# WEB App Dev

[WEB App Dev 1](#_Toc31350)

[Frontend Scripts and Languages 2](#_Toc3317)

[Html 2](#_Toc15919)

[CSS 3](#_Toc11853)

[JS 9](#_Toc22720)

[WEB API 15](#_Toc5744)

[References 15](#_Toc24051)

[Concepts 16](#_Toc7846)

[API Questions 16](#_Toc3081)

[Getting started 16](#_Toc5593)

[Testing 17](#_Toc8496)

[API security 17](#_Toc26821)

[Authentication and authorization 18](#_Toc26991)

[Claims 23](#_Toc25762)

[Advanced Authorization 23](#_Toc12581)

[Versoning 24](#_Toc12345)

[Monitoring 24](#_Toc26704)

[Protecting 24](#_Toc13956)

[Building 24](#_Toc25906)

[Consuming 24](#_Toc22331)

[Open API 24](#_Toc26903)

[Minimal API 24](#_Toc5387)

[Best practices 24](#_Toc24645)

[Misc and Questions 24](#_Toc30281)

[MVC 25](#_Toc24150)

[Razor Pages 25](#_Toc13396)

[Model 25](#_Toc8854)

[View 25](#_Toc30480)

[Controller 25](#_Toc3094)

[Routing 25](#_Toc1302)

## Frontend Scripts and Languages

### Html



#### Basic structure



#### Attributes

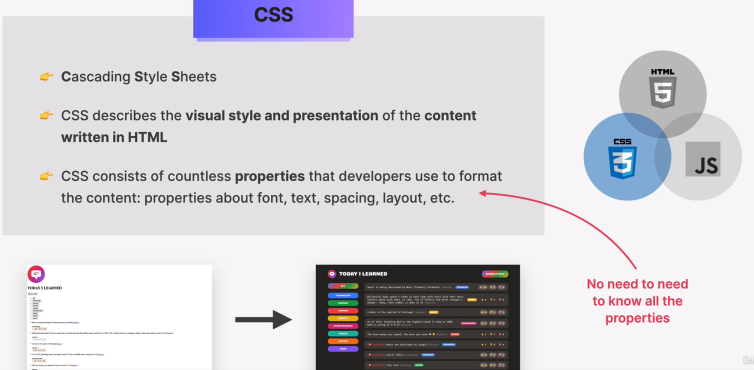
<img src="img\_girl.jpg" width="500" height="600">

Here **width** and **height** are known as attributes

#### Tags

List of tags: https://www.geeksforgeeks.org/html-tags-a-to-z-list/

### CSS





#### Ways to write CSS

Inline

<**p** style="color:#009900; font-size:50px;

             font-style:italic; text-align:center;">

        GeeksForGeeks

    </**p**>

Internal or embedded

 <**style**>

        .main {

            text-align: center;

        }

        .GFG {

            color: #009900;

            font-size: 50px;

            font-weight: bold;

        }

        .geeks {

            font-style: bold;

            font-size: 20px;

        }

    </**style**>

External

<**head**>

    <**link** rel="stylesheet" href="geeks.css" />

</**head**>

#### Selector

Dot operator is said to be class seletor. Use the while representing style for class



Below screenshot represents tag



Asterick represents **global selector**. This style will be applied for all elements.

#### Units

##### Px

Pixels are a unit of measurement commonly used in web design to define the size of various elements on a web page. The "px" unit represents a fixed-size, square area on a screen.

##### Fr

Fr is a fractional unit. Its an input that automatically calculates layout divisions when adjusting for gaps inside the grid.

grid-template-columns: 1fr 1fr 1fr 1fr;



grid-template-columns: 1fr 1fr 2fr 2fr;



##### Rem

Rem stands for root em - Relative to the font-size of the root element of the document

Em - Relative to the font-size of its nearest parent or the element itself

html {

font-size: 16px; /\* Set the base font size for the entire document \*/

}

body {

font-size: 1rem; /\* 1rem is equal to 16px in this example \*/

}

h1 {

font-size: 2rem; /\* 2rem is equal to 32px (2 times the base font size) \*/

}

p {

font-size: 1.2rem; /\* 1.2rem is equal to 19.2px (1.2 times the base font size) \*/

}

.container {

width: 50rem; /\* Width is set to 800px if the base font size is 16px (50 times the base font size) \*/

}

#### Pseudo class

Keywords that specify a special state of the selected elements. They allow you to style elements based on their state or position in the document. Pseudo-classes are denoted by a colon (:) followed by the pseudo-class name.

