

문자열

제6장

Python for Everybody
www.py4e.com



문자열 자료형

- 문자열은 문자 시퀀스
- 문자열은 따옴표를 사용해서 표기
'Hello' 또는 "Hello"
- 문자열에서, + 연산자는 “병합”을 의미
- 문자열이 숫자를 포함하고 있어도
여전히 문자열
- `int()` 함수를 이용해서 문자열 안의
숫자를 정수형으로 변환 가능

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

읽기와 변환하기

- 문자열을 통해 데이터를 읽고 파싱하고 필요한 데이터를 변환 하는 것을 선호
- 이 예시는 에러와 잘못된 사용자 입력에 대한 상황 파악에 좋은 길잡이
- 숫자 입력은 문자열에서 변환 되어야 함

```
>>> name = input('Enter:')
Enter:Chuck
>>> print(name)
Chuck
>>> apple = input('Enter:')
Enter:100
>>> x = apple - 10
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```



문자열 파악하기

- 문자열에 있는 어떤 문자든지 **대괄호** 안에 지정된 인덱스를 이용해서 가져올 수 있음
- 인덱스 값은 정수이고 0에서 시작
- 인덱스로 계산 가능한 표현식을 사용 가능

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

범위 밖 문자

- 문자열 크기를 넘어서
인덱스에 접근하려고 하면
파이썬 에러가 발생
- 인덱스 값을 계산 하거나
문자열을 자를 때 주의

```
>>> zot = 'abc'
>>> print(zot[5])
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
IndexError: string index out
of range
>>>
```

문자열의 길이

내장 함수 **len**는 문자열의
길이를 반환

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> print(len(fruit))
6
```

len 함수

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

함수는 우리가 사용하는 어떤 저장된 코드.

함수는 입력을 받아서
출력

'banana'
(문자열)



len()
함수



6
(숫자)

len 함수

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

함수는 우리가 사용하는 어떤 저장된 코드.

함수는 입력을 받아서 출력

'banana'
(a string)



```
def len(inp):
    blah
    blah
    for x in y:
        blah
        blah
```



6
(숫자)

문자열을 통한 루프

while 구문, 반복 변수, **len**
함수를 이용해서 문자열
안에 있는 각 문자를
독립적으로 확인하는
루프를 만들 수 있음

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

0 b
1 a
2 n
3 a
4 n
5 a

문자열을 통한 루프

- **for** 구문을 이용하는 유한 루프가 더 깔끔함
- 반복 변수는 **for** 루프에 의해 완벽하게 관리됨

```
fruit = 'banana'
for letter in fruit:
    print(letter)
```

b
a
n
a
n
a

문자열을 통한 루프

- **for** 구문을 이용하는 유한 루프가 더 깔끔함
- 반복 변수는 **for** 루프에 의해 완벽하게 관리됨

```
fruit = 'banana'
for letter in fruit :
    print(letter)
```

```
index = 0
while index < len(fruit) :
    letter = fruit[index]
    print(letter)
    index = index + 1
```

