

(DDA) 2.2
**DEVELOPING
DYNAMIC
APPLICATIONS**

2021

COVERAGE

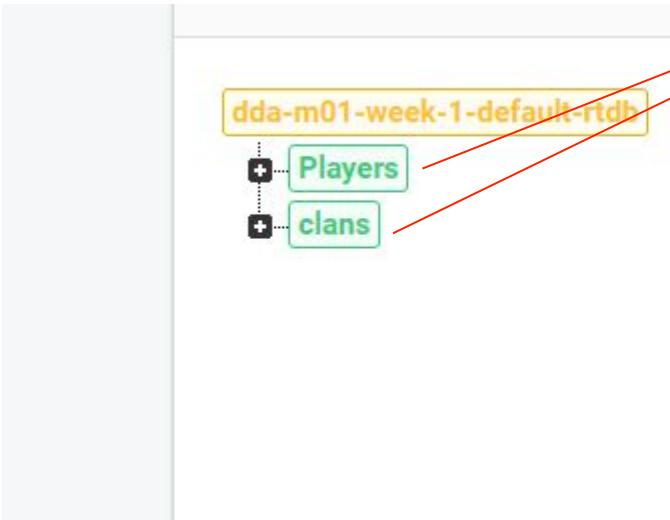
Week 6

Learning Objectives

1. Common Mistakes & Errors in Assg 1
2. Web Development Recap
3. Firebase Create & Read using JavaScript
4. Firebase Auth using JavaScript
5. Indexing in Firebase for Efficiency
6. Real-time updates via Web & Unity

Common #ERRORS

Common Errors/Misconceptions



Inconsistencies between nodes naming convention

Keep to camelCase 🐫

A better way

players

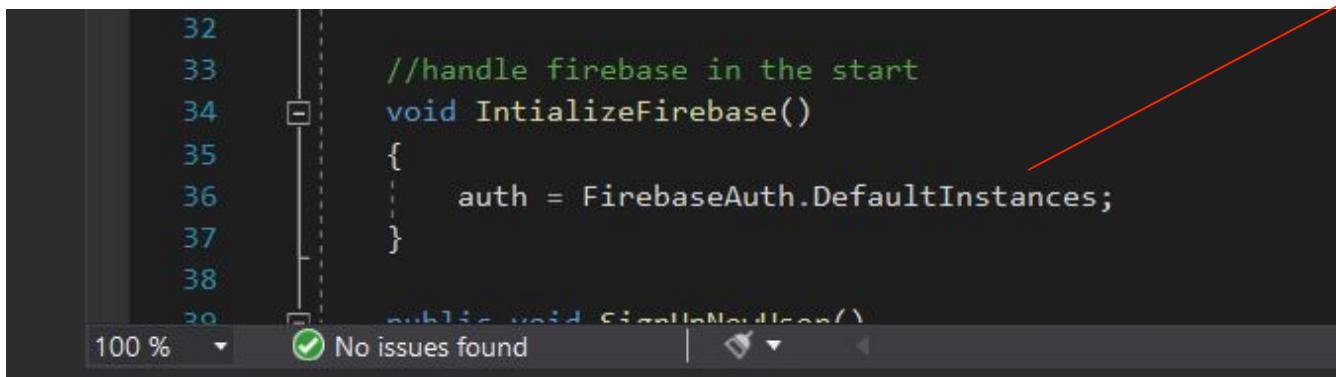
clans

playerStats

playerLogs



Common Errors/Misconceptions



A screenshot of the Visual Studio code editor. The code is as follows:

```
32
33     //handle firebase in the start
34     void IntializeFirebase()
35     {
36         auth = FirebaseAuth.DefaultInstances;
37     }
38
39     public void SignUpWithEmailAndPassword()
40     {
41         ...
42     }

```

The word "DefaultInstances" is underlined with a red squiggly line, indicating a spelling error. The status bar at the bottom shows "No issues found".

Error: Intellisense not working + naming is wrong

If your Visual Studio Intellisense is working, it will flag out this error

Fix:
FirebaseAuth.DefaultInstance
There's no plural form of it

Fix your Intellisense

<https://blog.terresquall.com/2020/11/fixing-visual-studios-intellisense-autocomplete-in-unity/>



Common Errors/Misconceptions

Ignoring C# Hints

Intellisense in Visual Studio helps you to be a better coder. Check out what's wrong

```
public void initializeFirebase()
{
    dbPlayerStatsReference = FirebaseDatabase.DefaultInstance.GetReference("player Stats");
    dbLeaderBoardReference = FirebaseDatabase.DefaultInstance.GetReference("leaderboards");
}
```

Error: Not being consistent

Having nodes with spaces might cause potential issues when you are referencing data. You might just forget “there’s a space”

Inconsistencies between nodes naming convention

Keep to camelCase 
A better way
players



Common Errors/Misconceptions

Not Reading The Squiggly Lines

Assuming Class Name is Filename!!!

This doesn't match at all!!

```
1 reference
public async Task CreateNewSimplePlayer(string uuid, string displayName, string userName, string email, string clan)
{
    SimpleGamePlayer newPlayer = new SimpleGamePlayer(displayName, userName, email, clan);
    Debug.LogFormat("Player Details: {0}", newPlayer.Print());
    await dbReference.Child("Players/" + uuid).SetRawJson(newPlayer.ToGamePlayerToJson());
    await AddPlayerToClan(uuid, clan);
    UpdatePlayerDisplayName(displayName);
}
```

This scenario, the class name is wrong. Hence C# can't find the class

The constructor name is derived from the **class name**. Constructors allows us to do this
SimplePlayerStats sp = new SimplePlayerStats();

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class SimplePlayerStats
7  {
8      public string userName;
9      public string displayName;
10     public string email;
11     public string clan;
12     public bool active;
13     public long lastLoggedIn;
14     public long createdOn;
15     public long updatedOn;
16
17     public SimplePlayerStats()
18     {
19     }
20
21
22     public SimplePlayerStats(string userName, string displayName, string em
23     {
24         this.userName = userName;
25         this.displayName = displayName;
26         this.email = email;
27         this.active = active;
28         this.clan = clan;
29
30         //timestamp properties
31         var timestamp = new DateTimeOffset(DateTime.UtcNow).ToUnixTimeSecond
32         this.lastLoggedIn = timestamp;
33         this.createdOn = timestamp;
34         this.updatedOn = timestamp;
35     }
}
```

Easy way is to ensure you keep your file name and class names the same

Common mistake, is duplicate file name and just rename the file, not realising the class name needs to be changed too and constructor



Common Errors/Misconceptions

Not using appropriate Firebase Database rules

The screenshot shows the Firebase Rules Editor interface. At the top, there are two tabs: "Edit rules" (which is selected) and "Monitor rules". Below the tabs, the code editor contains the following JSON:

```
1. {  
2.   "rules": {  
3.     ".read": false,  
4.     ".write": false  
5.   }  
6. }
```

A large red "X" is overlaid on the code editor area, indicating that this configuration is incorrect.

No one can read or write data into Firebase
You will keep getting no data written or read
even with very straight forward commands in
C#/JS

The screenshot shows the Firebase Rules Editor interface. At the top, there are two tabs: "Edit rules" (selected) and "Monitor rules". Below the tabs, the code editor contains the following JSON:

```
1. {  
2.   "rules": {  
3.     ".read": true, // 2021-11-19  
4.     ".write": true, // 2021-11-19  
5.   } //  
6. }
```

A green checkmark icon is positioned in the top right corner of the code editor area, indicating that this configuration is correct.

Allow public read, write access. This is for development purposes. Moving on, we will restrict the access to authenticated users



Common Errors/Misconceptions

The screenshot shows the Firebase Realtime Database interface. At the top, there's a navigation bar with 'DDA M02' and tabs for 'Data', 'Rules' (which is highlighted with a red box), 'Backups', and 'Usage'. Below the tabs, there's a security shield icon and a link to 'Configure App Check'. The main area shows a URL 'https://dda-m02-default-rtbd.firebaseioapp.com/' and a database reference 'ddm02-default-rtbd: null'. To the right, a large red box highlights the database rules code:

