

Fondamentaux du Web

HTML, CSS, JavaScript

Lab 7 — DOM et événements

Le premier quiz Jàng Afrig jouable

Licence 1 DUT1-Informatique — 2025–2026

Informations pratiques

Le grand jour. Jusqu'ici, tout votre code s'affichait dans la console. Aujourd'hui, JavaScript va enfin **modifier la page visible** et **réagir aux clics**. À la fin de la séance, Jàng Afrig sera un *vrai quiz jouable* du début à la fin.

Avant de commencer :

- `data.js` (Lab 6, tag v0.6) avec les pays et `choisirAuHasard`.
- `script.js` branché, et vos pages HTML/CSS des Labs 1 à 4.
- Les méthodes de tableaux du Lab 6 (`forEach`, accès par index).

Validation de fin : on lance le quiz, on répond à 10 questions, le score s'affiche, on peut rejouer. Tag v0.7 poussé + rapport déposé.

Objectifs du Lab 7

À la fin, vous saurez :

1. Comprendre le **DOM** : la page vue comme un arbre par JavaScript.
2. **Sélectionner** un élément (`querySelector`, `querySelectorAll`).
3. **Lire et modifier** le contenu d'un élément (`textContent`, `value`).
4. Afficher/masquer des éléments avec `classList`.
5. **Réagir** aux clics avec `addEventListener`.
6. Assembler tout cela pour faire un **quiz Capitales** complet et jouable.

Table des matières

Rappel express des Labs 5 et 6	2
1 Le DOM : la page vue comme un arbre	2
2 Sélectionner un élément	3
3 Lire et modifier le contenu	3
3.1 Le texte d'un élément : <code>textContent</code>	3
3.2 La valeur d'un champ : <code>value</code>	3

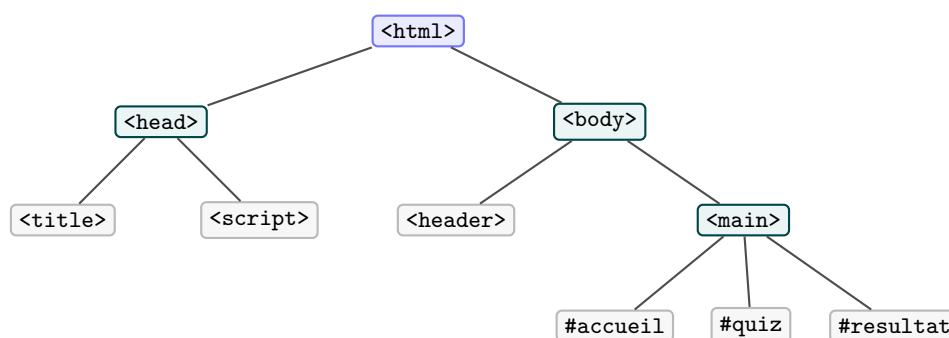
4	Afficher et masquer : <code>classList</code>	3
5	Créer un élément (pour aller plus loin)	4
6	Réagir aux clics : <code>addEventListener</code>	4
7	On assemble : le premier quiz Jàng Afrig	4
7.1	Étape 1 — restructurer <code>index.html</code> en trois écrans	5
7.2	Étape 2 — compléter <code>data.js</code> (10 pays + mélange)	6
7.3	Étape 3 — générer les questions (dans <code>script.js</code>)	6
7.4	Étape 4 — la logique des écrans (<code>script.js</code>)	7
7.5	Étape 5 — brancher les événements	8
8	Pour aller plus loin (facultatif)	9
9	Aide-mémoire du Lab 7	9
10	Sauvegarder mon travail (routine Git)	9
11	Résumé et transition	10

Rappel express des Labs 5 et 6

Vous savez écrire des fonctions, décider avec `if`, parcourir avec `forEach`, et vous avez `data.js` avec un tableau de pays et la fonction `choisirAuHasard`. On va brancher tout ça sur la page.

1 Le DOM : la page vue comme un arbre

Quand le navigateur lit votre HTML, il en construit une représentation en mémoire, organisée comme un **arbre**. On l'appelle le **DOM** (*Document Object Model*). *Analogie* : l'arbre généalogique d'une famille — une balise `<body>` a des « enfants » (`<header>`, `<main>`...), qui ont eux-mêmes des enfants.



