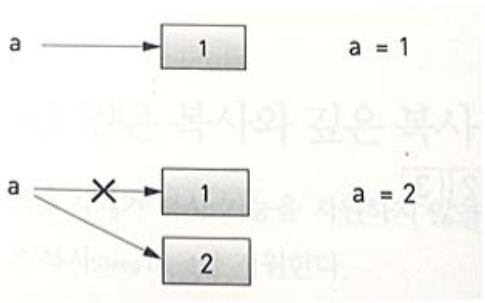
09. 객체의 복사와 형변환

파이썬 객체의 복사가 갖는 의미를 살펴보고 원하는 수준에서 객체 복사를 하는 방법을 설명한다. 각 종 데이터의 형변환을 일괄적으로 살펴본다.

9.1 객체의 복사

파이썬에서 복사는 두가지가 있다. 하나는 참조 복사이고 다른 하나는 객체 복사이다. 참조 복사란 객체는 그대로 두고 객체의 참조만 복사하는 것이다. 그림 9-1과 같이 치환문(=)을 이용하여 참조 복사를 한다. 치환문은 오른쪽 객체의 참조를 왼쪽의 심볼에 저장하라는 의미이다.

```
>>> a = 1
>>> a = 2
```

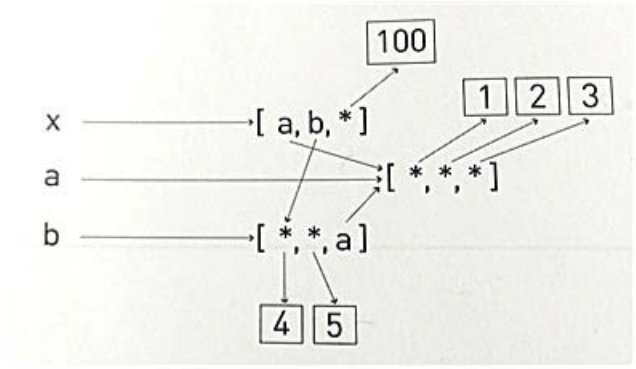


오른쪽 값이 심볼일때는 심볼의 값, 즉 객체의 참조를 왼쪽 심볼에 저장한다. 즉, 참조를 복사한다. 예를 들어, 다음 코드를 실행하면 격과는 객체1을 다른 이름 b로 한번 더 참조한다. 객체의 참조 횟수가 1만큼 증가하는 것이다.

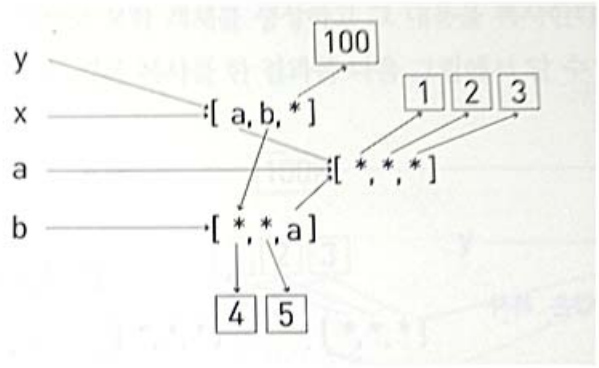
```
>>> a = 1
>>> b = a
```

좀더 복잡한 경우도 마찬가지이다. 다음과 같이 리스트를 참조 복사하는 예를 보자. 리스트는 값을 직접 저장하지 않고 객체에 대한 참조만 갖는다.

```
>>> a = [1, 2, 3]
>>> b = [4, 5, a]
>>> x = [a, b, 100]
>>> y = x                # 참조 복사
```



복사 전



복사 후

따라서 x의 변경은 곧 y의 변경과 같다.

```
>>> x
[[1, 2, 3], [4, 5, [1, 2, 3]], 100]
>>> x.append(200)
>>> y
[[1, 2, 3], [4, 5, [1, 2, 3]], 100, 200]
```