```
{  
  "rules": {  
    ".read": true,  
    ".write": true  
  }  
}
```

Common Error: Everything is ok but database not updated!

Setting up public access

The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to.

★ Note: By default, read and write access to your database is restricted so only authenticated users can read or write data. To get started without setting up [Authentication](#), you can [configure your rules for public access](#). This does make your database open to anyone, even people not using your app, so be sure to restrict your database again when you set up authentication.

Fix:
Change the rules in your Database

This is because it was not set to test mode earlier and permissions were locked

Common Errors/Misconceptions

- [12:37:23] Could not locate google-services.json or GoogleService-Info.plist files.
UnityEngine.Debug:LogWarning (object)
- [12:38:53] Could not locate google-services.json or GoogleService-Info.plist files.
UnityEngine.Debug:LogWarning (object)
- [12:42:41] Could not locate google-services.json or GoogleService-Info.plist files.
UnityEngine.Debug:LogWarning (object)
- [12:44:30] Android dependency resolution failed, your application will probably not run.
!
- [12:44:30] Resolution Failed.
!

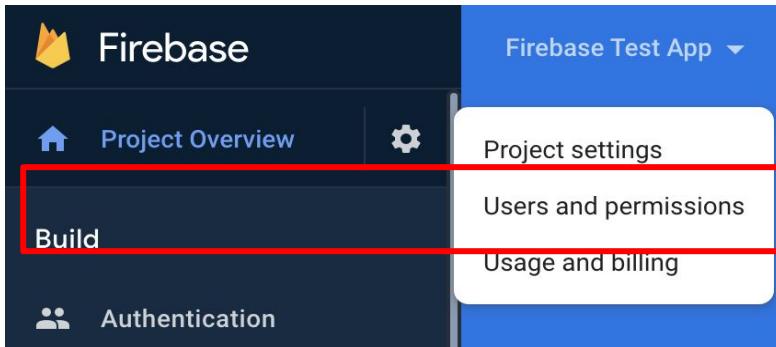
Common Error: Missing JSON

FIX:

Download and place google-services.json into your Unity Assets folder

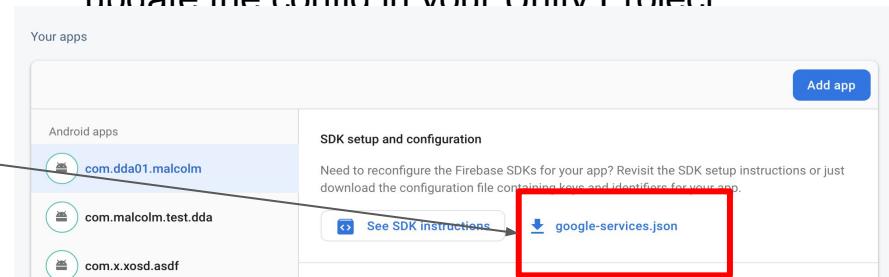
Ensure that you downloaded and place in the .json file. This is an important config file.

Go to your Firebase Console



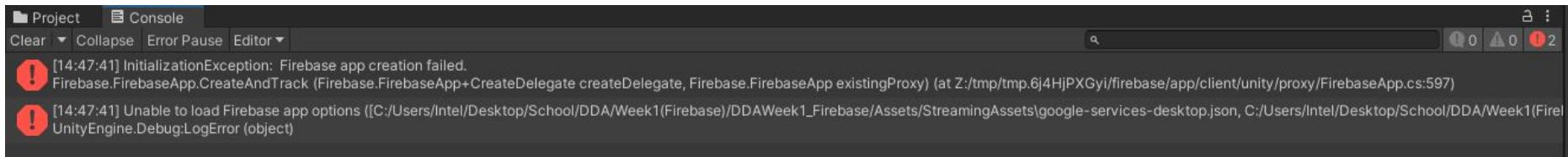
Note*

Once you enable any Firebase service, the config file gets updated. You can redownload and update the config in your Unity Project



Common Errors/Misconceptions

Common Error: JSON filename wrong



FIX:

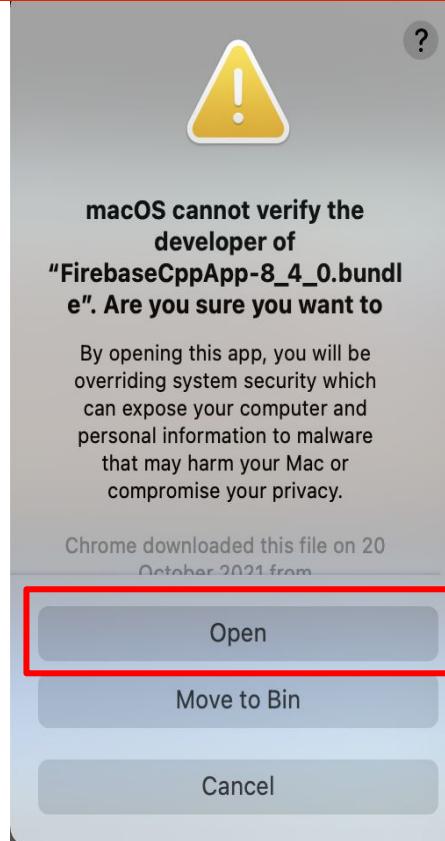
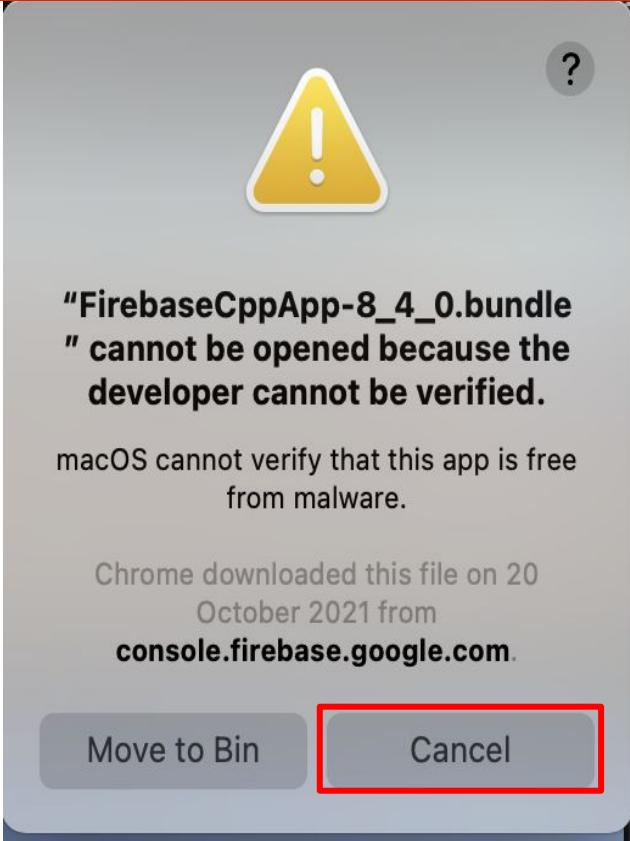
Double check, triple **check your google-services.json file .**

Ensure that it is **NOT** google-services (1).json.
The filename has to be exact.

Firebase will go and seek the file name and create a google-services-desktop.json. Hence the file has to be exact

To ensure you didn't go wrong, delete the one in your project, delete any downloads of google-sevices.json. Re-download the google-services.json file from Firebase

Common Errors/Misconceptions



Common Error: Apple Users MacOS Security Issue

1. Cancel the bundle issue

Go to System Preferences > Security > General Tab > Allow FirebaseCpp...

Re-import Firebase SDK or Re-Run project

2. Click on Open

Common Errors/Misconceptions

```
6  public class SimpleLeaderBoard : MonoBehaviour {  
7  
8      public string userName;  
9      public int highScore;  
10     public long updatedOn;  
11  
12     //simple constructor  
13     public SimpleLeaderBoard() { }  
14  
15     //constructor with parameters  
16     public SimpleLeaderBoard(string userName, int highScore)  
17     {  
18         this.userName = userName;  
19         this.highScore = highScore;  
20         this.updatedOn = GetTimeUnix();  
21     }  
22  
23     public long GetTimeUnix()  
24     {  
25         return new DateTimeOffset(DateTime.UtcNow).ToUnixTimeSeconds();  
26     }  
27  
28     public string SimpleLeaderBoardToJson()  
29     {  
30         return JsonUtility.ToJson(this);  
31     }  
32 }  
33
```

Common Error: Including MonoBehaviour in your Class

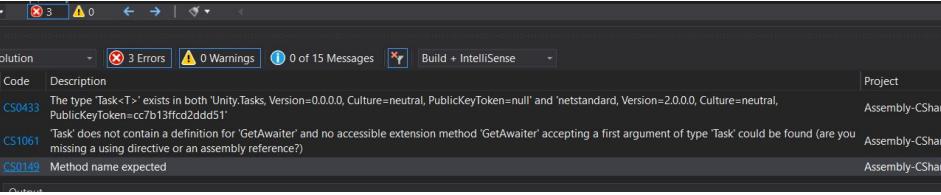
MonoBehaviour provides you with all the Unity capabilities, it inherits all the superpowers of Unity (functions, variables,etc) .

The easiest being able to attach a MonoBehaviour script as a Component to your Game Object

By itself, this issue is fine.
BUT when we are converting between JSON, it's an issue and won't be converted properly

Fix: Take it out!

Fixing Task Issues in Your Unity



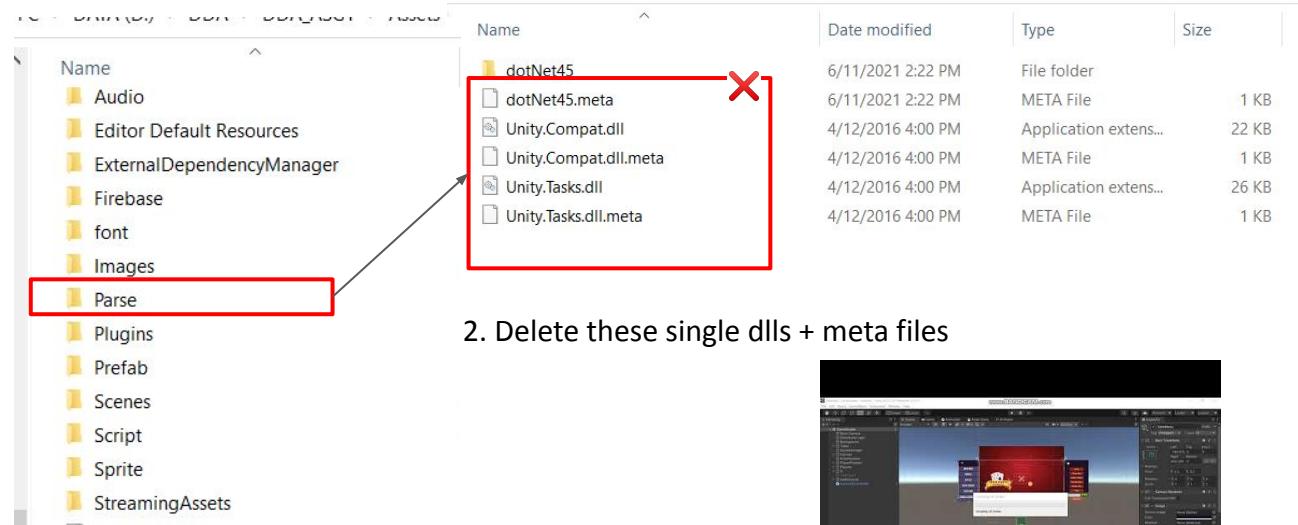
Solution | 3 Errors | 0 Warnings | 0 of 15 Messages | Build + IntelliSense

Code Description Project Assembly-CSharp
CS0433 The type 'Task<T>' exists in both 'Unity.Tasks, Version=0.0.0, Culture=neutral, PublicKeyToken=null' and 'netstandard, Version=2.0.0, Culture=neutral, PublicKeyToken=cc/b13ffcd2dd5d1'.
CS1061 'Task' does not contain a definition for 'GetAwaiter' and no accessible extension method 'GetAwaiter' accepting a first argument of type 'Task' could be found (are you missing a using directive or an assembly reference?)
CS0149 Method name expected

```
await CreateNewSimplePlayer(newPlayer.UserId, username, username, newPlayer.Email, clan);
await UpdatePlayerDisplayName(username); // Update user's display name in authenticate service
SceneManager.LoadScene(1);

reference
public async Task<FirebaseUser> SignupNewUserOnly(string email, string password)
{
    Debug.Log("Sign Up method...");
    FirebaseUser newUser = null;
    await auth.CreateUserWithEmailAndPasswordAsync(email, password).ContinueWithOnMainThread(task =>
    {
        if(task.isSuccessful)
        {
            Debug.Log("Task successful");
            ShowPotentialFixes();
        }
        else if (task.IsCanceled)
        {
            newUser = task.Result;
        }
        Debug.LogFormat("New Player Details {0} {1}", newUser.UserId, newUser.Email);
    });
    return newUser;
}
```

1. Map to your Unity Project Folder



2. Delete these single dlls + meta files



Saving JSON Export

Your file will be a nice json export

