EPICODE APPUNTI

M1_Week_01:

Tipologie di developers:
frontend - html, css, js, lato client
backend - lato server
full stack - tutto ovvero presentation, business logic, data access

Livelli:
presentation - interfaccia utente (html, css, js) - SPAs(React/Angular/Vue)
business logic - raccolta e invio info (.Net, Node js, ruby, php) - NodeJs, ExpressJS
data access - interazione e conservazione dati (MySql, Sql Server, MongoDB) - MongoDB

HTML
Vuol dire **HyperTextMarkupLanguage** è lo scheletro della pagina web

HTML-DOM
Vuol dire **Document Object Model** struttura della pagina in formato javascript

CSS:
Cascading style sheet, regole che il dom traduce in quello che vede utente
La priorità è !important e stile inline>>id>>classi>>tag

Database Relazionali:
Dati memorizzati sotto forma di tabelle, validazione rigorosa
Database non Relazionali:
Dati in struttura flessibile

HTML - Tags
Definiti con <>, hanno apertura e chiusura , ma non tutti
Possono essere annidati ed avere child, tranne i self closing
<head> è il tag con info principali pagina, import, titolo, metadata
<body> è il tag col contenuto della pagina
Titoli definiti in ordine di importanza da <h1> a <h6>
 è il paragrafo ed è elemento blocco
Elemento blocco si estende per tutta la larghezza della pagina
 evidenzia porzioni di testo
 e <ol> liste non ordinate e liste ordinate
 elemento contenitore di blocco
Attributi dei tag servono a configurare i tag
<a> tag ancora collega pagine o elementi interni o esterni, ha bisogno di href per mandare a URL dell'elemento

CSS - specificità

Box model: tutti elementi del dom sono scatole composte da **contenuto, padding, bordo, margine**
Unità di misura:
px - dimensione assoluta in pixel
em - relativo alla misura del genitore
rem - relativo alla misura del tag principale
vh/vw - relativo alla misura della finestra

Ereditarietà

Alcune regole (tipo colore) sono ereditarie in css, ovvero vengono passate ai child.

Selettori

Classi - possono ripetersi, hanno come selettore .class
Id - devono essere unici, hanno come selettore #id
I selettori vengono usati per definire classi e id nel css

Display

Le proprietà display vengono usate per definire comportamento rendering degli elementi, sono 25 ma le principali sono:
block - rende elemento blocco, occupa 100% della pagina in larghezza
inline - larghezza elemento impostata su fit-content, solo spazio che gli serve
inline-block - uguale ad inline ma prende proprietà di larghezza, altezza, padding, margin
none - nasconde elemento
flex

Position

Utilizzata per definire come viene posizionato elemento, ce ne sono diversi tipi:
Relative - sposta un elemento rispetto alla sua posizione predefinita, simile a fixed, ma non rimane fermo con scorrimento
Absolute - elemento spostato in base all'antenato non statico più vicino, se non c'è si sposta col body
Sticky - elemento posizionato in base alla posizione corrente di scroll
Fixed - posizionato in relazione al viewport, elemento rimane nello stesso posto se si scorre verso il basso
Static - impostazione predefinita, non risente di top, bottom, etc

M1_Week_02

Pseudoclassi
Usate per selezionare elementi in uno stato specifico (hover, focus etc)
Hanno stessa specificità classi
Sono definiti da : quindi a:hover
Alcune di queste sono:
:hover - applica regola con mouse sopra elemento
:focus - applica regola quando elemento ha il focus
:not(<condition>) - applica regola con condition non vera
:first-child || last-child - seleziona elemento primo o ultimo dei child
:nth-child - applica regola a n elementi
:empty - applica regola a elementi senza child