***li:nth-child(odd) {***

***background-color: #f0f0f0;***

***}***

***a:hover {***

***color: red;***

***}***

#### Property

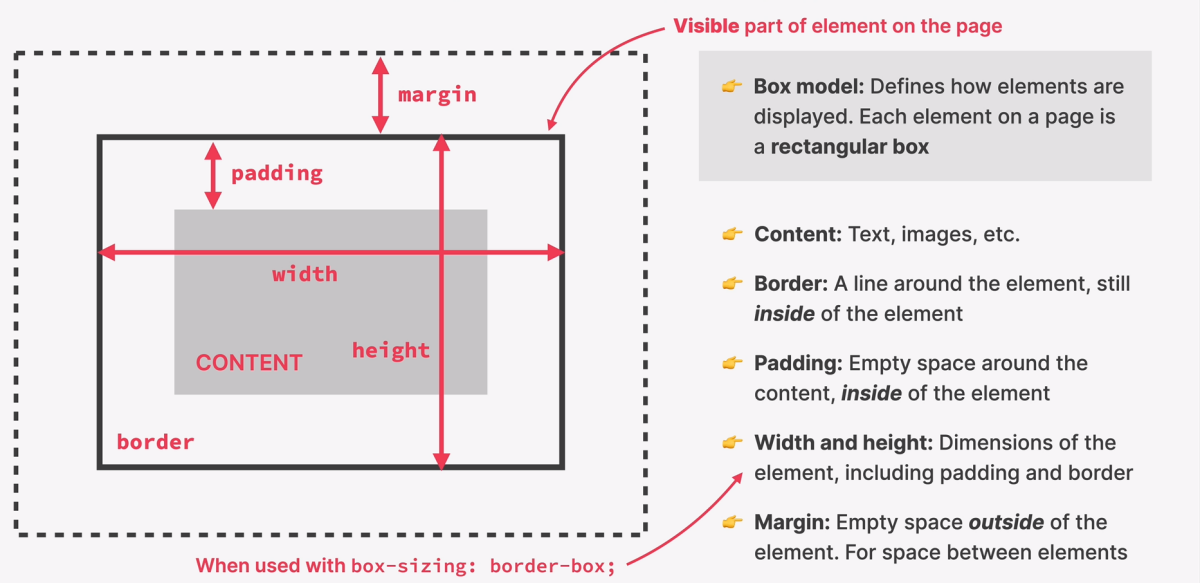
##### Transition property

***element {***

***transition: property duration timing-function delay;***

***}***

#### Box model



##### Property with multiple values

Padding: 16px 24px **(2)**

It can take values,

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 4 |
| Top and bottom padding.  Left and right padding. | Top padding.  Left and right padding.  Bottom padding. | Top padding.  Right padding.  Bottom padding.  Left padding. |

Similarly for border, margin, etc

#### Block and inline elements

##### Block level

Starts on a new line and stretch the full width of their containing element. They create a "block" on the web page. Examples of block-level elements include <div>, <p>, <hx> , <ul>, <ol>, <li>, <table>, and others.

##### Inline level

Do not start on a new line and only take up as much width as necessary. They flow within the content and do not create a new "block." Examples of inline elements include <span>, <a>, <strong>, <em>, <img>

**Display: block** property can be used to change inline element to block level element

#### Layouts

##### Inline

For text

##### Block

For sections in a webpage

##### Table

For two-dimensional table data

##### Positioned

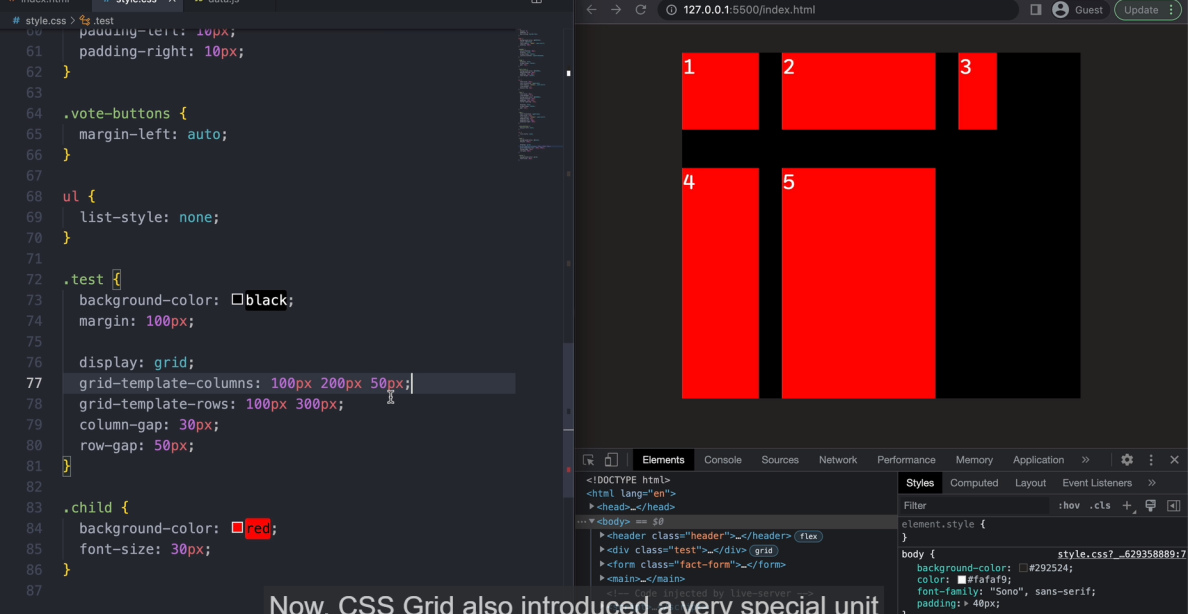
For explicit position of an element

##### Flexbox

This layout module makes it easier to design flexible responsive layout structure without using float or positioning.