Le DOM : un arbre de balises. JavaScript peut s'y promener pour lire ou modifier n'importe quel élément.

À voir dans DevTools

Ouvrez votre page, appuyez sur **F12**, et allez dans l'onglet **Elements** (ou *Éléments*). Vous voyez cet arbre, et vous pouvez déplier chaque branche. C'est la page telle que JavaScript la « voit ».

2 Sélectionner un élément

Avant de modifier un élément, il faut d'abord le **désigner**. L'outil principal pour cela est la fonction `querySelector` : on lui donne un **sélecteur CSS**, exactement comme au CM2.

```
const titre = document.querySelector("h1");           // par balise
const bouton = document.querySelector("#btn-commencer"); // par id (#)
const premiere = document.querySelector(".reponse");   // 1re classe (.)
```

Pour récupérer *tous* les éléments correspondants (par exemple les 4 boutons de réponse), on utilise `querySelectorAll`, qui renvoie une **liste** :

```
const boutons = document.querySelectorAll(".reponse"); // les 4 boutons
console.log(boutons.length); // 4
```

Le pont avec le CSS

Les sélecteurs sont *exactement* ceux du CSS : `h1` (balise), `#identifiant` (id), `.classe` (classe). Ce que vous saviez cibler en CSS, vous savez maintenant le saisir en JavaScript.

3 Lire et modifier le contenu

3.1 Le texte d'un élément : `textContent`

`textContent` permet de **lire** ou de **changer** le texte d'un élément.

```
const titre = document.querySelector("h1");

console.log(titre.textContent); // lit : "Jàng Afrig"
titre.textContent = "Bienvenue !"; // modifie : la page change aussitôt
```

3.2 La valeur d'un champ : `value`

Pour un champ de saisie (`<input>`), ce que l'utilisateur a tapé se trouve dans `.value`.

```
const champ = document.querySelector("#pseudo");
console.log(champ.value); // ce que l'utilisateur a écrit dans le champ
```

Démonstration en direct

On sélectionne le `<h1>` et on tape `titre.textContent = "Test"` : le titre change instantanément à l'écran. C'est la première fois que JavaScript *touche* la page !

4 Afficher et masquer : `classList`

JavaScript peut ajouter ou retirer des **classes CSS** à un élément, en direct, avec `classList` :

- `element.classList.add("cachee")` — ajoute la classe ;
- `element.classList.remove("cachee")` — retire la classe ;
- `element.classList.toggle("cachee")` — l'ajoute si absente, la retire si présente.

L'astuce qui change tout. Si on crée en CSS une classe qui cache un élément :

```
.cachee { display: none; }
```

...alors ajouter/retirer cette classe en JavaScript permet de **montrer ou cacher** n'importe quelle partie de la page. C'est exactement ce qu'on va utiliser pour passer de l'écran d'accueil à l'écran du quiz.

5 Créer un élément (pour aller plus loin)

On peut aussi fabriquer un nouvel élément et l'ajouter à la page :

```
const liste = document.querySelector("#mes-villes");

const item = document.createElement("li"); // créer un <li>
item.textContent = "Dakar";                // lui donner du texte
liste.appendChild(item);                    // l'accrocher dans la liste
```

`createElement` fabrique l'élément, `appendChild` l'accroche comme enfant. On n'en aura pas besoin pour le quiz d'aujourd'hui (nos 4 boutons existent déjà en HTML), mais c'est bon à connaître.

6 Réagir aux clics : `addEventListener`

Voici le cœur de l'interactivité. `addEventListener` permet de dire : « *quand cet événement se produit sur cet élément, exécute cette fonction* ». Analogie : on installe une sonnette ; quand on appuie (l'événement `click`), elle déclenche une action.

```
const bouton = document.querySelector("#btn-commencer");

bouton.addEventListener("click", () => {
  console.log("Le bouton a été cliqué !");
});
```

Les événements les plus courants : `click` (clic), `submit` (envoi de formulaire), `input` (saisie au clavier), `keydown` (touche pressée).

