# WEB programming

## HTTP server

The Go standard library provides built-in support for creating an HTTP server to serve your web content or making HTTP requests to those servers. Writing a simple HTTP server is a simple task in GO language. Here is an example published on [DigitalOcean](https://www.digitalocean.com/community/tutorials/how-to-make-an-http-server-in-go) site:

package main

import (

"fmt"

"io"

"log"

"net/http"

)

func getRoot(w http.ResponseWriter, r \*http.Request) {

fmt.Printf("got / request\n")

io.WriteString(w, "This is my website!\n")

}

func getHello(w http.ResponseWriter, r \*http.Request) {

fmt.Printf("got /hello request\n")

io.WriteString(w, "Hello HTTP!\n")

}

func main() {

mux := **http.NewServeMux()**

mux.HandleFunc("/", getRoot)

mux.HandleFunc("/hello", getHello)

log.Fatal(**http.ListenAndServe(":8080", mux)**)

}

The last parameter of the ListenAndServe function is called servemux. The GO WEB system has its own standard servemux, which is worth using in small applications. Alex Edwards provided a description of this [standard servemux](https://www.alexedwards.net/blog/an-introduction-to-handlers-and-servemuxes-in-go).

#### ListenAndServe

The main work in supplied program is done by the function [http.ListenAndServe](https://pkg.go.dev/net/http" \l "ListenAndServe).

func ListenAndServe(addr string, handler Handler) error

The http package documentation states that the second parameter of **ListenAndServe** function may be **nil**. In this case, we connect the router functions directly to the **http** object:

http.HandleFunc("/", getRoot)

http.HandleFunc("/hello", getHello)

**log.Fatal(http.ListenAndServe(":8080", nil))**

The [http.ListenAndServeTLS](https://pkg.go.dev/net/http" \l "ListenAndServeTLS) function is designed for the HTTPS protocol and performs the same functions as http.ListenAndServe: listens on the TCP network address **addr** and then calls Serve with **handler** to handle requests on incoming connections:

func ListenAndServeTLS(addr, certFile, keyFile string, handler Handler) error

Parameters:

* **addr** - the TCP network address should be in the format "**host:port**". If you omit the **host** (like we did with ":8080") then the server will listen on all your computer’s available network interfaces. You might sometimes see network addresses written using named ports like "**:http**" or "**:http-alt**" instead of a number. If you use a named port then Go will attempt to look up the relevant port number from your **/etc/services** file when starting the server, or will return an error if a match can’t be found. A slash at the end of the address indicates that it is the top of the tree. Longer patterns take precedence over shorter ones, so that if there are handlers registered for both "**/images/**" and "**/images/thumbnails/**", the latter handler will be called for paths beginning "**/images/thumbnails/**" and the former will receive requests for any other paths in the "**/images/**" subtree.
* **handler** - Go's **net/http** package ships with the simple but effective http.ServeMux [handler](https://pkg.go.dev/net/http" \l "ServeMux), plus a few functions to generate common handlers including **http.FileServer()**, **http.NotFoundHandler()** and **http.RedirectHandler()**. **Handler** is interface thus anything that satisfies the [http.Handler](https://pkg.go.dev/net/http" \l "Handler) interface, may be placed in the last parameter. An example of custom handler is presented on [alexedwards](https://www.alexedwards.net/blog/an-introduction-to-handlers-and-servemuxes-in-go) site, see section "Custom handlers".
* **certFile** - path to the SSL certificate file,
* **keyFile** - path to the private key file.

#### HTTPS protocol

* + - 1. When working with the HTTPS protocol, the .NET CLI creates a new certificate for each new project. After a while, when you dive into Trusted Root Certificated Authorities, you will see a bunch of certificates that you no longer have or don't remember for which project you created them. [Lewel Murithi](https://www.section.io/engineering-education/how-to-get-ssl-https-for-localhost/) proposed an elegant solution to this problem. The idea is this: you create a long-term self-signed certificate and place it into Trusted Root Certificated Authorities. Certificates for new projects will be signed with this certificate of your own CA. Obviously, this solution is only suitable for local networks, but Trusted Root Certificated Authorities will not be spammed.
      2. In order to start a server working under the HTTPS protocol, you need to generate an SSL certificate and a public-private key pair. The [OpenSSL](https://www.openssl.org/) command line interface (CLI) and toolkit allows you to generate SSL certificates, private keys, Certificate signing requests (CSR) and perform other kinds of cryptography operations. You need to create private key **.key** file and a certificate. Follow this instruction:

1. Generate **encrypted private key**

mkdir cert

cd cert

mkdir CA // Your own „Certificate authority“

cd CA

openssl genrsa -out CA.key -des3 3072

These commands will ask you a password. Select simple one and confirm it on next question.

1. Linux browsers require Subject Alternative Names. They can only be created via a configuration file. Create the text file given below and name it CA.CNF:

[req]

distinguished\_name = req\_distinguished\_name

req\_extensions = req\_ext

prompt = no

[req\_distinguished\_name]

C = LT

ST = N\_A

L = Kaunas

O = Home Computer

OU = IT

CN = localhost

[req\_ext]

subjectAltName = @alt\_names

[alt\_names]

IP.1 = 127.0.0.1

IP.2 = 192.168.1.107

IP.3 = 192.168.1.171

DNS.1 = localhost

DNS.2 = KIBIRVIBIR-PC

DNS.3 = gediminas-TravelMate-5760Z

Parameters:

**C** - country (2 letter code),

**ST** – state or Province name (full name),

**L** - Locality Name (e.g., city),

**O** - Organization Name (e.g., company),

**OU** - Organizational Unit Name (e.g., section)

**CN** - The fully-qualified domain name (FQDN) (e.g., [www.example.com](http://www.example.com/)),

**IP.x** – IP addresses, only firs row is mandatory, IP.2, IP.3 are address in my local network,

**DNS.x** - subject alternative names, once again only first name is mandatory, other two ones are names in my local network.

1. Generate a **root CA certificate** using the key generated, that will be valid for ten years

openssl req -x509 -sha256 -new -nodes -days 3650 -key CA.key \

-out CA.pem -config CA.CNF

A certificate for Windows can be created without a configuration file:

openssl req -x509 -sha256 -new -nodes -days 3650 -key CA.key -out CA.pem \

-subj "/C=LT/ST=N\_A/L=Kaunas/O=Home Computer/OU=IT/CN=localhost"

The values specified in the **subj** parameter (/C, /ST, /L, /O, /OU, /CN) are the same as in the configuration file.

Certificate importing tool on Linux requires CRT extension, thus copy the key:

cp CA.pem CA.crt

Certificate importing tool requires PFX format. Convert pem into pfx with command:

openssl pkcs12 -export -out CA.pfx -inkey CA.key -in CA.pem

Windows users must import CA.pfx file into **Trusted Root Certification Authorities**. Import root certificate with **mmc** snap-in. Run mmc, select File → Add/Remove Snap-in → Certificates → My user account → OK. Open the new snap-in and select Trusted Root Certification Authorities. Right click on Certificates → All Tasks → Import… Import the CA.pfx file from CA directory. You will find the details on  [thewindowsclub](https://www.thewindowsclub.com/manage-trusted-root-certificates-windows?utm_content=cmp-true) site.

Linux users must run the following commands in a terminal:

sudo apt-get install -y ca-certificates

sudo cp CA.crt /usr/local/share/ca-certificates

sudo update-ca-certificates

After this correction Ubuntu tools (curl, wget, …) will accept your certificates but **Chrome** and **Mozilla** browsers require you to upload the root certificate to their own repositories. Open site **about:preferences#privacy** in Mozilla browser and click the button "**View certificates…**". Click on button "**Import…**" and import the CA.crt file.

Open site **chrome://settings/certificates** in Chrome browser and click on "**Import**" button. Select the **CA.pfx** file. Open site **chrome://flags/#allow-insecure-localhost** after that. Enable button "**Allow invalid certificates for resources loaded from localhost.**".

1. Being in CA directory create **localhost** subfolder with **localhost.ext** inside:

mkdir localhost // New project

cd localhost

touch localhost.ext

Windows users have no the **touch** command. Use command **type** in this case:

type nul > localhost.ext

1. Write this information into localhost.ext:

authorityKeyIdentifier = keyid,issuer

basicConstraints = CA:FALSE

keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment

subjectAltName = @alt\_names

[alt\_names]

IP.1 = 127.0.0.1

IP.1 = 127.0.0.1

IP.2 = 192.168.1.107

DNS.1 = localhost

DNS.2 = KIBIRVIBIR-PC

DNS.3 = gediminas-TravelMate-5760Z

Parameters **localhost** and **127.0.0.1** are mandatory, other adresses are optional. An example contains settings from my home network. One can decide to add more domains or IP addresses to the file but ensure to edit the **/etc/hosts** file to have those domains point to the local machine (127.0.0.1). Windows users will find the **hosts** file in C:\**Windows\System32\drivers\etc** directory.

1. Generate private key with command:

openssl genrsa -out localhost.key -des3 3072

1. Generate CSR (signing request file) using the key:

openssl req -new -key localhost.key -out localhost.csr -config ../CA.CNF

Windows users can generate the CSR file without config:

openssl req -new -key localhost.key -out localhost.csr \

-subj "/C=LT/ST=N\_A/L=Kaunas/O=Home Computer/OU=IT/CN=localhost"

1. With this CSR, we can request the CA to sign a certificate as below. Since we are creating the certificate on the localhost computer, we will use our own Certificated Authority (CA.\* files). Note that the paths for **CA.key** and **CA.pem** files are dependent on where the user is running commands from. In this case, the commands below are ran from **/cert/CA/localhost**:

openssl x509 -req -in localhost.csr -CA ../CA.pem -CAkey ../CA.key \

-CAcreateserial -days 3650 -sha256 -extfile localhost.ext -out localhost.crt

1. The server will need the localhost.crt certificate file, and the decrypted key since our localhost.key is in encrypted form. Decrypt the localhost.key and store that file:

openssl rsa -in localhost.key -out localhost.decrypted.key

1. Windows users can import root certificate with **mmc** snap-in. Run mmc, select **File → Add/Remove Snap-in → Certificates → My user account → OK.** Open the new snap-in and select **Trusted Root Certification Authorities**. Right click on  **Certificates → All Tasks → Import…** Import the CA.pfx file from CA directory. Launch WEB server with ListenAndServeTLS function. Any browser would accept [https://localhost:8080](https://localhost:8080/) address. Sometimes this URL is occupied by site registered on your local IIS. Stop your local IIS and repeat the test. Chrome browser will inform you that the certificate is insecure but the site will be displayed.

#### Serving Static Files and Sites

The net/http package has a **FileServer** function that allows you to display static pages:

package main

import (

"log"

"net/http"

)

func main() {

fs := http.**FileServer**(http.Dir("./static"))

http.Handle("/", fs)

log.Print("Listening on :3000...")

err := http.ListenAndServe(":3000", nil)

if err != nil {

log.Fatal(err)

}

}

The provided file server will show any file from the ./static directory. Just type the command

localhost:3000/file\_name

in a browser’s address window. Put this an **example.html** file into the **./static** directory for testing the application

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width,initial-scale=1">

<title>A static page</title>

<link rel="stylesheet" href="styles/main.css">

</head>

<body>

<h1>Hello from static page</h1>

</body>

</html>

The page requires stylesheet in the styles subdirectory. Use this one:

body {

color: #c0392b

}

Structure of the project:

**home**

go.mod

main.go

**static**

example.html

**styles**

main.css

Name of the directories are typed in bold. The browser’s command localhost:3000/example.html will show you text „Hello from static page“. Alex Edwards explained this [example](https://www.alexedwards.net/blog/serving-static-sites-with-go).

**http.FileServer** is also used in dynamic sites. It serves **CSS files**, **images**, and other objects that can be downloaded by the client. For that purpose, you have to use the request multiplexer:

func main() {

**fs := http.FileServer(http.Dir("public"))**

**http.Handle("/public/", http.StripPrefix("/public/", fs))**

http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {

...

})

http.HandleFunc("/abc", func(w http.ResponseWriter, r \*http.Request) {

...

})

...

http.ListenAndServe(":8080", nil)

}

In this example, the static files are placed in the "public" directory. The file server will handle all requests starting with "/public/" :

<link rel="stylesheet" href="public/css/format.css">

The file server's root directory can have its own subdirectories, and its name can differ from the pattern specified in the http.Handle function. A complete example of such WEB application is provided at the [zetcode](https://zetcode.com/golang/http-serve-static-files/) site

#### Text templates

Any Go program can use the [text/template](https://pkg.go.dev/text/template) or [html/template](https://pkg.go.dev/html/template) package—both included in the Go standard library. Both packages allow you to write textual templates and pass data into them to render a document formatted to your liking. Package template (**html/template**) implements data-driven templates for generating HTML output safe against code injection. It provides the same interface as **text/template** and should be used instead of text/template whenever the output is HTML. Within the templates, you can loop over the data and use conditional logic to decide which items to include in the document and how they should appear. Templates live either as strings in your code or in their own files alongside your code. They contain boilerplate static text interlaced with conditional statements (i.e. ), flow control statements (i.e. loops), and function calls, all wrapped within {{. . .}} markers. You will pass some data into your template to render a final document.

One must follow this instruction working with templates:

1. import package **text/template** or **html/template,**
2. create a template using the operator **template.New(template\_name)**
3. compile the template using the **Parse()** function, this function has several modifications (**ParseFS**, **ParseFiles**, **ParseGlob**),
4. execute the compiled template using the **Exec()** function.

A complete description of the tex/template package is available at the [GO](https://pkg.go.dev/text/template" \l "pkg-overview) site. Here is an example of this scenario:

// file main.go

// https://www.digitalocean.com/community/tutorials/how-to-use-templates-in-go

package main

import (

"os"

"text/template"

)

type **Pet** struct {

Name string

Sex string

Intact bool

Age string

Breed string

}

func main() {

**dogs** := []Pet{

{

Name: "Jutube",

Sex: "Female",

Intact: false,

Age: "10 months",

Breed: "German Sheperd/Pitbul",

},

{

Name: "Zephyr",

Sex: "Male",

Intact: true,

Age: "13 years, 3 months",

Breed: "German Sheperd/Border Collie",

},

{

Name: "Bruce Wayne",

Sex: "Male",

Intact: false,

Age: "3 years, 8 months",

Breed: "Chihuahua",

},

}

var tmplFile = "pets.tmpl"

tmpl, err := template.**New**(tmplFile).**ParseFiles**(tmplFile)

if err != nil {

panic(err)

}

err = tmpl.**Execute**(os.Stdout, dogs)

if err != nil {

panic(err)

}

}

The template may be written as string contant. It is in a separate file here:

{{/\* file pets.tmpl \*/}}

Number of dogs: {{ . | len -}}

{{ range . }}

---

Name: {{ .Name }}

Sex: {{ .Sex }} ({{ if .Intact }}intact{{ else }}{{ if (.Sex | eq "Female") }}spayed{{ else }}neutered{{end}}{{ end }})

Age: {{ .Age }}

Breed: {{ .Breed }}

{{ end }}

The input text for a template is UTF-8-encoded text in any format. "**Actions**"--data evaluations or control structures--are delimited by **"{{"** and **"}}"**; all text outside actions is copied to the output unchanged. Write "- " (minus with space) immediately after the opening delimiter or " -" (space with minus) before the closing delimiter if you want to trim spaces before or after the action. **Actions** represent the data evaluations, functions or control loops. They access the data via the cursor, denoted by a dot (.). Initial value for the cursor supplies second parameter in the **Execute** command:

err = tmpl.**Execute**(os.Stdout, dogs)

Slice **dogs** is cursor for the template. The data passed into a template may be absolutely anything, but it is common to pass in a slice, array, or map—something iterable.

Arguments

An argument is a simple value, denoted by one of the following:

* A **boolean**, **string**, **character**, **integer**, **floating-point**, **imaginary** or **complex constant** in Go syntax. These behave like Go's untyped constants.
* The keyword **nil**, representing an untyped Go nil.
* The character '.' (period): . The result is the current value of the dot.
* A variable name, which is a (possibly empty) alphanumeric string preceded by a dollar sign, such as $piOver2 or $ The result is the value of the variable.
* The name of a field of the data, which must be a **struct**, preceded by a period, such as **.Field**; the result is the value of the field. Field invocations may be chained: .Field1.Field2 Fields can also be evaluated on variables, including chaining: $x.Field1.Field2.
* The name of a key of the data, which must be a map, preceded by a period, such as .Key The result is the map element value indexed by the key. Key invocations may be chained and combined with fields to any depth: .Field1.Key1.Field2.Key2 Although the key must be an alphanumeric identifier, unlike with field names they do not need to start with an upper case letter. Keys can also be evaluated on variables, including chaining: **$x.key1.key2**.
* The name of a niladic method of the data, preceded by a period, such as .Method The result is the value of invoking the method with dot as the receiver, dot.Method(). Such a method must have one return value (of any type) or two return values, the second of which is an error. If it has two and the returned error is non-nil, execution terminates and an error is returned to the caller as the value of Execute. Method invocations may be chained and combined with fields and keys to any depth: .Field1.Key1.Method1.Field2.Key2.Method2 Methods can also be evaluated on variables, including chaining: $**x.Method1.Field**.
* The name of a niladic function defined inside the template, such as fun The result is the value of invoking the function, fun(). The return types and values behave as in methods.
* A parenthesized instance of one the above, for grouping.

Arguments may evaluate to any type; if they are pointers the implementation automatically indirects to the base type when required. If an evaluation yields a function value, such as a function-valued field of a struct, the function is not invoked automatically, but it can be used as a truth value for an if action and the like.

Pipeline

A pipeline is a possibly chained sequence of "commands". A command is a simple value (argument) or a function or method call, possibly with multiple arguments:

**Argument**

The result is the value of evaluating the argument.

**.Method [Argument...]**

The method can be alone or the last element of a chain but,

unlike methods in the middle of a chain, it can take arguments.

The result is the value of calling the method with the

arguments:

dot.Method(Argument1, etc.)

**functionName [Argument...]**

The result is the value of calling the function associated

with the name:

function(Argument1, etc.)

A pipeline may be "chained" by separating a sequence of commands with pipeline characters '|'. In a chained pipeline, the result of each command is passed as the last argument of the following command. The output of the final command in the pipeline is the value of the pipeline. The output of a command will be either one value or two values, the second of which has type error. If that second value is present and evaluates to non-nil, execution terminates and the error is returned to the **caller** of Execute.

Variable

A pipeline inside an action may initialize a variable to capture the result:

$variable := pipeline

where $variable is the name of the variable. An action that declares a variable produces no output. Variables previously declared can also be assigned, using the syntax

$variable = pipeline

If a "range" action initializes a variable, the variable is set to the successive elements of the iteration. Also, a "range" may declare two variables, separated by a comma:

range $element := pipeline

range $index, $element := pipeline

A variable's scope extends to the "end" action of the control structure ("if", "with", or "range") in which it is declared, or to the end of the template if there is no such control structure. A template invocation does not inherit variables from the point of its invocation.

When execution begins, $ is set to the data argument passed to Execute, that is, to the starting value of dot.

Comments

The parser ignores comments. The comment may contain newlines. Comments do not nest and must start and end at the delimiters:

{{/\* a comment \*/}}

{{- /\* a comment with white space trimmed from preceding and following text \*/ -}}

Values

{{pipeline}}

The default textual representation (the same as would be printed by fmt.Print) of the value of the pipeline is copied to the output.

Conditionals

{{if \_cond\_pipeline\_}} T1 {{end}}

If the value of the pipeline is empty, no output is generated; otherwise, T1 is executed. The empty values are false, 0, any nil pointer or interface value, and any array, slice, map, or string of length zero. Dot is unaffected.

{{if \_cond\_pipeline\_}} T1 {{else}} T0 {{end}}

If the value of the pipeline is empty, T0 is executed; otherwise, T1 is executed. Dot is unaffected.

{{if \_cond\_pipeline\_}} T1 {{else if \_cond\_pipeline\_}} T0 {{end}}

This note is in abbreviated form. It means exactly the same as this action:

**{{if \_cond\_pipeline\_}} T1 {{else}}{{if \_cond\_pipeline\_}} T0 {{end}}{{end}}**

The **\_cond\_pipeline\_** can be a **boolean field** or a **function** that produces a boolean result. The parser has the following boolean functions:

**eq** Returns the boolean truth of arg1 == arg2

**ne** Returns the boolean truth of arg1 != arg2

**lt** Returns the boolean truth of arg1 < arg2

**le** Returns the boolean truth of arg1 <= arg2

**gt** Returns the boolean truth of arg1 > arg2

**ge** Returns the boolean truth of arg1 >= arg2

**and** Returns the boolean AND of its arguments

**or** Returns the boolean OR of its arguments

**not** Returns the boolean negation of its single argument

Loops

{{range pipeline}} T1 {{end}}

The value of the pipeline must be an array, slice, map, or channel. If the value of the pipeline has length zero, nothing is output; otherwise, dot is set to the successive elements of the array, slice, or map and T1 is executed. If the value is a map and the keys are of basic type with a defined order, the elements will be visited in sorted key order.

{{range pipeline}} T1 {{else}} T0 {{end}}

The value of the pipeline must be an array, slice, map, or channel. If the value of the pipeline has length zero, dot is unaffected and T0 is executed; otherwise, dot is set to the successive elements of the array, slice, or map and T1 is executed.