Pseudoelementi
Utilizzato per selezionare parte elementi, tipo p::fist-line
Stesso valore dei tag

flexbox
Per usarlo serve container flex, per crearlo basta impostare display flex su un qualsiasi elemento, tipo ** ** tutti i child saranno elemento flex
Due assi principali row e column che possono anche andare in direzione invertita quindi possiamo avere **flex-direction:row, flex-direction:column, flex-direction:row-reverse, flex-direction:column-reverse**

justify-content
Con **justify-content** definiamo come gli elementi flex si distribuiscono su asse principale e i tipi sono: **flex-start, flex-end, center, space-between, space-around, space-evenly**
align-items
Definisce come elementi flex si distribuiscono su asse trasversale e sono: **flex-start, flex-end, center, stretch, baseline (allineamento in fuzione al texto della flexbox)**
flex-wrap
Consente agli elementi di andare su una nuova linea del contenitore, altrimenti si rimpicciolirebbero
align-content
Definisce allineamento dei flex multilinea(wrap), come le righe si distribuiscono e ci sono **flex-start, flex-end, center, stretch, space-between, space-around**
gap
Consente di impostare uno spazio tra ogni elemento (non è un margine)
align-self
Sostituisce align-items solo per elemento specifico
Order
Come default segue ordine di html, può essere definito con order:numero

M2_Week_01:

Git
Control system per salvare checkpoints

Node.js
E' un sistema di runtime di JavaScript, che è nato per funzionare nel browser

Algoritmi
Insieme di steps per risolvere un problema

Tipi di dati
Nel coding tutti i valori sono o **Data type** o **Structural Type**

Tipizzazione
Nei linguaggi fortemente tipizzati una volta definito il tipo di una variabile non può essere cambiato, in quelli **debolmente tipizzati come javascript** il tipo può essere variabile

Tipi primitivi
Sono rappresentati dal loro valore e immutabili e sono:
Number - Javascript a differenza di altri linguaggi hanno valori diversi per interi decimali etc, js no, ci sono valori speciali ovvero **+infinity/-infinity** per gestire operazioni con infinito, e
NaN - ovvero not a number, valore non valido in operazioni numeriche
String - è una sequenza di caratteri delimitati dagli apici, se ci sono solo apici è stringa vuota
Boolean - entità logica vero/falso
Undefined - è un valore che non esiste ancora, variabile senza nulla assegnato
null - valore vuoto, variabile in quel momento non contiene nulla
BigInt - per numeri molto grandi, per dichiararlo mettere n alla fine del numero
Symbol

Structural types
Oggetti - sono raccolte di coppie chiave-valore, o gruppi di variabili che descrivono la stessa entità, tipo:
user = { name: "tizio", age: 30, isScemo:true}
Funzioni - non-data structure ovvero blocco di codice da eseguire
Array - lista di dati

Operatori aritmetici
Normali operatori, possono essere anche utilizzate parentesi algebriche visto che vigono le normali regole di precedenza:

"+"	somma
"+="	concatena il valore delle variabili
"-"	sottrazione
"*"	moltiplicazione
"/"	divisione
"%"	modulo(resto divisione intera)

Operatore +

Ha diverse funzioni:
numeri - esegue addizione
strings - esegue concatenazione (“dio”+”cane”=diocane)
mixed types - converte in strings ed esegue concatenazione (42+”13”=4213)

Analisi del tipo

Si può convertire un numero in una stringa in un tipo number utilizzando **parseInt** per il numero intero e **parseFloat** per decimali

Operatori logici

Valutano condizioni e restituiscono true o false tipo
let piùAlto => 170 (se 171 piùAlto=true)

Operatori di uguaglianza

== variabili con stesso valore (disuguaglianza **!=**)
=== variabili con stesso valore e stesso tipo (disuguaglianza **!==**)

Not

! significa not e serve per negare variabile o condizione tipo
piùAlto = true
let contrario = !piùAlto

Altri operatori

&& and quindi si può utilizzare per definire intervalli come
(a >=10)&&(b<=20)
|| è or quindi per dare diverse alternative **(a===3) || (b===5)**

M2_Week_02:

Ciclo if e if else

Valuta espressione booleana fornita e se è vera lancia il blocco di codice successivo, con else lancia il blocco di codice alternativo se espressione falsa, ad esempio

```
if(tizio===”stronzo”){
alert(“fanculo”)
}else{
alert(“ciao caro”)}
}
```

Operatore ternario

Serve a rendere più compatti cicli if else con solo un else, esempio:
let prezzo = scontato ? 79:99 dove il primo valore dopo il primo valore dopo ? è il true poi cè il false