```
{  
  "clans": {  
    "blue": {  
      "17WL498wuVRdGblZxqksITQsCnt1": true,  
      "IKzk5Rc4b5fKCoh11yYM7ZXpXLd2": true,  
      "OTLyLUJCSOha0l3hL01nCl9nJDr2": true,  
      "kSTan0TkPlbtEZECfRT0FhwUL02": true,  
      "vv7V5b3S0qs4r6tSiuZyISG44Xi1": true  
    }  
  },  
  "leaderboards": {  
    "17WL498wuVRdGblZxqksITQsCnt1": {  
      "highScore": 2,  
      "updatedOn": 1637639026,  
      "userName": "tester1"  
    }  
  },  
  "playerStats": {  
    "17WL498wuVRdGblZxqksITQsCnt1": {  
      "createdOn": 1637639026,  
      "highScore": 2,  
      "totalTimeSpent": 30,  
      "updatedOn": 1637639026,  
      "userName": "tester1",  
      "xp": 5  
    }  
  },  
  "players": {  
    "17WL498wuVRdGblZxqksITQsCnt1": {  
      "active": true,  
      "clan": "blue",  
      "createdOn": 1637639012,  
      "displayName": "tester1",  
      "email": "test999@test.com",  
      "lastLoggedIn": 1637639012,  
      "name": "tester1",  
      "xp": 5  
    }  
  }  
}
```

1.

Export JSON

Import JSON

Show legend

Disable database

You must delete all of your data before disabling

Create new database

Available with an upgrade to the Blaze plan

Web Dev

#RECAP

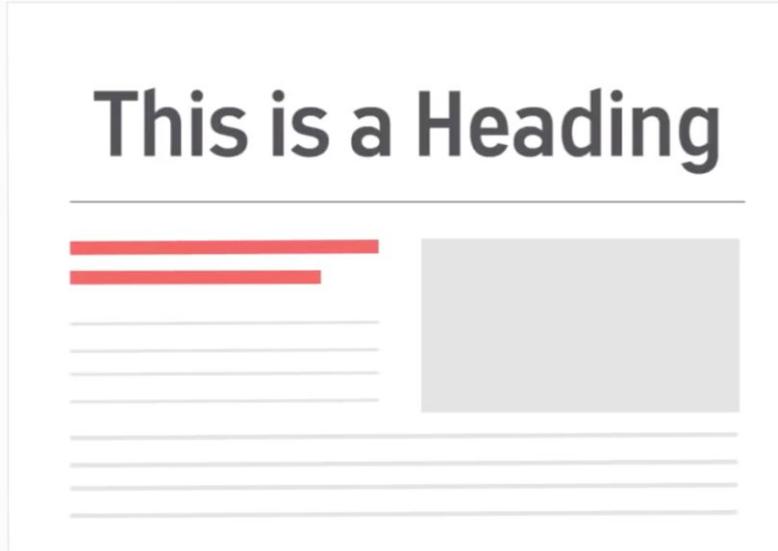


Element Description

What does an element look like

Headline in a
newspaper

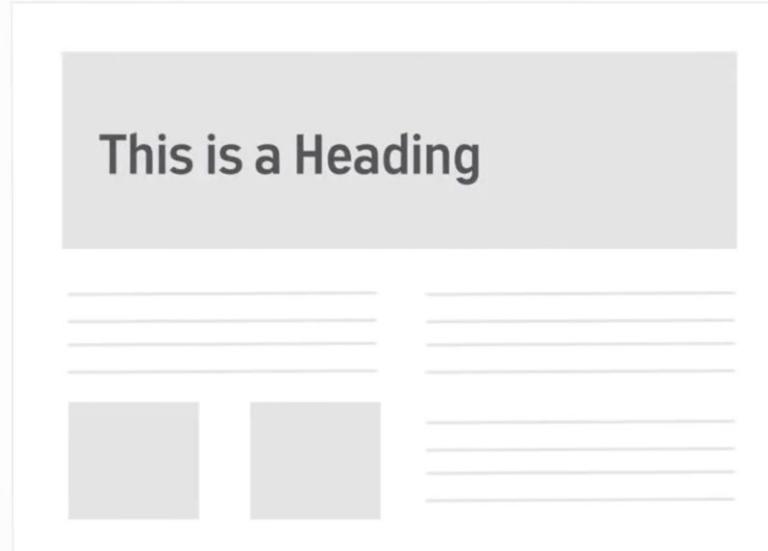
This is a Heading



`<h1>This is a Heading</h1>`

`<h1>` Headings are the center piece of a screen
space
Heading Title in a blog post

This is a Heading



Element Description

What does an element look like

<h1>

Header Element

Action starts here

Browser reads the content from left to right

Closing Tag

<h1>This is a Heading</h1>

Display this is big bold
text

Note:

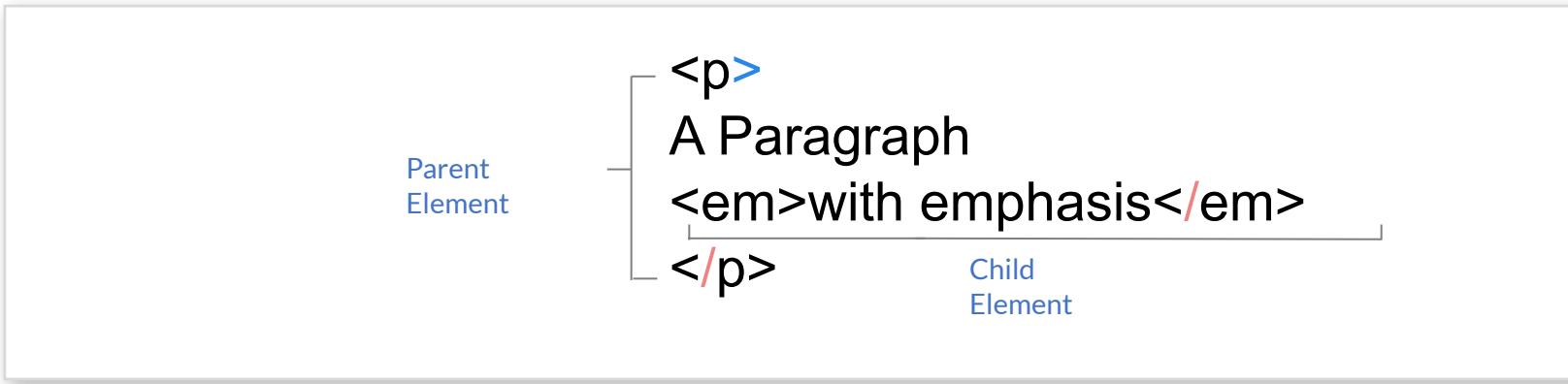
The forward "/" slash when we are closing the tag

Strive to only have one <h1> heading in a HTML element



Elements Within Elements

Child parent Relationship
Nested Relationship



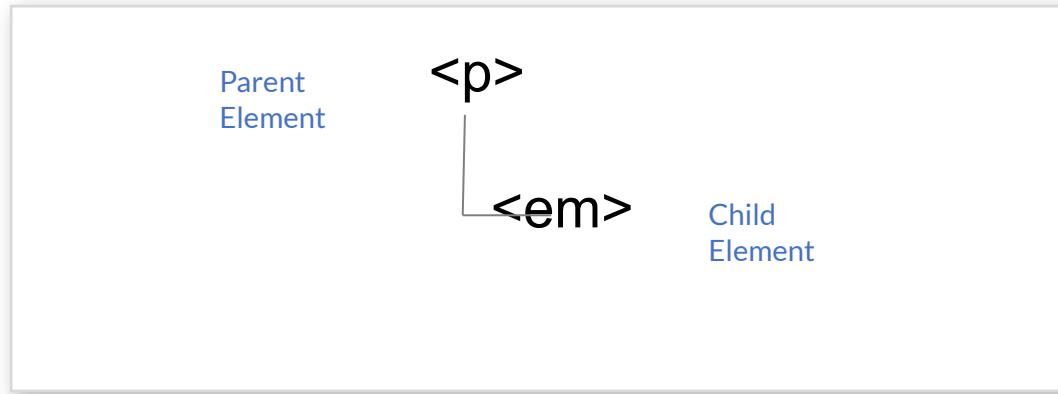
Note:

We can always nest multiple elements within
The em tag marks elements as important
Em provides emphasis and contextual meaning



Elements Within Elements

Structure



Note:

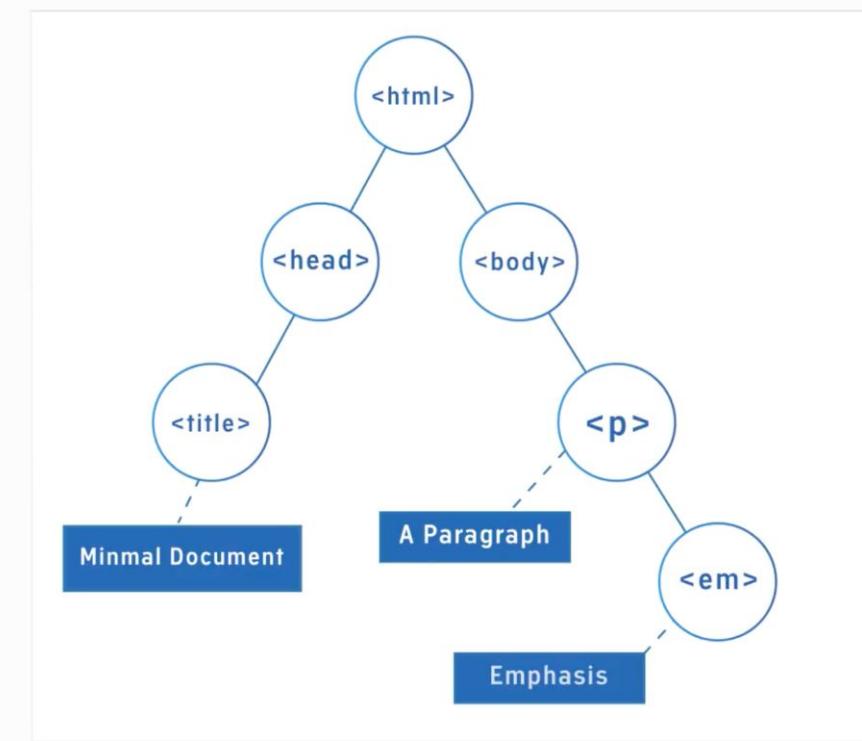
Parent: Any element that contains another element

Child: Contained elements



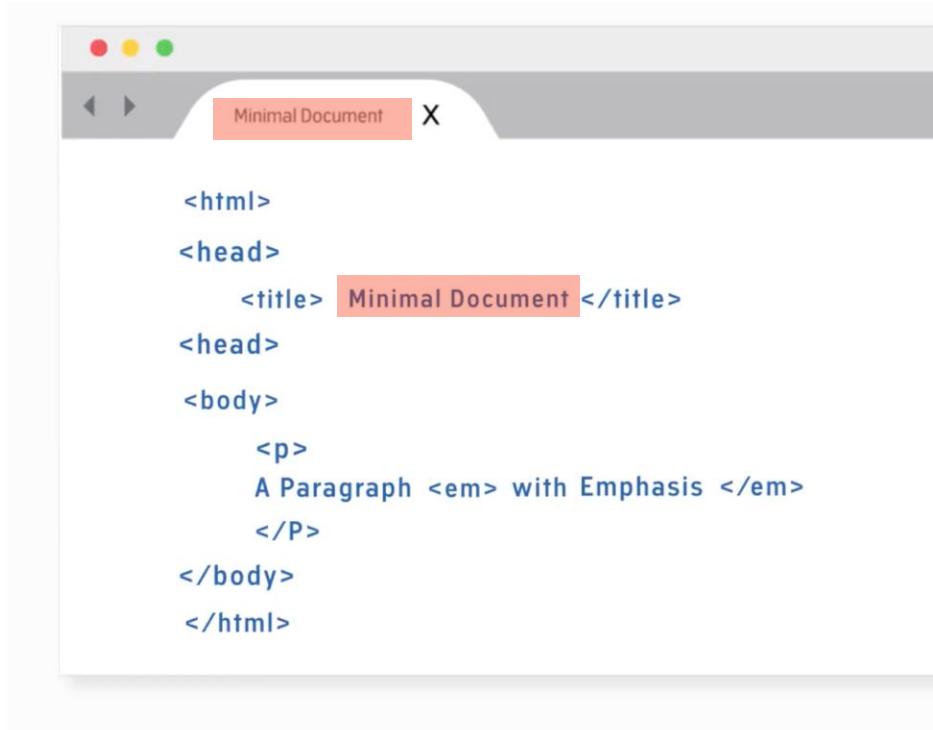
Elements Within Elements

Structure



Elements Within Elements

Structure



A screenshot of a web browser window titled "Minimal Document". The window displays the following HTML code:

```
<html>
  <head>
    <title> Minimal Document </title>
  <head>
  <body>
    <p>
      A Paragraph <em> with Emphasis </em>
    </P>
  </body>
</html>
```

The title "Minimal Document" is highlighted with a red box. The entire code block is displayed below the browser window.



Other Key Elements

Required Elements

<html>

<body>

<head>



Paragraph Element

Heading

Before you begin to decide to start coding, you must first understand what it is used for. This is **just** a paragraph with several elements intact

- First Item
- Second Item
- Third Item

<p>

<p>
Paragraph Element

Creates space above and below elements
Block level element
Paragraph Element
used for blocks of text when there's two or more sentences



Headings <h...>

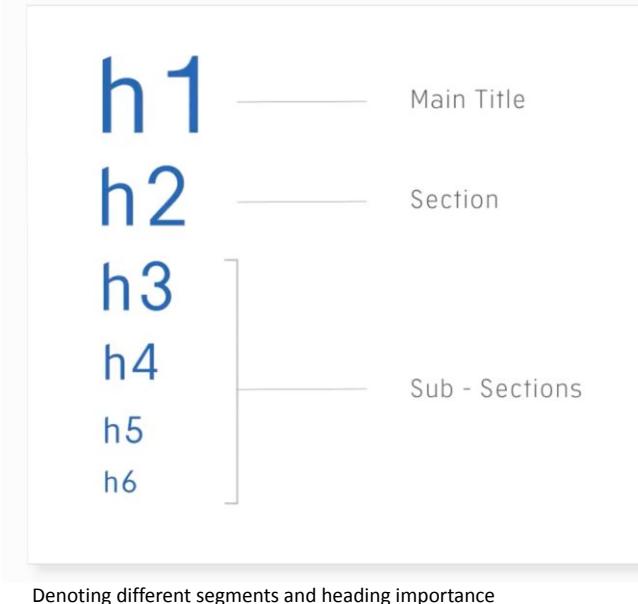


<h1>Most important HEADING</h1>
<h2>Second Most Important Heading</h2>
<h3> Third Most Important Heading</h3>
<h4> Fourth Most Important Heading</h4>
<h5> Fifth Most Important Heading</h5>
<h6>Least Important Heading</h6>

Provides visual context and meaning

*There's no such thing as <h7>...</h7> and beyond...

Maybe you can try :)



Heading 2 ▾ Arial

Normal text ▾

Title ▾

Subtitle ▾

Heading 1 ▾

✓ Heading 2 ▾

Heading 3 ▾

Heading 4 ▾

Options ▾

Google Docs



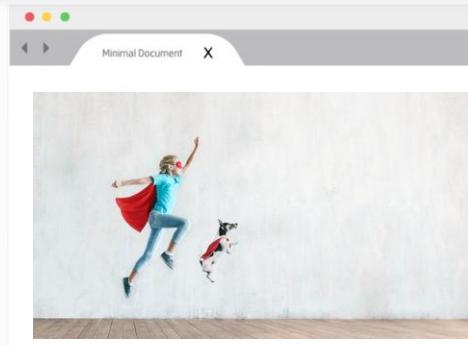
Image Tag

Used to display an image
Self-closing tag

<https://picsum.photos/>

```

```



<https://repl.it/@malcolmyam/wk01-image-example>



Image Tag

Absolute URL

`https://interactivedev.com/logo.png`

Absolute URLs are the type of links you're used to.

They include `http://` or `https://` and the full domain name before the directory (`/assets/logo.png`)

Relative URL

`/logo.png`

Relative URLs are shortcuts that allow you to skip the domain name.

It only works if you're linking to a file on the same domain on the current page.

<https://repl.it/@malcolmyam/wk01-image-example>



Lists

Unordered

- Item 1
- Item 2
- Item 3
- Item 4

```
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
<li>Item 4</li>
</ul>
```

Ordered

1. Item 1
2. Item 2
3. Item 3
4. Item 4

```
parent element
  |
<ol>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
<li>Item 4</li>
</ol>
```

child elements



Common HTML5 Elements

<p>

Defines a
paragraph
Block element

<h1><h6>

Title
<h1>
Section
block

emphasis/stress
text
italics
inline

Gives text
importance
Bold
Section headings
inline

*Note: Block elements are elements that create a spacing/line break below themselves



Anatomy of a HTML Page

<Tags>

Tags can also have attributes.

- A name optionally followed by a value.
- An attribute is used to:
 - Select between different options of element function
 - Provide extra information about what the element describes

```
<p class="special">  
    A special paragraph  
</p>
```

Example of a <p> tag with a “class” attribute



Common HTML Elements

[Link](#)

<a>

- Start tags can also have attributes
- a stands for anchor
- anchor/hyperlink element
- Allows linking to:
 - Internal locations on a document
 - Other documents and resources
 - text in <a> tags displayed in the browser as a link

text

`Interactive Development`

*Note: Every link has a destination
In order to make any link work, you'll need to specify a destination URL in the <a> element by adding the **href** attribute

[Interactive Development](#)

JumpStart to Interactive Development



Common HTML Elements

Image

- Display an image
- src attribute
 - Internal locations on a document
 - Other documents and resources
- alt attribute
 - Image description
 - Visually impaired users (accessibility)
 - In case of image not loading

```

```



Common HTML Elements

List

Unordered

```
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
<li>Item 4</li>
</ul>
```

- Item 1
- Item 2
- Item 3
- Item 4

Ordered

1. Item 1
2. Item 2
3. Item 3
4. Item 4

```
<ol>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
<li>Item 4</li>
</ol>
```

<https://repl.it/@malcolmyam/wk01-unordered-ordered-list-example>

ul -Unordered list. For listing things that do not need to be in any specific order. Navigation menus are often created with unordered lists, and are styled with CSS to display horizontally.

ol -Ordered list. A list that is automatically numbered.



Common HTML Elements

div + span

div + span
SEMANTIC-NEUTRAL

<div>

A container unit

Encapsulates other page elements

Divides the HTML documents into sections

An inline unit

Usually used for small chunks of HTML

Inside a line

e.g inside a paragraph

*Semantic-neutral elements do not provide meaning to the page but provides structure



Common HTML Elements

div +

```
<style>
div{
    border: 1px solid red;
}

span{
    border: 1px solid green;
}
</style>
```

This is a block level element
This is an inline element

Welcome to Interactive Development

<https://repl.it/@malcolmyam/wk01-divspan-example>

```
<div>This is a block level element </div>
<span>This is an inline element </span>

<!-- the H1 is a block element --&gt;
&lt;h1&gt;Welcome to &lt;span&gt;Interactive
Development&lt;/span&gt;&lt;/h1&gt;</pre>
```



Common HTML Elements

comment

In programming, we often use comments to explained our code

Comments are not displayed to the browser