9.2 얕은 복사와 깊은 복사

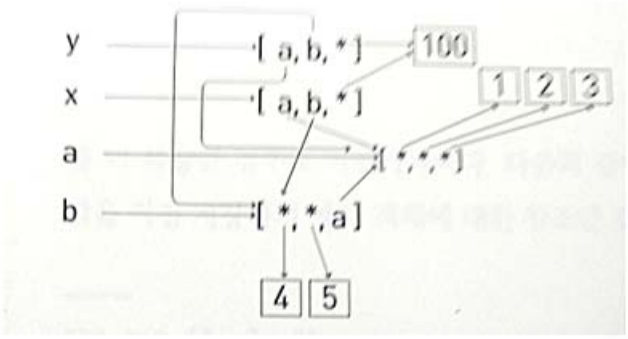
객체 자체가 복사 기능을 지원하지 않을 경우 사용하는 copy 모듈은 얕은 복사와 같은 복사를 지원한다.

● 얕은 복사 1 단계 복합 객체를 생성하고 원해 객체로부터 내용을 복사한다.
● 깊은 복사 복합 단계를 재귀적으로 생성하고 내용을 복사한다.

다음은 얕은 복사의 예이다.

```
>>> import copy
>>> a = [1, 2, 3]
>>> b = [4, 5, a]
>>> x = [a, b, 100]
>>> y = copy.copy(x)              # 얕은복사
```

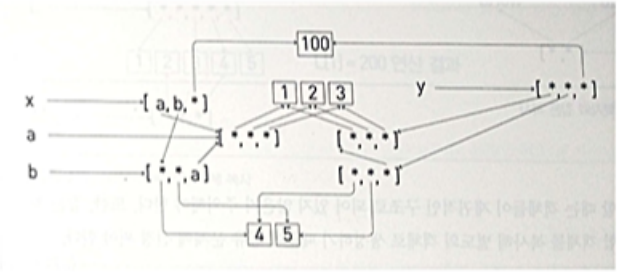
y는 리스트를 만들고 x로부터 내용을 채운다. y가 x와는 다른 객체지만 값들은 x의 내용과 동일하다.



다음은 깊은 복사의 예이다.

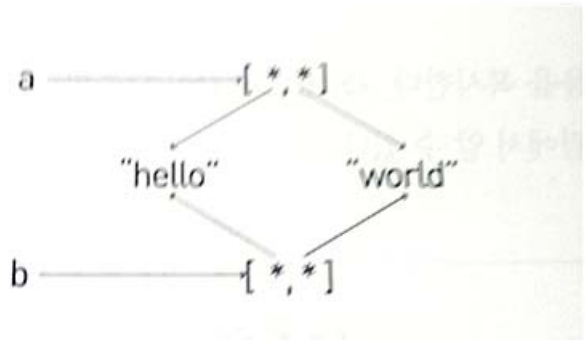
```
>>> import copy
>>> a = [1, 2, 3]
>>> b = [4, 5, a]
>>> x = [a, b, 100]
>>> y = copy.deepcopy(x)          # 깊은복사
```

y는 x로부터 재귀적으로 복합 객체를 생성하고 그 내용을 복사한다. 즉, 100과 같이 단순한 객체는 복사되지 않는다 깊은 복사를 한 결과를 다음 그림에서 알수 있다.



깊은 복사가 복합 객체만을 새로 생성하기 때문에 다음과 같이 복합 객체가 한 개만 있을 경우는 얕은 복사와 깊은 복사의 차이가 없다.

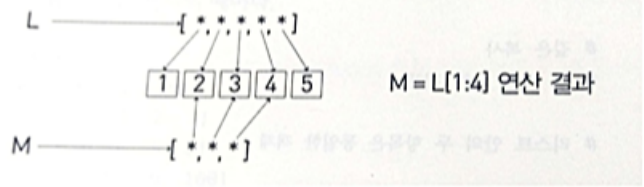
```
>>> a = ['hello','world']
>>> b = copy.copy(a)            # 얕은 복사
>>> a is b                      # 두 개는 다른 객체
False
>>> a[0] is b[0]                # 리스트 안의 두 항목은 동일한 객체
True
>>> c = copy.deepcopy(a)        # 깊은 복사
>>> a is c                      # 두 개는 다른 객체
False
>>> a[0] is c[0]                # 리스트 안의 두 항목은 동일한 객체
True
```

