

Table of Contents

- Numeric Data Type
- String Data Type
- List Data Type
- More on Assignments
- More on Boolean Data Types

Numeric Data Types [1/3]

항목	사용 예
<u>정수</u>	123, -345, 0
<u>실수</u>	123.45, -1234.5, 3.4e10
<u>8진수</u>	0o34, 0o25
<u>16진수</u>	0x2A, 0xFF

정수형

정수형(Integer)이란 말 그대로 정수를 뜻하는 자료형을 말한다.

```
>>> a = 123
>>> a = -178
>>> a = 0
```

실수형

파이썬에서 실수형(Floating-point)은 소수점이 포함된 숫자를 말한다.

```
>>> a = 1.2
>>> a = -3.45
```

```
>>> a = 4.24E10
>>> a = 4.24e-10
```

위의 방식은 "컴퓨터식 지수 표현 방식"으로 파이썬에서는 4.24e10 또는 4.24E10처럼 표현한다(e와 E 둘 중 어느 것을 사용해도 무방하다). 여기서 4.24E10은 $4.24 * 10^{10}$, 4.24e-10은 $4.24 * 10^{-10}$ 을 의미한다.

Numeric Data Types [2/3]

8진수와 16진수

8진수(Octal)를 만들기 위해서는 숫자가 0o 또는 0O(숫자 0 + 알파벳 소문자 o 또는 대문자 O)로 시작하면 된다.

```
>>> a = 0o177
```

16진수(Hexadecimal)를 만들기 위해서는 0x로 시작하면 된다.

```
>>> a = 0x8ff  
>>> b = 0xABC
```

8진수나 16진수는 파이썬에서 잘 사용하지 않는 형태의 숫자 자료형이니 간단히 눈으로 익히고 넘어가자.

Numeric Data Types [3/3]

x의 y제곱을 나타내는 ** 연산자

다음으로 알아야 할 연산자로 ****** 라는 연산자가 있다. 이 연산자는 `x ** y` 처럼 사용되었을 때 x의 y제곱(xy) 값을 리턴한다. 다음의 예를 통해 알아보자.

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

나눗셈 후 나머지를 반환하는 % 연산자

프로그래밍을 처음 접하는 독자라면 **%** 연산자는 본 적이 없을 것이다. **%** 는 나눗셈의 나머지 값을 반환하는 연산자이다. 7을 3으로 나누면 나머지는 1이 될 것이고 3을 7로 나누면 나머지는 3이 될 것이다. 다음의 예로 확인해 보자.

```
>>> 7 % 3
1
>>> 3 % 7
3
```

나눗셈 후 소수점 아랫자리를 버리는 // 연산자

/ 연산자를 사용하여 7 나누기 4를 하면 그 결과는 예상대로 1.75가 된다.

```
>>> 7 / 4
1.75
```

이번에는 나눗셈 후 소수점 아랫자리를 버리는 **//** 연산자를 사용한 경우를 보자.

```
>>> 7 // 4
1
```

Table of Contents

- Numeric Data Type
- String Data Type
- List Data Type
- More on Assignments
- More on Boolean Data Types

String Data Types [1/13]

문자열은 어떻게 만들고 사용할까?

1. 큰따옴표로 양쪽 둘러싸기

```
"Hello World"
```

2. 작은따옴표로 양쪽 둘러싸기

```
'Python is fun'
```

3. 큰따옴표 3개를 연속으로 써서 양쪽 둘러싸기

```
"""Life is too short, You need python"""
```

4. 작은따옴표 3개를 연속으로 써서 양쪽 둘러싸기

```
'''Life is too short, You need python'''
```

```
>>> food = "Python's favorite food is perl"
```

```
>>> say = '"Python is very easy." he says.'
```

```
>>> food = 'Python\'s favorite food is perl'  
>>> say = "Python is very easy." he says."
```

String Data Types [2/13]

```
Life is too short  
You need python
```

1) 줄을 바꾸기 위한 이스케이프 코드 `\n` 삽입하기

```
>>> multiline = "Life is too short\nYou need python"
```

위의 예제에서는 줄이 길어지는 단점

2) 연속된 작은따옴표 3개(`'''`) 또는 큰따옴표 3개(`"""`) 이용

```
>>> multiline='''  
... Life is too short  
... You need python  
... '''
```

작은따옴표 3개를 사용한 경우

```
>>> multiline="""  
... Life is too short  
... You need python  
... """
```

큰따옴표 3개를 사용한 경우

String Data Types [3/13]

[이스케이프 코드란?]

