String Methods

Table of contents

See csse1001[1] for course logistics — this note covers Lecture 6A's technical content. See python-string-methods[2] for the full reference on the string methods covered here.

Today's outline

- Built-in string methods and `help()`
- Method invocation syntax
- Exercises: `title`, `center`
- Splitting and joining strings
- Sanitizing data with `strip`

[1] courses/csse1001/csse1001.pdf
[2] courses/csse1001/python-string-methods.pdf

Why learn string methods?

During file processing (next lecture) we'll be working extensively with strings. We're already able to replicate the functionality of any string method with our current tools — but it's worth striving to implement any string method you use at least once.

Built-in string methods

Python's `str` type has a myriad of built-in methods. Review them via `help(str)`:

```
>>> help(str)
...
| capitalize(self, /)
|     Return a capitalized version of the string.
|
| casefold(self, /)
|     Return a version of the string suitable for
|     caseless comparisons.
```

(Type Enter to scroll, `q` to quit help.)

Methods

Strings are **objects** — we'll eventually learn what that means fully. Objects have **methods**, which are like functions but invoked differently. We don't say:

```
>>> capitalize("hello")
NameError: name 'capitalize' is not defined
```

but rather:

```
>>> "hello".capitalize()   # note the ()
'Hello'
```

For information on a particular method:

```
>>> help(str.find)
find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end]. Optional
    arguments start and end are interpreted as with slice notation.

    Return -1 on failure.
```

Interpreting `help`

Square brackets in a signature like `find(sub[, start[, end]])` indicate an *optional* parameter — and this rule recurses (an optional parameter can itself have optional parameters). The valid ways to call `find` are:

```
"team".find("I")
"team".find("I", 1)
"team".find("I", 1, -1)
```

whereas `find(1, -1)` is not allowed, since `sub` is required.

```
>>> "hello world".find("world", 6)
6
>>> "hello hello".find("hello")
0
>>> "hello hello".find("hello", 1)
6
>>> "hello hello".find("hello", 1, 3)
-1
```

Exercise: title case

Convert a string of text into title format:

```
def title_case(cs: str) -> str:
    """ Convert cs to title case.
    >>> title_case("a tale of two cities")
    'A Tale Of Two Cities'
    """
```

This can be accomplished by invoking the appropriate string method correctly:

```
>>> cs = "a tale of two cities"
>>> cs.title()
'A Tale Of Two Cities'
>>> "a tale of two cities".title()
'A Tale Of Two Cities'
```

Exercise: centering text

Write a function that, given a word and an integer width, centers the word in a sequence of xs:

```
def foo(word: str, width: int) -> str:
    """
    >>> foo('spam', 10)
    'xxxspamxxx'
    >>> foo('101', 20)
    'xxxxxxxx101xxxxxxxxx'
    >>> foo('UQQU', 30)
    'xxxxxxxxxxxxxUQQUxxxxxxxxxxxxx'
    """


>>> help(str.center)
center(self, width, fillchar=' ', /)
    Return a centered string of length width.

    Padding is done using the specified fill character
    (default is a space).

>>> 'spam'.center(10, 'x')
'xxxspamxxx'
>>> '101'.center(20, 'x')
'xxxxxxxx101xxxxxxxxx'
```

Splitting strings

```
>>> "a b c".split()          # default: split at whitespace
['a', 'b', 'c']
>>> "axbxc".split()
```

```
['axbxc']
>>> "axbxc".split('x')
['a', 'b', 'c']
>>> "axbxc".split('bx')
['ax', 'c']
>>> "a, b, c".split(',')     # useful for CSV processing
['a', ' b', ' c']
>>> "a, b, c".split(', ')    # removes leading/trailing spaces
['a', 'b', 'c']
```

Joining strings

```
>>> ",".join(["A", "B", "C"])
'A,B,C'
>>> "".join(["A", "B", "C"])
'ABC'
>>> ", ".join(["A", "B", "C"])
'A, B, C'
>>> "xxx".join(["A", "B", "C"])
'AxxxBxxxC'
```

Sanitizing data

```
>>> help(str.strip)
strip(self, chars=None, /)
    Return a copy of the string with leading and trailing
    whitespace removed.

    If chars is given and not None, remove characters in
    chars instead.

>>> "   123   \n".strip()   # a newline is considered whitespace
'123'
```

Summary

Sometimes datatypes come with extra functionality by way of methods. In particular, there are many string methods that will aid with text processing.

Next: 2025-09-01-file-io[3] (Lecture 6B).

[3] courses/csse1001/2025-09-01-file-io.pdf