Eating, wiring

(javascript)

Bricks, painting

(CSS)

House Plan

(HTML)

house

Use of Firefox inspect tools

To clear the console clear() and enter

**Javascript code to change background at every click**

const htmlBody = document.querySelector('body');

const randomClickFunction = function(){

const colors = ["green", "orange", "grey", "red", "blue" , "green"];

const randomIndex = Math.floor(Math.random() \* colors.length);

const randomColor = colors[randomIndex];

htmlBody.style.backgroundColor = randomColor;

console.log('The user clicked and the color is set to ' + randomColor);

}

htmlBody.onclick = randomClickFunction

Simple button modifications and application of simple javascript

<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <style>

  button{

   background-color: transparent;

   border: 1px solid navy;

   padding: 20px;

   font-size: 1.4rem;

   border-radius: 10px;

  }

  button:hover{

   background-color: navy;

   border: none;

   color: white;

  }

 </style>

 <meta http-equiv="X-UA-Compatible" content="IE=edge">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>Example</title>

</head>

<body>

 <button>CLICK ME</button>

 <div class="container"></div>

 <script>

  function onClickEvent(){

   const el = document.createElement('p');

   el.innerText = 'Clicked the button';

   document.querySelector('.container').appendChild(el);

  }

  document.querySelector('button').onclick = onClickEvent;

 </script>

</body>

</html>

**JavaScript Variables and Data Types**

const myArray = ["10", "20", "30", "40", "50"];

myArray

Array(5) [ "10", "20", "30", "40", "50" ]

myArray.push("70")

6

myArray

Array(6) [ "10", "20", "30", "40", "50", "70" ]

myArray.reverse()

Array(6) [ "70", "50", "40", "30", "20", "10" ]

Use console.log() to print something in console

One line comment //

Multiline comment /\* \*/

**Variables and datatypes**

Variable declaration and assignment

const firstVar = 'Amitha Shehan';

const secondVar = 50;

const thirdVar = {firstProperty: 'hello world'};

console.log(firstVar)

console.log(secondVar)

console.log(thirdVar)

declaring variable first and then assigning value to it

let newVar;

newVar = 50;

console.log(newVar)

**Accepted declaration and assigning**

Const var1 = 50; //cannot be reassigned/redeclared

Let var2 = 90; //used for reassignment

Var var3 = 50; // this is a legacy method. Do not recommend using this. Var can be redeclared

Use const when we have values not to change as the program runs

Use let when we assign new values to variables in the long run

Use capslock when defining const variables

const TAX\_RATE = 0.08;

**Use of const and function**

const variable1 = 10;

//const variable2 = variable1;

const variable3 = "Amitha";

const variable4 = {variableType:"object",variableValue:"somevalue"};

const variable5 = (function(){

return "Hello , my name is ";})();

const variable6 = variable5 + variable3;

**Boolean variables**

Const variable7 = false

**To get to know the datatype of a declared variable**

typeof variable1;

javascript is a dynamically typed language which means we do not need to define the variable type (number, string, Boolean) when we declare a variable.

Converting a string number and add it to a number

Both single quotes and double quotes can be used in string declaration. It is recommended to use single quotes

When assigning values to a number do not use commas

**Arrays**

Varieties of data can be included into arrays’

const array2 = [10, 'string 1', {prop:'sdkfjd'}, [10,20,330]];

console.log(array2[0]);

console.log(array2[1]);

console.log(array2[2]);

console.log(array2[3]);

console.log(array2[3][2]);

**Objects**

const objectVariable = {prop1: 20, prop2:50};

to access a certain property

objectVariable.prop2;

or

objectVariable[‘prop1’]

Nested object declaration and access to them using dot notation

const nestedVariable = {

layer1: {

layer2:{

layer3:{

targetValue: 20}}}}

nestedVariable.layer1.layer2.layer3.targetValue

**Functions**

Defining a function

const functionContainerVariable = function(){

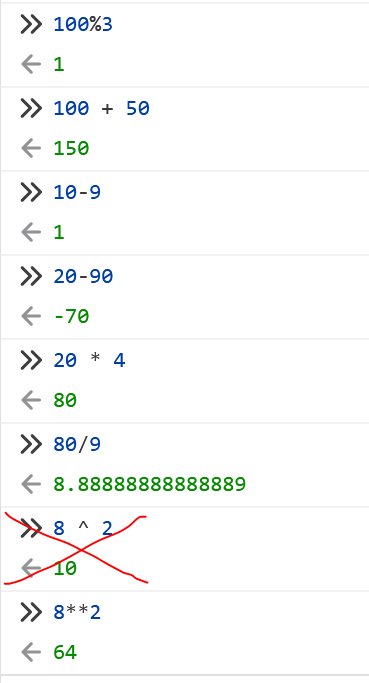
return 20;

}

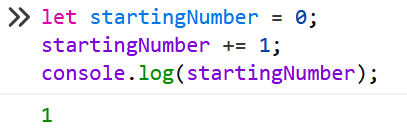
Calling a function

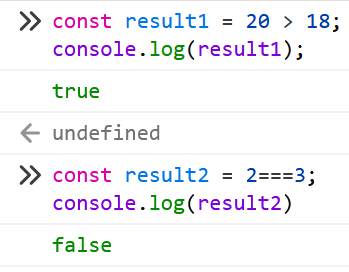
functionContainerVariable()

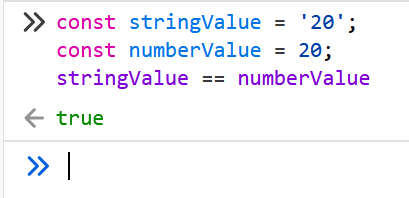
**Javascript operators**

Arithmetic operators

Increment and decrement operator does not work on const variables, but works on let.

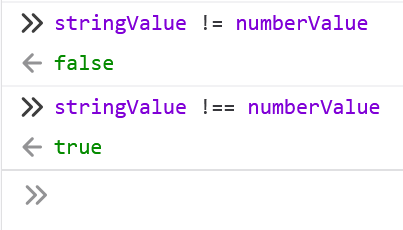


**Comparison operators**



JS converts both variables in to one type and then compares them, we get false if we try

stringValue === numberValue;

double equal does not compare data type, but triple equal compares both data type and value

If we compare two arrays, objects with different names but content is same, when we == or === them answer is false

We cannot use equal operators == or === on arrays and objects.

To do so we can use library called lodash.