문자열 예제에서 여러 줄의 문장을 처리할 때 백슬래시 문자와 소문자 n을 조합한 `\n` 이스케이프 코드를 사용했다. 이스케이프 코드란 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 "문자 조합"이다. 주로 출력물을 보기 좋게 정렬하는 용도로 이용된다. 몇 가지 이스케이프 코드를 정리하면 다음과 같다.

코드	설명
<code>\n</code>	개행 (줄바꿈)
<code>\t</code>	수평 탭
<code>\\</code>	문자 " <code>\</code> "
<code>\'</code>	단일 인용부호(<code>'</code>)
<code>\"</code>	이중 인용부호(<code>"</code>)

<code>\r</code>	캐리지 리턴
<code>\f</code>	폼 피드
<code>\a</code>	벨 소리
<code>\b</code>	백 스페이스
<code>\000</code>	널문자

String Data Types [4/13]

1) 문자열 더해서 연결하기(Concatenation)

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

2) 문자열 곱하기

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

3) 문자열 곱하기 응용

문자열 곱하기를 좀 더 응용해 보자.

```
# multistring.py

print("=" * 50)
print("My Program")
print("=" * 50)
```

결과값은 다음과 같이 나타날 것이다.

```
C:\Users>cd C:\Python
C:\Python>python multistring.py
=====
My Program
=====
```

String Data Types [5/13]

```
>>> a = "Life is too short, You need Python"
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
```

```
>>> a[0:2]
'Li'
>>> a[5:7]
'is'
>>> a[12:17]
'short'
```

```
>>> a = "Life is too short, You need Python"
>>> b = a[0] + a[1] + a[2] + a[3]
>>> b
'Life'
```

```
>>> a[19:]
'You need Python'
```

String Data Types [6/13]

["Pithon"이라는 문자열을 "Python"으로 바꾸려면?]

```
>>> a = "20010331Rainy"
>>> year = a[:4]
>>> day = a[4:8]
>>> weather = a[8:]

>>> year
'2001'
>>> day
'0331'
>>> weather
'Rainy'
```

```
>>> a = "Pithon"
>>> a[1]
'i'
>>> a[1] = 'y'
```

NO

```
>>> a = "Pithon"
>>> a[:1]
'p'
>>> a[2:]
'thon'
>>> a[:1] + 'y' + a[2:]
'Python'
```

String Data Types [7/13]

```
>>> "I eat %d apples." % 3  
'I eat 3 apples.'
```

```
>>> "I eat %s apples." % "five"  
'I eat five apples.'
```

```
>>> number = 3  
>>> "I eat %d apples." % number  
'I eat 3 apples.'
```

```
>>> number = 10  
>>> day = "three"  
>>> "I ate %d apples. so I was sick for %s days." % (number, day)  
'I ate 10 apples. so I was sick for three days.'
```

String Data Types [8/13]

문자열 포맷 코드

코드	설명
%s	문자열 (String)
%c	문자 1개(character)
%d	정수 (Integer)
%f	부동소수 (floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 % 자체)

```
>>> "I have %s apples" % 3
'I have 3 apples'
>>> "rate is %s" % 3.234
'rate is 3.234'
```

[포매팅 연산자 %d와 %를 같이 쓸 때는 %%를 쓴다]

```
>>> "Error is %d%." % 98
```

위 예문의 결과값으로 당연히 "Error is 98%."가 출력될 것
르지 않다는 값 오류(Value Error) 메시지를 보여 준다.

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: incomplete format
```

String Data Types [9/13]

1) 정렬과 공백

```
>>> "%10s" % "hi"  
'          hi'
```

```
>>> "%-10sjane." % 'hi'  
'hi          jane.'
```

2) 소수점 표현하기

```
>>> "%0.4f" % 3.42134234  
'3.4213'
```

```
>>> "%10.4f" % 3.42134234  
'          3.4213'
```

String Data Types [10/13]

문자 개수 세기(count)

```
>>> a = "hobby"
>>> a.count('b')
2
```

소문자를 대문자로 바꾸기(upper)

```
>>> a = "hi"
>>> a.upper()
'HI'
```

위치 알려주기1(find)

```
>>> a = "Python is best choice"
>>> a.find('b')
10
>>> a.find('k')
-1
```

대문자를 소문자로 바꾸기(lower)