Savoir *quel* élément a été cliqué : `event.target`. La fonction reçoit un objet `event` qui décrit ce qui s'est passé. `event.target` est l'élément précis sur lequel on a cliqué — très utile quand plusieurs boutons partagent le même comportement.

```
const boutons = document.querySelectorAll(".reponse");

boutons.forEach((bouton) => {
  bouton.addEventListener("click", (event) => {
    console.log("Tu as cliqué : " + event.target.textContent);
  });
});
```

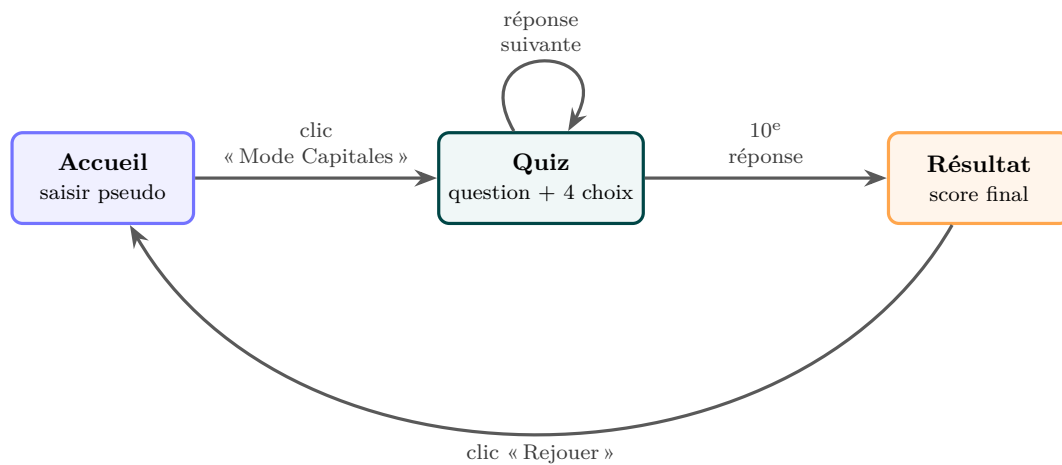
Démonstration en direct

On installe l'écouteur ci-dessus, puis on clique sur chacun des boutons : la console affiche le texte du bouton cliqué. **Notre quiz fonctionnera exactement sur ce principe.**

7 On assemble : le premier quiz Jàng Afrig

Place à la construction. On va transformer Jàng Afrig en un quiz à **trois écrans** : Accueil, Quiz, Résultat. Pour garder le score sans recharger la page, on rassemble ces trois écrans dans

`index.html` sous forme de `<section>`, et on n'en montre qu'une à la fois (avec la classe `cachee`). C'est la technique des applications web modernes : *une seule page, plusieurs vues*.



La navigation du quiz : trois écrans, pilotés par les clics.

7.1 Étape 1 — restructurer `index.html` en trois écrans

Remplacez le contenu du `<body>` de `index.html` par ceci :

```

<body>
  <header><h1>Jàng Afrig</h1></header>

  <main>
    <!-- ÉCRAN 1 : ACCUEIL -->
    <section id="accueil">
      <p>Teste tes connaissances sur les capitales africaines !</p>
      <label for="pseudo">Ton pseudo :</label>
      <input type="text" id="pseudo" placeholder="Ex : Aminata">
      <button id="btn-commencer">Mode Capitales</button>
    </section>

    <!-- ÉCRAN 2 : QUIZ -->
    <section id="quiz" class="cachee">
      <p id="progression">Question 1 / 10</p>
      <h2 id="enonce">Énoncé de la question</h2>
      <div id="reponses">
        <button class="reponse">Réponse A</button>
        <button class="reponse">Réponse B</button>
        <button class="reponse">Réponse C</button>
        <button class="reponse">Réponse D</button>
      </div>
    </section>

    <!-- ÉCRAN 3 : RÉSULTAT -->
    <section id="resultat" class="cachee">
      <h2>Quiz terminé !</h2>
      <p id="score-final">Ton score</p>
      <button id="btn-rejouer">Rejouer</button>
    </section>
  </main>
</body>

```

Puis ajoutez cette ligne à la fin de `style.css` :

```
.cachee { display: none; }
```

7.2 Étape 2 — compléter data.js (10 pays + mélange)