Loop

Ci sono vari tipi di loop che fanno tutti la stessa cosa in maniere diverse, ovvero ripetere un azione un tot di volte

Ciclo for

E’ il ciclo più importante e più usato, si ripete fino a che una condizione non risulta falsa, la struttura è inizializzazione; condizione; incremento, si usa quando si conosce il numero di iterazioni da eseguire

for (let i=0; i.length<=5; i++){} in questo caso deve ciclare fino a 5
si utilizzano degli operatori per fermare i cicli prima del tempo ovvero **break** e **continue**

for (let i=0; i.length<=5; i++){ in questo caso quando arriva a 3 ferma il ciclo

```
if (i===3){
break}
}
```

mentre

for (let i=0; i.length<=5; i++){ in questo caso quando arriva a 3 lo salta e continua

```
if (i===3){
continue}
}
```

Ciclo for of

Più elegante e serve meno codice di for ma non posso richiamare indice poi perchè non definisco i

```
const cars = [1,2,3];
let text = “”;
for(let x of cars){
text+= x;
}
```

Ciclo while

Esegue le sue istruzioni fino a che una condizione non risulta vera, si usa quando non si sa esattamente quante volte eseguire il loop ad esempio

let n=0, x=0
while(n<3){ questo vuol dire finche n è minore di 3 continua a ciclare
n++;
x+=n;
}

Ciclo do while

Simile a while ma garantisce che il codice venga eseguito almeno una volta

do{
//codice
}while(condizione)

Array

Struttura dati che permette di accedere a valori multipli tramite unico nome, in js sono una variante degli oggetti che ne permette iterazione proprietà

Dato array **let arr= [1,2,3,4,5]** con l'indice numerico ne posso andare a prendere il valore di una posizione specifica **let fist = arr[0]**

0 è la prima posizionedell'array in questo caso il numero 1

gli array possono essere misti e possono contenere anche altri array e oggetti

Ciclare array

Il metodo più comune per ciclare un array è col **for**, quindi

const arr=[1,2,3,4,5]
for (let i=0; i<arr.lenght; i++){}

altrimenti con **for of**

const arr=[1,2,3,4,5]
for (let element of arr){}

Metodi degli array

Azioni possibili con gli array

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

arr.concat(val1,val2)	copia valori dentro array
arr.indexOf(2)	ritorna indice elemento cercato o -1 se indice non c'è
arr.lastIndexOf(1)	ritorna ultimo indice elemento cercato o -1 se indice non c'è
arr.pop()	rimuove ultimo elemento array e lo ritorna
arr.shift()	rimuove primo elemento array e lo ritorna
arr.lenght()	restituisce lunghezza array
arr.push(6,7)	aggiunge alla fine di un array e ritorna la nuova lunghezza
arr.slice(-6,5)	seleziona porzione array con inizio e fine
arr.includes(2)	ritorna true se elemento è nell'array
arr.join(',')	ritorna stringa di elementi array insieme separati da carattere
arr.splice(1,2)	seleziona porzione array con inizio e fine e la rimuove

M2_Week_03:

Switch

Sostituisce blocchi if con molti else utilizzando invece i cases

```
const expr = 'Papayas';
switch(expr) {
case x:
//code
break;
case y:
//code
break;
case z:
//code
break;
default:
//code
}
```

Funzioni

Blocchi di codice riutilizzabile che possiamo chiamare ogni volta che vogliamo, che possono avere da 0 a molti parametri in input (dentro le tonde quindi) e possono restituire un valore ad esempio

```
function mutiply (a,b){          qui definisco la funzione
return a*b;
}
let result = multiply (3,4)      qui la lancio
console.log(result)
```

Arrow Function

Maniera più concisa ed elegante delle function normali, prima di tutto utilizza => anziche scrivere function:

```
const greet = name => 'Ciao, ${name}!';
greet ('tizio')
```

Posso omettere le graffe e return se il corpo della funzione è una singola espressione

```
const sum = (a, b) => a+b;
```

Metodi per stringhe

.toLowerCase()/.toUpperCase()	rende stringa tutta minuscola o maiuscola
.includes(substr)	controlla se substr è nella str principale, true/false
.concat(str)	concatena una str alla principale restituisce nuova
.repeat(num)	concatena str num volte
.replace(str,str2)	sostituisce str con str 2 la prima volta trovata str
.replaceAll(str,str2)	sostituisce str con str 2 la ogni volta trovata str

Dom

Vuol dire **document Object Model** ed è la rappresentazione ad oggetto dello stato della pagina. Per interagire col DOM prima si seleziona un elemento e poi si modifica.
La prima cosa da selezionare è un nodo, queste funzioni restituiranno un oggetto che rappresenta il node selezionato nel DOM.

