



Indrek Lasn

Swiss based Software Engineer 😍

Oct 27

The best front-end hacking cheatsheets — all in one place.

It's rather impossible to remember all the APIs by heart. This is when cheatsheets jump in! Here are the best front-end cheatsheets I've gathered.

Javascript ES2015 features

A quick overview of new JavaScript features in ES2015, ES2016, ES2017 and beyond.

Block scoping

Let

```
function fn () {
  let x = 0
  if (true) {
    let x = 1 // only inside this `if`
  }
}
```

Const

```
const a = 1
```

`let` is the new var. Constants work just like `let`, but can't be reassigned. See: [Let](#) and [const](#)

New methods

New string methods

```
"hello".repeat(3)
"hello".includes("ll")
```

Backtick strings

Interpolation

```
var message = `Hello ${name}`
```

Multiline strings

```
var str = `
hello
world
`
```

Templates and multiline strings. See: [Template strings](#)

Binary and octal literals

```
let bin = 0b1010010
let oct = 0o755
```

See: [Binary and octal literals](#)

Javascript

<https://devhints.io/es6>

Resource

Online
Official Website

Download
JavaScript Cheat Sheet [pdf]

Basic Objects

Array Properties
constructor
length
prototype

Date Object

Date Properties
constructor
prototype

Date Methods

Browser

Window Properties
closed
defaultStatus
document
frames

DOM Events

Mouse Events
click
dblclick
mousedown
mousemove

JavaScript Cheat Sheet [.pdf]
JavaScript Quick Reference Card [.pdf]
JavaScript and Browser Objects Quick Reference [.pdf]
JavaScript in one page

Related
AJAX
CSS
HTML DOM
GWT
HTML
jQuery
MooTools
Node.js
Prototype
XHTML

DOM Node

Node Properties

attributes
baseURI
childNodes
firstChild
lastChild
localName
namespaceURI
nextSibling
nodeValue

Array Methods
concat()
indexOf()
join()
lastIndexOf()
pop()
push()
reverse()
shift()
slice()
sort()
splice()
toString()
unshift()
valueOf()

Boolean Properties
constructor
prototype

Boolean Methods
toString()
valueOf()

Math Properties

E
LN2
LN10
LOG2E
LOG10E
PI
SQRT1_2
SQRT2

Date Methods
getDate()
getDay()
getFullYear()
getHours()
getMilliseconds()
getMinutes()
getMonth()
getSeconds()
getTime()
getTimezoneOffset()
getUTCDate()
getUTCDay()
getUTCFullYear()
getUICHours()
getUTCMilliseconds()
getUTCMonth()
getUTCSeconds()
parse()
setDate()
setFullYear()
setHours()
setMilliseconds()
setMinutes()
setMonth()
setSeconds()
setTime()
setUTCDate()
setUTCFullYear()
setUICHours()
setUTCMilliseconds()
setUTCMonth()
setUTCYear()

history
innerHeight
innerWidth
length
location
name
navigator
opener
outerHeight
outerWidth
pageXOffset
pageYOffset
parent
screen
screenLeft
screenTop
screenX
screenY
self
status
top

Window Methods

alert()
blur()
clearInterval()
clearTimeout()
close()
confirm()
focus()
moveBy()
moveTo()
open()

