**Section2:Lesson 15**

**Git is a version control system** (<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>) that helps you control your source code. For example, you can easily go back to older states of your code or revert in case you mess up your code.

Using it, is totally optional! It's also not a tool that's related to JavaScript in any way - you can indeed use Git in ANY (programming) project.

Since it is optional and not directly related to JavaScript, I will not dive into Git in this course. Nonetheless, it's definitely a tool worth learning for every developer.

In case you are interested, this tutorial gets you started: <https://academind.com/learn/web-dev/git-the-basics/>

Besides local version control and code management, you can also push your projects and all your commits ("snapshots") to remote storages. The most prominent (free!) remote service is GitHub: <http://github.com/>

You can learn more about it here: <https://academind.com/learn/web-dev/github-the-basics/>

Also check out the official docs: <https://git-scm.com/about>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String#Escape_notation>

Escape notation

Besides regular, printable characters, special characters can be encoded using escape notation:

| **Code** | **Output** |
| --- | --- |
| \*XXX* (where *XXX* is 1–3 octal digits; range of 0–377) | ISO-8859-1 character / Unicode code point between U+0000 and U+00FF |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \n | new line |
| \r | carriage return |
| \v | vertical tab |
| \t | tab |
| \b | backspace |
| \f | form feed |
| \u*XXXX* (where *XXXX* is 4 hex digits; range of 0x0000–0xFFFF) | UTF-16 code unit / Unicode code point between U+0000 and U+FFFF |
| \u{*X*} ... \u{*XXXXXX*} (where *X*…*XXXXXX* is 1–6 hex digits; range of 0x0–0x10FFFF) | UTF-32 code unit / Unicode code point between U+0000 and U+10FFFF |
| \x*XX* (where *XX* is 2 hex digits; range of 0x00–0xFF) | ISO-8859-1 character / Unicode code point between U+0000 and U+00FF |

**Note:** Unlike some programming languages, JavaScript makes no distinction between single-quoted strings and double-quoted strings. Therefore, the escape sequences above work in strings created with either single or double quotes.

Useful Resources & Links

Attached, you find the source code for this section. The snapshots are also attached to individual lectures throughout this module (i.e. the lectures to which each snapshot belongs).

---

The following resources may be helpful.

Please keep in mind that this course is structured differently and builds up knowledge step by step - the linked articles don't necessarily do that and hence can easily be overwhelming. So going through them is optional and neither required nor necessarily recommended at this point of time:

* MDN => JavaScript Basics: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
* MDN => Variables: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables>
* MDN => Operators: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math>
* MDN => Functions: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Functions>
* MDN => Arrays: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Arrays>

MDN => Objects: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics>The following resources may be helpful.

Resources for this lecture

* [control-01-starting-project.zip](javascript:void(0))
* [control-02-added-if.zip](javascript:void(0))
* [control-03-monster-killer-starting-project.zip](javascript:void(0))
* [control-04-attack-function.zip](javascript:void(0))
* [control-05-attack-if-else-if.zip](javascript:void(0))
* [control-06-strong-attack.zip](javascript:void(0))
* [control-07-heal-functionality.zip](javascript:void(0))
* [control-08-bonus-life-functionality.zip](javascript:void(0))
* [control-09-reset-logic.zip](javascript:void(0))
* [control-10-player-set-initial-health.zip](javascript:void(0))
* [control-11-added-logging.zip](javascript:void(0))
* [control-12-switch-case.zip](javascript:void(0))
* [control-13-for-loop.zip](javascript:void(0))
* [control-14-for-of-loop.zip](javascript:void(0))
* [control-15-for-in.zip](javascript:void(0))
* [control-16-while.zip](javascript:void(0))
* [control-17-break-continue.zip](javascript:void(0))
* [control-18-labeled-statements.zip](javascript:void(0))
* [control-19-try-catch-finished.zip](javascript:void(0))

