NOTE TRANSVERSALE — NEUTRALITÉ DES OUTILS (révision pédagogique, octobre 2026)

Ce manuel enseigne des compétences transférables et non l'obligation d'utiliser une marque. ChatGPT, Codex, GitHub et Netlify sont des exemples de parcours, jamais des prérequis exclusifs. L'apprenant peut utiliser un autre assistant IA, un éditeur local, une autre plateforme de dépôt ou un autre hébergeur, ou travailler hors ligne lorsque l'exercice le permet. Les solutions retenues doivent permettre d'atteindre les mêmes objectifs observables : comprendre, produire, comparer les versions, tester, documenter et, lorsque nécessaire, publier. Aucun abonnement payant ne doit être imposé pour réussir les apprentissages fondamentaux. Vérifier à chaque séance les quotas, les comptes requis, les conditions d'utilisation, la confidentialité et l'accessibilité. Une activité interactive facultative reste distincte des évaluations de compétences.

CONCEPTION ET CRÉATION NUMÉRIQUE ASSISTÉES PAR INTELLIGENCE ARTIFICIELLE

Manuel pédagogique — Débutants complets

Édition publique V1.0 • 12 modules • 120 heures • Novembre 2026

De l’idée à la publication : assistants IA au choix, HTML, CSS, JavaScript, gestion de versions et solutions de publication adaptées.

SOMMAIRE — PARCOURS DE FORMATION

Partie I — Architecture pédagogique

Module 01 — Culture numérique et conception (8 h)

Module 02 — Concevoir avec un assistant IA (10 h)

Module 03 — HTML5 (12 h)

Module 04 — CSS3 et responsive design (14 h)

Module 05 — JavaScript et interactivité (12 h)

Module 06 — Gestion de versions : Git, GitHub et alternatives (10 h)

Module 07 — Développement assisté par IA : Codex et alternatives (12 h)

Module 08 — Publication web : Netlify et alternatives (8 h)

Module 09 — Plugins, connecteurs et écosystème IA (8 h)

Module 10 — UX, accessibilité, SEO et qualité web (10 h)

Module 11 — Création numérique interactive (8 h)

Module 12 — Projet intégrateur et soutenance (8 h)

Total : 120 heures. Utiliser le sommaire cliquable et les signets du PDF pour accéder aux chapitres.

ÉDITION PUBLIQUE — GUIDE DE NAVIGATION

Ce manuel accompagne les douze modules du site Atelier numérique IA. Dans le PDF, le sommaire cliquable et les signets latéraux permettent de rejoindre les principales parties.

Les activités interactives et laboratoires proposés sur le site sont facultatifs et non certifiants ; les exercices écrits et leurs corrigés restent dans ce manuel.

Ce document est une édition publique autonome, distincte du document maître éditorial privé. Les liens externes cités dans les exercices restent facultatifs et peuvent évoluer.

Avant-propos

Ce manuel accompagne un apprenant qui sait utiliser un ordinateur mais n'a jamais programmé. Il privilégie la compréhension, l'expérimentation et la vérification. L'intelligence artificielle est un outil d'assistance, jamais une garantie de justesse. Les études de cas s'inspirent de projets créatifs réels, notamment des Récits Vivants, mais tous les exercices sont reproductibles indépendamment.

Ce parcours est une formation pédagogique et ne constitue ni un diplôme, ni une certification officiellement reconnue.

PARTIE I — Architecture pédagogique

1. Public, prérequis et moyens

- Public : adultes ou étudiants débutants en développement web.
- Prérequis : savoir utiliser un navigateur, créer un dossier, enregistrer et retrouver un fichier.
- Matériel : ordinateur Windows, macOS ou Linux, navigateur récent, éditeur de texte/code ; connexion Internet pour les services en ligne.
- Comptes utiles au fil du parcours : ChatGPT, GitHub et Netlify, ou solutions équivalentes selon les exercices ; vérifier les conditions d'accès, limites et autorisations à la date du cours. L'évaluation porte sur les réalisations et leur vérification, non sur le choix d'une marque.
- Principe : privilégier les exercices locaux sans compte au départ ; aucune offre payante n'est exigée pour comprendre les notions.

2. Compétences terminales

- C1 — Analyser un besoin et produire un cahier des charges simple.
- C2 — Expliquer navigateur, serveur, URL, fichiers et hébergement.
- C3 — Écrire et modifier une page HTML sémantique.
- C4 — Mettre en forme une interface CSS responsive.

- C5 — Programmer des interactions JavaScript simples.
- C6 — Formuler des demandes précises à une IA et vérifier ses réponses.
- C7 — Suivre les versions d'un projet avec Git ou une méthode équivalente, comparer les changements et restaurer une version.
- C8 — Publier et contrôler un site avec un hébergeur adapté, tel que Netlify ou une alternative.
- C9 — Appliquer des vérifications de sécurité, accessibilité, SEO et performance.
- C10 — Documenter et présenter un projet numérique autonome.

3. Progression des 12 modules

N°	Module	Heures	Production attendue
01	Culture numérique et conception	8	Schéma du Web, besoins, arborescence, vocabulaire
02	Assistants IA et demandes structurées	10	Prompts, limites, vérification, confidentialité
03	HTML5	12	Document, balises, liens, images, sémantique
04	CSS et responsive	14	Cascade, box model, Flexbox, Grid, media queries
05	JavaScript	12	Variables, fonctions, événements, DOM
06	Git et plateformes de dépôt	10	Dépôts, commits, branches, pull requests
07	Assistants IA et code assisté	12	Contexte, modification ciblée, diff, tests
08	Publication web	8	Déploiement, domaine, HTTPS, retour arrière
09	Plugins et connecteurs	8	Permissions, accès, intégrations, limites
10	UX, accessibilité et	10	Parcours, WCAG,

	SEO		métadonnées, performance
11	Création numérique interactive	8	Narration, état, mémoire, éthique
12	Projet intégrateur	8	Concevoir, réaliser, tester, publier, soutenir

Total : 120 heures. Bloc A : modules 1–5 (56 h) ; bloc B : modules 6–9 (38 h) ; bloc C : modules 10–12 (26 h).

4. Méthode pédagogique et évaluations

Chaque module contient : objectifs observables ; prérequis ; cours et lexique ; démonstration ; travaux pratiques ; erreurs fréquentes ; autoévaluation ; corrigés commentés ; synthèse et ressources. Le formateur observe autant la démarche que le résultat.

Évaluation finale proposée : conception 20 % ; HTML/CSS/JavaScript 25 % ; usage critique de l'IA 20 % ; publication et tests 20 % ; documentation et soutenance 15 %. Les critères sont explicités avant les travaux.

5. Projet fil rouge

Créer « Mon espace de découverte », un mini-site personnel non commercial avec une page d'accueil, une présentation, trois cartes de contenu, une interaction JavaScript, un affichage mobile et une publication en ligne. Chaque module apporte une amélioration contrôlée ; les versions sont conservées.

PARTIE II — MODULE 01 : Culture numérique et conception

Durée : 8 heures • Niveau : découverte • Livrable : fiche de conception et première page locale.

1. Objectifs du chapitre

À la fin du module, l'apprenant saura distinguer Internet et Web, expliquer le rôle d'un navigateur et d'un hébergeur, identifier HTML/CSS/JavaScript, organiser les fichiers d'un mini-site, rédiger une fiche de besoin et ouvrir sa première page locale.

2. Mise en situation : de l'idée à l'écran

Vous souhaitez présenter trois créations sur une page consultable depuis un téléphone. Vous avez une idée, mais pas encore de site. Il faut transformer cette intention en éléments observables : quels contenus ? pour qui ? quelle navigation ? quelles tailles d'écran ? quel résultat attendu ?

Une création numérique n'est pas uniquement du code : c'est une articulation entre besoin, contenu, structure, apparence, comportement, vérification et publication.

3. Comprendre Internet et le Web

Internet est l'infrastructure mondiale qui relie des réseaux et des machines. Le Web est un service fonctionnant sur Internet : on consulte des ressources au moyen d'adresses (URL), souvent grâce

aux protocoles HTTP et HTTPS. Un navigateur comme Firefox, Chrome ou Safari interprète les ressources reçues pour afficher une page.

Exemple : vous saisissez https://exemple.fr. Le navigateur résout le nom de domaine, contacte un serveur, demande une ressource et affiche le document reçu. Les détails techniques seront approfondis ultérieurement.

4. Les six mots essentiels

Navigateur — Logiciel qui interprète et affiche les pages web.

URL — Adresse d'une ressource, par exemple https://exemple.fr/page.html.

Nom de domaine — Nom lisible qui désigne un service sur Internet.

Serveur — Système qui répond aux demandes du navigateur.

Hébergement — Service qui conserve et rend accessibles les fichiers d'un site.

HTTPS — Protocole de communication web chiffré ; il ne garantit pas à lui seul la fiabilité du contenu.

5. Les trois langages du navigateur

HTML décrit le contenu et sa structure : titres, paragraphes, liens et images. CSS détermine la présentation : couleurs, espacements, alignements et adaptation aux écrans. JavaScript permet les comportements : clics, affichage conditionnel, calculs et interactions.

Image mentale : HTML est le plan et les pièces d'une maison ; CSS, leur aménagement visuel ; JavaScript, certains mécanismes qui réagissent aux actions. Cette analogie aide à démarrer mais ne remplace pas les définitions techniques.

6. Fichiers, dossiers et extensions

Un site simple peut fonctionner avec trois fichiers : index.html, style.css et script.js. Les extensions indiquent le type de contenu. Les chemins relatifs relient des fichiers dans un dossier ; les noms sont sensibles à la casse sur de nombreux hébergements.

```
mon-premier-site/
  index.html
  style.css
  script.js
  images/
    illustration.jpg
```

Conseil : utiliser des noms courts, sans espaces ni accents pour les fichiers destinés au Web ; conserver une copie du projet avant toute modification importante.

7. Concevoir avant de coder

La fiche de conception répond à sept questions : (1) À qui s'adresse la page ? (2) Quel problème résout-elle ? (3) Quelles informations sont indispensables ? (4) Quelle action le visiteur peut-il effectuer ? (5) Quels appareils sont visés ? (6) Comment vérifier que cela fonctionne ? (7) Quelles données personnelles sont réellement nécessaires ?

Exemple de réponse : « Présenter trois créations à des visiteurs curieux, avec un titre clair, une brève description et trois liens ; le contenu doit rester lisible sur mobile et les liens doivent ouvrir la bonne page. Aucune donnée personnelle n'est collectée. »

8. Travaux pratiques guidés

TP 1 — Observer un site (45 min)

- Ouvrir deux sites différents sur ordinateur ou mobile.
- Repérer le titre, la navigation, les contenus, les liens et le pied de page.
- Réduire la largeur de la fenêtre : que devient la navigation ?
- Noter trois choix qui améliorent la lisibilité et un choix qui la complique.
- Livrable : une grille d'observation d'une page.

TP 2 — Créer la première page (60 min)

1. Créer un dossier « mon-premier-site ». 2. Ouvrir un éditeur de code ou de texte brut. 3. Coller le code suivant. 4. Enregistrer sous index.html (et non index.html.txt). 5. Double-cliquer sur le fichier pour l'ouvrir dans un navigateur.

