Strings in Python

Text values are perhaps more fundamental to computer science than even numbers. As a case in point, Python programs are written and stored as text. The native data type for text in Python is called a string, and corresponds to the constructor `str`.

There are many details of how strings are represented, expressed, and manipulated in Python. Strings are another example of a rich abstraction, one which requires a substantial commitment on the part of the programmer to master. This section serves as a condensed introduction to essential string behaviors.

String literals can express arbitrary text, surrounded by either single or double quotation marks.

```
>>> 'I am string!'
'I am string!'
>>> "I've got an apostrophe"
"I've got an apostrophe"
>>> '您好'
'您好'
```

We have seen strings already in our code, as docstrings, in calls to `print`, and as error messages in `assert` statements.

Strings satisfy the two basic conditions of a sequence that we introduced at the beginning of this section: they have a length and they support element selection.

```
>>> city = 'Berkeley'
>>> len(city)
8
>>> city[3]
'k'
```

The elements of a string are themselves strings that have only a single character. A character is any single letter of the alphabet, punctuation mark, or other symbol. Unlike

many other programming languages, Python does not have a separate character type; any text is a string, and strings that represent single characters have a length of 1.

Like tuples, strings can also be combined via addition and multiplication.

```
>>> 'Berkeley' + ', CA'
'Berkeley, CA'
>>> 'Shabu ' * 2
'Shabu Shabu '
```

Membership. The behavior of strings diverges from other sequence types in Python. The string abstraction does not conform to the full sequence abstraction that we described for tuples and ranges. In particular, the membership operator `in` applies to strings, but has an entirely different behavior than when it is applied to sequences. It matches substrings rather than elements.

```
>>> 'here' in "Where's Waldo?"
True
```

Likewise, the `count` and `index` methods on strings take substrings as arguments, rather than single-character elements. The behavior of `count` is particularly nuanced; it counts the number of non-overlapping occurrences of a substring in a string.

```
>>> 'Mississippi'.count('i')
4
>>> 'Mississippi'.count('issi')
1
```

Multiline Literals. Strings aren't limited to a single line. Triple quotes delimit string literals that span multiple lines. We have used this triple quoting extensively already for docstrings.

```
>>> """The Zen of Python
claims, Readability counts.
Read more: import this."""
'The Zen of Python\nclaims, "Readability counts."\nRead
more: import this.'
```

In the printed result above, the `\n` (pronounced "*backslash en*") is a single element that represents a new line. Although it appears as two characters (backslash and "n"), it is considered a single character for the purposes of length and element selection.

String Coercion. A string can be created from any object in Python by calling the `str` constructor function with an object value as its argument. This feature of strings is useful for constructing descriptive strings from objects of various types.

```
>>> str(2) + ' is an element of ' + str(digits)
'2 is an element of (1, 8, 2, 8)'
```

The mechanism by which a single `str` function can apply to any type of argument and return an appropriate value is the subject of the later section on generic functions.

Methods. The behavior of strings in Python is extremely productive because of a rich set of methods for returning string variants and searching for contents. A few of these methods are introduced below by example.

```
>>> '1234'.isnumeric()
True
>>> 'rOBERT dE nIRO'.swapcase()
'Robert De Niro'
>>> 'snakeyes'.upper().endswith('YES')
True
```

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/objects.html#strings