

Programming in Python

06 – 문자열



2016년 8월, 국민대학교 컴퓨터공학부

문자열 (Strings)

문자열: 문자의 나열

문자: byte? character? multi-byte character?

- 문자열을 다루는 일은 실용 프로그램일수록 점점 중요해짐
- 특히, Python 과 같은 언어에서라면?
 - 문자열을 리스트, 사전 등의 인덱스로 이용할 수 있음
 - (데이터베이스에서의 키를 이용하는 것과 비슷한 연산을 언어가 제공)
- 특히, 영어로만 된 프로그램을 작성하는 것이 아니라면?
 - 한글, 한자, 일본어, 등등?
 - 인코딩 방식이 달라짐에 따라 여러 가지 처리가 필요해짐
 - 이 장에서 다룰 내용은 아님

문자열 상수 (String Literals)

- 작은 따옴표로 둘러싼 문자열
 - 'spa"m'
- 큰 따옴표로 둘러싼 문자열
 - "spa'm"
- 삼중 따옴표를 이용
 - ''' ... spam ... ''', """ ... spam ... """
- 이스케이프 순차
 - "s\tp\na\0m"
- Raw strings
 - r"C:\new\test.spm"
- 바이트 문자열
 - b'sp\x01am'
- 유니코드 문자열 (Python 2.6)
 - u'eggs\u0020spam'

각각을 Python interactive prompt 에 입력해 보고

```
>>> 'spa"m'
```

변수에 대입해서 프린트해 보세요.

```
>>> a = 'spa"m'
```

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

차이가 있나요?

작은 따옴표 문자열과 큰 따옴표 문자열

```
>>> 'shrubbery', "shrubbery"  
( 'shrubbery', 'shrubbery' )
```

```
>>> 'shrubbery' == "shrubbery"  
True
```

```
>>> 'shrubbery' is "shrubbery"  
True
```

왜 두 가지 방법을 제공하는가?
(따옴표 자체가 문자열에 포함되는 경우는?)

```
>>> 'knight"s', "knight's"  
( 'knight"s', "knight's" )
```

```
>>> 'knight's'  
File "<stdin>", line 1  
    'knight's'  
      ^  
SyntaxError: invalid syntax
```

두 종류의 따옴표가 모두 포함된 문자열을 원할 때는?
→ 이스케이프 순차를 이용

```
>>> "I'm here to say \"hello\"."  
'I\'m here to say "hello".'  
>>> print("I'm here to say \"hello\". ")  
I'm here to say "hello".
```

문자열의 병합 (String Concatenation)

```
>>> title = "Meaning " 'of' " Life"  
>>> title  
'Meaning of Life'
```

표현식에 연달아 등장하는 문자열 상수는
묵시적으로 병합됨

```
>>> title = "Meaning " + 'of' + " Life"  
>>> title  
'Meaning of Life'
```

두 문자열에 대한 '+' 연산자는
문자열 병합

```
>>> title + title  
'Meaning of LifeMeaning of Life'
```

이스케이프 순차 (Escape Sequences)

```
>>> S = "s\tp\na\x00m"
```

```
>>> S
```

```
's\tp\na\x00m'
```

```
>>> len(S)
```

```
7
```

```
>>> print(S)
```

```
s          p
```

```
am
```

```
>>> x = "C:\py\code"
```

```
>>> x
```

```
'C:\\py\\code'
```

```
>>> len(x)
```

```
10
```

비교:

```
>>> myfile = open('C:\new\text.dat', 'w')
```

```
>>> myfile = open(r'C:\new\text.dat', 'w')
```

Escape	Meaning
<code>\newline</code>	Ignored (continuation)
<code>\\</code>	Backslash (keeps a \)
<code>\'</code>	Single quote (keeps ')
<code>\"</code>	Double quote (keeps ")
<code>\a</code>	Bell
<code>\b</code>	Backspace
<code>\f</code>	Formfeed
<code>\n</code>	Newline (linefeed)
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical tab
<code>\N{id}</code>	Unicode database ID
<code>\uhhhh</code>	Unicode 16-bit hex
<code>\Uhhhh...</code>	Unicode 32-bit hex ^a
<code>\xhh</code>	Hex digits value
<code>\ooo</code>	Octal digits value
<code>\0</code>	Null (doesn't end string)
<code>\other</code>	Not an escape (kept)

블록 문자열 (Block Strings)

삼중 따옴표 (큰 따옴표 또는 작은 따옴표 이용) 로 둘러싸인 문자열
→ 있는 그대로 (줄바꿈을 포함하여) 문자열 상수로 취급

```
>>> mantra = """Always look
... on the bright
... side of life."""
```

```
>>> mantra
'Always look\n on the bright\nside of life.'
```

```
>>> print(mantra)
Always look
on the bright
side of life.
```

이 방법은 코드 블록을
여러 줄에 걸친 주석으로 처리하는 데에도
많이 이용됨 (일시적으로 코드 무효화한다든지)

```
X = 1
"""
import os
print(os.getcwd())
"""
Y = 2
```