{{break}}

The innermost {{range pipeline}} loop is ended early, stopping the current iteration and bypassing all remaining iterations.

{{continue}}

The current iteration of the innermost {{range pipeline}} loop is stopped, and the loop starts the next iteration.

A name of the template

The template can be named using:

1. Function **[template.New](https://pkg.go.dev/text/template" \l "New)**. The function allocates a new, undefined template with the given name.

template.New(name string) \*Template

1. Function [**template.ParseFiles**](https://cs.opensource.google/go/go/+/go1.21.1:src/text/template/helper.go;l=40). ParseFiles creates a new Template and parses the template definitions from the named files. The returned template's name will have the base name and parsed contents of the first file. There must be at least one file. If an error occurs, parsing stops and the returned \*Template is nil.

template.ParseFiles(filenames ...string) (\*Template, error)

1. Writing action **define** inside the template:

{{**define** "footer"}} // Name of the template is „footer“

<footer>

<p>

Copyright 2016 Calhoun.io

</p>

<p>

Contact information: <a href="mailto:jon@calhoun.io">jon@calhoun.io</a>.

</p>

</footer>

{{**end**}}

Execute template

{{template "name"}}

The template with the specified name is executed with nil data.

{{template "name" pipeline}}

The template with the specified name is executed with dot set to the value of the pipeline.

Define and execute template

{{block "name" pipeline}} T1 {{end}}

A block is shorthand for defining a template

**{{define "name"}} T1 {{end}}**

and then executing it in place

**{{template "name" pipeline}}**

Change a value of the dot

{{with pipeline}} T1 {{else}} T0 {{end}}

If the value of the pipeline is empty, dot is unaffected and T0 is executed; otherwise, dot is set to the value of the pipeline and T1 is executed.

Functions

The template engine has a wide list of embedded functions. This list is published on [GO](https://pkg.go.dev/text/template" \l "hdr-Functions) site. During execution functions are found in two function maps: first in the template, then in the global function map. By default, no functions are defined in the template but the Funcs method can be used to add them.

The template can call the function described in a context:

// file main.go

type ViewData struct {

User User

}

type User struct {

ID int

Email string

}

func (u User) HasPermission(feature string) bool {

if feature == "feature-a" {

return true

} else {

return false

}

}

...

// file hello.html; Context is ViewData

{{if **.User.HasPermission** "feature-a"}}

<div class="feature">

<h3>Feature A</h3>

<p>Some other stuff here...</p>

</div>

{{else}}

<div class="feature disabled">

<h3>Feature A</h3>

<p>To enable Feature A please upgrade your plan</p>

</div>

{{end}}

It is necessary to use action **call** when context has property of a **func** type:

{{if (call .User.HasPermission "feature-a")}}

<div class="feature">

<h3>Feature A</h3>

<p>Some other stuff here...</p>

</div>

{{else}}

<div class="feature disabled">

<h3>Feature A</h3>

<p>To enable Feature A please upgrade your plan</p>

</div>

{{end}}

Custom functions can be placed in a dictionary of type **map[string]any,** see description of [FuncMap](https://pkg.go.dev/text/template" \l "FuncMap) function:

func main() {

var err error

testTemplate, err = template.New("hello.gohtml").**Funcs**(**template.FuncMap**{

"**hasPermission**": func(user User, feature string) bool {

return user.ID == 1 && feature == "feature-a"

},

}).ParseFiles("hello.gohtml")

if err != nil {

panic(err)

}

http.HandleFunc("/", handler)

http.ListenAndServe(":3000", nil)

}

You don’t need the call action in this case:

{{if hasPermission .User "feature-a"}}

...

{{if hasPermission .User "feature-b"}}

...

Template functions are described on a [calhoun](https://www.calhoun.io/intro-to-templates-p3-functions/) site.

### Gorilla mux

Package gorilla/mux implements a request router and dispatcher for matching incoming requests to their respective handler.

* It implements the http.Handler interface so it is compatible with the standard http.ServeMux.
* Requests can be matched based on URL host, path, path prefix, schemes, header and query values, HTTP methods or using custom matchers.
* URL hosts, paths and query values can have variables with an optional regular expression.
* Registered URLs can be built, or "reversed", which helps maintaining references to resources.
* Routes can be used as subrouters: nested routes are only tested if the parent route matches. This is useful to define groups of routes that share common conditions like a host, a path prefix or other repeated attributes. As a bonus, this optimizes request matching.

Install the package with **go get**:

go get -u github.com/gorilla/mux

You can use GorillaMux exactly like http.ServeMux:

func main() {

r := mux.NewRouter()

r.HandleFunc("/", HomeHandler)

r.HandleFunc("/products", ProductsHandler)

r.HandleFunc("/articles", ArticlesHandler)

http.ListenAndServe(":3000", r)

}

Here we register three routes mapping URL paths to handlers. This is equivalent to how http.HandleFunc() works: if an incoming request URL matches one of the paths, the corresponding handler is called passing (http.ResponseWriter, \*http.Request) as parameters.

Paths can have variables. They are defined using the format {name} or {name:pattern}. If a regular expression pattern is not defined, the matched variable will be anything until the next slash. For example:

r := mux.NewRouter()

r.HandleFunc("/products/{key}", ProductHandler)

r.HandleFunc("/articles/{category}/", ArticlesCategoryHandler)

r.HandleFunc("/articles/{category}/{id:[0-9]+}", ArticleHandler)

The names are used to create a map of route variables which can be retrieved calling mux.Vars():

func ArticlesCategoryHandler(w http.ResponseWriter, r \*http.Request) {

vars := mux.Vars(r)

w.WriteHeader(http.StatusOK)

fmt.Fprintf(w, "Category: %v\n", vars["category"])

}

Routes can be restricted using predefined functions:

* **Domain or subdomain**

r := mux.NewRouter()

r.Host("[www.example.com](http://www.example.com/)") // Matches if domain is "www.example.com".

r.Host("{subdomain:[a-z]+}.example.com") // Matches a dynamic subdomain.

* **Path prefix** of the requests

r.PathPrefix("/products/")

* **HTTP methods**

r.Methods("GET", "POST")

* **URL schemes**

r.Schemes("https")

* **Header values**

r.Headers("X-Requested-With", "XMLHttpRequest")

Regex support also exists for matching Headers within a route:

r.HeadersRegexp("Content-Type", "application/(text|json)")

This route will match both requests with a **Content-Type** of **application/json** as well as **application/text**.

* **Query values**

r.Queries("key", "value")

* **Custom matcher function** which returns true or false

r.MatcherFunc(func(r \*http.Request, rm \*RouteMatch) bool {

return r.ProtoMajor == 0

})

It is possible to combine several matchers in a single route:

r.HandleFunc("/products", ProductsHandler).

Host("www.example.com").

Methods("GET").

Schemes("http")

Routes are tested in the order they were added to the router. If two routes match, the first one wins:

r := mux.NewRouter()

r.HandleFunc("/specific", specificHandler)

r.PathPrefix("/").Handler(catchAllHandler)

The **Subrouter()** function allows you to group routing rules. Selection conditions may be specified for this function. These conditions will be applied to all members of the group:

r := mux.NewRouter()

s := r.Host("www.example.com").Subrouter()

s.HandleFunc("/products/", ProductsHandler)

s.HandleFunc("/products/{key}", ProductHandler)

s.HandleFunc("/articles/{category}/{id:[0-9]+}", ArticleHandler)

The three URL paths we registered above will only be tested if the domain is [**www.example.com**](http://www.example.com/). When a subrouter has a path prefix, the inner routes use it as base for their paths:

r := mux.NewRouter()

s := r.PathPrefix("/products").Subrouter()

s.HandleFunc("/", ProductsHandler) // "/products/"

s.HandleFunc("/{key}/", ProductHandler) // "/products/{key}/"

// "/products/{key}/details"

s.HandleFunc("/{key}/details", ProductDetailsHandler)

Host and path may be defined separately use subrouters:

r := mux.NewRouter()

s := r.Host("{subdomain}.example.com").Subrouter()

s.Path("/articles/{category}/{id:[0-9]+}").

HandlerFunc(ArticleHandler).

Name("article")

Path provided to **PathPrefix()** represents a "wildcard": calling **PathPrefix("/static/").Handler(...)** means that the handler will be passed any request that matches "**/static/\***". This makes it easy to serve static files with mux:

func main() {

var dir string

flag.StringVar(&dir, "dir", ".", "the directory to serve files from. Defaults to the current dir")

flag.Parse()

r := mux.NewRouter()

// This will serve files under http://localhost:8000/static/<filename>

r.PathPrefix("/static/").Handler(http.StripPrefix("/static/", http.FileServer(http.Dir(dir))))

srv := &http.Server{

Handler: r,

Addr: "127.0.0.1:8000",

// Good practice: enforce timeouts for servers you create!

WriteTimeout: 15 \* time.Second,

ReadTimeout: 15 \* time.Second,

}

log.Fatal(srv.ListenAndServe())

}

#### Named routes and redirection

Routes can be named. All routes that define a name can have their URLs built, or "**reversed**". We define a name calling **Name()** on a route:

r := mux.NewRouter()

r.HandleFunc("/articles/{category}/{id:[0-9]+}", ArticleHandler).

**Name("article")**

To build a URL, get the route and call the URL() method, passing a sequence of key/value pairs for the route variables:

url, err := r.**Get("article")**.URL("category", "technology", "id", "42")

This request result will produce a **url.URL** with the following path:

"/articles/technology/42"

This also works for host and query value variables:

r := mux.NewRouter()

r.Host("{subdomain}.domain.com").

Path("/articles/{category}/{id:[0-9]+}").

Queries("filter", "{filter}").

HandlerFunc(ArticleHandler).

Name("article")

Request

url, err := r.Get("article").URL("subdomain", "news",

"category", "technology",

"id", "42", "filter", "gorilla")

will return **url.URL** with the following path:

"http://news.domain.com/articles/technology/42?filter=gorilla"

All variables defined in the route are required, and their values must conform to the corresponding patterns. These requirements guarantee that a generated URL will always match a registered route.

There's also a way to build only the URL host or path for a route: use the methods **URLHost()** or **URLPath()** instead of **URL()**:

r := mux.NewRouter()

r.Host("{subdomain}.example.com").

Path("/articles/{category}/{id:[0-9]+}").

Queries("filter", "{filter}").

HandlerFunc(ArticleHandler).

Name("article")

...

// "http://news.example.com/"

host, err := r.Get("article").URLHost("subdomain", "news")

// "/articles/technology/42"

path, err := r.Get("article").URLPath("category", "technology", "id", "42")

To find all the required variables for a given route when calling URL(), the method GetVarNames() is available:

r := mux.NewRouter()

r.Host("{domain}").

Path("/{group}/{item\_id}").

Queries("some\_data1", "{some\_data1}").

Queries("some\_data2", "{some\_data2}").

Name("article")

// Will print [domain group item\_id some\_data1 some\_data2] <nil>

fmt.Println(r.Get("article").GetVarNames())

The **Walk** function on **mux.Router** can be used to visit all of the routes that are registered on a router. An example of this function you will find on site [https://pkg.go.dev/github.com/gorilla/mux#section-readme](https://pkg.go.dev/github.com/gorilla/mux" \l "section-readme).

### Graceful shutdown http server

The document "03\_GoLanguage.docx" shows how to stop the WEB server, see to the "Signals" section. Here we will show you a slightly different way that is compatible with the Gorilla.mux package:

package main

import (

"context"

"log"

"net/http"

"os"

"os/signal"

"syscall"

"time"

"github.com/gorilla/mux" // need to use dep for package management

)

func TestEndpoint(w http.ResponseWriter, r \*http.Request) {

w.WriteHeader(200)

w.Write([]byte("Test is what we usually do"))

}

func main() {

router := mux.NewRouter()

router.HandleFunc("/test", TestEndpoint).Methods("GET")

srv := &http.Server{

Addr: ":8080",

Handler: router,

}

done := make(chan os.Signal, 1)

signal.Notify(done, os.Interrupt, syscall.SIGINT, syscall.SIGTERM)

go func() {

if err := srv.ListenAndServe(); err != nil && err != http.ErrServerClosed {

log.Fatalf("listen: %s\n", err)

}

}()

log.Print("Server Started")

<-done

log.Print("Server Stopped")

ctx, cancel := context.WithTimeout(context.Background(), 5\*time.Second)

defer func() {

// extra handling here

cancel()

}()

if err := srv.Shutdown(ctx); err != nil {

log.Fatalf("Server Shutdown Failed:%+v", err)

}

log.Print("Server Exited Properly")

}

**srv.ListenAndServe()** may be launched on main thread but this function is blocking and capturing of the <-done channel must be performed inside endless go-routine. Document **03\_GoLanguage.docx** demonstrates how to capture a value from <-done chanel when **srv.ListenAndServe()** is on main thread. Pushing **srv.ListenAndServe()** into the go-routine simplifies handling of the channel.

**Serve** accepts incoming **HTTP** connections on the listener l, creating a **new service goroutine** for each. The service goroutines read requests and then call handler to reply to them.

if err := srv.Shutdown(ctx); err != nil {

log.Fatalf("Server Shutdown Failed:%+v", err)

}

**Shutdown gracefully shuts down the server without interrupting any active connections. Shutdown works by first closing all open listeners, then closing all idle connections, and then waiting indefinitely for connections to return to idle and then shut down. If the provided context expires before the shutdown is complete, Shutdown returns the context’s error, otherwise it returns any error returned from closing the Server’s underlying Listener(s).**

The quote above came from Go documentation said “and then waiting indefinitely for connections to return to idle and then shut down”, this part you should worry because it means some zombie connections may still be there and use your memory, in order to solve this, the safest way is to setup a timeout threshold using the Context:

ctx, cancel := context.WithTimeout(context.Background(), 5\*time.Second)

...

if err := srv.Shutdown(ctx); err != nil {

...

}

Large system will have http server, usually combined with multiple other connections like database, message queues, etc. And we need to handle these properly as well by defining the defer func and using cancel() to release these resources:

defer func() {

// Close database, redis, truncate message queues, etc.

cancel()

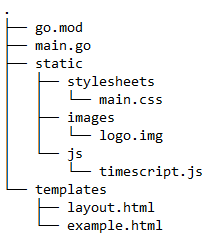
}()

// [Graceful shutdown in Go http server | by Sam Wang | honestbee-tw-engineering | Medium](https://medium.com/honestbee-tw-engineer/gracefully-shutdown-in-go-http-server-5f5e6b83da5a)

// [mux package - github.com/gorilla/mux - Go Packages](https://pkg.go.dev/github.com/gorilla/mux" \l "section-readme)

### Serving a complete web application

Consider this structure of the WEB application:



Here we have 2 problems:

1. specify the correct address for loading css, js and image files,
2. pass the parameter to the template parser.

#### With Http package

Problems are solved quite simply by working with a standard router (with http package, without ServeMux or GorillaMux):

1. Requests starting with the word "static" are directed to the file server:

fs := http.FileServer(http.Dir("./static"))

http.Handle("/static/", http.StripPrefix("/static/", fs))

After providing these settings, a link to the CSS file will look like this:

<link rel="stylesheet" type="text/css" href="**/static/stylesheets/main.css**">

In the first parameter of the http.Handle() function, you specify a name of the route. The parameter in the http.FileServer() function defines the name of the directory., These names match in the given example, but they can be different.

1. The second parameter of the **http.Handle** function is the **http.Handler** interface. Any object that implements this interface can be passed there. **http.Handler** interface has only one function - **ServeHTTP**. The following example demonstrates passing parameters with object:

type LayoutStruct struct {

StaticDir string

ListenPort string

Subject string

}

func (cnt LayoutStruct) ServeHTTP(w http.ResponseWriter, r \*http.Request) {

lp := filepath.Join("templates", "layout.html")

fp := filepath.Join("templates", filepath.Clean(r.URL.Path))

tmpl, err := template.ParseFiles(lp, fp)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

err = tmpl.ExecuteTemplate(w, "layout", cnt)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

}

func main() {

port := flag.String("p", "3000", "port to serve on")

flag.Parse()

progSettings := LayoutStruct{"static", \*port, "SqlLedger"}

fs := http.FileServer(http.Dir("./static"))

http.Handle("/static/", http.StripPrefix("/static/", fs))

http.Handle("/", progSettings)

log.Printf("Serving %s on HTTP port: %s\n",

progSettings.StaticDir, progSettings.ListenPort)

log.Fatal(http.ListenAndServe(fmt.Sprintf(":%s", progSettings.ListenPort), nil))

}

1. Parameters can also be passed to the **HandleFunc()** function, but in this case, the second parameter must specify a function that returns the **func(ResponseWriter, \*Request)** function:

func CreateHTTPHandler(cnt LayoutStruct)

func(w http.ResponseWriter, r \*http.Request) {

return func(w http.ResponseWriter, r \*http.Request) {

lp := filepath.Join("templates", "layout.html")

fp := filepath.Join("templates", filepath.Clean(r.URL.Path))

info, err := os.Stat(fp)

if err != nil {

if os.IsNotExist(err) {

http.NotFound(w, r)

return

}

}

if info.IsDir() {

http.NotFound(w, r)

return

}

tmpl, err := template.ParseFiles(lp, fp)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

err = tmpl.ExecuteTemplate(w, "layout", cnt)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

}

}

Call this function in **http.HandleFunc**:

http.HandleFunc("/", CreateHTTPHandler(progSettings))

#### With GorillaMux

All requests must travel through this router's PathPrefix when you are working with a Gorilla router. The main function is slightly different:

func main() {

port := flag.String("p", "3000", "port to serve on")

flag.Parse()

progSettings := LayoutStruct{"static", \*port, "SqlLedger"}

route := mux.NewRouter()

fs := http.FileServer(http.Dir("./static/"))

**route.PathPrefix("/static/")**.Handler(http.StripPrefix("/static/", fs))

**route.PathPrefix("/app")**.HandlerFunc(CreateHTTPHandler(progSettings))

log.Printf("Serving %s on HTTP port: %s\n", progSettings.StaticDir,

progSettings.ListenPort)

log.Fatal(http.ListenAndServe(fmt.Sprintf(":%s", progSettings.ListenPort), route))

}

The **filepath.Clean(r.URL.Path)** will return **app/example.html** after sending the **localhost:3000/app/example.html** request. Therefore, you need to push the **example.html** file into the **app** subdirectory or hardcode a name of the file. The second choice simplifies logic of the handler:

func CreateHTTPHandler(cnt LayoutStruct) http.HandlerFunc {

return func(w http.ResponseWriter, r \*http.Request) {

fp := filepath.Join("templates", "example.html")

lp := filepath.Join("templates", "layout.html")

tmpl, err := template.ParseFiles(lp, fp)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

err = tmpl.ExecuteTemplate(w, "layout", cnt)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

}

}

Gorilla mux also allows you to use an object with the ServeHTTP function:

type LayoutStruct struct {

StaticDir string

ListenPort string

Subject string

}

func (cnt LayoutStruct) ServeHTTP(w http.ResponseWriter, r \*http.Request) {

lp := filepath.Join("templates", "layout.html")

fp := filepath.Join("templates", "example.html")

tmpl, err := template.ParseFiles(lp, fp)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

err = tmpl.ExecuteTemplate(w, "layout", cnt)

if err != nil {

log.Print(err.Error())

http.Error(w, http.StatusText(500), 500)

return

}

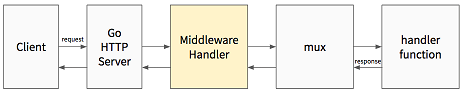
}

In this case, you specify name of this object in the routing table:

route.PathPrefix("/app").Handler(progSettings)

### Middleware handler

A middleware handler is an http.Handler that wraps another http.Handler to do some pre- and/or post-processing of the request. It's called "middleware" because it sits in the middle between the Go web server and the actual handler.



#### Name/password login

[How to correctly use Basic Authentication in Go – Alex Edwards](https://www.alexedwards.net/blog/basic-authentication-in-go" \l "protecting-a-web-application)

[Middleware Patterns in Go (drstearns.github.io)](https://drstearns.github.io/tutorials/gomiddleware/)

[Making and Using HTTP Middleware – Alex Edwards](https://www.alexedwards.net/blog/making-and-using-middleware)

## WEB Components

A Web Component is a way to create an encapsulated, single-responsibility code block that can be reused on any page. It plays the same role as a function in other programming languages. You can make a nice page header, name it **page-header** and use it on all pages just by writing a single element

<page-header></page-header>

The WEB component consists of three parts:

* Custom element,
* HTML template,
* Shadow DOM

#### Custom element

Here is an example of the hello-world custom element (file **HelloWorld.js**):

class HelloWorld extends HTMLElement {

connectedCallback() {

this.textContent = 'Hello World';

}

}

customElements.define('hello-world', HelloWorld);

Each custom element is an ES6 class that must be binded to the **CustomElementRegistry**:

customElements.define('hello-world', HelloWorld);

The first parameter of this operator specifies a name of the element. By convention, it must be two words joined by a minus sign. The second parameter is a name of the class. Again, by convention, one must use the same words as were used in a name of custom element.

