Converting File Input

- As with the input function, the readline() method can only return strings
- If the file contains numerical data, the strings must be converted to the numerical value using the int() or float() function:

```
value = float(line)
```

- The newline character at the end of the line is ignored when the string is converted to a numerical value

Writing To A File

- For example, we can write the string "Hello, World!" to our output file using the statement:

```
outfile.write("Hello, World!\n")
```

  - Unlike print() when writing text to an output file, *you must explicitly write the newline character to start a new line*

- You can also write formatted strings to a file with the write method:

```
outfile.write("Number of entries: %d\nTotal: %8.2f\n"
  % (count, total))
```

Example: File Reading/Writing

- Suppose you are given a text file that contains a sequence of floating-point values, stored one value per line
- You need to read the values and write them to a new output file, aligned in a column and followed by their total and average value
- If the input file has the contents

```
32.0
54.0
67.5
80.25
115.0
```

Example: File Reading/Writing (2)

- The output file will contain

```
32.00
54.00
67.50
80.25
115.00
---------
Total: 348.75
Average: 69.75
```

Example One

- Open the file total.py

Common Error

- Backslashes in File Names
  - When using a String literal for a file name with path information, you need to supply each backslash twice:

```
infile = open("c:\\homework\\input.txt", "r")
```

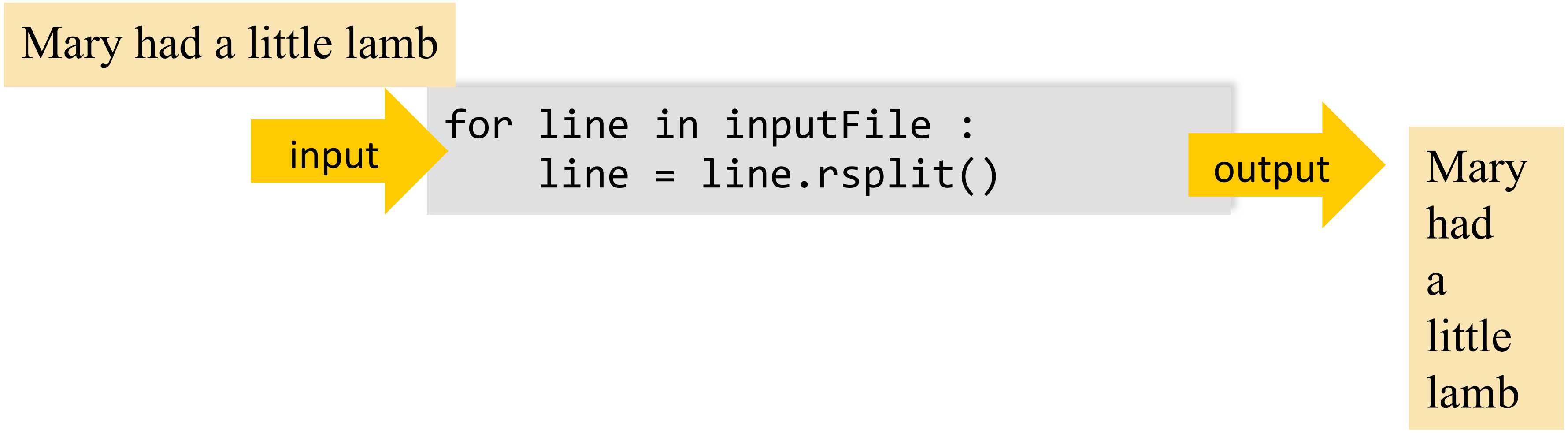
  - A single backslash inside a quoted string is the *escape character*, which means the next character is interpreted differently (for example, '\n' for a newline character)
  - When a user supplies a filename into a program, the user should not type the backslash twice

Text Input and Output

SECTION 7.2

Text Input and Output

- In the following sections, you will learn how to process text with complex contents, and you will learn how to cope with challenges that often occur with real data
- Reading Words Example:



```
for line in inputFile :
    line = line.rsplit()
```

Processing Text Input

- There are times when you want to read input by:
  - Each word
  - Each line
  - A single character
- Python provides methods such: read(), split() and strip() for these tasks

Processing text input is required for almost all types of programs that interact with the user

