Laboratorul 2

Linkuri utile

1. Seminarii înregistrate: https://student.nextlab.tech/
2. Cursuri înregistrate: https://www.youtube.com/@manvscode3496
3. Link creat echipe pentru proiect: https://forms.gle/i7P2WTTGuMRGBxrN9
4. Exemple de la cursul de zi și exemple de subiecte de examen: https://github.com/hypothetical-andrei/tw-live-2023.git
5. Link GitHub pentru seminarii: https://github.com/MihaiAdrianLungu

Introducere JavaScript

JavaScript is **high-level programming language**, often **just-in-time compiled** and **multi-paradigm**. It has **curly-bracket syntax**, **dynamic typing** and **first-class functions**.

More than this, JavaScript is **single threaded**, **non-blocking**, **asynchronous**.

Videoclip explicativ: https://youtu.be/DHjqpvDnNGE

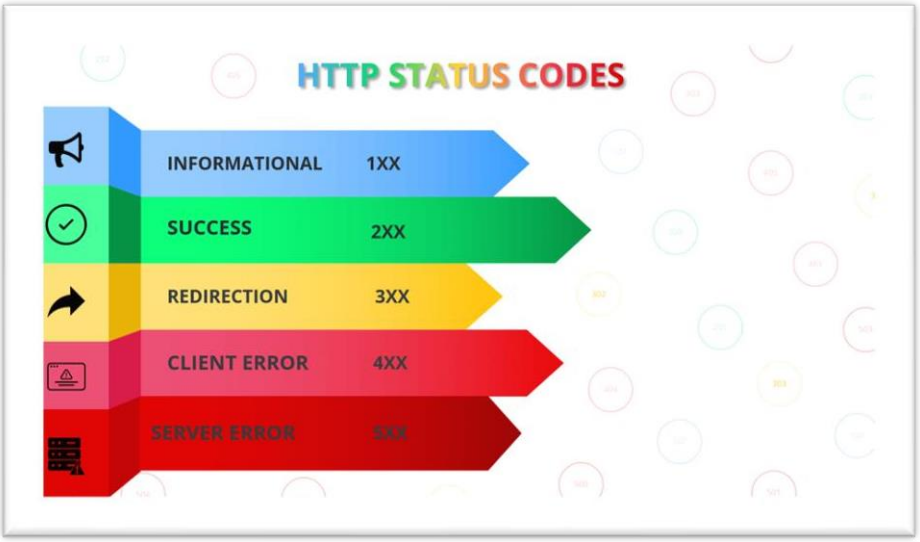
Inspectare Developer Tools

Ghid de utilizare: https://nira.com/chrome-developer-tools/

Deschiderea uneltelor de dezvoltare din Browser => **F12 / Right-click + Inspect;**

1. Tab-ul **Elements** permite inspectarea și manipularea structurii HTML a unei pagini web. Este folosit pentru a vizualiza și modifica codul HTML și CSS al paginii web în timp real.
2. Tab-ul **Console** puteți vedea și executa comenzi JavaScript, puteți depista erori, avertizări și mesaje de la pagină sau de la codul JavaScript al paginii. Este o modalitate utilă pentru a testa și depana codul JavaScript al unei pagini.

3. Tab-ul **Sources** oferă o interfață pentru a explora și a depana codul sursă JavaScript al unei pagini web. Puteți pune puncte de oprire (breakpoints), urmări execuția codului și examina variabilele în timpul rulării.
4. Tab-ul **Network** permite monitorizarea tuturor cererilor HTTP/HTTPS făcute de către pagina web, inclusiv cererile pentru fișiere CSS, JavaScript, imagini și cereri AJAX. Puteți vedea detalii despre timpul de încărcare, dimensiunea fișierelor și starea cererilor.



