정보교육을 위한 파이썬

정보 탐색

Version 0.0.9-d2

저자: Charles Severance
번역: 이광춘, 한정수
(xwmooc)

Printing history:

October 2013: Major revision to Chapters 13 and 14 to switch to JSON and use OAuth. Added new chapter on Visualization.

September 2013: Published book on Amazon CreateSpace

January 2010: Published book using the University of Michigan Espresso Book machine.

December 2009: Major revision to chapters 2-10 from *Think Python: How to Think Like a Computer Scientist* and writing chapters 1 and 11-15 to produce *Python for Informatics: Exploring Information*

June 2008: Major revision, changed title to *Think Python: How to Think Like a Computer Scientist*.

August 2007: Major revision, changed title to *How to Think Like a (Python) Programmer*.

April 2002: First edition of *How to Think Like a Computer Scientist*.

The LaTeX source for the *Think Python: How to Think Like a Computer Scientist* version of this book is available from `http://www.thinkpython.com`.

Chapter 1

문자열

1.1 문자열은 순열이다.

문자열은 문자의 순열이다. 대괄호 연산자로 한번에 하나씩 문자에 접근할 수 있다.

```
>>> fruit = 'banana'
>>> letter = fruit[1]
```

두 번째 명령문은 변수 `fruit`에서 1번 위치의 문자를 추출하여 변수 `letter`에 할당한다. 대괄호의 표현식을 **인덱스(index)**라고 부른다. 인덱스는 순열의 무슨 문자를 사용자가 원하는지 표시한다.

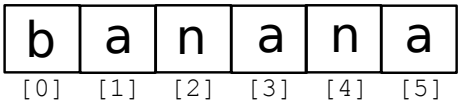
하지만, 여려분이 기대한 것을 얻지 못합니다.

```
>>> print letter
a
```

대부분의 사람에게 `'banana'`의 첫 분자는 `a`가 아니라 `b`입니다. 하지만, 파이썬에서 인텍스는 문자열의 처음부터 값이 시작됩니다. 첫 글자의 시작 값(오프셋, offset)은 0입니다.

```
>>> letter = fruit[0]
>>> print letter
b
```

`b`가 `'banana'`의 0번째 문자가 되고 `a`가 첫번째, `n`이 두번째 문자가 됩니다.



인덱스로 문자와 연산자를 포함하는 어떤 표현식도 사용가능지만, 인덱스의 값은 정수여야 합니다. 정수가 아닌 경우 다음과 같은 결과를 얻게 됩니다.

```
>>> letter = fruit[1.5]
TypeError: string indices must be integers
```

1.2 len함수를 사용하여 문자열의 길이 구하기

len은 문자열의 문자 갯수를 반환하는 내장함수다.

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

문자열의 가장 마지막 문자를 얻기 위해서, 아래와 같이 시도하려고 할 것입니다.

```
>>> length = len(fruit)
>>> last = fruit[length]
IndexError: string index out of range
```

IndexError의 이유는 'banana'에 6번 인덱스의 문자가 없기 때문입니다. 0에서부터 시작했기 때문에 6개의 문자는 0 5로 번호가 매겨졌습니다. 마지막 문자를 얻기 위해서 length에서 1을 빼야 합니다.

```
>>> last = fruit[length-1]
>>> print last
a
```

대안으로 문자의 끝에서 역으로 수를 세는 음의 인덱스를 사용할 수 있습니다.

fruit[-1]는 마지막 문자를 fruit[-2]는 끝에서 두 번째 등등 활용할 수 있습니다.

1.3 루프를 사용한 문자열 운행법

많은 연산의 경우 문자열을 한번에 한 문자씩 처리하는 합니다. 종종 처음에서 시작해서, 차례로 각 문자를 선택하고, 선택된 문자에 임의의 연산을 수행하고, 끝까지 계속합니다. 이런 처리 패턴을 **운행법(traversal)**라고 합니다. 운행법을 작성하는 한 방법은 while 루프입니다.