Custom element also may be defined using anonymous class

customElements.define('input-plus-minus', **class extends HTMLElement {**

constructor() {

super();

// rest of the Javascript

}

// properties and methods of your custom element

});

After executing the '**customElements.define**' command, you can place the custom element into the page. Here is an example of the **Autonomous custom elements**, the class inherits from [**HTMLElement**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement) in this case. Loading an element into a page is exactly the same as any other JS file:

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

**<script type="module" src="./helloworld.js"></script>**

</head>

<body>

<h1>This is an example of the custom element</h1>

**<hello-world></hello-world>**

</body>

</html>

**Type="module"** may be omitted in the **script** element, but then add the **deferred** attribute:

<script src="./helloworld.js" defer></script>

Custom elements cannot be self-closing because HTML only allows a few elements to be self-closing. Always write a closing tag (<app-drawer></app-drawer>).

<hello-world/> <!-- **ERROR** -->

<hello-world></hello-world> <!-- Correct -->

The form of two elements allows you to pass additional information into the component:

<hello-world>This text will be passed to the component</hello-world>

The functionality of a custom element is defined using an ES2015 [**class**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes) which extends HTMLElement. Extending HTMLElement ensures the custom element inherits the entire DOM API and means any properties/methods that you add to the class become part of the element's DOM interface. Essentially, use the **class** to create a public JavaScript API for your tag.

class AppDrawer extends HTMLElement {

// A getter/setter for an open property.

**get open() {**

return this.hasAttribute('open');

}

**set open(val) {**

// Reflect the value of the open property as an HTML attribute.

if (val) {

this.setAttribute('open', '');

} else {

this.removeAttribute('open');

}

this.toggleDrawer();

}

// A getter/setter for a disabled property.

**get disabled() {**

return this.hasAttribute('disabled');

}

**set disabled(val) {**

// Reflect the value of the disabled property as an HTML attribute.

if (val) {

this.setAttribute('disabled', '');

} else {

this.removeAttribute('disabled');

}

}

// Can define constructor arguments if you wish.

**constructor()** {

// If you define a constructor, always call super() first!

// This is specific to CE and required by the spec.

super();

// Setup a click listener on <app-drawer> itself.

this.addEventListener('click', e => {

// Don't toggle the drawer if it's disabled.

if (this.disabled) {

return;

}

this.toggleDrawer();

});

}

**toggleDrawer()** {

// ...

}

}

customElements.define('app-drawer', AppDrawer);

A neat feature of custom elements is that **this inside a class definition refers to the DOM element itself** i.e. the instance of the class. In our example, this refers to <app-drawer>.

WEB component allows you to create **Customized built-in elements**. The base class can be any other element in that case.

class WordCount **extends HTMLParagraphElement** {

constructor() {

// Always call super first in constructor

super();

// Element functionality written in here

}

}

customElements.define(**'word-count'**, WordCount, { **extends: 'p'** });

In this case it is necessary to specify the base element when registering the component (see the object in the third parameter). In an HTML page, you specify the **base element** with the **is** attribute:

<p **is="word-count"**></p>

The custom element may be created using JavaScript. Use

document.createElement("hello-world")

**Customized built-in element** requires a bit different operator:

document.createElement("p", { is: "word-count" })

Note that the first parameter specifies the **base** element and object in the second parameter defines the actual element.

#### Lifecycle of the custom element

Examples in the previous paragraph show two very important methods of custom element: **constructor()** and **connectedCallback()**. These methods are called at different time:

* **constructor()** is called when the element is created; the constructor should be used to set up initial state and default values, to set up event listeners and possibly a shadow root; You can't add Nodes inside the normal DOM, and you can't add or set an attribute either.
* **connectedCallback()** is called when (after) the element is attached to the DOM.
* **disconnectedCallback()** is called every time the element is removed from the DOM. Useful for running clean up code.
* **attributeChangedCallback(attrName, oldVal, newVal)** is called when an observed attribute has been added, removed, updated, or replaced. Also called for initial values when an element is created by the parser, or upgraded. Note: only attributes listed in the [observedAttributes](https://codepen.io/ratchasak/pen/ggxmgX) property will receive this callback.
* **adoptedCallback** is called when the custom element has been moved into a new document (e.g. someone called [document.adoptNode(el)](https://developer.mozilla.org/en-US/docs/Web/API/Document/adoptNode)).

Reaction callbacks are synchronous. If someone calls el.setAttribute() on your element, the browser will immediately call attributeChangedCallback(). Similarly, you'll receive a disconnectedCallback() right after your element is removed from the DOM (e.g. the user calls el.remove()). But be careful! You can't rely on your element being removed from the DOM in all circumstances. For example, disconnectedCallback() will never be called if the user closes the tab.

Definion

A custom element is defined when customElements.define is called:

class HelloWorld extends HTMLElement {

connectedCallback() {

this.textContent = 'Hello World';

}

}

customElements.define('hello-world', HelloWorld);

Defining an element doesn't trigger either the constructor or the connectedCallback methods since it does not create an instance of an element. **An element can only be defined once.**

Create

An element can be created in JavaScript in two ways:

// can happen before definition

const myElement = document.createElement('my-element');

// can only happen if already defined

const myElement = new MyElement();

Creation triggers the **constructor**, if the element has already been defined. The constructor is called once per element instance.

Insert

An element is inserted into the DOM imperatively with JS:

document.body.append(myElement);

Any HTML element may be used instead of **document.body.** Insertion triggers the connectedCallback method, if the element has already been defined. Here is a trap:

<html>

<head>

<title>Page Title</title>

<!-- <script type="module" src="./helloworld.js"></script> -->

<script src="./helloworld.js"></script>

</head>

<body>

<h1>This is an example of the custom element</h1>

<p id="content">

</p>

<!-- <hello-world></hello-world> -->

<script>

const myElement = new HelloWorld();

parent = document.getElementById("content");

parent.append(myElement);

</script>

</body>

</html>

This HTML page will show nothing if you use

<script type="module" src="./helloworld.js"></script>

The browser defers module parsing, thus the **hello-world** element is undefined during the second script execution.

Declare

An element is declared when parsed as HTML. It may be done with HTML or JavaScript:

<my-element></my-element>

document.body.innerHTML = '<my-element></my-element>';

Declaration triggers the constructor and connectedCallback methods, if the element has already been defined.

Upgrade

If the browser encounters any <**time-formatted**> elements before **customElements.define**, that’s not an error. But the element is yet unknown, just like any non-standard tag. Such “undefined” elements can be styled with CSS selector **:not(:defined)**.

When customElement.define is called, they are “upgraded”: a new instance of **TimeFormatted** is created for each, and **connectedCallback** is called. They become :defined.

#### Properties and Attributes

When a browser loads a page, it parses or “reads” the HTML and generates a DOM Model of the page, which is an object-oriented representation of the page that contains several Objects, called Elements. All these objects have, like most objects in OOP, properties that can be modified with a scripting language such as JavaScript. Properties can be complex data like Objects, Functions or Arrays and can be considered as instance variables for a particular element.

Attributes are defined in the HTML document and provide additional information about HTML elements. When a browser parses the HTML to create DOM objects, it recognizes the attributes and creates DOM properties from them. They’re placed inside the element’s opening tag and always come in name/value pairs. They can be useful for configuring elements declaratively and accessibility APIs and CSS selectors rely heavily on them. An attribute is always a string!

* Some DOM Properties don’t have corresponding HTML attributes. [innerHTML](https://www.w3schools.com/jsref/prop_html_innerhtml.asp) is a good example.
* Some HTML attributes don’t have corresponding DOM properties. [colspan](https://www.w3schools.com/TAGS/att_colspan.asp) is a good example.
* Some attributes have a direct relation with DOM properties. [id](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/id) is a good example.

Attribute supplies default value for the corresponding property but values of these two entities may be different:

<a href="foo.html"/>

...

const anchor = document.querySelector('a');

anchor.getAttribute('href'); // returns "foo.html"

anchor.href; // returns "<https://test.com/foo.html>"

Custom attributes are attributes that are not part of the standard HTML5 attributes but are explicitly created. They allow us to add our information to HTML tags. These attributes store private information specific to a page or application, providing a way to add custom data to HTML elements. Any attribute whose name starts with **data-** is a custom attribute. The **data-\*** attributes allow us to embed custom attributes on all HTML elements. These are completely ignored by the user. The data stored can be used in JavaScript of the page. We can also use these data attributes to style our elements.

<element data-\*="value">

HTML DOM **getAttribute()** can be used to get the stored data from the attribute. It will either return a null or an empty string if the asked attribute does not exist. HTML DOM **setAttribute()** can be used to modify the value of any existing attribute or to add a new attribute.

One can access Data Attribute with **Dataset**. This feature provides a DOMStringMap object that holds custom data attributes. Each attribute in this map has a “**data-**” prefix removed, transforming the attribute name into camelCase. To access these attributes, you can use the camelCase name as a key, like **element.dataset.keyname** or **element.dataset[keyname]**. When an item is clicked, the JavaScript specified in the onclick attribute will be triggered.

<!DOCTYPE html>

<html>

<head>

<title>GeeksForGeeks</title>

<sript type="application/javascript">

function handleFood(food) {

var foodType = food.**dataset.foodType**;

alert(food.innerHTML + " is " + foodType +".");

}

</sript>

</head>

<body>

<h3>Welcome To GFG</h3>

<ul>

<li onclick="handleFood(this)" **data-food-type="fruit"**>

Apple

</li>

<li onclick="handleFood(this)" **data-food-type="vegetable"**>

Potato

</li>

<li onclick="handleFood(this)" **data-food-type="veg"**>

Cabbage

</li>

<li onclick="handleFood(this)" **data-food-type="non-veg"**>

Chicken

</li>

</ul>

</body>

</html>

CSS selector works with custom attributes but here you muste write full name (with **data-** prefix)

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<style>

p[data-about='blog'] {

background-color: #C2F784;

}

p[data-info='blogathon'] {

background-color: #DF2E2E;

}

</style>

</head>

<body>

<h2>Welcome To GFG</h2>

<p data-about="blog">Informational text entries</p>

<p data-info="blogathon">Blogathon 2021</p>

</body>

</html>

The programmer must take care of synchronizing HTML attributes with the corresponding properties of the DOM object. When using the **attributeChangedCallback** function, it is necessary to beware of an infinite loop and here it is important to remember that simple comparison of values ​​will not work:

attributeChangedCallback(attrName, oldVal, newVal) {

if ( oldVal != newVal) { // **Error** !

// null != null ==> **true**

// 253 != 0253 ==> **true** because attribute is always string

}

}

#### CustomElementRegistry

The [CustomElementRegistry](https://developer.mozilla.org/en-US/docs/Web/API/CustomElementRegistry) interface provides methods for registering custom elements and querying registered elements. To get an instance of it, use the **window.customElements** property. Here is a list of instance methods

define

Defines a new [custom element](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements).

define(name, constructor)

define(name, constructor, options)

get

Returns the **constructor** for the named custom element, or undefined if the custom element is not defined.

let ctor = customElements.get("my-paragraph");

upgrade

The customElements **upgrade**() method upgrades all shadow-containing custom elements of the document in a node subtree, even before they are connected to the main document.

customElements.upgrade(root);

This method returns void. Parameter **root** is a node instance with shadow-containing descendant elements that are to be upgraded.

const el = document.createElement("spider-man");

class SpiderMan extends HTMLElement {}

customElements.define("spider-man", SpiderMan);

console.assert(!(el instanceof SpiderMan)); // not yet upgraded

customElements.upgrade(el);

console.assert(el instanceof SpiderMan); // upgraded!

whenDefined

The **whenDefined()** method of the **CustomElementRegistry** interface returns a Promise that resolves when the named element is defined. This interesting example of this function is provided on the [MDN](https://udn.realityripple.com/docs/Web/API/CustomElementRegistry/whenDefined) page. An asynchronous function hides the **Loading...** text when all menu items become defined.

#### Parameters

The data can be sent to the custom element using the element attributes. It is possible to send scalar data or JSON

<!DOCTYPE html>

<html>

<head>

<title>Creating Custom Elements</title>

<script src="./indexOne.js" type="module"></script>

</head>

<body>

<salutation-element

name="Vladimir"

id=25

details='{"age": "20", "designation": "Developer"}'>

</salutation-element>

</body>

</html>

The custom control can read the parameters in the **connectedCallback** function:

class SalutationElement extends HTMLElement {

connectedCallback() {

const userName = t**his.attributes.name**.value;

const userDetails = JSON.parse(**this.attributes.details**.value);

this.innerHTML = `<p>

Hello ${userName}<br/>

your age is ${userDetails}</p>`

}

}

customElements.define('salutation-element', SalutationElement);

A better example of using parameters is provided in the [github](https://github.com/mappmechanic/super-button-web-component/blob/master/Steps/tutorial1-Basic%20Custom%20Element.md) site. An example is build using a bit outdated constructions, so I'm providing a corrected version here.

File **super-button.js**:

class SuperButton extends HTMLElement {

// Property **labelText**

**get labelText**() {

return this.getAttribute('label-text');

}

**set labelText**(value) {

if(value) {

this.setAttribute('label-text', value);

}

}

// Property **color**

**get color**() {

return this.getAttribute('color');

}

**set color**(value) {

this.setAttribute('color', value);

}

static get observedAttributes() {

return ['label-text', 'color'];

}

// Do that with template and shadow DOM

connectedCallback() {

this.innerHTML = `

<div class="button">

${this.labelText}

</div>

<style>

.button {

width: 150px;

cursor: pointer;

border-radius: 10px;

text-align: center;

padding: 5px;

border: 1px solid #999;

box-shadow: 0px 4px 2px -2px;

}

.grey { background: #e5e5e5 }

.red { background: #ea5e4c }

.yellow { background: #ffc60e }

.blue { background: #93cefc }

</style>

`;

this.updateColor(this.color);

}

attributeChangedCallback(name, oldValue, newValue) {

switch(name) {

case 'label-text':

if(this.querySelector('.button') && newValue) {

this.querySelector('.button').textContent = newValue;

}

break;

case 'color':

this.updateColor(newValue, oldValue);

break;

}

}

updateColor(newValue, oldValue) {

const defaultColor = 'grey';

const allowedColors = ['red','yellow','blue'];

const newColor = allowedColors.indexOf(newValue) > -1 ? newValue : defaultColor;

const btnContainer = this.querySelector('.button');

if(btnContainer && oldValue) {

btnContainer.classList.remove(oldValue);

}

if(btnContainer && newColor) {

btnContainer.classList.add(newColor);

}

}

}

window.customElements.define('super-button', SuperButton);

Lines 2-18: all updatable attributes are defined as **getters** and **setters** of the ES6 class. The component can retrieve the getter’s value using the **this** pointer, see to the 27th line. The setter value can be changed both on the page where the component is declared and inside the component.

The static function **observedAttributes()** returns an array with the names of the tracked attributes. to which the component must respond, see lines 20-22. The component calls **attributeChangedCallback** function after every change of tracked value .

The **connectedCallback()** function (lines 24-46) describes a content and styles of the component. Possible, but not the best option. A better description will be provided after examining the **templates** and **shadowDom**.

The **attributeChangedCallback(name, oldValue, newValue)** function (lines 48-59) is called automatically after any tracked attribute has been changed. This function may call any other method of the component, see lines 56, 61-72.

File **index.html:**

<!DOCTYPE>

<html>

<head>

<title>Testing - Web Components</title>

<script type="module" src="./super-button.js"></script>

</head>

<body>

Button1: <super-button label-text="Super Button1"></super-button> <br />

Button2: <super-button label-text="Super Button2"></super-button>

<br /><br />

Update Button1 Label <br />

<input type="text" id="newLabel"><br />

<button onclick="updateBtn1Label()">Update Label</button>

<br /><br />

New Color <br />

<select id="newColor">

<option value="">Default</option>

<option value="red">Red</option>

<option value="yellow">Yellow</option>

<option value="blue">Blue</option>

<option value="green">Green</option>

</select> <br />

<button onclick="updateBtn2Color()">Update Color</button>

<script>

function updateBtn1Label() {

const newLabel = document.getElementById('newLabel').value;

const superBtns = document.getElementsByTagName('super-button');

superBtns[0].labelText = newLabel;

};

function updateBtn2Color() {

const newColor = document.getElementById('newColor').value;

const superBtns = document.getElementsByTagName('super-button');

superBtns[1].color = newColor;

};

</script>

</body>

</html>

WEB component is loaded with the link tag in the original version:

<link rel="import" href="./super-button.html">

HTML imports feature has been deprecated thus I changed it with script tag, see line 5. You can also use plain JavaScript with an attribute **deferred**. Loading a module gives you a lot of extra options.

Lines 8, 9 demonstrate how to declare a component by giving an initial value for the parameter .

Lines 11-14 with a helper function **updateBtn1Label()** updates attribute **label-text**. Direct call of the setAttribute was used in the original:

superBtns[0].setAttribute('label-text', newLabel);

Both variants are equivalent.

Lines 11-14 with helper function **updateBtn2Color()** upates attribute color. Once again, setter was used instead of direct call of **setAttribute** function.

#### Template

The **<template>** HTML element is a mechanism for holding HTML that is not to be rendered immediately when a page is loaded but may be instantiated subsequently during runtime using JavaScript. While the parser does process the contents of the **<template>** element while loading the page, it does so only to ensure that those contents are valid; the element's contents are not rendered, however. Corresponding **HTMLTemplateElement** interface has a standard **content** property (without an equivalent content/markup attribute), which is a read-only DocumentFragment containing the DOM subtree which the template represents. Let's examine this HTML snippet:

<table id="producttable">

<thead>

<tr>

<td>UPC\_Code</td>

<td>Product\_Name</td>

</tr>

</thead>

<tbody>

</tbody>

</table>

<template id="productrow">

<tr>

<td class="record"></td>

<td></td>

</tr>

</template>

This page will display an empty table, but it will have two rows after executing this JavaScript:

const tbody = document.querySelector("tbody");

**const template = document.querySelector("#productrow");**

**const clone = template.content.cloneNode(true);**

let td = clone.querySelectorAll("td");

td[0].textContent = "1235646565";

td[1].textContent = "Stuff";

tbody.appendChild(clone);

const clone2 = template.content.cloneNode(true);

td = clone2.querySelectorAll("td");

td[0].textContent = "0384928528";

td[1].textContent = "Acme Kidney Beans 2";

tbody.appendChild(clone2);

Events cannot be attached to a **DocumentFragment**, so make a copy of the **template.content** and attach events to that copy. See [MDN](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template) for an example of this action.

By default, the element's content is not rendered. The corresponding HTMLTemplateElement interface includes a standard content property (without an equivalent content/markup attribute). This content property is read-only and holds a [DocumentFragment](https://developer.mozilla.org/en-US/docs/Web/API/DocumentFragment) that contains the DOM subtree represented by the template. This fragment can be cloned via the [cloneNode](https://developer.mozilla.org/en-US/docs/Web/API/Node/cloneNode) method and inserted into the DOM.

Templates can also be created with JavaScript:

const template = document.createElement(**'template'**);

**template.innerHTML** = `

<h1>Hello, World!</h1>

<p>And all who inhabit it</p>

`;

**document.body.appendChild(template.content);**

The latter variant is much better suited for building custom components, since it can be placed in the same file as code of the component. What's more, the template written this way can be placed in a separate file and loaded into the component using the **import** statement.

#### Shadow dom

An important aspect of web components is encapsulation — being able to keep the markup structure, style, and behavior hidden and separate from other code on the page so that different parts do not clash. The **Shadow DOM** API is a key part of this, providing a way to attach a hidden separated DOM to an element.

Shadow DOM allows hidden DOM trees to be attached to elements in the regular DOM tree — this shadow DOM tree starts with a **shadow root**, underneath which you can attach any element, in the same way as the normal DOM.



Image was copied from "[Using shadow dom](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM)" link. The following terminology is used in the figure:

* **Shadow host**: The regular DOM node that the shadow DOM is attached to.
* **Shadow tree**: The DOM tree inside the shadow DOM.
* **Shadow boundary**: the place where the shadow DOM ends, and the regular DOM begins.
* **Shadow root**: The root node of the shadow tree.

You can attach a shadow root to any element using the Element.attachShadow() method:

const shadowOpen = elementRef.attachShadow({ mode: "open" });

const shadowClosed = elementRef.attachShadow({ mode: "closed" });

If you attach a **shadow root** to a custom element with mode: **closed** set, you won't be able to access the shadow DOM from the outside — **myCustomElem.shadowRoot** returns null. Use this expression attching a shadow root inside the constructor of custom element:

const shadow = this.attachShadow({ mode: "open" }); // or {mode: "closed"}

The shadow dom may be also connected inside connectedCallback() function:

connectedCallback() {

const shadow = this.attachShadow({ mode: 'closed' });

shadow.innerHTML = `

<style>

p {

text-align: center;

font-weight: normal;

padding: 1em;

margin: 0 0 2em 0;

background-color: #eee;

border: 1px solid #666;

}

</style>

<p>Hello ${ this.name }!</p>`;

}

You can attach stylesheet with link element:

const linkElem = document.createElement("link");

linkElem.setAttribute("rel", "stylesheet");

linkElem.setAttribute("href", "style.css");

shadow.appendChild(linkElem);

An example of such custom element is presented on [GitHub](https://github.com/mdn/web-components-examples/blob/main/popup-info-box-external-stylesheet/main.js) site.

<!-- Base page (HTML) -->

<**template id="custom-paragraph"**>

<style>

p {

color: white;

background-color: #666;

padding: 5px;

}

</style>

<p>My paragraph</p>

<**/template**>

...