b
a
n
a
n
a

루프와 개수 세기

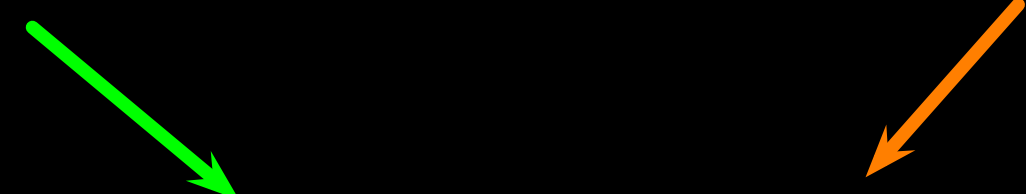
문자열에 있는 각 문자에
대해 루프를 실행해서 문자
'a'의 개수를 세는 간단한
루프

```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

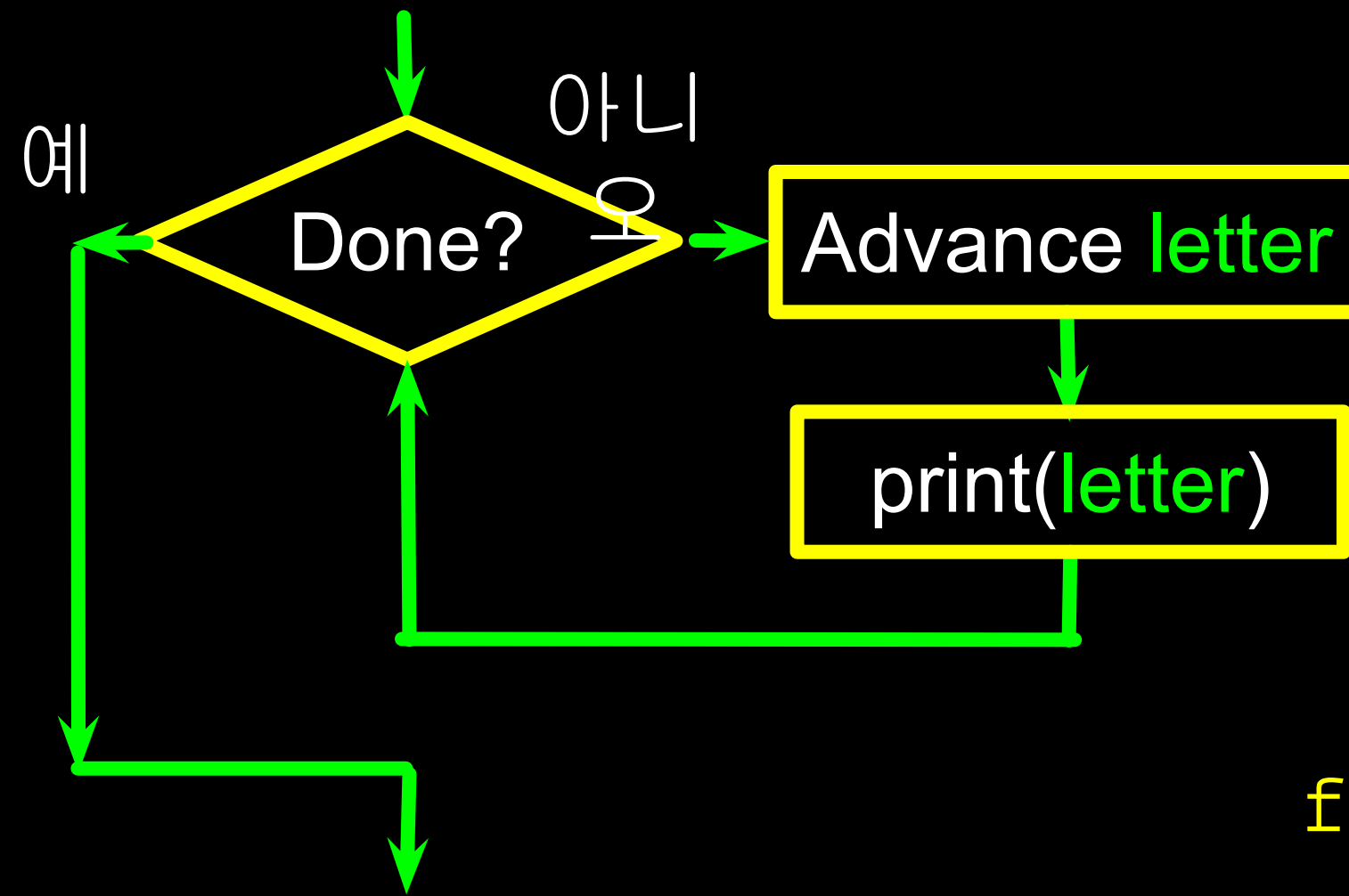
in 파헤치기

- 반복 변수는 시퀀스(순서가 있는 집합)를 통해 “반복”
- 코드의 루프 블록 (본문)은 시퀀스 안의 각 값에 대해 한번씩 실행
- 반복 변수는 시퀀스 안의 모든 값을 가지고 실행

반복 변수 문자 6개로 이루어진 문자열