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

while 루프가 문자열을 운행하여 문자열을 한줄에 한자씩 화면에 출력합니다. 루프 조건이 index < len(fruit)이여서, index가 문자열의 길와 같을 때, 조건은 거짓이 되고, 루프의 몸통 부문은 실행이 되지 않습니다. 파이썬이 접근한 마지막 문자는 문자열의 마지막 문자인 len(fruit)-1 인덱스의 문자입니다.

Exercise 1.1 문자열의 마지막 문자에서 시작해서,문자열의 처음으로 역진행하면서 한줄에 한자씩 화면에 출력하는 while 루프를 작성하세요.

운행법을 작성하는 다른 방법은 for 루프입니다.

```
for char in fruit:
    print char
```

루프를 매번 반복할 때, 문자열의 다음 문자가 변수 char에 할당됩니다. 루프는 더 이상 남겨진 문자가 없을 때까지 계속 실행됩니다.

1.4 문자열 조각

문자열의 일부분을 **문자열 조각(slice)**이라고 합니다. 문자열 조각을 선택하는 것은 문자를 선택하는 것과 유사합니다.

```
>>> s = 'Monty Python'
>>> print s[0:5]
Monty
>>> print s[6:13]
Python
```

[n:m] 연산자는 n번째 문자부터 m번째 문자까지의 문자열 - 첫 번째는 포함하지만 마지막은 제외 - 부분을 반환합니다.

콜론 앞의 첫 인텍스를 생략하면, 문자열 조각은 문자열의 처음부터 시작합니다. 두 번째 인텍스를 생략하면, 문자열 조각은 문자열의 끝까지 갑니다.

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

만약 첫번째 인텍스가 두번째보다 크거나 같은 경우 결과는 두 인용부호로 표현되는 **빈 문자열(empty string)**이 됩니다.

```
>>> fruit = 'banana'
>>> fruit[3:3]
''
```

빈 문자열은 어떠한 문자도 포함하고 있지 않아서 길이가 0 이지만, 이외에 다른 문자열과 동일합니다.

Exercise 1.2 fruit이 문자열로 주어졌을 때, fruit[:]의 의미는 무엇인가요?

1.5 문자열은 불변이다.

문자열의 문자를 변경하려는 의도로 할당문의 왼쪽편에 [] 연산자를 사용하고 싶은 유혹이 있을 것입니다. 예를 들어 다음과 같습니다.

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: object does not support item assignment
```

이 경우 "개체"는 문자열이고, 할당하고자 하는 문자는 "항목"이다. 지금 당장은 **개체**는 값과 같은 것이지만, 후에 좀더 정의를 상세화할 것입니다. **항목**은 순열의 값중의 하나입니다.

오류의 이유는 문자열이 **불변(immutable)**이기 때문입니다. 따라서 존재하는 문자열을 바꿀 수 없다는 의미입니다. 최선은 원래 문자열에 변화를 준 새로운 문자열을 생성하는 것입니다.

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print new_greeting
Jello, world!
```

새로운 첫 문자에 greeting 문자열 조각을 연결해서, 원래 문자열에는 어떤한 영향도 주지 않는 새로운 문자열을 만들었습니다.

1.6 루프 돌기(looping)와 세기(counting)

다음 프로그램은 문자열에 문자 a가 나타나는 횟수를 셉니다.

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

상기 프로그램은 **카운터(counter)**라고 부르는 또다른 연산 패턴을 보여줍니다. 변수 count는 0으로 초기화 되고, 매번 a를 찾을 때마나 증가합니다. 루프를 빠져나갔을 때, count는 결과 값, a가 나타난 총 횟수를 담고 있습니다.

Exercise 1.3 상기 코드를 캡슐화(encapsulation)하여 문자열과 문자를 인수로 받는 count라는 함수를 작성해서 일반화하세요.

1.7 in 연산자

연산자 in은 불 연산자로 두개의 문자열을 받아, 첫 번째 문자열이 두 번째 문자열의 일부이면 참(True)을 반환한다.