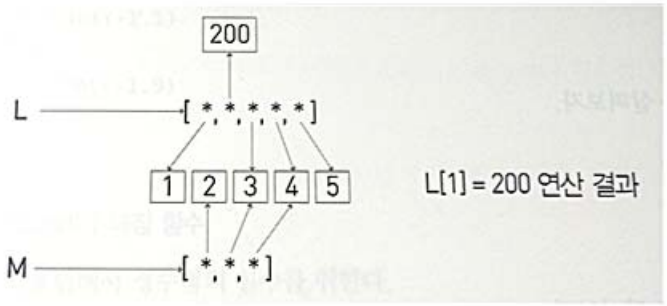


깊은 복사를 할때는 객체들이 재귀적인 구조로 되었지 않은지 주의해야 한다. 또한, 깊은 복사는 모든 복합 객체를 복사해 별도로 객체로 생성하기 때문에 공유 문제에 신경 써야 한다.

내장 자료형은 일부복사 기능을 지원한다. 리스트의 슬라이싱과 사전의 dict.copy()는 얕은 복사이다. 슬라이싱은 리스트를 새로 만들고 참조를 복사한다.

```
>>> L = [1, 2, 3, 4, 5]
>>> M = L[1:4]
>>>
>>> L[1] = 200
>>> L
[1, 200, 3, 4, 5]
>>> M                    # L의 변경이 M에 영향을 미치지 않는다.
[2, 3, 4]
```




200
L
[*, *, *, *, *]
1
2
3
4
5
L[1] = 200 연산 결과
M
[*, *, *]

9.3 형변환
파이썬의 여러 형들 간의 변환을 살펴보자.

- 수치형 변환
각종 수치형 사이의 형변환 방법을 알아보자.

1. 정수형으로의 변환
다른 자료형에서 정수형으로 형을 변환하려면 기본적으로 int() 내장 함수를 사용한다. 변환할 수 없으면 ValueError 에러가 발생한다.

```
>>> int('1234')
1234
>>> int(4.56)
4
>>> int('12.34')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '12.34'
```

실수에서 정수로 형을 변환하는 방법에는 몇가지가 있다.

◆ int() 내장 함수
소수 부분을 없애고 정수 부분만 취한다.

```
>>> int(1.1)
1
>>> int(1.9)
1
>>> int(-1.1)
-1
>>> int(-1.9)
-1
```

◆ round() 내장함수
반올림해서 정수형의 실수를 취한다.

```
>>> round(1.1)
1
>>> round(1.9)
2
>>> round(-1.1)
-1
>>> round(-1.5)
-2
```

자릿수 지정이 가능하다.

```
>>> round(1.23456, 3)   # 소수점 세 자리에서 반올림한다.
1.235
>>> round(12375, -2)    # 음수 자릿수
12400
```

◆ math 모듈의 floor() 함수

주어진 인수보다 작거나 같은 수 중에서 가장 큰 정수형의 실수를 취한다.

```
>>> import math
>>> math.floor(1.1)
1
>>> math.floor(1.9)
1
>>> math.floor(-1.0)
-1
>>> math.floor(-1.1)
-2
>>> math.floor(-1.9)
-2
```

◆ math 모듈의 ceil() 함수

주어진 실수보다 크거나 같은 수 중에서 가장 작은 정수형의 실수를 취한다.

```
>>> math.ceil(1.0)
1
>>> math.ceil(1.1)
2
>>> math.ceil(1.9)
2
>>> math.ceil(-1.0)
-1
>>> math.ceil(-1.1)
-1
>>> math.ceil(-1.9)
-1
```

2. 실수형으로의 변환

실수형으로 형을 변환할때는 float() 함수를 사용한다.

```
>>> float('1234')              # 문자열 -> float
1234.0
>>> float('12.34')             # 문자열 -> float
12.34
```

```
>>> float(1234)                # int -> float
1234.0
```

3. 복소수로의 변환

- 실수(혹은 정수)

```
>>> complex(1)          # 값이 한 개 주어지면 실수부로 처리한다.리한다.
(1+0j)
>>>
>>>
>>>
>>> complex(1)          # 값이 한 개 주어지면 실수부로 처리한다.
(1+0j)
>>> complex(1, 3)
(1+3j)
>>> complex(0, 3)
3j
>>> 3 * 1j              # 정수 값을 허수부로 변환한다.
3j
```