**Ternary operator**

const result = 20 === 20 ? 'values match': 'values does not match';

Ternary operator is same as if condition

let resultVariable;

if(20 == 20){

resultVariable = 'values match';

}

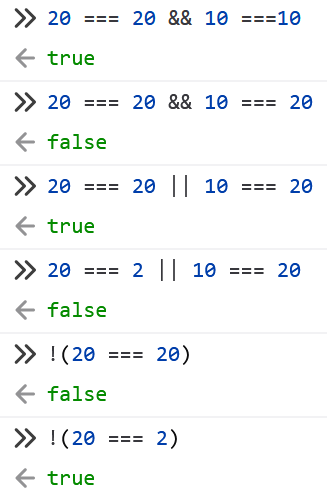
else{

resultVariable = 'values do not match';

}

console.log(resultVariable);

**Logical Operators**



**Conditions**

**If condition if(){} else{}**

if('some string' === 'another string'){

console.log('Strings are equal');

}else{

console.log('strings are not equal');

}

**Switch-case**

const colors = ["red", "green", "blue", "orange"];

const randomIndex = Math.floor(Math.random() \* colors.length);

const randomColor = colors[randomIndex];

console.log(randomColor);

switch (randomColor) {

case "orange":

console.log("color is orange");

break;

case "blue":

console.log("color is blue");

break;

case "green":

console.log("color is green");

break;

case "red":

console.log("color is red");

break;

default:

console.log('no color found');

}

**Looops**

for (let i = 0; i < 10; i++){

console.log(i);

}

To print only numbers in a array

for (let i = 0; i < array.length; i++){

if (typeof array[i] == 'number') {

console.log(array[i]);

}

}

Arrays, Objects and loops in one code

const myArray1 = [

{

name: "amitha",

age: 29,

},

{

name: "kasun",

age: 19,

},

{

name: "John",

age: 24,

},

{

name: "kane",

age: 89,

},

];

for (let i = 0; i < myArray1.length; i++){

const theName = myArray1[i].name;

const theAge = myArray1[i].age;

console.log(theName + ' is ' + theAge + ' years old')

}

**Functions**

Function myFunction(){content}

After defining a function we need to execute, call or invoke the function

function myFunction() {

console.log('hello world this is my first javascript function');

}

myFunction();

The way to call the function just after defining the function (immediately invoked function) (declaring and execution at the same time)

**(function myFunction(){content})()**

(function myOtherFunction() {

console.log('declared and executed in same line ');

})();

**Parameters and Arguments**

function myFunction1(param1, param2) {

console.log(param1);

console.log(param2);

}

myFunction1('argument 1', 'argument 2');

**Another way to declare function in JS**

const anotherFunction = function () {

console.log('another function');

}

anotherFunction();

This is an anonymous function. The function does not have a name, but it is assigned to a variable.

**Arrow Function**

const arrowFunction = () => {

console.log('I am arrow function');

}

arrowFunction();

**Return statement in functions**

const myFunction = () => {

const a = 15;

return a;

}

console.log(myFunction());

we can return Boolean values and functions as well.

We should not put any line of code after return statement

It’s not going to be read after return statement

const myNumber = 50; // defining a global variable

let myFunction = () => {

if (myNumber < 25) { //accessing global variable

return 'the number is less than 25';

}

return 'the number is greater than 25';

}

console.log(myFunction());

**Assigning a variable to a function**

function myFunction() {

return 50;

}

console.log(myFunction());

const aliasVariable = myFunction;

console.log(aliasVariable() + 20);

**Assigning a variable to an object**

function myFunction() {

return 50;

const myObj = {

prop1: 20,

prop2: myFunction

}

console.log(myObj.prop2());

When a function is bound to an object it is called a method.

const myName = 'Amiths';

console.log(myName.replace('s', 'a'));

console.log(myName);

const myCorrectName = myName.replace('s', 'a');

console.log(myCorrectName);

We cannot change myName variable, but instead we can use replace method to assign a new variable.

We can call methods one upon another

const str = 'Hello World';

console.log(str.toUpperCase().split(" ").indexOf('WORLD'));

**Built in JavaScript utility methods**

**Callback functions**

function myCallback(someNumber){

return someNumber \* 2;

}

function mainFunction(randomNumber, shouldCall, callback){

let result = randomNumber;

if(shouldCall){

result = callback(randomNumber)

}

return result;

}

mainFunction(20,true,myCallback);

or without declaring the function in a separate block we can include it in the mainFunction calling block

mainFunction(20, true, function(num){

return num\*2;

})

We can use arrow function as well.

mainFunction(20, true, (num) => {

return num \* 2;

})

Another way

mainFunction(20, true, num => num \* 2)

Reason for using call back functions

\* Reusablility

\* Asynchronous programming

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

---------------------------------------------------------------------------------------

const myArray = [2, 4, 6, 8];

function myCustomMapOperationCallback(eachElementFromArray){

return eachElementFromArray \* 2; //trasformed value

}//map function creates a new array it cannot modify existing array

//.map is the built in function for array

const myNewArray = myArray.map(myCustomMapOperationCallback);

console.log(myNewArray);

const strArr = ['amitha', 'shehan', 'perera'];

const newStrArr = strArr.map(eachElementFromArray => eachElementFromArray[0])

console.log(newStrArr)

Array(3) [ "a", "s", "p" ]

---------------------------------------------------------------------------------------

basic idea is we can use .map function over and over again

Primitives

everything in JS is an object.

--------------------------------------------------------------------------------------

const string1 = new String('hello, world!'); // this is an object, not a string, new is an operator

const string2 = 'hello, world!'; //this is a primitive

console.log(string1 === string2); //false

console.log(string1 == string2); //true

--------------------------------------------------------------------------------------

false

true

--------------------------------------------------------------------------------------

const myDate = new Date(); //creates a new instance of Date object (taking a copy)

javascript Date (refer zack gollwitzer guide - a complete guide to javascript dates)

--------------------------------------------------------------------------------------

const myDate = new Date();

myDate

myDate.toString()

myDate.toISOString()

myDate.getMonth()

myDate.getFullYear()

--------------------------------------------------------------------------------------

Javascript Expressions

knowing a wrong email address

used for validation

pattern matching

--------------------------------------------------------------------------------------

const emailValidatorRegEx = new RegExp('^.+@.+\..+$');

const userInput = 'invalidemail@g';

const isValid = emailValidatorRegEx.test(userInput);

console.log(isValid);

--------------------------------------------------------------------------------------

^ - start at the beginning of text

. - all characters

+ - i want to match any character one or more time

@ - looking to @ symbol

+ - i want to match any character one or more time

\ - escape

. - look for a period

. - any characters

+ - i want to match any character one or more time

$ - end of the line

Regular Expressions is a combination of identifiers and quantifiers

identifiers - can be special characters, or phraces that we are trying to match

--------------------------------------------------------------------------------------

const regex1 = RegExp('g');

const string1 = 'my favorite food is steak';

const string2 = 'my favorite thing to do is code';

regex1.test(string1)

--------------------------------------------------------------------------------------

false

we can compare not only characters but also phraces like 'favorite'.

this is case sensitive

common way of writing regular Expressions

/favorite/ test(string1)

comparing character groups

/[a-z]/.test(string1) //compares simple letters from a to zack

[A-Z] compares caplital letters from A - Z.