Text Input and Output

- Python can treat an input file as though it were a container of strings in which each line comprises an individual string
- For example, the following loop reads all lines from a file and prints them:

```
for line in infile :
    print(line)
```

  - At the beginning of each iteration, the loop variable line is assigned the value of a string that contains the next line of text in the file
- There is a critical difference between a file and a container:
  - Once you read the file you must close it before you can iterate over it again

An Example of Reading a File

- We have a file that contains a collection of words; one per line:

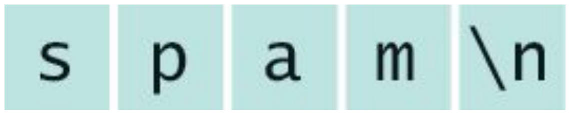
```
spam

and

eggs
```

Removing The Newline (1)

- Recall that each input line ends with a newline (\n) character
- Generally, the newline character must be removed before the input string is used
- When the first line of the text file is read, the string line contains

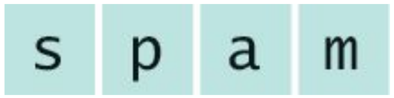


Removing The Newline (2)

- To remove the newline character, apply the rstrip() method to the string:

```
line = line.rstrip()
```

- This results in the string:



Character Strip Methods

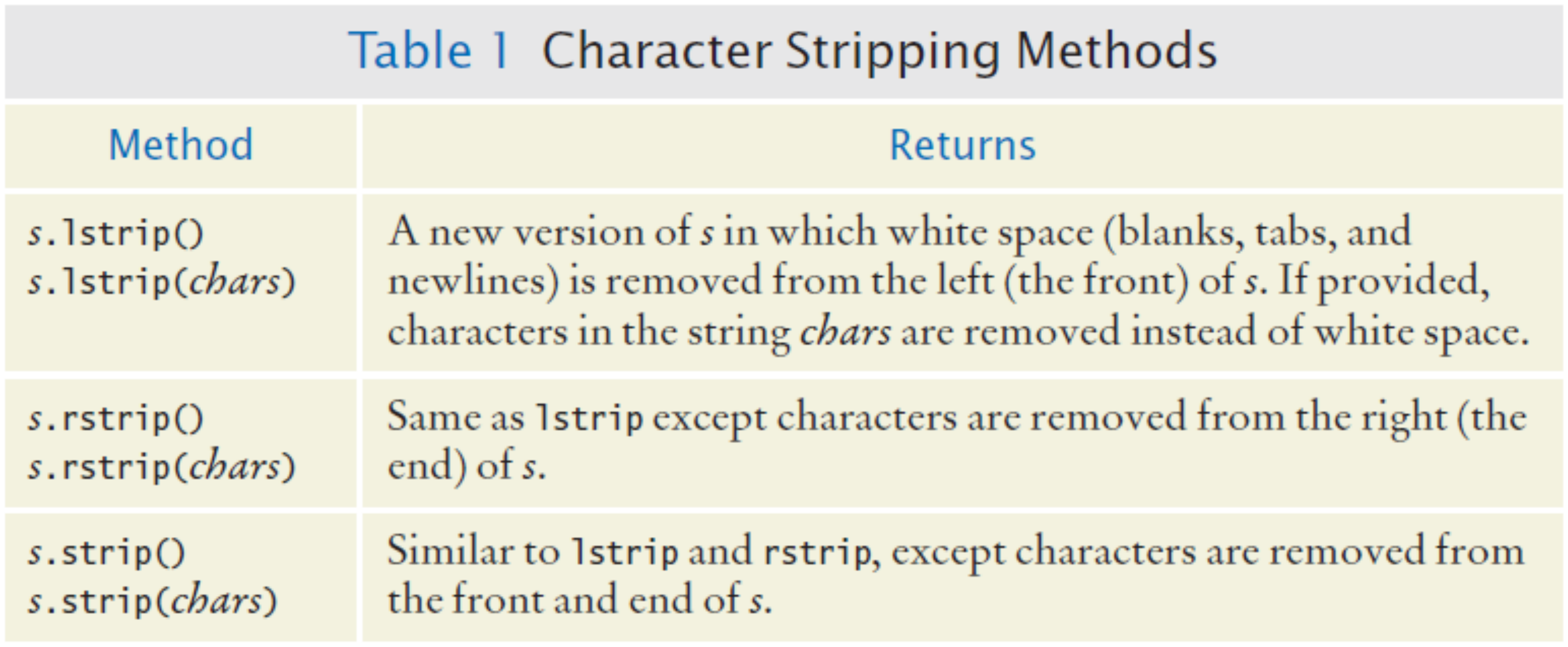


Table 1 Character Stripping Methods

Method	Returns
s.lstrip() *s*.lstrip(*chars*)	A new version of *s* in which white space (blanks, tabs, and newlines) is removed from the left (the front) of *s*. If provided, characters in the string *chars* are removed instead of white space.
s.rstrip() *s*.rstrip(*chars*)	Same as lstrip except characters are removed from the right (the end) of *s*.
s.strip() *s*.strip(*chars*)	Similar to lstrip and rstrip, except characters are removed from the front and end of *s*.