##### CSS Grid

Refer test class in below screenshot for some of the example of grid related properties



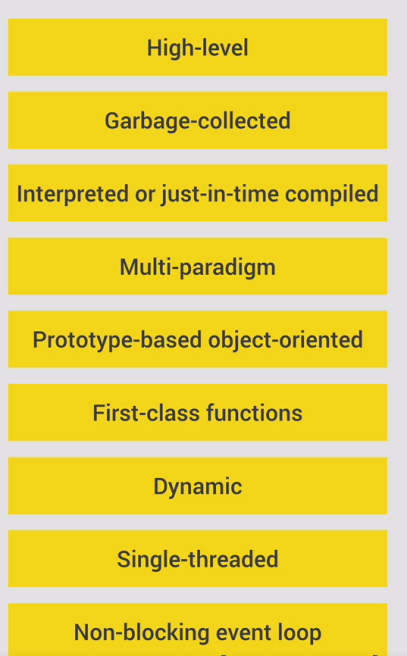
#### References

1. Use google fonts to download/get URL for required font style
2. For color palette: <https://tailwindcss.com/docs/customizing-colors>
3. Html emojis: <https://www.w3schools.com/charsets/ref_emoji_smileys.asp>
4. https://developer.mozilla.org/en-US/docs/Web/CSS

### JS

#### Definition





#### Strict mode

The "use strict"; directive is used in JavaScript to enable strict mode, which helps catch common coding errors and prevents the use of certain error-prone features.