mousemove
mouseover
mouseout
mouseup

Keyboard Events

keydown
keypress
keyup

Frame Events

abort
error
load
resize
scroll
unload

Form Events

blur
change
focus
reset
select
onsubmit

Event Object Constant

AT_TARGET
BUBBLING_PHASE
CAPTURING_PHASE

Event Object Properties

<http://overapi.com/javascript>

Javascript Regular expression

◀ ▶ 🔍

JavaScript Regex Cheatsheet

Quick Filter

Regular Expression Basics		Regular Expression Character Classes		Regular Expression Flags	
.	Any character except newline	[ab-d]	One character of: a, b, c, d	g	Global Match
a	The character a	[^ab-d]	One character except: a, b, c, d	i	Ignore case
ab	The string ab	\b	Backspace character	m	^ and \$ match start and end of line
a b	a or b	\d	One digit		
a*	0 or more a's	\D	One non-digit		
\	Escapes a special character	\s	One whitespace		
Regular Expression Quantifiers		\S	One non-whitespace		
*	0 or more	\w	One word character		
+	1 or more	\W	One non-word character		
?	0 or 1	Regular Expression Assertions			
{2}	Exactly 2	^	Start of string		
{2, 5}	Between 2 and 5	\$	End of string		
{2,}	2 or more	\b	Word boundary		
Default is greedy. Append ? for reluctant.		\B	Non-word boundary		
Regular Expression Groups		(?=...)	Positive lookahead		
(...)	Capturing group	(?!...)	Negative lookahead		
(?:...)	Non-capturing group				
\Y	Match the Y'th captured group				
Regular Expression Special Characters					
\n	Newline				
\r	Carriage return				
\t	Tab				
\0	Null character				
\YYY	Octal character YYY				
\xYY	Hexadecimal character YY				
\uYYYY	Hexadecimal character YYYY				
\cY	Control character Y				
Regular Expression Replacement					
\$\$	Inserts \$				
\$.&	Insert entire match				
\$`	Insert preceding string				
\$'	Insert following string				
\$Y	Insert Y'th captured group				

New to Debuggex? Check out the regex tester!

<https://www.debuggex.com/cheatsheet/regex/javascript>

React

React is a JavaScript library for building user interfaces. This guide targets React v15 to v16.

Components

```
import React from 'react'
import ReactDOM from 'react-dom'

class Hello extends React.Component {
  render () {
    return <div className='message-box'>
      Hello {this.props.name}
    </div>
  }
}

const el = document.body
ReactDOM.render(<Hello name='John' />, el)
```

Use the [React.js jsfiddle](#) to start hacking. (or the [unofficial jsbin](#))

Properties

```
<Video fullscreen={true} />

render () {
  this.props.fullscreen
  ...
}
```

Use `this.props` to access properties passed to the component.

See: [Properties](#)

Children

```
<AlertBox>
  <h1>You have pending notifications</h1>
</AlertBox>

class AlertBox extends React.Component {
  render () {
    return <div className='alert-box'>
      {this.props.children}
    </div>
  }
}
```

States

```
this.setState({ username: 'rstacruz' })
```

```
render () {
  this.state.username
  ...
}
```

Use `states` (`this.state`) to manage dynamic data.

See: [States](#)

Nesting

```
class Info extends React.Component {
  render () {
    const { avatar, username } = this.props

    return <div>
      <UserAvatar src={avatar} />
      <UserProfile username={username} />
    </div>
  }
}
```

Nest components to separate concerns.

<https://devhints.io/react>

Redux



Redux Cheat Sheet

```
import React from 'react'
import ReactDOM from 'react-dom'
import { createStore, combineReducers, applyMiddleware, bindActionCreators } from 'redux'

const greetingReducer = (state='', action) => {
  switch (action.type) {
    case 'SAY_HELLO': return 'Hello '
    case 'SAY_GOODBYE': return 'Goodbye '
  }
  return state
}

const nameReducer = (state='John', action) => {
  switch (action.type) {
    case 'CHANGE_NAME': return 'Joel'
  }
  return state
}

const actionLogger = ({dispatch, getState}) => (next) => (action) => {
  console.log(action)
  return next(action)
}

const reducers = combineReducers({
  greeting: greetingReducer,
  name: nameReducer
})

const middleware = applyMiddleware(actionLogger)
const store = createStore(reducers, middleware)
```

Welcome to the egghead.io Redux cheat sheet! On your left you will find a full-fledged Redux application with a React.js front-end (React is not required).

`function reducer(state, action) => State`

Takes the previous state and an action, and returns the next state.

Splitting your app into multiple reducers (`greetingsReducer, nameReducer`) allows for a clean separation of concerns when modifying your application's state.

`function middleware(dispatch, getState) => next => action`

Receives Store's `dispatch` and `getState` functions as named arguments, and returns a function. That function will be given the next middleware's `dispatch` method, and is expected to return a function of action calling `next(action)` with a potentially different argument, or at a different time, or maybe not calling it at all. The last middleware in the chain will receive the real store's `dispatch` method as the next parameter, thus ending the chain.

`combineReducers(reducers) => Function`

Combines multiple reducers into a single reducing function with each reducer as a key/value pair. Can then be passed to `createStore()`.