Character Strip Examples

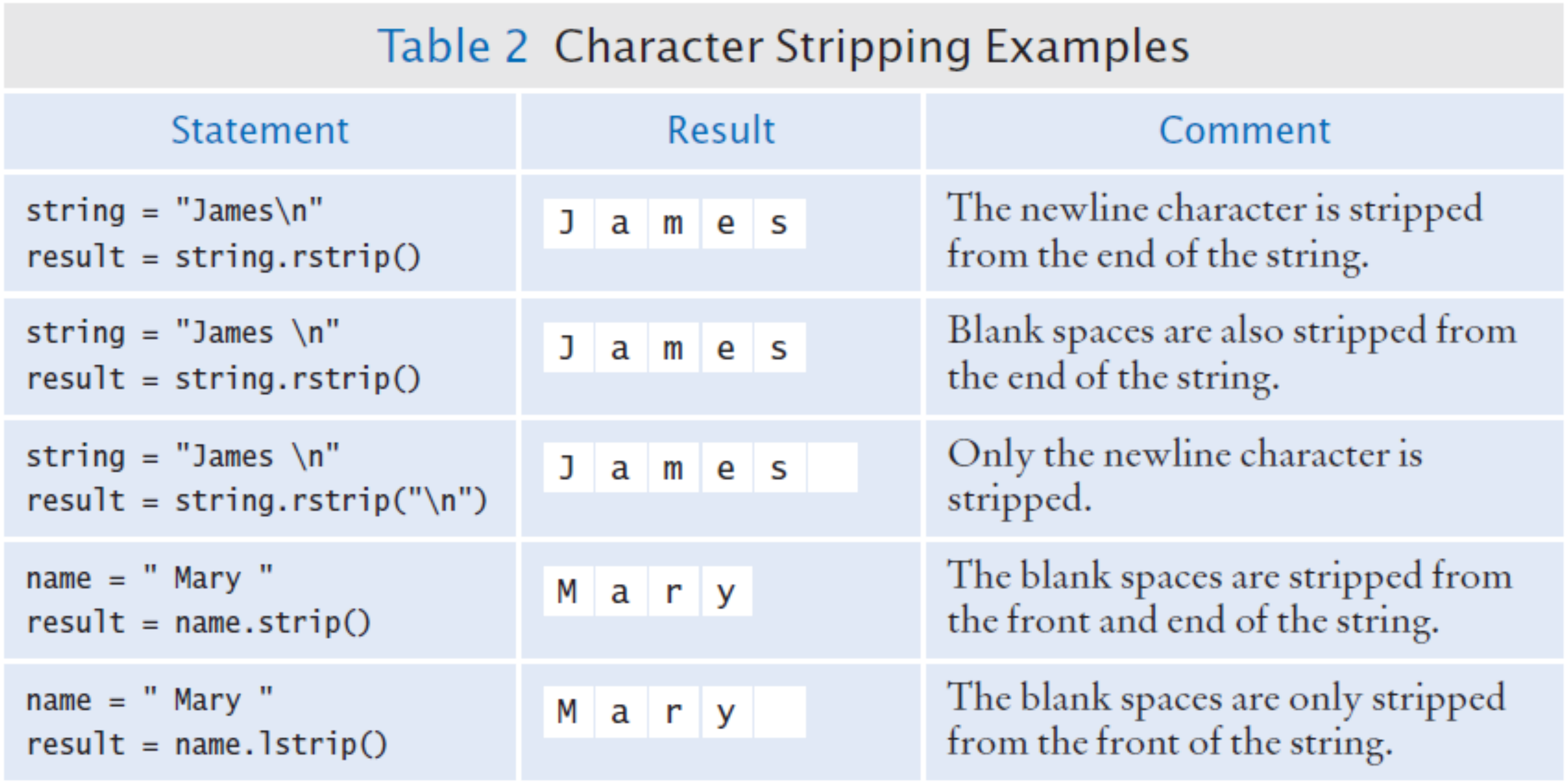


Table 2 Character Stripping Examples

Statement	Result	Comment

`string = "James\n"`
`result = string.rstrip()`
	`J a m e s`	The newline character is stripped from the end of the string.

`string = "James \n"`
`result = string.rstrip()`
	`J a m e s`	Blank spaces are also stripped from the end of the string.

`string = "James \n"`
`result = string.rstrip("\n")`
	`J a m e s  `	Only the newline character is stripped.

`name = " Mary "`
`result = name.strip()`
	`M a r y`	The blank spaces are stripped from the front and end of the string.

`name = " Mary "`
`result = name.lstrip()`
	`M a r y  `	The blank spaces are only stripped from the front of the string.

Reading Words

- Sometimes you may need to read the individual words from a text file
- For example, suppose our input file contains two lines of text

  Mary had a little lamb,

  whose fleece was white as snow

Reading Words (2)

- We would like to print to the terminal, one word per line

  Mary

  had

  a

  little

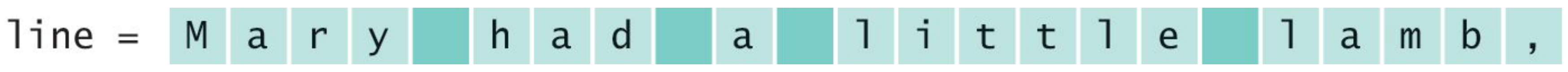
