**Strings and Character Data in Python**

>>> s = 'foo'

>>> t = 'bar'

>>> u = 'baz'

>>> s + t

'foobar'

>>> s + t + u

'foobarbaz'

>>> print('Go team' + '!!!')

Go team!!!

>>> s = 'foo.'

>>> s \* 4

'foo.foo.foo.foo.'

>>> 4 \* s

'foo.foo.foo.foo.'

>>> s = 'foo'

>>> s in 'That\'s food for thought.'

True

>>> s in 'That\'s good for now.'

False

>>> 'z' not in 'abc'

True

>>> 'z' not in 'xyz'

False

#### **ord(c)**

Returns an integer value for the given character.

>>> ord('a')

97

>>> ord('#')

35

#### **chr(n)**

Returns a character value for the given integer.

>>> chr(97)

'a'

>>> chr(35)

'#'

For any string s and any integer n (0 ≤ n ≤ len(s)), s[:n] + s[n:] will be equal to s:

>>> s = 'foobar'

>>> s[:4] + s[4:]

'foobar'

>>> s[:4] + s[4:] == s

True

**The [:] syntax works for lists. However, there is an important difference between how this operation works with a list and how it works with a string.**

>>> s = 'foobar'

>>> s[:]

'foobar'

>>> s[:] is s

True

If s is a string, s[:] returns a reference to the same object:

>>> s = 'foobar'

>>> s[:]

'foobar'

>>> s[:] is s

True

Conversely, if a is a list, a[:] returns a new object that is a copy of a:

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[:]

['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[:] is a

False

### **Interpolating Variables Into a String**

In Python version 3.6, a new string formatting mechanism was introduced. This feature is formally named the Formatted String Literal, but is more usually referred to by its nickname **f-string**.

>>> n = 20

>>> m = 25

>>> prod = n \* m

>>> print('The product of', n, 'and', m, 'is', prod)

The product of 20 and 25 is 500

>>> n = 20

>>> m = 25

>>> prod = n \* m

>>> print(f'The product of {n} and {m} is {prod}')

The product of 20 and 25 is 500

Any of Python’s three quoting mechanisms can be used to define an f-string:

>>> var = 'Bark'

>>> print(f'A dog says {var}!')

A dog says Bark!

>>> print(f"A dog says {var}!")

A dog says Bark!

>>> print(f'''A dog says {var}!''')

A dog says Bark!

>>> s = 'foO BaR BAZ quX'

>>> s.capitalize()

'Foo bar baz qux'

>>> 'FOO Bar 123 baz qUX'.lower()

'foo bar 123 baz qux'

>>> 'FOO Bar 123 baz qUX'.swapcase()

'foo bAR 123 BAZ Qux'

>>> 'the sun also rises'.title()

'The Sun Also Rises'

>>> 'foo goo moo'.count('oo')

3

s.endswith(<suffix>) returns True if s ends with the specified <suffix> and False otherwise:

>>> 'foobar'.endswith('bar')

True

>>> 'foobar'.endswith('baz')

False

>>> 'foo bar foo baz foo qux'.find('foo')

0

This method returns -1 if the specified substring is not found:

>>> 'foo bar foo baz foo qux'.find('grault')

-1

s.rfind(<sub>) returns the highest index in s where substring <sub> is found:

>>> 'foo bar foo baz foo qux'.rfind('foo')

16

As with .find(), if the substring is not found, -1 is returned:

>>> 'foo bar foo baz foo qux'.rfind('grault')

-1

>>> 'abc123'.isalnum()

True

>>> 'abc$123'.isalnum()

False

>>> ''.isalnum()

False

>>> 'ABCabc'.isalpha()

True

>>> 'abc123'.isalpha()

False

>>> '123'.isdigit()

True

>>> '123abc'.isdigit()

False

The most commonly encountered whitespace characters are space ' ', tab '\t', and newline '\n':

>>> ' \t \n '.isspace()

True

>>> ' a '.isspace()

False

s.istitle() returns True if s is nonempty, the first alphabetic character of each word is uppercase, and all other alphabetic characters in each word are lowercase. It returns False otherwise:

>>> 'This Is A Title'.istitle()

True

>>> 'This is a title'.istitle()

False

>>> 'Give Me The #$#@ Ball!'.istitle()

True

s.isupper()

>>> 'ABC'.isupper()

True

>>> 'ABC1$D'.isupper()

True

>>> 'Abc1$D'.isupper()

False

s.center(<width>) returns a string consisting of s centered in a field of width <width>. By default, padding consists of the ASCII space character:

>>> 'foo'.center(10)

' foo '

>>> 'bar'.center(10, '-')

'---bar----'

>>> 'foo bar foo baz foo qux'.replace('foo', 'grault')

'grault bar grault baz grault qux'

>>> 'foo bar foo baz foo qux'.replace('foo', 'grault', 2)

'grault bar grault baz foo qux'

s.strip() is essentially equivalent to invoking s.lstrip() and s.rstrip() in succession

>>> 'www.realpython.com'.strip('w.moc')

'realpython'

s.join(<iterable>)

s.join(<iterable>) returns the string that results from concatenating the objects in <iterable> separated by s.

>>> ', '.join(['foo', 'bar', 'baz', 'qux'])

'foo, bar, baz, qux'

>>> list('corge')

['c', 'o', 'r', 'g', 'e']

>>> ':'.join('corge')

'c:o:r:g:e'

s.partition(<sep>)

Divides a string based on a separator.

>>> 'foo.bar'.partition('.')

('foo', '.', 'bar')

>>> 'foo@@bar@@baz'.partition('@@')

('foo', '@@', 'bar@@baz')

s.split(sep=None, maxsplit=-1)

Splits a string into a list of substrings.

>>> 'www.realpython.com'.split('.', maxsplit=1)

['www', 'realpython.com']

>>> 'www.realpython.com'.rsplit('.', maxsplit=1)

['www.realpython', 'com']