```
>>> a = "HI"
>>> a.lower()
'hi'
```

String Data Types [11/13]

왼쪽 공백 지우기(lstrip)

```
>>> a = " hi "  
>>> a.lstrip()  
'hi '
```

오른쪽 공백 지우기(rstrip)

```
>>> a= " hi "  
>>> a.rstrip()  
' hi '
```

양쪽 공백 지우기(strip)

```
>>> a = " hi "  
>>> a.strip()  
'hi'
```


String Data Types [12/13]

위치 알려주기2(index)

```
>>> a = "Life is too short"
>>> a.index('t')
8
>>> a.index('k')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: substring not found
```

문자열 삽입(join)

```
>>> a = ","
>>> a.join('abcd')
'a,b,c,d'
```

문자열 나누기(split)

```
>>> a = "Life is too short"
>>> a.split()
['Life', 'is', 'too', 'short']
>>> a = "a:b:c:d"
>>> a.split(':')
['a', 'b', 'c', 'd']
```

문자열 바꾸기(replace)

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'
```

숫자 바로 대입하기

```
>>> "I eat {0} apples".format(3)
'I eat 3 apples'
```

숫자 값을 가진 변수로 대입하기

```
>>> number = 3
>>> "I eat {0} apples".format(number)
'I eat 3 apples'
```

문자열 바로 대입하기

```
>>> "I eat {0} apples".format("five")
'I eat five apples'
```

2개 이상의 값 넣기

```
>>> number = 10
>>> day = "three"
>>> "I ate {0} apples. so I was sick for {1} days.".format(number, day)
'I ate 10 apples. so I was sick for three days.'
```

Character to Numeric Conversion

- `ord()` : Python built-in function which returns the numeric (ordinal) code of a single character
- `chr()` : Python built-in function which converts a numeric code to the corresponding character

```
>>> ord("A")    # argument character의 ascii code에서의 위치
65
>>> ord("a")
97
>>> chr(97)      # 0 ~~ 255를 받아서 ascii code를 return
'a'
>>> chr(65)
'A'
```

UTF-8 code (가변형 4 bytes)을 사용했다 하더라도 처음 1byte는 ASCII code와 동일함

Representations of Characters

[1/2]

Character Encoding

ASCII	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0000	N _U	S _H	S _X	E _X	E _T	E _O	A _K	B _L	B _S	H _T	L _F	Y _T	F _F	C _R	S _O	S _I
0001	P _L	P ₁	P ₂	P ₃	P ₄	N _K	S _Y	E _Σ	C _N	E _M	S _B	E _C	F _S	G _S	R _S	U _S
0010		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	p	q	r	s	t	u	v	w	x	y	z	{		}	~	␣
1000	s ₀	s ₁	s ₂	s ₃	I _N	N _L	S _S	E _S	H _S	H _J	Y _S	P _D	P _V	R _I	S ₂	S ₃
1001	P _C	P ₁	P ₂	S _E	C _C	M _M	S _P	E _P	O _S	O _D	O _A	C _S	S _T	O _S	P _M	A _P
1010	A _O	i	ç	£	¤	¥		\$	¨	©	ª	«	¬	-	®	—
1011	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
1100	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
1101	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
1110	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
1111	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Figure 7.3 ASCII, the American Standard Code for Information Interchange.

Note: The original 7-bit ASCII is the top half of the table; the whole table is known as Extended ASCII (ISO-8859-1). The 8-bit symbol for a letter is the four row bits followed by the four column bits (e.g., A = 0100 0001, while z = 0111 1010). Characters shown as two small letters are control symbols used to encode nonprintable information (e.g., **S**_S = 0000 1000 is backspace). The bottom half of the table represents characters needed by Western European languages, such as Icelandic's eth (ð) and thorn (þ).