`applyMiddleware(middleware) => Function`

Extends Redux with custom functionality by wrapping the store's `dispatch` method.

`createStore(reducer, initialState, enhancer) => Store`

Redux's Three Principles

Single source of truth

State is read-only

Changes are made with pure functions

Glossary

State

`type State = any`

Action

`type Action = { type: STRING, payload: ANY }`

Reducer

`type Reducer<State, Action> = (State, Action) => State`

Dispatching Functions

`type BaseDispatch = (Action) => Action`

`type Dispatch = (Action | AsyncAction) => any`

Action Creator

`type ActionCreator = (ANY) => Action | AsyncAction`

Async Action

`type AsyncAction = any`

Middleware

`type MiddlewareAPI = { dispatch: Dispatch, getState: () => State }`

```
reducers
{
  greeting: '(Roll over me) ',
  middleware
}

const changeName = () => {return { type: 'CHANGE_NAME' } }
const hello = () => {return { type: 'SAY_HELLO' } }
const goodbye = () => {return { type: 'SAY_GOODBYE' } }

const Hello = (props) =>
<div>
  onMouseOver={props.hello}
  onMouseOut={props.goodbye}
  onClick={props.changeName}>
    {props.greeting}{props.name}
  </div>

const render = () => {
  ReactDOM.render(
    <Hello
      greeting={store.getState().greeting}
      name={store.getState().name}
      {...bindActionCreators({changeName, hello, goodbye}, store.dispatch)}>
  
```

Creates a Redux store that holds the complete state tree of your app. There should only be a single store in your app.

```
store = { ... }

Brings together your application's state and has the following responsibilities:

• Allows access to state via getState();
• Allows state to be updated via dispatch(action);
• Registers listeners via subscribe(listener);
• Handles unregistering of listeners via the function returned by subscribe(listener).
```

```
action = { type: String, ...payload: any }

Holds action payloads in plain javascript objects. Must have a type property that indicates the performed action, typically be defined as string constants. All other properties are the action's payload.
```

```
function actionCreator( ?any ) => Action|AsyncAction
```

```
type Middleware = ( MiddlewareAPI ) => ( Dispatch ) => Dispatch

Store

type Store =

{

  dispatch( Action | AsyncAction ) => any,
  getState() => State,
  subscribe( () => VOID ) => () => void,
  replaceReducer( Reducer ) => void
}

Store Creator

type StoreCreator = ( Reducer , ?initialState , ?enhancer ) => Store

Store Enhancer

type StoreEnhancer = ( StoreCreator ) => StoreCreator
```

<https://github.com/linkmesrl/react-journey-2016/blob/master/resources/egghead-redux-cheat-sheet-3-2-1.pdf>

Vuejs



<https://vuejs-tips.github.io/cheatsheet/>

Vuex

entry.js	store.js	import map	Plugin Dev
<pre>import Vue from 'vue' import Vuex from 'vuex' import store from './store' Vue.use(Vuex)</pre>	<pre>new Vuex.Store({ strict: true, plugins: [], modules: {}},</pre>	<pre>mapState() mapGetters() mapMutations() mapActions()</pre>	<pre>replaceState() watch() subscribe() registerModule()</pre>

```

new Vue({
  el: '#app',
  store
})

```

```

state: {},
getters: {},
mutations: {},
actions: {},
namespaced: false
})

```

Instance	store
state	
getters	
commit()	
dispatch()	

Context Object
state
rootState
getters
rootGetters
commit()
dispatch()

<https://vuejs-tips.github.io/vuex-cheatsheet/>

Angular 4

⇒ Cheat Sheet

Bootstrapping

```

import { platformBrowserDynamic } from
'@angular/platform-browser-dynamic';

```

```
platformBrowserDynamic().bootstrapModule(AppModule);
```

Bootstraps the app, using the root component from the specified NgModule .

NgModules

```

import { NgModule } from '@angular/core';

```

```

@NgModule({ declarations: ..., imports: ...,
exports: ..., providers: ..., bootstrap: ...})
class MyModule {}

```

Defines a module that contains components, directives, pipes, and providers.

```

declarations: [MyRedComponent, MyBlueComponent,
MyDatePipe]

```

List of components, directives, and pipes that belong to this module.

```
imports: [BrowserModule, SomeOtherModule]
```

List of modules to import into this module. Everything from the imported modules is available to declarations of this module.

