Table of Contents

Numeric Data Types [1/3]

항목	사용 예
정수	123, -345, 0
실수	123.45, -1234.5, 3.4e10
8진수	0o34, 0o25
16진수	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

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
`\n`	개행 (줄바꿈)
`\t`	수평 탭
`\\`	문자 "`\`"
`\'`	단일 인용부호(`'`)
`\"`	이중 인용부호(`"`)
`\r`	캐리지 리턴
`\f`	폼 피드
`\a`	벨 소리
`\b`	백 스페이스
`\000`	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 = "Life is too short, You need Python"
>>> b = a[0] + a[1] + a[2] + a[3]
>>> b
'Life'
```

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

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

String Data Types [6/13]

```
>>> a = "20010331Rainy"
>>> year = a[:4]
>>> day = a[4:8]
>>> weather = a[8:]
>>> year
'2001'
>>> day
'0331'
>>> weather
'Rainy'
```

["Pithon"이라는 문자열을 "Python"으로 바꾸려면?]

```
>>> 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 것이라고 예상하겠지만 파이썬은 값이 올바르지 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'
```

abcd라는 문자열의 각각의 문자 사이에 변수 a의 값인 ','를 삽입한다.

문자열 바꾸기(replace)

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

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']
```

String Data Types [13/13]

[고급 문자열 포매팅]

숫자 바로 대입하기

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

문자열 바로 대입하기

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

숫자 값을 가진 변수로 대입하기

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

Table of Contents

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 함수이다.

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

pop(x)는 리스트의 x번째 요소를 돌려 주고 그 요소는 삭제한다.

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

리스트 확장(extend)

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

리스트에 포함된 요소 x의 개수 세기(count)

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

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

```
>>> 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

파이썬에는 입력한 자료형에 대한 참조 개수를 알려주는 sys.getrefcount라는 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 = 3과 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
```

변수를 만드는 여러 가지 방법

```
>>> 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'
```

메모리에 생성된 변수 없애기

a=3을 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)
```

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

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

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

위의 예에서 b = copy(a)는 b = a[:]과 동일하다.

두 변수가 같은 값을 가지면서 다른 객체를 제대로 생성했는지 확인하려면 다음과 같이 is 함수를 이용하면 된다. 이 함수는 서로 동일한 객체인지 아닌지에 대한 판단을 하여 참과 거짓을 리턴한다.

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

Table of Contents

True or False of Various Data Types

값	참 or 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
()	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 *x*. Otherwise, return *y*.
x or *y*	If *x* is true, return *x*. 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: ")
```

- Suppose he wrote

```
if response == "y" or "Y"
    print("ok")
```

- What is the evaluation result of the expressions if `response = "y"`?
- What if `response = "Y"`?
- 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, ➔ "Y" is returned

- The correct way is

```
if (response == "y") or (response == "Y")
        print("ok")
```

- Or a simpler correct way is

```
if response.lower() == "y"
   print("ok")
```