- 진수 변환

1. 임의의 진수를 10진수로의 변환

임의의 진수에서 10진수로 변환하는 것은 간단하다. int() 함수의 두 번째 인수에 진수를 지정하면 된다.

```
>>> int('64', 16)       # 16진수  '64' 를 10진수로 변환한다.
100
>>> int('144',8)        # 8진수  '144' 를 10진수로 변환한다.
100
>>> int('101111',2)     #  2진수  '101111' 을 10진수로 변환한다.
47
>>> int('14',5)         #  5진수  '14' 를 10진수로 변환한다.
9
```

2. 10진수를 임의의 진수로의 변환

10진수에서 8진수, 16진수로 변환하는 것은 간단함. hex()와 oct() 함수를 사용한다.

```
>>> hex(100)            # 10진수 100을 16진수 문자열로 변환한다.
'0x64'
>>> oct(100)            # 10진수 100을 8진수 문자열로 변환한다.
'0o144'
>>> bin(100)            # 10진수 100을 2진수 문자열로 변환한다.
'0b1100100'
```

문자열 서식 지정을 이용할 수도 있다.

```
>>> "{0:x}".format(100)
'64'
>>> "{0:o}".format(100)
'144'
>>> "{0:b}".format(100)
'1100100'
```

파이썬은 내장 함수로 정수에서 임의의 진수 출력을 지원하지 않으므로 코드를 직접 작성해야 한다.

```
>>> import string
>>> digs = string.digits + string.ascii_lowercase
>>> def int2base(x, base):
...     if x < 0: sign = -1
...     elif x == 0:return '0'
...     else: sign = 1
...     x *= sign
...     digits = []
...     while x:
...             digits.append(digs[x % base])
...             x //= base
...     if sign < 0:
...             digits.append('-')
...     digits.reverse()
...     return ''.join(digits) )
...
>>> print(int2base(70, 5))
240
>>> print(int2base(70, 12))
5a
>>> print(int2base(70, 16))
46
```

- 시퀀스형으로의 변환

집합 자료형들 간의 형변환에는 다양한 방법을 사용한다. 우선, 시퀀스 자료형들 간의 변환에는 자료형들의 이름에 해당하는 함수가 준비되어 있다.

● list() 함수 리스트로 변환한다.
● tuple() 함수 튜플로 변환한다.

리스트와 튜플의 예를 보자.

```
>>> t = (1, 2, 3, 4)
>>> l = [5, 6, 7, 8]
```

```
>>> s = 'abcd'
>>>
>>> list(t)
[1, 2, 3, 4]
>>> list(s)
['a', 'b', 'c', 'd']
>>> tuple(l)
(5, 6, 7, 8)
>>> tuple(s)
('a', 'b', 'c', 'd')
```

- 리스트와 튜플, 사전의 변환

1. 사전에서 리스트로의 변환

사전에서 리스트로 변환하는 것은 이미 설명했듯이 사전의 keys()와 values(), items() 메서드를 사용한다.

```
>>> d = {1:'one',2:'two',3:'three'}
>>> list(d.keys())
[1, 2, 3]
>>> list(d.values())
['one', 'two', 'three']
```

2. 리스트에서 사전으로의 변환

(key, value) 쌍으로 된 리스트가 주어져 있으면 다음과 같이 dict 생성자를 사용해서 사전으로 쉽게 변환할 수 있다.

```
>>> keys = ['a','b','c','d']
>>> values = [1, 2, 3, 4]
>>> dict(zip(keys,values))
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

- 문자열로의 변환

1. 임의의 객체를 문자열로 변환

객체를 문자열로 변환하려면 다음과 같이 두가지 방법을 사용한다.

● str() 함수 문자열로 변환한다.
● repr() 함수 문자열로 변환한다.

str()함수는 비형식적인 문자열 변환이다. 즉, 보기좋게 출력될 문자열을 만든다고 생각하면 된다. repr() 함수는 좀더 형식적인 문자열 변환이다. 객체의 자료형을 정확하게 문자열로 표현하며, 많은 경우 eval() 함수로 역 표현을 할수 있다.