```
<!-- this is an example of a comment -->
```

#CODINGTIME #CA #ACTIVITY

ACTIVITY



#CODINGTIME #CA #ACTIVITY

RECIPE/PROFILE CARD

In this activity, you will start to make your own recipe/player profile. You will be introduced to five common HTML tags; headings, paragraphs, image, ordered lists and unordered lists.

1. Based on the wireframe on the right, try to create your own profile card
2. Start by forking the base template
<http://bit.ly/id-MakeARecipe>

Submit a html titled "Wk06-studentid-studentname-make-a-card.html"
E.g *DDAWk01-16012010D-UncleRoger-make-a-card.html*

Submit the file in **Google Classroom**

You may download your repl.it file once you are satisfied :)



15 minutes

FORK this base template

FINISHED?

- + Publish your recipe and share it with your teacher/class!
- + Help a neighbour!
- + Challenge yourself! Research how to add a link to a website or insert a YouTube video.

TABLES

(\cup \circ \square \diamond) \curvearrowleft $\frac{1}{\perp}$



Does using a table make it **more clear** what the content means?

Model	B76	C75	D92	E88
Price	\$	\$\$	\$\$\$	\$\$\$\$
Rating	★★☆☆☆	★★★☆☆	★★★★☆	★★★★★
Resolution	4608 x 3072	5472 x 3648	6000 x 4000	6720 x 4480

Standings								
#	Team	MP	W	D	L	PD	Pts	Form
1.	Invictus Gaming	16	14	0	2	28:11	14	W W W W L
2.	JD Gaming	16	12	0	4	26:10	12	W W W W W
3.	FunPlus Phoenix	16	12	0	4	27:13	12	L W W W L
4.	Top Esports	16	11	0	5	23:14	11	W W W L L
5.	eStar	16	11	0	5	23:14	11	L W W L L
6.	EDward Gaming	16	9	0	7	23:18	9	L L W L W
7.	Royal Never Give Up	16	8	0	8	20:18	8	L L W L L
8.	Team WE	16	8	0	8	20:21	8	W W L L L
9.	Vici Gaming	16	7	0	9	21:22	7	W L W L L
10.	Bilibili Gaming	16	7	0	9	20:22	7	W W L W W



Common HTML Elements

TABLES

```

<table>
  <tr>
    <th>Name</th>
    <th>Class</th>
  </tr>
  <tr>
    <td>Uncle Roger</td>
    <td>T01</td>
  </tr>
  <tr>
    <td>Naomi Neo</td>
    <td>T02</td>
  </tr>
  <tr>
    <td>Xia Xue</td>
    <td>T03</td>
  </tr>
</table>

```

Element	Name	Purpose	Attributes
<table></table>	Table	Wraps the whole table	
<tr></tr>	TR – table row	Wraps around a set of elements, defining them as belonging to the same row	colspan, rowspan, headers
<th></th>	TH – table header	Defines a header for a column	colspan, rowspan, scope
<td></td>	TD – table data	Marks the actual bits of data	



Common Elements

<header>

<nav>

<article>

<section>

<table>

<details>

<figure>

<form>

<header>

<aside>

<mark>

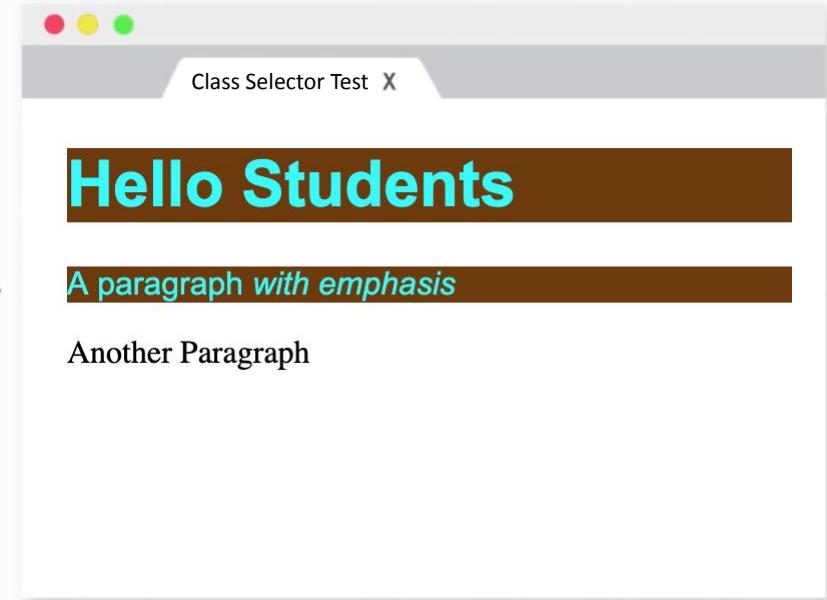
<main>



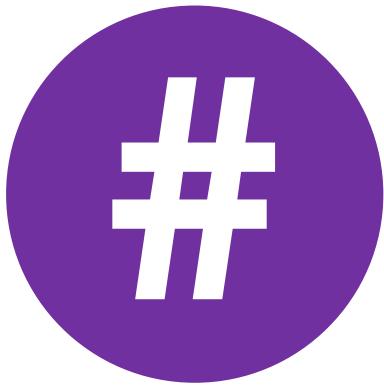
Class Attribute

```
<!DOCTYPE html>
<html>
<head>
<title>Class Selector Test</title>
<style>
.myClass{font-family:Arial, Helvetica, sans-serif;
color: #23FBFF;background-color: #6E3B04;}
</style>
</head>
<body>
<h1 class="myClass">Hello Students</h1>
<p class="myClass">
A paragraph <em> with emphasis</em>
</p>

<p>Another Paragraph</p>
</body>
</html>
```

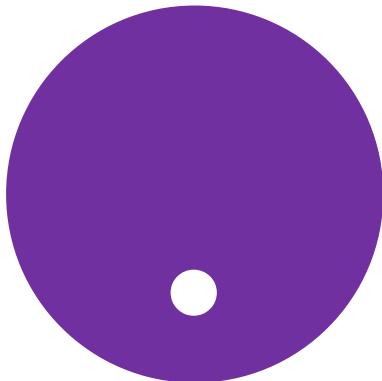


ID Attribute - TARGETING AN INDIVIDUAL



- ◆ #header { } Selects any element with the **id** "header",
e.g. <p id="header"></p>
- ◆ Unique to that element
- ◆ Hash symbol
- ◆ The "#" is how you tell CSS "this is an id."
 - ◆ Selector name matches element ID value,
prefixed with # symbol

Class Attribute - TARGETING A GROUP



- ◆ .warning { color: red; } Selects any element with the **class name** “warning”, e.g.<p class="warning"></p>
- ◆ Used to style an element grouping
- ◆ Targets one or more element's class attribute
- ◆ Elements can be of different type
- ◆ The “.” is how you tell CSS "this is a class name."
- ◆ An element can have only 1 id but multiple classes, so you can take advantage of that for greater flexibility.
 - ◆ Eg <p class="desc intro">hi -> hi aka class chaining

Pseudo Classes

A set of "pseudo classes" can style anchor elements depending on their state.

```
a:link { /* unvisited link */ color: red; }  
a:visited { /* visited link */ color: blue; }  
a:hover { /* moused over link */ color: green; }  
a:active { /* current link */ color: purple; }  
a:focus { /* focused link */ color: purple; }
```



Best Practices

Some rules to follow when making IDs and class names:

- ◆ Describe the **content**, not the presentation ("warning", not "redbox").
- ◆ Use **all lowercase, and hyphens when needed for readability** ("header-info", not "headerInfo").
- ◆ Use hyphens to show that a class or ID is part of something else. (e.g. "footer", "footer-copyright", and "footer-logo").
- ◆ Preferably **NO camel case**





5-10 minutes

CHALLENGE: Applying Class & ID

1. Refer to the repl.it file
2. Apply the #heading ID to **one** appropriate element to affect the result to look like the one on the right
3. Apply the class styles to the appropriate elements to affect the result

Submit a html titled:

"DDAWK06-studentid-studentname-css-id.html"

E.g DDAWk02-16012010D-UncleRoger-css-id.html

Submit the file in **Mel**

You may download your repl.it file once you are satisfied

:)

<https://code.visualstudio.com/>

Some KickAss Visual Studio Code Extensions

Live-Server, Prettier, HTML Boilerplate, HTML

Snippets

FORK

<https://repl.it/@malcolmyam/wk02-css-id-class>

History

Giants & Heroes

Track & Field

Some sporting feats ridicule expectations, fewer violate logic but the rarest of all added a defying of gravity to a hat-trick of achievements. At 3.45pm on 18 October 1968 in Mexico City's Estadio Olímpico Universitario Bob Beamon accomplished all three.

Rugby

With the game locked at 10 – 10 with thirty minutes on the clock, a Munster scrum just outside the Biarritz line enabled Peter Stringer to do the unexpected and break on the blindside for an unforgettable try. A moment of sheer brilliance by the Munster and Ireland scrumhalf. The try proved to be crucial as Munster won by 23 – 19 to lift the European Cup in Cardiff.

The first Programmer

Ada Lovelace was an English mathematician and writer, chiefly known for her work on Charles Babbage's proposed mechanical general-purpose computer, the Analytical Engine. She was the first to recognise that the machine had applications beyond pure calculation, and published the first algorithm intended to be carried out by such a machine. As a result, she is sometimes regarded as the first to recognise the full potential of a "computing machine" and the first computer programmer.

The first Compiler

Grace Hopper was an American computer scientist and United States Navy rear admiral. One of the first programmers of the Harvard Mark I computer, she was a pioneer of computer programming who invented one of the first compiler related tools.



1

The Google Fonts homepage displays a grid of font families. The first row includes Roboto, Noto Sans SC, Commissioner, and Staatliches. The second row shows examples of how these fonts render the sentence "Almost before we knew it, we had left the ground." in both English and Chinese. A yellow box labeled '1' points to the top-left corner of the page.

2

The Roboto font family page shows its various styles: Thin 100, Thin 100 italic, Light 300, Light 300 italic, and Regular 400. Each style is previewed with the same sentence. A yellow box labeled '2' points to the top-left of the Roboto section.

3

The 'Selected family' review screen shows the Roboto font family. It highlights the 'Thin 100' style and provides an 'Add more styles' button. A yellow box labeled '3' is at the top right of the review area.

4

The 'Embed' section contains instructions for copying the font code into an HTML head and adding a CSS rule. It also shows a sample CSS rule for specifying the font family. A yellow box labeled '4' is at the top right of the embed section.

5

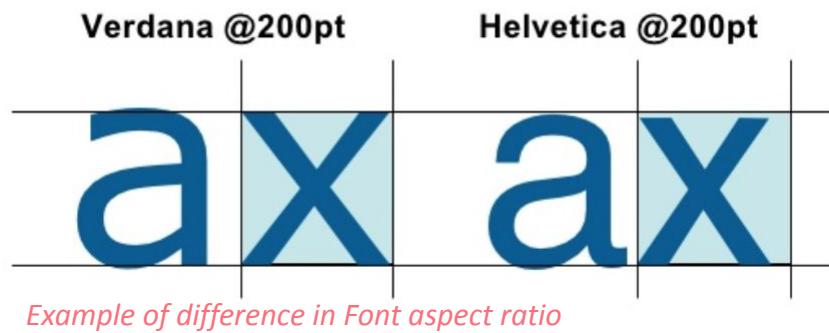
The final step involves placing a link into the HTML head and testing the result. A yellow box labeled '5' is at the top right of the instructions.

Links:

- <https://repl.it/@malcolmyam/wk02-simple-googlefonts#index.html>

FONT STACKING

- Think about fallback fonts
- Platform usage (Windows, Mac, Linux etc)
- Font aspect ratio
 - If you use fonts with different aspect ratios, Some people may see your site with smaller fonts