[0-9] compares digits 0-9

[A-Z0-9a-z ] compares capital letters + simple letters + digits + spaces

[\d] same as [0-9]

[\s] white space characters

[\w] all letters and numbers

/^f/ first letter of the text f(if is f true otherwise false)

/f$/ first letter of the text f(if is f true otherwise false)

/(code | steak)/ match either code or steak

Quantifiers

/[a-z]/.exce(string1) returns Array ["m"] because first letters matches

/[a-z ]{5}/exec.(string1) retur first five characters including space

Flags

const str = 'hello world, 2021 @ more of a string'; //to match the following string

/^[a-z ]+,[0-9 ]+@[a-z ]+$/.exec(str)

or

/.+/.exec(str) // this checks any character one or more times

String methods

we will be covering replaceAll, toUpperCase, substring, trim, match

const myString = 'My Dog jumped on the bed, my dog is a bad Dog';

const newString = myString.replaceAll('Dog', 'cat');

console.log(newString);//this is case sensitive

//so lets use regular expressions

const correctStr = myString.replaceAll(/[Dd]{1}og/g,'cat') // g is the global flag

console.log(correctStr);

str.toUpperCase();

str.substring(indexStart, [indexEnd]) // indexEnd is optional.

to capitalize first letter in a sentence

const myString = 'hello world'

const result = myString[0].toUpperCase() + myString.substring(1, myString.length);

Trim method removes white spaces from both ends not in the middle, use full when removing white spaces in start and end from data from external sources

Array methods

pop removes the last element in the array

const arr = [1,2,3,4,5]

arr.pop()//removed 5 from array

push adds element(s) to the end of the array

arr.push(5,6,7)

//arr = [1,2,3,4,5,6,7]

unshift adds elements to the beginning of the array

arr.unshift(0)// arr = [0,1,2,3,4,5,6,7]

slice creates a basic copy(not nested elements) of an array slide(start,[end])

const copyArr = arr.slice();//copyArr = [0,1,2,3,4,5,6,7]

const copyArr2 = arr.slice(3) //copyArr2 = [3,4,5,6,7]

The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place

To create a new array with a segment removed and/or replaced without mutating the original array, use toSpliced(). To access part of an array without modifying it,

splice(start)

splice(start, deleteCount)

splice(start, deleteCount, item1)

splice(start, deleteCount, item1, item2, itemN)

const months = ['Jan', 'March', 'April', 'June'];

months.splice(1, 0, 'Feb');

// Inserts at index 1

console.log(months);

// Expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');

// Replaces 1 element at index 4

console.log(months);

// Expected output: Array ["Jan", "Feb", "March", "April", "May"]

findIndex() method

The findIndex() method returns the index of the first element in an array that satisfies the provided testing function. If no elements satisfy the testing function, -1 is returned.

we can use call back functions in findIndex() method.

the find() method, which returns the first element that satisfies the testing function (rather than its index).

const array1 = [5, 12, 8, 130, 44];

const isLargeNumber = (element) => element > 13;

console.log(array1.findIndex(isLargeNumber));

// Expected output: 3

const moreComplexArray = [

{

firstName: 'Bob',

lastName: 'Smith'

},

{

firstName: 'Alice',

lastName: 'Smith'

},

{

firstName: 'Jon',

lastName: 'Smith'

},

{

firstName: 'Jon',

lastName: 'Smith'

}

];

//we cannot use indexOf method in array + objects

const moreComplexArray = [

{

firstName: "Bob",

lastName: "Smith",

},

{

firstName: "Alice",

lastName: "Smith",

},

{

firstName: "Jon",

lastName: "Smith",

},

{

firstName: "Jon",

lastName: "Smith",

},

{

firstName: "Jon",

lastName: "Doe",

}

];

console.log(

moreComplexArray.findIndex((arrItem) => {

return arrItem.lastName !== "Smith";

})

);

map method

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

const array1 = [1, 4, 9, 16];

// Pass a function to map

const map1 = array1.map(x => x \* 2);

console.log(map1);

// Expected output: Array [2, 8, 18, 32]

to change the category of all blogposts to webdev

const blogPostsFromDatabase = [

{

title: 'How to use the map() function',

category: 'uncategorized'

},

{

title: 'What is JavaScript',

category: 'uncategorized'

},

{

title: 'Why are you crazy enough to learn to code?',

category: 'uncategorized'

},

];

function changeCategory(arrayItem) {

arrayItem.category = 'Web Development';

return arrayItem;

};

const newDBArray = blogPostsFromDatabase.map(changeCategory);

console.log(newDBArray);

//or we could use

blogPostsFromDatabase.map(arrItem => {

arrItem.category = 'Web Development';

return arrItem;

})

forEach() method

the forEach() method executes a provided function once for each array element. just like for loop

const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// Expected output: "a"

// Expected output: "b"

// Expected output: "c"

const array1 = [1, 2, 3, 4, 5];

array1.forEach(element => console.log(element));

includes() method

The includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.

const array1 = [1, 2, 3];

console.log(array1.includes(2));

// Expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));

// Expected output: true

console.log(pets.includes('at'));

// Expected output: false

filter() method

The filter() method creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);

// Expected output: Array ["exuberant", "destruction", "present"]

const allOrders = [

{

productName: "Tea Pot",

isDigital: false,

isCancelled: false,

isOpen: false,

},

{

productName: "Mens hoodie",

isDigital: false,

isCancelled: true,

isOpen: false,

},

{

productName: "Coding book",

isDigital: true,

isCancelled: true,

isOpen: false,

},

{

productName: "Atomic habits book",

isDigital: true,

isCancelled: false,

isOpen: false,

},

];

const digitalOrders = allOrders.filter((arrItem) => {

return arrItem.isDigital;

});

console.log(digitalOrders);

const cancelledDigitalOrders = allOrders.filter(arrItem => {

return arrItem.isDigital && !arrItem.isCancelled

})

console.log(cancelledDigitalOrders);

reduce() method

The reduce() method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

The first time that the callback is run there is no "return value of the previous calculation". If supplied, an initial value may be used in its place. Otherwise the array element at index 0 is used as the initial value and iteration starts from the next element (index 1 instead of index 0).