*function myFunction() {*

*"use strict";*

*// The strict mode rules apply only to the code within this function.*

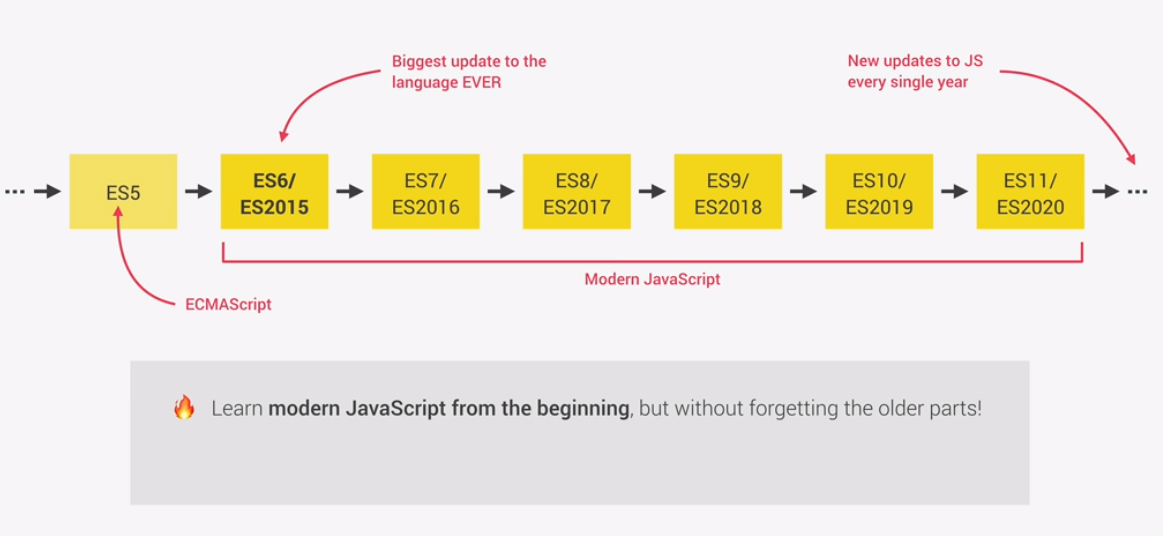
*// Your code here*

*var x = 10;*

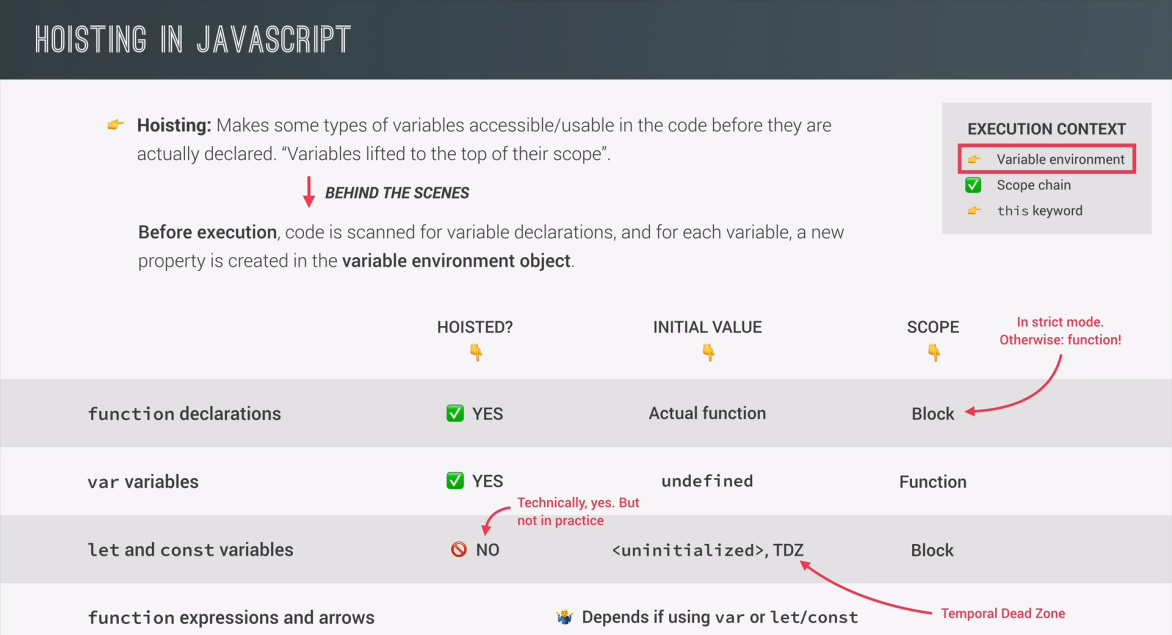
*console.log(x);*

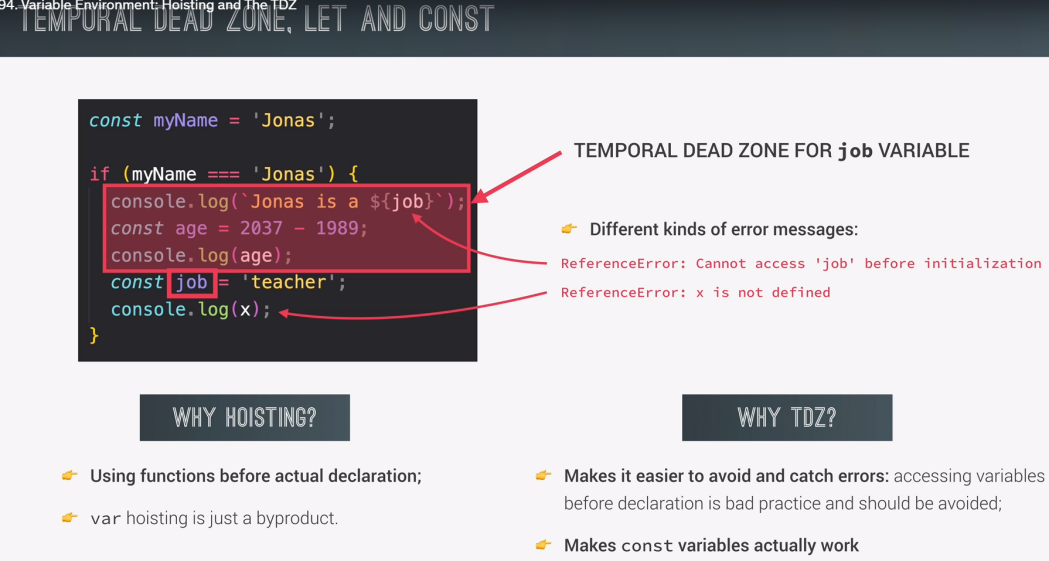
*}*

#### Versions

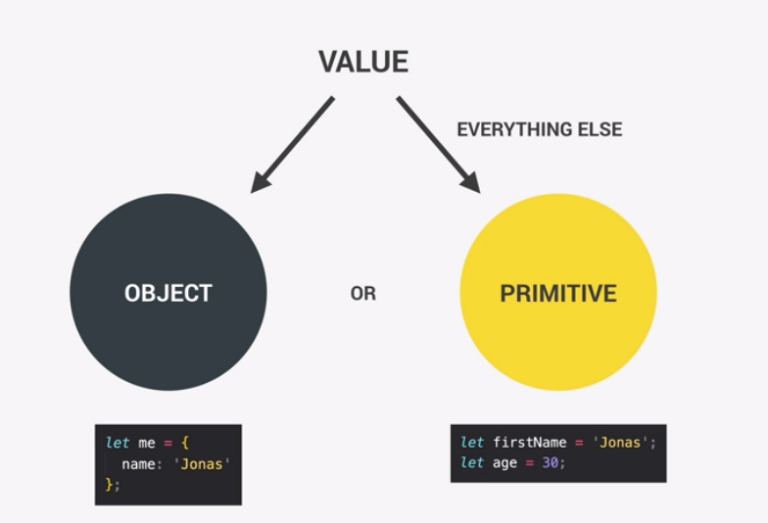


#### Hoisting





#### Types



##### Var,let, const

Age=30;//Variables without any keyword

Console.log(Age);//This will not throw any error

**Var**

Mutable

Used in older js version

Can be used before declare

Age=30;

Var Age;