<!-- Custom component (JS) -->

**customElements.define**(

"my-paragraph",

class extends HTMLElement {

constructor() {

super();

let template = document.getElementById("custom-paragraph");

let templateContent = template.content;

const shadowRoot = **this.attachShadow({ mode: "open" })**;

shadowRoot.appendChild(templateContent.cloneNode(true));

}

},

);

Now we can use it by just adding it to our HTML document:

<my-paragraph></my-paragraph>

#### Slot

Slots are used inside templates of the custom elements:

<!-- Custom template -->

<template id="template-id">

<h2>Template with Slot</h2>

<p>

<slot **name="slot-id"**>Named Slot</slot>

</p>

<p>

<slot **name="sl-id"**>Named Slot without declaration</slot>

</p>

<p>

**<slot>Unnamed slot</slot>**

</p>

</template>

<!-- Host -->

<our-custom-element>

<span **slot="slot-id"**>This text will replace named slot</span>

**<span>This text will replace unnamed slot</span>**

</our-custom-element>

The text inside **slot** element is its default value the slot. The browser will display this text if the declaration of the custom element has no corresponding code.

Slots are identified by their name attribute, and allow you to define placeholders in your template that can be filled with any markup fragment you want when the element is used in the markup.

So, if we want to add a slot into our trivial example, we could update our template's paragraph element like this:

<template id="custom-paragraph">

<p><**slot name="my-text"**>My default text</slot></p>

</template>

To define the slot's content, we include an HTML structure inside the **<my-paragraph>** element with a slot attribute whose value is equal to the name of the slot we want it to fill. As before, this can be anything you like, for example:

<my-paragraph>

<span **slot="my-text"**>Let's have some different text!</span>

</my-paragraph>

or

<my-paragraph>

<ul **slot="my-text"**>

<li>Let's have some different text!</li>

<li>In a list!</li>

</ul>

</my-paragraph>

An **unnamed <slot>** will be filled with all of the custom element's top-level child nodes that do not have the slot attribute. This includes text nodes.

## Custom events

Custom events can be created in two ways:

1. Using the **Event** constructor

const myEvent = new **Event**('myevent', {

bubbles: true,

cancelable: true,

composed: false

})

1. Using the **CustomEvent** constructor

const myEvent = new **CustomEvent**("myevent", {

detail: {},

bubbles: true,

cancelable: true,

composed: false,

});

Parameters:

* **bubbles** - the property specifies whether the event should be propagated upward to the parent element. The default value of this property is **false** - the custom event does not propagate. Set this property to **true**, then the custom event will behave exactly like native DOM events. You can stop the propagation of the event via **event.stopPropagation()**.
* **cancelable** - the property specifies whether the event should be cancelable. Native DOM events are cancelable by default, so you can call **event.preventDefault()** on them, which will prevent the default action of the event. If the custom event has **cancelable** set to **false**, **calling event.preventDefault()** will not perform any action.
* **composed** - the property specifies whether an event should bubble across from the **shadow** **DOM** (created when using web components) to the real **DOM**. If **bubbles** is set to **false**, the value of this property won’t matter because you’re explicitly telling the event not to bubble upward. However, if you want to dispatch a custom event in a web component and listen on it on a parent element in the real DOM, then the **composed** property needs to be set to true.
* **detail** – any data that needs to be passed to the listener. Set a value to this property before performing the **dispatchEvent** operation.

Events can be dispatched to any object that extends **EventTarget**, and they include all **HTML** **elements**, the document, the window, etc.

const myEvent = new CustomEvent("myevent", {

detail: {},

bubbles: true,

cancelable: true,

composed: false,

});

document.querySelector("#someElement").**dispatchEvent(myEvent)**;

To listen for the custom event, add an event listener to the element you want to listen on, just as you would with native DOM events.

document.querySelector("#someElement").addEventListener("myevent", (event) => {

console.log("I'm listening on a custom event");

});

You have to set correct value of the this pointer when calling a method inside of the object. Use a **lambda expression** and the **bind()** function for doing that:

var myButton = {

content: 'OK',

click() {

console.log(this.content + ' clicked');

}

hookEvent(element) {

// Use bind() to ensure 'this' is the 'this' inside click()

element.addEventListener('click', this.click.bind(this));

}

};

Lambda expressions do not change the this pointer, so you may not need to use bind():

element.addEventListener('click', () => this.click());

You can also add extra parameters after the 1st (this) parameter and bind will pass in those values to the original function. Any additional parameters you later pass to the bound function will be passed in after the bound parameters:

var sum = function(a, b) {

return a + b;

};

var add5 = sum.bind(null, 5);

console.log(add5(10));

Value of **this** pointer may be changed by **call** and **apply** functions. Both methods are parts of the function prototype:

call(thisArg)

call(thisArg, arg1, /\* …, \*/ argN)

apply(thisArg)

apply(thisArg, argsArray)

### Event Bubbling and Event Capturing

There are two ways of event propagation in the HTML DOM, bubbling and capturing. Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?

<div>

<p>My paragraph</p> <!-- Click on this element -->

</div>

In bubbling the inner most element's event is handled first and then the outer: the **<p>** element's click event is handled first, then the **<div>** element's click event.

In capturing the outer most element's event is handled first and then the inner: the **<div>** element's click event will be handled first, then the **<p>** element's click event.

With the **addEventListener()** method you can specify the propagation type by using the "**useCapture**" parameter:

addEventListener(event, function, useCapture);

### Custom states and custom state pseudo-class CSS selectors

Built in HTML elements can have different states, such as "hover", "disabled", and "read only". Some of these states can be set as attributes using HTML or JavaScript, while others are internal, and cannot. Whether external or internal, commonly these states have corresponding CSS pseudo-classes that can be used to select and style the element when it is in a particular state.

**Autonomous cust**om elements (but not elements based on built-in elements) also allow you to define states and select against them using the **:state()** pseudo-class function. The code below shows how this works using the example of an autonomous custom element that has an internal state "collapsed".

The collapsed state is represented as a boolean property (with setter and getter methods) that is not visible outside of the element. To make this state selectable in CSS the custom element first calls [HTMLElement.attachInternals()](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/attachInternals) in its constructor in order to attach an [ElementInternals](https://developer.mozilla.org/en-US/docs/Web/API/ElementInternals) object, which in turn provides access to a [CustomStateSet](https://developer.mozilla.org/en-US/docs/Web/API/CustomStateSet) through the [ElementInternals.states](https://developer.mozilla.org/en-US/docs/Web/API/ElementInternals/states) property. The setter for the (internal) collapsed state adds the identifier hidden to the CustomStateSet when the state is true, and removes it when the state is false. The identifier is just a string: in this case we called it hidden, but we could have just as easily called it collapsed.

class MyCustomElement extends HTMLElement {

constructor() {

super();

**this.\_internals = this.attachInternals();**

}

**get collapsed()** {

return this.\_internals.states.has("hidden");

}

**set collapsed(flag)** {

if (flag) {

// Existence of identifier corresponds to "true"

this.\_internals.states.add("hidden");

} else {

// Absence of identifier corresponds to "false"

this.\_internals.states.delete("hidden");

}

}

}

customElements.define("my-custom-element", MyCustomElement);

We can use the identifier added to the custom element's **CustomStateSet** (this.\_internals.states) for matching the element's custom state. This is matched by passing the identifier to the **:state()** pseudo-class:

my-custom-element {

border: dashed red;

}

my-custom-element:state(hidden) {

border: none;

}

// [Creating and triggering events - Event reference | MDN](https://developer.mozilla.org/en-US/docs/Web/Events/Creating_and_triggering_events)

[Custom events in JavaScript: A complete guide - LogRocket Blog](https://blog.logrocket.com/custom-events-in-javascript-a-complete-guide/)

[What is Lit? – Lit](https://lit.dev/docs/)

## LIT

Lit is a simple library for building fast, lightweight web components. Every Lit component is a standard web component. Lit components can be used across multiple apps and sites, even if those apps and sites are built on a variety of front-end stacks. Each Lit component is a self-contained unit of UI, assembled from smaller building blocks: standard HTML elements and other web components. In turn, each Lit component is itself a building block that can be used–within an HTML document, another web component, or a framework component–to build larger and more complex interfaces. LIT components may be written in TypeScript or JavaScript.

## View Transitions

The **View Transition API** gives you the power to create seamless visual transitions between different views on your website. This creates a more visually engaging user experience for users as they navigate your site, regardless of whether it's built as a multi-page application (MPA) or a single-page application (SPA). Typical situations where you would use view transitions include:

* A **thumbnail image** on a product listing page that transitions into a full-size product image on the product detail page.
* A **fixed navigation bar** that stays in place as you navigate from one page to another.
* A **grid with items** moving positions as you filter through.

**View transitions do not work on Mozilla Firefox.** View transitions are not tied to a specific application architecture or framework and can be triggered not only on a single document and also between two different documents. Both types of view transition rely on the very same building blocks and principles:

1. The browser takes snapshots of the old and new states.
2. The DOM gets updated while rendering is suppressed.
3. The transitions are powered by CSS Animations.

The one thing that's different between both types is how you trigger them.

### Same-document view transitions

When a view [transition](https://developer.chrome.com/docs/web-platform/view-transitions/same-document) runs on a single document it is called a **same-document view transition**. This is typically the case in single-page applications (SPAs). Trigger a same-document view transitions by calling **document.startViewTransition**:

function handleClick(e) {

// Fallback for browsers that don't support this API:

if (!document.startViewTransition) {

updateTheDOMSomehow();

return;

}

// With a View Transition:

**document.startViewTransition(() => updateTheDOMSomehow());**

}

When invoked, the browser automatically captures snapshots of all elements that have a view-transition-name CSS property declared on them. It then executes the passed in callback that updates the DOM, after which it takes snapshots of the new state.

These snapshots are then arranged in a tree of pseudo-elements and animated using the power of CSS animations. Pairs of snapshots from the old and new state smoothly transition from their old position and size to their new location, while their content crossfades. If you want, you can use CSS to customize the animation.

The default view transition is a cross-fade:

function spaNavigate(data) {

// Fallback for browsers that don't support this API:

if (!document.startViewTransition) {

updateTheDOMSomehow(data);

return;

}

// With a transition:

document.startViewTransition(() => updateTheDOMSomehow(data));

}

Where updateTheDOMSomehow changes the DOM to the new state. That can be done however you want.

When **.startViewTransition()** is called, the API captures the current state of the page. This includes taking a snapshot. Once complete, the callback passed to .startViewTransition() is called. That's where the DOM is changed. Then, the API captures the new state of the page. Once the new state is captured, the API constructs a pseudo-element tree like this:

::view-transition

└─ ::view-transition-group(root)

└─ ::view-transition-image-pair(root)

├─ ::view-transition-old(root)

└─ ::view-transition-new(root)

The ::view-transition sits in an overlay, over everything else on the page. ::view-transition-old(root) is a screenshot of the old view, and ::view-transition-new(root) is a live representation of the new view. The old view animates from opacity: 1 to opacity: 0, while the new view animates from opacity: 0 to opacity: 1, creating a cross-fade.

All of the animation is performed using CSS animations, so they can be customized with CSS.

All of the view transition pseudo-elements can be targeted with CSS, and since the animations are defined using CSS, you can modify them using existing CSS animation properties. For example:

::view-transition-old(root),

::view-transition-new(root) {

animation-duration: 5s;

}

With that one change, the fade is now really slow. More detailes one can read on [Chrome](https://developer.chrome.com/docs/web-platform/view-transitions/same-document) site.

### Cross-document view transitions

When a view [transition](https://developer.chrome.com/docs/web-platform/view-transitions/cross-document) occurs between two different documents it is called a **cross-document view transition**. This is typically the case in multi-page applications (MPA). Cross-document view transitions rely on the very same building blocks and principles as same-document view transitions, which is very intentional:

1. The browser takes snapshots of elements that have a unique view-transition-name on both the old and new page.
2. The DOM gets updated while rendering is suppressed.
3. And finally, the transitions are powered by CSS animations.

What's different when compared with same-document view transitions, is that with cross-document view transitions you don't need to call **document.startViewTransition** to start a view transition. Instead, the trigger for a cross-document view transition is a same-origin navigation from one page to another, an action that is typically performed by the user of your website clicking a link.The browser takes snapshots of elements that have a unique view-transition-name on both the old and new page. There is no API to call in order to start a view transition between two documents. However, there are two conditions that need to be fulfilled:

1. Both documents need to exist on the same origin.
2. Both pages need to opt-in to allow the view transition.

Cross-document view transitions are limited to same-origin navigations only. A navigation is considered to be same-origin if the origin of both participating pages is the same.

The origin of a page is a combination of the used scheme, hostname, and port:

For example, you can have a cross-document view transition when navigating from developer.chrome.com to developer.chrome.com/blog, as those are same-origin. You can't have that transition when navigating from developer.chrome.com to www.chrome.com, as those are cross-origin and same-site. To have a cross-document view transition between two documents, both participating pages need to opt-in to allowing this. This is done with the [@view-transition](https://css-tricks.com/almanac/rules/v/view-transition/) at-rule in CSS. In the @view-transition at-rule, set the navigation descriptor to auto to enable view transitions for cross-document, same-origin navigations:

@view-transition {

navigation: auto;

}

The HTML specification includes two new events that you can use: **pageswap** and **pagereveal**:

* The pageswap event fires before the last frame of a page is rendered. You can use this to do some last-minute changes on the outgoing page, right before the old snapshots get taken.
* The pagereveal event fires on a page after it has been initialized or reactivated but before the first rendering opportunity. With it, you can customize the new page before the new snapshots get taken.

If you want, you can decide to skip the transition in both events.

// https://www.w3schools.com/css/

## CSS

Cascading Style Sheets (CSS) is used to format the layout of a web page. With CSS, you can control the color, font, the size of text, the spacing between elements, how elements are positioned and laid out, what background images or background colors are to be used, different displays for different devices and screen sizes, and much more! The [w3schools](https://www.w3schools.com/css/) site provides a complete description of this language.

CSS can be added to HTML documents in 3 ways:

* Inline – by using the **style** attribute inside HTML elements,
* Internal – by using a <**style**> element in the <**head**> section,
* External – by using a <**link**> element to link to an external CSS file.

The most common way to add CSS, is to keep the styles in external CSS files.

### Inline CSS

An inline CSS is used to apply a unique style to a single HTML element. An inline CSS

uses the style attribute of an HTML element.

The following example sets the text color of the <h1> element to blue, and the text color of

the <p> element to red:

<h1 style="color:blue;">A Blue Heading</h1>

<p style="color:red;">A red paragraph.</p>

### Internal CSS

An internal CSS is used to define a style for a single HTML page. An internal CSS is defined in the **<head>** section of an HTML page, within a **<style>** element.

The following example sets the text color of ALL the <h1> elements (on that page) to blue, and the text color of ALL the <p> elements to red. In addition, the page will be displayed with a "powderblue" background color:

<!DOCTYPE html>

<html>

<head>

<**style**>

body {background-color: powderblue;}

h1 {color: blue;}

p {color: red;}

</**style**>

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

### External CSS

An external style sheet is used to define the style for many HTML pages. To use an external style sheet, add a link to it in the <**head**> section of each HTML page:

<!DOCTYPE html>

<html>

<head>

<**link rel="stylesheet"** **href="https://www.w3schools.com/w3css/4/w3.css"**>

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

An external CSS file can be stored in your site. In this case, make it accessible to the service of static files [ 6 ].

### CSS Syntax

CSS block or external file consists of collection of rules. Each rule starts on new line and consists of selector and declarations inside curly braces:



Declarations are separated from each other by a semicolon. Spaces and newlines are ignored by the browser.

* **Selector** - this is the HTML element name at the start of the rule. It defines the element(s) to be styled (in this example, <h1> elements). To style a different element, change the selector.
* **Declaration** - this is a single rule like color: blue. It specifies which of the element's properties you want to style.
* **Properties** - these are features of an HTML element that you can change the values of, to make it styled differently. (In this example, color is a property of the <h1> elements.) In CSS, you choose which properties you want to affect in the rule.
* **Property value** - to the right of the property—after the colon—there is the property value. This chooses one out of many possible appearances for a given property. (For example, there are many color values in addition to red.)
* **Braces** - each rule must be wrapped in curly braces. (**{}**)
* **Colon** - within each declaration, you must use a colon (**:**) to separate the property from its value or values.
* **Semicolon** - within each rule, you must use a semicolon (**;**) to separate each declaration from the next one. Trailing semicolon is optional.

You can write multiple selectors, separated by commas. This is equivalent to multiple rules with identical declaration blocks:

a, .btn-link {

color: #0071c1;

}

This rule is is equivalent to

a {

color: #0071c1;

}

.btn-link {

color: #0071c1;

}

#### Selector

A **CSS selector** is a pattern of elements and other terms that tell the browser which HTML elements should be selected to have the CSS property values inside the rule applied to them. The element or elements which are selected by the selector are referred to as the subject of the selector.

Type selectors

A **type selector** is sometimes referred to as a tag name selector or element selector because it selects an HTML tag/element in your document. In the example below, we have used the **body**, **span**, **em** and **strong** selectors:

**body** {

font-family: sans-serif;

}

**span** {

background-color: yellow;

}

**strong** {

color: rebeccapurple;

}

**em** {

color: rebeccapurple;

}

Class selectors

The case-sensitive class selector starts with a dot (.) character. It will select everything in the document with that class applied to it.

body {

font-family: sans-serif;

}

**.highlight** {

background-color: yellow;

}

You can create a selector that will target specific elements with the class applied. We do this by using the type selector for the element we want to target, with the class appended using a dot, with no white space in between:

**span.highlight** {

background-color: yellow;

}

**h1.highlight** {

background-color: pink;

}

You can apply multiple classes to an element and target them individually, or only select the element when all of the classes in the selector are present. We can tell the browser that we only want to match the element if it has two classes applied by chaining them together with no white space between them.

<div **class="notebox"**>This is an informational note.</div>

<div **class="notebox warning"**>This note shows a warning.</div>

<div **class="notebox danger"**>This note shows danger!</div>

<div **class="danger"**>

This won't get styled — it also needs to have the notebox class

</div>

...

body {

font-family: sans-serif;

}

.notebox {

border: 4px solid #666;

padding: 0.5em;

margin: 0.5em;

}

**.notebox.warning** {

border-color: orange;

font-weight: bold;

}

**.notebox.danger** {

border-color: red;

font-weight: bold;

}

ID selectors

The case-sensitive ID selector begins with a # rather than a dot character, but is used in the same way as a class selector. The difference is that an ID can be used only once per page, and elements can only have a single id value applied to them. It can select an element that has the id set on it, and you can precede the ID with a type selector to only target the element if both the element and ID match:

<h1 **id="heading"**>ID selector</h1>

<p>

Veggies es bonus vobis, proinde vos postulo essum magis kohlrabi welsh onion

daikon amaranth tatsoi tomatillo melon azuki bean garlic.

</p>

<p **id="one"**>

Gumbo beet greens corn soko <strong>endive</strong> gumbo gourd. Parsley

shallot courgette tatsoi pea sprouts fava bean collard greens dandelion okra

wakame tomato. Dandelion cucumber earthnut pea peanut soko zucchini.

</p>

...

body {

font-family: sans-serif;

}

**#one** {

background-color: yellow;

}

**h1#heading** {

color: rebeccapurple;

}

The universal selector

The universal selector is indicated by an asterisk (\*). It selects everything in the document. If \* is chained using a [descendant combinator](https://developer.mozilla.org/en-US/docs/Web/CSS/Descendant_combinator), it selects everything inside that ancestor element. For example, **p \*** selects all the nested elements inside the **<p>** element.

One use of the universal selector is to make selectors easier to read and more obvious in terms of what they are doing. For example, if we wanted to select any descendant elements of an **<article>** element that are the first child of their parent, including direct children, and make them bold, we could use the **:first-child** [pseudo-class](https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Pseudo_classes_and_elements):

article **:first-child** {

font-weight: bold;

}

article **\*:first-child** {

font-weight: bold;

}

Although both do the same thing, the readability of a second one is significantly improved.

Attribute selectors

HTML elements can have attributes that give further detail about the element being marked up. In CSS you can use attribute selectors to target elements with certain attributes.

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Description** |
| [attr] | a[title] | Matches elements with an attr attribute (whose name is the value in square brackets). |
| [attr="value"] | a[href="https://example.com"] | Matches elements with an attr attribute whose value is exactly value — the string inside the quotes. |
| [attr~="value"] | p[class~="special"] | Matches elements with an attr attribute whose value is exactly value, or contains value in its (space-separated) list of values. |
| [attr|="value"] | div[lang|="en"] | Matches elements with an attr attribute whose value begins with value immediately followed by a hyphen. Matches element with **lang="en-GB"**. |
| attr^=value | div[lang^="en-"] | Matches elements with an attr attribute, whose value begins with value. Matches element with **lang="en-GB"**. |
| attr$=value | div[lang$="GB"] | Matches elements with an attr attribute whose **last word** ends with value. Matches element with **lang="Canada en-GB"**. |
| attr\*=value | li[class\*="ound"] | Matches elements with an attr attribute whose value contains value anywhere within the string. Matches element with **class="msg rounded box"]**. |
| [attr operator value i] | div[lang$="gb" i] | Adding an i (or I) before the closing bracket causes the value to be compared case-insensitively (for characters within the ASCII range). Matches element with **lang="Canada en-GB"**. |
| [attr operator value s] | div[lang$="GB"] | Adding an s (or S) before the closing bracket causes the value to be compared case-sensitively (for characters within the ASCII range). Matches element with **lang="Canada en-GB"**. |

Pseudo-classes

A CSS [pseudo-class](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes) is a keyword added to a selector that lets you style a specific state of the selected element(s). For example, the pseudo-class **:hover** can be used to select a button when a user's pointer hovers over the button and this selected button can then be styled.

button**:hover** {

color: blue;

}

A pseudo-class consists of a colon (:) followed by the pseudo-class name (e.g., :hover). A functional pseudo-class also contains a pair of parentheses to define the arguments (e.g., :dir()). The element that a pseudo-class is attached to is defined as an anchor element (e.g., button in case button:hover).