Perhaps the easiest-to-understand case for reduce() is to return the sum of all the elements in an array:

simple example is to take the sum of an array

const array1 = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4

const initialValue = 0;

const sumWithInitial = array1.reduce(

(accumulator, currentValue) => accumulator + currentValue,

initialValue

);

console.log(sumWithInitial);

// Expected output: 10

//another way

const array = [1, 2, 3, 4];

function reducerCallBack(sum, arrItem) {

return sum + arrItem;

}

console.log(array.reduce(reducerCallBack, 0));

//another way to write callback function

array.reduce((sum, arrItem) => sum+=arr,0)

we cannot use == or === to check similarity between two objects

const obj1 = { prop1: 'some Value' };

const obj2 = { prop1: 'some Value' };

console.log(obj1 === obj2);

console.log(obj1 == obj2);

//answers we get are

false

false

we can use isEqual() method from lodash library to check similarity between objects

Math object

no need create(cannot) instances

Math.PI //return pi value

Math.abs(-60) //return 60

Math.ceil(Math.PI) //return 4

Math.max(2,5,8) //return 8

Math.random() // return a random value between 0 & 1

//generating a random value between 0-100

/\*

const randomNumber = Math.random();

const randomNumberTo100 = randomNumber \* 100;

const randomNumberFloor = Math.floor(randomNumberTo100);

console.log(randomNumberFloor);

\*/

//all in one line

const randNum = Math.floor(Math.random() \* 100);

console.log(randNum)

const myArray = [2, 5, 6, 8, 1.2, 23];

const randomIndex = Math.floor(Math.random() \* myArray.length);

console.log(myArray[randomIndex]);

javascript errors

3 main types of errors

1.reference error

2.syntax error

3.type error

reference error

The ReferenceError object represents an error when a variable that doesn't exist (or hasn't yet been initialized) in the current scope is referenced.

syntax error

The SyntaxError object represents an error when trying to interpret syntactically invalid code

type error

The TypeError object represents an error when an operation could not be performed, typically (but not exclusively) when a value is not of the expected type.

A TypeError may be thrown when:

an operand or argument passed to a function is incompatible with the type expected by that operator or function; or

when attempting to modify a value that cannot be changed; or

when attempting to use a value in an inappropriate way.

ex- when we add 2 objects

Error Handling

try {

var num = 45;

num.toUpperCase();

}

catch (error) {

console.log(error instanceof TypeError);

console.log(error.message);

}

//answer is

true

num.toUpperCase is not a function

//NaN, null, undefined

we get NaN when we are going to convert a string into a number or doing math operations on it

const myString = 'some string';

Number(myString) //NaN

null

const num = null; //null has a false value

if(num){

console.log(this will not be reached)

} else{

console.log(this will be reached)

}

undefined

just declaring a variable and not assigning a value

works same as in if condition

to extract digits in a number

const num = 299;

+num.toString()[0];

**Document Object Model (DOM)**

window.scrollBy({top:window.innerHeight,

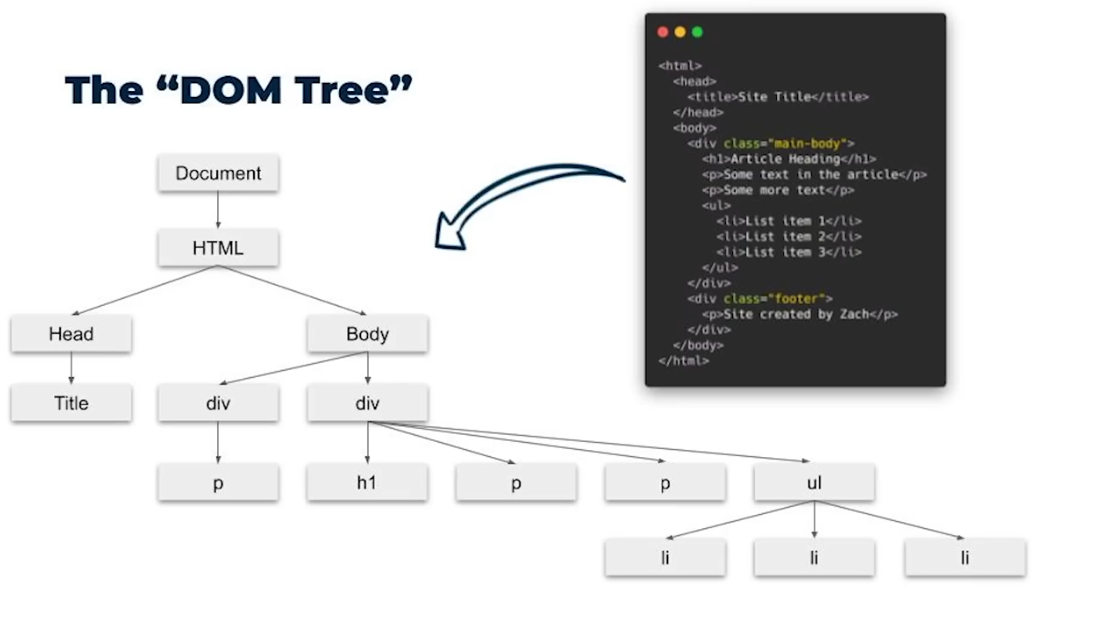
behavior:'smooth'})  
  
to reload the window

window.location.reload()

We cannot use dot notation to access html elements.

Document.html.body //does not work

We can think of a tree. We call them nodes



<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <meta http-equiv="X-UA-Compatible" content="IE=edge">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>Document</title>

</head>

<body>

 <div class="main-body">

  <h1>Article Heading</h1>

  <p>some text in the article</p>

  <p>some text</p>

  <ul>

   <li>item 1</li>

   <li>item 2</li>

   <li>item 3</li>

  </ul>

 </div>

 <div class="footer">

  <p>Site created by Amitha</p>

 </div>

 <script src="./ex2.js"></script>

</body>

</html>

Jf file to add a paragraph content at the end

const elementNode = document.createElement('p');

const textNode = document.createTextNode('Newly added text');

const attributeNode = document.createAttribute('class');

elementNode.appendChild(textNode);

attributeNode.value = 'some-class';

elementNode.setAttributeNode(attributeNode);

document.body.appendChild(elementNode)

Add a new paragraph at each click of a button

1. Identify the button element
2. Register a click event on this button
3. Write a function

In the following example btn-1 and my-btn are gloabal attributes

Different methods to select button in javascript

document.querySelector('button')

document.querySelector('#btn-1');

document.querySelector('.my-btn');

document.getElementById('btn-1'); // best way

HTML file

<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <meta http-equiv="X-UA-Compatible" content="IE=edge">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>Document</title>

</head>

<body>

 <h2>Welcome to my HTML Lessons</h2>

 <p>click the button below to begin</p>

 <button id = 'btn-1' class="my-btn">Click Me</button>

 <script src="./example1.js"></script>

</body>

</html>

JS file

const btn = document.getElementById('btn-1');

function addParagraph() {

 const randomNum = Math.floor(Math.random() \* 100);

 const pContent = 'the random number is ' + randomNum;

 const newElement = document.createElement('p');

 newElement.textContent = pContent;

 document.body.appendChild(newElement);

//console.log(pContent)

}

btn.addEventListener('click', addParagraph);

/\*

or

btn.onclick = addParagraph;

or

<button id = 'btn-1' class = 'my-btn' onclick = "addParagraph()">Click Me</button>