**Let**

Used in modern JS

*Let age=30;*

*Age=31;//This works*

*Let withoutvalue; //This works*

**Const**

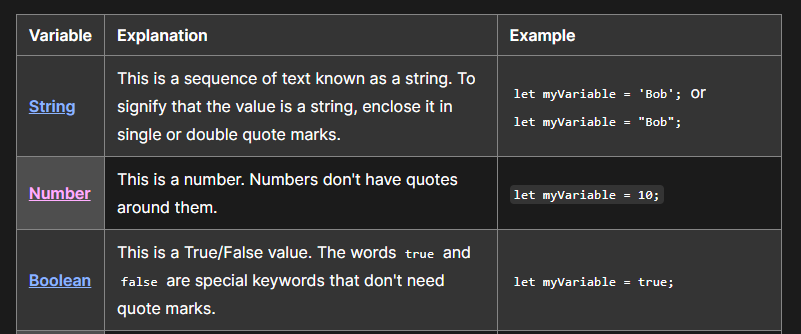
Immutable variable

*Const age=30;*

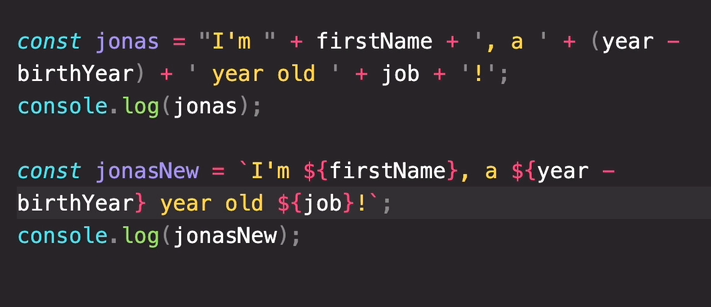
*Age=31;//This line will throw error*

*Const novalue;//This will throw error. Should have some variable.*

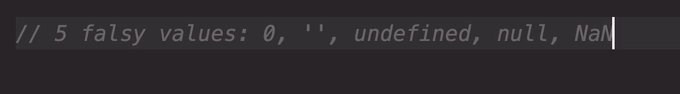
##### Primitive types



###### String



###### Bool



All other values will return true if we convert to bool

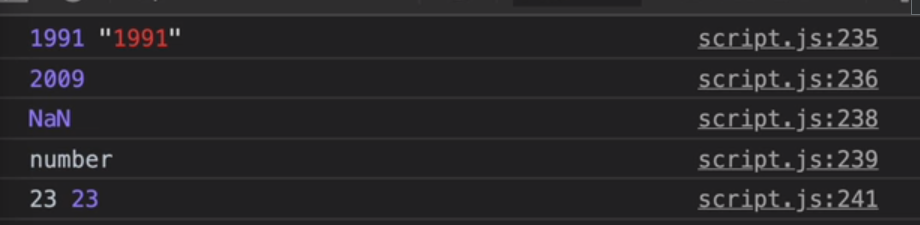
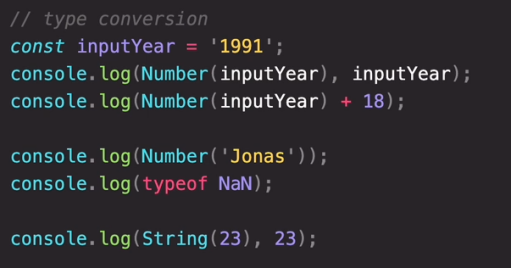
##### Non-primitive

###### Array

###### Object

##### Conversion and coercion

###### Conversion



###### Coercion

**(+) Converts number to string**

var num = "5";

var sum = num + 10; // sum is "510" (num is coerced into a string before addition)

**(-, /, \*) Converts string to number**

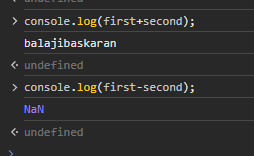
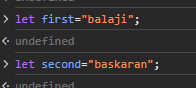
var result = "10" - 5;

console.log(result); // Output: 5 (string "10" is coerced into a number)