لماذا لا يتكلمون اللغة العربية فحسب؟
 Защо те просто не могат да говорят български?
 Per què no poden simplement parlar en català?
 他們為什麼不說中文（台灣）？
 Proč prostě nemluví česky?
 Hvorfor kan de ikke bare tale dansk?
 Warum sprechen sie nicht einfach Deutsch?
 Μα γιατί δεν μπορούν να μιλήσουν Ελληνικά;
Why can't they just speak English?
 ¿Por qué no pueden simplemente hablar en castellano?
 Miksi he eivät yksinkertaisesti puhu suomea?
 Pourquoi, tout simplement, ne parlent-ils pas français?
 למה הם פשוט לא מדברים עברית?
 Miért nem beszélnek egyszerűen magyarul?
 Af hverju geta þeir ekki bara talað íslensku?
 Perché non possono semplicemente parlare italiano?
 なぜ、みんな日本語を話してくれないのか？
 의 모든 사람들이 한국어를 이해한다면 얼마나
 Waarom spreken ze niet gewoon Nederlands?
 Hvorfor kan de ikke bare snakke norsk?
 Dlaczego oni po prostu nie mówią po polsku?
 Porque é que eles não falam em Português (do Brasil)?
 Oare ăştia de ce nu vorbesc româneşte?
 Почему же они не говорят по-русски?
 Zašto jednostavno ne govore hrvatski?
 Pse nuk duan të flasin vetëm shqip?
 Varför pratar dom inte bara svenska?
 ทำไมเขาถึงไม่พูดภาษาไทย
 Neden Türkçe konuşmıyorlar?

Figure 7.4 “Why can’t they just speak _____?” A Web page, www.trigeminal.com/samples/pro displaying that question expressed in more than 100 languages. Can you name all of them in this partial list?

Representations of Characters [2/2]

- **ASCII: 7 bits for 128 characters (1960년)**
- **E-ASCII: 8 bits for 256 characters (1963년)**
- **유니코드(Unicode)**
 - 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현되도록 설계된 산업 표준
 - 유니코드 협회(Unicode Consortium)
- **UTF (Unicode Transformation Format) -8 : 32bits (1992년)**
 - A character encoding for all possible characters in **Unicode**
 - **variable-length** (→ uses 1 to 4 units of 8-bit code)
 - **UTF-8의 unit 1 → E-ASCII 문자**
 - the dominant character encoding for the **World Wide Web**, accounting for **84.6% of all Web pages**

Table of Contents

- Numeric Data Type
- String Data Type
- List Data Type
- More on Assignments
- More on Boolean Data Types

List Data Types [1/6]

```
>>> a = [ ]  
>>> b = [1, 2, 3]  
>>> c = ['Life', 'is', 'too', 'short']  
>>> d = [1, 2, 'Life', 'is']  
>>> e = [1, 2, ['Life', 'is']]
```

```
>>> a = [1, 2, ['a', 'b', ['Life', 'is']]]
```

```
>>> a[2][2][0]  
'Life'
```

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
```

다음의 예를 따라 해보자.

```
>>> a[0]  
1  
>>> a[-1]  
['a', 'b', 'c']  
>>> a[3]  
['a', 'b', 'c']
```

List Data Types [2/6]

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]
>>> c = a[2:]
>>> b
[1, 2]
>>> c
[3, 4, 5]
```

```
>>> a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
>>> a[2:5]
[3, ['a', 'b', 'c'], 4]
>>> a[3][:2]
['a', 'b']
```

1) 리스트 더하기(+)

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

2) 리스트 반복하기(`*`)

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```


List Data Types [3/6]

1. 리스트에서 하나의 값 수정하기

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

2. 리스트에서 연속된 범위의 값 수정하기

```
>>> a[1:2]
[2]
>>> a[1:2] = ['a', 'b', 'c']
>>> a
[1, 'a', 'b', 'c', 4]
```

3. [] 사용해 리스트 요소 삭제하기

```
>>> a[1:3] = [ ]
>>> a
[1, 'c', 4]
```

4. del 함수 사용해 리스트 요소 삭제하기

```
>>> a
[1, 'c', 4]
>>> del a[1]
>>> a
[1, 4]
```

List Data Types [4/6]

리스트에 요소 추가(append)

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

```
>>> a.append([5,6])
>>> a
[1, 2, 3, 4, [5, 6]]
```

리스트 정렬(sort)

sort 함수는 리스트의 요소를 순서대로 정렬해 준다.

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

문자 역시 알파벳 순서로 정렬할 수 있다.

```
>>> a = ['a', 'c', 'b']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

List Data Types [5/6]

리스트 뒤집기(reverse)

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

위치 반환(index)

index(x) 함수는 리스트에 x라는 값이 있으면 x의 위치값을 리턴한다.

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

리스트에 요소 삽입(insert)

insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수이다.