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Mon premier site</title>
</head>
<body>
  <header><h1>Bienvenue dans mon univers</h1></header>
  <main>
    <p>Voici ma première page web.</p>
    <a href="https://developer.mozilla.org/fr/">Découvrir le Web</a>
  </main>
</body>
</html>
```

Observer : le titre apparaît-il ? Le paragraphe est-il lisible ? Le lien fonctionne-t-il ? Modifier le texte, enregistrer et actualiser le navigateur. Comprendre que la page locale ne nécessite pas encore d'hébergement.

TP 3 — Rédiger un cahier des charges (45 min)

Sur une page, décrire votre futur mini-site : public, objectif, trois rubriques, trois contenus, une interaction envisagée, contraintes mobiles, critères de réussite et éventuels risques. Dessiner un croquis rapide de l'écran mobile.

TP 4 — Utiliser un assistant IA au choix sans déléguer la compréhension (30 min)

Exemple de demande : « Je débute totalement en HTML. Voici ma fiche de conception [coller le texte non confidentiel]. Pose-moi trois questions pour vérifier qu'elle est claire, puis propose une arborescence simple. Explique chaque terme technique sans générer de code pour l'instant. »

Comparer la réponse à votre intention. Rejeter les propositions inutiles. Ne pas partager de mot de passe, de clé API, de données de clients ou de document confidentiel. Un résultat convaincant n'est pas automatiquement exact.

9. Erreurs fréquentes et diagnostic

• Le fichier s'appelle index.html.txt : afficher les extensions et le renommer.

• La page ne s'actualise pas : enregistrer le fichier puis recharger l'onglet.

• Les accents s'affichent mal : vérifier l'encodage UTF-8 et la balise charset.

• Un lien ne fonctionne pas : contrôler son adresse et la connexion réseau.

• Un chemin d'image échoue : vérifier dossier, extension, orthographe et majuscules.

10. Autoévaluation (sans aide)

1. Quelle différence existe entre Internet et le Web ?

2. Quel est le rôle d'un navigateur ?

3. À quoi sert une URL ?

4. Quel langage structure une page ?

5. Quel langage gère l'apparence ?

6. Quel langage gère les interactions ?

7. Pourquoi nommer précisément les fichiers ?

8. Quelle différence entre une page locale et une page publiée ?

9. Citer deux critères de réussite d'un site mobile.

10. Pourquoi vérifier une réponse générée par une IA ?

11. Corrigé synthétique

1. Internet est le réseau ; le Web est un service accessible par ce réseau.

2. Il interprète les ressources et affiche les pages.

3. Elle identifie l'emplacement d'une ressource.

4. HTML.

5. CSS.

6. JavaScript.

7. Pour éviter les erreurs de chemin et faciliter la maintenance.

8. Locale : sur l'ordinateur ; publiée : accessible via un serveur et une adresse publique.

9. Lisibilité, navigation utilisable, absence de débordement horizontal, liens accessibles.

10. Elle peut être erronée, incomplète ou inadaptée au besoin.

12. Critères de validation du module

- Le fichier index.html s’ouvre dans le navigateur et présente un titre, un paragraphe et un lien.
- L’apprenant explique oralement les rôles de HTML, CSS et JavaScript.
- La fiche de conception identifie un public, un objectif, des contenus et deux critères de test.
- L’apprenant sait modifier, enregistrer et vérifier une page sans intervention de l’IA.

13. Ressources et transition

Documentation de référence : MDN Web Docs (developer.mozilla.org/fr/), GitHub Docs (docs.github.com), documentation Netlify (docs.netlify.com), documentation OpenAI (help.openai.com et developers.openai.com). Les interfaces et conditions d’accès des outils peuvent évoluer : vérifier les documentations officielles lors des séances.

Au module 2, nous apprendrons à dialoguer méthodiquement avec un assistant IA au choix : cadrer une demande, demander des explications progressives, faire contrôler un raisonnement et repérer les réponses incertaines.

PARTIE III — MODULE 02 : Apprendre à concevoir avec un assistant IA

Durée indicative : 10 heures | Niveau : débutant complet | Prérequis : module 01. Objectif général : savoir conduire une démarche de conception assistée par IA sans perdre sa capacité à expliquer, reproduire et vérifier.

1. Compétences et critères de réussite

- Expliquer en termes simples ce qu'est un modèle conversationnel et pourquoi ses réponses peuvent être fausses.
- Rédiger une demande précisant objectif, contexte, contraintes, format et critères de réussite.
- Décomposer un projet numérique en étapes compréhensibles et vérifiables.
- Fournir un fichier pertinent, protéger les données personnelles et contrôler l'analyse obtenue.
- Repérer une réponse incertaine, demander des preuves, recouper et tester.
- Reproduire une tâche élémentaire sans IA et tenir un journal des décisions.

2. Séquence pédagogique et temps

- Séance A (1 h 30) : fonctionnement et limites des réponses IA.
- Séance B (2 h) : construire et améliorer un prompt.
- Séance C (1 h 30) : dialogue de conception et découpage en tâches.
- Séance D (1 h 30) : travailler avec des fichiers et des extraits.
- Séance E (1 h 30) : erreurs, vérification et sécurité.
- Séance F (2 h) : projet fil rouge, évaluation et correction.

3. Comprendre le rôle de ChatGPT

ChatGPT est un assistant conversationnel capable de proposer des explications, des plans, des textes et du code. Il génère des réponses à partir du contexte fourni et de ses capacités ; il ne garantit ni la vérité ni le bon fonctionnement de chaque proposition. Ses outils et accès externes dépendent de la configuration disponible. ChatGPT sert ici d'exemple pédagogique : un autre assistant conversationnel adapté peut être utilisé si les objectifs, les vérifications et les livrables sont conservés.

Différence fondamentale : une réponse plausible n'est pas une preuve. Pour le code, le test réel dans un navigateur reste indispensable. Pour un fait récent, il faut vérifier une source actuelle et sa date.

4. Anatomie d'une bonne instruction

Une instruction utile précise cinq éléments : (1) objectif ; (2) contexte et public ; (3) contraintes ; (4) résultat attendu ; (5) mode de vérification. Le mot « prompt » désigne simplement la demande transmise au modèle.

Exemple faible
Fais-moi un site moderne.

Exemple exploitable
Je débute en HTML. Je veux une page de présentation d'un atelier de création numérique.
Public : adultes débutants.
Contraintes : un seul fichier index.html, sans bibliothèque externe ; lisible sur téléphone.
Résultat : propose d'abord un plan de la page et explique chaque section sans écrire le code.
Vérification : donne une liste de 5 points que je pourrai contrôler moi-même.

Pourquoi cela fonctionne mieux : le résultat est borné, le niveau de langage est adapté et l'apprenant garde une étape de validation avant la production.

5. Dialoguer par étapes plutôt que tout demander d'un coup
1. Étape 1 — Définir le besoin : à qui s'adresse le projet, quel usage et quel contenu ?
2. Étape 2 — Demander deux options et leurs compromis, puis choisir.
3. Étape 3 — Produire une petite partie du résultat.
4. Étape 4 — Expliquer les décisions et les termes inconnus.
5. Étape 5 — Tester et signaler les écarts observés.
6. Étape 6 — Corriger uniquement le problème identifié ; conserver une version de référence.

Règle pédagogique : avant de demander la suite, reformuler en une phrase ce qui vient d'être réalisé. Si cette reformulation est impossible, revenir à l'explication.

6. Concevoir un mini-projet : fiche de cadrage
Projet fil rouge : page de présentation d'une expérience numérique librement exploratoire. La page comportera un titre, une courte introduction, trois sections et un lien de contact fictif. Aucun développement n'est requis pendant cette séance.

- Besoin : quel problème ou désir le site sert-il ?
- Public : qui le consultera et sur quel appareil ?
- Contenus : quels textes, images et liens sont nécessaires ?
- Contraintes : temps, accessibilité, confidentialité, absence de paiement ?
- Critères d'acceptation : comment constater que le résultat répond au besoin ?

7. Utiliser des fichiers de manière responsable
Selon les fonctions disponibles, on peut joindre un document texte, un PDF, une image ou un fichier de code pour demander un résumé, une explication ou une comparaison. Le modèle peut mal lire un document, ignorer un détail ou confondre des versions. Toujours vérifier le nom, la version, les extraits cités et les modifications réellement effectuées.

1. Préparer une copie de travail ; garder l'original intact.
2. Retirer mots de passe, clés API, identifiants privés et données personnelles non nécessaires.
3. Nommer clairement les fichiers : index_v1.html, cahier_charges_v1.docx.
4. Préciser la tâche : expliquer, comparer ou proposer une modification ciblée.

5. Demander les passages exacts justifiant les conclusions.
6. Ouvrir et tester le fichier de sortie avant de remplacer une version publiée.

8. Identifier les erreurs et les incertitudes

Invention de faits
L'IA peut affirmer une information non vérifiée : demander la source et la contrôler.

Code incorrect
Une syntaxe peut sembler convaincante et pourtant échouer : tester le fichier.

Oubli d'une contrainte
Comparer la sortie avec la demande initiale point par point.

Modification non souhaitée
Examiner les différences avant/après et garder une sauvegarde.

Source périmée
Vérifier la date de publication et la documentation officielle actuelle.

Confiance excessive
Exprimer les incertitudes ; ne pas confondre assurance du ton et fiabilité.

9. Méthode de vérification en quatre niveaux
1. Niveau 1 — Compréhension : puis-je expliquer la réponse avec mes mots ?
2. Niveau 2 — Cohérence : répond-elle à toutes les contraintes de la consigne ?
3. Niveau 3 — Preuve : quelle documentation, quel extrait ou quel test la confirme ?
4. Niveau 4 — Reproduction : puis-je obtenir le même résultat et diagnostiquer une erreur simple sans l'IA ?

Exemple : si ChatGPT annonce que la page est responsive, ouvrir le fichier dans le navigateur, réduire la largeur de la fenêtre et vérifier l'absence de débordement horizontal. Une affirmation ne remplace pas l'observation.

10. Sécurité, droit et responsabilité
Ne jamais copier de secrets d'accès dans une conversation. Vérifier les droits d'utilisation des images et des textes ; distinguer ses propres créations, les contenus sous licence et les ressources dont les droits sont inconnus. Ne pas publier des données personnelles sans base légitime. La responsabilité de la publication reste humaine.

11. Travaux pratiques guidés

TP 1 — Améliorer une demande (30 min)
Consigne : transformer « Fais un beau site » en demande comportant les cinq éléments du §4. Échanger les demandes entre apprenants ; relever les éléments absents. Livrable : prompt initial, prompt corrigé et justification en cinq lignes.

TP 2 — Conception sans code (45 min)
Consigne : demander un plan de page pour une petite exposition numérique. Comparer deux structures, choisir et rédiger un cahier des charges de dix lignes. Livrable : plan retenu et trois critères de réussite observables.

TP 3 — Analyser un fichier (45 min)
Consigne : réutiliser le fichier index.html du TP 2 du module 1, qui contient déjà les balises h1, p et a et aucune donnée personnelle. Fournir ce fichier à l'assistant si cette fonction est disponible ; sinon copier son code dans la conversation. Demander l'explication des trois balises, puis ouvrir le fichier dans un éditeur pour les retrouver. Livrable : tableau « élément / rôle / preuve dans le fichier ». Aucun fichier externe supplémentaire n'est nécessaire.

TP 4 — Mettre une réponse à l'épreuve (45 min)
Consigne : repartir du fichier index.html du TP 2 du module 1. Dans une copie de travail, remplacer volontairement l'adresse du lien « Découvrir le Web » par href="documentation-inexistante.html". Ouvrir la page dans un navigateur, cliquer sur ce lien et constater l'échec. Demander à l'IA une méthode de diagnostic, puis rétablir href="https://developer.mozilla.org/fr/" et retester (connexion Internet nécessaire pour le lien externe). Livrable : journal de test avec problème, hypothèse, action et constat. Conserver l'original intact.

TP 5 — Concevoir en autonomie (55 min)
Consigne : sans IA pendant 20 minutes, rédiger les rubriques d'une page « Mon atelier numérique ». Utiliser ensuite un assistant IA disponible pour demander une critique, sans lui laisser réécrire le projet. Terminer par une version personnelle et argumentée. Livrable : deux versions et note réflexive.

12. Autoévaluation — 10 questions
1. Que signifie « prompt » ?

2. Pourquoi une réponse fluide peut-elle être fausse ?

3. Quels sont les cinq éléments d'une instruction exploitable ?

4. Pourquoi demander un plan avant le code ?

5. Quelle différence entre une proposition et une validation ?

6. Que faut-il retirer d'un fichier avant de le partager ?

7. Comment contrôler une affirmation sur le responsive ?

8. À quoi sert un journal de test ?

9. Que faire si une correction casse une autre fonction ?

10. Comment montrer que l'on n'est pas dépendant de l'IA ?

13. Corrigé et repères attendus
1. La demande adressée à l'assistant.

2. La génération peut produire des erreurs, des omissions ou des inventions.

3. Objectif, contexte, contraintes, format de sortie, vérification.

4. Pour valider la structure avant d'investir dans la réalisation.

5. La proposition est une hypothèse ; la validation repose sur des critères et tests.

6. Secrets, identifiants et données personnelles inutiles.

7. Ouvrir le site sur plusieurs largeurs et observer les débordements.

8. Tracer l'erreur, les actions effectuées et le résultat observable.

9. Revenir à la version stable, isoler le changement et retester.

10. Expliquer, refaire et dépanner une tâche simple sans assistant.

14. Grille de validation du module

Barème indicatif sur 20 : prompt complet (4), cahier des charges cohérent (4), analyse de fichier (3), protocole de vérification (4), autonomie et justification (5). Validation proposée : 12/20 minimum, avec démonstration obligatoire d'un test et d'une explication sans IA. En cas de difficulté, reprendre le TP concerné.

15. Étude de cas : Récits Vivants

Situation : une page de découverte contient un lien ou un QR code vers une autre page. L'assistant peut suggérer une correction, mais l'auteur doit vérifier l'adresse cible, le chemin relatif, le comportement sur mobile et l'absence de régression. Cette situation illustre la différence entre déléguer une tâche et déléguer son jugement.

16. Transition vers le module 03 : HTML5

À la fin de ce module, l'apprenant possède un cahier des charges simple, un plan de page et des critères de test. Le module suivant transformera cette conception en structure HTML : titres, paragraphes, liens, images et organisation sémantique.

Note de mise à jour : les interfaces, outils connectés et limites d'usage de ChatGPT évoluent. Les captures d'écran et procédures seront datées lors de leur ajout ; les compétences de conception et de vérification restent la référence.

MODULE 3 — HTML5 : comprendre et construire une page web

Durée : 12 heures | Public : débutants complets | Édition pédagogique V2.2 — relecture en cours

Objectifs et compétences attendues

- Expliquer le rôle de HTML, du navigateur et du fichier index.html.
- Créer et enregistrer une page HTML5 avec un éditeur de texte.
- Reconnaître les balises de structure, de titre, de paragraphe, de lien, d'image et de liste.
- Vérifier les chemins relatifs, l'accessibilité élémentaire et les erreurs de syntaxe.
- Modifier puis expliquer une page sans solliciter l'IA.

Progression des 12 heures

- Séquence 1 (1 h 30) — Web, fichiers, éditeur et navigateur
- Séquence 2 (2 h) — Structure minimale d'un document HTML5
- Séquence 3 (2 h) — Texte, titres, listes et sémantique
- Séquence 4 (2 h) — Liens, images, chemins et accessibilité
- Séquence 5 (2 h) — Construction du mini-site de présentation
- Séquence 6 (2 h 30) — Tests, débogage, évaluation et autonomie

1. HTML : ce que c'est et ce que ce n'est pas

HTML signifie HyperText Markup Language : c'est un langage de balisage qui décrit la structure et le sens du contenu. CSS gère la présentation visuelle ; JavaScript peut ajouter des comportements. Le navigateur interprète le fichier HTML et construit la page affichée. HTML ne nécessite ni serveur ni abonnement pour un premier essai : un fichier local suffit.

2. Préparer l'environnement

1. Créer un dossier nommé mon-premier-site.
2. Ouvrir un éditeur de texte brut (par exemple VS Code ou un éditeur simple).
3. Créer un fichier nommé exactement index.html, et non index.html.txt.
4. Enregistrer en UTF-8, puis ouvrir le fichier dans un navigateur.
5. Après chaque changement, enregistrer et actualiser la page dans le navigateur.

Attention : le nom index.html est une convention courante pour la page d'entrée d'un site. Une page locale peut fonctionner sans connexion Internet ; les ressources distantes, elles, peuvent nécessiter une connexion.

3. Premier document HTML5, ligne par ligne

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ma première page</title>
</head>
<body>
  <h1>Bienvenue sur ma page</h1>
  <p>Je découvre la création numérique.</p>
</body>
</html>
```

<!doctype html> — Indique au navigateur l'utilisation du mode HTML moderne.