Pseudo-classes let you apply a style to an element not only in relation to the content of the document tree, but also in relation to external factors like the history of the navigator (:visited, for example), the status of its content (like :checked on certain form elements), or the position of the mouse (like :hover, which lets you know if the mouse is over an element or not):

* The [**:active**](https://developer.mozilla.org/en-US/docs/Web/CSS/:active) CSS pseudo-class represents an element (such as a button) that is being activated by the user. When using a mouse, "activation" typically starts when the user presses down the primary mouse button. The :active pseudo-class is commonly used on [<a>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a) and [<button>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button) elements. Other common targets of this pseudo-class include elements that are contained in an activated element, and form elements that are being activated through their associated [<label>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/label).
* The [**:any-link**](https://developer.mozilla.org/en-US/docs/Web/CSS/:any-link) CSS pseudo-class selector represents an element that acts as the source anchor of a hyperlink, independent of whether it has been visited. In other words, it matches every [<a>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a) or [<area>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/area) element that has an **href** attribute. Thus, it matches all elements that match :link or :visited.
* The [**:autofill**](https://developer.mozilla.org/en-US/docs/Web/CSS/:autofill) CSS pseudo-class matches when an <input> element has its value autofilled by the browser. The class stops matching if the user edits the field. Take into account that each browser has its own settings that allow a user to enable or disable auto-completion. The user agent style sheets of many browsers use !important in their :-webkit-autofill style declarations, making them non-overridable by webpages without resorting to JavaScript hacks. For example Chrome has the following in its internal stylesheet:

background-color: rgb(232 240 254) !important;

background-image: none !important;

color: -internal-light-dark(black, white) !important;

This means that you cannot set the background-color, background-image, or color in your own rules.

* The [**:checked**](https://developer.mozilla.org/en-US/docs/Web/CSS/:checked) CSS pseudo-class selector represents any radio (**<input type="radio">**), checkbox (**<input type="checkbox">**), or option (**<option>** in a **<select>**) element that is checked or toggled to an on state.
* The [**:default**](https://developer.mozilla.org/en-US/docs/Web/CSS/:default) CSS pseudo-class selects form elements that are the default in a group of related elements. This selector matches the **<button>**, **<input type="checkbox">**, **<input type="radio">**, and **<option>** elements:
  + A default option element is the first one with the selected attribute, or the first enabled option in DOM order. multiple <select>s can have more than one selected option, so all will match :default.
  + <input type="checkbox"> and <input type="radio"> match if they have the checked attribute.
  + <button> matches if it is a <form>'s default submission button: the first <button> in DOM order that belongs to the form. This also applies to <input> types that submit forms, like image or submit.
* The [**:defined**](https://developer.mozilla.org/en-US/docs/Web/CSS/:defined) CSS pseudo-class represents any element that has been defined. This includes any standard element built into the browser and custom elements that have been successfully defined (i.e. with the CustomElementRegistry.define() method).
* The [**:dir()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:dir) CSS pseudo-class matches elements based on the directionality of the text contained in them. The direction of writing must be indicated in brackets: **ltr** or **rtl**.

/\* Selects any element with right-to-left text \*/

:dir(rtl) {

background-color: red;

}

The :dir() pseudo-class uses only the semantic value of the directionality, i.e., the one defined in the document itself. It doesn't account for styling directionality, i.e., the directionality set by CSS properties such as direction.

* The [**:disabled**](https://developer.mozilla.org/en-US/docs/Web/CSS/:disabled) CSS pseudo-class represents any disabled element. An element is disabled if it can't be activated (selected, clicked on, typed into, etc.) or accept focus.
* The [**:enabled**](https://developer.mozilla.org/en-US/docs/Web/CSS/:enabled) CSS pseudo-class represents any enabled element. An element is enabled if it can be activated (selected, clicked on, typed into, etc.) or accept focus.
* The [**:empty**](https://developer.mozilla.org/en-US/docs/Web/CSS/:empty) CSS pseudo-class represents any element that has no children. Children can be either element nodes or text (including whitespace). Comments, processing instructions, and CSS content do not affect whether an element is considered empty.
* The [**:first**](https://developer.mozilla.org/en-US/docs/Web/CSS/:first) CSS pseudo-class, used with the @page at-rule, represents the first page of a printed document. (See :first-child for general first element of a node.). You can't change all CSS properties with this pseudo-class. You can only change the margins, orphans, widows, and page breaks of the document.
* The [**:first-child**](https://developer.mozilla.org/en-US/docs/Web/CSS/:first-child) CSS pseudo-class represents the first element among a group of sibling elements.
* The [**:first-of-type**](https://developer.mozilla.org/en-US/docs/Web/CSS/:first-of-type) CSS pseudo-class represents the first element of its type among a group of sibling elements.

<h2>Heading</h2>

<p>Paragraph 1</p>

<p>Paragraph 2</p>

...

**p:first-of-type** {

color: green;

font-style: italic;

}

* The [**:focus**](https://developer.mozilla.org/en-US/docs/Web/CSS/:focus) CSS pseudo-class represents an element (such as a form input) that has received focus. It is generally triggered when the user clicks or taps on an element or selects it with the keyboard's Tab key.
* The [**:focus-visible**](https://developer.mozilla.org/en-US/docs/Web/CSS/:focus-visible) pseudo-class applies while an element matches the :focus pseudo-class and the UA (User Agent) determines via heuristics that the focus should be made evident on the element. (Many browsers show a "focus ring" by default in this case).
* The [**:focus-within**](https://developer.mozilla.org/en-US/docs/Web/CSS/:focus-within) CSS pseudo-class matches an element if the element or any of its descendants are focused. In other words, it represents an element that is itself matched by the :focus pseudo-class or has a descendant that is matched by :focus. (This includes descendants in shadow trees).
* The [**:fullscreen**](https://developer.mozilla.org/en-US/docs/Web/CSS/:fullscreen) CSS pseudo-class matches every element that is currently in fullscreen mode. If multiple elements have been put into fullscreen mode, this selects them all. The **:fullscreen** pseudo-class lets you configure your stylesheets to automatically adjust the size, style, or layout of content when elements switch back and forth between fullscreen and traditional presentations.
* The [**:future**](https://developer.mozilla.org/en-US/docs/Web/CSS/:future) CSS pseudo-class selector is a time-dimensional pseudo-class that will match for any element which appears entirely after an element that matches [:current](https://developer.mozilla.org/en-US/docs/Web/CSS/:current).
* The functional [**:has()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:has) CSS pseudo-class represents an element if any of the relative selectors that are passed as an argument match at least one element when anchored against this element. This pseudo-class presents a way of selecting a parent element or a previous sibling element with respect to a reference element by taking a relative selector list as an argument:

/\* Selects an h1 heading with a paragraph element that immediately follows

the h1 and applies the style to h1 \*/

h1:**has(+ p)** {

margin-bottom: 0;

}

* The [**:host**](https://developer.mozilla.org/en-US/docs/Web/CSS/:host) CSS pseudo-class selects the shadow host of the shadow DOM containing the CSS it is used inside — in other words, this allows you to select a custom element from inside its shadow DOM.
* The [**:hover**](https://developer.mozilla.org/en-US/docs/Web/CSS/:hover) CSS pseudo-class matches when the user interacts with an element with a pointing device, but does not necessarily activate it. It is generally triggered when the user hovers over an element with the cursor (mouse pointer).
* The [**:in-range**](https://developer.mozilla.org/en-US/docs/Web/CSS/:in-range) CSS pseudo-class represents an <input> element whose current value is within the range limits specified by the min and max attributes.
* The [**:indeterminate**](https://developer.mozilla.org/en-US/docs/Web/CSS/:indeterminate) CSS pseudo-class represents any form element whose state is indeterminate, such as checkboxes that have been set to an indeterminate state with JavaScript, radio buttons which are members of a group in which all radio buttons are unchecked, and <progress> elements with no value attribute.
* The [**:invalid**](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class represents any <form>, <fieldset>, <input> or other <form> element whose contents fail to validate.
* The [**:is()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:is) CSS pseudo-class function takes a selector list as its argument, and selects any element that can be selected by one of the selectors in that list. Elements of the list are separated by comma. This is useful for writing large selectors in a more compact form.
* The [**:lang()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:lang) CSS pseudo-class matches elements based on the language they are determined to be in.
* The [**:last-child**](./The%20:last-child%20CSS%20pseudo-class%20represents%20the%20last%20element%20among%20a%20group%20of%20sibling%20elements.) CSS pseudo-class represents the last element among a group of sibling elements.
* The [**:last-of-type**](./The%20:last-of-type%20CSS%20pseudo-class%20represents%20the%20last%20element%20of%20its%20type%20among%20a%20group%20of%20sibling%20elements.) CSS pseudo-class represents the last element of its type among a group of sibling elements.
* The [**:link**](https://developer.mozilla.org/en-US/docs/Web/CSS/:link) CSS pseudo-class represents an element that has not yet been visited. It matches every unvisited <a> or <area> element that has an href attribute.
* The [**:modal**](https://developer.mozilla.org/en-US/docs/Web/CSS/:modal) CSS pseudo-class matches an element that is in a state in which it excludes all interaction with elements outside it until the interaction has been dismissed. Multiple elements can be selected by the :modal pseudo-class at the same time, but only one of them will be active and able to receive input.
* The [**:not()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:not) CSS pseudo-class represents elements that do not match a list of selectors. Elements of the list are separated by comma. Since it prevents specific items from being selected, it is known as the negation pseudo-class.
* The [**:nth-child()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-child) CSS pseudo-class matches elements based on the indexes of the elements in the child list of their parents. In other words, the :nth-child() selector selects child elements according to their position among all the sibling elements within a parent element. :nth-child() takes a single argument that describes a pattern for matching element indices in a list of siblings. Element indices are 1-based. The function accepts complex expressions: **even**, **odd**, **An + B**. Look for details on [Mozilla](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-child) site.
* The [**:nth-last-child()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-last-child) CSS pseudo-class matches elements based on their position among a group of siblings, counting from the end. The function accepts complex expressions: **even**, **odd**, **An + B**. Look for details on [Mozilla](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-last-child) site.
* The [**:nth-last-of-type()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-last-of-type) CSS pseudo-class matches elements based on their position among siblings of the same type (tag name), counting from the end. The function accepts complex expressions: **even**, **odd**, **An + B**. Look for details on [Mozilla](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-last-of-type) site.
* The [**:nth-of-type()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-of-type) CSS pseudo-class matches elements based on their position among siblings of the same type (tag name). The function accepts complex expressions: **even**, **odd**, **An + B**. Look for details on [Mozilla](https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-of-type) site.
* The [**:only-child**](https://developer.mozilla.org/en-US/docs/Web/CSS/:only-child) CSS pseudo-class represents an element without any siblings. This is the same as :first-child:last-child or :nth-child(1):nth-last-child(1), but with a lower specificity.
* The [**:only-of-type**](https://developer.mozilla.org/en-US/docs/Web/CSS/:only-of-type) CSS pseudo-class represents an element that has no siblings of the same type.
* The [**:optional**](https://developer.mozilla.org/en-US/docs/Web/CSS/:optional) CSS pseudo-class represents any [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input), [<select>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select), or [<textarea>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea) element that does not have the **required** attribute set on it.
* The [**:out-of-range**](https://developer.mozilla.org/en-US/docs/Web/CSS/:out-of-range) CSS pseudo-class represents an [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) element whose current value is outside the range limits specified by the [min](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input" \l "min) and [max](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input" \l "max) attributes.
* The [**:past**](https://developer.mozilla.org/en-US/docs/Web/CSS/:past) CSS pseudo-class selector is a time-dimensional pseudo-class that will match for any element which appears entirely before an element that matches :current. For example in a video with captions which are being displayed by WebVTT.
* The [**:paused**](https://developer.mozilla.org/en-US/docs/Web/CSS/:paused) CSS pseudo-class selector represents an element that is playable, such as [<audio>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio) or [<video>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video), when that element is "paused" (i.e. not "playing").
* The [**:picture-in-picture**](https://developer.mozilla.org/en-US/docs/Web/CSS/:picture-in-picture) CSS pseudo-class matches the element which is currently in picture-in-picture mode.
* The [**:placeholder-shown**](https://developer.mozilla.org/en-US/docs/Web/CSS/:placeholder-shown) CSS pseudo-class represents any **<input>** or **<textarea>** element that is currently displaying [placeholder](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input" \l "placeholder) text.
* The [**:popover-open**](https://developer.mozilla.org/en-US/docs/Web/CSS/:popover-open) CSS pseudo-class represents a [popover](https://developer.mozilla.org/en-US/docs/Web/API/Popover_API) element (i.e. one with a popover attribute) that is in the showing state. You can use this to apply style to popover elements only when they are shown.
* The [**:read-only**](https://developer.mozilla.org/en-US/docs/Web/CSS/:read-only) CSS pseudo-class selects elements (such as certain **<input>** types and **<textarea>**) that are not editable by the user. Elements on which the HTML attribute readonly doesn't have an effect (such as **<input type="radio">**, **<input type="checkbox">**, and all other non-form elements) are also selected by the :read-only pseudo-class. In fact, :read-only matches anything that :read-write doesn't match, making it equivalent to :not(:read-write).
* The [**:read-write**](https://developer.mozilla.org/en-US/docs/Web/CSS/:read-write) CSS pseudo-class represents an element (such as input or textarea) that is editable by the user.
* The [**:required**](https://developer.mozilla.org/en-US/docs/Web/CSS/:required) CSS pseudo-class represents any **<input>**, **<select>**, or **<textarea>** element that has the **required** attribute set on it.
* The [**:right**](https://developer.mozilla.org/en-US/docs/Web/CSS/:right) CSS pseudo-class, used with the @page at-rule, represents all right-hand pages of a printed document.
* The [**:root**](https://developer.mozilla.org/en-US/docs/Web/CSS/:root) CSS pseudo-class matches the root element of a tree representing the document. In HTML, **:root** represents the **<html>** element and is identical to the selector html, except that its specificity is higher.
* The [**:scope**](https://developer.mozilla.org/en-US/docs/Web/CSS/:scope) CSS pseudo-class represents elements that are a reference point, or scope, for selectors to match against. Which element(s) :scope matches depends on the context in which it is used:
  + When used at the root level of a stylesheet, **:scope** is equivalent to **:root**, which in a regular HTML document matches the <html> element.
  + When used inside a **@scope** block, :scope matches the block's defined scope root. It provides a way to apply styles to the root of the scope from inside the @scope block itself.
  + When used within a DOM API call — such as **querySelector()**, **querySelectorAll()**, **matches()**, or **Element.closest()** — **:scope** matches the element on which the method was called.
* The [**:state()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:state) CSS pseudo-class matches [custom elements](https://developer.mozilla.org/en-US/docs/Web/API/CustomStateSet) that have the specified custom state.
* The [**:target**](https://developer.mozilla.org/en-US/docs/Web/CSS/:target) CSS pseudo-class selects the target element of the document. When the document is loaded, the target element is derived using the document's URL fragment identifier. Without the URL fragment identifier, the document has no target element. The **:target** pseudo-class allows styling the document's target element. The element could be focused, highlighted, animated, etc. The target element is set at **document load** and **history.back()**, **history.forward()**, and **history.go()** method calls. But it is not changed when **history.pushState()** and **history.replaceState()** methods are called. Consider the example in the [MDN](https://developer.mozilla.org/en-US/play) site.
* The [**:user-invalid**](https://developer.mozilla.org/en-US/docs/Web/CSS/:user-invalid) CSS pseudo-class represents any validated form element whose value isn't valid based on their [validation constraints](https://developer.mozilla.org/en-US/docs/Web/HTML/Constraint_validation), after the user has interacted with it. The **:user-invalid** pseudo-class must match an [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid), [:out-of-range](https://developer.mozilla.org/en-US/docs/Web/CSS/:out-of-range), or blank-but [:required](https://developer.mozilla.org/en-US/docs/Web/CSS/:required) element between the time the user has attempted to submit the form and before the user has interacted again with the form element.
* The [**:user-valid**](https://developer.mozilla.org/en-US/docs/Web/CSS/:user-valid) CSS pseudo-class represents any validated form element whose value validates correctly based on its [validation constraints](https://developer.mozilla.org/en-US/docs/Web/HTML/Constraint_validation). However, unlike [:valid](https://developer.mozilla.org/en-US/docs/Web/CSS/:valid) it only matches once the user has interacted with it. This pseudo-class is applied if the form control is valid and any of the following has occurred:
  + The user made a change to the form control and committed the change such as by moving focus elsewhere.
  + The user has attempted to submit the form, even if no change was made to the control.
  + The value was invalid when it gained focus, and the user made a change making it valid, even if focus is still in the control.

Once this pseudo-class has been applied, the user-agent re-validates whether the control is valid at every keystroke when the control has focus.

* The [**:valid**](https://developer.mozilla.org/en-US/docs/Web/CSS/:valid) CSS pseudo-class represents any [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) or other [<form>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form) element whose contents validate successfully. This allows to easily make valid fields adopt an appearance that helps the user confirm that their data is formatted properly.
* The [**:visited**](https://developer.mozilla.org/en-US/docs/Web/CSS/:visited) CSS pseudo-class applies once the link has been visited by the user. For privacy reasons, the styles that can be modified using this selector are very limited. The :visited pseudo-class applies only to [<a>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a) and [<area>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/area) elements that have an **href** attribute.
* The [**:where()**](https://developer.mozilla.org/en-US/docs/Web/CSS/:where) CSS pseudo-class function takes a selector list as its argument, and selects any element that can be selected by one of the selectors in that list. The difference between [:where()](https://developer.mozilla.org/en-US/docs/Web/CSS/:where) and [:is()](https://developer.mozilla.org/en-US/docs/Web/CSS/:is) is that :where() always has 0 specificity, whereas :is() takes on the specificity of the most specific selector in its arguments. Consider an example on [MDN](https://developer.mozilla.org/en-US/play) site.

CSS Pseudo-elements

A CSS **pseudo-element** is used to style specific parts of an element. It can be used to:

* Style the first letter or line, of an element,
* Insert content before or after an element,
* Style the viewbox behind a dialog box.

The syntax of pseudo-elements:

selector**::**pseudo-element {

property: value;

}

* The [**::after**](https://developer.mozilla.org/en-US/docs/Web/CSS/::after) pseudo-element creates an element that is the last child of the selected one. It is often used to add cosmetic content to an element with the content property. It is inline by default. Syntax:

::after {

content: /\* value \*/;

/\* properties \*/

}

If the content property is not specified, has an invalid value, or has normal or none as a value, then the **::after** pseudo-element is not rendered. It behaves as if **display: none** is set.