\*/

How to access a certain element in html document using js

document.querySelector('ul')

to get to know what type of variable we are working with use this technique

const list = document.querySelector('ul');

list.nodeType

this returns 1 means it is html collection

following methods can be implemented in html collections

list.children

list.children.item(1)//returns second item in the list

to change the text in html element(li)

const listItem = list.children.item(1);

listItem.textContent = 'some new text';

html file

<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <meta http-equiv="X-UA-Compatible" content="IE=edge">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>Document</title>

</head>

<body>

 <div class="main-body">

  <h1>Article Heading</h1>

  <p>some text in the article</p>

  <p>some text</p>

  <ul>

   <li>item 1</li>

   <li>item 2</li>

   <li>item 3</li>

  </ul>

 </div>

 <div class="footer">

  <p>Site created by Amitha</p>

 </div>

</body>

</html>

To add a new li item to the end

listItem.appendChild(document.createElement('li'));

list.children.item(3).textContent = 'new list item';

query selector identifies only the first relevant element. If we want to select the second p tag in above html document

document.querySelectorAll('p').item(1);

**insertbefore in js**

The insertBefore() method of the Node interface inserts a node before a reference node as a child of a specified parent node.

const pElm = document.querySelectorAll('p').item(1);

const parent = document.querySelector('.main-body');

const newP = document.createElement('p');

newP.textContent = 'programmatically added between';

parent.insertBefore(newP, pElm);

**SEMENTIC HTML**

Use nav, section, footer, main, article tags in html files

**CSS**

CSS Combinators

we can select HTML elements in several ways.

Here are a few

1. Descendant Selectors
2. Child Selectors
3. Sibling Selectors

**Descendant Selectors**

Selects children and all grand children

Html file

<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <meta http-equiv="X-UA-Compatible" content="IE=edge">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <link rel="stylesheet" href="./lesson1.css">

 <title>Document</title>

</head>

<body>

 <div class="main-content">

 <h1>CSS Combinators</h1>

<p>we can select HTML elements in several ways.</p>

<p>Here are a few</p>

<ul>

 <li>Descendant Selectors</li>

 <li>Child Selectors</li>

 <li>Sibling Selectors</li>

</ul>

<div>

 <ol>

  <li>List item 3</li>

 </ol>

</div>

</div>

<div class="footer">

 <ul>

  <li>footer link 1</li>

  <li>footer link 2</li>

  <li>footer link 3</li>

 </ul>

</div>

</body>

</html>

Css file

.main-content li{

 color: green;

}

Output

**CSS Combinators**

we can select HTML elements in several ways.

Here are a few

1. Descendant Selectors
2. Child Selectors
3. Sibling Selectors
4. List item 3

* footer link 1
* footer link 2
* footer link 3

all the list items in main-content class are colored green.

Child Selectors

Html body

<body>

<div id="main">

 <p>first paragraph</p>

 <p>second paragraph</p>

 <p>third paragraph</p>

 <div>

  <p>fourth paragraph</p>

 </div>

</div>

</body>

Css

#main > p{

 color: red;

}

Output

first paragraph

second paragraph

third paragraph

fourth paragraph

Only direct children are colored green.

Sibling Selectors

Selects the first element after a list

Html body

<body>

<div id="main">

 <ul>

  <li>List item 1</li>

  <li>List item 2</li>

  <li>List item 3</li>

 </ul>

 <p>paragraph 1</p>

 <p>paragraph 2</p>

 <p>paragraph 3</p>

 <ul>

  <li>list 1</li>

  <li>list 2</li>

  <li>list 3</li>

 </ul>

  <p>paragraph 1</p>

 <p>paragraph 2</p>

 <p>paragraph 3</p>

</div>

</body>

Css

ul + p{

 color: blue;

}

Output

* List item 1
* List item 2
* List item 3

paragraph 1

paragraph 2

paragraph 3

* list 1
* list 2
* list 3

paragraph 1

paragraph 2

paragraph 3

if the html structure changes the styling may break. So its not recommended to use this method. And this is rarely used

method to select all the sibling under ul

css

ul ~ p{

 color: blue;

}

Output

* List item 1
* List item 2
* List item 3

paragraph 1

paragraph 2

paragraph 3

* list 1
* list 2
* list 3

paragraph 1

paragraph 2

paragraph 3

if we want to style a certain element (but class name on several elements are same)

html body

<body>

<p class="regular-text">some text here</p>

<div class="regular-text">some other text</div>

</body>

Css

p[class = regular-text]{

 color: red;

}

Output

some text here

some other text

to apply same styles to more classes use comma separator to select required classes

or else we can add multiple classes in html (shared class)

html body

<body>

<div class="box-1 box">

 <p>Box 1</p>

</div>

<div class="box-2 box">

 <p>Box 2</p>

</div>

Css

.box-1{

 border: 1px solid red;

 padding : 20px;

 color: blueviolet;

}

.box-2{

 border: 1px solid green;

 padding : 20px;

 color: red;

}

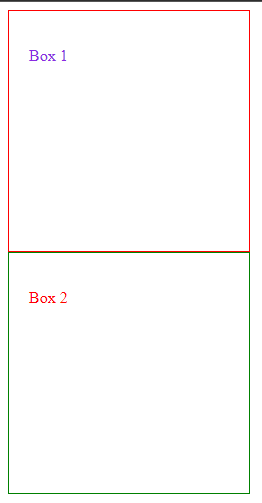
.box{

 width: 200px;

 height: 200px;

}

output



**Pseudo elements**

Change color on hover

Html body

<button>Hover me</button>

Css

button{

 background-color: navy;

 color: white;

 border: none;

 padding: 20px;

}

button:hover{

 cursor: pointer;

 opacity: 0.8;

}

Precedence in css

Two factors determine CSS precedence

* 1. location of the CSS rule in the file – applies the last(read from top to bottom, last rule applies)
  2. specificity of the CSS rule (tags < class < id)
     1. count the number of ID selectors
     2. count the number of class selectors
     3. count the number of tag selectors

If it combines with a combinatory precedence is very high (i.e- ol #list-item{}

Inline styles take precedence over everything

To override any style from any style sheet we can use !important keyword (overrides inline styling as well)

#paragraph1{

 color: green !important;

}

Use inline and important as low as possible

Simple use of html and css to create a box and center some text inside

Html body

<div>

 <p>Some Text</p>

</div>

Css

div{

 background-color: navy;

 color: white;

 width: 200px;

 height : 200px;

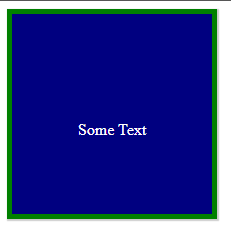
 line-height: 200px;

 text-align: center;

 box-shadow: 1px 1px 1px 1px rgba(0,0,0,0.2);

 border : 5px solid green

}

Output

**CSS display property**

Block elements use entire width and always start in a new line, width and height can be modified.