```
>>> a = [1, 2, 3]
>>> a.insert(0, 4)
[4, 1, 2, 3]
```

리스트 요소 제거(remove)

remove(x)는 리스트에서 첫 번째로 나오는 x를 삭제하는 함수이다.

```
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
[1, 2, 1, 2, 3]
```

List Data Types [6/6]

리스트 요소 끄집어내기(pop)

pop()은 리스트의 맨 마지막 요소를 돌려 주고 그 요소는 삭제하는 함수

```
>>> a = [1,2,3]
>>> a.pop()
3
>>> a
[1, 2]
```

pop(x)는 리스트의 x번째 요소를 돌려 주고 그 요소는 삭제

```
>>> a = [1,2,3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

리스트에 포함된 요소 x의 개수 세기(count)

count(x)는 리스트 내에 x가 몇 개 있는지 조사하여 그 개수를 돌려주는 함수

```
>>> a = [1,2,3,1]
>>> a.count(1)
2
```

리스트 확장(extend)

extend(x)에서 x에는 리스트만 올 수 있으며 원래의 a 리스트에 x 리스트를 더하게 된다.

```
>>> a = [1,2,3]
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
>>> b = [6, 7]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

Table of Contents

- Numeric Data Type
- String Data Type
- List Data Type
- More on Assignments
- More on Boolean Data Types

변수를 만드는 여러 가지 방법

Remember? `>>> a, b = 'python', 'life'`

```
>>> a, b = ('python', 'life')
```

위의 예문처럼 튜플로 a, b에 값을 대입할 수 있다. 이 방법은 다음 예문과 완전히 동일하다.

```
>>> (a, b) = 'python', 'life'
```

튜플 부분에서도 언급했지만 튜플은 괄호를 생략해도 된다.

아래처럼 리스트로 변수를 만들 수도 있다.

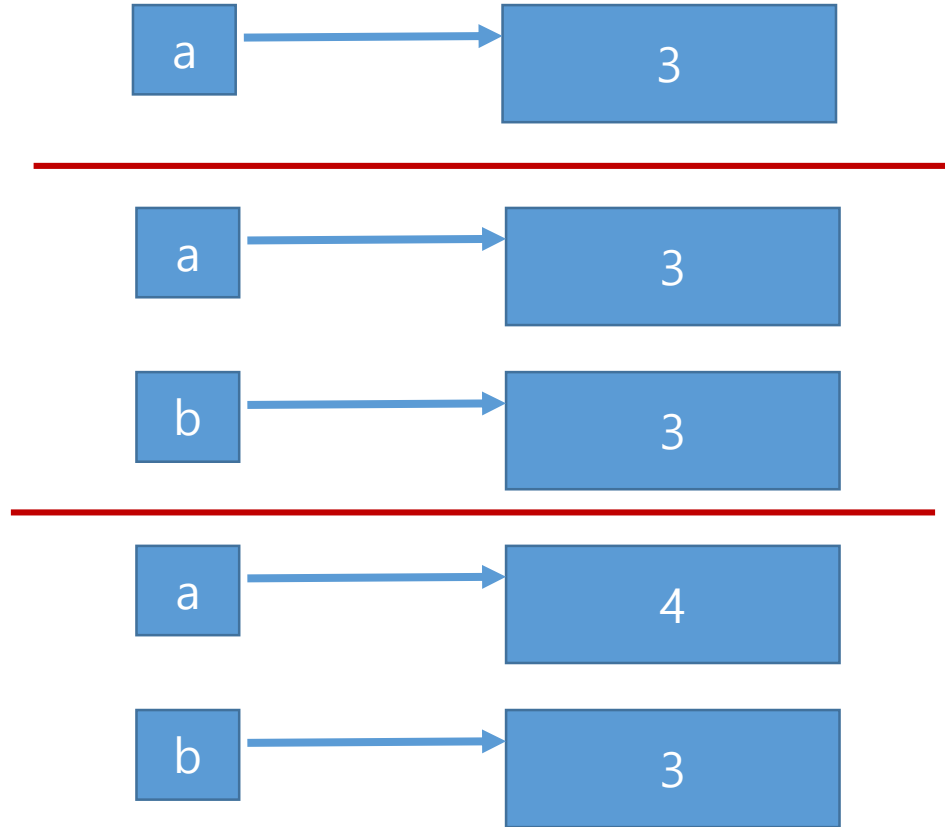
```
>>> [a,b] = ['python', 'life']
```

또한 여러 개의 변수에 같은 값을 대입할 수도 있다.