```
for letter in 'banana' :  
    print(letter)
```



b	a	n	a	n	a
---	---	---	---	---	---

```
for letter in 'banana' :  
    print(letter)
```

반복 변수 문자열을 통해 “반복”하고
코드 블록 (본문) 은 시퀀스 안에 있는 값 하나에 대해서 한 번씩 실행

다른 문자열 연산들

문자열 슬라이싱

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- 콜론 연산자를 사용해서 문자열의 연속적인 구간을 가져올 수 있음
- 두 번째 숫자는 문자열 조각보다 한 글자 나머를 가리킴 - “~까지 이지만 포함하지 않음”
- 두 번째 숫자가 문자열 마지막 나머를 가리키는 경우 문자열의 마지막에서 멈춤

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```


문자열 슬라이싱

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

첫 번째 숫자나 두 번째 숫자를
생략하면 각각 문자의 시작과
마지막을 가리킨다고 가정

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

문자열 병합

+ 연산자가 문자열에
적용되면, “병합”을 의미

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

논리 연산자로서의 in

- **in** 키워드는 어떤 문자열이 다른 문자열에 “포함”되는지 확인하기 위해서도 사용
- **in** 표현식은 참 또는 거짓값을 반환하는 논리 표현식이며 **if** 구문에 사용될 수 있음

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

문자열 비교

```
if word == 'banana':  
    print('All right, bananas.')  
if word < 'banana':  
    print('Your word, ' + word + ', comes before banana.')elif word > 'banana':  
    print('Your word, ' + word + ', comes after banana.')else:  
    print('All right, bananas.')
```

문자열 라이브러리

- 파이썬은 여러 개의 문자열
함수를 정의하는 문자열
라이브러리가 존재
- 이 함수는 모든 문자열에 이미
내장 되어 있음 - 함수를 문자열
변수에 붙임으로써 호출
- 이 함수는 원본 문자열을
수정하지 않고, 대신 바뀐
새로운 문자열을 반환

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
Hello Bob
>>> print('Hi There'.lower())
hi there
>>>
```

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rstrip', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

<https://docs.python.org/3/library/stdtypes.html#string-methods>

str.replace(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

str.rfind(*sub*[, *start*[, *end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within *s*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

str.rindex(*sub*[, *start*[, *end*]])

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

str.rjust(*width*[, *fillchar*])

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

str.rpartition(*sep*)