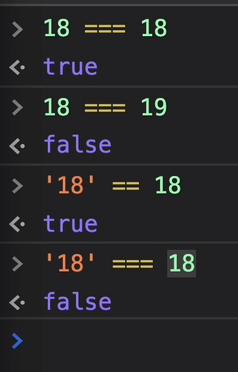
#### Operators

2\*\*3 represents 2 power 3

String addition and subtraction

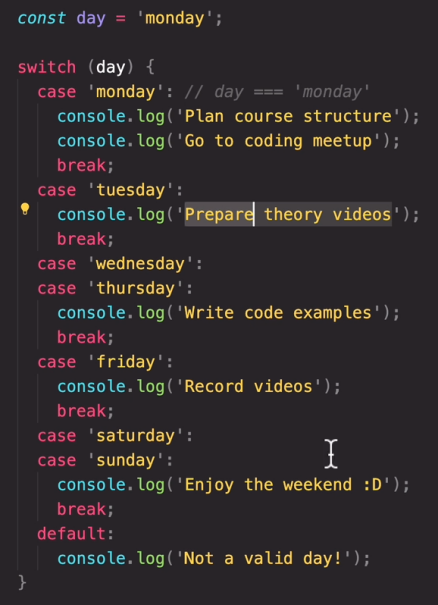


##### Equality operator



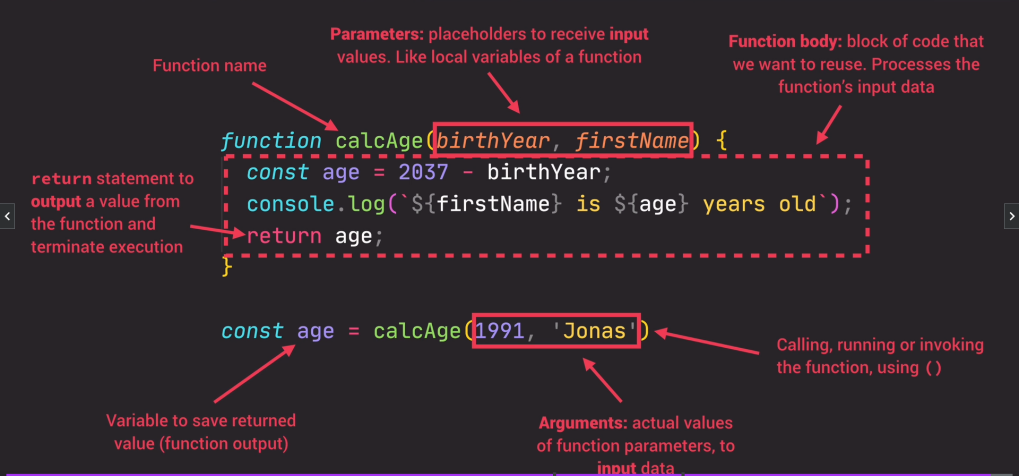
Always recommended to use ===

#### Switch

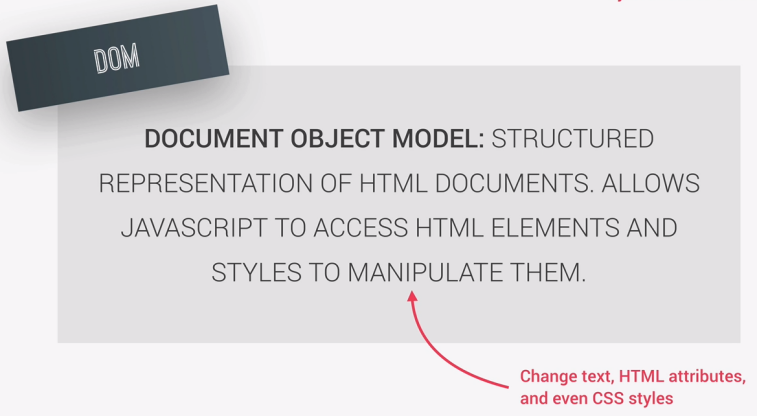


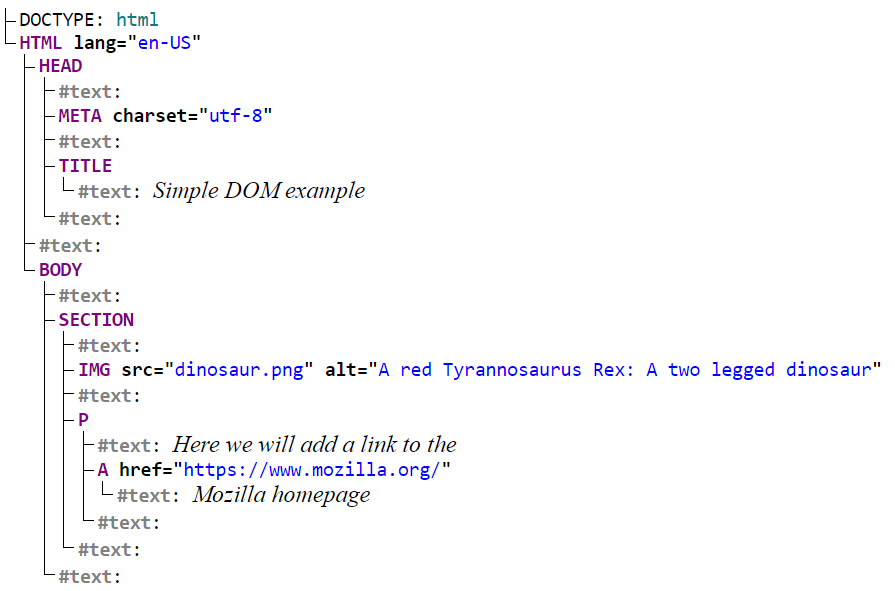
#### Function

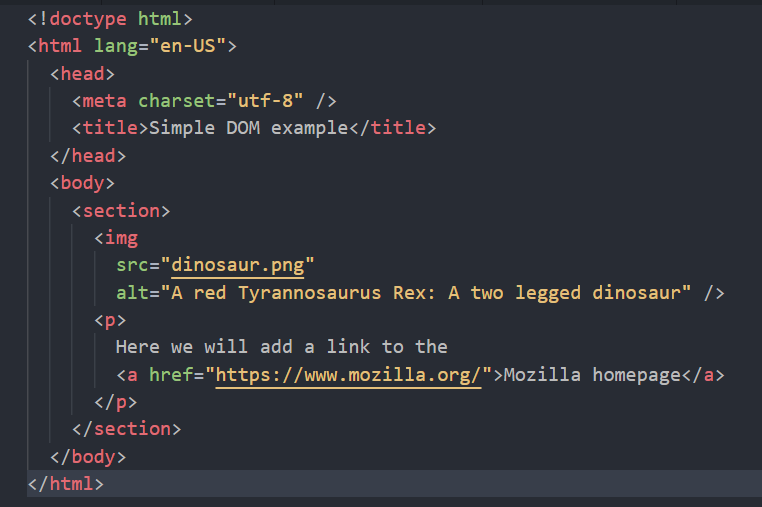
***Refer: DotnetDev\Webpages\Udemy\JsMyRef\function.js for code***



#### DOM







## WEB API

### References

Tim videos: Tim\Web API

Code: DotnetDev\WebAPI\WebAPI

### Concepts

### API Questions

### Getting started

Project creation “ASP net core web API”

#### Launchsettings.json

Helps us to launch the app with various profiles.