* The [**::backdrop**](https://developer.mozilla.org/en-US/docs/Web/CSS/::backdrop) CSS pseudo-element is a box the size of the [viewport](https://developer.mozilla.org/en-US/docs/Glossary/Viewport), which is rendered immediately beneath any element being presented in the top [layer](https://developer.mozilla.org/en-US/docs/Glossary/Top_layer).
* The [**::before**](https://developer.mozilla.org/en-US/docs/Web/CSS/::before) creates a pseudo-element that is the first child of the selected element. It is often used to add cosmetic content to an element with the [content](https://developer.mozilla.org/en-US/docs/Web/CSS/content) property. It is inline by default.
* The [**::cue**](https://developer.mozilla.org/en-US/docs/Web/CSS/::cue) CSS pseudo-element matches [WebVTT](https://developer.mozilla.org/en-US/docs/Web/API/WebVTT_API) cues within a selected element. This can be used to style [captions and other cues](https://developer.mozilla.org/en-US/docs/Web/API/WebVTT_API) in media with VTT tracks.
* The [**::file-selector-button**](https://developer.mozilla.org/en-US/docs/Web/CSS/::file-selector-button) CSS pseudo-element represents the button of an [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) of [type="file"](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/file).
* The [**::first-letter**](https://developer.mozilla.org/en-US/docs/Web/CSS/::first-letter) CSS pseudo-element applies styles to the first letter of the first line of a block [container](https://developer.mozilla.org/en-US/docs/Web/CSS/Visual_formatting_model" \l "block_containers), but only when not preceded by other content (such as images or inline tables).
* The [**::first-line**](https://developer.mozilla.org/en-US/docs/Web/CSS/::first-line) CSS pseudo-element applies styles to the first line of a [block container](https://developer.mozilla.org/en-US/docs/Web/CSS/Visual_formatting_model" \l "block_containers).
* The [**::grammar-error**](https://developer.mozilla.org/en-US/docs/Web/CSS/::grammar-error) CSS pseudo-element represents a text segment which the user agent has flagged as grammatically incorrect. **Does not work on Mozilla Firefox.**
* The [**::highlight()**](https://developer.mozilla.org/en-US/docs/Web/CSS/::highlight) CSS pseudo-element applies styles to a custom highlight. A custom highlight is a collection of [Range](https://developer.mozilla.org/en-US/docs/Web/API/Range) objects and is registered on a webpage using the [HighlightRegistry](https://developer.mozilla.org/en-US/docs/Web/API/HighlightRegistry). **Does not work on Mozilla Firefox.**
* The [**::marker**](https://developer.mozilla.org/en-US/docs/Web/CSS/::marker) CSS pseudo-element selects the marker box of a list item, which typically contains a bullet or number. It works on any element or pseudo-element set to [display](https://developer.mozilla.org/en-US/docs/Web/CSS/display): list-item, such as the [<li>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/li) and [<summary>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/summary) elements.
* The [**::part**](https://developer.mozilla.org/en-US/docs/Web/CSS/::part) CSS pseudo-element represents any element within a shadow tree that has a matching part attribute. The global part attribute makes a shadow tree element visible to its parent DOM. The part names declared using the part attribute are used as the parameter of the ::part() pseudo-element. In this way, you can apply CSS styles to elements in the shadow tree from outside of it. Part names are similar to CSS classes: multiple elements can have the same part name, and a single element can have multiple part names. All part names used in ::part() pseudo-element must be present in the part value declared on the shadow tree element but the order of the part names doesn't matter, i.e., the selectors ::part(tab active) and ::part(active tab) are the same. The ::part() pseudo-element is only visible to the parent DOM. This means that when a shadow tree is nested, the parts are not visible to any ancestors other than the direct parent. The [exportparts](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/exportparts) attribute solves this limitation by explicitly exporting already defined part names, making them globally stylable.
* The [**::placeholder**](https://developer.mozilla.org/en-US/docs/Web/CSS/::placeholder) CSS pseudo-element represents the placeholder text in an [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) or [<textarea>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea) element.
* The [**::selection**](https://developer.mozilla.org/en-US/docs/Web/CSS/::selection) CSS pseudo-element applies styles to the part of a document that has been highlighted by the user (such as clicking and dragging the mouse across text). **Does not work on Safari.**
* The [**::slotted()**](https://developer.mozilla.org/en-US/docs/Web/CSS/::slotted) CSS pseudo-element represents any element that has been placed into a slot inside an HTML template (see [Using templates and slots](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_templates_and_slots) for more information). This only works when used inside CSS placed within a shadow DOM. Note that this selector won't select a text node placed into a slot; it only targets actual elements.
* The [**::spelling-error**](https://developer.mozilla.org/en-US/docs/Web/CSS/::spelling-error) CSS pseudo-element represents a text segment which the [user agent](https://developer.mozilla.org/en-US/docs/Glossary/User_agent) has flagged as incorrectly spelled. **Does not work on Mozilla Firefox.**
* The [**::target-text**](https://developer.mozilla.org/en-US/docs/Web/CSS/::target-text) CSS pseudo-element represents the text that has been scrolled to if the browser supports [text fragments](https://developer.mozilla.org/en-US/docs/Web/URI/Fragment/Text_fragments). It allows authors to choose how to highlight that section of text.
* The [**::view-transition**](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition) CSS pseudo-element represents the root of the [view transitions](https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API) overlay, which contains all view transition snapshot groups and sits over the top of all other page content. During a view transition, ::view-transition is included in the associated pseudo-element tree as explained in [The view transition pseudo-element tree](https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API/Using" \l "the_view_transition_pseudo-element_tree). It is the top-level node of this tree, and has one or more [::view-transition-groups](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-group) as children. **Does not work on Mozilla Firefox.**
* The [**::view-transition-group**](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-group) CSS pseudo-element represents a single view transition snapshot group. During a view transition, ::view-transition-group is included in the associated pseudo-element tree as explained in [The view transition pseudo-element](https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API/Using" \l "the_view_transition_pseudo-element_tree) tree. It is only ever a child of [::view-transition](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition), and has a [::view-transition-image-pair](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-image-pair) as a child. **::view-transition-group** is given the following default styling in the UA stylesheet:

html::view-transition-group(\*) {

position: absolute;

top: 0;

left: 0;

animation-duration: 0.25s;

animation-fill-mode: both;

}

By default, selected elements initially mirror the size and position of the ::view-transition-old pseudo-element representing the "old" view state, or the ::view-transition-new pseudo-element representing the "new" view state if there isn't an "old" view state. **Does not work on Mozilla Firefox.**

* The [**::view-transition-image-pair**](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-image-pair) CSS pseudo-element represents a container for a [view transition's](https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API) "old" and "new" view states — before and after the transition. During a view transition, ::view-transition-image-pair is included in the associated pseudo-element tree as explained in [The view transition pseudo-element tree](https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API/Using" \l "the_view_transition_pseudo-element_tree). It is only ever a child of a [::view-transition-group](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-group). In terms of children, it can have a [::view-transition-new](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-new) or a [::view-transition-old](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-old), or both.

::view-transition-image-pair is given the following default styling in the UA stylesheet:

:root::view-transition-image-pair(\*) {

position: absolute;

inset: 0;

animation-duration: inherit;

animation-fill-mode: inherit;

animation-delay: inherit;

}

During a view transition, ::view-transition-image-pair has isolation: isolate set on it in the view transition style sheet so that its children can be blended with non-normal blend modes without affecting other visual outputs. **Does not work on Mozilla Firefox.**

* The [**::view-transition-new**](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-new) CSS pseudo-element represents the "new" view state of a view transition — a snapshot live representation of the state after the transition. During a view transition, ::view-transition-new is included in the associated pseudo-element tree as explained in [The view transition pseudo-element](https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API/Using" \l "the_view_transition_pseudo-element_tree) tree. It is only ever a child of a [::view-transition-image-pair](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-image-pair), and never has any children. It is a replaced element and therefore can be manipulated with properties such as [object-fit](https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit) and [object-position](https://developer.mozilla.org/en-US/docs/Web/CSS/object-position). It has natural dimensions equal to the content's size. **Does not work on Mozilla Firefox.**
* The [**::view-transition-old**](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-old) CSS pseudo-element represents the "old" view state of a view transition — a static snapshot of the old view, before the transition. During a view transition, ::view-transition-old is included in the associated pseudo-element tree as explained in [The view transition pseudo-element](https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API/Using" \l "the_view_transition_pseudo-element_tree) tree, provided there's an "old" view state to represent. It is only ever a child of a [::view-transition-image-pair](https://developer.mozilla.org/en-US/docs/Web/CSS/::view-transition-image-pair), and never has any children. It is a replaced element and therefore can be manipulated with properties such as [object-fit](https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit) and [object-position](https://developer.mozilla.org/en-US/docs/Web/CSS/object-position). It has natural dimensions equal to the content's size. **Does not work on Mozilla Firefox.**

At-rules

CSS2

[:active - CSS: Cascading Style Sheets | MDN](https://developer.mozilla.org/en-US/docs/Web/CSS/:active)

<https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Attribute_selectors>

<https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors>

<https://www.w3schools.com/css/css_selectors.asp>

<https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics>

### Templates

It is worth using the templates provided by [W3C](https://www.w3schools.com/css/css_rwd_templates.asp) when creating a new page. I have used Template for the [Responsive WEB Page](https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_templates_webpage&stacked=h).

[Introduction · Bootstrap v5.0 (getbootstrap.com)](https://getbootstrap.com/docs/5.0/getting-started/introduction/)

[Bootstrap 5 Tutorial (w3schools.com)](https://www.w3schools.com/bootstrap5/index.php)

## Bootstrap 5

Bootstrap is the world’s most popular framework for building responsive, mobile-first sites. Use jsDelivr for installation CSS and JS into the page.

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap 5 Example</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body>

...

<script

src="[https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js](https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js)">

</script>

</body>

</html>

File **bootstrap.bundle.min.js** includes **Popper** for tooltips and popovers.

The main differences between Bootstrap 5 and Bootstrap 3 & 4, is that Bootstrap 5 has switched to vanilla JavaScript instead of jQuery. Bootstrap 5 also requires a containing element to wrap site contents. Bootstrap 5 is designed to be responsive to mobile devices. Mobile-first styles are part of the core framework.

## W3.CSS

W3.CSS is a modern, responsive, mobile first CSS framework. It works with:

* all browsers: Chrome. Firefox. Edge. IE. Safari. Opera.
* -all devices: Desktop. Laptop. Tablet. Mobile.

W3.CSS is standard CSS only (No jQuery or JavaScript library). To use W3.CSS in your web site, just add a link to "w3.css" from your web pages:

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap 5 Example</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

**<link href="https://www.w3schools.com/w3css/4/w3.css" rel="stylesheet">**

</head>

<body>

...

</body>

</html>

or download w3.css from [w3css\_downloads](https://www.w3schools.com/w3css/w3css_downloads.asp) and add a link to w3.css in the head of your HTML. You have to use the request multiplexer and process CSS file as static resource. The process is explained in the "**Serving Static Files and Sites**" topic of this document. The link will be

<link rel="stylesheet" href="public/css/w3 .css">

if a path to w3.css is /public/css/ w3.css.

### W3.CSS Colors

The default color scheme used in W3.CSS is inspired by Material Design Colors (colors used in marketing, road signs, and sticky notes). It defines [28 colors](https://www.w3schools.com/w3css/w3css_colors.asp):

w3-red w3-khaki w3-pink w3-yellow

w3-purple w3-amber w3-deep-purple w3-orange

w3-indigo w3-deep-orang w3-blue w3-blue-grey

w3-light-blue w3-brown w3-cyan w3-light-grey

w3-aqua w3-grey w3-teal w3-dark-grey

w3-green w3-pale-red w3-light-green w3-pale-yellow

w3-lime w3-pale-green w3-sand w3-pale-blue

The **w3-color** classes set the background color for any HTML element:

<table class="w3-table w3-bordered **w3-light-gray**">

<tbody>

<tr class="**w3-black**">

<th>Name</th>

<th>Points</th>

</tr>

<tr>

<td>Jill Smith</td>

<td>50</td>

</tr>

<tr>

<td>Eve Jackson</td>

<td>94</td>

</tr>

<tr>

<td>Adam Johnson</td>

<td>67</td>

</tr>

</tbody>

</table>

Prepend color name with **w3-text-...** for coloring text:

<div>

<span class="w3-xlarge **w3-text-red** w3-margin-right">Red</span>

<span class="w3-xlarge **w3-text-purple** w3-margin-right">Purple</span>

<span class="w3-xlarge **w3-text-blue** w3-margin-right">Blue</span>

<span class="w3-xlarge **w3-text-green** w3-margin-right">Green</span>

<span class="w3-xlarge **w3-text-orange**">Orange</span>

</div>

Prepend color name with **w3-border-** for coloring borders:

<div style="width:50%">

<div class="w3-panel w3-border **w3-border-red**"><p>Red</p></div>

<div class="w3-panel w3-border **w3-border-green**"><p>Green</p></div>

<div class="w3-panel w3-border **w3-border-blue**"><p>Blue</p></div>

<div class="w3-panel w3-border **w3-border-yellow**"><p>Yellow</p></div>

<div class="w3-panel w3-border **w3-border-gray**"><p>Gray</p></div>

<div class="w3-panel w3-border **w3-border-black**"><p>Black</p></div>

</div>

The **w3-hover-**color classes define the background hover color for any HTML element:

<div class="w3-container w3-orange **w3-hover-red**">

<h2>London</h2>

<p>

London is the most populous city in the United Kingdom, with a metropolitan

area of over 9 million inhabitants.

</p>

</div>

In addition to standard W3.CSS colors, W3.CSS can also use colors from a lot of different color libraries:

* Windows Colors
* Fashion Colors
* Highway Colors

Look at [w3c](https://www.w3schools.com/w3css/w3css_colors.asp) site for details.

### W3.CSS Containers

The [w3-container](https://www.w3schools.com/w3css/w3css_containers.asp) class adds a **16px left and right padding**  to any HTML element. There no top or bottom padding inside w3-container. The w3-container class is the perfect class to use for all HTML container elements like: <div>, <article>, <section>, <header>, <footer>, <form>, and more. The **w3-container** provides equality for all HTML container elements:

* Common margins
* Common paddings
* Common alignments
* Common fonts
* Common colors

Normally you will not have to change the default padding of a container, because paragraphs and heading provide margins that will simulate padding.

<div class="**w3-container** w3-black">

<h2>This is my Header</h2>

</div>

<div class="**w3-containe**r w3-light-grey w3-text-brown">

<h2>This is my Article</h2>

<p>

This article is light grey and the text is brown.

</p>

</div>

<div class="**w3-container** w3-dark-grey">

<p>This is my Footer</p>

</div>

### W3.CSS Panels

The [w3-panel](https://www.w3schools.com/w3css/w3css_panels.asp) class adds a **16px top and bottom margin** and a **16px left and right padding** to any HTML element. It is container with top and bottom margins.

<div class="w3-panel w3-red">

<p>I am a panel.</p>

</div>

The **w3-panel** class is perfect for displaying [notes](https://www.w3schools.com/w3css/w3css_notes.asp). Notes are often displayed with a pale color:

<div class="w3-panel w3-pale-green">

<p>London is the most populous city in the United Kingdom,

with a metropolitan area of over 9 million inhabitants.</p>

</div>

The **w3-panel** class is perfect for displaying [quotes](https://www.w3schools.com/w3css/w3css_quotes.asp). Quotes are often displayed with a grey background, a left border bar, and an italic font style:

<div class="w3-panel w3-leftbar w3-light-grey w3-xxlarge w3-serif">

<p><i>"Make it as simple as possible, but not simpler."</i></p>

</div>

If you use the standard HTML <blockquote> element, notice that the browser will add an extra left and right margin:

<blockquote class="w3-panel w3-leftbar w3-light-grey">

<p class="w3-large">

<i>"Make it as simple as possible, but not simpler."</i></p>

<p>Albert Einstein</p>

</blockquote>

The **w3-panel** class is perfect for displaying [alerts](https://www.w3schools.com/w3css/w3css_alerts.asp). Alerts are often displayed using a strong color:

<div class="w3-panel w3-red">

<h3>Danger!</h3>

<p>Red often indicates a dangerous or negative situation.</p>

</div>

To create the X that closes the alert, add a <**span**> element with class **w3-button** and an **onclick** event:

<div class="w3-panel w3-red w3-display-container">

<span onclick="this.parentElement.style.display='none'"

class="w3-button w3-large **w3-display-topright**">&times;</span>

<h3>Danger!</h3>

<p>Red often indicates a dangerous or negative situation.</p>

</div

Showing (a hidden) panel is easy:

<**button** class="w3-btn"

onclick="document.getElementById('id01').style.display='block'">

Show panel

</**button**>

<div id="id01" class="w3-panel w3-green w3-display-container"

style="display:none">

<span onclick="this.parentElement.style.display='none'"

class="w3-button w3-display-topright">X</span>

<p>Click on the X to close this panel.</p>

</div>

### W3.CSS Border

W3.CSS provides the following [border](https://www.w3schools.com/w3css/w3css_borders.asp) classes:

|  |  |  |
| --- | --- | --- |
| **#** | **Class** | **Defines** |
| 1 | w3-border | Adds borders (top, right, bottom, left) to an element |
| 2 | w3-border-top | Adds a top border to an element |
| 3 | w3-border-right | Adds a right border to an element |
| 4 | w3-border-bottom | Adds a bottom border to an element |
| 5 | w3-border-left | Adds a left border to an element |
| 6 | w3-border-0 | Removes all borders |
| 7 | w3-border-color | Displays the border in a specified color (like red, blue, etc) |
| 8 | w3-hover-border-color | Adds a hoverable border color |
| 9 | w3-bottombar | Adds a thick bottom border to an element |
| 10 | w3-leftbar | Adds a thick left border to an element |
| 11 | w3-rightbar | Adds a thick right border to an element |
| 12 | w3-topbar | Adds a thick top border to an element |
| Write color names without prefix w3-: red, blue, sand,... | | |

<div class="w3-panel **w3-border**">

<p>I have borders.</p>

</div>

<div class="w3-panel **w3-border-left**">

<p>I have only a left border.</p>

</div>

<div class="w3-panel **w3-border-top w3-border-bottom**">

<p>I have top and bottom borders.</p>

</div>

The w3-border-color classes are used to add colors to borders:

<div class="w3-panel **w3-border** **w3-border-red**">

<p>I have red borders.</p>

</div>

<div class="w3-panel **w3-border-left** **w3-border-blue**">

<p>I have a blue left border.</p>

</div>

<div class="w3-panel **w3-border-top w3-border-bottom** **w3-border-green**">

<p>I have a green top and bottom border.</p>

</div>

Write color names without **w3-** prefix.

### W3.CSS Round, Circle

The [w3-round-size](https://www.w3schools.com/w3css/w3css_round.asp) classes add rounded corners to any HTML element:

|  |  |  |
| --- | --- | --- |
| # | Class | Defines |
| 1 | w3-round | Element rounded (border-radius) 4px |
| 2 | w3-round-small | Element rounded (border-radius) 2px |
| 3 | w3-round-medium | Element rounded (border-radius) 4px |
| 4 | w3-round-large | Element rounded (border-radius) 8px |
| 5 | w3-round-xlarge | Element rounded (border-radius) 16px |
| 6 | w3-round-xxlarge | Element rounded (border-radius) 32px |

<div class="w3-container">

<h2>Round Classes</h2>

<div class="w3-panel **w3-round-small** w3-teal">

<p>w3-round-small</p>

</div>

<div class="w3-panel **w3-round** w3-teal">

<p>w3-round</p>

</div>

<div class="w3-panel **w3-round-large** w3-teal">

<p>w3-round-large</p>

</div>

<div class="w3-panel **w3-round-xlarge** w3-teal">

<p>w3-round-xlarge</p>

</div>

<div class="w3-panel **w3-round-xxlarge** w3-teal">

<p>w3-round-xxlarge</p>

</div>

</div>

The **w3-circle** class displays content inside a [circle](https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_display_circle):

<img class="w3-circle" src="img\_car.jpg" alt="Car">

A Circle Inside a Circle:

<div class="w3-padding-32 w3-green **w3-circle**">

<img src="img\_car.jpg" class="**w3-circle**" style="width:75%">

</div>

### W3.CSS Padding

The [w3-padding-number](https://www.w3schools.com/w3css/w3css_padding.asp) classes add **top** and **bottom** padding to any HTML element:

|  |  |  |
| --- | --- | --- |
| # | Class | Defines |
| 1 | w3-padding-16 | Adds 16px top and bottom padding to an element |
| 2 | w3-padding-24 | Adds 24px top and bottom padding to an element |
| 3 | w3-padding-32 | Adds 32px top and bottom padding to an element |
| 4 | w3-padding-48 | Adds 48px top and bottom padding to an element |
| 5 | w3-padding-64 | Adds 64px top and bottom padding to an element |

<div class="w3-container">

<div class="w3-panel **w3-padding-16** w3-orange">

<h4>w3-padding-16</h4>

<p>I have 16px top and bottom padding</p>

</div>

<div class="w3-panel **w3-padding-24** w3-orange">

<h4>w3-padding-24</h4>

<p>I have 24px top and bottom padding</p>

</div>

<div class="w3-panel **w3-padding-32** w3-orange">

<h4>w3-padding-32</h4>

<p>I have 32px top and bottom padding</p>

</div>

<div class="w3-panel **w3-padding-48** w3-orange">

<h4>w3-padding-48</h4>

<p>I have 48px top and bottom padding</p>

</div>

<div class="w3-panel **w3-padding-64** w3-orange">

<h4>w3-padding-64</h4>

<p>I have 64px top and bottom padding</p>

</div>

</div>

The **w3-padding-size** classes add **top, bottom, right**, and **left** padding to any HTML element:

|  |  |  |
| --- | --- | --- |
| # | Class | Defines |
| 1 | w3-padding | Adds 8px top and bottom, and 16px left and right padding |
| 2 | w3-padding-small | Adds 4px top and bottom, and 8px left and right padding |
| 3 | w3-padding-large | Adds 12px top and bottom, and 24px left and right padding |

<div class="w3-container">

<div class="w3-panel **w3-padding-small** w3-blue">

<h4>w3-padding-small</h4>

<p>I have 4px top and bottom padding and 8px left and right padding.</p>

</div>

<div class="w3-panel **w3-padding** w3-blue">

<h4>w3-padding</h4>

<p>I have 8px top and bottom padding and 16px left and right padding.</p>

</div>

<div class="w3-panel **w3-padding-large** w3-blue">

<h4>w3-padding-large</h4>

<p>I have 12px top and bottom padding and 24px left and right padding.</p>

</div>

### W3.CSS Margin

[W3.CSS](https://www.w3schools.com/w3css/w3css_margins.asp) provides the following margin classes:

|  |  |  |
| --- | --- | --- |
| # | Class | Defines |
| 1 | w3-margin | Adds a 16px margin to all sides of an element |
| 2 | w3-margin-top | Adds a 16px top margin to an element |
| 3 | w3-margin-righ | Adds a 16px right margin to an element |
| 4 | w3-margin-bottom | Adds a 16px bottom margin to an element |
| 5 | w3-margin-left | Adds a 16px left margin to an element |
| 6 | w3-section | Adds a 16px top and bottom margin to an element |

<div class="w3-container">

<h1>Using w3-margin</h1>

<div class="w3-border">

<div class="w3-container **w3-margin** w3-green">

<p>The w3-margin class adds 16px margin to all sides of an element.</p>

</div>

</div>

<div class="w3-border">

<div class="w3-container **w3-margin-top** w3-green">

<p>The w3-margin-top class adds a 16px top margin to an element.</p>

</div>

</div>

<div class="w3-border">

<div class="w3-container **w3-margin-bottom** w3-green">

<p>The w3-margin-bottom class adds a 16px bottom margin to an element.</p>

</div>

</div>

<div class="w3-border">

<div class="w3-container **w3-margin-left** w3-green">

<p>The w3-margin-left class adds a 16px left margin to an element.</p>

</div>

</div>

<div class="w3-border">

<div class="w3-container **w3-margin-right** w3-green">

<p>The w3-margin-right class adds a 16px right margin to an element.</p>

</div>

</div>

<div class="w3-container **w3-section** w3-blue">

<h1>I am Blue</h1>

</div>

<div class="w3-container **w3-section** w3-green">

<h1>I am Green</h1>

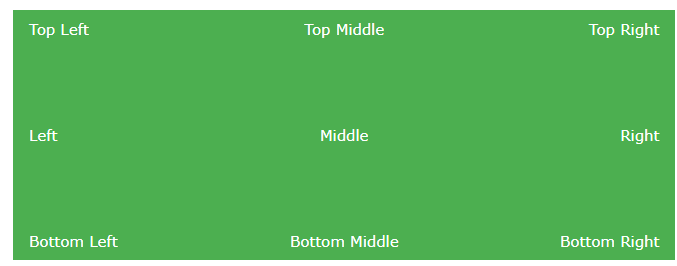
</div>

</div>

### W3.CSS Display

The display classes allow you to display HTML elements in specific positions inside other HTML elements.