<html lang="fr"> — Déclare la langue principale du document, utile notamment aux technologies d'assistance.

<head> — Contient les informations sur la page, généralement non affichées dans le contenu.

<meta charset="utf-8"> — Permet d'afficher correctement les caractères accentués.

<meta name="viewport" ...> — Aide à adapter l'affichage à la largeur de l'écran ; le responsive sera développé au module 4.

<title> — Définit le titre visible dans l'onglet du navigateur.

<body> — Contient le contenu visible de la page.

<h1> — Définit le titre principal de la page.

** ** — Définit un paragraphe.

Balises fermantes — , </body> et </html> terminent les éléments correspondants.

4. Enrichir le contenu avec des éléments sémantiques

Les éléments header, nav, main, section et footer décrivent le rôle des régions de la page. Utiliser un seul contenu principal main et une hiérarchie cohérente de titres. Les listes ul/li regroupent des éléments sans ordre ; ol/li indique une succession ordonnée.

Un lien s'écrit Visiter le site. Une image s'écrit <img src="images/photo.jpg" alt="Description utile">. L'attribut alt décrit l'information apportée par l'image ; pour une image purement décorative, alt="" peut être approprié.

5. Comprendre les chemins de fichiers

Depuis index.html, href="contact.html" désigne un fichier voisin ; src="images/photo.jpg" désigne un fichier placé dans le sous-dossier images. Respecter l'orthographe, les majuscules et les

extensions. Éviter les chemins propres à son ordinateur (C:\...) pour les ressources destinées à être publiées.

6. Travail pratique guidé : une page de présentation

Livrable : un fichier index.html autonome à créer en recopiant le modèle complet ci-dessous. Il contient un en-tête, une navigation interne, une présentation, une liste de compétences, un lien externe, une section de contact fictive et un pied de page. Aucun CSS ni JavaScript n'est nécessaire à ce stade.

Modèle de départ — enregistrer le code suivant dans index.html :

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Mon espace de découverte</title>
</head>
<body>
  <header>
    <h1>Mon espace de découverte</h1>
    <nav><a href="#presentation">Présentation</a> | <a href="#competences">Compétences</a> | <a href="#contact">Contact</a></nav>
  </header>
  <main>
    <section id="presentation">
      <h2>Présentation</h2>
      <p>Bienvenue sur mon espace de découverte.</p>
    </section>
    <section id="competences">
      <h2>Compétences</h2>
      <ul><li>Observer</li><li>Créer</li></ul>
    </section>
    <section id="contact">
```

```
      <h2>Contact</h2>
      <p>Contact fictif : aucune donnée personnelle à saisir.</p>
      <a href="https://developer.mozilla.org/fr/">Documentation du Web</a>
    </section>
  </main>
  <footer><p>Exercice pédagogique — page fictive.</p></footer>
</body>
</html>
```

1. Ouvrir le fichier index.html créé à partir du modèle ci-dessus et repérer head et body.
2. Changer le titre d'onglet et le h1. Constater qu'ils n'ont pas le même rôle.
3. Ajouter une troisième compétence dans la liste.
4. Créer une nouvelle section avec un h2 et un paragraphe.
5. Vérifier que les liens internes conduisent aux bonnes sections.
6. Expliquer oralement le rôle de header, nav, main et footer.

7. Diagnostic et erreurs fréquentes

La page affiche le code brut → Vérifier l'extension .html et l'ouverture dans le navigateur.

Les accents sont incorrects → Contrôler UTF-8 dans le fichier et la balise meta charset.

Un lien interne ne fonctionne pas → Comparer href="#contact" et id="contact" caractère par caractère.

Une image ne s'affiche pas → Vérifier le nom du fichier, le dossier et l'extension.

La mise en page semble rudimentaire → C'est normal : HTML structure le contenu ; CSS sera appris au module suivant.

8. Utiliser ChatGPT sans lui déléguer la compréhension

Exemple de demande : « Je débute en HTML. Explique-moi la fonction de chaque balise de cette page sans réécrire le code. Pose-moi ensuite trois questions de compréhension. » Puis : « Identifie seulement l'erreur dans mon lien interne et propose une méthode pour la trouver moi-même. » Comparer les suggestions à ce que le navigateur affiche réellement.

9. Évaluation formative et autonomie

1. Que signifie HTML ?
2. Quelle différence entre title et h1 ?
3. Où se place le contenu visible ?

4. Pourquoi utiliser lang="fr" ?
5. À quoi sert alt sur une image ?
6. Comment relier un lien à une section de la même page ?
7. Pourquoi un fichier local ne doit-il pas référencer C:\Users\... ?
8. Quel est le rôle de main ?

Exercice autonome (45 minutes) : sans assistant IA, créer une page « Mon projet numérique » avec un titre, deux sections, trois éléments de liste, un lien interne et un lien externe. Ouvrir la page, tester les liens, puis écrire cinq phrases expliquant ses choix. Réussite : au moins 6 réponses exactes sur 8 et page fonctionnelle, avec correction des erreurs identifiées.

10. Corrigé synthétique de l'autoévaluation

1. HyperText Markup Language ; langage de balisage.
2. title : onglet ; h1 : titre principal du contenu.
3. Dans body.
4. Déclarer la langue principale, notamment pour l'accessibilité.
5. Donner une alternative textuelle appropriée à l'image.
6. Avec href="#identifiant" et id="identifiant" sur la cible.
7. Ce chemin dépend de la machine et ne sera pas portable.
8. Identifier le contenu principal de la page.

11. Étude de cas — Récits Vivants

Dans un site d'auteur, une page « Découvrir » peut présenter une œuvre, son intention, des liens vers les autres pages et une navigation claire. Avant d'ajouter des effets visuels ou une expérience immersive, la structure HTML doit rester compréhensible et accessible. L'apprenant peut reproduire une version fictive simplifiée sans copier de contenu protégé ni utiliser de données personnelles.

12. Passage au module 4

Le fichier index.html obtenu fonctionne, mais sa présentation reste volontairement sobre. Au module suivant, nous ajouterons une feuille style.css, des espacements, des couleurs, Flexbox/Grid et des règles responsive. La structure HTML sera conservée pour montrer la séparation entre contenu et apparence.

MODULE 4 — CSS3 et responsive design

14 heures • Niveau débutant • Continuité directe du fichier index.html du module 3

1. Objectifs et critères de réussite

À la fin de ce module, l'apprenant sait relier un fichier CSS à une page HTML, expliquer sélecteur/propriété/valeur, distinguer marge intérieure et extérieure, réaliser une mise en page Flexbox et Grid, puis tester trois tailles d'écran. Il doit pouvoir reproduire et dépanner une modification simple sans IA.

2. Organisation des 14 heures

- Séquence 1 (2 h) : CSS et liaison au HTML ; cascade, sélecteurs et commentaires.
- Séquence 2 (2 h) : couleurs, typographie, unités, contrastes.
- Séquence 3 (2 h) : modèle de boîte, margin, padding, border et box-sizing.
- Séquence 4 (2 h) : Flexbox pour la navigation et l'alignement.
- Séquence 5 (2 h) : CSS Grid pour organiser les sections.
- Séquence 6 (2 h) : responsive mobile-first et media queries.
- Séquence 7 (2 h) : tests, débogage, autonomie et évaluation.

3. Le projet fil rouge : conserver la page du module 3

Le fichier index.html du module 3 est conservé à l'identique, à une seule ligne près : dans head, juste après la balise viewport, ajouter :

```
<link rel="stylesheet" href="style.css">
```

Créer style.css dans le même dossier que index.html. Le navigateur charge le HTML puis applique les règles CSS. Sans CSS, le contenu reste lisible : c'est une séparation volontaire entre structure et présentation.

4. Lire une règle CSS, ligne par ligne

```
h1 {
  color: #183153;
  font-size: 2rem;
}
```

h1 est le sélecteur : il cible les titres h1. Les accolades encadrent les déclarations. color règle la couleur du texte. #183153 est une couleur hexadécimale. font-size règle la taille ; rem dépend de la taille de police racine. Chaque déclaration se termine par un point-virgule.

La cascade détermine quelle règle s'applique en cas de conflit : origine, importance, spécificité et ordre interviennent. Pour débuter, éviter les styles en ligne et !important ; utiliser des sélecteurs simples.

5. Couleurs, typographie et unités

Une palette sobre améliore la cohérence : fond clair, texte foncé, couleur d'accent. Tester le contraste texte/fond ; ne pas transmettre une information par la couleur seule. px correspond à une unité CSS, rem s'adapte à la taille de référence, % dépend du contexte, vw dépend de la largeur du viewport.

```
:root {
  --fond: #f6f7fa;
  --texte: #18283c;
  --accent: #235c9f;
}
body { color: var(--texte); background: var(--fond); }
```

6. Le modèle de boîte : content, padding, border, margin

Le contenu occupe l'espace central. padding ajoute un espace intérieur ; border trace une bordure ; margin ajoute un espace extérieur. Avec box-sizing: border-box, largeur et hauteur déclarées incluent padding et bordure, ce qui simplifie les calculs.

```
*, *::before, *::after { box-sizing: border-box; }
section { padding: 1.5rem; border: 1px solid #d7dfeb; margin-block: 1rem; }
```

7. Flexbox : organiser une navigation

display:flex aligne les éléments sur un axe principal. flex-wrap:wrap autorise le retour à la ligne. gap espace les éléments sans multiplier les marges. Les liens restent des liens HTML, et non des boutons fictifs.

```
nav { display: flex; flex-wrap: wrap; gap: .75rem; }
nav a { padding: .5rem .75rem; }
```

Dans la page initiale, les caractères « | » séparant les liens restent du texte : dans l'exercice guidé, les supprimer afin que gap gère seul les espaces. Le fichier de départ livré conserve toutefois la version HTML initiale, sauf la liaison CSS.

8. CSS Grid : composer plusieurs zones

Grid répartit les éléments en lignes et colonnes. Pour ce projet, main devient une grille à une colonne sur mobile ; à partir de 768 px, la présentation et les compétences passent côte à côte, tandis que Contact occupe toute la largeur.

```
@media (min-width: 768px) {
  main { grid-template-columns: repeat(2, minmax(0, 1fr)); }
  #contact { grid-column: 1 / -1; }
}
```

9. Responsive design : mobile-first

Concevoir d'abord pour les petits écrans : largeur fluide, textes lisibles, éléments qui passent à la ligne. Les media queries enrichissent la présentation quand l'espace augmente. Le meta viewport du module 3 est déjà présent ; il ne suffit pas, à lui seul, à rendre une page responsive.

Repères de test : 375 px (mobile), 768 px (tablette) et 1280 px (ordinateur). Ce sont des largeurs de test, pas des catégories universelles d'appareils. Vérifier également le zoom à 200 % et la navigation clavier.

10. Feuille style.css complète et commentée