Profile with app name runs in kestrel(used for internal testing) server

Development or Production settings. Production will be selected by default.

Authentication settings

#### Appsettings.json

Configuration settings. Used to avoid hard coding in c#.

Example: Logging information - loglevel, etc

Separate files will be present for production and development mode mentioned in appsettings.json file

Need of this file?

Text file, so we can change the information during production for debug. Connection string based on the stage of application release we can select the appropriate server.

#### Secrets.json

Update app settings that are specific to particular machine. This will not be in source code location. So this data will not be committed to repo.

#### Program.cs

Dependency injection of service. Scoped(one instance per user of API), transient(new instance every time) or singleton(one instance for entire instance of API) service

WebApplication supports many features(middleware) like UseAuthorization, UseHttpsRedirection, UseSwagger, Run(Actually starts the API service).

#### Controllers

Define various endpoints. Various decorator like [ApiController] , Route

#### Http Verbs

**Http Verbs: Put, Post, Get, Patch and Delete**

#### FromBody attribute

From the body of the request

Builder.Services.AddEndPointsApiExplorer();

Builder.Services.AddSwaggerGen();

### Testing

Use Postman, swagger to verify the endpoints without client.

#### Swagger/Open API

Provides information about all endpoints and also helps to initiate requests to all endpoints.

Add nuget package Swashbuckle.AspNetCore

### API security

Authentication provider: Identity 4.0, Auth 0, Azure active directory, B2C.

#### Secrets

Right click on project and select “**Manage user secrets**” option to generate **secrets.json** file. Created in “C:\Users\balaj\AppData\Roaming\Microsoft\UserSecrets\7b0f6a97-a343-4b6a-924d-1ff07ec07cfd” location. Settings in secrets.json will override appsettings.json. Useful for local development. Its not required that all fields in Secrets.json should be present in appsettings.json. But its recommended to add so that we can understand the structure that is present in secrets.json.

#### Tokens

Token will be provided by server to user at the end of successful authentication. Lasts for few minutes to few hours.

### Authentication and authorization

Authentication - Valid user name and password.

#### OAuth 2.0 for accessing google drive API

To implement authentication in a C# application, you can use the OAuth 2.0 framework, which is a widely used protocol for secure authorization. Here's a step-by-step guide to implementing OAuth 2.0 authentication in a C# application:

1. Register your application with the authentication provider (e.g., Google, Facebook, Microsoft, etc.) to obtain the client ID and client secret. This step will vary depending on the provider you choose.

2. Install the appropriate **NuGet package** for OAuth 2.0 in your C# project. For example, if you are using OAuth with Google, you can use the **`Google.Apis.Auth`** NuGet package.

3. Add the necessary **using statements** at the top of your C# file:

```csharp

using Google.Apis.Auth.OAuth2;

using Google.Apis.Auth.OAuth2.Flows;

using Google.Apis.Auth.OAuth2.Responses;

using Google.Apis.Services;

using Google.Apis.Drive.v3; // Replace this with the appropriate API you want to access.

```

4. **Implement the OAuth 2.0** authentication flow **in your code**:

```csharp

public class OAuth2Authentication

{

// Set these values with the client ID and client secret obtained during registration

private const string ClientId = "YOUR\_CLIENT\_ID";

private const string ClientSecret = "YOUR\_CLIENT\_SECRET";

public static UserCredential GetCredentials()

{

// Define the scopes of the APIs you want to access

string[] scopes = { DriveService.Scope.DriveReadonly }; // Replace this with the appropriate scope for your use case.

// Generate the OAuth 2.0 flow

var flow = new GoogleAuthorizationCodeFlow(new GoogleAuthorizationCodeFlow.Initializer

{

ClientSecrets = new ClientSecrets

{

ClientId = ClientId,

ClientSecret = ClientSecret

},

Scopes = scopes

});

// Start the OAuth 2.0 authorization process

UserCredential credential = GoogleWebAuthorizationBroker.AuthorizeAsync(

flow,

new[] { DriveService.Scope.DriveReadonly }, // Replace this with the appropriate scope for your use case.

"user", // User identifier

CancellationToken.None).Result;

return credential;

}

}

```

5. Use the obtained credentials to make API calls to the service:

```csharp

public class Program

{

public static void Main()

{

UserCredential credential = OAuth2Authentication.GetCredentials();

// Use the credential to make API calls

// For example, if using Google Drive API:

var service = new DriveService(new BaseClientService.Initializer()

{

HttpClientInitializer = credential,

ApplicationName = "YourAppName"

});

// Now you can use the 'service' variable to interact with the Drive API or other APIs based on the scopes you requested.

// For example, to list files from Google Drive:

var files = service.Files.List().Execute();

foreach (var file in files.Files)

{

Console.WriteLine(file.Name);

}

}

}

```

Note: This example demonstrates how to use OAuth 2.0 with the Google Drive API. If you're implementing OAuth 2.0 for a different provider or API, you may need to modify the scopes and API-specific code accordingly.

Make sure to handle exceptions and errors appropriately and store the credentials securely if required for future use.