```
>>> print(str([1, 2, 3]), str((4, 5, 6 )), str('abc'))
[1, 2, 3] (4, 5, 6) abc
>>> print(repr([1, 2, 3]), repr((4, 5, 6)),repr('abc'))
[1, 2, 3] (4, 5, 6) 'abc'
```

출력된 문자열에서 다시 객체를 생성하려면 eval() 함수를 사용한다.

```
>>> eval('[5, 6, 7, 8]')        # 문자열 '[5, 6, 7, 8]'로부터 리스트를 만든다.
[5, 6, 7, 8]
```

eval() 함수는 문자열로 표현된 식을 실행하는 내장 함수이다.

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

일반적으로 어떤 객체를 문자열로 변환한 후, 다시 이 문자열을 원래의 객체로 변환하려면 repr()과 eval() 함수를 사용한다.

```
>>> a = {1:"one",2:"two"}
>>> b = repr(a)                 # a를 형식적인 문자열로 변환한다.
>>> b
"{1: 'one', 2: 'two'}"
>>> c = eval(b)                 # b를 실행하여(즉, 사전이 만들어진다.) c에 치환한다.
>>> print(c)
{1: 'one', 2: 'two'}
```

2. 유니코드와 문자 간 변환

유니코드 값과 문자의 변환은 다음과 같다.

```
>>> chr(97)             # 유니코드 -> 문자
'a'
>>> chr(44032)          # 문자 -> 유니코드
'가'
>>> ord('가')
44032
```

3. 문자열과 바이트의 변환

문자열에서 바이트로의 변환은 encode() 메서드를, 바이트에서 문자열로의 변환은 decode() 메서드를 사용한다.

```
>>> b = b'bytes'                # 바이트
>>> type(b)
<class 'bytes'>
>>> b.decode('utf-8')           # 바이트에서 문자열로 변환한다.
'bytes'
>>> s = 'string'
>>> s.encode('utf-8')           # 문자열에서 바이트로 변환한다.
b'string'
```

4. 이진 바이트 열과 16진 바이트 열 변환

binascii 모듈의 hexlify() 함수를 사용하면 이진 바이트 열을 16진 바이트 열로 변환할수 있다.

```
>>> hex(ord('a'))               # 코드 값을 확인한다.
'0x61'
>>>
>>> import binascii
>>> binascii.hexlify(b'abc')    # 바이트 열
b'616263'
>>>
>>> buf = bytearray(b'abcde')   # 바이트 배열
>>> binascii.hexlify(buf)
b'6162636465'
```

binascii 모듈의 unhexlify() 함수를 사용하면 16진 바이트 열을 이진 바이트열로 변환할수 있다.

```
>>> binascii.unhexlify(b'6162636465')
b'abcde'
```

5. 정수를 쉼표가 있는 문자열로의 변환

format() 함수를 사용하여 변환할수 있다.

format() 메서드를 사용하여 문자열에 서식을 지정할수도 있다.

```
>>> "{:,},{:,}".format(10030405, 12345)        # 인수 위치를 지정하지 않은 경우
'10,030,405,12,345'
>>>
>>> "{0:,}{1:,}".format(10030405, 12345)       # 인수 위치를 지정한 경우은 경우
'10,030,40512,345'
```

locale 모듈은 나라마다 문화적으로 표현이 다른 것들을 처리하게 도와준다.

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, "")        # 사용자 기본환경(국가나 언어)으로 설정
```

```
'Korean_Korea.949'
>>> print(locale.format("%d",10030405, grouping = True))
10,030,405
```

9.4 파이썬 자료형의 이진 변환

고수준의 파이썬 자료형을 C언어 등에서 사용할수 있는 이진 바이트 열로 변환하거나 반대로 이진 바이트 열을 파이썬 고수준 자료형으로 변환할 때 struct 모듈을 사용한다. 파이썬 자료형을 이진 바이트 열로 변환하기 위해 pack() 과 pack_into() 함수를 사용하고, 이진 바이트 열을 파이썬 자료형으로 변환하기 위해 unpack()과 unpack_from() 함수를 사용한다.