<https://angular.io/guide/cheatsheet>

Flexbox

display

[Change theme](#)w3.org/TR/css-flexbox-1/#flex-containers[flex](#) [inline-flex](#)

Some text



```
.parent {  
  display: flex;  
}
```

A *flex container* establishes a new *flex formatting context* for its contents. This is the same as establishing a block formatting context, except that flex layout is used instead of block layout. For example, floats do not intrude into the flex container, and the flex container's margins do not collapse with the margins of its contents. *Flex containers* form a containing block for their contents exactly like block containers do. The `overflow` property applies to *flex containers*.

<https://yoksel.github.io/flex-cheatsheet/>

SCSS

Sass cheatsheet

Proudly sponsored by

TrackJS JavaScript Happens. You better know when it breaks. Fix Bugs Fast with TrackJS 🚀

powered by codesponsor.io

Variables

```
$red: #833;
```

```
body {  
  color: $red;  
}
```

Mixin properties

```
@ mixin font-size($n) {  
  font-size: $n * 1.2em;  
}
```

Nesting

```
.markdown-body {  
  p {  
    color: blue;  
  }  
  
  &:hover {  
    color: red;  
  }  
}
```

Extend

```
.button {
```

Comments

```
/* Block comments */  
// Line comments
```

Mixins

```
@ mixin heading-font {  
  font-family: sans-serif;  
  font-weight: bold;  
}  
  
h1 {  
  @include heading-font;  
}
```

```
h1 {  
  @include heading-font;  
}
```

```

body {
  @include font-size(2);
}

.push-button {
  @extend .button;
}

```

Composing

```
@import './other_sass_file';
```

<https://devhints.io/sass>

Stylus

Stylus cheatsheet

Proudly sponsored by

Pusher  Add realtime WebSocket goodness to your app in minutes!

powered by codesponsor.io

CSS syntax

```
.box {
  color: blue;

.button {
  color: red;
}
}
```

Stylus is a CSS pre-processor.

See: stylus-lang.com

Indent syntax

```
.box
  color: blue

.button
  color: red
```

Also works! The colon is optional, as well. This is typically the syntax used with Stylus documents.

Mixins

```
caps-type()
  text-transform: uppercase
  letter-spacing: 0.05em
```

```
h5
  caps-type()
```

See Mixins below.

Variables

```
royal-blue = #36a
```

```
div
  color: royal-blue
```

<https://devhints.io/stylus>

GraphQL



GraphQL Schema Language Cheat Sheet

The definitive guide to express your GraphQL schema succinctly

Last updated: 28 January 2017

Prepared by: Hafiz Ismail / @sogko

What is GraphQL Schema Language?

It is a shorthand notation to succinctly express the basic shape of your GraphQL schema and its type system.

What does it look like?

Below is an example of a typical GraphQL schema expressed in shorthand:

```
# define Entity interface
interface Entity {
  id: ID!
  name: String
```

Schema

schema	GraphQL schema definition
query	A read-only fetch operation
mutation	A write followed by fetch operation
subscription	A subscription operation (experimental)

Built-in Scalar Types

Input Arguments

Basic Input

```
type Query {
  users(limit: Int): [User]
}
```

Input with default value

```
type Query {
  users(limit: Int = 10): [User]
}
```

Input with multiple arguments

```
type Query {
  users(limit: Int, sort: String): [User]
}
```

Interfaces

Object implementing one or more Interfaces

```
interface Foo {
  is_foo: Boolean
}
```

```
interface Goo {
  is_goo: Boolean
}
```

```
type Bar implements Foo, Goo {
  is_foo: Boolean
  is_bar: Boolean
}
```

```
type Baz implements Foo, Goo {
  is_foo: Boolean
  is_goo: Boolean
  is_baz: Boolean
}
```

```

} }

# define custom Url scalar
scalar Url

# User type implements Entity interface
type User implements Entity {
  id: ID!
  name: String
  age: Int
  balance: Float
  is_active: Boolean
  friends: [User]!
  homepage: Url
}

# root Query type
type Query {
  me: User
  friends(limit: Int = 10): [User]!
}

# custom complex input type
input ListUsersInput {
  limit: Int!
  since_id: ID
}

# root mutation type
type Mutation {
  users(params: ListUsersInput): [User]!
}

# GraphQL root schema type
schema {
  query: Query
  mutation: Mutation
  subscription: ...
}

