """
    여러줄에 걸친 문자열은 따음표 세 개로 둘러싼다.
    """
```

문자열 c가 실제로 4줄의 텍스트를 담고 있다는 사실에 놀랄 수 있다. """ 뒤에 오는 개행 문자도 c에 포함된다. 개행 문자의 수는 count 메서드를 이용해서 확인할 수 있다.

```
In [61]: c = """^M
    ...:     여러줄에 걸친 문자열은 따음표 세 개로 둘러싼다. ^M
    ...:     """

In [63]: c.count('\n')
Out[63]: 2
```

파이썬의 문자열은 변경 불가능하다.

```
In [64]: a = 'this is a string'
```

```
In [65]: a[10] = 'f'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-65-2151a30ed055> in <module>
----> 1 a[10] = 'f'

TypeError: 'str' object does not support item assignment

In [66]: b=a.replace('string','longer string')

In [67]: b
Out[67]: 'this is a longer string'
```

이 작업 이후 변수 a는 변경되지 않는다.

```
In [68]: a
Out[68]: 'this is a string'
```

많은 파이썬 객체는 str함수를 이용해서 문자열로 변환할 수 있다.

```
In [69]: a=5.6

In [70]: s=str(a)

In [71]: print(a)
5.6
```

문자열은 일련의 유니코드 문자이며 따라서 리스트나 튜플 같은 다른 순차적인 자료형과 같이 취급된다.

```
In [72]: s= 'python'

In [73]: list(s)
Out[73]: ['p', 'y', 't', 'h', 'o', 'n']

In [74]: s[:3]
Out[74]: 'pyt'
```

슬라이싱이라고 부르는 s[:3]문법은 대부분의 파이썬 시퀀스 자료구조에 구현되어 있다. 이 문법은 앞으로 자주 사용하게 되므로 나중에 좀 더 살펴보도록 한다.

역슬래시()는 **이스케이프 문자**로 , 개행 문자 (\n)나 유니코드 문자 같은 특수한 목적의 문자를 나타내기 위해 사용한다. 역슬래시를 나타내려면 다음처럼 역슬래시 자체를 이스케이프 한다.

```
In [77]: s = '12\\34'

In [78]: print(s)
12\34
```

특수 문자 없이 역슬래시 (\) 가 많이 포함된 문자열을 나타내려면 좀 귀찮을 수 있는데, 다행히도 문자열 앞에 r을 써서 문자열을 있는 그래도 해석하도록 할 수 있다.

```
In [81]: s = r'this\has\no\special'

In [82]: s
Out[82]: 'this\\has\\no\\special'
```

여기서 r은 raw를 뜻한다.

두 문자열을 더하면 두 문자열을 이어붙인 새로운 문자열이 생성된다.

```
In [84]: a = 'this is the first half'

In [85]: b = 'and this is the second half'

In [86]: a+b
Out[86]: 'this is the first halfand this is the second half'
```

문자열의 템플릿이나 형식을 지정하는 것은 중요한 주제다. 파이썬 3에서 이 방식이 많이 확장 되었지만 여기서는 주요 인터페이스 하나만 간단히 설명하겠다. 문자열 객체는 포멧에 따라 문자열을 대체하여 새로운 문자열을 반환하는 format 메서드를 가지고 있다.

```
In [87]: templete = '{0:.2f}{1:s} are worth US${2:d}'
```

- {0:.2f}는 첫 번째 인자를 소수점 아래 2자리까지만 표시하는 부동소수점 형태로 출력하라는 의이다.
- {1:s}는 두 번째 인자가 문자열이라는 의미이다.
- {2:d}는 세 번째 인자가 정수라는 의미이다.

인자를 이런 포맷 매개변수를 이용해 대치하려면 인자를 format매서드에 전달해야 한다.