- pack(fmt, v1, v2, ...)
  서식 문자열인 인수 fmt에 따라서 v1, v2, ...를 바이트 열로 변환한다.
- pack_into(fmt, buffe, offset, v1, v2, ...)
  pack() 함수와 같으나 결과를 인수 buffer 의 인수 offset 위치부터 기록한다.
- unpack(fmt, buffer)
  인수 buffer의 바이트 열로 지정된 서식 문자열인 인수 fmt에 따라서 파이썬 객체로 변환한다. 변환 결과는 튜플로 반환한다.
- unpack_from(fmt, buffer, offset=0)
  인수 buffer의 인수 offset 위치부터 서식 문자열인 인수 fmt에 따라서 파이썬 객체로 변환한다. 변환 결과를 튜플로 반환한다.
- calcsize(fmt)
  서식 문자열인 인수 fmt으로 패킹했을대의 데이터 크기를 반환한다.

서식 문자열에 사용하는 문자는 다음과 같다.

표) struct 모듈의 서식 문자열

서식	C 형식	파이썬 형식	표준 크기
x	pad byte	값없음	
c	char	길이가 1인 문자열	1
b	signed char	int	1
B	unsigned char	int	1
?	_Bool	bool	1
h	short	int	2
H	unsigned short	int	2
i	int	int	4
I	unsigned int	int	4
l	long	int	4
L	unsigned long	int	4
q	long long	int	8
Q	unsigned long long	int	8
n	ssize_t	int	
N	size_t	int	
f	float	float	4
d	double	float	8
s	char[]	bytes	
p	char[]	bytes	
p	void *	int	

- 파이썬 데이터를 바이트 열로 변환

파이썬의 데이터를 이진 바이트 열로 변환하는 예를 보자. 우선 short, short, long 세 개의 정수를 변환해 보자.

```
>>> from struct import *
>>> pack('hhl', 1, 2, 3)
b'₩x01₩x00₩x02₩x00₩x03₩x00₩x00₩x00'
>>> pack('2h l', 1, 2, 3)       #  'hhl' 과 같은 서식
b'₩x01₩x00₩x02₩x00₩x03₩x00₩x00₩x00'
```

short int는 2바이트, long은 4바이트이다. 시스템은 리틀 엔디안을 따른다. 서식 문자 앞의 숫자(2)는 반복을 의미한다. 각 서식문자는 공백으로 떨어져 있어도 무방하다.

실수와 문자열을 변환한 예는 다음과 같다.

```
>>> pack('d 6s', 3.14, b'python')       #①
b'₩x1f₩x85₩xebQ₩xb8₩x1e₩t@python'
>>> pack('d 7s', 3.14, b'python')       # ②
b'₩x1f₩x85₩xebQ₩xb8₩x1e₩t@python₩x00'
```

① 문자열은 바이트 열로의 변환이 필요하고 길이를 서식 문자 s앞에 적어 주어야 한다.
② 문자열 마지막에 널 코드를 추가하고 싶으면 길이를 하나 증가하면 된다.

pack_into() 함수는 버퍼로 결과를 출력한다.

```
>>> buf = bytearray(8)          # 8 바이트의 버퍼를 준비한다.
>>> offset = 0
>>> pack_into('hhl',buf,offset, 1, 2, 3)        # 1, 2, 3을 출력한다.
>>> buf
bytearray(b'₩x01₩x00₩x02₩x00₩x03₩x00₩x00₩x00')
>>>
>>> bytes(buf)
b'₩x01₩x00₩x02₩x00₩x03₩x00₩x00₩x00'
>>>
>>> import binascii
>>> binascii.hexlify(buf)
b'0100020003000000'
```

pack() 함수가 반환할 바이트 열의 크기를 알고 싶으면 calcsize() 함수를 사용한다.

```
>>> calcsize('d 8s')
16
```

- C 바이트 열을 파이썬 데이터로의 변환

앞에서 변환한 데이터를 파이썬 데이터로 역변환해 보자. 역변환에는 unpack() 함수를 사용한다.

```
>>> from struct import *
>>> s = pack('hhl', 1, 2, 3) )
>>> unpack('hhl',s)
(1, 2, 3)
>>>
>>> s = pack('d 6s', 3.14, b'python')
>>> unpack('d 6s', s) )
(3.14, b'python')
```