Resources for this lecture

* [basics-01-starting-project.zip](javascript:void(0))
* [basics-02-added-scripts-imports.zip](javascript:void(0))
* [basics-03-variables-operators.zip](javascript:void(0))
* [basics-04-added-constant.zip](javascript:void(0))
* [basics-05-added-template-literals.zip](javascript:void(0))
* [basics-06-basic-function.zip](javascript:void(0))
* [basics-07-function-with-return.zip](javascript:void(0))
* [basics-08-user-input-function.zip](javascript:void(0))
* [basics-09-parseint.zip](javascript:void(0))
* [basics-10-function-refactoring.zip](javascript:void(0))
* [basics-11-subtract-multiply-divide.zip](javascript:void(0))
* [basics-12-comments.zip](javascript:void(0))
* [basics-13-shorthand-operators.zip](javascript:void(0))
* [basics-14-basic-array-with-push.zip](javascript:void(0))
* [basics-15-extract-array-elements.zip](javascript:void(0))
* [basics-16-basic-object.zip](javascript:void(0))
* [basics-17-finished.zip](javascript:void(0))

Useful Resources & Links

Attached, you find the source code for this section. The snapshots are also attached to individual lectures throughout this module (i.e. the lectures to which each snapshot belongs).

---

The following resources may be helpful.

* VS Code Docs: <https://code.visualstudio.com/docs>
* VS Code Keybindings: <https://code.visualstudio.com/docs/getstarted/keybindings>
* VS Code Extensions Docs: <https://code.visualstudio.com/docs/editor/extension-gallery>
* Google Chrome DevTools Docs: <https://developers.google.com/web/tools/chrome-devtools/>

Resources for this lecture

* [dev-debugging-finished.zip](javascript:void(0))

Useful Resources & Links

Attached, you find the source code for this section. The snapshots are also attached to individual lectures throughout this module (i.e. the lectures to which each snapshot belongs).

---

The following resources may be helpful.

* Control Structures (MDN): <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling>
* JavaScript Loops (MDN): <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration>

Resources for this lecture

* [control-01-starting-project.zip](javascript:void(0))
* [control-02-added-if.zip](javascript:void(0))
* [control-03-monster-killer-starting-project.zip](javascript:void(0))
* [control-04-attack-function.zip](javascript:void(0))
* [control-05-attack-if-else-if.zip](javascript:void(0))
* [control-06-strong-attack.zip](javascript:void(0))
* [control-07-heal-functionality.zip](javascript:void(0))
* [control-08-bonus-life-functionality.zip](javascript:void(0))
* [control-09-reset-logic.zip](javascript:void(0))
* [control-10-player-set-initial-health.zip](javascript:void(0))
* [control-11-added-logging.zip](javascript:void(0))
* [control-12-switch-case.zip](javascript:void(0))
* [control-13-for-loop.zip](javascript:void(0))
* [control-14-for-of-loop.zip](javascript:void(0))
* [control-15-for-in.zip](javascript:void(0))
* [control-16-while.zip](javascript:void(0))
* [control-17-break-continue.zip](javascript:void(0))
* [control-18-labeled-statements.zip](javascript:void(0))
* [control-19-try-catch-finished.zip](javascript:void(0))

122 -Useful Resources & Links

The following resources may be helpful - please be aware that some of them are also very advanced.

* More on JavaScript Memory Management (Garbage Collection): <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management>
* V8's Garbage Collection Logic: https://v8.dev/blog/free-garbage-collection
* V8's JavaScript Engine in Great Detail: <https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef>
* More on Primitive vs Reference Values: <https://academind.com/learn/javascript/reference-vs-primitive-values/>

Resources for this lecture

* [behind-scenes-01-starting-setup.zip](javascript:void(0))
* [behind-scenes-02-block-scope.zip](javascript:void(0))
* [behind-scenes-03-strict-mode.zip](javascript:void(0))
* [behind-scenes-04-starting-setup-2.zip](javascript:void(0))
* [behind-scenes-05-callback-example.zip](javascript:void(0))

Different Arrow Function Syntaxes

For arrow functions, you got a couple of different syntaxes which you can use - here's a summary.