```
>>> a = b = 'python'
```

Aliasing vs Shallow Copy [1/3]

```
>>> a = 3  
>>> b = a  
>>> a = 4  
>>> b  
3
```

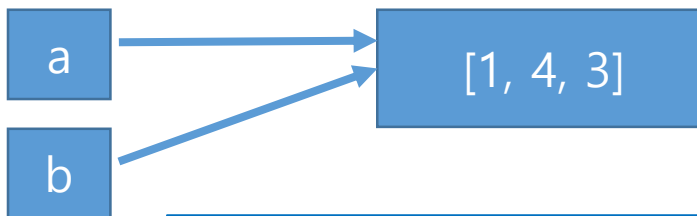
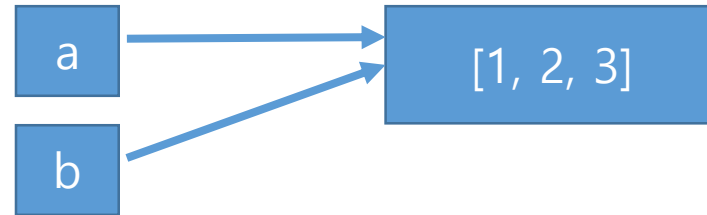
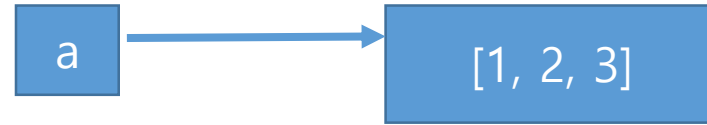


Int, Float, Char, String, Boolean 같은 Basic Data Type의 값들은 `x = y` 형태의 assignment에서 물리적인 복제 (deep copy)를 한다

Aliasing and Shallow Copy [2/3]

Basic Data Type 보다 Advanced된 Data Type들 (list, set, dictionary, tuple 등등) 값들은 $x = y$ 형태의 assignment에서 논리적인 복제 (aliasing)를 한다

```
>>> a = [1, 2, 3]
>>> b = a
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 4, 3]
```



두 변수 x, y 가 같은 값을 가지는지는 `==`로 확인
같은 객체를 가르키고 있는지는 `is`로 확인

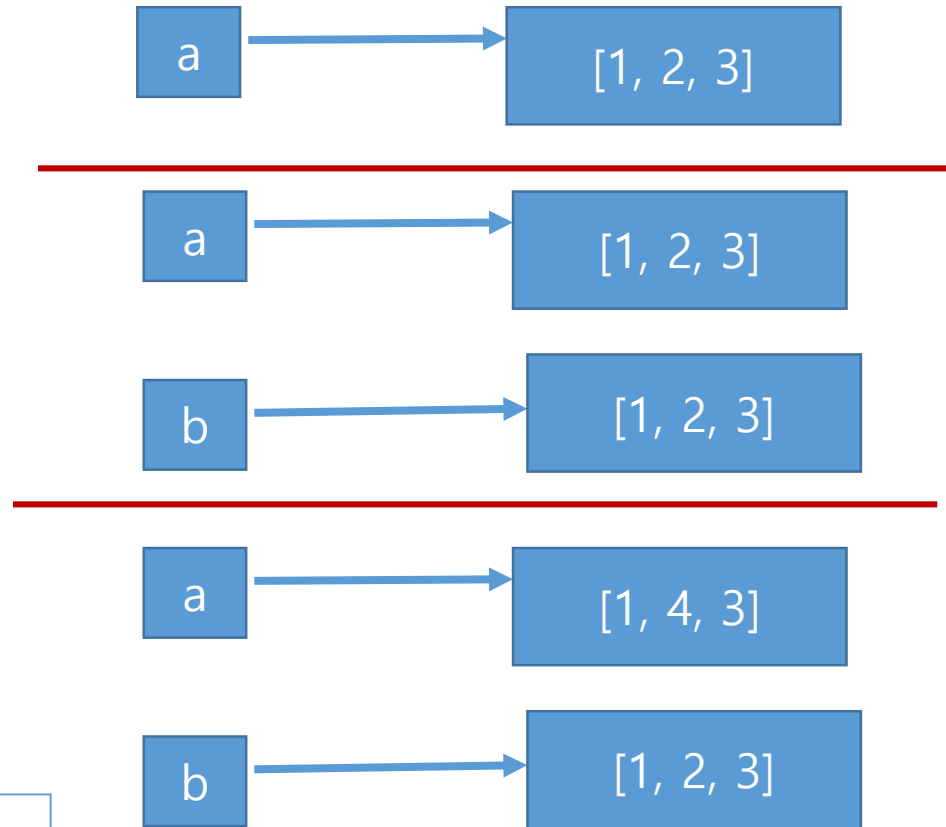
```
>>> b is a
>>> True
>>> b == a
>>> True
```


Aliasing and Shallow Copy [3/3]

Advanced된 Data Type들 (list, set, dictionary, tuple 등등) 값들을 물리적복제 (shallow copy)를 하려면.....

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 2, 3]
```

```
>>> from copy import copy
>>> b = copy(a)
```



```
>>> b is a
>>> False
>>> b == a
>>> True
```

```
>>> b is a
>>> False
>>> b == a
>>> False
```