결괏값은 언제나 튜플로 전달된다. unpack_from() 함수를 사용하면 버퍼에서 파이썬 객체로 변환할수 있다.

```
>>> buf = bytearray(20)
>>> pack_into('hhl',buf, 5, 1, 2, 3)          # 오프셋 5부터 저장한다.
>>> unpack_from('hhl',buf, 5)                 # 오프셋 5부터 읽는다.
(1, 2, 3)
>>> unpack_from('hhl',buf)                    # 기본값 오프셋은 0이다. 해독 에러 발생
(0, 0, 33554688)
```

- 바이트 저장 순서

파이썬 자료형을 바이트 열로 변환할 때 바이트 열의 순서를 지정할 필요가 있다. 대표적으로는 리틀 엔디안과 빅 엔디안이 있다. 이때 사용하는 제어 문자는 표 9-2에 있다.

표. 바이트 순서 제어 문자

제어 문자	바이트 순서	크기와 정렬
@	시스템을 따름*	시스템을 따름(기본값)**
=	시스템을 따름*	표준
<	리틀 엔디안	표준
>	빅 엔디안	표준
!	네트워크(빅 엔디안)	표준

*바이트 순서 필드에서 '시스템을 따름' 은 CPU에 따라 적용되는 엔디안 방식을 적용받는다.
**크기와 정렬 필드에서 '시스템을 따름' 은 시스템이 사용하는 방식을 적용받는다는 의미이고, '표준' 은 정해진 크기나 방법을 적용받는다는 의미이다.

바이트 순서를 지정하는 제어 문자를 사용하지 않으면 @가 적용된다. 사용하는 예를 보면 다음과 같다.

```
>>> pack('<hi', 1, 2)          #  리틀 엔디안
b'₩x01₩x00₩x02₩x00₩x00₩x00'
>>>
>>> pack('>hi', 1, 2)          # 빅 엔디안
b'₩x00₩x01₩x00₩x00₩x00₩x02'
>>>
>>> pack('!hi', 1, 2)          # 네트워크 = 빅 엔디안
b'₩x00₩x01₩x00₩x00₩x00₩x02'
```

- 정렬 문제

다음과 같은 예를 보자.

```
>>> calcsize('h')
2
```

원래는 6이어야 맞을 것 같은데 8이 나왔다. 이것은 정렬의 문제이다.

```
>>> pack('=hil', 1, 2, 3)
b'₩x01₩x00₩x02₩x00₩x00₩x00₩x03₩x00₩x00₩x00'
>>> pack('@hil', 1, 2, 3)
b'₩x01₩x00₩x00₩x00₩x02₩x00₩x00₩x00₩x03₩x00₩x00₩x00'
```

같은 서식 문자열(hil)이지만, 첫 문자에 따라 길이가 달라졌다. 제어 문자가 @인 경우는 자동정렬이 적용되는데, 규칙은 다음과 같다. 앞 형식의 길이를 a, 뒤 형식의 길이를 b라고 하자. 이때 앞 형식의 길이는 뒤 형식의 길이에 다음과 같은 수식에 의해 맞추어진다.

```
new_a = (a + b - 1) / b * b
```

@hi인 경우 h의 길이는 2, i 의 길이는 4이므로 다음과 같이 계산된다.

```
>>> a = 2
>>> b = 4
>>> (a + b - 1) / b * b
5.0
```

즉, a의 길이가 2이지만 4단위로 정렬되므로 2바이트의 패딩 바이트가 추가된다. 결국 다음과 같은 결과가 나온다.

```
>>> pack('@hi', 1, 2)
b'₩x01₩x00₩x00₩x00₩x02₩x00₩x00₩x00'
```

하지만, 다른 형식들은 이러한 정렬을 적용받지 않는다.

```
>>> pack('=hi', 1, 2)
b'₩x01₩x00₩x02₩x00₩x00₩x00'
```

리틀 엔디안과 빅 엔디안도 자동 정렬 기능은 없다.

```
>>> pack('<hi', 1, 2)   # 리틀 엔디안
b'₩x01₩x00₩x02₩x00₩x00₩x00'
>>> pack('>hi', 1, 2)   # 빅 엔디안
b'₩x00₩x01₩x00₩x00₩x00₩x02'
```