```
/* Module 4 — CSS de départ, sans bibliothèque */
:root {
  --fond: #f6f7fa;
  --surface: #ffffff;
  --texte: #18283c;
  --accent: #235c9f;
  --bordure: #d7dfeb;
}
*, *::before, *::after { box-sizing: border-box; }
html { scroll-behavior: smooth; }
body {
  margin: 0;
  font-family: system-ui, -apple-system, "Segoe UI", sans-serif;
  line-height: 1.6;
  color: var(--texte);
  background: var(--fond);
}
header, main, footer {
  width: min(100% - 2rem, 68rem);
  margin-inline: auto;
}
header { padding-block: 2.5rem 1.5rem; }
h1 { font-size: clamp(1.8rem, 4vw, 3rem); line-height: 1.15; }
h2 { font-size: 1.35rem; }
a { color: var(--accent); }
a:focus-visible { outline: 3px solid #b36a0a; outline-offset: 3px; }
nav { display: flex; flex-wrap: wrap; gap: .75rem; margin-top: 1rem; }
nav a { padding: .5rem .75rem; background: var(--surface); border-radius: .5rem; }
main { display: grid; grid-template-columns: 1fr; gap: 1rem; }
section {
  min-width: 0;
  padding: 1.5rem;
  border: 1px solid var(--bordure);
  border-radius: .75rem;
  background: var(--surface);
}
footer { padding-block: 2rem; font-size: .9rem; }
@media (min-width: 768px) {
  main { grid-template-columns: repeat(2, minmax(0, 1fr)); }
  #contact { grid-column: 1 / -1; }
}
@media (prefers-reduced-motion: reduce) {
  html { scroll-behavior: auto; }
}
```

11. TP guidé : construire, observer, expliquer (90 min)

1. Copier index.html du module 3 dans un nouveau dossier « module-4 ». Ne pas réécrire son contenu.
2. Ajouter uniquement la balise link dans head et créer style.css dans ce dossier.
3. Coller la feuille CSS ci-dessus. Ouvrir index.html dans un navigateur.

4. Supprimer les deux séparateurs « | » de nav pour laisser Flexbox gérer les espaces.
5. Modifier --accent et expliquer ce qui change ; restaurer la valeur initiale.
6. Passer de padding: 1.5rem à 2rem dans section et décrire la différence avec margin.
7. Tester 375, 768 et 1280 px avec les outils de développement du navigateur.
8. Désactiver temporairement le lien CSS : constater que le contenu et les liens HTML restent disponibles.
9. Réactiver le CSS et vérifier au clavier Tab que chaque lien est identifiable.

12. Diagnostic : comprendre les erreurs fréquentes

- Aucun style visible → vérifier le nom style.css, son emplacement, le href et l'enregistrement du fichier.
- Une règle ne s'applique pas → vérifier l'orthographe, les accolades, le point-virgule et une éventuelle règle plus prioritaire.
- Un élément déborde → inspecter width, padding, min-width et les contenus trop longs.
- Les colonnes ne changent pas → vérifier la largeur réelle de la fenêtre et la media query.
- Les liens de navigation sont trop espacés → retirer les séparateurs « | » du HTML de départ.
- Le texte est difficile à lire → vérifier contraste, taille, line-height et zoom.

13. ChatGPT : assistance contrôlée

Demande conseillée : « Je débute en CSS. Voici mon HTML et mon CSS. Mon lien de feuille de style ne fonctionne pas. Pose-moi d'abord trois questions de diagnostic ; ne réécris pas tout mon projet. Explique ensuite la correction minimale. »

Règle de méthode : observer → formuler une hypothèse → tester → expliquer. Ne jamais remplacer l'ensemble du code sans avoir compris le problème.

14. Autoévaluation : dix questions

1. À quoi sert la balise link rel="stylesheet" ?
2. Que ciblent h1 et nav a ?
3. Différence entre margin et padding ?
4. Pourquoi utiliser box-sizing: border-box ?
5. Quel est le rôle de gap dans Flexbox ?
6. Quand préférer Grid à Flexbox ?
7. Que signifie mobile-first ?
8. À quoi sert @media (min-width: 768px) ?
9. Pourquoi vérifier le zoom et le clavier ?
10. Comment vérifier que le CSS est bien chargé ?

15. Corrigé et critères d’autonomie

1. Elle relie le document HTML à une feuille CSS externe.
2. h1 cible tous les titres h1 ; nav a cible les liens contenus dans nav.
3. margin est extérieur à la bordure ; padding est intérieur.
4. Pour inclure bordure et padding dans les dimensions déclarées.
5. Il crée des espaces entre les éléments flex.
6. Grid pour une organisation en lignes et colonnes ; Flexbox pour un axe principal.
7. Concevoir les styles de base pour les petits écrans, puis enrichir.
8. Elle active des règles lorsque le viewport atteint au moins 768 px.
9. Pour vérifier la lisibilité et l’accessibilité de l’interface.
10. Inspecter le href, le fichier, les outils développeur et désactiver/réactiver la feuille.

Exercice autonome (45 min) : sans IA, changer la palette, augmenter l’espace intérieur d’une section, modifier le seuil de passage en colonnes, puis résoudre volontairement un mauvais nom de fichier CSS. Rédiger quatre phrases expliquant les changements. Réussite : page fonctionnelle sur trois largeurs, navigation clavier utilisable, absence de débordement horizontal et explications exactes.

16. Étude de cas et transition

Dans une page de découverte des Récits Vivants, HTML porte le sens du contenu, CSS organise sa présentation et le responsive permet une lecture sur différents écrans. Le module 5 ajoutera JavaScript à ce même projet : une interaction simple, compréhensible et testable, sans perturber l’accès au contenu.

MODULE 5 — JavaScript et interactivité

12 heures • Bloc A : Comprendre et construire • Niveau débutant • Édition pédagogique V2.2 — relecture en cours

1. Finalité et compétences

À la fin de ce module, l'apprenant sait expliquer le rôle de JavaScript, relier un fichier script.js à index.html, écouter un événement, modifier un texte du document et conserver un choix non sensible dans localStorage. Il doit aussi savoir diagnostiquer une erreur simple sans assistance de l'IA.

2. Déroulement des 12 heures

- Séquence 1 (2 h) — Rôle de JavaScript, console et première instruction.
- Séquence 2 (2 h) — Variables, chaînes, nombres, conditions et fonctions.
- Séquence 3 (2 h) — DOM, sélection des éléments et événements.
- Séquence 4 (2 h) — Interaction accessible : bouton et message.
- Séquence 5 (2 h) — Stockage local, limites et confidentialité.
- Séquence 6 (2 h) — Tests, débogage, TP autonome et évaluation.

3. Comprendre la place de JavaScript

HTML décrit le contenu, CSS règle son apparence et JavaScript peut modifier le comportement de la page. JavaScript s'exécute ici dans le navigateur. Il ne faut pas le confondre avec Java. Une page informative doit rester lisible même si le script ne se charge pas.

4. Premières notions, expliquées

Variable

const crée une liaison qui ne peut pas être réaffectée ; let permet une réaffectation. Exemple : const nom = "Camille" ;

Condition

if (condition) { ... } exécute des instructions seulement si la condition est vraie.

Fonction

Une fonction regroupe des instructions réutilisables. Exemple : function saluer() { return "Bonjour"; }

DOM
Le Document Object Model représente les éléments HTML comme des objets manipulables par JavaScript.

Événement
Un clic, une saisie ou une touche peuvent déclencher une fonction. addEventListener associe une réaction à un événement.

Stockage local
localStorage conserve des chaînes de caractères dans le navigateur, pour l'origine du site. Il ne remplace ni un compte utilisateur ni une sauvegarde garantie.

5. Faire évoluer le projet des modules 3 et 4
Nous conservons exactement les sections Présentation, Compétences et Contact, ainsi que style.css. Nous ajoutons un petit panneau interactif dans la section Présentation, puis un script externe chargé avec defer dans <head>. Conserver les fichiers réalisés aux modules 3 et 4 ; les ajouts JavaScript sont décrits dans les étapes suivantes.

Étape A — Le HTML ajouté dans la section Présentation
```
<div class="interaction">
  <button type="button" id="bonjour">Afficher un message</button>
  <p id="message" role="status" aria-live="polite">Aucun message affiché.</p>
  <label for="pseudo">Votre prénom (facultatif)</label>
  <input id="pseudo" type="text" maxlength="40" autocomplete="off">
  <button type="button" id="memoriser">Mémoriser ce prénom</button>
  <button type="button" id="effacer">Effacer le prénom</button>
</div>
```

button déclenche une action ; id identifie chaque élément ; label associe une consigne au champ ; role="status" et aria-live permettent d'annoncer les changements de texte aux technologies d'assistance.

Étape B — Charger le script
Dans <head>, ajouter : <script src="script.js" defer></script>. defer permet d'exécuter le script après l'analyse du HTML, sans bloquer sa lecture.

Étape C — Comprendre script.js ligne par ligne
```
const bouton = document.querySelector("#bonjour");  // Sélectionne le bouton grâce à son identifiant.

const message = document.querySelector("#message");  // Sélectionne le paragraphe qui affichera le résultat.

bouton.addEventListener("click", () => {  // Déclenche une fonction au clic.

  message.textContent = "Bonjour et bienvenue !";  // Modifie le texte sans interpréter du HTML.
```

```
});  // Ferme la fonction et l'appel à addEventListener.
```

Exemple complet de script.js — à recopier dans le fichier (TP 1 à 4) :

```
const bouton = document.querySelector("#bonjour");
const message = document.querySelector("#message");
const champ = document.querySelector("#pseudo");
const memoriser = document.querySelector("#memoriser");
const effacer = document.querySelector("#effacer");

bouton.addEventListener("click", () => {
  const prenom = champ.value.trim();
  message.textContent = prenom ? "Bonjour " + prenom + " !" : "Bonjour et bienvenue !";
});

function sauvegarderPrenom(valeur) {
  try {
    localStorage.setItem("prenom", valeur);
    return true;
  } catch (erreur) {
    message.textContent = "Stockage indisponible : le prénom ne sera pas conservé.";
    return false;
  }
}

memoriser.addEventListener("click", () => {
  const prenom = champ.value.trim();
  if (sauvegarderPrenom(prenom)) message.textContent = "Prénom mémorisé sur cet appareil.";
});

effacer.addEventListener("click", () => {
  champ.value = "";
```

```
  try { localStorage.removeItem("prenom"); } catch (erreur) { /* stockage indisponible */ }
  message.textContent = "Prénom effacé du formulaire.";
});

try {
  champ.value = localStorage.getItem("prenom") || "";
} catch (erreur) {
  message.textContent = "Stockage indisponible : saisie possible sans mémorisation.";
}
```

Fin de l'exemple. Tester d'abord le clic, puis la saisie, la mémorisation, le rechargement et l'effacement. Ne saisir qu'un prénom fictif pour l'exercice.

6. Stocker un choix avec prudence

localStorage.setItem("prenom", valeur) enregistre une chaîne ; getItem la relit ; removeItem la supprime. La donnée reste propre au navigateur et à l'origine du site, et peut disparaître si l'utilisateur efface ses données ou change d'appareil. Sur une page ouverte en file://, le comportement de localStorage peut varier selon les navigateurs : le TP prévoit un message d'erreur explicite et une solution de repli. Ne jamais y stocker de mot de passe, secret ou donnée sensible.

7. Méthode de diagnostic

1. Vérifier les noms exacts : index.html, style.css, script.js.
2. Ouvrir les outils de développement (souvent F12) puis la console.
3. Lire la première erreur et identifier fichier et ligne.
4. Contrôler les id HTML et les sélecteurs querySelector.
5. Modifier une seule chose, recharger, retester au clavier et sur mobile.
6. Revenir à la dernière version fonctionnelle si nécessaire.

8. Travaux pratiques guidés

TP 1 — Premier clic (30 min)

Ajouter le bouton, charger script.js, afficher un message, puis expliquer les trois lignes essentielles.

TP 2 — Une condition (30 min)

Modifier le texte selon que le champ prénom est vide ou non ; utiliser trim() pour retirer les espaces.

TP 3 — Une erreur volontaire (30 min)
Renommer temporairement l'id du bouton ; constater l'erreur, trouver la cause et rétablir le fonctionnement.

TP 4 — Mémoire locale (45 min)
Enregistrer un prénom facultatif, actualiser la page, constater sa restauration puis l'effacer. Vérifier le cas où le stockage est indisponible.

TP 5 — Accessibilité (30 min)
Tester Tabulation, Entrée et Espace ; vérifier le focus visible et le message annoncé.

9. Autoévaluation — dix questions
1. Que fait JavaScript que HTML seul ne fait pas ?

2. À quoi sert defer ?

3. Quelle différence entre const et let ?

4. Que représente le DOM ?

5. Que renvoie querySelector si aucun élément ne correspond ?

6. Pourquoi employer textContent plutôt que innerHTML pour afficher un prénom ?

7. Quel événement écoute le bouton ?

8. Comment supprimer une valeur de localStorage ?

9. Pourquoi localStorage ne constitue-t-il pas une sauvegarde fiable entre appareils ?

10. Que vérifier si un clic ne produit aucun effet ?

10. Corrigé de l'autoévaluation
1. Il peut réagir aux actions et modifier la page.

2. Il diffère l'exécution du script externe après l'analyse du document.

3. const ne peut pas être réaffecté, let peut l'être.

4. La représentation manipulable du document HTML.

5. null.

6. Pour traiter le prénom comme du texte et éviter d'injecter du HTML.

7. click.

8. removeItem("clé").

9. Les données sont locales au navigateur, à l'origine et peuvent être effacées.

10. Le fichier chargé, la console, les identifiants et les écouteurs d'événements.

11. Exercice autonome (60 min)

Sans demander de code à l'IA : ajouter un bouton « Changer le message » qui alterne entre deux phrases ; mémoriser le dernier choix si le stockage est disponible ; ajouter un bouton de remise à zéro ; vérifier le fonctionnement au clavier et après rechargement. Fournir trois captures (mobile, ordinateur, console sans erreur) et un paragraphe expliquant chaque changement.

12. Grille de réussite et transition

☐ HTML reste sémantique et lisible sans JavaScript.

☐ Le script est externe et les événements fonctionnent.

☐ Le stockage est facultatif et effaçable.

☐ Les erreurs courantes sont identifiées dans la console.

☐ L'apprenant explique et reproduit le résultat sans IA.

Fin du bloc A — 56 heures cumulées. Le module 6 introduira la gestion de versions avec Git et, facultativement, une plateforme distante pour conserver l'historique de cette réalisation, comparer les modifications et préparer la publication.

MODULE 6 — GESTION DE VERSIONS : GIT ET PLATEFORMES ALTERNATIVES

Bloc B — Développer et publier | Durée : 10 heures | Niveau : débutants complets

Objectifs et critères de réussite

- Expliquer avec ses mots la différence entre Git (historique local) et GitHub (service d'hébergement de dépôts).
- Créer un dépôt, vérifier les fichiers avant publication et rédiger un README.
- Enregistrer une modification cohérente dans un commit et expliquer ce qu'elle change.
- Comparer deux versions, identifier une erreur et revenir à un état fonctionnel sans effacer son travail.
- Reconnaître les risques de publication : secrets, données personnelles, licences et visibilité publique.

Plan des 10 heures

Séquence 1 — Pourquoi versionner ? Git, GitHub, sauvegarde et vocabulaire (1 h 30).

Séquence 2 — Créer un dépôt local ; compte distant facultatif, choisir public/privé si publication (1 h 30).

Séquence 3 — Ajouter le projet HTML/CSS/JS et écrire README.md (1 h 30).

Séquence 4 — Modifications, commits et historique (2 h).

Séquence 5 — Comparer, corriger, restaurer et branches (2 h).

Séquence 6 — Évaluation et autonomie (1 h 30).

1. Comprendre la gestion de versions

Sans historique, un fichier index-final-v2-corrigé.html devient vite difficile à suivre. Git enregistre des instantanés cohérents du projet. Un commit est un point de l'historique identifié et commenté. GitHub permet de conserver et partager un dépôt Git en ligne ; il ne remplace pas à lui seul une stratégie de sauvegarde.

Lexique : dépôt = dossier suivi par Git ; commit = enregistrement ; branche = ligne de travail ; différence (diff) = comparaison ; clone = copie locale ; push = envoi de commits vers un dépôt distant ; pull = récupération et intégration des changements distants ; rollback = retour contrôlé à une version antérieure.

2. Préparer notre projet évolutif

Nous réutilisons les trois fichiers issus du module 5 : index.html, style.css et script.js. Avant toute publication, ouvrir index.html, tester les boutons, vérifier les liens et supprimer les informations privées. Un fichier de configuration contenant un mot de passe ou une clé API ne doit jamais être ajouté au dépôt.

mon-premier-projet/

index.html

style.css

script.js

README.md

.gitignore

Le fichier README.md explique ce que fait le projet et comment le lancer. Le fichier .gitignore indique à Git les fichiers locaux à ne pas suivre. Il ne supprime pas un secret déjà enregistré dans l'historique.

3. Méthode illustrative : interface web GitHub (facultative)

Étape 1. Se connecter à GitHub et choisir « New repository ».

Étape 2. Nommer le dépôt mon-premier-projet ; choisir sa visibilité en connaissance de cause.

Étape 3. Créer le dépôt avec un README, puis ajouter les fichiers du projet par l'interface « Add file » (les libellés peuvent évoluer).

Étape 4. Vérifier la liste des fichiers, rédiger un message précis (« Ajouter la page HTML du module 5 ») et valider le commit.

Étape 5. Ouvrir l'historique des commits, puis inspecter un changement.

Attention : un dépôt public peut être consulté et copié par d'autres. Une suppression ultérieure n'efface pas nécessairement les copies déjà réalisées.

4. Deuxième méthode : commandes Git élémentaires

Sur un ordinateur où Git est installé, ouvrir un terminal dans le dossier du projet. Les commandes ci-dessous sont expliquées une à une. Ne pas les exécuter dans un dossier contenant des documents personnels.