Pour un quiz de 10 questions, étoffez le tableau `pays` à dix pays, et ajoutez une fonction `melanger` (pour brouiller l'ordre des propositions) :

```
const pays = [
  { cca3: "SEN", nom: "Sénégal",      capitale: "Dakar" },
  { cca3: "MLI", nom: "Mali",         capitale: "Bamako" },
  { cca3: "GHA", nom: "Ghana",        capitale: "Accra" },
  { cca3: "KEN", nom: "Kenya",         capitale: "Nairobi" },
  { cca3: "MAR", nom: "Maroc",         capitale: "Rabat" },
  { cca3: "NGA", nom: "Nigéria",      capitale: "Abuja" },
  { cca3: "EGY", nom: "Égypte",       capitale: "Le Caire" },
  { cca3: "CIV", nom: "Côte d'Ivoire", capitale: "Yamoussoukro" },
  { cca3: "TUN", nom: "Tunisie",      capitale: "Tunis" },
  { cca3: "ETH", nom: "Éthiopie",     capitale: "Addis-Abeba" }
];

function choisirAuHasard(tableau) {
  const index = Math.floor(Math.random() * tableau.length);
  return tableau[index];
}

// Mélange un tableau (algorithme classique d'échange)
function melanger(tableau) {
  const copie = [...tableau]; // copie du tableau (à prendre tel quel, CM3)
  for (let i = copie.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    const temp = copie[i];
    copie[i] = copie[j];
    copie[j] = temp;
  }
  return copie;
}
```

7.3 Étape 3 — générer les questions (dans script.js)

Une question = un pays « bonne réponse », sa capitale, et trois autres capitales en distracteurs, le tout mélangé.

```
function genererQuestion() {
  const bonPays = choisirAuHasard(pays);
  const bonneReponse = bonPays.capitale;

  // on part de la bonne réponse, puis on ajoute 3 distracteurs distincts
  const propositions = [bonneReponse];
  while (propositions.length < 4) {
    const autre = choisirAuHasard(pays);
    if (!propositions.includes(autre.capitale)) {
      propositions.push(autre.capitale);
    }
  }

  return {
```

```

    enonce: `Quelle est la capitale du pays : ${bonPays.nom} ?`,
    bonneReponse: bonneReponse,
    propositions: melanger(propositions)
  };
}

function genererQuestions(nombre) {
  const liste = [];
  for (let i = 0; i < nombre; i++) {
    liste.push(genererQuestion());
  }
  return liste;
}

```

Deux petites nouveautés, à prendre telles quelles

`while (condition) { ... }` signifie « *tant que la condition est vraie, répète* » (vous l'avez vu en pseudo-code). `tableau.includes(x)` renvoie `true` si le tableau contient déjà `x` : on s'en sert pour éviter deux fois la même capitale.

7.4 Étape 4 — la logique des écrans (script.js)

```

// --- On saisit les éléments de la page (une seule fois) ---
const ecranAccueil = document.querySelector("#accueil");
const ecranQuiz = document.querySelector("#quiz");
const ecranResultat = document.querySelector("#resultat");

const champPseudo = document.querySelector("#pseudo");
const btnCommencer = document.querySelector("#btn-commencer");
const btnRejouer = document.querySelector("#btn-rejouer");

const elProgression = document.querySelector("#progression");
const elEnonce = document.querySelector("#enonce");
const elScoreFinal = document.querySelector("#score-final");
const boutonsReponse = document.querySelectorAll(".reponse"); // 4 boutons

// --- L'état du jeu ---
let questions = [];
let indexQuestion = 0;
let score = 0;
let pseudo = "";

// --- N'afficher qu'un seul écran ---
function afficherEcran(ecran) {
  ecranAccueil.classList.add("cachee");
  ecranQuiz.classList.add("cachee");
  ecranResultat.classList.add("cachee");
  ecran.classList.remove("cachee");
}

// --- Afficher la question courante ---
function afficherQuestion() {
  const q = questions[indexQuestion];
  elProgression.textContent = `Question ${indexQuestion + 1} / ${questions.length}`;
  elEnonce.textContent = q.enonce;
}

```

```

// remplir les 4 boutons avec les 4 propositions (i = position du bouton)
boutonsReponse.forEach((bouton, i) => {
  bouton.textContent = q.propositions[i];
});
}

// --- Traiter une réponse cliquée ---
function repondre(texteChoisi) {
  const q = questions[indexQuestion];
  if (texteChoisi === q.bonneReponse) {
    score = score + 1;
  }
  indexQuestion = indexQuestion + 1;

  if (indexQuestion < questions.length) {
    afficherQuestion(); // question suivante
  } else {
    terminerQuiz(); // c'était la dernière
  }
}

// --- Fin du quiz ---
function terminerQuiz() {
  elScoreFinal.textContent =
    `Bravo ${pseudo} ! Ton score : ${score} / ${questions.length}`;
  afficherEcran(ecranResultat);
}

// --- Démarrer une partie ---
function demarrerQuiz() {
  pseudo = champPseudo.value;
  if (pseudo === "") {
    pseudo = "Joueur";
  }
  score = 0;
  indexQuestion = 0;
  questions = genererQuestions(10);
  afficherEcran(ecranQuiz);
  afficherQuestion();
}