W3.CSS provides the following [display classes](https://www.w3schools.com/w3css/w3css_display.asp):



|  |  |  |
| --- | --- | --- |
| # | Class | Defines |
| 1 | w3-display-container | Container for w3-display-classes |
| 2 | w3-display-topleft | Displays content at the top left corner of the container |
| 3 | w3-display-topright | Displays content at the top right corner of the container |
| 4 | w3-display-bottomleft | Displays content at the bottom left corner of the container |
| 5 | w3-display-bottomright | Displays content at the bottom right corner of the container |
| 6 | w3-display-left | Displays content to the left (middle left) of the container |
| 7 | w3-display-right | Displays content to the right (middle right) of the container |
| 8 | w3-display-middle | Displays content in the middle (center) of the container |
| 9 | w3-display-topmiddle | Displays content at the top middle of the container |
| 10 | w3-display-bottommiddle | Displays content at the bottom middle of the container |
| 11 | w3-display-position | Displays content at a specified position in the container |
| 12 | w3-display-hover | Displays content on hover inside the container |
| 13 | w3-left | Floats an element to the left (float: left) |
| 14 | w3-right | Floats an element to the right (float: right) |
| 15 | w3-show | Shows an element (display: block) |
| 16 | w3-hide | Hides an element (display: none) |
| 17 | w3-mobile | Adds mobile-first responsiveness to any element. Displays elements as block elements on mobile devices |

### W3.CSS Card

A [card](https://www.w3schools.com/w3css/w3css_cards.asp) is a flexible and extensible content container. It can include header, footer and a wide variety of content. W3.CSS provides the following classes for displaying paper-like cards:

|  |  |  |
| --- | --- | --- |
| # | Class name | Description |
| 1 | w3-card | It is used to add a box shadow of 2px to the element. |
| 2 | w3-card-2 | It is same as the w3-card class |
| 3 | w3-card-4 | It is used to add a box shadow of 4px to the element. |

To display colored cards, just add w3-color class:

<div class="w3-container">

<h2>Cards</h2>

<p>Create paper-like cards with the w3-card classes:</p>

<div class="w3-panel w3-card **w3-cyan**"><p>w3-card</p></div>

<div class="w3-panel w3-card-2 **w3-light-blue**"><p>w3-card-2</p></div>

<div class="w3-panel w3-card-4 **w3-yellow**"><p>w3-card-4</p></div>

</div>

Add containers inside the card to create different sections:

<div class="w3-card-4">

<**header** class="w3-container w3-blue">

<h1>Header</h1>

</**header**>

<**div** class="w3-container">

<p>Lorem ipsum...</p>

</**div**>

<**footer** class="w3-container w3-blue">

<h5>Footer</h5>

</**footer**>

</div>

In similar way, one can use **img** tag to insert image in the card if he/she wants.

<div class="w3-container">

<h2>Photo Card</h2>

<div class="w3-card-4" style="width:50%">

<**img** src="img\_snowtops.jpg" alt="Alps" style="width:100%">

<div class="w3-container w3-center">

<p>The Italian / Austrian Alps</p>

</div>

</div>

</div>

The **w3-hover-shadow** class adds a shadow effect on hover - this will make any element look like a card on mouse-over (same style as w3-card-4).

<div class="w3-green **w3-hover-shadow** w3-center">

<p>Hover over me!</p>

</div>

### W3.CSS Defaults

W3.CSS pages have listed below [default values](https://www.w3schools.com/w3css/w3css_defaults.asp):

* font-size is 15px,
* font is Verdana,
* line-spacing 1.5.

Default values for headings are:

<div class="w3-container">

<h1>Heading 1 (36px)</h1>

<h2>Heading 2 (30px)</h2>

<h3>Heading 3 (24px)</h3>

<h4>Heading 4 (20px)</h4>

<h5>Heading 5 (18px)</h5>

<h6>Heading 6 (16px)</h6>

</div>

Font sizes can be set by the following w3-size classes:

<div class="w3-container">

<h2>Font-Size Classes</h2>

<p class="**w3-tiny**">w3-tiny</p>

<p class="**w3-small**">w3-small</p>

<p class="**w3-medium**">w3-medium (Default)</p>

<p class="**w3-large**">w3-large</p>

<p class="**w3-xlarg**e">w3-xlarge</p>

<p class="**w3-xxlarge**">w3-xxlarge</p>

<p class="**w3-xxxlarge**">w3-xxxlarge</p>

<p class="**w3-jumbo**">w3-jumbo</p>

</div>

You have three options for overriding default values:

1. Override the default settings in the <head> section of your page

<!DOCTYPE html>

<html>

<title>W3.CSS</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<style>

h1 {font-size: 64px;}

h2 {font-size: 48px;}

h3 {font-size: 36px;}

h4 {font-size: 24px;}

h5 {font-size: 20px;}

h6 {font-size: 18px;}

</style>

<body>

...

</body>

</html>

1. Add your own CSS file after W3.CSS

<!DOCTYPE html>

<html>

<title>W3.CSS</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<link rel="stylesheet" href="myStyle.css">

<body>

...

</body>

</html>

1. Download and change the content of the W3.CSS file.

### W3.CSS Fonts

Verdana is the default font used in [W3.CSS](https://www.w3schools.com/w3css/w3css_fonts.asp). Segoe UI is the default font for headings. It has a more narrow letter spacing. Using another default font in W3.CSS is very easy:

<!DOCTYPE html>

<html>

<title>W3.CSS</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<style>

body, h1, h2, h3, h4, h5, h6 {

font-family: Arial, Helvetica, sans-serif ;

}

</style>

<body>

...

</body>

</html>

With W3.CSS you can use 4 different built-in font classes:

1. w3-serif
2. w3-sans-serif
3. w3-monospace
4. w3-cursive

An example of these styles is provided on [W3C](https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_fonts_sans-serif) site. With W3.CSS it is extremely easy to add new fonts. Here is an example of including Roboto font from Google library:

<!DOCTYPE html>

<html>

<title>W3.CSS</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

**<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto">**

<style>

**body, h1, h2, h3, h4, h5, h6 {**

**font-family: Roboto, sans-serif;**

**}**

</style>

<body>

<div class="w3-container">

<h2>Font Roboto</h2>

<p>

Norway has a total area of 385,252 square kilometers and a population of 5,438,657 (December 2020). Norway is bordered by Sweden, Finland and Russia to the north-east, and the Skagerrak to the south, with Denmark on the other side.

</p>

<p class="w3-large">

Norway has a total area of 385,252 square kilometers and a population of 5,438,657 (December 2020). Norway is bordered by Sweden, Finland and Russia to the north-east, and the Skagerrak to the south, with Denmark on the other side.

</p>

</div>

</body>

</html>

It is possible to create a font class and mark elements on the page:

<!DOCTYPE html>

<html>

<title>W3.CSS</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia">

<style>

.w3-sofia {

font-family: Sofia, cursive;

}

</style>

<body>

<div class="w3-container w3-sofia">

<p class="w3-xlarge">Making the Web</p>

<p class="w3-xxlarge">Making the Web</p>

<p class="w3-xxxlarge">Making the Web</p>

</div>

</body>

</html>

Look at Google fonts are listed on their [site](https://fonts.google.com/).

### W3.CSS Text

The **w3-left-align** and the **w3-right-align** classes are used to align text.

<div class="w3-container">

<h2>Text Alignments</h2>

<div class="w3-container w3-border w3-large">

<div class="**w3-left-align**"><p>Left aligned text.</p></div>

<div class="**w3-right-alig**n"><p>Right aligned text.</p></div>

</div>

</div>

The **w3-center** class is used to center-align elements:

<div class="w3-container **w3-center**">

<h2>Centered Content</h2>

<img src="img\_car.jpg" alt="car" style="width:80%;max-width:320px">

<p>Some centered text.</p>

</div>

The **w3-wide** class specifies a wider text:

<div class="w3-container">

<h2>Wider Text</h2>

<p>The w3-wide class specifies a wider text.</p>

<p class="**w3-wide**">The w3-wide class specifies a wider text.</p>

</div>

The **w3-opacity** class is designed to work with all colors:

<div class="w3-container">

<h2>Text Opacity</h2>

<div class="w3-panel w3-pink">

<h2 class="**w3-opacity**">Text Opacity</h2>

</div>

<div class="w3-panel w3-blue">

<h2 class="**w3-opacity**">Text Opacity</h2>

</div>

<div class="w3-panel w3-orange">

<h2 class="**w3-opacity**">Text Opacity</h2>

</div>

<div class="w3-panel w3-black">

<h2 class="**w3-opacity**">Text Opacity</h2>

</div>

</div>

The CSS3 [text-shadow](https://developer.mozilla.org/en-US/docs/Web/CSS/text-shadow) property can be used to add shadow or blur effects to text:

<div class="w3-container w3-light-grey">

<h2 style="text-shadow:1px 1px 0 #444">Text Shadow</h2>

</div>

<div class="w3-container w3-blue">

<h2 style="text-shadow:1px 1px 0 #444">Text Shadow</h2>

</div>

<div class="w3-container w3-green">

<h2 style="text-shadow:1px 1px 0 #444">Text Shadow</h2>

</div>

Text effects may be combined:

<div class="w3-container">

<h2>Special Effects</h2>

<div class="w3-panel w3-pink">

<h1 class="w3-opacity">

<b>Text Opacity + Bold</b></h1>

</div>

<div class="w3-panel w3-amber">

<h1 class="w3-text-yellow" style="text-shadow:1px 1px 0 #444">

<b>Yellow Text + Shadow + Bold</b></h1>

</div>

<div class="w3-panel w3-blue">

<h1 class="w3-text-orange" style="text-shadow:1px 1px 0 #444">

<b>Orange Text + Shadow + Bold</b></h1>

</div>

</div>

[Docker with CGI perl. hello world tutorial | by lojorider lojo | Medium](https://medium.com/@lojorider/docker-with-cgi-perl-a4558ab6a329)

[SQL-Ledger ERP](https://www.sql-ledger.com/cgi-bin/nav.pl?page=source/index.html&title=Download) (download)

[SQL-Ledger ERP](https://www.sql-ledger.com/cgi-bin/nav.pl?page=source/readme.txt&title=README) (install)

[step\_by\_step\_installation:ubuntu [International SQL-Ledger Network Association] (sql-ledger-network.com)](https://www.sql-ledger-network.com/dokuwiki/doku.php?id=step_by_step_installation:ubuntu)

[Tutorial - Perl CGI on Apache [ Step by step ] (techexpert.tips)](https://techexpert.tips/apache/perl-cgi-apache/) //veikia

[Ubuntu 18.04 LTS : Apache2 : Use Perl Scripts : Server World (server-world.info)](https://www.server-world.info/en/note?os=Ubuntu_18.04&p=httpd&f=2)

[GitHub - Tekki/sql-ledger: :books: The official SQL-Ledger version, plus some additions.](https://github.com/Tekki/sql-ledger)

[bootstrap CDN by jsDelivr - A CDN for npm and GitHub](https://www.jsdelivr.com/package/npm/bootstrap)

https://www.alexedwards.net/blog/serving-static-sites-with-go

### Responsive Design

Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones). To create a responsive website, add the following **<meta>** tag to all your web pages:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

This tag will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

#### Responsive Images

If the CSS width property is set to 100%, the image will be responsive and scale up and down.

<img src="img\_girl.jpg" style="width:100%;">

The image can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the max-width property instead.

<img src="img\_girl.jpg" style="max-width:100%;height:auto;">

If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size.

#### Show Different Images Depending on Browser Width

The HTML <picture> element allows you to define different images for different browser window sizes.

<picture>

<source srcset="img\_smallflower.jpg" media="(max-width: 600px)">

<source srcset="img\_flowers.jpg" media="(max-width: 1500px)">

<img src="img\_smallflower.jpg" alt="Flowers">

</picture>

The <picture> element contains two tags: one or more **<source**> tags and one **<img**> tag. The browser will look for the first <source> element where the media query matches the current viewport width, and then it will display the proper image (specified in the srcset attribute). The **<img>** element is required as the last child of the **<picture>** element, as a fallback option if none of the source tags matches.

#### Responsive Text Size

The text size can be set with a "vw" unit, which means the "viewport width".

That way the text size will follow the size of the browser window:

<h1 style="font-size:10vw">Hello World</h1>

1vw = 1% of viewport width. If the viewport is 50cm wide, 1vw is 0.5cm.

#### CSS relative units

**Pixels** are absolute units. They won't change when you resize the window.

**EM** and **REM** are not units. They are multipliers of a base number. The REM unit depends on the root element [the HTML element]. The default font-size of the root element [in HTML] is 16px. So, 1 REM = 16px. If we change the root element font size, the REM unit changes also.

Making responsive websites with the REM unit is very easy. Just write your styles in rem units instead of the pixels and change the root elements at different breakpoints using media queries.

// large screen

@media (max-width: 1400px) {

html {

font-size: 25px;

}

}

// Tablet screen

@media (max-width: 768px) {

html {

font-size: 18px;

}

}

// Mobile screen

@media (max-width: 450px) {

html {

font-size: 12px;

}

}

The **EM** unit is the same as the REM unit but it depends on the **parent** font size.

The full form of **VW** is viewport width. It works like the percentage unit. 1vw is equal to 1% of entire visible screen width.

The full form of **VH** is viewport height. It works like the percentage unit as well. Specifying 10vh is equivalent to occupying 10% of entire visible screen height.

Size expressed in VW or VH does not change on resizing the window.

#### The External Resource Link

The [<link>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link) HTML element specifies relationships between the current document and an external resource. This element is most commonly used to link to stylesheets, but is also used to establish site icons:

<link href="main.css" rel="stylesheet" />

<link href="favicon.ico" rel="icon" />

There are a number of other icon rel values, mainly used to indicate special icon types for use on various mobile platforms:

<link

rel="apple-touch-icon"

sizes="114x114"

href="apple-icon-114.png"

type="image/png" />

The sizes attribute indicates the icon size, while the type contains the MIME type of the resource being linked. You can also provide a media type or query inside a media attribute; this resource will then only be loaded if the media condition is true:

<link href="print.css" rel="stylesheet" media="print" />

<link

href="mobile.css"

rel="stylesheet"

media="screen and (max-width: 600px)" />

A **<link>** element can occur either in the **<head>** or **<body>** element, depending on whether it has a link type that is [body-ok](https://html.spec.whatwg.org/multipage/links.html" \l "body-ok).

#### Using media queries

[Media queries](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries/Using_media_queries) allow you to apply CSS styles depending on a device's media type (such as print vs. screen) or other features or characteristics such as screen resolution or orientation, aspect ratio, browser viewport width or height, user preferences such as preferring reduced motion, data usage, or transparency.

Media queries are used for the following:

* To conditionally apply styles with the [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) [@media](https://developer.mozilla.org/en-US/docs/Web/CSS/@media) and [@import](https://developer.mozilla.org/en-US/docs/Web/CSS/@import) [at-rules](https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule).
* To target specific media for the [<style>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/style), [<link>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link), [<source>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link), and other HTML elements with the **media=** or **sizes="** attributes.
* To [test and monitor media states](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link) using the [Window.matchMedia()](https://developer.mozilla.org/en-US/docs/Web/API/Window/matchMedia) and [EventTarget.addEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener) methods.

A media query is composed of an optional **media type** and any number of **media feature expressions**, which may optionally be combined in various ways using logical operators. Media queries are case-insensitive.

* **Media ty**pes define the broad category of device for which the media query applies: **all**, **print**, **screen**. The type is optional (assumed to be all) except when using the **only** logical operator.
* **Media features** describe a specific characteristic of the user agent, output device, or environment:
  + [any-hover](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/any-hover) feature is specified as a keyword value (**none** or **hover**)

<a href="#">Try hovering over me!</a>

...

@media (**any-hover: hover**) {

a:hover {

background: yellow;

}

}

* + The [any-pointer](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/any-pointer) CSS media feature tests whether the user has any pointing device (such as a mouse), and if so, how accurate it is. The any-pointer feature is specified as a keyword value chosen from the list: **none**, **coarse**, **fine**. If you want to test the accuracy of the primary pointing device, use pointer instead.

<input id="test" type="checkbox" /> <label for="test">Look at me!</label>

...

input[type="checkbox"]:checked {

background: gray;

}

@media (**any-pointer: fine**) {

input[type="checkbox"] {

appearance: none;

width: 15px;

height: 15px;

border: 1px solid blue;

}

}

@media (**any-pointer: coarse**) {

input[type="checkbox"] {

appearance: none;

width: 30px;

height: 30px;

border: 2px solid red;

}

}

* + The [aspect-ratio](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/aspect-ratio) CSS media feature can be used to test the aspect ratio of the viewport. The aspect-ratio feature is specified as a [<ratio>](https://developer.mozilla.org/en-US/docs/Web/CSS/aspect-ratio) value representing the width-to-height aspect ratio of the viewport. It is a range feature, meaning you can also use the prefixed min-aspect-ratio and max-aspect-ratio variants to query minimum and maximum values, respectively.

<div id="inner">

Watch this element as you resize your viewport's width and height.

</div>

...

/\* Minimum aspect ratio \*/

@media (**min-aspect-ratio: 8/5**) {

div {

background: #9af; /\* blue \*/

}

}

/\* Maximum aspect ratio \*/

@media (**max-aspect-ratio: 3/2**) {

div {

background: #9ff; /\* cyan \*/

}

}

/\* Exact aspect ratio, put it at the bottom to avoid override\*/

@media (aspect-ratio: 1/1) {

div {

background: #f9a; /\* red \*/

}

}

* + The [color](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/color) CSS media feature can be used to test the number of bits per color component (red, green, blue) of the output device. The color feature is specified as an <integer> value that represents the number of bits per color component (red, green, blue) of the output device. If the device is not a color device, the value is zero. It is a range feature, meaning that you can also use the prefixed **min-color** and **max-color** variants to query minimum and maximum values, respectively. If the various color components are represented by different numbers of bits, the smallest number is used.

<p>

This text should be black on non-color devices, red on devices with a low

number of colors, and greenish on devices with a high number of colors.

</p>

...

p {

color: black;

}

/\* Any color device \*/

@media (color) {

p {

color: red;

}

}

/\* Any color device with at least 8 bits per color component \*/

@media (min-color: 8) {

p {

color: #24ba13;

}

}

* + The [color-gamut](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/color-gamut) CSS media feature is used to apply CSS styles based on the approximate range of color gamut supported by the user agent and the output device. A **color gamut** is a subset of colors, usually representing the colors that a display or a printing device can represent. Traditionally, in web development, the only gamut used was [sRGB](https://en.wikipedia.org/wiki/SRGB) (Standard Red-Green-Blue), where each color is described using three bytes, one for each primary color. However, "wide-color" monitors and professional printers support a wider range of colors, that can't be represented using this gamut. Since 2021, browsers have started to provide functionality for other gamuts, like [P3](https://en.wikipedia.org/wiki/DCI-P3), widely used in the movie industry, and rec2020. The **color-gamut** feature is specified as one of the following color spaces as keyword values:
    - [srgb](https://developer.mozilla.org/en-US/docs/Glossary/Color_space" \l "srgb) - the user agent and the output device can support approximately the sRGB gamut or more. This includes the vast majority of color displays
    - [p3](https://www.color.org/chardata/rgb/DisplayP3.xalter) - the user agent and the output device can support approximately the gamut specified by the Display P3 color space or more. The P3 gamut is larger than and includes the sRGB gamut.
    - [rec2020](https://en.wikipedia.org/wiki/Rec._2020) - the user agent and the output device can support approximately the gamut specified by the ITU-R Recommendation BT.2020 color space or more. The REC. 2020 gamut is larger than and includes the P3 gamut.