```
git init

git status

git add index.html style.css script.js README.md

git commit -m "Créer la première version du site"

git log --oneline

git diff
```

- git init : démarre le suivi du dossier.
- git status : indique ce qui a changé et ce qui est prêt à être enregistré.
- git add : prépare les fichiers désignés pour le prochain commit.
- git commit : enregistre l'instantané préparé avec un message.
- git log --oneline : affiche l'historique condensé.

- git diff : montre les changements non préparés ; git diff --staged montre ceux déjà préparés. Juste après un commit, si aucun fichier n'a été modifié, git diff n'affiche rien : c'est normal. Pour examiner un commit enregistré, utiliser git show --stat HEAD ou git show HEAD.

Pour le premier commit, Git peut demander une identité : exécuter git config --local user.name "Apprenant" puis git config --local user.email "apprenant@example.invalid" (identité fictive uniquement pour l'exercice local). Ne pas publier une identité fictive en prétendant qu'elle est réelle. Selon la configuration, Git peut demander un nom et une adresse électronique avant le premier commit. L'envoi vers GitHub demande en plus de configurer un dépôt distant et une authentification adaptée ; la méthode web est suffisante pour le premier TP.

5. Atelier guidé : une modification, un commit

Ouvrir index.html du projet fil rouge : dans le modèle du module 3, le titre visible h1 est « Mon espace de découverte » (ou sa version personnalisée par l'apprenant). Le modifier en « Mon projet numérique interactif ». Actualiser la page dans le navigateur, exécuter git diff avant git add pour constater le changement, puis git add index.html et git diff --staged pour vérifier exactement ce qui sera enregistré. Créer ensuite un commit nommé « Clarifier le titre de la page ». Le commit doit concerner uniquement cette modification.

Question : pourquoi ne pas enregistrer dans le même commit un changement de titre, une correction CSS et la suppression d'un fichier sans rapport ? Réponse attendue : un commit ciblé se comprend, se teste et se restaure plus facilement.

6. Diagnostiquer une régression et revenir en arrière

Situation : après un changement de script.js, le bouton ne réagit plus. 1) Reproduire l'erreur dans le navigateur. 2) Consulter git diff ou l'onglet de comparaison GitHub. 3) Repérer la ligne modifiée. 4) Corriger l'erreur et enregistrer un nouveau commit. 5) Retester sur mobile et ordinateur.

Si le problème n'est pas compris, travailler sur une copie ou une branche de test. Pour annuler un commit déjà partagé, préférer une opération qui conserve une trace (par exemple git revert avec accompagnement), plutôt qu'une réécriture destructrice de l'historique. Éviter git reset --hard sur un projet contenant du travail non sauvegardé.

7. Branches et pull requests : première découverte

Une branche permet d'essayer une modification sans changer immédiatement la version principale. Une pull request propose de fusionner les changements après comparaison et revue. Dans notre exercice, une branche « essai-couleurs » servira à tester une variante de style.css, puis à expliquer ce qui doit être vérifié avant fusion.

8. Étude de cas : conserver une version stable des Récits Vivants

Dans un projet narratif interactif, une amélioration peut affecter les liens, les animations ou la mémoire du visiteur. Avant modification, identifier la version stable, décrire le changement attendu, tester la nouvelle version et conserver la possibilité de restaurer l'ancienne. Git aide à retracer les changements ; il ne garantit pas que l'expérience reste cohérente sur le plan artistique.

9. Travaux pratiques et corrigés

TP1 — Dessiner le cycle : modifier → comparer → tester → commit.

Consigne : réaliser la tâche, noter les étapes et fournir une capture ou une explication écrite.

Corrigé : le commit vient après comparaison et test ; un nouveau changement relance le cycle.

TP2 — Écrire un README : objectif, trois fichiers, mode d'ouverture et limites.

Consigne : réaliser la tâche, noter les étapes et fournir une capture ou une explication écrite.

Corrigé : une autre personne doit comprendre le projet et ouvrir index.html sans explication orale.

TP3 — Identifier cinq fichiers qui ne devraient pas être publiés.

Consigne : réaliser la tâche, noter les étapes et fournir une capture ou une explication écrite.

Corrigé indicatif : mot de passe, clé API, fichier de données clients, export personnel, configuration secrète.

TP4 — Provoquer volontairement une erreur de texte, retrouver la différence et la corriger.

Consigne : réaliser la tâche, noter les étapes et fournir une capture ou une explication écrite.

Corrigé : observer le diff, identifier la modification exacte, réparer et retester.

TP5 — Créer une branche d'essai et expliquer sa différence avec la branche principale.

Consigne : réaliser la tâche, noter les étapes et fournir une capture ou une explication écrite.

Corrigé : la branche isole un travail expérimental jusqu'à sa revue et sa fusion éventuelle.

10. Autoévaluation : dix questions

1. Git et GitHub sont-ils identiques ?

Réponse : Non : Git est un système de gestion de versions ; GitHub héberge des dépôts et propose des outils de collaboration.

2. Qu'est-ce qu'un dépôt ?

Réponse : Un ensemble de fichiers et leur historique Git.

3. Qu'est-ce qu'un commit ?

Réponse : Un enregistrement identifié d'un état préparé du projet.

4. Que montre un diff ?

Réponse : Les changements entre deux états.

5. Pourquoi tester avant un commit ?

Réponse : Pour éviter de considérer une version défectueuse comme validée.

6. Que fait git status ?

Réponse : Il montre l'état des fichiers suivis, modifiés ou préparés.

7. À quoi sert un README ?

Réponse : À présenter le projet et les instructions de démarrage.

8. Peut-on publier une clé API dans un dépôt privé ?

Réponse : Non : le caractère privé réduit l'exposition mais ne justifie pas d'y placer des secrets.

9. Pourquoi créer une branche ?

Réponse : Pour isoler un changement ou un essai.

10. Comment traiter une régression après publication ?

Réponse : Reproduire, comparer, corriger ou annuler de façon traçable, puis retester.

11. Épreuve d'autonomie (sans IA) — 45 minutes

À partir des trois fichiers fournis : créer un dépôt, écrire un README, effectuer deux changements distincts, créer deux commits explicites, retrouver les différences et expliquer oralement comment revenir à une version fonctionnelle. L'évaluation porte sur les preuves de compréhension, pas seulement sur la présence d'un dépôt.

Barème sur 20 : structure du dépôt et README (4), commits cohérents (4), lecture de l'historique et des différences (4), diagnostic et retour à un état stable (5), sécurité et explication autonome (3). Réussite visée : 14/20 et aucune publication de secret.

12. Transition vers le module 7 — Codex

Le projet du module 6 servira de support à un assistant IA de développement, Codex constituant l'exemple guidé. Selon l'outil et les autorisations, l'apprenant pourra connecter un dépôt, ouvrir un dossier local ou transmettre manuellement les fichiers et extraits utiles. Avant d'accepter ou d'appliquer une proposition, il devra lire les différences, tester le résultat et conserver une version traçable. GitHub facilite la revue, sans constituer un accès obligatoire pour l'assistant.

MODULE 7 — Développement assisté par IA : Codex et alternatives (12 heures)

Bloc B — Développer et publier | Prérequis : modules 1 à 6 | Projet fil rouge : mon-premier-projet (index.html, style.css, script.js, README.md).

1. Compétences visées

• Décrire ce qu'est un agent de développement et distinguer ChatGPT conversationnel de Codex intervenant sur des fichiers.

• Préparer un dossier ou dépôt versionné, une copie de travail et une demande de modification vérifiable.

• Lire un diff et expliquer les changements ligne par ligne, y compris leurs conséquences.

• Tester un changement, reconnaître une régression et décider de conserver, corriger ou refuser une proposition.

• Documenter une décision et préserver la sécurité des données et des accès.

2. Parcours horaire

Séquence 1 (2 h) — ChatGPT, Codex et la notion d'agent : rôles et limites.

Séquence 2 (2 h) — Préparer le dépôt, le contexte et la branche de travail.

Séquence 3 (2 h) — Formuler une tâche bornée et examiner la proposition.

Séquence 4 (2 h) — Lire le diff, tester et vérifier l'accessibilité.

Séquence 5 (2 h) — Régressions, sécurité, refus et retour arrière.

Séquence 6 (2 h) — Travaux pratiques, autonomie et évaluation.

3. Comprendre les outils

ChatGPT aide à clarifier les besoins, expliquer des concepts et concevoir des critères de réussite. Codex est un agent de développement susceptible, selon l'interface et les autorisations, de lire, modifier et tester les fichiers d'un projet. Git conserve l'historique local ; GitHub héberge le dépôt et facilite la revue. L'accès à un dépôt, les fonctionnalités et les quotas dépendent de la configuration du service : ne jamais présumer qu'une connexion est déjà autorisée. D'autres assistants IA peuvent remplir tout ou partie de ces fonctions. Sans accès direct aux fichiers ou à GitHub, fournir une copie des fichiers nécessaires, demander une proposition de modifications clairement localisées, puis les appliquer et les tester manuellement. Ne jamais présenter un changement proposé comme déjà exécuté.

Règle : l'agent propose et peut exécuter des actions dans le périmètre accordé ; l'apprenant reste responsable de la revue, des tests et de la décision de fusionner ou publier.

4. Préparer une base saine

Ouvrir le projet issu du module 6. Vérifier que index.html, style.css et script.js fonctionnent. Lire README.md et identifier les interactions existantes. Vérifier qu'aucun secret, jeton, donnée personnelle ou fichier inutile n'est présent. Sauvegarder les changements non enregistrés avant de

créer une branche. Si l'assistant ne peut pas ouvrir ce projet, lui transmettre uniquement les extraits ou fichiers nécessaires, sans secrets ; conserver une copie de référence et réaliser localement les opérations Git.

```
git status

git switch -c exercice-codex-message

git log --oneline -5
```

Lecture : status affiche les changements en cours ; switch -c crée une branche et s'y place ; log présente les derniers commits. En cas d'erreur, demander l'explication de la commande avant toute exécution supplémentaire.

5. Définir une tâche précise et mesurable

Demande d'exercice : ajouter, sur la page existante, un bouton « Afficher une astuce » qui fait apparaître une courte phrase. Le bouton doit fonctionner au clavier ; le texte doit rester lisible sur mobile. Ne modifier que les fichiers nécessaires, conserver le comportement du bouton de prénom et expliquer chaque modification.

Modèle d'instruction pour un assistant IA de développement (exemple : Codex)

```
Contexte : site statique HTML/CSS/JS du module 6.

Objectif : ajouter un bouton « Afficher une astuce ».

Périmètre : index.html, style.css, script.js uniquement.

Contraintes : pas de dépendance, pas de refonte, conserver les interactions.

Accessibilité : vrai élément <button>, texte lisible, focus visible.

Validation : décrire les fichiers modifiés, ou les modifications proposées si l'outil n'a pas accès aux fichiers, et les tests à réaliser.

Avant toute modification : résumer le plan en 3 étapes.
```

Un bon prompt précise l'objectif, les limites, les critères d'acceptation et les éléments à préserver. Un résultat qui semble joli mais casse une fonction existante n'est pas acceptable. L'assistant peut proposer un diff ou du code à recopier ; l'apprenant vérifie le résultat réel dans son propre environnement. Les fonctions d'édition et d'exécution varient selon l'outil.

6. Lire et comprendre les différences (diff)

Un diff montre les lignes supprimées (–) et ajoutées (+). Examiner séparément HTML (structure), CSS (présentation) et JavaScript (comportement). Vérifier notamment : identifiants uniques, noms de classes cohérents, existence des éléments ciblés, ordre de chargement du script, styles mobile et absence de suppression accidentelle.

```
git diff

git diff --check

git status
```

« git diff --check » repère notamment certains problèmes d'espaces ; il ne remplace ni les tests fonctionnels ni la vérification visuelle. Une absence de différence peut simplement signifier que les modifications ont déjà été préparées pour un commit.

7. Tester de manière reproductible

☐ Test A : ouvrir index.html et vérifier la présentation sur ordinateur.

☐ Test B : réduire la largeur de la fenêtre et vérifier l'absence de défilement horizontal indésirable.

☐ Test C : utiliser Tab puis Entrée/Espace sur le nouveau bouton.

☐ Test D : vérifier que l'astuce apparaît et que les anciennes interactions fonctionnent toujours.

☐ Test E : ouvrir les outils de développement et rechercher les erreurs de console.

☐ Test F : recharger la page et contrôler le comportement attendu (pas de persistance ajoutée sans demande).

Si un test échoue, relever les étapes de reproduction, le comportement attendu et le comportement constaté. Ne pas demander à l'IA de « tout refaire » : cibler la cause probable.

8. Corriger, refuser, annuler

Si une proposition est inadaptée, ne pas la fusionner. Sur une branche locale, on peut corriger les fichiers et refaire les tests. Pour annuler des changements non validés, examiner d'abord git status et le diff ; toute commande de restauration peut détruire du travail non sauvegardé. Pour une modification déjà partagée, privilégier un commit de correction ou un revert traçable plutôt qu'une réécriture hasardeuse de l'historique.