Pentru mai multe detalii legate de statusuri: https://dev.to/bisrategebriel/http-status-codes-101-6jh

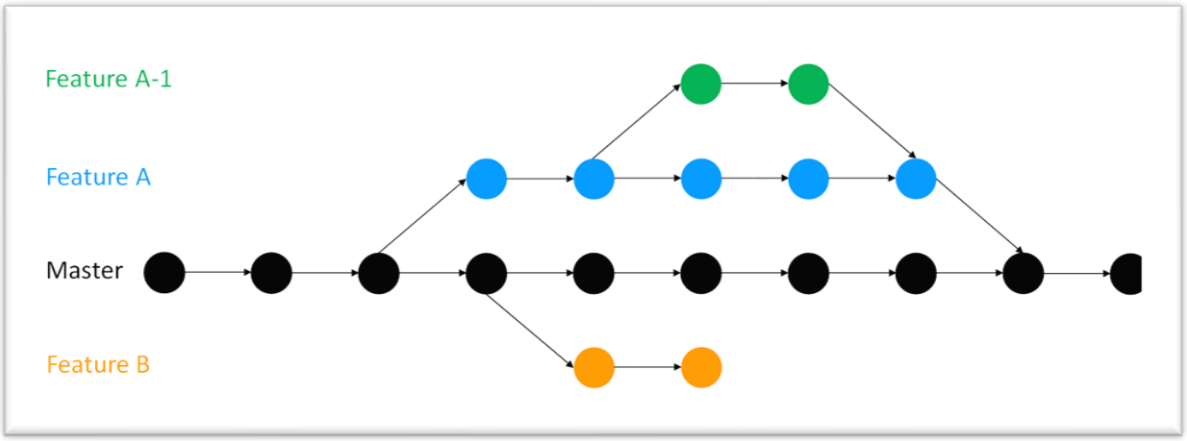
5. Tab-urile **Performance** este folosit pentru a analiza și optimiza performanța unei pagini web. Puteți înregistra profiluri de performanță, vizualiza resursele și cronometra cât timp durează diferite operațiuni.
6. Tab-ul **Application** oferă informații despre starea aplicației web, inclusiv starea cookie-urilor, stocarea locală (LocalStorage și SessionStorage) și IndexDB. Aici puteți gestiona, șterge sau modifica datele de stocare ale paginii web
7. Tab-ul **Security** furnizează informații despre securitatea paginii web, inclusiv detalii despre certificatul SSL (HTTPS) și avertizări de securitate legate de conținutul mixt (Mixed Content).

Pentru mai multe detalii legate de tipurile de tab-uri:https://developer.chrome.com/docs/devtools/

GitHub

GitHub oferă un sistem de control al versiunilor (**Version Control System** - VCS) care permite dezvoltatorilor să urmărească și să gestioneze modificările în codul sursă al proiectului. Acest lucru facilitează colaborarea între diferiți dezvoltatori, deoarece fiecare modificare este înregistrată și documentată.

Repository-urile sunt spații de stocare pentru codul sursă, fișierele și documentația proiectului. Aceste repo-uri pot fi publice (vizibile pentru toată lumea) sau private (accesibile doar pentru membrii autorizați).



Explicarea celor mai importante comenzi de git:

1. **git init**: această comandă inițializează un nou repository Git în **directorul curent** (local).

   După rularea acestei comenzi, se va crea fișierul ascuns **.git** care conține toate informațiile necesare pentru gestionarea și urmărirea modificărilor în codul sursă al proiectului;

   La început, toate fișierle sunt **tracked**. Pentru a putea face anumite foldere **tracked/untracked** trebuie să creăm un fișier **.gitignore** în care o să menționăm ce fișiere nu vrem să salvăm în repository-ul de github (ex: node-modules, .env);

2. **git branch -M main**: redenumeste branch-ul main (numele default al branch-ului pentru repo-uri create folosind linie de comanda este master);
3. **git remote add origin**: realizează legătura dintre repository-ul local și cel remote;
4. **git clone**: este folosită pentru a crea o copie locală a unui repository existent de pe un server remote (de obicei, de pe GitHub). Copia creată conține toate fișierele, istoricul și branch-urile repository-ului original;