```
In [87]: templete = '{0:.2f}{1:s} are worth US${2:d}'

In [88]: templete.format(4.5560, 'argentina',1)
Out[88]: '4.56argentina are worth US$1'
```

문자열 포캣은 광범위한 주제이며 결과 문자열에서 값을 어떻게 표시할 것인지에 대한 다양한 옵션과 메서드, 메서드, 팁이 있다.

바이트와 유니코드

파이썬 3.0부터는 아스키와 비-아스키 텍스트를 일관되게 다루기 위해 유니코드가 최상위 문자열 타입이 되었다. 파이썬 구 버전에서 문자열은 유니코드 인코딩을 명시하지 않은 바이트 였다. 문자 인코딩을 알고 있다는 가정 하에 유니코드로 변환할 수 있었다. 다음 예제를 살펴보자.

```
In [89]: val = "espanol"

In [90]: val
Out[90]: 'espanol'

In [91]: val_utf8 = val.encode('utf-8')

In [92]: val_utf8
Out[92]: b'espanol'

In [94]: type(val_utf8)
Out[94]: bytes
```

bytes 객체의 유니코드 인코딩을 알고 있다면 encode 메서드를 이용해서 다시 거꾸로 되돌릴 수 있다.

```
In [95]: val_utf8.decode('utf-8')
Out[95]: 'espanol'
```

형변환

str, bool, int,float자료형은 형변환을 위한 함수로 쓰인다.

```
In [96]: s='3.14159'

In [97]: fval = float(s)

In [98]: type(fval)
Out[98]: float

In [99]: int(fval)
Out[99]: 3

In [100]: bool(fval)
Out[100]: True

In [101]: bool(0)
Out[101]: False
```

None

None은 파이썬에서 사용하는 null 값이다. 만약 어떤 함수에서 명시적으로 값을 변환하지 않으면 묵시적으로 None을 변환한다.

날짜와 시간

파이썬 내장 datetime 모듈은 datetime, , date 그리고 time형을 지원한다. datetime형은 이름에서 알 수 있듯 date 와 time 정보를 함께 저장하며 주로 사용되는 자료형이다.

```
In [105]: from datetime import datetime, date, time

In [106]: dt = datetime(2011,10,29,20,30,21)

In [107]: dt.day
Out[107]: 29

In [108]: dt.minute
Out[108]: 30
```

datetime인스턴스에서 date 매서드와 time 매서드를 사용해서 해당 datetime의 날짜와 시간을 추출할 수 있다.

```
In [109]: dt.date()
Out[109]: datetime.date(2011, 10, 29)

In [110]: dt.time()
Out[110]: datetime.time(20, 30, 21)
```

strftime 메서드는 datetime을 문자열로 만들어준다.

```
In [113]: dt.strftime('%m/%d/%Y %H:%M:%S')
Out[113]: '10/29/2011 20:30:21'
```

strftime 함수를 이용하면 문자열을 해석하여 datetime 객체로 만들어준다.

```
In [115]: datetime.strptime('20091031','%Y%m%d')
Out[115]: datetime.datetime(2009, 10, 31, 0, 0)
```

다른 방식으로 그룹핑된 시계열 데이터를 집계할 때 datetime의 필드를 치환하는 것이 유용한 경우가 종종 발생한다. 예를 들어 분과 초 필드를 0으로 치환해서 새로운 객체를 생성할 수 있다.

```
In [118]: dt.replace(minute=0, second=0)
Out[118]: datetime.datetime(2011, 10, 29, 20, 0)
```

datetime.datetime은 변경 불가능하며, 이런 메서드들은 항상 새로운 객체를 반환한다.

두 datetime 객체의 차는 datetime.timedelta 객체를 반환한다.

```
In [118]: dt.replace(minute=0, second=0)
Out[118]: datetime.datetime(2011, 10, 29, 20, 0)

In [119]: dt2 = datetime(2011,11,15,22,30)
     ...: delta = dt2 -dt

In [120]: delta
Out[120]: datetime.timedelta(days=17, seconds=7179)

In [121]: type(delta)
Out[121]: datetime.timedelta
```

datetime.timedelta(days=17, seconds=7179) 은 17일과 7179초만큼의 시간차이를 나타낸다.

```
In [122]: dt
Out[122]: datetime.datetime(2011, 10, 29, 20, 30, 21)

In [123]: dt+delta
Out[123]: datetime.datetime(2011, 11, 15, 22, 30)
```

for문

for문 리스트나 리스트나 튜플같은 컬렉션이나 이터레이터로 순회한다. for문의 기본 문법은 다음과 같다.

for문은 continue 예약어를 사용해서 남은 블록을 건너뛰고 다음 순회로 넘어갈 수 있다. None값은 건너뛰고 리스트에 있는 모든 정수를 더하는 다음 코드를 살펴보자.