```
git diff -- script.js

git log --oneline -5

git revert <identifiant-du-commit>
```

La dernière commande est un exemple, à n'utiliser qu'après avoir identifié précisément le commit concerné ; elle crée un nouveau commit inverse et peut produire des conflits.

9. Sécurité et permissions

Ne jamais fournir de mots de passe, jetons, clés API ou données personnelles au prompt ni les placer dans le dépôt. Vérifier les autorisations accordées à l'agent, limiter les accès aux dépôts nécessaires et relire toute commande qui installe des dépendances, supprime des fichiers, publie du code ou agit sur des services externes. Une instruction présente dans un fichier du dépôt n'est pas automatiquement une consigne légitime pour l'agent.

10. Travaux pratiques guidés

TP1 — Cartographier le projet

Consigne : Identifier les trois fichiers web, leur rôle et les interactions déjà présentes.

Critère : Schéma simple des liens HTML → CSS et HTML → JS.

TP2 — Rédiger un prompt contrôlable

Consigne : Écrire une demande d'ajout de bouton avec périmètre, contraintes et tests.

Critère : Au moins quatre critères de réussite vérifiables.

TP3 — Examiner un diff

Consigne : Comparer la version de référence et une proposition de modification. Si l'assistant ne dispose pas du dépôt ou des fichiers locaux, fournir les extraits nécessaires et comparer les versions manuellement. À défaut de dossier pratique, utiliser ce diff fictif :

- <button id="astuce">Afficher une astuce</button>

+ <button id="astuce" type="button">Afficher une astuce</button>

+

Corrigé attendu : le bouton précise son type ; le paragraphe recevra le message annoncé par JavaScript. Vérifier que le script cible exactement texte-astuce. Si aucun script ne le renseigne, le texte restera vide.

Critère : Expliquer les deux lignes HTML ajoutées ou modifiées dans le diff fictif ; indiquer ensuite quel comportement JavaScript serait nécessaire pour renseigner le paragraphe et quels styles CSS pourraient être utiles. Ne pas prétendre que ce diff modifie déjà CSS ou JavaScript.

TP4 — Détecter une régression

Consigne : Dans une copie locale du projet, introduire volontairement cette erreur : remplacer document.querySelector("#bonjour") par document.querySelector("#bonjou") dans script.js, sans modifier le HTML. Recharger la page, observer l'erreur dans la console du navigateur, puis rétablir #bonjour. L'opération peut être réalisée directement par l'apprenant, sans accès de l'assistant aux fichiers. Corrigé attendu : l'identifiant ne correspond plus à celui du bouton ; querySelector renvoie null et addEventListener échoue si on l'appelle sur null. Retester les interactions après correction.

Critère : Repérer l'identifiant erroné, expliquer et corriger sans refonte.

TP5 — Décider et documenter

Consigne : Rédiger une fiche de revue : accepter, demander correction ou refuser.

Critère : Décision motivée et liste de tests effectivement réalisés.

11. Étude de cas — Récits Vivants

Dans une œuvre immersive, une amélioration d'interface peut changer le sens de l'expérience. Exemple : automatiser une transition temporelle sans imposer une action au visiteur. Avant de modifier le code, formuler ce qui doit rester libre, ce qui peut être mémorisé et ce qui doit être vérifié après 20 jours. La qualité du changement est à la fois technique, ergonomique et artistique.

12. Évaluation et corrigés

1. Codex est-il identique à GitHub ?

Corrigé : Non. Codex intervient sur le développement ; GitHub héberge et organise le dépôt.

2. Pourquoi créer une branche ?

Corrigé : Pour isoler les changements avant revue et fusion.

3. Que signifie le signe + dans un diff ?

Corrigé : Une ligne ajoutée dans la comparaison affichée.

4. Un test réussi suffit-il ?

Corrigé : Non. Il faut vérifier les autres comportements et le périmètre.

5. Quel élément HTML choisir pour une action ?

Corrigé : Un bouton <button> plutôt qu'un élément non interactif détourné.

6. Pourquoi vérifier la console ?

Corrigé : Pour détecter des erreurs JavaScript et obtenir des indices.

7. Pourquoi limiter les fichiers modifiés ?

Corrigé : Pour réduire les risques de régression et faciliter la revue.

8. Que faire si Codex supprime une fonction existante ?

Corrigé : Refuser ou demander une correction ciblée, puis retester.

9. Peut-on partager un jeton dans un prompt ?

Corrigé : Non. Les secrets doivent rester hors des échanges et du dépôt.

10. Que prouve un commit ?

Corrigé : Qu'un état a été enregistré, pas qu'il est exempt de défauts.

13. Épreuve d'autonomie (sans IA) — 60 minutes

À partir du dossier pratique : 1) expliquer le rôle des trois fichiers ; 2) ajouter un bouton « Afficher une astuce » sans agent ; 3) tester au clavier et sur écran étroit ; 4) décrire un diff ; 5) écrire une note de validation de 150 mots. Barème /20 : compréhension (4), réalisation (6), tests (5), revue et justification (5). Réussite conseillée : 14/20 avec interaction fonctionnelle et sans régression majeure.

MODULE 8 — Publication web : Netlify et alternatives (8 heures)

Bloc B — Développer et publier | Prérequis : modules 1 à 7 | Projet fil rouge : mon-premier-projet. Les menus des services évoluent : vérifier les intitulés et les possibilités disponibles au moment de l'exercice.

1. Objectifs et critères de réussite

• Expliquer la différence entre fichiers locaux, dépôt GitHub, hébergement, URL et domaine.

• Publier une version stable du site HTML/CSS/JavaScript sur un hébergeur adapté, ou documenter une simulation vérifiable si la publication est inaccessible.

• Distinguer publication manuelle et déploiement lié à GitHub, sans mélanger les deux procédures.

• Identifier le statut d'un déploiement et diagnostiquer une erreur simple.

• Expliquer le rôle de HTTPS et du DNS sans confondre chiffrement et fiabilité du contenu.

• Retrouver un déploiement antérieur et choisir une stratégie de restauration contrôlée.

2. Découpage horaire

Séquence 1 — Comprendre hébergement, URL, domaine et DNS (1 h).

Séquence 2 — Vérifier le projet et créer un site Netlify (1 h).

Séquence 3 — Déployer depuis GitHub (1 h 30).

Séquence 4 — Contrôler le déploiement, les liens et HTTPS (1 h).

Séquence 5 — Domaine personnalisé et configuration DNS (1 h).

Séquence 6 — Historique, restauration et diagnostic (1 h).

Séquence 7 — TP autonome et évaluation (1 h 30).

3. Les notions fondamentales

Le navigateur affiche une page demandée à une adresse (URL). Le dépôt GitHub conserve le code et son historique ; Netlify distribue les fichiers publiés. Un déploiement est une version du site mise à disposition. Un domaine personnalisé est un nom que l'on possède et configure ; un sous-domaine fourni par Netlify suffit pour ce cours. Le DNS relie les noms aux services selon les enregistrements configurés. HTTPS protège les échanges avec le site, mais ne garantit ni la justesse du contenu ni l'absence de bugs.

Pour notre site statique, index.html est le point d'entrée. Aucune compilation n'est nécessaire : la commande de build peut rester vide et le répertoire de publication est celui contenant index.html (souvent la racine du dépôt). Pour d'autres frameworks, ces réglages peuvent différer.

4. Préparer le dépôt avant publication

☐ Ouvrir index.html localement et vérifier le bouton, le message et la mise en page.

☐ Confirmer que les chemins vers style.css et script.js sont relatifs et corrects.

☐ Lire git status et vérifier que les changements utiles sont enregistrés et poussés sur GitHub.

☐ Vérifier qu'aucune donnée personnelle, clé secrète ou fichier temporaire n'est public.

☐ Identifier la branche stable, généralement main, et relever le dernier commit.