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

str.rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

문자열 라이브러리

`str.capitalize()`

`str.center(width[, fillchar])`

`str.endswith(suffix[, start[, end]])`

`str.find(sub[, start[, end]])`

`str.lstrip([chars])`

`str.replace(old, new[, count])`

`str.lower()`

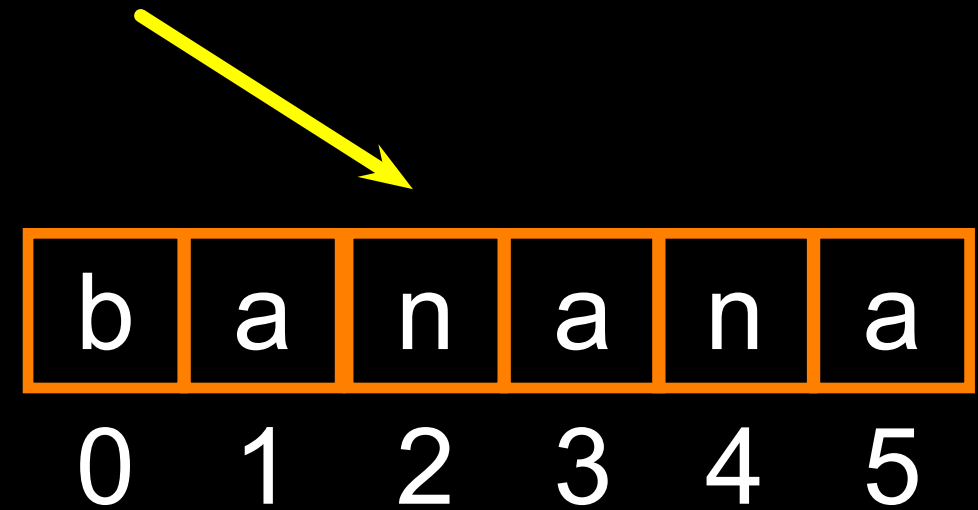
`str.rstrip([chars])`

`str.strip([chars])`

`str.upper()`

문자열 탐색

- `find()` 함수를 이용해서 하위 문자열을 다른 문자열에서 탐색 가능
- `find()` 하위 문자열의 첫 번째로 나타나는 위치를 검색
- 하위 문자열을 찾지 못하면, `find()` 는 `-1`을 반환
- 문자열 좌표는 0에서 시작한다는 것에 주의



```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

모든 문자를 대문자로 만들기

- 문자열의 복사본을 모두 You can make a copy of a string in 소문자 또는 대문자로 치환 가능
- `find()`를 이용해서 문자열을 탐색할 때, 문자열을 먼저 소문자로 바꾼 뒤 탐색하면 대소문자와 관계 없이 문자열을 탐색 가능

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print(nnn)
HELLO BOB

>>> www = greet.lower()
>>> print(www)
hello bob

>>>
```

찾아서 바꾸기

- The `replace()` 함수는
워드프로세서에서 “찾아서
바꾸기”와 같은 역할
- 나타나는 모든 탐색
문자열을 대체 문자열로
치환

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob', 'Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o', 'x')
>>> print(nstr)
Hel1x Bxb
>>>
```

공백 제거

- 종종 문자열의 끝과 마지막에 남아있는 공백을 제거
- `lstrip()` 과 `rstrip()`은 각각 문자열 왼쪽과 오른쪽에 있는 공백을 제거
- `strip()` 문자열 시작과 끝에 있는 모든 공백을 제거

```
>>> greet = '    Hello Bob    '  
>>> greet.lstrip()  
'Hello Bob '  
>>> greet.rstrip()  
'    Hello Bob'  
>>> greet.strip()  
'Hello Bob'  
>>>
```

접두사

```
>>> line = 'Please have a nice day'
```

```
>>> line.startswith('Please')
```

```
True
```

```
>>> line.startswith('p')
```

```
False
```

파싱과 추출

21



31



From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ', atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```



두 종류의 문자열

Python 2.7.10

```
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

Python 3.5.1

```
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

파이썬 3에서는, 모든 문자열이
유니코드입니다

요약

- 문자열 자료형
- 읽기와 변환
- 문자열 인덱싱 []
- 문자열 슬라이싱 [2:4]
- **for** 루프와 **while** 루프를 이용해서 문자열로 반복문 실행하기
- **+** 로 문자열 합치기
- 문자열 연산
- 문자열 라이브러리
- 문자열 비교
- 문자열에서 탐색하기
- 문자열 바꾸기
- 공백 제거하기



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance
(www.dr-chuck.com) of the University of Michigan School of
Information and open.umich.edu and made available under a
Creative Commons Attribution 4.0 License. Please maintain this
last slide in all copies of the document to comply with the
attribution requirements of the license. If you make a change,
feel free to add your name and organization to the list of
contributors on this page as you republish the materials.

...

Initial Development: Charles Severance, University of Michigan
School of Information

Contributor:

- Seung-June Lee (plusjune@gmail.com)
- Connect Foundation

Translator:

- Hakyong Kim
- Jeungmin Oh (tangza@gmail.com)