Selezione

document.getElementById - restituisce singolo oggetto (ID unico)
document.getElementsByClassName - restituisce gamma di oggetti (tutti i nodi che hanno classe CSS className)
getElementsByTagName - restituisce una gamma di oggetti ovvero tutti i nodi del tipo specificato
document.querySelector - restituisce singolo oggetto ovvero il primo node che corrisponde al selettore CSS specifico, cattive prestazioni
document.querySelectorAll - restituisce una gamma di oggetti ovvero il primo node che corrisponde al selettore CSS specifico, cattive prestazioni
node.children - restituisce una gamma di nodi figli ovvero tutti gli elementi dei child di un parent se ce ne sono

Manipolazione

Una volta selezionato l'elemento che rappresenta un nodo nel DOM possiamo modificarlo
let divContent = node.innerText - leggere le loro proprietà, come testo, classi etc
node.style.color - modificare proprietà cambiando valori
node.class.toggle("selected") - richiamare metodi disponibili per togliere elementi classi CSS o aggiungerne

Eseguire il codice

Per eseguire il codice ci sono vari modi ad esempio fare una funzione da poi richiamare con bottoni nel HTML
<input type="button" value="Click me" onClick="myFunction()"/>
o
<button onclick="myFunction()">Click me </button>

Creare elemento

Per prima cosa dobbiamo creare un nuovo elemento, poi può essere personalizzato
let newDiv = document.createElement('div') questo genera un
poi
newDiv.innertext= "Ciao"
newDiv.style.color= "red"
newDiv.id= "new-div"

Collocamento

Una volta creato e stilizzato il nodo però deve essere collocato nel DOM nella pagina altrimenti esisterebbe solo in js quindi
let body = document.querySelector('body')
body.appendChild(newDiv)
appendChild appende il nodo creato come ultimo elemento del parent(in questo caso body)
Un'altra cosa che si può fare è allegare il nuovo elemento al container prima di un altro elemento specificato
containerNode.insertBefore(newNode, subsequentNode)

M2_Week_04:

Input tag

questo tag ha un value dove ci sono le info fornite dall'utente che possono essere mandate ad un server tramite il tag **<form>** o essere lette tramite JavaScript
HTML fornisce diversi tipi per i tag **<input/>** come **button, checkbox, color, date, datetime-local, email, file, hidden**

Eventi

Vengono utilizzati per attivare un blocco di codice quando accade qualcosa ad esempio

click	utente clicca qualcosa
scroll	utente scrolla fino a un certo punto
resize	finestra ridimensionata
keyup/keydown	utente digita qualcosa in un campo di input
mouseover	utente posiziona il cursore su elemento

Ogni evento attivato fornisce un oggetto specifico con info su cosa ha attivato l'evento e come (ad esempio click ha le coordinate del cursore)
Gli input dicono **cosa** vuole l'utente gli eventi **quando** elaborarlo
Un onclick base ha forma **onclick="openMailbox(event)"** dove onclick è l'**event**, openMailbox il **listener** e event **event data**

Aggiungere listener

Per aggiungere un listener basta chiamare la funzione all'interno di onQualcosa ad esempio **onkeyup="functionName(event)"**
Se elemento è creato e aggiunto tramite codice js invece ci sono due alternative equivalenti element.onclick = **functionName**
element.addEventListener("click", functionName)
Dopo la funzione non vanno le () perchè non va eseguita ma solo collegata all'evento

window.onload

Tutto il codice che deve essere eseguito appena la pagina finisce di caricare deve andare in una funzione ed essere collegato a **window.onload**, il resto del codice va in altre funzioni