```
body{  
    font-family: Verdana, Arial, sans-serif  
}
```

Larger aspect ratio
Smaller aspect ratio



The <form> Tag

```
<form action="someurl.php" method="post">  
|   <!-- All our inputs will go in here -->  
</form>
```

action – the URL to send form data to

method – the type of HTTP request

*Until we work with a database noSQL/mongoDB/backend
languages, our forms are actually quite "sad" :(*

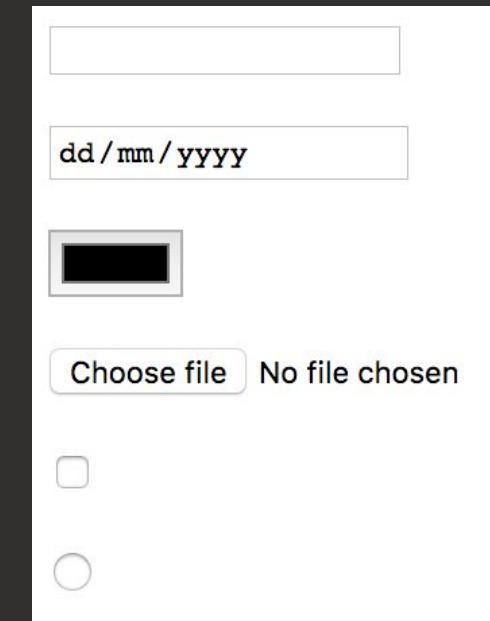
<https://developer.mozilla.org/en-US/docs/Learn/Forms>

The <input> Tag

The input tag creates interactive controls.

The "type" attribute determines the type of input

```
<input type="text">  
  
<input type="date">  
  
<input type="color">  
  
<input type="file">  
  
<input type="checkbox">  
  
<input type="radio">
```



The image shows a white rectangular box containing six examples of different input types. From top to bottom: 1. A standard text input field. 2. A date input field with the placeholder "dd / mm / yyyy". 3. A color input field, which is a small square with a black rectangle inside. 4. A file input field with a "Choose file" button and the text "No file chosen". 5. A checkbox, represented by a small square with a horizontal line through it. 6. A radio button, represented by a small circle.

A Simple Form

```
<form action="/some-url" method="POST">
  <input type="text" placeholder="hinting text">
  <input type="password" placeholder="Password Pls!">
  <button>Login</button>
  <input type="submit" value="Login Me In">
</form>
```

Let's try sending data.

Sign In

hinting text	Password Pls!	Login	Login Me In
--------------	---------------	-------	-------------

*UX: When designing forms it's a whole process.
Got to think about the platform, device, readability, audience
Also think about are the fields **REALLY NECESSARY?**

Recommending reading:

https://static.lukew.com/webforms_lukew.pdf

Labels

Let's us add captions to the form elements

Labels are really important for making site accessible (visual impairment, etc)

Username: Password:

```
<form action="/some-url" method="GET">
  <label>Username:
    <input type="text" placeholder="hinting text" name="username">
  </label>
  <label>Password:
    <input type="password" placeholder="Password Pls!" name="password">
  </label>

  <button>Login</button>
  <input type="submit" value="Login Me In">
</form>
```

```
<form action="/some-url" method="GET">
  <label for="username">Username:</label>
  <input type="text" placeholder="hinting text" name="username" id="username">
  <label for="password">Password:</label>
  <input type="password" placeholder="Password Pls!" name="password" id="password">
  <button>Login</button>
  <input type="submit" value="Login Me In">
</form>
```

Alternate syntax, using "**for**" and "**id**" attributes

Validations

- The 'required' attribute validates that an input is not empty
- There are also type validations. Try changing "type" from "text" to "email"

```
<!-- not every single browser allows form validation -->
<form action="/some-url" method="GET">
  <label for="email">Email:</label>
  <input type="email" placeholder="Email.." name="email" id="email" required>
  <label for="password">Password:</label>
  <input type="password" placeholder="Password Pls!" name="password" id="password" required>
  <button>Login</button>
  <input type="submit" value="Login Me In">
</form>
```

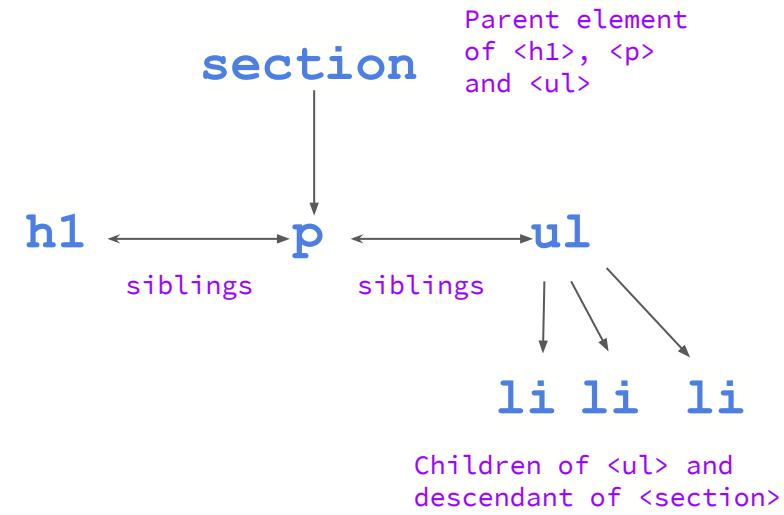
The image contains two side-by-side screenshots of a web browser window. Both screenshots show a login form with fields for Email and Password, and a Login button.

Screenshot 1 (Left): The Email field contains the character 'a'. A red border highlights the Email field, and a red exclamation mark icon in a yellow box at the bottom left of the field indicates an error. A tooltip message says: "Please include an '@' in the email address. 'a' is missing an '@'."

Screenshot 2 (Right): The Password field contains the placeholder text "Password Pls!". A red border highlights the Password field, and a red exclamation mark icon in a yellow box at the bottom left of the field indicates an error. A tooltip message says: "Please fill in this field."

Nested Elements & the DOM

```
<section>
  <h1>Heading</h1>
  <p>Paragraph</p>
  <ul>
    <li>List item</li>
    <li>List item</li>
    <li>List item</li>
  </ul>
</section>
```



Descendant Selector

```
/* Selects only links, inside of a paragraph, inside of section */
```

```
section p a { ... }
```

↓
Descendant element

```
<section>
```

```
  <p>There's a <a href="#">link</a> inside this paragraph.</p>
```

```
  <p>Paragraph</p>
```

```
  <a href="#">Link</a>
```

```
</section>
```

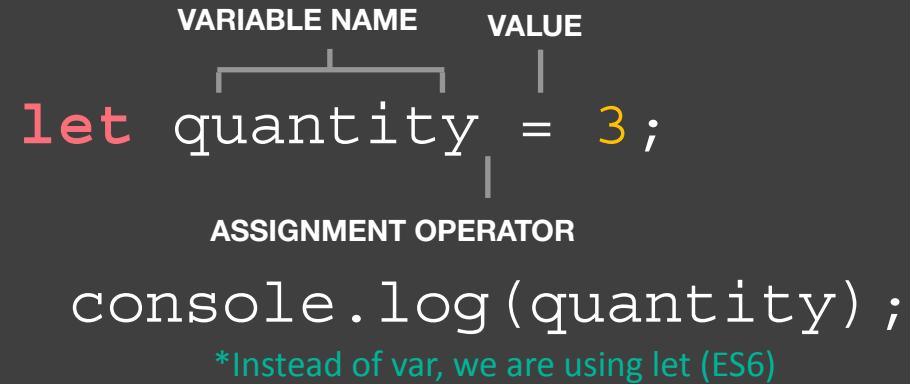
```
<a href="#">Link</a> → Not affected
```

JavaScript

#RECAP

```
Python  
quantity = 3
```

```
C#  
type variableName = value;  
int quantity = 3;
```



3

PRIMITIVE DATA TYPES

Numbers: Floating point numbers, decimals & Integers

Strings: Sequence of characters, text

Booleans: Logical Data (True or False)

2 OTHER PRIMITIVE DATA TYPES

Undefined: Data type of a variable that does not have a value yet

Null: Also means non-existent

DATA TYPES

Numbers

```
let x = 1;
```

Strings

```
let myName = "Bob";
```

Boolean

```
let myName = true;
```

Array

```
let myName = [1, 2, 3, "hi"];
```

Syntax

```
let name = <value>;
```

Using Quotes for Strings

Note that for **String** variable, the value must be enclosed using either the double quotes or the single quotes. (It's more common to use the double quotes).

Example:

```
let item = "durian cake";  
let fruit = 'durian';
```

** Notice how the string is placed inside quote marks

*if start with double quotes, end with double quotes
If start with single quotes, end with single quotes

✓ "hello" ✗ "hello'

✓ 'hello' ✗ 'hello"

✓ " " ✗ " "

✓ ' ' ✗ ' '

String Processing

Concatenation

string1 + string2

```
"some" + "text"; // returns "sometext"  
let first = "my";  
let second = "string";  
let union = first + second; // union variable has the string "mystring"
```

Length of a String

string.length

```
"My name".length // 7 , white space is also counted  
"".length // 0
```

Case Manipulation

```
"my name".toUpperCase(); // Returns "MY NAME"  
"MY NAME".toLowerCase(); // Returns "my name"
```

trim() // replace() // charAt() // substring() // indexOf()

For more: http://www.w3schools.com/jsref/jsref_obj_string.asp

Template literals are enclosed by the back-tick (` `)

backtick
|
`string text`

backtick
|
`string text line 1
string text line 2`
Multiple line template literal

|
backtick

`string text \${expression} string text`

[]

Expression to evaluate

Usual method of writing strings

```
var age = 21;  
console.log('My age is ' + age);
```

Substitution using Template String