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

1.8 문자열 비교

비교 연산자도 문자열에서 동작합니다. 두 문자열이 같은지를 살펴봅시다.

```
if word == 'banana':
    print  'All right, bananas.'
```

다른 비교 연산자는 단어를 알파벳 순으로 정렬하는데 유용하다.

```
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.'
```

파이썬은 사람과 동일하는 방식으로 대문자와 소문자를 다루지 않습니다. 모든 대문자는 소문자 앞에 위치합니다.

```
Your word, Pineapple, comes before banana.
```

이러한 문제를 다루는 일반적인 방식은 비교연산을 수행하기 전에 문자열을 표준 포맷으로 예를 들어 모두 소문자, 변환하는 것입니다. "Pineapple"로 무장한 사람들로부터 여러분을 보호하는 경우를 명심하세요.

1.9 string 메쏘드

문자열은 파이썬 **개체(objects)**의 한 예이다. 개체는 데이터(실제 문자열 자체)와 **메쏘드(methods)**를 담고 있다. 메쏘드는 개체내부에 내장되고 개체의 어떤 **인스턴스(instance)**에도 사용될 수 있는 효과적인 함수다.

파이썬은 개체에 대해서 이용가능한 메쏘드를 보여주는 dir 함수가 있다. type 함수는 개체의 형(type)을 보여주고, dir은 개체의 사용될 수 있는 메쏘드를 보여준다.

```
>>> stuff = 'Hello world'
>>> type(stuff)
<type 'str'>
>>> dir(stuff)
['capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
>>> help(str.capitalize)
Help on method_descriptor:

capitalize(...)
    S.capitalize() -> string

    Return a copy of the string S with only its first character
    capitalized.
>>>
```

dir 함수가 메쏘드를 보여주고, 메쏘드에 대한 간단한 문서를 help를 사용할 수 있지만, 문자열 메쏘드에 대한 좀더 좋은 문서 정보는 docs.python.org/library/string.html에서 찾을 수 있다.

인수를 받고 값을 반환한다는 점에서 **메쏘드(method)**를 호출하는 것은 함수를 호출하는 것과 유사하지만, 구문은 다르다. 구분자로 점을 사용해서 변수명에 메쏘드명을 붙여 메쏘드를 호출한다.

예를 들어, upper 메쏘드는 문자열을 받아 모두 대문자로 변경된 새로운 문자열을 반환한다.

함수 구문 upper(word) 대신에, word.upper() 메쏘드 구문을 사용한다.

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print new_word
BANANA
```

이런 형태의 점 표기법은 메쏘드 이름(upper)과 메쏘드가 적용되는 문자열 이름(word)을 명세한다. 빈 괄호는 메쏘드가 인수를 갖지 않는 것을 나타낸다.

메쏘드를 부르는 것을 **호출(invocation)**이라고 부른다. 상기의 경우, word에 upper 메쏘드를 호출한다고 말한다.

예를 들어, 문자열안에 한 문자의 위치를 찾는 find라는 문자열 메쏘드가 있다.

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print index
1
```

상기 예제에서, word 문자열의 find 메쏘드를 호출하여 매개 변수로 찾고 있는 문자를 넘긴다.

find 메쏘드는 문자뿐만 아니라 부분 문자열도 찾을 수 있다.

```
>>> word.find('na')
2
```

검색 시작 위치를 지정하는 두 번째 인텍스를 인수로 갖을 수도 있다.

```
>>> word.find('na', 3)
4
```

한가지 자주 있는 적업은 strip 메쏘드를 사용해서 문자열의 시작과 끝의 공백(공백 여러개, 탭, 새줄)을 제거하는 것이다.

```
>>> line = '  Here we go  '
>>> line.strip()
'Here we go'
```

startswith 메쏘드는 참, 거짓 같은 불 값을 반환한다.

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

startswith가 대소문자를 구별하는 것을 요구하기 때문에 `lower` 메쏘드를 사용해서 임의의 검증을 수행하기 전에, 한 줄을 입력받아 모두 소문자로 변환하는 것이 필요하다.

```
>>> line = 'Please have a nice day'
>>> line.startswith('p')
False
>>> line.lower()
'please have a nice day'
>>> line.lower().startswith('p')
True
```

마지막 예제에서 결과 문자열이 문자 "p"로 시작하는지를 검증하기 위해서, `lower` 메쏘드가 호출되고 나서 바로 `startswith` 메쏘드를 사용한다. 순서만 주의깊게 다룬다면, 한줄에 여러개의 메쏘드를 호출할 수 있다.