Inline elements use only required width and can start in the same line

If we make an inline element it does not work with height and width properties

Inlineblock can work with width and height

**Box model variations**

In normal border (content box) padding and border thickness adds with given height and width, but if we want to put everything in given height and width

Box-sizing = border-box;

Html{

Box-sizing: border-box;

}

\*, \*:before, \*:after{

Box-sizing:inherit;

}

**Position property**

Default is static

Fixed : used in nav bars to make the nav bar fixed on top of the web page

Html body

<nav>

      <h4>this is the nav bar</h2>

    </nav>

    <div class="boxes">

      <div class="box"></div>

      <div class="box"></div>

      <div class="box"></div>

      <div class="box"></div>

      <div class="box"></div>

    </div>

CSS

h4{

 margin: 0;

 padding-left: 10px;

}

body{

 margin: 0;

}

nav{

 background-color: navy;

 height: 60px;

 color: white;

 line-height: 60px; /\*to center the text vertically\*/

 position: fixed;

 width: 100%;/\*to take up the entire space 100vw\*/

 top: 0; /\*to move the nav to the top most\*/

 left: 0; /\*to move the nav to the left most\*/

}

.box{

 width: 200px;

 height: 200px;

 background-color: green;

 margin: 20px;

}

.boxes{

 margin-top: 60px; /\*to move down boxes from the nav bar\*/

}

Absolute and relative positioning

Html body

 <body>

    <nav>

      <h4>this is the nav bar</h2>

    </nav>

    <div class="boxes">

      <div class="box rel-position">

        <div class="abs-position"></div>

      </div>

      <div class="box"></div>

      <div class="box"></div>

      <div class="box"></div>

      <div class="box"></div>

    </div>

Css

h4{

 margin: 0;

 padding-left: 10px;

}

body{

 margin: 0;

}

nav{

 background-color: navy;

 height: 60px;

 color: white;

 line-height: 60px; /\*to center the text vertically\*/

 position: fixed;

 width: 100%;/\*to take up the entire space 100vw\*/

 top: 0; /\*to move the nav to the top most\*/

 left: 0; /\*to move the nav to the left most\*/

 z-index: 1;

}

.box{

 width: 200px;

 height: 200px;

 background-color: green;

 margin: 20px;

}

.boxes{

 margin-top: 80px; /\*to move down boxes from the nav bar\*/

}

.rel-position{

 position: relative; /\*enables z index property\*/

 top: 10px;

  left: 10px;

}

.abs-position{

 position: absolute; /\*absolute relate to the immediate parent \*/

 width: 20px;

 height: 20px;

 background-color: orange;

 top: 20px;

}

CSS measurement units

Padding, margin ranges from 10-20px

Border ranges from 4-7px

Em and rem are specific to fonts

If we define font size in html{font-size:16px}, in a following div font-size: 1.2rem the font size would become 16X120%.

Em works same as this, the difference is we can define font size in pixels in parent element

Better to define font size in html{}tag

Better to use fallback font which is the second argument.

We can only use copied thickness from google fonts

Use of google fonts

Html file

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <link rel="stylesheet" href="./lesson1.css" />

    <link rel="preconnect" href="https://fonts.googleapis.com" />

    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />

    <link

      href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"

      rel="stylesheet"

    />

    <title>Document</title>

  </head>

  <body>

    <h2>Heading element</h2>

    <p>some paragraph</p>

    <p id="original">original style text</p>

  </body>

</html>

CSS file

html{

  /\* font-family: 'Gill Sans',sans-serif; \*/

  font-family: 'Roboto', sans-serif;

  font-size: 16px;

  font-weight: 700;

}

Line height property gives us the space between texts in vertical manner

**Background images**

HTML body

<body>

    <div class="main-content">

      <h1>some text</h1>

    </div>

  </body>

CSS file

body{

  margin: 0;

  color: white;

  text-align: center;

}

h1{

  margin: 0;

  padding: 50px;

}

.main-content{

  background-image: url(./Images/1.JPG);

  width: 100vw;

  height: 100vh;

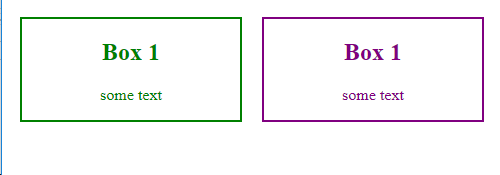
  background-size: cover;

  background-repeat: no-repeat;

  background-position: center;

}

Simple challenge



Html body

<div class="container">

      <div class="box-1">

        <h2>Box 1</h2>

        <p>some text</p>

      </div>

      <div class="box-2">

        <h2>Box 1</h2>

        <p>some text</p>

      </div>

    </div>

CSS file

.container{

  display: flex;

}

.box-1 {

  border: 2px solid green;

  color: green;

}

.box-2 {

  border: 2px solid purple;

  color: purple;

}

.box-1, .box-2{

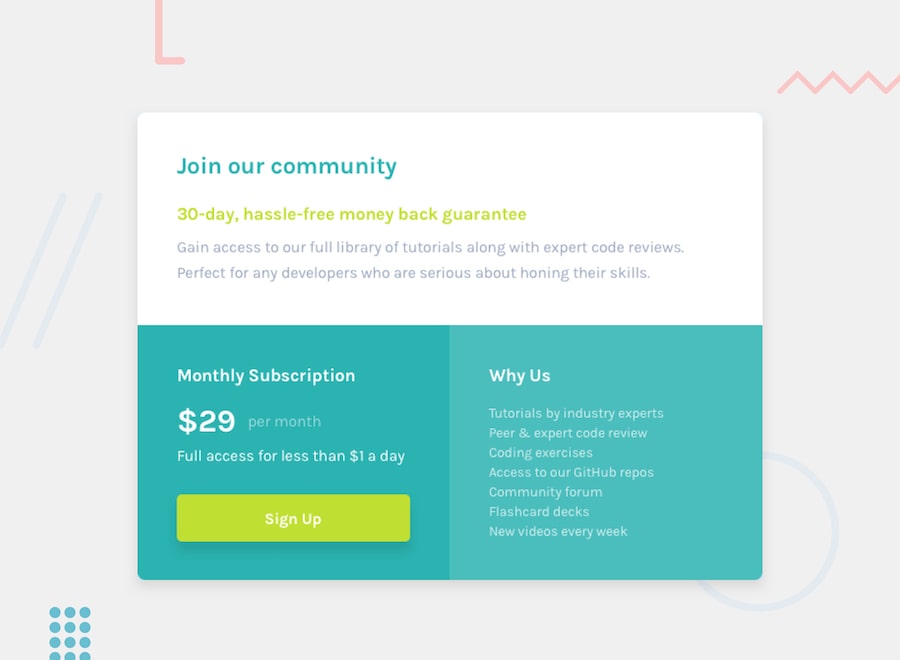
  width: 50%;

  margin: 10px;

  text-align: center;

}

Challenge 2



HTML code

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Pricing Card Challenge</title>

    <link rel="stylesheet" href="./lesson2.css" />

    <link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?family=Karla:wght@400;700&display=swap" rel="stylesheet">

  </head>

  <body>

    <div id="card">

      <div id="top-section">

        <h2>Join our community</h2>

        <h4>30-day, hassle-free money back guarantee</h4>

        <p>

          Gain access to our full library of tutorials along with expert code

          reviews. Perfect for any developers who are serious about honing their

          skills.

