

Scientific Computing 272

Section 2: Strings in Python

Last updated: 14 March 2019



UNIVERSITEIT
STELLENBOSCH
UNIVERSITY

Willem Bester

Section Outline

Basic Properties

Escape Sequences

Multiline Strings

Print

Formatted Printing

User Input



Strings

- ▶ We don't only do number crunching with computers
- ▶ What about the following?
 - ▶ Audiovisual data, such as photos and music
 - ▶ Textual data in student records, such as addresses
 - ▶ DNA analysis
- ▶ These days, computers spend a lot of time processing text
- ▶ In Python, we represent a piece of text as a **string**
- ▶ A string is a sequence of **characters**—letters, digits, and other symbols

Strings in Formal Language Theory

Definition (Alphabet, Symbols, and Strings)

An **alphabet** is a non-empty finite set. The members of an alphabet are called its **symbols**. A **string over an alphabet** is a finite sequence of symbols from that alphabet.

- ▶ In Python, there are two data types (that is, alphabets) for sequences of characters
- ▶ `str` can store characters from the Latin alphabet, found on most keyboards
- ▶ `unicode` can store almost any character, from Chinese ideograms to Klingon

Strings in Python

In Python, **delimit** (surround) a string with single or double quotes; note that the quotes must match

Example (String Delimiting)

```
>>> "Isaac Newton"
'Isaac Newton'
>>> 'Charles Darwin'
'Charles Darwin'
>>> 'Albert Einstein'
      File "<stdin>", line 1
        'Albert Einstein'
              ^
SyntaxError: EOL while scanning string literal
```

String Concatenation

We may join two string literals by putting them side by side, that is, by **juxtaposing** them.

Example (String Concatenation by Juxtaposition)

```
>>> 'Albert' 'Einstein'
'AlbertEinstein'
>>> 'Albert ' 'Einstein'
'Albert Einstein'
>>> 'Albert' " Einstein"
'Albert Einstein'
```

Note that spaces must be included explicitly: Only the symbols between a pair of quotes become part of the string.

String Concatenation

We may also join strings with the `+` operator; for the sake of clarity, this is the preferred way.

Example (Explicit String Concatenation)

```
>>> 'Alan' + ' Turing'
'Alan Turing'
>>> "Grace Hopper" + ""
'Grace Hopper'
>>> '' + 'Dennis Ritchie'
'Dennis Ritchie'
```

The **empty string** has length 0, and it contains no characters at all.

String Algebra

The set of all strings over a particular alphabet together with the concatenation operation form a **monoid**¹:

- ▶ Concatenation of two strings yields another string
- ▶ String concatenation is **associative**
- ▶ The empty string is the **identity element**: Concatenating it to any other string leaves this other string unchanged

Example (Associativity)

```
>>> ('Science' + ' is ') + 'cool!'
'Science is cool!'
>>> 'Science' + (' is ' + 'cool!')
'Science is cool!'
```

¹For the mathematically inclined: A **monoid** is a set S that is **closed** under an **associative binary operation** \odot and has an **identity element** $e \in S$ such that for all $a \in S$ we have $a \odot e = e \odot a = a$.

Operator Overloading

Operator Overloading

When the operation performed by an operator differs, depending on the type of its operand, it is called **operator overloading**.

- ▶ `+` is overloaded
- ▶ `+` performs concatenation on string arguments
- ▶ But `+` performs addition on numeric arguments
- ▶ What if we combine string and numeric arguments?

Operator Overloading

Example

```
>>> "Scientific Computing " + 272
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> 8 + ' planets'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In the above example, a human might easily see what is required, but what about the following?

Example (Concatenation of Different Types)

```
>>> '123' + 4
```

Strong and Dynamic Typing in Python

Python is **strongly typed**: The interpreter keeps track of variable and value types, and it restricts how values of different types can be intermingled. Therefore, it is impossible to “add” a number to a string. Python forces you to be explicit when combining values of different types.

Example (Combining Values of Different Types)

```
>>> int('123') + 4
127
>>> '123' + str(4)
'1234'
```

Python is still **dynamically typed**. For example, after the assignment `x = 'Hello'`, the variable `x` is automatically of type `str`.

String Repetition

- ▶ Other operators can combine string and numeric operands
- ▶ We can repeat a string with `*`

Example (String Repetition)

```
>>> 'AT' * 7
'ATATATATATATAT'
>>> 4 * "_*"
'_*_*_*_*_'
>>> 'GC' * 0
''
>>> 'TATATA' * -3
''
```

Note that repeating with an integer of less than 1 yields the empty string.

Quotes inside Strings

What if we want to put a single quote inside a string?

Example (Quotes inside Strings)

```
>>> 'That's not going to work.'  
File "<stdin>", line 1  
    'That's not going to work.'  
      ^  
SyntaxError: invalid syntax  
>>> "That's better."  
"That's better."  
>>> '"That' + "'" + 's hard to read," she said.'  
'"That\'s hard to read," she said.'
```

We can delimit with single quotes when we include double quotes, and vice versa.

Escape Sequences

Example (Quote Escape Sequence)