Exercise 1.4 앞선 예제와 유사한 함수인 `count`로 불리는 문자열 메쏘드가 있다. `docs.python.org/library/string.html`에서 `count` 메쏘드에 대한 문서를 읽고, 문자열 `'banana'`의 문자가 몇 개인지 세는 메쏘드 호출 프로그램을 작성하세요.

1.10 문자열 파싱(Parsing)

종종, 문자열을 들여다 보고 부속 문자열(substring)을 찾고 싶다. 예를 들어, 아래와 같은 형식으로 구성된 일련의 자료 라인이 주어졌다고 가정하면,

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

각 라인의 뒤쪽 전자우편 주소(`uct.ac.za`)만 뽑아내고 싶을 것이다. `find` 메쏘드와 문자열 조각(string sliceing)을 사용해서 작업을 수행할 수 있다.

우선, 문자열에서 골뱅이(@, at-sign) 기호의 위치를 찾는다. 그리고 골뱅이 기호 *뒤* 첫 공백 위치를 찾는다. 그리고 나서 찾고자 하는 부속 문자열을 뽑아내기 위해서 문자열 조각을 사용한다.

```
>>> 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
>>>
```

find 메쏘드를 사용해서 찾고자 하는 문자열의 시작 위치를 명세한다. 문자열 조각낼(slicing) 때, 골뱅기 기호 뒤부터 빈 공백을 *포함하지 않는* 위치까지 문자열을 뽑아낸다.

find 메쏘드에 대한 문서는 docs.python.org/library/string.html에서 참조 가능하다.

1.11 형식 연산자

형식 연산자(format operator), %는 문자열의 일부를 변수에 저장된 값으로 바꿔 문자열을 구성한다. 정수에 포맷 연산자가 적용될 때, %는 나머지 연산가 된다. 하지만 첫 피연산자가 문자열이면, %은 포맷 연산자가 된다.

첫 피연산자는 **형식 문자열 format string**로 두번째 피 연산자가 어떤 형식으로 표현되는지를 명세하는 하나 혹은 그 이상의 **형식 열 format sequence**을 담고 있다. 결과값은 문자열이다.

예를 들어, 형식 열 '%d'의 의미는 두번째 피연산자가 정수 형식으로 표현됨을 뜻한다. (d는 "decimal"를 나타낸다.)

```
>>> camels = 42
>>> '%d' % camels
'42'
```

결과는 '42' 문자열로 정수 42와 혼동하면 안 된다.

형식 열은 문자열 어디에서도 나타날 수 있어서 문장 중간에 값을 임베드(embed)할 수 있다.

```
>>> camels = 42
>>> 'I have spotted %d camels.' % camels
'I have spotted 42 camels.'
```

만약 문자열의 형식 열이 하나 이상이라면, 두번째 인수는 튜플(tuple)이 된다. 형식 열 각각은 튜플의 요소와 순서대로 일치된다.

다음 예제는 정수 형식을 표현하기 위해서 '%d', 부동 소수점 형식을 표현하기 위해서 '%g', 문자열 형식을 표현하기 위해서 '%s'을 사용한 것을 보여준다. 여기서 왜 부동 소수점 형식이 '%f'대신에 '%g'인지는 질문하지 말아주세요.

```
>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')
'In 3 years I have spotted 0.1 camels.'
```

튜플 요소 숫자는 문자열의 형식 열의 숫자와 일치해야 하고, 요소의 형도 형식 열과 일치해야 한다.

```
>>> '%d %d %d' % (1, 2)
TypeError: not enough arguments for format string
>>> '%d' % 'dollars'
TypeError: illegal argument type for built-in operation
```

첫 예제는 충분한 요소 개수가 않돼고, 두 번째 예제는 형식이 맞지 않는다.

형식 연산자는 강력하지만, 사용하기가 여럽다. 더 많은 정보는 docs.python.org/lib/typesseq-strings.html에서 찾을 수 있다.

1.12 디버깅

프로그램을 작성하면서 배양해야 하는 기술은 항상 자신에게 질문을 하는 것이다. "여기서 무엇이 잘못 될 수 있을까?" 혹은 "사용자가 내가 작성한 완벽한 프로그램을 망가뜨리기 위해 무슨 엄청난 일을 할 것인가?"

