Python Strings

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

October 5, 2011

Outline

Outline

Strings

Description

- Besides numbers, Python can also manipulate strings, which can be expressed in several ways.
  They can be enclosed in single quotes or double quotes:
- String literals can span multiple lines in several ways.
  - Continuation lines can be used, with a backslash as the last character on the line indicating that the next line is a logical continuation of the line.
  - Or, strings can be surrounded in a pair of matching triple-quotes: `"""` or `'''`. End of lines do not need to be escaped when using triple-quotes, but they will be included in the string.

Strings

Example

```
>>> 'no'
'no'
>>> "no"
'no'
>>> 'I can\'t get no'
"I can't get no"
>>> "I can't get no"
"I can't get no"
>>> '"Ha. Ha." said the clown'
'"Ha. Ha." said the clown'
>>> "\"Ha. Ha.\" said the clown"
'"Ha. Ha." said the clown'
>>> '"I can\'t get no"'
'"I can\'t get no"'
>>>
```

Strings

Example

```
>>> hello = "This is a rather long string containing\n\
... several lines of text just as you would do in C.\n\
... Note that whitespace at the beginning of the line is\
... significant."
>>> hello
'This is a rather long string containing\nseveral lines of text just as you would
do in C.\nNote that whitespace at the beginning of the line issignificant.'
>>> print hello
This is a rather long string containing
several lines of text just as you would do in C.
Note that whitespace at the beginning of the line issignificant.
>>>
```

Strings

Example

```
>>> print """
... Usage: thingy [OPTIONS]
...      -h                        Display this usage message
...      -H hostname               Hostname to connect to
... """
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
>>> print '''
... Usage: thingy [OPTIONS]
...      -h                        Display this usage message
...      -H hostname               Hostname to connect to
... '''
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
>>>
```

Outline

Concatenating and repeating strings

Example

```
>>> # Strings can be concatenated (glued together) with the + operator
...
>>> "str" + "ing"
'string'
>>> 'str' + 'ing'
'string'
>>> 'concat' + 'ening' + ' ' + 'str' + 'ings'
'concatening strings'
>>>
>>> # Strings can be repeated with *
...
>>> "bla" * 5
'blablablablabla'
>>> s = "bla"
>>> '<' + s*5 + '>'
'<blablablablabla>'
>>>
```

Subscript

Example

```
>>> # Strings can be subscripted; the first character of a string has subscript 0
...
>>> s = "abcdef"
>>> s[0]
'a'
>>> len(s)
6
>>> s[len(s)-1]
'f'
>>>
>>> # A character is simply a string of size one
...
>>> len("a")
1
>>> "a"[0]
'a'
>>>
```

Slice

Example

```
>>> s = "abcdefgh"
>>> s[:]
'abcdefgh'
>>> s[3:]
'defgh'
>>> s[:5]
'abcde'
>>> s[3:5]
'de'
>>> s[3:5] = "x"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> s[-1]
'h'
>>> s[2:-2]
'cdef'
>>>
```

Non-mutable objects

Example

```
>>> s
'Computer science is no more about computers than astronomy is about telescopes.'
>>> s[0] = 'X'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> s[9:16]
'science'
>>> s[9:16] = 'COMPUTER'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> t = s[:9] + 'COMPUTER' + s[16:]
>>> t
'Computer COMPUTER is no more about computers than astronomy is about telescopes.'
>>>
```

Occurrences and equalities

Example

```
>>> s = "abcdef"
>>> "a" in s
True
>>> "f" in s
True
>>> "h" in s
False
>>> "" in s
True
>>> "ab" in s
True
>>> "dc" in s
False
>>> t = "abcdef"
>>> s == t
True
>>> s == t[:-1] + "f"
True
>>>
```

Outline

Some methods

Description

- `string.count(sub[, start[, end]])`
  Return the number of occurrences of substring `sub` in string `string[start:end]`. Optional arguments `start` and `end` are interpreted as in slice notation.
- `string.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.
- `string.upper()`
  Return a copy of the string converted to uppercase.

Methods

Example

```
>>> s = "Computer science is no more about computers than \
... astronomy is about telescopes."
>>> s
'Computer science is no more about computers than astronomy is about telescopes.'
>>> s.upper()
'COMPUTER SCIENCE IS NO MORE ABOUT COMPUTERS THAN ASTRONOMY IS ABOUT TELESCOPES.'
>>> s.count("e")
8
>>> s.replace("e", "E")
'ComputEr sciEncE is no morE about computErs than astronomy is about tElEscopEs.'
>>> s[9:16], s.count("e", 9, 16)
('science', 2)
>>> s.replace('e', 'E', 4)
'ComputEr sciEncE is no morE about computers than astronomy is about telescopes.'
>>> s.count("te")
3
>>> s.replace("te", "TE")
'CompuTEr science is no more about compuTErs than astronomy is about TElescopes.'
>>>
```

Some methods

Description

- `string.endswith(suffix[, start[, end]])`
  Return True if the string ends with the specified suffix, otherwise return False. `suffix` can also be a tuple of suffixes to look for. With optional `start`, test beginning at that position. With optional `end`, stop comparing at that position.
- `string.find(sub[, start[, end]])`
  Return the lowest index in the string where substring `sub` is found, such that `sub` is contained in the range `[start, end]`. Optional arguments start and end are interpreted as in slice notation. Return -1 if sub is not found.
- `string.index(sub[, start[, end]])`
  Like `find()`, but raise `ValueError` when the substring is not found.

Methods

Example

```
>>> s
'Computer science is no more about computers than astronomy is about telescopes.'
>>> s.endswith('scopes')
False
>>> s.endswith('scopes.')
True
>>> s.endswith(('.', ',', '!'))
True
>>>>>> s.find("te")  # equivalent to s.index("te")
5
>>> s[34:]
'computers than astronomy is about telescopes.'
>>> s.find("te", 34)  # equivalent to s.index("te", 34)
39
>>> s[68:-1]
'telescopes'
>>> s.find("te", 68, -1)  # equivalent to s.index("te", 68, -1)
68
>>>
```

Methods

Example

```
>>> s
'Computer science is no more about computers than astronomy is about telescopes.'
>>> s.find("me")
-1
>>> # Indeed
... s.count("me")
0
>>> s.index("me")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: substring not found
>>>
```

Some methods

Description

- `string.islower()`
  Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.
- `string.isupper()`
  Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise.
- `string.isspace()`
  Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.
- `string.join(seq)`
  Return a string which is the concatenation of the strings in the sequence seq. The separator between elements is the string providing this method.

Methods

Example

```
>>> s = 'abcd'
>>> s.isupper()
False
>>> s.islower()
True
>>> s.upper()
'ABCD'
>>> s.upper().isupper()
True
>>> s.upper().lower().isupper()
False
>>> s.upper().lower().islower()
True
>>> s.isspace()
False
>>> "     ".isspace()
True
>>>
```

Methods

Example

```
>>> t = ["ab", "cde", "fghi"]
>>> s = "--"
>>> s.join(t)
'ab--cde--fghi'
>>> "--".join(t)
'ab--cde--fghi'
>>> "--".join(["ab", "cde", "fghi"])
'ab--cde--fghi'
>>> "".join(["ab", "cde", "fghi"])
'abcdefghi'
>>> "a long separator".join(["ab", "cde", "fghi"])
'aba long separatorcdea long separatorfghi'
>>>
```