Deep Copy는 나중에 배웁니다

모든 데이터는 생성되면 그 데이터가 참조되는 숫자를 유지

파이썬에는 입력한 자료형에 대한 참조 개수를 알려주는 `sys.getrefcount`라는 함수가 있다. 이 함수를 이용해 3이라는 정수형 객체에 참조 개수가 몇 개 있는지 살펴보자.

```
>>> import sys
>>> sys.getrefcount(3)
30
```

이후 `a = 3`, `b = 3`, `c = 3`과 같이 3을 가리키는 변수를 늘리면 참조 개수가 증가하는 것을 볼 수 있다.

```
>>> a = 3
>>> sys.getrefcount(3)
31
>>> b = 3
>>> sys.getrefcount(3)
32
>>> c = 3
>>> sys.getrefcount(3)
33
```

메모리에 생성된 변수 없애기

`a=3`을 입력하면 3이라는 정수형 객체가 메모리에 생성된다고 했다. 그렇다면 이 값을 메모리에서 없앨 수 있을까? 3이라는 객체를 가리키는 변수들의 개수를 레퍼런스 카운트라고 하였는데, 이 레퍼런스 카운트가 0이 되는 순간 3이라는 객체는 자동으로 사라진다. 즉, 3이라는 객체를 가리키고 있는 것이 하나도 없을 때 3이라는 객체는 메모리에서 사라지게 되는 것이다. 이것을 어려운 말로 가비지 콜렉션(Garbage collection, 쓰레기 수집)이라고도 한다.

다음은 특정한 객체를 가리키는 변수를 없애는 예이다.

```
>>> a = 3
>>> b = 3
>>> del(a)
>>> del(b)
```

```
>>> del a    # 똑같은 효과
```

Table of Contents

- Numeric Data Type
- String Data Type
- List Data Type
- More on Assignments
- More on Boolean Data Types

True or False of Various Data Types

값	참 or 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
0	거짓
{}	거짓
1	참
0	거짓
None	거짓

```
>>> a = [1, 2, 3, 4]
>>> while a:
...     a.pop()
...
4
3
2
1
```

```
>>> if [1, 2, 3]:
...     print("True")
... else:
...     print("False")
...
True
```

Boolean Expressions: As Short-Circuit Operators

- Python's Booleans are *short-circuit operators*:
 - meaning that a true or false is returned as soon as the result is known
- Python will not evaluate the second expression;
 - in an **and** where the **first expression is false** and
 - in an **or**, where the **first expression is true**

Operator	Operational definition	
x and y	If x is false, return <i>False</i> .	Otherwise, return y .
x or y	If x is true, return <i>True</i> .	Otherwise, return y .
not x	If x is false, return True.	Otherwise, return False.

Boolean Expressions: Confusion due to Short-Cut Property

- Suppose that a student intends to check if user's input is a yes
 `response = input("Type your decision: ")`
 `if response == "y" or "Y":`
 `print("ok")`
- What is the evaluation result of the expressions if `response = "y"`?
- What if `response = "N"`?
- Due to short-cut property, above expression will be evaluated as `(response == "y") or "Y"`
 - if `response == "y"` is True, ➔ True is returned
 - if `response == "y"` is not True, ➔ ("N"가 입력이 되어도) "Y" is returned (which is True)
- The correct way is
 `if (response == "y") or (response == "Y"):`
 `print("ok")`
- Or a simpler correct way is
 `if response.lower() == "y":`
 `print("ok")`