gosamples.dev

8 ways to split a string in Go (Golang) | gosamples.dev

6–7 minutes

String splitting is one of the most common actions in everyday programming. Often, one single string has to be split into a list of substrings according to a specific separator, such as when parsing user-entered arguments, environment variables, or column data from a CSV file row. In Go, there are many different ways to split strings. We found 8 of them in the standard packages, so you have a wide range of tools you can use to perform this task.

Split a string by separator

To split a string in Go, use the `strings.Split()` function from the `strings` package. It splits a string into a list of substrings using the specified delimiter. The output substrings do not contain the separator.

The function signature is

where

- `s` is the string to split
- `sep` is a separator (delimiter) by which the string is split

Example

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    str := "strawberry, blueberry, raspberry"
    fmt.Printf("strings.Split(): %#v\n",
strings.Split(str, ", "))
}
```

Output:

```
strings.Split(): []string{"strawberry",
"blueberry", "raspberry"}
```

Split a string without removing the separator

To split a string in Go into substrings containing the separator by which they were split, use the `strings.SplitAfter()` function. It splits a string after each occurrence of the delimiter.

The functions signature:

- `s` is the string to split
- `sep` is a separator (delimiter) by which the string is split

Example

```
package main

import (
    "fmt"
```

```
    "strings"
)

func main() {
    str := "strawberry, blueberry, raspberry"
    fmt.Printf("strings.SplitAfter(): %#v\n",
strings.SplitAfter(str, ", "))
}
```

Output:

```
strings.SplitAfter(): []string{"strawberry, ",
"blueberry, ", "raspberry"}
```

Cut a string into 2 parts

To cut a string on the first occurrence of the delimiter in Go, use the `strings.Cut()` function. It slices a string and returns the text before and after the separator.

The function signature:

The `strings.Cut()` takes a string `s` and a separator `sep` as arguments and splits the string `s` on the first occurrence of the `sep`. It returns the text before and after the `sep`, and the boolean value `found` indicating whether `sep` appears in the `s`.

Example

```
package main

import (
    "fmt"
    "strings"
)
```

```
func main() {
    str := "strawberry, blueberry, raspberry"
    before, after, found := strings.Cut(str, ",
")
    fmt.Printf("strings.Cut():\nbefore:
%s\nafter: %s\nseparator found: %t\n", before,
after, found)
}
```

Output:

```
strings.Cut():
before: strawberry
after: blueberry, raspberry
separator found: true
```

Split a string to at most n substrings

To split a string in Go and receive at most n substrings, use the `strings.SplitN()` function. The last substring in this case will be the unsplit remainder.

The function signature:

The `strings.SplitN()` function works the same way as `strings.Split()` except that it finishes after n substrings.

Example

```
package main

import (
    "fmt"
    "strings"
```

```
)

func main() {
    str := "strawberry, blueberry, raspberry"
    fmt.Printf("strings.SplitN(): %#v\n",
strings.SplitN(str, ", ", 2))
}
```

Output:

```
strings.SplitN(): []string{"strawberry",
"blueberry, raspberry"}
```

Split a string without removing the separator to at most n substrings

To split a string in Go into output substrings containing a separator and getting at most n substrings, use the `strings.SplitAfterN()` function. It splits a string after each occurrence of the delimiter, and the last substring will be the unsplit remainder.

The function signature:

The `strings.SplitAfterN()` function works the same way as `strings.SplitAfter()` except that it finishes after n substrings.

Example

```
package main

import (
    "fmt"
    "strings"
```

```
)

func main() {
    str := "strawberry, blueberry, raspberry"
    fmt.Printf("strings.SplitAfterN(): %#v\n",
strings.SplitAfterN(str, ", ", 2))
}
```

Output:

```
strings.SplitAfterN(): []string{"strawberry, ",
"blueberry, raspberry"}
```

Split a string by white space characters

To split a string by white space characters in Go, use the `strings.Fields()` function. It takes a string as an argument and splits it according to the white space characters defined by the `unicode.IsSpace()` function.

The function signature:

Example

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    str := "strawberry, blueberry, raspberry"
    fmt.Printf("strings.Fields(): %#v\n",
strings.Fields(str))
```

```
}
```

Output:

```
strings.Fields(): []string{"strawberry,",
"blueberry,", "raspberry"}
```

Split a string by a splitting function

To split a string according to a custom split function in Go, use the `strings.FieldsFunc()`. As arguments, it gets the string to split and the `func(rune) bool` function, which should return `true` if splitting should be done for a given rune.

The function signature:

Example

In the example, the string is split on runes that are not Unicode letters.

```
package main

import (
    "fmt"
    "strings"
    "unicode"
)

func main() {
    str := "strawberry, blueberry, raspberry"
    fmt.Printf("strings.FieldsFunc(): %#v\n",
strings.FieldsFunc(str, func(r rune) bool {
        return !unicode.IsLetter(r)
    }))
```

```
}
```

```
strings.FieldsFunc(): []string{"strawberry",
"blueberry", "raspberry"}
```

Split a string using the regexp

In Go, you can also split a string using a Regular Expression. To do this, you must first create a new regular expression object `Regexp`, for example, by calling the `regexp.MustCompile()` function. The `Regexp` object has the `Split()` method that splits a given string `s` by the regular expression into at most n substrings (the last substring will be the unsplit remainder).

The method signature:

Example

```
package main

import (
    "fmt"
    "regexp"
)

func main() {
    str := "strawberry,blueberry,    raspberry"

    regex := regexp.MustCompile(",\\s*")

    fmt.Printf("Regexp.Split(): %#v\n",
regex.Split(str, -1))
}
```

Output:

```
Regexp.Split(): []string{"strawberry",
"blueberry", "raspberry"}
```