기본 문자열 연산

```
>>> len('abc')  
3
```

문자열의 길이: 포함된 문자의 수

비교:

```
>>> 'abc' + 'def'  
'abcdef'
```

문자열 병합

```
>>> len('국민대학교')  
15
```

```
>>> 'Ni!' * 4  
'Ni!Ni!Ni!Ni!'
```

문자열 반복

```
>>> myjob = "hacker"
```

```
>>> for c in myjob: print(c, end=' ')
```

각 문자에 대하여 반복

```
h a c k e r
```

```
>>> "k" in myjob
```

문자열 내에 포함

```
True
```

```
>>> "z" in myjob
```

문자열 내에 포함되지 않음

```
False
```

```
>>> 'spam' in 'abcspamdef'
```

포지션을 알려주지는 않음

```
True
```


문자열 인덱싱 (Indexing)

```
>>> S = 'Spam'
>>> len(S)          # 문자열의 길이를 파악
4
>>> S[0]            # S 를 구성하는 첫번째 원소 (문자) - 포지션은 0 부터 시작함
'S'
>>> S[1]            # 왼쪽으로부터 두번째 위치하는 원소 (문자)
'p'
>>> S[-1]           # S 를 구성하는 마지막 문자 (가장 오른쪽)
'm'
>>> S[-2]           # 마지막으로부터 두번째 위치하는 원소 (문자)
'a'
>>> S[len(S) - 1]   # 어떻게?
???
```

문자열 슬라이싱 (Slicing)

```
>>> S = 'Spam'
>>> S[1:3]      # 포지션 1 부터 시작하여 2 까지 (3 미만) 로 이루어진 S 의 슬라이스
'pa'
>>> S[1:]       # 끝 인덱스를 명시하지 않으면 마지막까지
'pam'
>>> S[0:3]      # 처음부터 시작해서 3 미만까지
'Spa'
>>> S[:3]       # 시작을 명시하지 않으면 처음부터 (S[0:3] 와 동일)
'Spa'
>>> S[:-1]      # 끝 인덱스를 마지막 문자 하나를 제외하는 것으로 지정
'Spa'
>>> S[:]        # 범위는 [0:len(S)] 와 동일 (len(S) = 4)
'Spam'
```



문자열 연산

```
>>> S = 'Spam'
>>> S.find('pa')      # 처음으로 등장하는 substring 의 오프셋 (인덱스)
1
>>> S.find('xy')      # 발견되지 않으면?
-1
>>> S.replace('pa', 'XYZ')  # 'pa' 를 찾아 다른 문자열로 대체
'SXYZm'
>>> S                # 하지만 S 는 변화하지 않음
'Spam'
>>> S.upper()        # 포함된 모든 문자를 대문자로
'SPAM'
>>> S.lower()        # 포함된 모든 문자를 소문자로
'spam'
>>> S                # 하지만 S 는 변화하지 않음
'Spam'
>>> S.isalpha()      # 알파벳으로 이루어져 있는지? (그렇다면 isdigit() 는?)
True
```

문자열 연산

```
>>> line = 'aaa,bbb,cccc,dd'
>>> x = line.split(',') # ',' 를 구분자 (delimiter) 로 하여 문자열로부터 substring 의 리스트를 생성
>>> x
['aaa', 'bbb', 'cccc', 'dd']
```

```
>>> ','.join(x)
'aaa,bbb,cccc,dd'
```

```
>>> line = 'aaa,bbb,cccc,dd\n'
>>> line = line.rstrip() # 문자열의 오른쪽 끝에서 공백 문자나 개행 문자를 제거
>>> line
'aaa,bbb,cccc,dd'
```

```
>>> line = '  abcd'
>>> line.lstrip() # 문자열의 왼쪽 끝에서 공백 문자나 개행 문자를 제거
'abcd'
```

문자열 포매팅 표현식 (String Formatting Expressions)

% 연산자 (문자열 대체 연산) 를 이용

```
>>> 'That is %d %s bird!' % (1, 'dead')
'That is 1 dead bird!'
```

```
>>> exclamation = "Ni"
>>> "The knights who say %s!" % exclamation
'The knights who say Ni!'
```

```
>>> "%d %s %d you" % (1, 'spam', 4)
'1 spam 4 you'
```

```
>>> "%s -- %s -- %s" % (42, 3.14159, [1, 2, 3])
'42 -- 3.14159 -- [1, 2, 3]'
```

```
>>> "%d %s %d you" % (1, 'spam')
TypeError: not enough arguments for format string
```

```
>>> "%d birds" % 'x'
TypeError: %d format: a number is required, not str
```

Code	Meaning
%s	String (or any object)
%r	s, but uses repr, not str
%c	Character
%d	Decimal (integer)
%i	Integer
%u	Unsigned (integer)
%o	Octal integer
%x	Hex integer
%X	x, but prints uppercase
%e	Floating-point exponent
%E	e, but prints uppercase
%f	Floating-point decimal
%g	Floating-point e or f
%G	Floating-point E or f
%%	Literal %