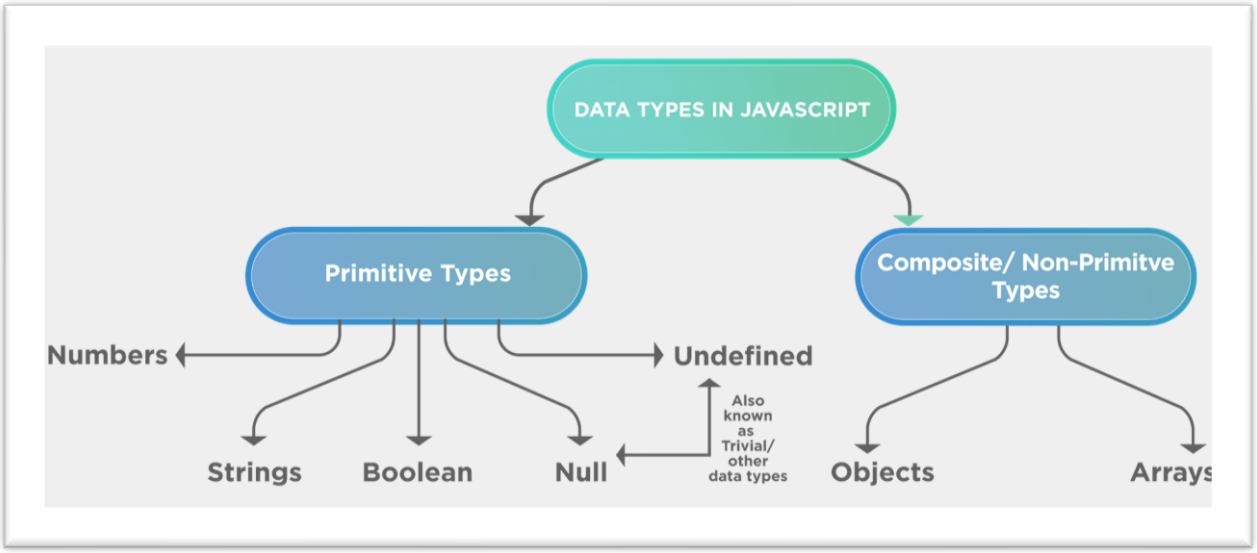
   ```
   git clone https://github.com/utilizator/repo.git
   ```

5. **git add**: această comandă este utilizată pentru a adăuga modificările efectuate în directorul de lucru la zona de stage (staging area). Modificările din zona de stage vor fi pregătite pentru commit. Totodată, putem adăuga fie fișiere specifice (**git add file.js**) sau toate fișierele editate (**git add .**);
6. **git commit**: este folosită pentru a crea o nouă înregistrare în istoricul repository-ului, care conține modificările din zona de stage. La fiecare commit, trebuie să furnizezi și un mesaj care descrie modificările efectuate;

   Eroare potențială de configurare globală (trebuie să se știe al cui este acest commit identificabil prin mesajul introdus) => **git config --global user.email** mail@gmail.com urmat de **git config --global user.name "Name"**;
7. **git config**: este folosită pentru a configura și gestiona setările specifice pentru un repository Git sau pentru utilizatorul Git

8. **git push**: este folosită pentru a încărca (push) modificările din branch-ul tău local către repository-ul remote, astfel încât și ceilalți dezvoltatori să poată vedea și lucra cu aceste modificări;
9. **git log**: afișează lista de commit-uri;
10. **git checkout / git switch**: este utilizată pentru a comuta între branch-uri sau pentru a reveni la o anumită revizuire din istoricul repository-ului. De asemenea, poate fi folosită pentru a crea un branch nou;
11. **git branch**: afișează lista branch-urilor existente în repository;
12. **git merge**: este folosit pentru a integra schimbările dintr-un branch în altul;
13. **git status**: furnizează informații despre starea curentă a repository-ului de Git.
14. **git pull**: este folosit pentru a aduce schimbările din repository-ul remote în repository-ul tău local și pentru a le integra în branch-ul curent. Acest lucru face ca repository-ul local să fie sincronizat cu repository-ul remote, permițându-ți să lucrezi cu cele mai recente modificări din proiect.

Tipuri de date în JavaScript



Tipurile de date primitive din JavaScript sunt tipuri de date care se referă la o singură valoare.

Mai multe detalii legate de tipurile de date în JavaScript: https://www.edureka.co/blog/data-types-in-javascript/

Definirea variabilelor

- **var** - variabile de tip “function-scoped”, ceea ce înseamnă că sunt vizibile numai în cadrul funcției în care sunt declarate. Dacă sunt declarate în afara unei funcții, acestea devin la nivel global.
- **let** - variabile de tip “block-scoped”, înseamnă că sunt limitate la blocul (închis de acolade { }) în care sunt declarate, cum ar fi o buclă sau o instrucțiune if. Nu sunt accesibile în afara blocului respectiv.
- **const** – variabile de tip “block-scoped”, limitate la blocul în care sunt declarate, dar a căror valoare nu poate fi schimbata.

În JavaScript, variabilele declarate cu **var** au un domeniu de aplicare la nivel de funcție, ceea ce înseamnă că ele sunt accesibile în întreaga funcție în care sunt declarate, chiar și înainte de declarația lor efectivă. Această comportare este cunoscută sub numele de "hoisting" și este o caracteristică specifică **var**.