Keep in mind that OAuth 2.0 is a complex topic, and the implementation details can vary based on the specific provider and API you are using. Always refer to the official documentation for the specific API you want to access to ensure the correct implementation.

#### OAuth 2.0 for user defined APIs

Implementing OAuth 2.0 for your own API involves setting up an Authorization Server to issue access tokens and protecting your API resources using those access tokens. Below is a simplified example of how you can implement OAuth 2.0 for your own API using the `IdentityServer4` library, which is a popular choice for building an Authorization Server in C#.

**Step 1: Set up the Authorization Server**

First, create a new ASP.NET Core Web Application and select the "Empty" template. Then, install the `IdentityServer4` NuGet package.

```bash

Install-Package IdentityServer4

```

**Step 2: Configure the Authorization Server**

In the `Startup.cs` file, configure the `IdentityServer` services and add a test user for demonstration purposes:

```csharp

using IdentityServer4.Models;

using IdentityServer4.Test;

public class Startup

{

public void **ConfigureServices**(IServiceCollection services)

{

// Configure IdentityServer

services.AddIdentityServer()

.AddInMemoryClients(GetClients())

.AddInMemoryApiResources(GetApiResources())

.AddTestUsers(GetUsers())

.AddDeveloperSigningCredential();

// Other service configurations...

}

private IEnumerable<Client> GetClients()

{

return new List<Client>

{

new Client

{

ClientId = "your-client-id",

ClientSecrets = { new Secret("your-client-secret".Sha256()) },

AllowedGrantTypes = GrantTypes.ClientCredentials,

AllowedScopes = { "your-api-scope" }

}

};

}

private IEnumerable<ApiResource> GetApiResources()

{

return new List<ApiResource>

{

new ApiResource("your-api-scope", "Your API Name")

};

}

private List<TestUser> GetUsers()

{

return new List<TestUser>

{

new TestUser

{

SubjectId = "1",

Username = "testuser",

Password = "testpassword"

}

};

}

// Other methods...

}

```

**Step 3: Protect your API with OAuth 2.0**

In your API project, you need to protect the resources using OAuth 2.0. This can be done using the **`Authorize` attribute**.

```csharp

[ApiController]

[Route("api/[controller]")]

public class SampleController : ControllerBase

{

[HttpGet]

**[Authorize]**

public IActionResult Get()

{

// Your API logic here...

return Ok("Hello from your protected API!");

}

}

```

**Step 4: Requesting an Access Token**

To access the protected API, clients need to request an access token from the Authorization Server. In this example, we'll use the **`HttpClient`** to simulate a client application making the request:

```csharp

using System.Net.Http;

using System.Net.Http.Headers;

public class Program

{

private static async Task Main()

{

var token = await GetAccessTokenAsync();

if (token != null)

{

await CallProtectedApiAsync(token);

}

}

private static async Task<string> GetAccessTokenAsync()

{

var client = new HttpClient();

var disco = await client.GetDiscoveryDocumentAsync("https://your-authorization-server-url");

if (disco.IsError)

{

Console.WriteLine(disco.Error);

return null;

}

var tokenResponse = await client.RequestClientCredentialsTokenAsync(new ClientCredentialsTokenRequest

{

Address = disco.TokenEndpoint,

ClientId = "your-client-id",

ClientSecret = "your-client-secret",

Scope = "your-api-scope"

});

if (tokenResponse.IsError)

{

Console.WriteLine(tokenResponse.Error);

return null;

}

return tokenResponse.AccessToken;

}

private static async Task CallProtectedApiAsync(string token)

{

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

var response = await client.GetAsync("https://your-protected-api-url");

if (response.IsSuccessStatusCode)

{

var content = await response.Content.ReadAsStringAsync();

Console.WriteLine("API Response: " + content);

}

else

{

Console.WriteLine("Failed to call API. Status code: " + response.StatusCode);

}

}

}

```

Replace the placeholders (e.g., "your-authorization-server-url", "your-client-id", "your-client-secret", "your-api-scope", "https://your-protected-api-url") with the appropriate values.

Remember that this is a simplified example for demonstration purposes. In a real-world scenario, you need to handle various aspects like token expiration, token storage, and token validation more securely and efficiently.

### Claims

### Advanced Authorization

### Versoning

### Monitoring

### Protecting

### Building

### Consuming

### Open API

### Minimal API

### Best practices

### Misc and Questions

#### CORS??????

#### Redis Output caching

Reference youtube video: <https://www.youtube.com/watch?v=_bg5dGnudPs>

#### Sample Task

## MVC

### Razor Pages

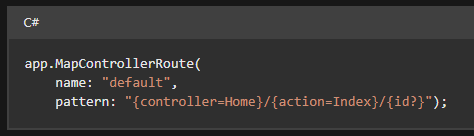
### Model

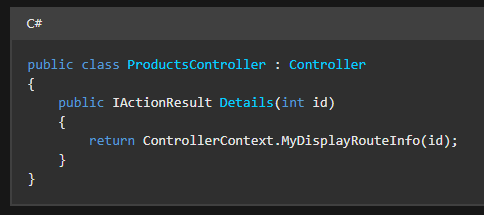
### View

### Controller

### Routing

#### Convention routing





Url: /Products/Details/5

{ controller = Products, action = Details, id = 5 }

#### Attribute routing