        </p>

      </div>

      <div id="bottom-section">

       <div id="bottom-left-section">

          <h4>Monthly Subscription</h4>

          <div id="price-container">

            <div id="price">&dollar;29</div> //this is to 1\*

            <div id="price-frequency">per month</div>

          </div>

          <p>Full access for less than &dollar;1 a day</p>

          <button onclick="alertUser()">Sign Up</button>

        </div><div id="bottom-right-section">

          <h4>Why Us</h4>

          <ul>

            <li>Tutorials by industry experts</li>

            <li>Peer &amp; expert code review</li>

            <li>Coding exercises</li>

            <li>Access to our GitHub repos</li>

            <li>Community forum</li>

            <li>Flashcard decks</li>

            <li>New videos every week</li>

          </ul>

        </div>

      </div>

    </div>

  </body>

  <script src="./lesson2.js"></script>

</html>

1\* = to avoid the bottom div to go down. We need to clear all white spaces available. That’s why ending div and starting div are in same line

CSS

html {

  box-sizing: border-box;

  font-family: 'Karla', sans-serif;

  font-size: 16px;

  font-weight: 400;

}

body{

 padding: 20px;

}

h2,h4,p{

 margin: 0;

}

\*,

\*:before,

\*::after {

  box-sizing: inherit;

}

#card {

  width: 600px;

  height: 400px;

  box-shadow: 13px -11px 49px -6px rgba(0, 0, 0, 0.29);

  border-radius: 5px;

  overflow: hidden;

  margin: auto;

}

#top-section {

  width: 100%;

  height: 45%;

  padding: 25px;

}

#bottom-section {

  width: 100%;

  height: 55%;

  color: white;

}

#bottom-left-section {

  display: inline-block;

  vertical-align: top;

  width: 50%;

  height: 100%;

  background-color: #00B1A8;

  padding: 25px;

}

#bottom-right-section {

  display: inline-block;

  vertical-align: top;

  width: 50%;

  height: 100%;

  background-color: #00E0D5;

  padding: 25px;

}

#top-section h2{

 color : #00B1A8;

 margin-bottom: 20px;

}

#top-section h4 {

 color: #c0df34;

 margin-bottom: 10px;

}

#top-section p {

 color: #98a6bd;

}

button{

 color: white;

 background-color: #c0df34;

 border: none;

 padding: 10px;

 border-radius: 5px;

 width: 100%;

 font-weight: 700;

}

button:hover{

 cursor: pointer;

 background-color: a9c42d;

 transform: scale(1.02);

}

#price{

 font-weight: 700;

 font-size: 1.4rem;

}

#bottom-left-section h4{

 margin-bottom: 25px;

}

#bottom-left-section p{

 margin-bottom: 40px;

 font-size: 0.85rem;

}

#price-frequency, #price{

 display: inline-block;

}

#price-frequency{

 position: relative;

 top: -3px;

 font-size: 0.8rem;

 opacity: 0.78;

 margin-left: 5px;

}

#price-container{

 margin-bottom: 10px;

}

#bottom-right-section ul{

 padding: 0;

 list-style: none;

 opacity: 0.8;

 font-size: 0.9rem;

}

JS

function alertUser() {

 alert('Button Clicked')

}

**Responsive web designing**

Basic syntax

@media (min-width: 756px){}

There are lots of @ rules in css. @media is one of them.

@media (min-width: 756px){

 h1{

  color: red;

 }

}

When the screen is equal or greater than 756px h1 text color is red

How to target 400-600px media screens

1st we need add operator “only”

Html body

 <h1>Welcome to Responsive web designing</h1>

CSS

@media

only screen

and (min-width:400px)

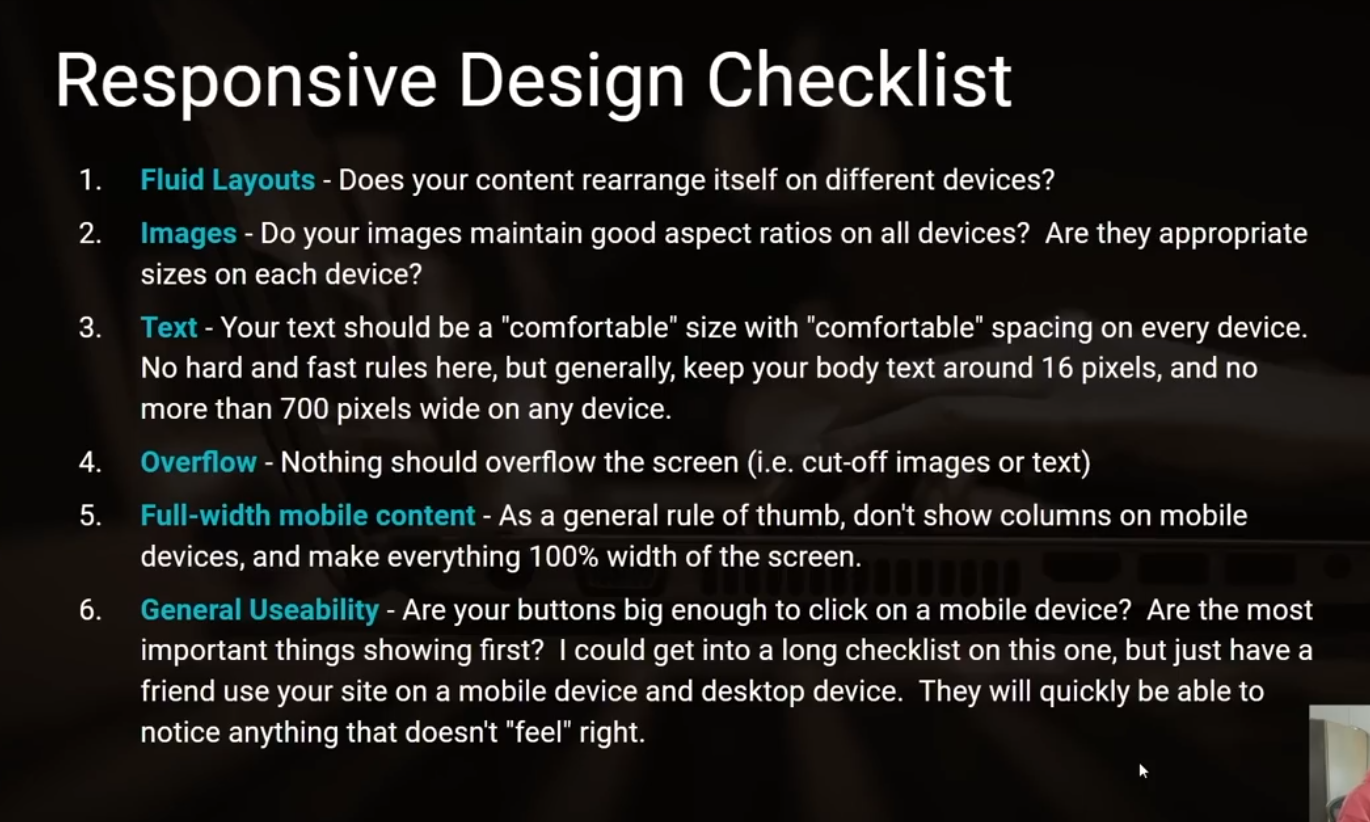
and (max-width:600px) {

 h1{

  color: red;