Procesul de hoisting (ridicare) este un comportament specific JavaScript în timpul interpretării codului, prin care declarațiile de variabile și funcții sunt "ridicate" sau mutate în partea de sus a contextului lor de execuție.

Link explicativ: https://www.youtube.com/watch?v=9WIJQDvt4Us&ab_channel=WebDevSimplified

Definirea obiectelor

Un obiect este o colecție de perechi cheie-valoare, în care fiecare cheie este un șir (sau un simbol în JavaScript-ul modern) care acționează ca un identificator, iar fiecare valoare poate fi de orice tip de date, inclusiv alte obiecte, funcții sau valori primitive.

```
const student = {
    name: 'John Doe',
    email: 'johnDoe@gmail.com',
    yearsOld: 20,
}
```

Fiecare proprietate are asignată o cheie (de exemplu, nume) și o valoare (de exemplu, ’John Doe’). Proprietățile pot fi accesate în felul următor:

student.name; // would return 'John Doe'

student['name'];

Definirea array-urilor

Un array este o structură de date care poate stoca mai multe valori într-un moment dat.

Îl scriem astfel:

```
const arr = [1, 2, 3, 4];
```

Fiecare element poate fi accesat folosind indexul, care începe de la 0 (de exemplu, arr[0] este 1). Există câteva proprietăți și funcții care sunt utile în lucrul cu array-uri:

- **length** - proprietate pentru a obține lungimea array-ului
- **push()** - metodă care ne permite să inserăm un element la sfârșitul array-ului
- **pop()** - metodă pentru a elimina și returna ultimul element al unui array
- **shift()** - elimină primul element dintr-un array și returnează acel element eliminat.
- **slice()** - returnează o copie superficială a unei porțiuni a unui array într-un nou obiect array selectat de la început la sfârșit.

Atunci când lucrăm cu array-uri, adesea avem nevoie să iterăm peste elemente și, dacă este cazul, să efectuăm acțiuni asupra lor. Iată câteva metode și structuri de iterație utile:

- **for** - creează o buclă care constă din trei expresii opționale, încadrate între paranteze și separate prin punct și virgulă, urmate de o instrucțiune (de obicei o instrucțiune de bloc) care să fie executată în buclă.
- **forEach()** - este o metodă a array-urilor care vă permite să iterăm prin fiecare element al unui array și să aplicăm o funcție pentru fiecare element.
- **for...of** - execută o buclă care operează pe o secvență de valori preluate dintr-un obiect iterabil
- **for...in** - iterează peste toate proprietățile de șir enumerabile ale unui obiect (ignorând proprietățile cheiate cu simboluri), inclusiv proprietățile enumerate moștenite.

Definirea funcțiilor

O funcție JavaScript este definită cu cuvântul cheie "function", urmat de un nume și de paranteze (). În plus, în JavaScript modern, o funcție poate fi definită și utilizând expresii de funcție sau funcții arrow (arrow function), care au o sintaxă mai concisă.

Numele funcției poate conține litere, cifre, sublinieri și semnele dolarului (aceleași reguli ca și pentru variabile).

Parantezele pot include nume de parametri separate prin virgule: (parametru1, parametru2, …)

Codul care trebuie executat de funcție este plasat în interiorul acoladelor: {}

Mai multe detalii legate de funcții: https://www.w3schools.com/js/js_functions.asp

Mai multe detalii legate de arrow functions: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Exerciţii

Ex 1. Implemetează o funcție arrow care primește ca parametru un array de string și îmi returnează un singur string obținut prin concatenarea string-urilor din array-ul primit ca parametru.

Ex2. Implementează o funcție care returnează numărul de caractere diferite între două string-uri de aceeași lungime primite ca parametri. Dacă string-urile primite nu sunt de aceeași lungime, funcția va returna -1.

Ex3. Implementează o funcție care primește ca și parametri un string și o literă și returnează de câte ori se regăsește litera în respectivul text.

Ex4. Implementează o funcție care primește ca parametrii două array-uri de aceeași lungime și returnează un array cu elementele din cele două surse intercalate. Dacă cele două array-uri nu au aceași lungime, se va returna -1.

Ex5. Implementează o funcție care primește ca și parametru un array de numere și care calculează media aritmetică a numerelor.