```
var age = 21;  
console.log(`My age is ${age}`);
```



Template
String

- ✓ String Substitution
- ✓ Cleaner
- ✓ More Readable

&&

LOGICAL AND

This operator test **more than one** condition.

`((2 < 5) && (3 >= 2))`
returns **true**

If both expressions evaluate to true then the expression returns true. If just one of these returns false, then the expression returns false.

true && true returns **true**
true && false returns **false**
false && true returns **false**
false && false returns **false**



LOGICAL OR

This operator test **at least one** condition.

`((2 < 5) || (3 >= 2))`
returns **true**

If both expressions evaluate to true then the expression returns true. If just one of these returns false, then the expression returns false.

true && true returns **true**
true && false returns **false**
false && true returns **false**
false && false returns **false**

! ●

NOT

This operator inverts true or false values

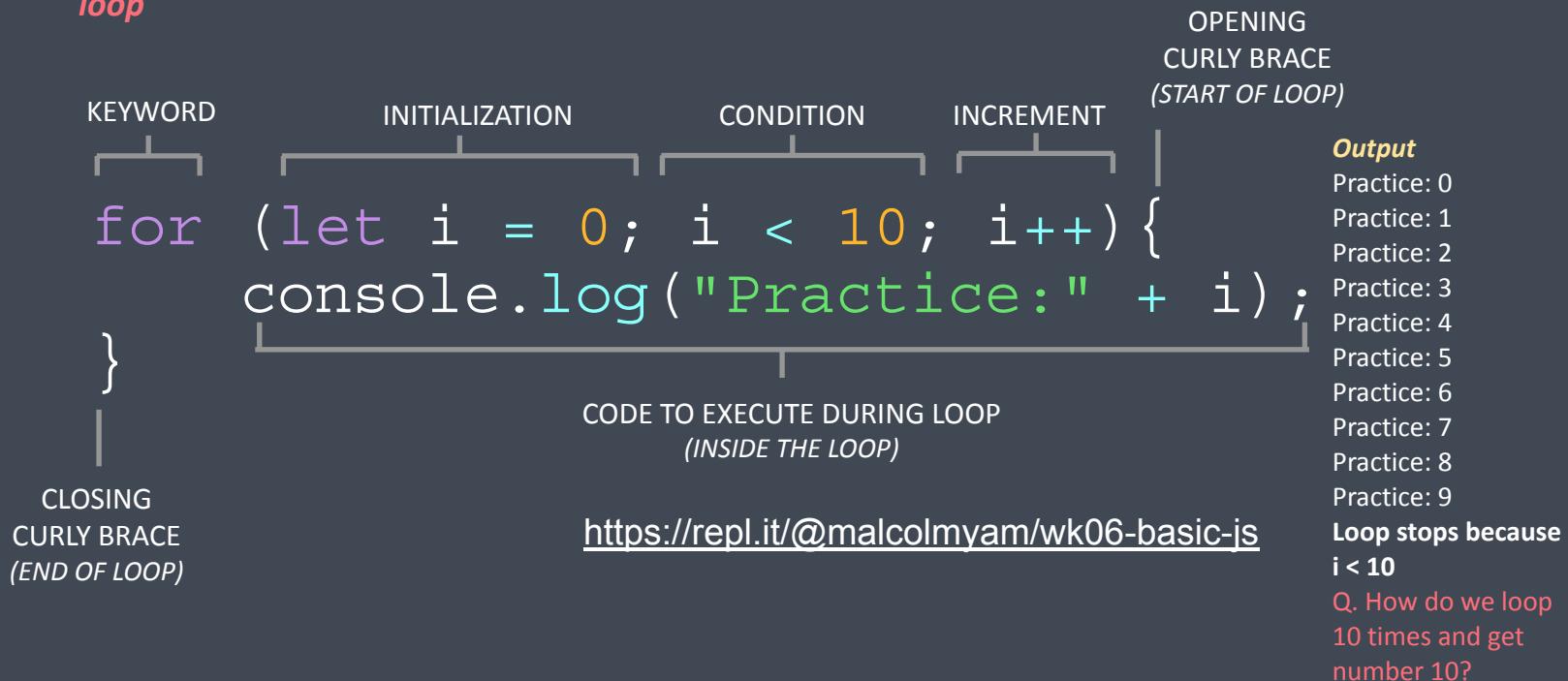
(!true)
returns **false**

Operator simply inverts the value contained within

FOR LOOPS

```
for(var i = 0; i < 10; i++){ ....}
```

*Example: Display the word "Practice: x" using **for loop***



INITIALISATION

Create a variable and set it to 0
Acts as a counter

```
let i = 0;           i < 10;
```

Variable is only created the **first time** the loop is run.

CODE SAMPLE

```
for (let i = 0; i < 10; i++) {  
    console.log("Practice:" + i);  
}
```

CONDITION

Loop continues to run until the counter reaches a specific number

INCREMENT / UPDATE

Every time the loop has run the statements in the curly braces, it adds **one to the counter**

i++

The value of i was initially set to 0, so in this case the loop will run 10 times before stopping.

One is added to the counter using the increment (++) operator.

It is also possible for loops to count downwards using the decrement operator (--)

FOR LOOP

```
for (let i = 0; i < 10; i++) {  
    console.log("Practice:" + i);  
}
```

V

WHILE LOOP

S

```
let i = 0;  
while (i < 10) {  
    console.log("Practice:" + i);  
    i++;  
}
```

Both are equivalent and produce the same results

ACTIVITY

“Loops Recap” (10min)

Q3 - EVEN LOOP

Use a for loop to print out the following set of numbers that counts down from the number 10. For each even number, print out “(even)” beside it.

Desired Output

10 (even)

...

6 (even)

..

2 (even)

0

End of output

Q4 - SIMPLE ARRAY

Use a for loop to print out an array. Fill in the blanks in the code below:

```
let a = [1001, 1002, 1003, 1004, 1005];
for (let i = 0; i < ????; ++i)
{
    console.log ( a [ ??? ] );
}
```

INSTRUCTIONS

Create a document

Wk06-JS-studentid-studentname.docx

Label each question in the docx and insert your answers inside

E.g

Q1. <answer>

Complete all 5 exercises and Submit your .docx file

- OBJECT
- KEY / NAME
- VALUE

LITERAL OBJECTS

```
let hotel = {  
  
    name: 'Raffles Hotel',  
    rooms: 100,  
    booked: 24,  
    gym: true,  
    roomTypes:  
        ['twin', 'suite', 'delux'],  
  
    checkAvailability: function() {  
        return this.rooms - this.booked;  
    }  
};
```



- Object is the curly braces {...} and its contents
- Object stored in a variable **hotel**
- These are variables Separate each key from its value using a colon
- Separate each property and method with a comma
- The **this** keyword in **checkAvailability()** method, References the **rooms** and **booked** projects of the Object (**hotel**)

<https://repl.it/@malcolmyam/wk06-objects#script.js>

ACCESSING AN OBJECT



```
let hotelName = hotel.name;
let roomsFree = hotel.checkAvailability();
```

MEMBER OPERATOR

```
let hotelName = hotel['name'];
let roomsFree = hotel['checkAvailability']();
```

<https://repl.it/@malcolmyam/wk06-objects#script.js>



UPDATING AN OBJECT

OBJECT PROPERTY NAME PROPERTY VALUE

```
hotel.name = 'Favcho Royale Hotel';
```

MEMBER OPERATOR ASSIGNMENT OPERATOR

```
hotel['name'] = 'Favcho Royale Hotel';
```

```
delete hotel.name; // Delete a property using the delete keyword  
hotel.name = ''; // Clear the value of a property by assigning a blank string
```

* **Note:** If the object does not have the property you are trying to update, it will be added to the object



FUNCTION BASED OBJECTS

PARAMETERS

```
function Hotel(name, rooms, booked){  
  
    this.name = name;  
    this.rooms = rooms;  
    this.booked = 24;  
  
    this.checkAvailability = function()  
    {  
        return this.rooms - this.booked;  
    };  
}
```

PROPERTIES

METHOD

Creating a function Hotel allows it to be used as a template for creating multiple objects

This is called a function based object.

The **this** keyword is used instead of the object name to indicate the property/method belongs to the object that **this** function creates.

Each statement creates a new property or method. Uses **semi-colon** instead of comma (literal object syntax)

<https://repl.it/@malcolmyam/wk06-objects#script.js>

- OBJECT
- KEY / NAME
- VALUE

ARRAY INDEXING

```
0           1           2  
|           |           |  
colors= ['pink', 'yellow', 'green'];
```

Defining Arrays

Declare an array
called list

```
let list=[];  
list[0] = "First";  
list[1] = "Second";
```



We can insert a **variable** into an array by *specifying which index it goes into*

DECLARING A FUNCTION

```
Python
def my_function():
    print("Hello from a function")

my_function()
```

```
KEYWORD           FUNCTION NAME  
[-----] [-----]  
function sayHello() {  
    document.write('Hello');  
}  
}
```

CODE BLOCK (IN CURLY BRACES)

```
sayHello();
```

FUNCTION NAME

<https://repl.it/@malcolmyam/wk06-basic-js>

```
PARAMETER  PARAMETER  
function getArea(width, height) {  
    return width * height;  
}
```

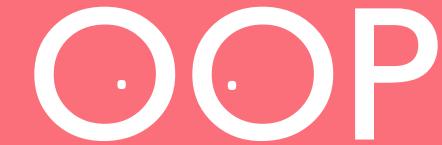
THE PARAMETERS ARE USED LIKE
VARIABLES WITHIN THE FUNCTION

CALLING A FUNCTION
THAT NEEDS
INFORMATION

```
getArea(3, 5);
```

ARGUMENTS

<https://repl.it/@malcolmyam/wk06-basic-js>



Object Oriented
Programming

- OBJECT
- KEY / NAME
- VALUE

LITERAL OBJECTS

```
let hotel = {  
  
    name: 'Raffles Hotel',  
    rooms: 100,  
    booked: 24,  
    gym: true,  
    roomTypes:  
        ['twin', 'suite', 'delux'],  
  
    checkAvailability: function() {  
        return this.rooms - this.booked;  
    }  
};
```



Object is the curly braces { ... } and its contents
Object stored in a variable **hotel**

PROPERTIES
These are variables Separate each key from its value using a colon
Separate each property and method with a comma

METHOD
This is a function
The `this` keyword in `checkAvailability()` method,
References the `rooms` and `booked` projects of the
Object (**hotel**)

<https://repl.it/@malcolmyam/wk06-objects#script.js>

ACCESSING AN OBJECT