```

7.5 Étape 5 — brancher les événements

C'est ce qui relie les boutons aux fonctions. À placer **tout en bas** de `script.js` :

```

// Cliquer sur "Mode Capitales" démarre le quiz
btnCommencer.addEventListener("click", demarrerQuiz);

// Cliquer sur une réponse appelle repondre() avec le texte du bouton
boutonsReponse.forEach((bouton) => {
  bouton.addEventListener("click", (event) => {
    repondre(event.target.textContent);
  });
});

// Cliquer sur "Rejouer" revient à l'accueil
btnRejouer.addEventListener("click", () => {
  afficherEcran(ecranAccueil);
});

```

```
});
```

Vérification — le grand moment

Lancez Live Server. Saisissez un pseudo, cliquez sur **Mode Capitales** : l'écran du quiz apparaît avec une question et quatre capitales. Cliquez sur une réponse : la question suivante s'affiche. Après 10 réponses, l'écran Résultat montre votre score. Cliquez sur **Rejouer** : retour à l'accueil. **Vous venez de coder votre première application interactive !**

8 Pour aller plus loin (facultatif)

Deux améliorations à tenter

1. **Compter les erreurs** : ajoutez une variable `erreurs` (avec `let`), incrémentée dans le `else` de `repondre`, et affichez-la dans l'écran Résultat.
2. **Un petit retour visuel** : ajoutez un élément `<p id="feedback"></p>` dans l'écran Quiz, et faites afficher « Bonne réponse ! » ou « Raté » à chaque clic, via `textContent`.

9 Aide-mémoire du Lab 7

Besoin	Code
Sélectionner un élément	<code>document.querySelector("#id")</code>
Sélectionner plusieurs	<code>document.querySelectorAll(".classe")</code>
Lire / changer un texte	<code>element.textContent</code>
Lire un champ de saisie	<code>champ.value</code>
Cacher / montrer	<code>element.classList.add/remove("cachee")</code>
Réagir à un clic	<code>element.addEventListener("click", fonction)</code>
Élément cliqué	<code>event.target</code>
Créer / accrocher un élément	<code>createElement(...)</code> / <code>appendChild(...)</code>

10 Sauvegarder mon travail (routine Git)

Routine Git de fin de Lab — tag v0.7

```
git status
git add .
git commit -m "Lab 7 : premier quiz fonctionnel (DOM + evenements)"
git tag v0.7
git push
git push --tags
```

Vérifiez sur GitHub que le commit *et* le tag v0.7 apparaissent.

Rapport à produire

Déposez `rapport-lab-7-NOM-Prenom.pdf` dans `rapports/`, avec quatre sections :

1. **Ce que j'ai fait** : le quiz à trois écrans, la génération de questions, les événements.
2. **Difficultés rencontrées** : vos erreurs (messages copiés). Un quiz a beaucoup de pièces : c'est normal d'avoir des bugs.
3. **Comment je les ai résolues**.
4. **Lien vers mon repo** avec le tag `v0.7`.

11 Résumé et transition

C'est une étape majeure : Jàng Afrig est désormais une **application interactive**. Vous savez sélectionner des éléments, lire et modifier la page, et réagir aux clics. Vous avez relié HTML, CSS et JavaScript en un tout cohérent.

Mais il y a une limite : le quiz ne connaît que **dix pays**, écrits à la main dans `data.js`. L'Afrique en compte **54** ! Au **CM4**, on apprendra comment une page web va chercher de *vraies* données en ligne, en parlant à un service distant (une *API*). Puis, au Lab 8, on remplacera nos dix pays par les 54 pays réels, récupérés automatiquement.

Le quiz est jouable. Bientôt, il connaîtra toute l'Afrique.

Dr. El Hadji Bassirou TOURÉ — UCAD
Licence 1 DUT1-Informatique — Fondamentaux du Web — 2025–2026