```
>>> '"That' + "'" + 's hard to read," she said.'  
'"That\'s hard to read," she said.'
```

- ▶ Consider the result of the previous expression
- ▶ The combination of backslash and another character is called an **escape sequence**
- ▶ Hereby, we “escape” the normal syntax rules of Python
- ▶ We can use special characters in strings without too much song and dance

Escape Sequences

Table: Common escape sequences

Escape sequence	Description
<code>\n</code>	End-of-line
<code>\\</code>	Backslash
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\t</code>	Tab

- ▶ The backslash is the **escape character** and signals the start of an escape sequence
- ▶ The escape character means that the following character represents something special

Single-Line Strings

Strings delimited by single or double quotes must fit on a single line.

Example

```
>>> 'one
      File "<stdin>", line 1
        'one
          ^
SyntaxError: EOL while scanning string literal
```

- ▶ “EOL” stands for “end of line”
- ▶ Python complains that it reached the end of the line before it found the end of the string

Multiline Strings

To span multiple lines, delimit a string with three double quotes or three single quotes.

Example (Multiline Strings)

```
>>> '''one  
... two  
... three'''  
'one\ntwo\nthree'
```

Note that Python uses the escape sequence `\n` to indicate where we started a new line in the input.

Multiline Strings

Example (Newlines in Multiline Strings)

```
>>> """A
... B
... C
... """
'A\nB\nC\n'
```

- ▶ Note that every ⟨Enter⟩ key press results in a **newline** escape sequence
- ▶ Different OSes use different sets of characters for newline, but Python uses the Linux convention of `\n`

The Print Function

- ▶ So far, we've only displayed the value of one variable or expression at a time
- ▶ The `print` function is more powerful

Example (The `print` Function)

```
>>> print(3 * 7)
21
>>> print("The Latin 'rattus norvegicus' means 'Norwegian rat'.")
The Latin 'rattus norvegicus' means 'Norwegian rat'.
```

- ▶ The first statement produced what we expect for numbers
- ▶ But the second one stripped the quotes surrounding the string

The Print Function

- ▶ `print` interprets escape sequences
- ▶ It takes a list of comma-separated items to display
- ▶ We may mix types in the list
- ▶ Python always inserts a single space between each value

Example (Arguments to print)

```
>>> print('one\t\ttwo\nthree\tfour')
one      two
three    four
>>> area = 3.1415927 * 5 * 5
>>> print("The area of the circle is", area, "sq mm.")
The area of the circle is 78.5398175 sq mm.
```

Formatted Printing

We may specify the output format with a **format string**.

Example (Format String)

```
>>> area = 3.1415927 * 5 * 5
>>> print('The area of the circle is {0} sq.mm'.format(area))
The area of the circle is 78.5398175 sq.mm
>>> print('{0} and {1}'.format('bread', 'butter'))
bread and butter
>>> print('{1} and {0}'.format('bread', 'butter'))
butter and bread
```

The braces and characters within them are called **format fields**:
They are replaced with the values (or objects) passed into the **format** method.

Formatted Printing

- ▶ The format string may also specify alignment, width, precision, and some other aspects
- ▶ Python Tutorial, §7.1.3:
<https://docs.python.org/3.7/library/string.html#format-string-syntax>

Example (Format String with More Specifiers)

```
>>> planck = 6.62606957e-34
>>> planck
6.62606957e-34
>>> print("5 decs: {0:.5e}; 4 decs: {0:.4e}".format(planck))
5 decs: 6.62607e-34; 4 decs: 6.6261e-34
>>> print("'{0:>10.3f}' '{0:<10.3f}'".format(1 / 3))
'      0.333'0.333'
```

User Input

- ▶ `input` reads a single line of text from the keyboard
- ▶ A string is returned when the user enters something, even if the input looks like a number

Example (Raw Input)

```
>>> line = input()
Fermat's Last Theorem
>>> print(line)
Fermat's Last Theorem
>>> line = input()
317
>>> print(line * 2)
317317
```

Converting User Input to Numbers

- ▶ Use the functions `int` or `float` to convert an input string to the required type
- ▶ Note that we may “wrap” `input` in another function

Example (Input Function Wrapping)

```
>>> value = input()
317
>>> value = int(value)
>>> print(value * 2)
634
>>> value = float(input())
17.7
>>> print(value / 3)
5.9
```


Conversion Errors

Example (Conversion Errors)

```
>>> value = float(input())
Fermat
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'Fermat'
```

- ▶ The **literal** refers to the string that was entered
- ▶ The error means that Python could not convert the input string to type `float`

Prompting the User for Input

`input` takes an optional string argument, which may be used to prompt the user for input.

Example (Prompting for Input)

```
>>> name = input("Please enter your name: ")
Please enter your name: Gauss
>>> print(name)
Gauss
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 13
>>> print(x ** 7)
62748517
```