```
let hotelName = hotel.name;
let roomsFree = hotel.checkAvailability();
```

MEMBER OPERATOR

```
let hotelName = hotel['name'];
let roomsFree = hotel['checkAvailability']();
```

<https://repl.it/@malcolmyam/wk06-objects#script.js>



UPDATING AN OBJECT

OBJECT PROPERTY NAME PROPERTY VALUE

```
hotel.name = 'Favcho Royale Hotel';
```

MEMBER OPERATOR ASSIGNMENT OPERATOR

```
hotel['name'] = 'Favcho Royale Hotel';
```

```
delete hotel.name; // Delete a property using the delete keyword  
hotel.name = ''; // Clear the value of a property by assigning a blank string
```

* **Note:** If the object does not have the property you are trying to update, it will be added to the object



CONSTRUCTOR NOTATION

```
let hotel = new Object();
```

```
hotel.name = 'Raffles Hotel';
hotel.rooms = 100;
hotel.booked = 24;
hotel.gym = true;
hotel.roomTypes =
['twin','suite','delux'];
```

```
hotel.checkAvailability: function() {
    return this.rooms - this.booked;
}
```



Create an object using the "new" keyword and the **Object()** constructor (Blank object)

Add properties, methods to the newly created blank object

- OBJECT
- KEY / NAME
- VALUE

FUNCTION BASED OBJECTS

PARAMETERS

```
function Hotel(name, rooms, booked){  
  
    this.name = name;  
    this.rooms = rooms;  
    this.booked = 24;  
  
    this.checkAvailability = function()  
{  
        return this.rooms - this.booked;  
    };  
}
```

PROPERTIES

METHOD

Creating a function Hotel allows it to be used as a template for creating multiple objects

This is called a function based object.

The **this** keyword is used instead of the object name to indicate the property/method belongs to the object that **this** function creates.

Each statement creates a new property or method. Uses **semi-colon** instead of comma (literal object syntax)

<https://repl.it/@malcolmyam/wk06-objects#script.js>

- OBJECT
- KEY / NAME
- VALUE

MULTIPLE OBJECT INSTANCES

```
let favchoHotel = new Hotel('Favcho Hotel', 100, 25);  
let lizzieHotel = new Hotel('Lizzie Inn', 48, 12);
```

OBJECT

CONSTRUCTOR FUNCTION

ASSIGNMENT OPERATOR

NEW KEYWORD

VALUES USED IN PROPERTIES OF THIS OBJECT

The first object **favchoHotel**. Name is "Favcho Hotel".
100 rooms, 25 booked

The second object **lizzieHotel**. Name is "Lizzie Inn".
48 rooms, 12 booked

* **Note:** Even when multiple objects are created using the same constructor function, the methods stay the same.

<https://repl.it/@malcolmyam/wk06-objects#script.js>



ACTIVITY

“Objects” (30min) - GROUP Work

Q1. Working with Literal Objects

Based on your assignment 1, create literal objects of your class models.

Create appropriate literal objects and for each property, indicate the appropriate data types to use.

Eg. converting SimpleLeaderBoard

```
public class SimpleLeaderBoard {  
    public string userName;  
    public int highScore;  
    public long updatedOn;  
  
    //simple constructor  
    public SimpleLeaderBoard() {}  
  
    //constructor with parameters  
    public SimpleLeaderBoard(string userName, int highScore)  
    {  
        this.userName = userName;  
        this.highScore = highScore;  
        this.updatedOn = GetTimeUnix();  
    }  
}
```

```
let simpleLeaderBoard =  
{  
    "userName": "Royden", //string/  
    "highScore": 392, //int  
    "updatedOn": 1637707595 //int  
}
```

Q2. Working with Function Based Objects

Based on your assignment 1, create function based objects of your class models, indicate the appropriate data types to use.

Create sample object variables based on your models
You may assume your own property values.

Select the best structured data in your team
Submit your JavaScript files and your C# class models
//save as DDAWk06-studentid-studentname-objects.zip
Upload to Google Classroom

Firebase & Web

#FirebaseWeb

Setting up Firebase for Web

1. Go to your Firebase Console
2. Select your project
3. Click on Project Overview -> Project Settings
4. Select Add App
5. A popup will appear -> Choose web

Firebase

M03 Test DDA ▾ Proj 1

Project Overview

Project settings

Your apps

Android apps

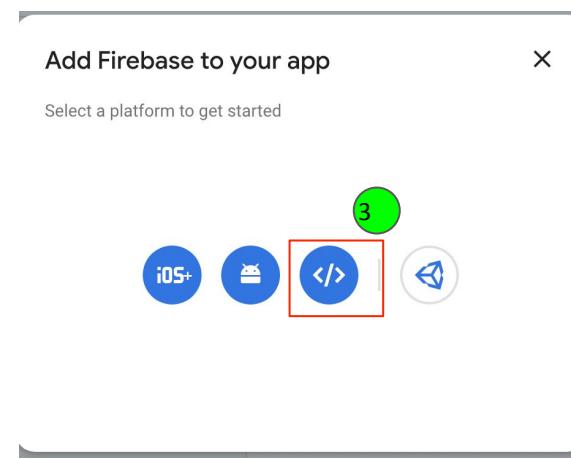
com.malcolm.dda.m03

SDK setup and configuration

Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.

See SDK instructions

google-services.json



Setting up Firebase for Web

1. Key in an nickname
2. Click on Register app
3. Select “Use a <script> tag”
4. Create a js file, copy and paste the script from Firebase

× Add Firebase to your web app

1 Register app

App nickname 

My DDA web app 

Also set up **Firebase Hosting** for this app. [Learn more !\[\]\(3627fc0ff4ec9910032cebadf7b66aa1_img.jpg\)](#)

Hosting can also be set up later. It's free to get started at any time.

Register app 



2 Add Firebase SDK

Use npm 

Use a <script> tag 

Copy and paste these scripts into the bottom of your <body> tag, but before you use any Firebase services:

```
<script type="module">
  // Import the functions you need from the SDKs you need
  import { initializeApp } from "https://www.gstatic.com/firebasejs/9.5.0.firebaseio.js"
  import { getAnalytics } from "https://www.gstatic.com/firebasejs/9.5.0/firebase-analytics.js"
  // TODO: Add SDKs for Firebase products that you want to use
  // https://firebase.google.com/docs/web/setup#available-libraries
```

```
// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
```

This contains your app settings

```
};
```



```
// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```

```
</script>
```

Connect with Firebase via JS (Read Data)

1. Create a .html file
2. Create a JS file and link it up
3. Start with the skeleton JS script that Firebase provides
4. Let's retrieve data using **get** and **ref**

Note: How we reference nodes/data depends on how you structure your data inside Firebase Database

```
const db = getDatabase();
const playerRef = ref(db, "players");

getPlayerData();
function getPlayerData(){
    //const playerRef = ref(db, "players");
    //PlayerRef is declared at the top using a constant
    //get(child(db,`players/`))
    get(playerRef)
        .then((snapshot) => {//retrieve a snapshot of the data using a callback
            if (snapshot.exists()){//if the data exist
                try {
                    //let's do something about it
                    var content = "";
                    snapshot.forEach((childSnapshot) => {//looping through each snapshot
                        //https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach
                        console.log("GetPlayerData: childkey " + childSnapshot.key);
                    });
                }catch(error){
                    console.log("Error getPlayerData" + error);
                }
            }
        });
    };//end getPlayerData
```



Connect with Firebase via JS (Create User)

1. Start with the skeleton JS script that Firebase provides
2. Create a form using inputs and a button
3. Create your button listener
4. Test retrieve your form values
5. Let's create auth data using **createUserWithEmailAndPassword** and **userCredential**

Note: How we reference nodes/data depends on how you structure your data inside Firebase Database

```
//Working with Auth
const auth = getAuth();
//retrieve element from form
var frmCreateUser = document.getElementById("frmCreateUser");
//we create a button listener to listen when someone clicks
frmCreateUser.addEventListener("submit", function(e){
  e.preventDefault();
  var email = document.getElementById("email").value;
  var password = document.getElementById("password").value;
  createUser(email, password);
  console.log("email" + email + "password" + password);
});

//create a new user based on email n password into Auth service
//user will get signed in
//userCredential is an object that gets
function createUser(email, password){
  console.log("creating the user");
  createUserWithEmailAndPassword(auth, email, password)
    .then((userCredential)=>{
      //signedin
      const user = userCredential.user;
      console.log("created user ... " + userCredential.toJson());
      console.log("User is now signed in ");
    }).catch((error)=>{
      const errorCode = error.code;
      const errorMessage = error.message;
      console.log(`ErrorCode: ${errorCode} -> Message: ${errorMessage}`);
    });
}
```



Additional Reading

Additional Reference CRUD with Firebase RealTimeDB & Web

<https://www.youtube.com/watch?v=oxqVnWPg0So>

Simple Firebase DB Setup with Web

<https://www.youtube.com/watch?v=S8D9Cxb2ILA>

Working with #Auth.CurrentUser

Auth.CurrentUser

The Authentication in Firebase is very powerful and packed with features.

Once we are logged in, we can use our authentication reference to retrieve the current user session and get user's details (userId, DisplayName, ProfilePic, etc)

Additional Reading

<https://firebase.google.com/docs/database/security/indexing-data>



Working with #Indexes

What is an Index (Optimisation)

An index is a powerful tool. Say for example, our NRICs are unique and we know that it is unique. So it is treated like a key in our database. So once we know exactly the key we can retrieve the data

In databases, an index works by “compiling” that data nicely. So that we can sort our data efficiently.

In Firebase terms we use

.OrderByChild(“somechildproperty”) or
.OrderByKey(“somekey”)

When we index, the database will query and find the data much more efficiently. However, having said so, firebase is pretty efficient. So it depends on how much data you have, and how you want to manipulate the data.

```
{  
  "rules": {  
    ".read": true, // 2021-11-11  
    ".write": true, // "now < 1636560000000", // 2021-11-11  
    "playerStats": {  
      ".indexOn": ["highScore"]  
    },  
    "leaderboards": {  
      ".indexOn": ["highScore"]  
    }  
  }  
}
```

Using OrderByChild hence we index the child properties

Additional Reading

<https://firebase.google.com/docs/database/security/indexing-data>