예를 들어 앞장의 반복 while 루프를 시연하기 위해 사용한 프로그램을 살펴봅시다.

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'done':
        break
    print line

print 'Done!'
```

사용자가 입력값으로 빈 공백 줄을 입력하게 될때 무엇이 발생하는지 살펴봅시다.

```
> hello there
hello there
> # don't print this
> print this!
print this!
>
Traceback (most recent call last):
  File "copytildone.py", line 3, in <module>
    if line[0] == '#' :
```

빈 공백줄이 입력될 때까지 코드는 잘 작동합니다. 0번째 문자가 없어서 트레이스백(traceback)이 발생한다. 입력줄이 비여있을 때, 코드 3번째 줄을 "안전"하게 만드는 두가지 방법이 있다.

startswith 메쏘드를 사용해서 빈 문자열이면 거짓(False)을 반환한다.

```
    if line.startswith('#') :
```

가디언 패턴(guardian pattern)을 사용한 if문으로 문자열에 적어도 하나의 문자가 있는 경우만 두번째 논리 표현식이 평가되도록 하는 코드를 작성한다.

```
    if len(line) > 0 and line[0] == '#' :
```

1.13 용어정의

카운터(counter): 무언가를 세기 위해서 사용되는 변수로 일반적으로 0으로 초기화 되고 증가한다.

빈 문자열(empty string): 두 인용부호로 표현되는 어떤 문자도 없고 길이가 0인 문자열.

형식 연산자(format operator): 형식 문자열과 튜플을 받아, 형식 문자열에 지정된 형식으로 튜플 요소를 포함하는 문자열을 생성하는 연산자, `%`.

행식 열(format sequence): `%d`처럼 어떻게 값이 형식적으로 표현되어야 하는지를 명세하는 "형식 문자열"의 문자의 열.

형식 문자열(format string): 형식 열을 포함하는 형식 연산자와 함께 사용되는 문자열.

플래그(flag): 조건이 참인지를 표기위해 사용하는 불 변수(boolean variable)

호출(invocation): 메쏘드를 호출하는 명령문.

불변(immutable): 열의 항목에 할당할 수 없는 특성.

인덱스(index): 문자열의 문자처럼 열의 항목을 선택하기 위해 사용되는 정수 값.

항목(item): 열에 있는 값의 하나.

메쏘드(method): 개체와 연관되어 점 표기법을 사용하여 호출되는 함수.

개체(object): 변수가 참조하는 무엇. 지금에서는 "개체"와 "값"을 구별없이 사용한다.

검색(search): 찾고자 하는 것을 찾았을 때 멈추는 운행법 패턴.

열(sequence): 정돈된 세트. 즉, 정수 인텍드로 각각의 값이 확인되는 값의 집합.

조각(slice): 인텍스의 범위로 지정된 문자열 부분.

운행법(traverse): 열의 항목을 반복적으로 훑기, 각각에 대해서는 동일한 연산을 수행.

1.14 연습문제

Exercise 1.5 아래 문자열을 파이썬 코드를 작성하세요.

```
str = 'X-DSPAM-Confidence: 0.8475'
```

`find` 메쏘드와 문자열 조각내기를 사용하여 콜론(:) 문자 뒤의 문자열을 뽑아내고 `float` 함수를 사용하여 뽑아낸 문자열을 부동 소수점 숫자로 변환하세요.

Exercise 1.6 https://docs.python.org/2.7/library/stdtypes.html#string-methods에서 문자열 메쏘드 문서를 읽어보세요. 어떻게 동작하는가를 이해도를 확인하기 위해서 몇개를 골라 실험을 해보세요. strip과 replace가 특히 유용합니다.

문서는 문서는 좀 혼동스러울 수 있는 구문을 사용합니다. 예를 들어, find(sub[, start[, end]])의 꺾쇠기호는 선택(옵션) 인수를 나타냅니다. 그래서, sub은 필수지만, start은 선택 사항이고, 만약 start가 인수로 포함된다면, end는 선택이 된다.