```
git status

git log --oneline -3

git remote -v
```

Ces commandes permettent de lire l'état et l'origine du dépôt ; elles ne publient rien. La présence du dernier commit sur GitHub doit aussi être vérifiée dans l'interface.

5. Deux modes de publication à distinguer

Mode A — Déploiement manuel : importer le dossier contenant index.html dans l'interface Netlify. Utile pour une démonstration, mais une modification sur GitHub ne met pas automatiquement à jour ce déploiement manuel.

Mode B — Déploiement continu : dans Netlify, choisir l'ajout d'un nouveau projet depuis un fournisseur Git, autoriser uniquement les dépôts nécessaires, sélectionner le dépôt pédagogique et la branche stable. Pour ce projet statique : commande de build vide ; dossier de publication « . » si index.html se trouve à la racine. Vérifier les paramètres avant de lancer le déploiement.

Le mode B illustre le déploiement continu ; un déploiement manuel ou une autre plateforme répond aussi à l'objectif si les vérifications sont identiques. L'interface et les quotas peuvent varier selon le compte et la période.

6. Première publication pas à pas

Étape 1 — Se connecter à Netlify et sélectionner la création d'un projet à partir d'un dépôt Git.

Étape 2 — Connecter GitHub après lecture des permissions demandées ; sélectionner mon-premier-projet.

Étape 3 — Contrôler la branche de publication, la commande de build (aucune) et le répertoire publié (racine).

Étape 4 — Lancer le déploiement puis attendre un statut de réussite, sans déduire la réussite du seul clic.

Étape 5 — Ouvrir l'URL fournie par Netlify dans un navigateur privé et tester le site.

Étape 6 — Noter dans une fiche : URL, date, branche, identifiant de commit et résultat des tests.

7. Vérifier un site publié

☐ L'URL s'ouvre sur ordinateur et sur téléphone.

☐ Le titre, les styles CSS et les interactions JavaScript sont présents.

☐ Les liens ne conduisent pas vers des chemins locaux du type C:\Utilisateurs\...

☐ Le bouton fonctionne avec la souris et au clavier (Tab, Entrée/Espace).

☐ Le site ne présente pas de défilement horizontal inattendu.

☐ L'adresse utilise https:// et le navigateur ne signale pas d'erreur de certificat.

☐ La console et les requêtes réseau ne signalent pas de ressources essentielles manquantes.

8. Domaine personnalisé : comprendre avant de modifier

Un domaine personnalisé est facultatif. Si l'apprenant en possède un, l'ajouter dans les paramètres de domaines Netlify puis appliquer exactement les enregistrements DNS indiqués pour sa configuration. Selon le cas, cela implique des enregistrements CNAME, A ou d'autres réglages. Ne pas recopier d'adresse IP ou de valeur DNS générique depuis un cours : elles peuvent changer. Les

changements DNS peuvent demander un délai de propagation. Attendre la validation du domaine et du certificat HTTPS avant de communiquer l'adresse.

Ne pas modifier le DNS d'un site réel de production pour un TP. Utiliser un domaine de test ou rester sur le sous-domaine Netlify.

9. Comprendre les déploiements et les retours arrière

Dans l'historique des déploiements Netlify, consulter le statut, la source Git et les journaux. Si une nouvelle version est défectueuse, conserver les informations sur le problème et examiner les options de restauration disponibles dans l'interface. Selon le type de déploiement et les réglages, un déploiement antérieur peut être republié ; sur un site relié à Git, il faut aussi corriger ou annuler le commit responsable pour éviter qu'une prochaine publication ne réintroduise le défaut.

Privilégier une correction Git traçable, par exemple git revert sur un commit identifié, après vérification. Ne jamais réécrire l'historique partagé sans comprendre les conséquences.

10. Dépannage : quatre cas fréquents

Page introuvable

Cause possible : Le répertoire publié ne contient pas index.html à l'endroit attendu.

Vérification : Vérifier l'arborescence, la casse des noms et le répertoire de publication.

Page sans styles

Cause possible : Chemin CSS incorrect ou fichier absent du dépôt.

Vérification : Ouvrir les outils réseau du navigateur et vérifier href et nom du fichier.

Bouton inactif

Cause possible : script.js non chargé ou erreur JavaScript.

Vérification : Contrôler la console, l'attribut defer et les identifiants HTML.

Nouvelle version non visible

Cause possible : Déploiement non déclenché, échoué, branche incorrecte ou cache.

Vérification : Comparer commit GitHub et source du déploiement ; vérifier le statut puis actualiser.

11. Travaux pratiques

TP1 — Schématiser le trajet

Consigne : Dessiner ordinateur → dépôt local ou distant → hébergeur choisi → navigateur, puis expliquer chaque flèche ; une publication manuelle peut supprimer l'étape du dépôt distant.

Livrable : Schéma légendé.

TP2 — Préparer le site

Consigne : Examiner les trois fichiers, tester localement et rédiger une checklist.

Livrable : Checklist datée.

TP3 — Publier

Consigne : Si un compte Netlify et les autorisations nécessaires sont disponibles, connecter le dépôt pédagogique et publier sans commande de build. Sinon, effectuer la simulation de publication à partir d'une capture ou d'une fiche de paramètres fictifs, sans prétendre qu'un site est réellement publié.

Livrable : URL et commit réellement relevés en cas de publication ; sinon, fiche de simulation explicitement identifiée comme telle.

TP4 — Diagnostiquer

Consigne : Dans une copie d'exercice, casser le chemin vers style.css, constater le problème puis corriger.

Livrable : Cause, correction et nouveau test.

TP5 — Revenir à une version saine

Consigne : Créer une modification réversible sur une branche de test, examiner l'historique et expliquer le retour arrière.

Livrable : Note de décision sans perte de travail.

12. Étude de cas — Récits Vivants

Pour une page de découverte accessible par QR code, une URL stable est importante : modifier la page sans changer l'adresse publique évite de rendre obsolètes des supports imprimés. Vérifier les images, les chemins, les métadonnées et la page sur téléphone. Distinguer une erreur de contenu, une erreur de déploiement et une erreur d'indexation : elles n'ont pas les mêmes causes ni les mêmes solutions.

13. Autoévaluation corrigée

1. GitHub héberge-t-il automatiquement le site sur Netlify ?

Corrigé : Non, il faut configurer un déploiement ou importer les fichiers.

2. Qu'est-ce qu'un déploiement ?

Corrigé : Une version du site rendue accessible sur un hébergement.

3. Faut-il un build pour notre HTML statique ?

Corrigé : Non, pas pour ces trois fichiers simples.

4. Où doit se trouver index.html ?

Corrigé : Dans le dossier effectivement publié.

5. À quoi sert HTTPS ?

Corrigé : À chiffrer et authentifier la connexion selon le certificat.

6. Le domaine personnalisé est-il obligatoire ?

Corrigé : Non, le sous-domaine Netlify convient.

7. Pourquoi consulter les journaux ?

Corrigé : Pour identifier la cause d’un déploiement échoué.

8. Pourquoi tester sur mobile ?

Corrigé : Pour détecter les défauts responsive et d’interaction.

9. Pourquoi relever le commit publié ?

Corrigé : Pour relier précisément la version visible au code source.

10. Un rollback Netlify suffit-il toujours ?

Corrigé : Non, une prochaine publication Git peut réintroduire le problème.

14. Épreuve d’autonomie — 90 minutes

Sans assistance de l’IA, deux parcours équivalents sont proposés selon les accès disponibles. Parcours A (compte et autorisations disponibles) : publier le projet depuis GitHub sur un site de test ; relever l’URL et le commit réellement publiés ; vérifier cinq comportements ; identifier une erreur de chemin simulée ; expliquer comment revenir à une version saine. Parcours B (sans compte ou autorisation) : à partir d’une fiche de paramètres fictifs et du projet local, décrire la configuration GitHub–Netlify, produire une checklist de cinq tests avec résultats locaux réellement observés, diagnostiquer une erreur de chemin dans une copie et expliquer le retour arrière ; inscrire explicitement « simulation — aucune publication réalisée », sans inventer d’URL ni de commit publié. Barème commun /20 : compréhension (4), configuration réelle ou simulée correctement identifiée (5), tests réels locaux ou publiés (5), diagnostic (3), retour arrière et traçabilité (3). Seuil conseillé : 14/20 et aucune exposition de secrets. La publication réelle n’est attestée que pour le parcours A.

15. Bilan et transition

L’apprenant peut expliquer la chaîne de publication et ne confond plus « code enregistré » et « site en ligne ». Module 9 : plugins, connecteurs, permissions et outils d’intégration ; la connexion à un service ne remplace jamais le contrôle des actions autorisées.

MODULE 9 — Plugins, connecteurs et écosystème IA

Durée : 8 heures | Niveau : débutants complets | Bloc B : Développer et publier

Compétences visées

- Distinguer une application ChatGPT, une intégration entre services, un outil Codex et une automatisation.
- Identifier les données consultables et les actions permises par une connexion.
- Appliquer le principe du moindre privilège et savoir révoquer un accès.
- Comparer les parcours gratuits et payants sans présumer de quotas fixes.
- Contrôler une modification de code avant sa publication.

9.1 — Comprendre l'écosystème (1 h)

Une application ou un connecteur étend les possibilités de ChatGPT vers un service externe. Les fonctions exactes dépendent du service, du compte et des autorisations accordées.

Un outil Codex permet à un agent de travailler sur du code ou un environnement de développement. Une intégration GitHub–Netlify permet de déclencher des déploiements à partir d'un dépôt. Ces mécanismes ne sont pas interchangeables.

Exemple : lire un fichier du dépôt ne donne pas nécessairement le droit de modifier la branche principale ou de publier le site.

9.2 — GitHub, Codex et chaîne de travail (1 h 30)

Rappel du projet : index.html définit le contenu, style.css la présentation, script.js les interactions et README.md explique le projet.

Méthode : ouvrir une branche de travail, préciser les fichiers à modifier, demander une proposition à Codex, examiner le diff, tester localement, puis décider si la modification doit être fusionnée.

Une connexion à GitHub peut être limitée à certains dépôts. Il faut vérifier dans les paramètres du service connecté les autorisations réellement consenties.

Une demande utile : « Dans une branche séparée, propose une amélioration de l'accessibilité du bouton d'astuce. Ne modifie pas les autres fonctionnalités. Explique chaque changement et indique les tests à réaliser. »

9.3 — Permissions et accès aux fichiers (1 h 30)

Lecture : l'outil peut consulter les données autorisées. Écriture : il peut modifier des ressources. Administration : il peut parfois modifier des paramètres ou autorisations. Les intitulés varient selon le fournisseur.

Vérifier avant connexion : identité du fournisseur, liste des données accessibles, périmètre des dépôts, capacité d'écriture, modalités de retrait de l'accès.

Ne jamais donner un accès général à tous les dépôts si un seul dépôt suffit. Ne jamais accepter une action de suppression ou publication sans comprendre ses conséquences.

Exercice guidé : rédiger une fiche d'autorisation pour un connecteur hypothétique, en séparant ce qui est nécessaire, facultatif et interdit.

9.4 — Sécurité, secrets et confidentialité (1 h 30)

Une clé API, un jeton d'accès ou un mot de passe est un secret : il ne doit jamais être placé dans index.html, script.js, un prompt public, une capture d'écran ou un dépôt GitHub.

Utiliser les paramètres sécurisés prévus par les services pour les secrets. Un fichier .gitignore empêche le suivi futur de fichiers correspondants, mais ne retire pas un secret déjà publié dans l'historique.

Si un secret a été publié : le révoquer ou le faire tourner immédiatement, puis traiter l'historique et examiner les accès.

Examiner aussi les risques d'instructions malveillantes dans les contenus externes : un fichier README ou une page web ne doit pas pouvoir imposer des actions non autorisées à l'agent.

Sauvegarder le projet avant toute intervention majeure ; vérifier les changements avant fusion et publication.

9.5 — Gratuit, payant et quotas (1 h)

Distinguer : accès au service, capacité de connecter une application, nombre de requêtes, fonctionnalités d'agent, minutes de déploiement et ressources d'hébergement.

Ces conditions évoluent. Ne pas annoncer qu'un outil est illimité ou gratuit pour toujours. Consulter les pages officielles de tarification et les limites affichées dans son compte.

Parcours pédagogique de repli : lecture manuelle du dépôt GitHub, copie locale des fichiers, demande de conseils à l'IA, modification contrôlée, puis déploiement manuel Netlify lorsque disponible.

Une formation solide doit rester réalisable même si certaines fonctions d'agent ou de connecteur sont absentes.

9.6 — TP et évaluation autonome (1 h 30)

TP 1 (15 min) : classer dix exemples d'outils en application, agent de code, intégration et automatisation.

Jeu de dix exemples fictifs à classer : (1) application de recherche documentaire dans ChatGPT ; (2) agent Codex chargé de proposer un correctif ; (3) liaison GitHub–Netlify déclenchant un

déploiement ; (4) rappel hebdomadaire programmé ; (5) application de consultation d'un calendrier ; (6) agent chargé de modifier un fichier CSS ; (7) liaison entre un dépôt et un service de tests ; (8) rapport envoyé automatiquement chaque lundi ; (9) application de consultation de fichiers ; (10) agent chargé de préparer un diff. Corrigé : applications 1, 5, 9 ; agents 2, 6, 10 ; intégrations 3, 7 ; automatisations 4, 8. Certaines solutions réelles peuvent combiner plusieurs catégories : classer ici selon la fonction décrite.

TP 2 (20 min) : analyser une demande d'autorisation fictive ; cocher les permissions excessives et proposer une version minimale.

Cas fictif : un assistant doit uniquement lire le README d'un dépôt pédagogique pour en expliquer les consignes. Autorisations demandées : lecture du dépôt ciblé ; écriture sur tous les dépôts ; suppression de dépôts ; gestion des membres de l'organisation ; accès aux dépôts privés sans rapport avec le cours. Corrigé attendu : seule la lecture du dépôt ciblé est nécessaire ; toutes les autres permissions sont excessives. Si le fournisseur ne permet pas cette limitation, renoncer à la connexion et fournir un extrait non sensible du README.

TP 3 (20 min) : écrire un prompt de modification ciblée pour le site fil rouge, avec interdiction de toucher au stockage local.

TP 4 (20 min) : lire un diff fictif et détecter une modification non demandée.

Diff fictif à analyser (les lignes précédées de - sont retirées ; celles précédées de + sont ajoutées) :

- <button id="astuce">Afficher une astuce</button>

+ <button id="astuce" type="button">Afficher une astuce</button>

- localStorage.removeItem("prenom");

+ localStorage.clear();

Consigne : l'objectif autorisé est uniquement de préciser le type du bouton. Repérer la modification hors périmètre. Corrigé : l'ajout de type="button" est conforme ; remplacer removeItem("prenom") par clear() est non demandé et peut effacer toutes les données localStorage de cette origine. Refuser cette seconde modification et retester la mémorisation du prénom.

Évaluation (15 min) : expliquer oralement le circuit de validation et décrire la procédure de révocation d'un accès.

Étude de cas — Récits Vivants

Situation : une page de découverte doit être mise à jour sans casser son QR code ni sa navigation mobile. L'apprenant identifie le dépôt concerné, limite les fichiers autorisés, fait réaliser une proposition sur une branche, teste les liens et le responsive, puis valide la publication. Une capture de la version antérieure et un commit repérable facilitent le retour arrière.

Grille de vérification avant autorisation

☐ Le fournisseur est identifié.

☐ Le périmètre des données est connu.

☐ Les droits de lecture et d’écriture sont distingués.

☐ Les accès sont limités au projet nécessaire.

☐ Les secrets restent hors du code et des prompts.

☐ La méthode de révocation est connue.

☐ Un retour arrière est possible.

Autoévaluation — 10 questions

1. Que fait un connecteur ?

2. Une lecture GitHub autorise-t-elle forcément une écriture ?

3. Qu’est-ce que le moindre privilège ?

4. Pourquoi créer une branche ?

5. Qu’est-ce qu’un diff ?

6. Peut-on publier une clé API dans script.js ?

7. Un .gitignore efface-t-il un secret déjà commité ?

8. Pourquoi tester après une modification de Codex ?

9. Les quotas gratuits sont-ils garantis à vie ?

10. Que faire si une connexion n’est plus nécessaire ?

Corrigé de l’autoévaluation

1. Il relie un assistant à un service selon des capacités et permissions définies.

2. Non.

3. N’accorder que les droits strictement nécessaires.

4. Pour isoler une modification avant validation.

5. La comparaison des modifications entre deux états du code.

6. Non.

7. Non.

8. Pour vérifier la demande et détecter les régressions.

9. Non, ils peuvent changer.

10. Révoquer son accès et vérifier les autorisations restantes.

Épreuve finale sans IA — 30 minutes

Sur papier ou dans un éditeur de texte, construire une fiche de connexion pour un outil fictif ayant accès à un dépôt GitHub. Indiquer : objectif, données nécessaires, permissions acceptées/refusées, risques, tests de validation et procédure de révocation. Expliquer ensuite la fiche en cinq minutes.

Critères de réussite

- Identifier correctement les quatre types d'outils (20 %).
- Définir des permissions minimales (25 %).
- Détecter et protéger les secrets (25 %).
- Décrire une validation et un retour arrière (20 %).
- Expliquer les limites des offres et quotas (10 %).

Ressources à actualiser

Documentation officielle ChatGPT : https://help.openai.com/

Documentation Codex : https://developers.openai.com/codex/

Documentation GitHub : https://docs.github.com/

Documentation Netlify : https://docs.netlify.com/

Note éditoriale : vérifier les interfaces, permissions, offres et quotas à chaque nouvelle session de formation (édition octobre 2026).

MODULE 10 — UX, ACCESSIBILITÉ, SEO ET QUALITÉ WEB

Durée : 10 heures | Bloc C — Concevoir une expérience autonome | Public : débutants complets

Finalité : transformer le site fil rouge en une expérience compréhensible, utilisable, accessible, découvrable et vérifiable. Les tests humains complètent les outils automatiques.

Compétences attendues

- Décrire un parcours utilisateur et repérer une difficulté de navigation.
- Contrôler la navigation au clavier, les libellés, les contrastes et les alternatives textuelles.
- Tester la présentation sur plusieurs largeurs d'écran sans perte de contenu.
- Expliquer le rôle des titres, métadonnées, liens, sitemap.xml et URL canonique.
- Contrôler les liens et les QR codes sur la destination réelle.
- Utiliser une grille de recette et documenter les anomalies et corrections.

Déroulé des 10 heures

- Séquence 1 — UX et parcours utilisateur : 1 h 30
- Séquence 2 — Accessibilité et clavier : 2 h
- Séquence 3 — Responsive et lisibilité : 1 h
- Séquence 4 — SEO et indexation : 2 h
- Séquence 5 — Performance, liens et QR codes : 1 h 30
- Séquence 6 — Recette et autonomie : 2 h

1. UX : partir du visiteur

L'expérience utilisateur (UX) désigne la manière dont une personne comprend et utilise un service. Avant de modifier le site, écrire trois tâches : découvrir le projet, trouver les informations pratiques, revenir à l'accueil. Pour chacune, compter les étapes et noter les hésitations. Un lien intitulé « En savoir plus » est moins explicite que « Découvrir la carte intérieure ». Un test simple consiste à demander à une personne de réaliser la tâche sans lui expliquer l'interface.

2. HTML sémantique et repères

Une page doit présenter un titre principal h1 pertinent, des sous-titres hiérarchisés et des zones structurées avec header, nav, main et footer. Le contenu reste lisible sans CSS. Les boutons déclenchent une action ; les liens conduisent à une ressource. Ne pas utiliser un div cliquable lorsque button ou a convient.

3. Accessibilité : test au clavier

Depuis le début de la page, utiliser Tab et Maj+Tab : chaque élément interactif doit être atteignable dans un ordre logique, avec un indicateur de focus visible. Entrée active un lien ou un bouton ; Espace active un bouton. Vérifier qu'aucun piège au clavier n'empêche de sortir d'un composant.

Un test clavier n’est pas un audit complet : vérifier aussi les lecteurs d’écran et les usages mobiles lorsque c’est possible.

4. Images, formulaires et contraste

Rédiger un texte alternatif décrivant la fonction ou l’information de l’image ; une image purement décorative peut avoir alt="". Associer chaque champ à un label. Ne pas transmettre une information uniquement par la couleur. Vérifier les contrastes : pour le texte courant, viser au moins 4,5:1 et pour le grand texte au moins 3:1 (WCAG AA), sous réserve des exceptions prévues par les critères.

5. Responsive : une seule page, plusieurs contextes

Tester environ 320, 375, 768 et 1280 pixels de largeur ; ces valeurs sont des points de contrôle, pas des catégories absolues d’appareils. Aucun défilement horizontal ne doit être nécessaire pour lire les contenus ordinaires. Vérifier les textes agrandis à 200 %, les menus, les images et les zones tactiles. Ne pas supprimer des informations importantes sur mobile.

6. SEO : rendre une page interprétable

Le référencement naturel commence par un contenu utile, un title spécifique, une meta description pertinente, des titres cohérents et des liens descriptifs. Une meta description n’impose pas le texte affiché dans les résultats. Un sitemap aide à découvrir des URL mais ne garantit pas leur indexation. robots.txt et la balise noindex ne remplissent pas la même fonction.

7. Canonical et indexation

La balise <link rel="canonical" href="https://exemple.fr/page/"> indique une URL préférée parmi des variantes ; c’est un signal, non une obligation pour Google. Vérifier que la destination est absolue, correcte, accessible et cohérente avec les liens internes et le sitemap. Après publication, utiliser l’inspection d’URL de Search Console pour examiner l’URL déclarée et l’URL choisie par Google. Une demande d’indexation n’est pas une garantie.

8. QR codes et liens

Un QR code encode une donnée, souvent une URL. Avant impression, le scanner avec au moins deux appareils ou applications ; vérifier l’URL affichée, la page d’arrivée, HTTPS, la lecture sur mobile et la pérennité du lien. Ne jamais conclure qu’un QR fonctionne seulement parce que le motif paraît correct. Prévoir une URL lisible imprimée comme solution de secours.

9. Performance et qualité

Observer la taille des images, la stabilité visuelle, les temps de chargement et les erreurs réseau. Utiliser les outils de développement du navigateur et Lighthouse comme aides au diagnostic ; leurs scores ne remplacent pas un test réel. Définir une priorité : défaut bloquant, défaut important, amélioration souhaitable. Corriger une anomalie à la fois, puis refaire les tests de non-régression.

10. Confidentialité et données locales

Ne pas envoyer d’informations sensibles à des services tiers sans nécessité. Les données enregistrées dans localStorage restent liées à une origine et peuvent être supprimées par l’utilisateur ; elles ne sont ni chiffrées par défaut ni une sauvegarde durable. Expliquer les mécanismes de mémorisation dans l’interface.

Exemple de code : repères et référencement (page autonome de démonstration, à ne pas substituer au projet fil rouge). Pour intégrer cet exercice dans le projet des modules 3 à 5, conserver les sections et les interactions existantes, ajouter uniquement le bouton id="astuce" dans la section Présentation, réutiliser l'unique paragraphe id="message" déjà présent et ajouter le gestionnaire JavaScript indiqué ci-dessous. Ne pas créer de second id="message".

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Mon premier projet — Présentation</title>
  <meta name="description" content="Découvrez un premier projet web interactif.">
  <!-- À remplacer par la vraie URL publique après déploiement -->
  <link rel="canonical" href="https://exemple.fr/">
  <link rel="stylesheet" href="style.css">
  <script src="script.js" defer></script>
</head>
<body>
  <a class="skip-link" href="#contenu">Aller au contenu</a>
  <header><nav aria-label="Navigation principale"><a href="index.html">Accueil</a></nav></header>
  <main id="contenu"><h1>Mon premier projet</h1><p>Une création numérique accessible.</p>
    <button type="button" id="astuce">Afficher une astuce</button>
    <p id="message" aria-live="polite"></p>
  </main>
  <footer><p>Projet pédagogique.</p></footer>
</body>
</html>
```