**Important: Don't miss the "function only returns an object" special case at the end of this article!**

**1) Default syntax:**

1. const add = (a, b) => {
2. const result = a + b;
3. return result; // like in "normal" functions, parameters and return statement are OPTIONAL!
4. };

*Noteworthy: Semi-colon at end, no function keyword, parentheses around parameters/ arguments.*

**2) Shorter parameter syntax, if exactly one parameter is received:**

1. const log = message => {
2. console.log(message); // could also return something of course - this example just doesn't
3. };

*Noteworthy: Parentheses around parameter list can be omitted (for exactly one argument).*

**3) Empty parameter parentheses if NO arguments are received:**

1. const greet = () => {
2. console.log('Hi there!');
3. };

*Noteworthy: Parentheses have to be added (can't be omitted)*

**4) Shorter function body, if exactly one expression is used:**

1. const add = (a, b) => a + b;

*Noteworthy: Curly braces and return statement can be omitted, expression result is always returned automatically*

**5) Function returns an object (with shortened syntax as shown in 4)):**

1. const loadPerson = pName => ({name: pName });

*Noteworthy: Extra parentheses are required around the object, since the curly braces would otherwise be interpreted as the function body delimiters (and hence a syntax error would be thrown here).*

That last case can be confusing: Normally, in JavaScript, curly braces always can have exactly one meaning.

1. const person = { name: 'Max' }; // Clearly creates an object
2. if (something) { ... } // Clearly used to mark the if statement block

But when using arrow functions, curly braces can have two meanings:

1) Mark the function body (in default form)

2) Create an object which you want to return (in shorter function body form)

To "tell" JavaScript what you want to do, wrap the expression (e.g. object creation) in parentheses like shown above.

Useful Resources & Links

Attached, you find the source code for this section. The snapshots are also attached to individual lectures throughout this module (i.e. the lectures to which each snapshot belongs).

---

The following resources may be helpful.

* More on Functions (MDN): <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions>
* bind(): <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind>

Resources for this lecture

* [functions-01-starting-code.zip](javascript:void(0))
* [functions-02-function-expressions.zip](javascript:void(0))
* [functions-03-anonymous-functions.zip](javascript:void(0))
* [functions-04-game-user-choice.zip](javascript:void(0))
* [functions-05-arrow-functions.zip](javascript:void(0))
* [functions-06-finished-game-logic.zip](javascript:void(0))
* [functions-07-default-arguments.zip](javascript:void(0))
* [functions-08-rest-parameters.zip](javascript:void(0))
* [functions-09-functions-in-functions.zip](javascript:void(0))
* [functions-10-callback-functions.zip](javascript:void(0))
* [functions-11-bind-finished.zip](javascript:void(0))
* [functions-revisited.zip](javascript:void(0))

**QUERY METHODS**

---

1. document.querySelector(<CSS selector>);

Takes any CSS selector (e.g. '#some-id', '.some-class' or 'div p.some-class') and returns the first (!) matching element in the DOM. Returns null if no matching element could be found.

More information: <https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector>

1. document.getElementById(<ID>);

Takes an ID (without #, just the id name) and returns the element that has this id. Since the same ID shouldn't occur more than once on your page, it'll always return exactly that one element. Returns null if no element with the specified ID could be found.

More information: <https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById>

1. document.querySelectorAll(<CSS selector>);

Takes any CSS selector (e.g. '#some-id', '.some-class' or 'div p.some-class') and returns all matching elements in the DOM as a static (non-live) NodeList. Returns and empty NodeList if no matching element could be found.

More information: <https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll>

1. document.getElementsByClassName(<CSS CLASS>);

Takes a CSS class g (e.g. 'some-class') and returns a live HTMLCollection of matched elements in your DOM. Returns an empty HTMLCollection if not matching elements were found.

More information: <https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName>

1. document.getElementsByTagName(<HTML TAG>);

Takes an HTML tag (e.g. 'p') and returns a live HTMLCollection of matched elements in your DOM. Returns an empty HTMLCollection if not matching elements were found.

More information: <https://developer.mozilla.org/en-US/docs/Web/API/Element/getElementsByTagName>

There also is the getElementsByName() method which really isn't used commonly (<https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByName>).