```
In [126]: for i in range (4):
     ...:     for j in range(4):
     ...:         if j>i:
     ...:             break
     ...:         print((i,j))
     ...:
     ...:
(0, 0)
(1, 0)
(1, 1)
(2, 0)
(2, 1)
(2, 2)
(3, 0)
(3, 1)
(3, 2)
(3, 3)
```

더 자세히 살펴보겠지만, 컬렉션의 원소나 이터레이터가 순차적인 자료 (예를 들면 튜플이나 리스트)라면 for문 안에서 여러 개의 변수로 꺼낼 수 있다.

```
for a,b,c in iterator:
#필요한 코드 작성
```

while문

while문은 조건을 명시하여 해당 조건이 False가 되거나 break 문을 사용해서 명시적으로 반복을 끝낼 때까지 블록 내의 코드를 수행한다.

```
x = 256
total = 0
while x>0:
    if total > 500:
        break
    total +=x
    x = x//2
```

pass

파이썬에서 pass는 아무 것도 하지 않음을 나타낸다. 이는 블록 내에서 어떤 작업도 실행하지 않을 때 사용한다(또는 아직 구현하지 않은 코드를 나중에 추가하기 위한 플레이스홀더 용도로 사용한다). pass를 사용하는 이유는 파이썬이 공백 문자로 블록을 구분하는 데 사용하기 때문이다.

```
if x<0:
    print('negative!')
elif x == 0:
    pass
else :
    print('positive!')
```

range

range함수는 연속된 정수를 넘겨주는 이터레이터를 반환한다.

```
In [1]: range(10)
Out[1]: range(0, 10)

In [2]: list(range(10))
Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

start, end, step(음수가 될 수도 있다) 값을 지정할 수 있다.

```
In [3]: list(range(0,20,2))
Out[3]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [4]: list(range(5,0,-1))
Out[4]: [5, 4, 3, 2, 1]
```

위에서 알 수 있듯이 range는 마지막 값 바로 이전 정수까지의 값을 반환한다. range 함수는 일반적으로 색인으로 시퀀스를 반복하기 위해 사용한다.

```
seq = [1,2,3,4]
for i in range(len(seq)):
    val = seq[i]
```

list 같은 함수를 이용해서 range 함수에서 생성되는 모든 정수를 다른 자료구조에 저장할 수도 있지만 보통은 기본적인 이터레이터의 용법으로 사용하게 될 것이다. 아래 코드 예제는 0부터 99,999 까지 모든 정수 중에서 3 또는 5의 배수를 모두 더한다.

```
sum = 0
for i in range(100000):
    # %는 나머지 연산자다.
    if i % 3 == 0 or i % 5 ==0:
        sum += i
```

range 함수는 임의 크기로 값을 생성해낼 수 있지만 메모리 사용량은 매우 적다.

삼항 표현식

파이썬의 **삼항 표현식**은 if - else 블록을 한 줄로 표현할 수 있도록 한다. 문법은 아래와 같다.

```
value = true-expr if condition else false-expr
```

여기서 true-expr과 false-expr은 어떤 파이썬 표현이라도 상관없다. 삼항 표현식은 아래 코드와 동일하게 작동한다.

```
if condition:
    value = true-expr
else:
    value = false-expr
```

다음 예제를 보자

```
In [5]: x =5

In [6]: 'Non-negative' if x>=0 else 'Negative'
Out[6]: 'Non-negative'
```

삼항 표현식은 if-else 블록처럼 조건이 참인 경우의 표현식만 실행된다. 따라서 삼항 표현식에서 if와 else에 비용이 많이 드는 계산이 올 수 있으며 조건이 참인 파이썬 표현만 실행된다.

코드를 줄이기 위해 항상 삼항 표현식을 쓰고 싶을 수 있겠지만 평가해야 할 조건이 복잡할 경우에는 가독성을 떨어뜨린다는 점을 기억하자.