  . . .

- Because there is no method for reading a word from a file, you must first read a line and then split it into individual words

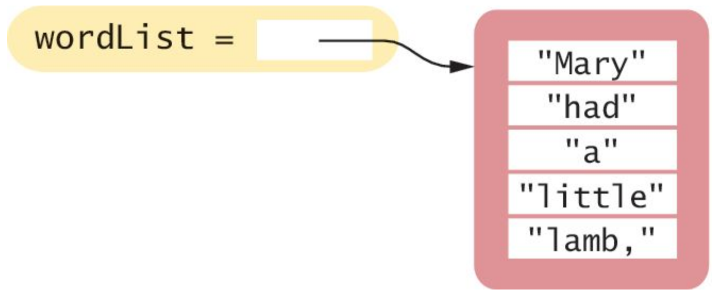
```
line = line.rstrip()
wordlist = line.split()
```

Reading Words (3)

- The split method returns the list of substrings that results from splitting the string at each blank space
- For example, if line contains the string:



- It will be split into 5 substrings that are stored in a list in the same order in which they occur in the string:



Reading Words (4)

- Notice that the last word in the line contains a comma
- If we only want to print the words contained in the file without punctuation marks, we can strip those from the substrings using the rstrip() method introduced in the previous section:

```
word = word.rstrip(".,?!")
```

Reading Words: Complete Example

```
inputFile = open("lyrics.txt", "r")
for line in inputFile :
    line = line.rstrip()
    wordList = line.split()
    for word in wordList :
        word = word.rstrip(".,?!")
        print(word)

inputFile.close()
```