<p>This is a test.</p>

...

p {

padding: 10px;

border: solid;

}

@media (color-gamut: srgb) {

p {

background: #f4ae8a;

}

}

* + The [color-index](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/color-index) feature is specified as an <integer> value representing the number of entries in the output device's color lookup table. (This value is zero if the device does not use such a table.) It is a range feature, meaning that you can also use the prefixed **min-color-index** and **max-color-index** variants to query minimum and maximum values, respectively.

<p>This is a test.</p>

...

p {

color: black;

}

@media (**color-index**) {

p {

color: red;

}

}

@media (**min-color-index**: 15000) {

p {

color: #1475ef;

}

}

* + The [display-mode](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/display-mode) CSS media feature can be used to test whether a web app is being displayed in a normal browser tab or in some alternative way, such as a standalone app or fullscreen mode. The display-mode value applies to the top-level browsing context and any child browsing contexts. The display-mode feature is specified as a keyword value chosen from the list below:
    - **browser** - the application opens in a conventional browser tab or new window, depending on the browser and platform.
    - **fullscreen** - all of the available display area is used and no user agent chrome is shown. This can be used to apply CSS only when the app has been put in fullscreen mode by the [Fullscreen API](https://developer.mozilla.org/en-US/docs/Web/API/Fullscreen_API) or by using the fullscreen value of the [display](https://developer.mozilla.org/en-US/docs/Web/Manifest/display) member of the Wep App Manifest.
    - **minimal-ui** - the application will look and feel like a standalone application, but will have a minimal set of UI elements for controlling navigation. The elements will vary by browser.
    - **picture-in-picture** - this mode allows users to continue consuming specific content while they interact with other sites or applications on their device. The app is displayed in a floating and always-on-top window. This can be used to apply CSS only when the app has been put in Picture-in-Picture mode by the [Document Picture-in-Picture API.](https://developer.mozilla.org/en-US/docs/Web/API/Document_Picture-in-Picture_API)
    - **standalone** - the application will look and feel like a standalone application. This can include the application having a different window, its own icon in the application launcher, etc. In this mode, the user agent will exclude UI elements for controlling navigation, but can include other UI elements such as a status bar.
    - **window-controls-overlay** - in this mode, the application looks and feels like a standalone desktop application, and the Window Controls Overlay feature is enabled.

@media (display-mode: picture-in-picture) and (prefers-color-scheme: light) {

body {

background: antiquewhite;

}

}

@media (display-mode: picture-in-picture) and (prefers-color-scheme: dark) {

body {

background: #333;

}

a {

color: antiquewhite;

}

}

* + The [dynamic-range](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/dynamic-range) CSS media feature can be used to test the combination of brightness, contrast ratio, and color depth that are supported by the user agent and the output device. Some devices have high dynamic range capabilities that are not always 'on' and need to be activated (sometimes programmatically, sometimes by the user, sometimes based on the content). This media feature does not test whether the dynamic range capability is active; it only tests whether the device is capable of high dynamic range visuals. The dynamic-range feature is specified as a keyword value chosen from the list below:
    - **standard** - this value matches any visual device and excludes devices lacking visual capabilities. A user agent or an output device that matches high will also match the standard value.
    - **high** - this value matches user agents and output devices that support high peak brightness, high contrast ratio, and color depth greater than 24 bit or 8 bit per color component of RGB.

@media (**dynamic-range: standard**) {

p {

color: red;

}

}

@media (**dynamic-range: high**) {

p {

color: green;

}

}

* + The [forced-colors](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/forced-colors) CSS media feature is used to detect if the user agent has enabled a forced colors mode where it enforces a user-chosen limited color palette on the page. An example of a forced colors mode is Windows High Contrast mode. The forced-colors media feature indicates whether or not the browser is currently in forced-colors mode. Read about forced-query feature on [Polypane](https://polypane.app/blog/forced-colors-explained-a-practical-guide/) site. These values may be defined in the query:
    - **none** - forced colors mode is not active; the page's colors are not being forced into a limited palette.
    - **active** - indicates that forced colors mode is active. The browser provides the color palette to authors through the [CSS system color](https://developer.mozilla.org/en-US/docs/Web/CSS/system-color) keywords and, if appropriate, triggers the appropriate value of prefers-color-scheme so that authors can adapt the page. The browser selects the value for [prefers-color-scheme](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/prefers-color-scheme) based on the lightness of the Canvas system color (see the color adjust spec for more details).

@media (**forced-colors: active**) {

/\* forced colors are active \*/

}

@media (**forced-colors: active**) and (**prefers-color-scheme: dark**) {

/\* High Contrast #1, #2 or Black is active (or custom theme) \*/

}

@media (**forced-colors: active**) and (**prefers-color-scheme: light**) {

/\* High Contrast White is active (or custom theme) \*/

}

* + The [grid](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/grid) CSS media feature can be used to test whether the output device uses a **grid-based** screen. Most modern computers and smartphones have **bitmap-based** screens. Examples of grid-based devices include text-only terminals and basic phones with only one fixed font. The grid feature is specified as a <mq-boolean> value (0 or 1) representing whether or not the output device is grid-based.

<p class="unknown">I don't know if you're using a grid device. :-(</p>

<p class="bitmap">You are using a bitmap device.</p>

<p class="grid">You are using a grid device! Neato!</p>

...

:not(.unknown) {

color: lightgray;

}

@media (grid: 0) {

.unknown {

color: lightgray;

}

.bitmap {

color: red;

text-transform: uppercase;

}

}

@media (grid: 1) {

.unknown {

color: lightgray;

}

.grid {

color: black;

text-transform: uppercase;

}

}

* + The [height](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/height) CSS media feature can be used to apply styles based on the height of the viewport (or the page box, for [paged media](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_paged_media)). The height feature is specified as a <length> value representing the viewport height. It is a range feature, meaning that you can also use the prefixed **min-height** and **max-height** variants to query minimum and maximum values, respectively.

<div>Watch this element as you resize your viewport's height.</div>

...

/\* Exact height \*/

@media (**height: 360px**) {

div {

color: red;

}

}

/\* Minimum height \*/

@media (**min-height: 25rem**) {

div {

background: yellow;

}

}

/\* Maximum height \*/

@media (**max-height: 40rem**) {

div {

border: 2px solid blue;

}

}

* + The [hover](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/hover) CSS media feature can be used to test whether the user's primary input mechanism can hover over elements. The hover feature is specified as a keyword value chosen from the list below:
    - **none** - the primary input mechanism cannot hover at all or cannot conveniently hover (e.g., many mobile devices emulate hovering when the user performs an inconvenient long tap), or there is no primary pointing input mechanism.
    - **hover** - the primary input mechanism can conveniently hover over elements.

<a href="#">Try hovering over me!</a>

...

/\* default hover effect \*/

a:hover {

color: black;

background: yellow;

}

@media (**hover: hover**) {

/\* when hover is supported \*/

a:hover {

color: white;

background: black;

}

}

* + The [inverted-colors](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/inverted-colors) CSS media feature is used to test if the user agent or the underlying operating system has inverted all colors. Inversion of colors can have unpleasant side effects, such as shadows turning into highlights, which can reduce the readability of the content. Using this media feature, you can detect if inversion is happening and style the content accordingly while respecting user preference. This WEB [site](https://www.youtube.com/watch?v=59uBJlXr6Xs) explains how to invert colors on Windows 11. The inverted-colors feature is specified as one of the following keyword values:
    - **none** - indicates that the colors are displayed normally and no inversion of colors has happened. This keyword value evaluates as false.
    - **inverted** - indicates that all pixels within the displayed area have been inverted. This keyword value evaluates as true.

<p>

If color inversion is detected, this text will appear blue on white (the

inverse of yellow on black) along with a line over the text. If no color

inversion is happening, the text will appear red on light gray without the

line over the text.

</p>

<p>

If the text is gray and no overline is present, it means your browser doesn't

support the

<code>inverted-colors</code> media feature.

</p>

...

p {

color: gray;

}

@media (inverted-colors: inverted) {

p {

background: black;

color: yellow;

text-decoration: overline;

}

}

@media (inverted-colors: none) {

p {

background: #eee;

color: red;

}

}

* + The [monochrome](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/monochrome) CSS media feature can be used to test the number of bits per pixel in the monochrome frame buffer of the output device. The **monochrome** feature is specified as an <integer> representing the number of bits per pixel in the monochrome frame buffer. If the device is not a monochrome device, the value is zero. It is a range feature, meaning that you can also use the prefixed **min-monochrome** and **max-monochrome** variants to query minimum and maximum values, respectively.

<p class="mono">Your device supports monochrome pixels!</p>

<p class="no-mono">Your device doesn't support monochrome pixels.</p>

...

p {

display: none;

}

/\* Any monochrome device \*/

@media (**monochrome**) {

p.mono {

display: block;

color: #333;

}

}

/\* Any non-monochrome device \*/

@media (**monochrome: 0**) {

p.no-mono {

display: block;

color: #ee3636;

}

}

* + The [orientation](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/orientation) CSS media feature can be used to test the orientation of the viewport (or the page box, for paged media). Note: This feature does not correspond to device orientation. Opening the soft keyboard on many devices in portrait orientation will cause the viewport to become wider than it is tall, thereby causing the browser to use landscape styles instead of portrait. The orientation feature is specified as a keyword value chosen from the list below:
    - **portrait** - the viewport is in a portrait orientation, i.e., the height is greater than or equal to the width.
    - **landscape** - the viewport is in a landscape orientation, i.e., the width is greater than the height.

<div>Box 1</div>

<div>Box 2</div>

<div>Box 3</div>

...

body {

display: flex;

}

div {

background: yellow;

width: 200px;

height: 200px;

margin: 0.5rem;

padding: 0.5rem;

}

@media (**orientation: landscape**) {

body {

flex-direction: row;

}

}

@media (**orientation: portrait**) {

body {

flex-direction: column;

}

}

* + The [overflow-block](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/overflow-block) CSS media feature can be used to test how the output device handles content that overflows the initial containing block along the block axis. The **overflow-block** feature is specified as a keyword value chosen from the list below:
    - **none** - content that overflows the block axis is not displayed.
    - **scroll** - content that overflows the block axis can be seen by scrolling to it.
    - **optional-paged** - content that overflows the block axis can be seen by scrolling to it, but page breaks can be manually triggered (such as via [break-inside](https://developer.mozilla.org/en-US/docs/Web/CSS/break-inside), etc.) to cause the following content to display on the following page.
    - **paged** - content is broken up into discrete pages; content that overflows one page in the block axis is displayed on the following page.
  + The [overflow-inline](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/overflow-inline) feature is specified as a keyword value chosen from the list below.
    - **none** - content that overflows the inline axis is not displayed.
    - **scroll** - content that overflows the inline axis can be seen by scrolling to it.

<p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ac turpis

eleifend, fringilla velit ac, aliquam tellus. Vestibulum ante ipsum primis in

faucibus orci luctus et ultrices posuere cubilia Curae; Nunc velit erat,

tempus id rutrum sed, dapibus ut urna. Integer vehicula nibh a justo imperdiet

rutrum. Nam faucibus pretium orci imperdiet sollicitudin. Nunc id facilisis

dui. Proin elementum et massa et feugiat. Integer rutrum ullamcorper eleifend.

Proin sit amet tincidunt risus. Sed nec augue congue eros accumsan tincidunt

sed eget ex.

</p>

...

p {

white-space: nowrap;

}

@media (overflow-inline: scroll) {

p {

color: red;

}

}

* + The [pointer](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/pointer) CSS media feature tests whether the user has a pointing device (such as a mouse), and if so, how accurate the primary pointing device is. If you want to test the accuracy of any pointing device, use [any-pointer](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/any-pointer) instead. The pointer feature is specified as a keyword value chosen from the list below:
    - **none** - the primary input mechanism does not include a pointing device.
    - **coarse** - the primary input mechanism includes a pointing device of limited accuracy, such as a finger on a touchscreen.
    - **fine** - the primary input mechanism includes an accurate pointing device, such as a mouse.

<input id="test" type="checkbox" /> <label for="test">Look at me!</label>

...

input[type="checkbox"]:checked {

background: gray;

}

@media (any-pointer: fine) {

input[type="checkbox"] {

appearance: none;

width: 15px;

height: 15px;

border: 1px solid blue;

}

}

@media (any-pointer: coarse) {

input[type="checkbox"] {

appearance: none;

width: 30px;

height: 30px;

border: 2px solid red;

}

}

* + The [prefers-color-scheme](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/prefers-color-scheme) CSS media feature is used to detect if a user has requested light or dark color themes. A user indicates their preference through an operating system setting (e.g. light or dark mode) or a user agent setting.

For **SVG** and **iframes**, prefers-color-scheme lets you set a CSS style for the SVG or iframe based on the color-scheme of the parent element in the web page. SVGs must be used embedded (i.e., **<img src="circle.svg" alt="circle" />**) as opposed to [inlined](https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/SVG_In_HTML_Introduction" \l "basic_example) in HTML. Using prefers-color-scheme is allowed in cross-origin <svg> and <iframe> elements. **Cross-origin** elements are elements retrieved from a different host than the page that is referencing them.

The prefers-color-scheme feature is specified as a keyword value chosen from the list below:

* + - **light** - indicates that user has notified that they prefer an interface that has a light theme, or has not expressed an active preference.
    - **dark** - indicates that user has notified that they prefer an interface that has a dark theme.

<div class="box theme-a">Theme A (initial)</div>

<div class="box theme-a adaptive">Theme A (changed if dark preferred)</div>

<br />

<div class="box theme-b">Theme B (initial)</div>

<div class="box theme-b adaptive">Theme B (changed if light preferred)</div>

...

.theme-a {

background: #dca;

color: #731;

}

@media (**prefers-color-scheme: dark**) {

.theme-a.adaptive {

background: #753;

color: #dcb;

outline: 5px dashed #000;

}

}

.theme-b {

background: #447;

color: #bbd;

}

@media (**prefers-color-scheme: light**) {

.theme-b.adaptive {

background: #bcd;

color: #334;

outline: 5px dotted #000;

}

}

* + The [prefers-contrast](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/prefers-contrast) CSS media feature is used to detect whether the user has requested the web content to be presented with a lower or higher contrast. The **prefers-contrast** feature is specified as a keyword value chosen from the list below:
    - **no-preference** - indicates that the user has made no preference known to the system. This keyword value evaluates as false in the Boolean context.
    - **more - indicates** that user has notified the system that they prefer an interface that has a higher level of contrast.
    - **less** - indicates that user has notified the system that they prefer an interface that has a lower level of contrast.
    - **custom** - indicates that user has notified the system for using a specific set of colors, and the contrast implied by these colors matches neither more nor less. This value will match the color palette specified by users of forced-colors: active.

<div class="contrast">low contrast box</div>

...

.contrast {

width: 100px;

height: 100px;

outline: 2px dashed black;

}

@media (**prefers-contrast: more**) {

.contrast {

outline: 2px solid black;

}

}

* + The [prefers-reduced-motion](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/prefers-reduced-motion) CSS media feature is used to detect if a user has enabled a setting on their device to minimize the amount of non-essential motion. The setting is used to convey to the browser on the device that the user prefers an interface that removes, reduces, or replaces motion-based animations. Such animations can trigger discomfort for those with vestibular motion disorders. Animations such as scaling or panning large objects can be vestibular motion triggers. The **prefers-reduced-motion** feature is specified as a keyword value chosen from the list below:
    - **no-preference** - indicates that a user has made no preference known on the device. This keyword value evaluates as false.
    - **reduce** - indicates that a user has enabled the setting on their device for reduced motion. This keyword value evaluates as true.

<div class="animation">animated box</div>

...

.animation {

animation: pulse 1s linear infinite both;

background-color: purple;

}

/\* Tone down the animation to avoid vestibular motion triggers. \*/

@media (**prefers-reduced-motion**) {

.animation {

animation: dissolve 4s linear infinite both;

background-color: green;

text-decoration: overline;

}

}

* + The [resolution](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/resolution) CSS media feature can be used to test the pixel density of the output device. The resolution feature is specified as a <resolution> value representing the pixel density of the output device. It is a range feature, meaning that you can also use the prefixed min-resolution and max-resolution variants to query minimum and maximum values, respectively. The [<resolution>](https://developer.mozilla.org/en-US/docs/Web/CSS/resolution) CSS data type, used for describing resolutions in media queries, denotes the pixel density of an output device, i.e., its resolution. The **<resolution>** data type consists of a strictly positive <number> followed by one of the units listed below. As with all CSS dimensions, there is no space between the unit literal and the number:
    - **dpi** - represents the number of dots per inch. Screens typically contains 72 or 96 dots per inch, but the dpi for printed documents is usually much greater. As 1 inch is 2.54 cm, 1dpi ≈ 0.39dpcm.
    - **dpcm** - represents the number of dots per centimeter. As 1 inch is 2.54 cm, 1dpcm ≈ 2.54dpi.
    - **dppx** - represents the number of dots per px unit. Due to the 1:96 fixed ratio of CSS in to CSS px, 1dppx is equivalent to 96dpi, which corresponds to the default resolution of images displayed in CSS as defined by image-resolution.
    - **x** - alias for dppx.

<p>This is a test of your device's pixel density.</p>

...

/\* Exact resolution with unit `dpi` \*/

@media (**resolution: 150dpi**) {

p {

color: red;

}

}

/\* Minimum resolution synonym units: `dppx` and `x` \*/

@media (**min-resolution: 2dppx**) {

p {

text-decoration: underline;

}

}

@media (**min-resolution: 2x**) {

p {

text-decoration: underline;

}

}

/\* Maximum resolution with unit `dpcm` \*/

@media (**max-resolution: 2dpcm**) {

p {

background: yellow;

}

}

* + The [scripting](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/scripting) feature is specified as a keyword value chosen from the list below:
    - **none** - scripting is completely unavailable on the current document.
    - **initial-only** - scripting is enabled during the initial page load, but not afterwards.
    - **enabled** - scripting is supported and active on the current document.

<p class="script-none">You do not have scripting available. :-(</p>

<p class="script-initial-only">

Your scripting is only enabled during the initial page load. Weird.

</p>

<p class="script-enabled">You have scripting enabled! :-)</p>

...

p {

color: lightgray;

}

@media (scripting: none) {

.script-none {

color: red;

}

}

@media (scripting: initial-only) {

.script-initial-only {

color: red;

}

}

@media (scripting: enabled) {

.script-enabled {

color: red;

}

}

* + The [update](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/update) CSS media feature can be used to test how frequently (if at all) the output device is able to modify the appearance of content once rendered. The update feature is specified as a single keyword value chosen from the list below:
    - **none** - once it has been rendered, the layout can no longer be updated. Example: documents printed on paper.
    - **slow** - the layout may change dynamically according to the usual rules of CSS, but the output device is not able to render or display changes quickly enough for them to be perceived as a smooth animation. Examples: e-book readers or severely underpowered devices.
    - **fast** - the layout may change dynamically according to the usual rules of CSS, and the output device is not unusually constrained in speed, so regularly-updating things like CSS Animations can be used. Example: computer screens.

<p>

If this text animates for you, your browser supports `update` and you are

using a fast-updating device.

</p>

...

@keyframes jiggle {

from {

transform: translateY(0);

}

to {

transform: translateY(25px);

}

}

@media (**update: fast**) {

p {

animation: 1s jiggle linear alternate infinite;

}

}

* + The [width](https://developer.mozilla.org/en-US/docs/Web/CSS/@media/width) CSS media feature can be used to test the width of the viewport (or the page box, for paged media). The width feature is specified as a [<length>](https://developer.mozilla.org/en-US/docs/Web/CSS/length) value representing the viewport width. It is a range feature, meaning that you can also use the prefixed min-width and max-width variants to query minimum and maximum values, respectively.

<div>Watch this element as you resize your viewport's width.</div>

...

/\* Exact width \*/

@media (width: 360px) {

div {

color: red;

}

}

/\* Minimum width \*/

@media (min-width: 35rem) {

div {

background: yellow;

}

}

/\* Maximum width \*/

@media (max-width: 50rem) {

div {

border: 2px solid blue;

}

}

* bbb

// https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\_media\_queries/Using\_media\_queries

// [Responsive design - Learn web development | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design)

// [Using media queries - CSS: Cascading Style Sheets | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries/Using_media_queries)

// [<link>: The External Resource Link element - HTML: HyperText Markup Language | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link)

// https://developer.mozilla.org/en-US/docs/Glossary/User\_agent

## WebPack

**WebPack** examines all of the modules in your application, creates a dependency graph, then intelligently puts all of them together into one or more bundle(s) that your index.html file can reference. Importantly, **WebPack** not only collects all the files into one big file, but also arranges them in the right order. I.e. if File a.xxx has a reference to file b.xxx, then b.xxx will be placed before a.xxx in the generated file. WebPack can also perform the necessary transformations: SASS/LESS to regular CSS, TypeScript to JS, and so on.

#### Installation

sudo apt -y install webpack

**WebPack** is launched via **node**, so a sufficient number of packages will be installed additionally. Windows users using **WebPack** using **npm**:

npm install webpack webpack-cli –save-dev

Check presence of the **npm** package and install it if you don't have **npm:**

npm –version

sudo apt install npm

https://ui.dev/webpack

## Material design