 }

}

Breakpoint order matters, in min-width go in ascending order as css in read from top to bottom. Greatest width at bottom. Add media queries at the bottom of the page  


HTML body

<h1>Welcome to Responsive web designing</h1>

 <img src="https://placehold.co/400x400" alt="">

 <img src="https://placehold.co/400x400" alt="">

 <img src="https://placehold.co/400x400" alt="">

 <img src="https://placehold.co/400x400" alt="">

CSS

html{

 box-sizing: border-box;

}

\*, \*:before, \*:after{

 box-sizing: inherit;

}

img{

 width: 100%;

}

/\*576px 992px\*/

@media (min-width:576px){

img{

 width: 49%;

}

}

@media (min-width:992px) {

img{

 width: 24%;

}

}

Flex Box

When we put display: flex it applies certain default properties with it.

When flex is applied to the parent the parent is called flex container and the direct children are called flex items. And flex does not apply for grandchildren. A flex item can act as a flex item as well as a flex container in the same time

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

**justify content**

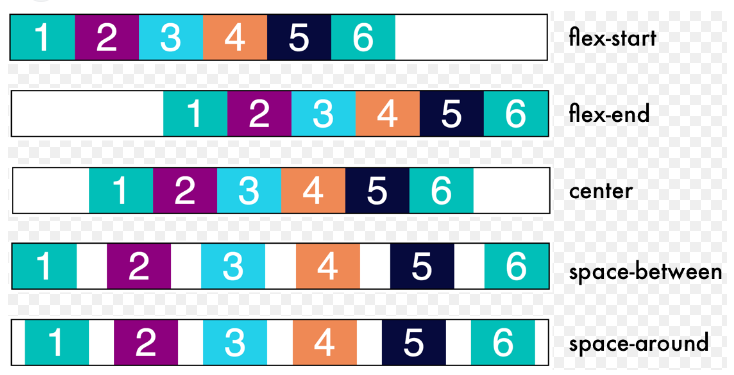
Default value is normal, same as start

If we give end content will move to the end of the container

Center will move them to middle

Space between will spread items with equal white spaces in the container

Space around will space content first and last items also will have some white space from start and end (start and end spaces are comparatively small).

Space even will space all items with equal space

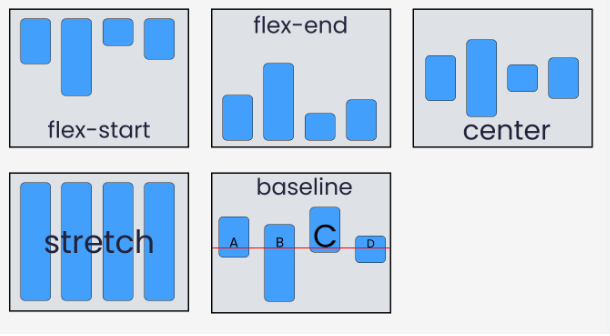
**Align-Items**

By default align items is (normal) stretch, which means they are stretched to the element height of container unless specified.

Flex start will move items to the top and will not have entire height of container

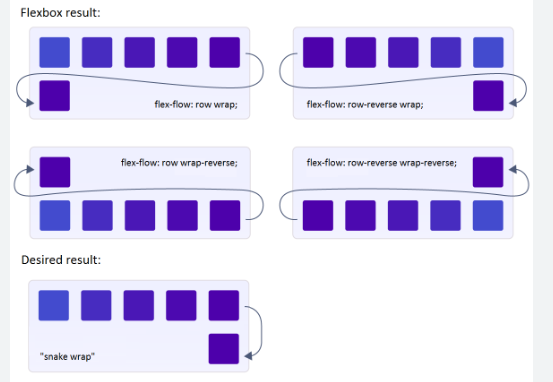
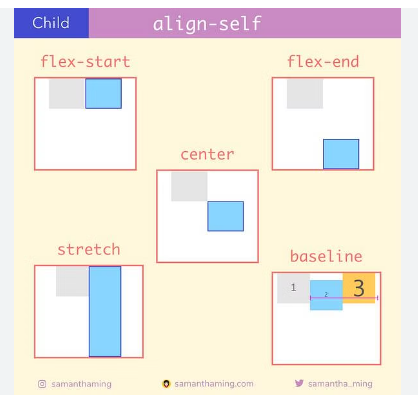
Center will move items to the center of the container and will not stretch

Baseline will move content to the top but it will be centered based on highest element.



If we fill the container with more items than it can bear based on width (px) items’ width(even if we defined) will shrink to accommodate items with maximum possible way.If we set overflow: auto in container we will get a scroll bar within the container. To get rid of the scroll bar and have a nice look remove overflow: auto and add flex-wrap: wrap in container. By this items will go back to its original.

If flex-wrap is set to anything other than default(no-wrap) we need to add align-content :flex-start to remove white spaces in between two lines.



How to set alignment of each flex box item

In item, align-self: flex-start

Flex item sizing (flex grow, shrink and basis)

**Flex-grow**

By default each item takes 0% of empty space. If we make it 1 relevant item will take 100% of empty space. If we apply flex-grow:1 to each item those items will allocate empty space equally. If we want we can give 2, 3, 5 empty allocated space will be allocated proportionally.

If we have more items in container (overflow) than it can hold by default the items will shrink. If we keep adding more items in to the container after shrinking as much as possible extra items will start to overflow. (unless we wrapped)

**Flex Shrink**

If we have 9 items with defined 50px wide in a container of 400px wide. There is a 50px overflow. Overflow value is 50(overflow)/9 = 5.55px. If the content is too big to shrink it will overflow.

If we do not want certain items to shrink we can give flex-shrink:0;

**Flex basis**

Applies on items. Flex-basis:40% - changes item based on defined width. Default is auto. What default does is if width and height are defined, follow those values.

If there are four items within container and there is empty space, in each item flex-basis:25% will stretch all items evenly.

Need to be mindfull if it doesn’t add up to 100 there will be empty space.

In flexbox we can change the order of items. To do so order:x where x is the index number. Note: if we add order:x to one element we must replicate it in every item with respective order(1,2,3..)

* + useful in responsive design

short hand method

flex: flex\_grow flex\_shrink flex\_basis