slugify

Slugify a string

Useful for URLs, filenames, and IDs.

It handles most major languages, including German (umlauts), Vietnamese, Arabic, Russian, and more.

Install

```
$ npm install @sindresorhus/slugify
```

Usage

```
import slugify from '@sindresorhus/
        slugify';

slugify('I ♥ Dogs');
//=> 'i-love-dogs'

slugify('  Déjà Vu!  ');
//=> 'deja-vu'

slugify('fooBar 123 $#%');
//=> 'foo-bar-123'
```

```
slugify('я люблю единорогов');
//=> 'ya-lyublyu-edinorogov'
```

API

slugify(string, options?)

string

Type: `string`

String to slugify.

options

Type: `object`

separator

Type: `string`

Default: `'-'`

```
import slugify from '@sindresorhus/
        slugify';

slugify('BAR and baz');
//=> 'bar-and-baz'
```

```
slugify('BAR and baz', {separator:
        '_'});
//=> 'bar_and_baz'

slugify('BAR and baz', {separator: ''});
//=> 'barandbaz'
```

lowercase

Type: `boolean`

Default: `true`

Make the slug lowercase.

```
import slugify from '@sindresorhus/
        slugify';

slugify('Déjà Vu!');
//=> 'deja-vu'

slugify('Déjà Vu!', {lowercase: false});
//=> 'Deja-Vu'
```

decamelize

Type: `boolean`

Default: `true`

Convert camelcase to separate words. Internally it does `fooBar` → `foo bar`.

```
import slugify from '@sindresorhus/
        slugify';

slugify('fooBar');
```

```
//=> 'foo-bar'

slugify('fooBar', {decamelize: false});
//=> 'foobar'
```

customReplacements

Type: `Array<string[]>`

Default: `[ ['&', ' and '], ['🦄', ' unicorn '], ['♥', ' love '] ]`

Add your own custom replacements.

The replacements are run on the original string before any other transformations.

This only overrides a default replacement if you set an item with the same key, like &.

```
import slugify from '@sindresorhus/
        slugify';

slugify('Foo@unicorn', {
    customReplacements: [
        ['@', 'at']
    ]
});
//=> 'fooatunicorn'
```

Add a leading and trailing space to the replacement to have it separated by dashes:

```
import slugify from '@sindresorhus/
        slugify';

slugify('foo@unicorn', {
```



```
slugify('foo bar');
//=> 'foo-bar-2'

slugify.reset();

slugify('foo bar');
//=> 'foo-bar'
```

Related

- slugify-cli - CLI for this module
- transliterate - Convert Unicode characters to Latin characters using transliteration
- filenamify - Convert a string to a valid safe filename

Use-case example of counter

If, for example, you have a document with multiple sections where each subsection has an example.

```
## Section 1

### Example

## Section 2

### Example
```

You can then use `slugifyWithCounter()` to generate unique HTML `id`'s to ensure anchors will link to the right headline.

slugify.reset()

Reset the counter

Example

```
import {slugifyWithCounter} from
        '@sindresorhus/slugify';

const slugify = slugifyWithCounter();

slugify('foo bar');
//=> 'foo-bar'
```



```
    customReplacements: [
        ['@', ' at ']
    ]
});
//=> 'foo-at-unicorn'
```

Another example:

```
import slugify from '@sindresorhus/
        slugify';

slugify('I love 🐶', {
    customReplacements: [
        ['🐶', 'dogs']
    ]
});
//=> 'i-love-dogs'
```

preserveLeadingUnderscore

Type: `boolean`

Default: `false`

If your string starts with an underscore, it will be preserved in the slugified string.

Sometimes leading underscores are intentional, for example, filenames representing hidden paths on a website.

```
import slugify from '@sindresorhus/
        slugify';

slugify('_foo_bar');
//=> 'foo-bar'
```

```
slugify('_foo_bar',
        {preserveLeadingUnderscore:
        true});
//=> '_foo-bar'
```

preserveTrailingDash

Type: `boolean`

Default: `false`

If your string ends with a dash, it will be preserved in the slugified string.

For example, using slugify on an input field would allow for validation while not preventing the user from writing a slug.

```
import slugify from '@sindresorhus/
        slugify';

slugify('foo-bar-');
//=> 'foo-bar'

slugify('foo-bar-',
        {preserveTrailingDash: true});
//=> 'foo-bar-'
```

preserveCharacters

Type: `string[]`

Default: `[]`

Preserve certain characters.

It cannot contain the `separator`.

For example, if you want to slugify URLs, but preserve the HTML fragment # character.


```
import slugify from '@sindresorhus/
        slugify';

slugify('foo_bar#baz',
        {preserveCharacters: ['#']});
//=> 'foo-bar#baz'
```

slugifyWithCounter()

Returns a new instance of `slugify(string, options?)` with a counter to handle multiple occurrences of the same string.

Example

```
import {slugifyWithCounter} from
        '@sindresorhus/slugify';

const slugify = slugifyWithCounter();

slugify('foo bar');
//=> 'foo-bar'

slugify('foo bar');
//=> 'foo-bar-2'

slugify.reset();

slugify('foo bar');
//=> 'foo-bar'
```