Example Two

- Open the file lyrics.py

Additional String Splitting Methods

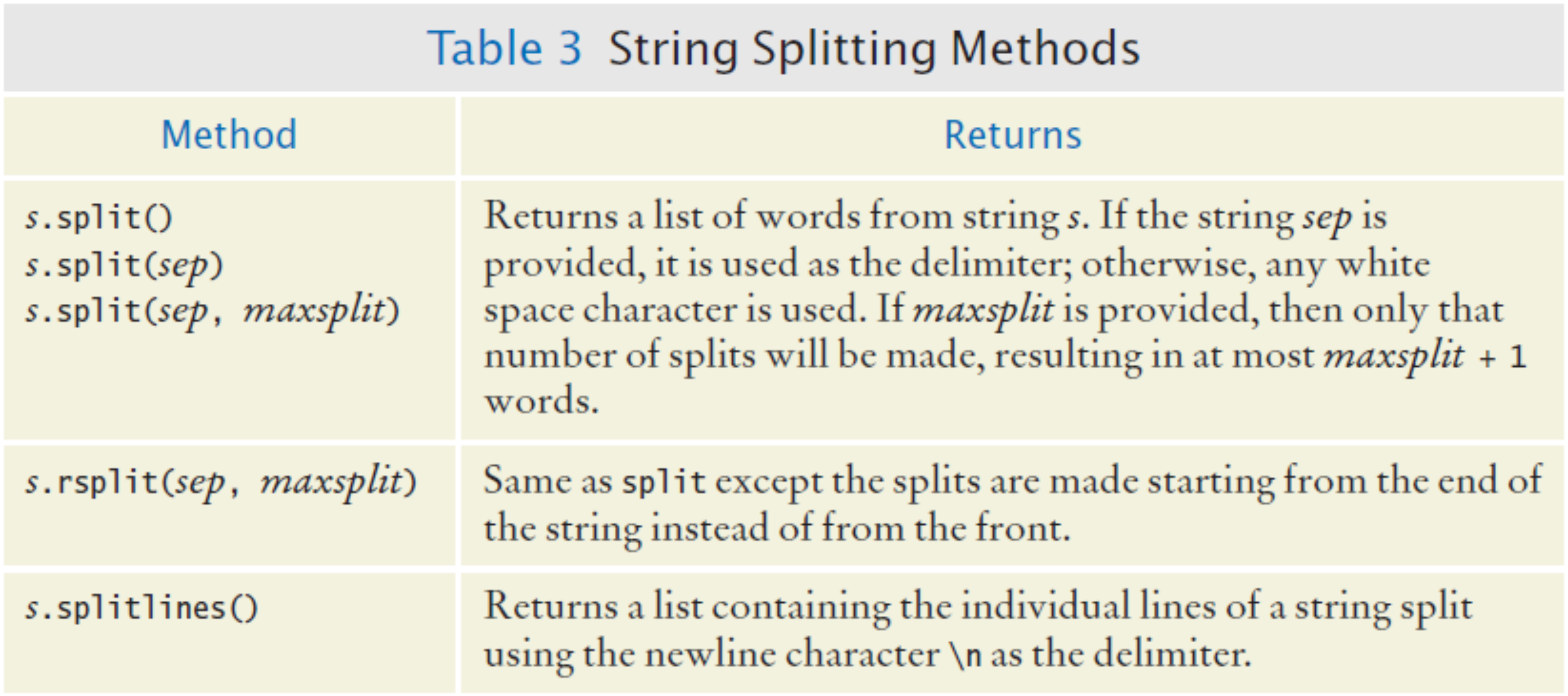


Table 3 String Splitting Methods

Method	Returns

`s.split()`
`s.split(`*sep*`)`
`s.split(`*sep*`,` *maxsplit*`)`
	Returns a list of words from string *s*. If the string *sep* is provided, it is used as the delimiter; otherwise, any white space character is used. If *maxsplit* is provided, then only that number of splits will be made, resulting in at most *maxsplit* `+ 1` words.

`s.rsplit(`*sep*`,` *maxsplit*`)`	Same as `split` except the splits are made starting from the end of the string instead of from the front.
`s.splitlines()`	Returns a list containing the individual lines of a string split using the newline character `\n` as the delimiter.