문자열 포매팅 (String Formatting)

```
>>> x = 1234
```

```
>>> res = "integers: ...%d...%-6d...%06d" % (x, x, x)
```

표시될 자릿수를 지정

```
>>> res
```

```
'integers: ...1234...1234 ...001234'
```

```
>>> x = 1.23456789
```

```
>>> '%e | %f | %g' % (x, x, x)
```

표시될 형식을 지정

```
'1.234568e+00 | 1.234568 | 1.23457'
```

```
>>> '%-6.2f | %05.2f | %+06.1f' % (x, x, x)
```

정밀도를 지정

```
'1.23 | 01.23 | +001.2'
```

```
>>> '%f, %.2f, %.*f' % (1/3.0, 1/3.0, 4, 1/3.0)
```

```
'0.333333, 0.33, 0.3333'
```

사전을 이용한 문자열 포매팅

```
>>> "%(n)d %(x)s" % {"n":1, "x":"spam"}  
'1 spam'
```

n 과 x 를 키로 가지는 사전을 이용

```
>>> reply = ""  
... Greetings...  
... Hello %(name)s!  
... Your age sqaured is %(age)s  
... ""  
>>> values = {'name': 'Bob', 'age': 40}  
>>> print(reply % values)
```

name 과 age 를 키로 가지는 사전을 이용

```
Greetings...  
Hello Bob!  
Your age sqaured is 40
```

```
>>> food = 'spam'  
>>> age = 40  
>>> vars()  
{'age': 40, 'food': 'spam', ...}  
>>> "%(age)d %(food)s" % vars()  
'40 spam'
```

vars() 는 변수 이름과 그 값을 대응시키는 사전!

문자열 포매팅 메서드

```
>>> template = '{0}, {1} and {2}'  
>>> template.format('spam', 'ham', 'eggs')  
'spam, ham and eggs'
```

```
>>> template = '{motto}, {pork} and {food}'  
>>> template.format(motto='spam', pork='ham', food='eggs')  
'spam, ham and eggs'
```

```
>>> template = '{motto}, {0} and {food}'  
>>> template.format('ham', motto='spam', food='eggs')  
'spam, ham and eggs'
```

```
>>> '{motto}, {0} and {food}'.format(42, motto=3.14, food=[1, 2])  
'3.14, 42 and [1, 2]'
```

```
>>> import sys  
>>> 'My {1[spam]} runs {0.platform}'.format(sys, {'spam': 'laptop'})  
'My laptop runs darwin'
```


문자 코드 변환

ord 와 chr 의 Python built-in 함수를 이용하여 문자와 그 코드 사이의 변환을 행할 수 있음

```
>>> ord('s')
115
>>> chr(115)
's'
>>> S = '5'
>>> S = chr(ord(S) + 1)
>>> S
'6'
>>> ord('5') - ord('0')
5
>>> B = '1101'
>>> l = 0
>>> while B != '':
...     l = l * 2 + (ord(B[0]) - ord('0'))
...     B = B[1:]
...
>>> l
13
```

그런데, 이렇게 할 이유가 있을까?

Quiz

- 문자열은 변경 가능한가?
 - `>>> x = '1234'`
 - `>>> x[0] = 'a'`
- 그렇다면, 문자열을 변경하려면 어떻게 해야 하는가?
 - `replace` 같은 메서드들이 있는데도?
- `S = 's,pa,m'` 일 때, 가운데 두 글자 ('pa') 를 추출하여 `x` 에 담는 Python 코드는?
 - 문자열 슬라이싱을 이용하여
 - 문자열 메서드를 이용하여

Exercise

- (1) 문자열을 인자로 받아, 그 문자열을 역순으로 재구성한 것을 반환하는 함수를 작성해 보자.
 - 예: 입력이 "Now I see the light." 이라면, 결과는 ".thgil eht ees I woN"

- (2) 문자열을 인자로 받아, 그 문자열이 palindrome 인지 판단하는 함수를 작성해 보자.
 - Palindrome: 앞뒤 순서를 바꾸어도 동일한 문자열
 - 알파벳이 아닌 문자 (구두점이나 숫자 등) 는 무시하자.
 - 예: Madam I'm Adam.

Answers

```
def revstr(s):  
    x = ""  
    for i in range(len(s)):  
        x = s[i] + x  
  
    return x
```

```
def revstr(s):  
    return s[-1::-1]
```

```
def rec_ispalin(s):  
    x = ""  
    for i in range(len(s)):  
        if s[i].isalpha():  
            x += s[i].lower()  
    if len(x) == 0 or len(x) == 1:  
        return True  
    elif x[0] == x[-1] and palin(x[1:-1]):  
        return True  
    else:  
        return False
```

```
def ispalin(s):  
    x = ""  
    for i in range(len(s)):  
        if s[i].isalpha():  
            x += s[i].lower()  
    l = 0  
    r = len(x) - 1  
    while l < r:  
        if x[l] != x[r]:  
            break  
        l += 1  
        r -= 1  
    else:  
        return True  
    return False
```

Q & A