```

Int	Int
Float	Float
String	String
Boolean	Boolean
ID	ID

Type Definitions

scalar	Scalar Type
type	Object Type
interface	Interface Type
union	Union Type
enum	Enum Type
input	Input Object Type

Type Modifiers

String	Nullable String
String!	Non-null String
[String]	List of nullable Strings
[String]!	Non-null list of nullable Strings
[String!]!	Non-null list of non-null Strings

Input with multiple arguments and default values

```

type Query {
  users(limit: Int = 10, sort: String): [User]
}

type Query {
  users(limit: Int, sort: String = "asc"): [User]
}

type Query {
  users(limit: Int = 10, sort: String = "asc"): [User]
}

```

Input Types

```

input ListUsersInput {
  limit: Int!
  since_id: ID
}

type Mutation {
  users(params: ListUsersInput): [User]!
}

```

Custom Scalars

```

scalar Url
type User {
  name: String
  homepage: Url
}

```

Unions

Union of one or more Objects

```

type Foo {
  name: String
}

type Bar {
  is_bar: String
}

union SingleUnion = Foo
union MultipleUnion = Foo | Bar

type Root {
  single: SingleUnion
  multiple: MultipleUnion
}

```

Enums

```

enum USER_STATE {
  NOT_FOUND
  ACTIVE
  INACTIVE
  SUSPENDED
}

type Root {
  stateForUser(userID: ID!): USER_STATE!
  users(state: USER_STATE, limit: Int = 10): [User]
}

```

<https://raw.githubusercontent.com/sogko/graphql-shorthand-notation-cheat-sheet/master/graphql-shorthand-notation-cheat-sheet.png>

Missing your favorite cheatsheet? Please let me know in the comments!

If you found this post useful, please give me some claps so more people see it. Thanks!

How to setup Webpack +2.0 from scratch

Best courses to learn Javascript in 2017

Learn how to: Build A Cryptocurrency Native Mobile App With RN + Redux

And of course, don't forget to follow me for more! ❤

edit: If you liked this, [follow my Twitter as well!](#)

JavaScript

Web Development

Software Development

Tech

Programming

One clap, two clap, three clap, forty?

By clapping more or less, you can signal to us which stories really stand out.



9.2K

Q 30 ↑ ⌂ ⌂ ...



Indrek Lasn

Medium member since Oct 2017

Swiss based Software Engineer 😍

Follow



freeCodeCamp

Our community publishes stories worth reading on development, design, and data science.

Following ▾



More on JavaScript from freeCodeCamp

React's JSX: The Other Side of the Coin



Cory House



998



Related reads

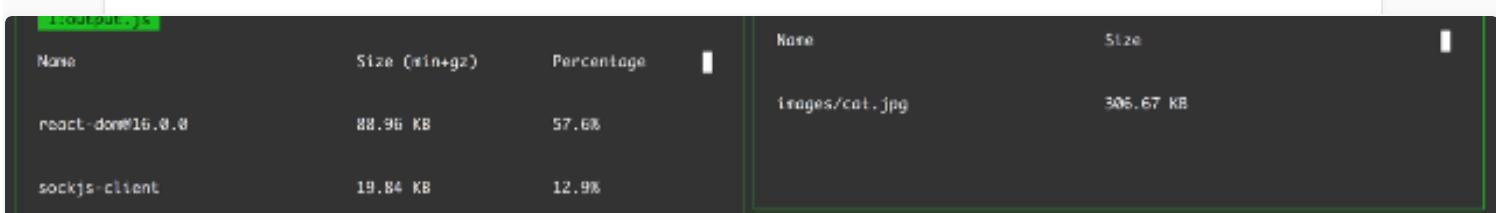
The Front-End Checklist



Brandon Morelli



3.6K



Related reads

Next Level Webpack Dashboard



Indrek Lasn



1K



Responses Conversation with Indrek Lasn.



Michael Skweres

Oct 30

Great collection. Hope you don't mind me sharing the post on my social channels :)



7

1 response ▾



Indrek Lasn

Oct 30

Thanks, go ahead! 😊