Additional String Splitting Examples

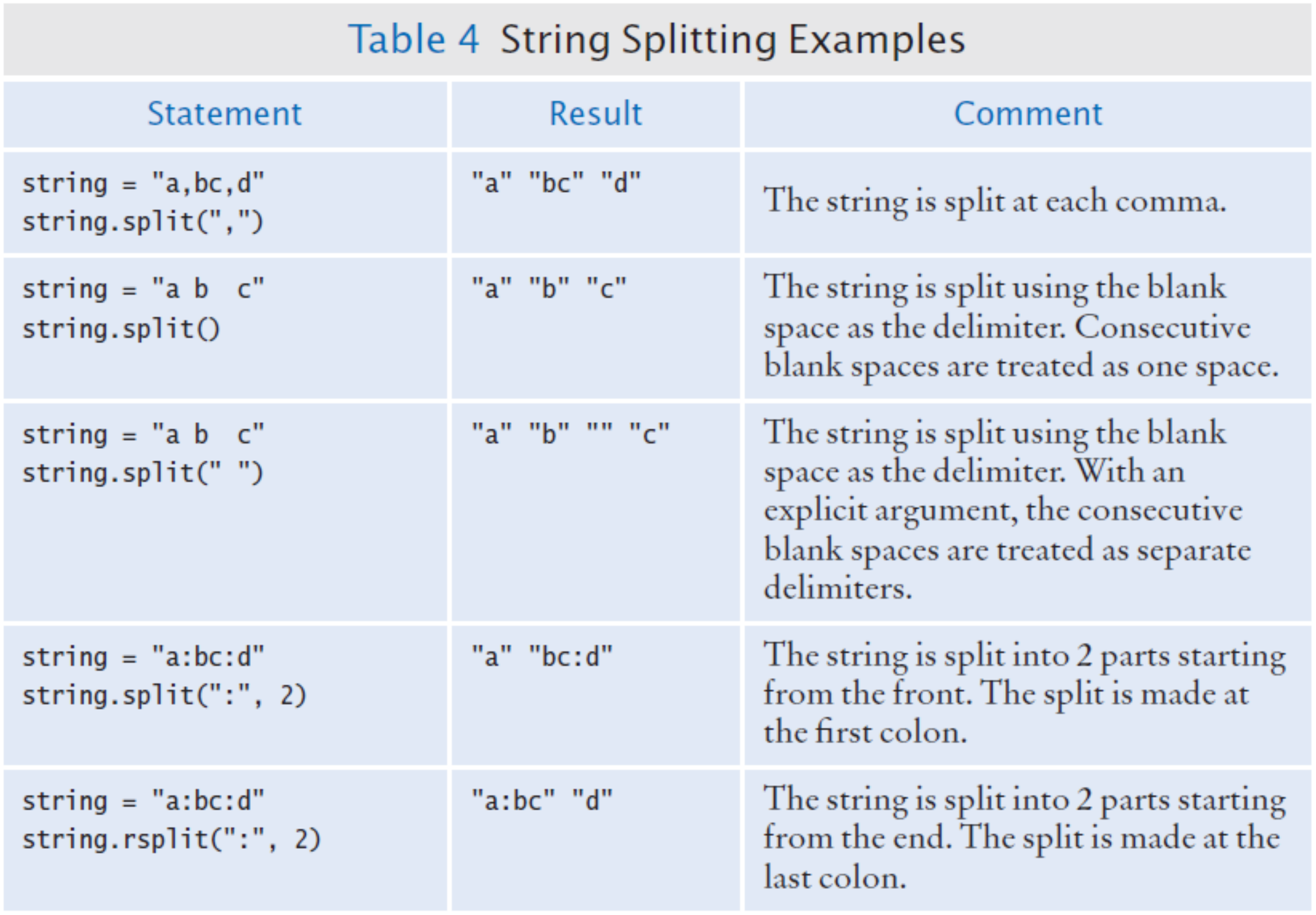


Table 4 String Splitting Examples

Statement	Result	Comment

`string = "a,bc,d"`
`string.split(",")`
	`"a" "bc" "d"`	The string is split at each comma.

`string = "a b  c"`
`string.split()`
	`"a" "b" "c"`	The string is split using the blank space as the delimiter. Consecutive blank spaces are treated as one space.

`string = "a b  c"`
`string.split(" ")`
	`"a" "b" "" "c"`	The string is split using the blank space as the delimiter. With an explicit argument, the consecutive blank spaces are treated as separate delimiters.

`string = "a:bc:d"`
`string.split(":", 2)`
	`"a" "bc:d"`	The string is split into 2 parts starting from the front. The split is made at the first colon.

`string = "a:bc:d"`
`string.rsplit(":", 2)`
	`"a:bc" "d"`	The string is split into 2 parts starting from the end. The split is made at the last colon.