Vérification pratique : ajouter dans script.js, sans effacer les fonctions précédentes, ce gestionnaire autonome :

```
document.querySelector("#astuce")?.addEventListener("click", () => {
  const zone = document.querySelector("#message");
  if (zone) zone.textContent = "Une idée peut naître d'une simple observation.";
});
```

Recharger la page, activer le bouton au clavier et vérifier que le message apparaît. Le bouton ne fonctionne pas si ce gestionnaire n'est pas chargé.

Explication : lang="fr" indique la langue ; viewport permet la mise en page mobile ; title nomme la page ; canonical doit être adapté à l'URL effectivement publiée ; le lien d'évitement facilite l'accès au contenu ; aria-live annonce les mises à jour du message. Ne pas copier l'URL exemple.fr sur un site réel.

Travaux pratiques guidés

TP1 — Parcours utilisateur (30 min)
Faire réaliser à un pair trois tâches ; noter les clics, incompréhensions et pistes de correction. Si aucun pair n'est disponible, réaliser une inspection guidée des trois tâches et indiquer qu'il ne s'agit pas d'un test utilisateur. Livrable : tableau tâche / observation / amélioration.

TP2 — Clavier et sémantique (45 min)
Naviguer sans souris ; corriger les titres, les liens et l'indicateur de focus. Livrable : captures ou compte rendu des tests avant/après.

TP3 — Responsive (45 min)
Tester quatre largeurs et le zoom à 200 %. Corriger un débordement sans masquer le contenu. Livrable : grille de résultats.

TP4 — SEO et canonical (60 min)
Rédiger title et description. Si le site est publié, vérifier l'URL canonique réelle et la cohérence d'un éventuel sitemap ; inspecter l'URL avec Search Console uniquement si l'accès est disponible. Si le site n'est pas publié, préparer une proposition de canonical sans insérer d'URL fictive dans le projet et noter « URL publique non disponible — contrôle différé ». Le sitemap n'est pas obligatoire pour ce site à page unique. Livrable : fiche SEO distinguant vérifications réalisées et contrôles différés.

TP5 — QR et liens (30 min)
Tester trois liens existants du projet et, si un QR code est fourni ou généré pour une URL de test, le scanner et noter l'URL encodée et l'URL finale. Sans QR code disponible, préparer une fiche de contrôle décrivant la procédure et indiquer « scan non exécuté » ; ne pas inventer de résultat. Livrable : tableau de validation des liens et statut explicite du test QR.

TP6 — Recette finale (60 min)
Identifier trois anomalies de catégories différentes, les classer, corriger et retester. Livrable : rapport de recette daté.

Étude de cas : les pages de découverte des Récits Vivants
Situation : une page de découverte liée à une carte imprimée doit rester lisible sur mobile, accessible depuis un QR code et indexable. Le contrôle ne porte pas uniquement sur l'apparence : vérifier la destination exacte du QR, les liens, la balise canonique, la cohérence du sitemap et l'absence de contenu masqué. Ne pas attribuer automatiquement une absence d'indexation à la balise canonique : examiner le rapport d'inspection et les autres causes possibles.

Autoévaluation : 10 questions avec corrigés
7. 1. Quelle différence entre UX et interface graphique ?

Corrigé : L'UX couvre l'expérience d'utilisation ; l'interface graphique en est une composante.

8. 2. Comment tester sans souris ?

Corrigé : Tab, Maj+Tab, Entrée et Espace ; contrôler ordre, focus et absence de piège.

9. 3. À quoi sert alt ?

Corrigé : À transmettre l'information ou la fonction d'une image aux personnes qui ne la voient pas.

10. 4. Que signifie responsive ?

Corrigé : Une interface qui s'adapte aux contextes et tailles d'affichage.

11. 5. Un sitemap garantit-il l'indexation ?

Corrigé : Non, il facilite la découverte des URL.

12. 6. La balise canonical force-t-elle Google ?

Corrigé : Non, elle indique une préférence.

13. 7. Comment vérifier un QR code ?

Corrigé : Le scanner et contrôler l'URL réelle et la page de destination.

14. 8. Que mesure un audit automatique ?

Corrigé : Une partie des critères techniques ; il ne remplace pas les tests humains.

15. 9. Pourquoi conserver une grille de recette ?

Corrigé : Pour rendre les contrôles reproductibles et suivre les corrections.

16. 10. Que faire après une correction CSS ?

Corrigé : Retester le problème initial et les fonctions susceptibles d'être affectées.

Épreuve d'autonomie — 60 minutes

Sans IA : partir du projet du module 9. Produire une grille de 12 contrôles couvrant UX, clavier, responsive, SEO, liens et QR. Détecter au moins deux défauts réels ou démontrer leur absence par des tests. Corriger un défaut simple, documenter les modifications et expliquer oralement pourquoi la correction est adaptée. Barème : diagnostic 8 pts, correction 5 pts, non-régression 4 pts, explication 3 pts. Réussite proposée : 12/20 avec aucun défaut bloquant laissé sans signalement.

Références pour approfondir

- W3C WAI — Web Content Accessibility Guidelines (WCAG) : https://www.w3.org/WAI/standards-guidelines/wcag/
- MDN Web Docs — HTML, CSS et accessibilité : https://developer.mozilla.org/fr/
- Google Search Central — Documentation SEO : https://developers.google.com/search/docs?hl=fr
- Google Search Console : https://search.google.com/search-console/about

- web.dev — Performance et Core Web Vitals : https://web.dev/

Note pédagogique : les interfaces des plateformes et leurs procédures évoluent. Les captures d'écran et étapes de navigation devront être vérifiées lors de chaque édition du manuel.

MODULE 11 — CONCEVOIR UNE EXPÉRIENCE NUMÉRIQUE CRÉATIVE

8 heures • Bloc C — Créer et évaluer • Prérequis : modules 1 à 10

Édition pédagogique V2.2 — relecture en cours — Projet fil rouge : Mon espace de découverte

1. Finalité et compétences observables

L'apprenant conçoit une expérience interactive courte qui ne réduit pas le visiteur à une suite de clics obligatoires. Il sait décrire une intention artistique, représenter un parcours non linéaire, modéliser un état simple en JavaScript, utiliser une mémoire locale facultative, prévoir une sortie et une remise à zéro, vérifier l'accessibilité et justifier les limites de l'intervention de l'IA.

Livrable : une nouvelle section « Explorer » intégrée au site existant, accompagnée d'une fiche d'intention, d'un schéma des états, d'un journal de tests et d'une notice de confidentialité.

2. Répartition des huit heures

Séquence A — Intention, visiteur et expérience (1 h).

Séquence B — Parcours, embranchements et liberté de ne pas choisir (1 h 30).

Séquence C — État, événements et affichage JavaScript (2 h).

Séquence D — Mémoire locale, effacement et consentement pratique (1 h).

Séquence E — Accessibilité, éthique et non-régression (1 h).

Séquence F — Travaux pratiques, présentation et évaluation (1 h 30).

3. Du site fonctionnel au milieu d'exploration

Une interface permet d'agir ; une expérience propose aussi une situation à interpréter. Pour notre projet, trois cartes « Observer », « Relier » et « Imaginer » sont offertes sans classement, score, compte à rebours ni parcours imposé. Le visiteur peut ouvrir une carte, revenir au point de départ, ne rien sélectionner ou quitter la page.

Question de conception : qu'est-ce qui est rendu possible, et qu'est-ce qui serait imposé par erreur ? La fiche d'intention distingue ce que le créateur souhaite rendre sensible, ce qu'il refuse de mesurer et les interprétations qu'il laisse ouvertes.

Étude de cas Récits Vivants : le Milieu des possibles inspire une exploration où l'œuvre ne récompense pas le « bon » comportement et n'assimile pas une hésitation à un échec. Il s'agit d'une inspiration pédagogique, non d'une reproduction de l'œuvre.

4. Écrire une fiche d'intention

Compléter les six rubriques : (1) public visé ; (2) expérience proposée en une phrase ; (3) trois possibilités offertes ; (4) liberté de ne pas participer ; (5) informations éventuellement mémorisées ; (6) critères observables de qualité.

Exemple : « Le visiteur découvre trois propositions de lecture, dans l'ordre qu'il souhaite. Aucun choix n'est enregistré sans action explicite. Les contenus restent accessibles si JavaScript échoue. »

Critères de réussite : consignes compréhensibles, absence de piège, navigation clavier, affichage mobile, suppression facile des préférences.

5. Modéliser le parcours et les états

Un état représente la situation courante de l'interface. Dans l'exercice, l'état initial est « accueil » ; les états possibles sont « observer », « relier » et « imaginer ». Les transitions proviennent d'un bouton explicite ; « revenir » ramène à « accueil ». Il n'existe pas d'état « gagné » ou « perdu ».

Schéma textuel : ACCUEIL ↔️ OBSERVER ; ACCUEIL ↔️ RELIER ; ACCUEIL ↔️ IMAGINER. Chaque branche peut revenir à l'accueil. Une option « Effacer mon choix mémorisé » reste indépendante de la navigation.

Éviter d'utiliser la couleur seule pour signaler l'état courant. Le titre et le texte affichés doivent également l'indiquer.

6. Construire l'interaction, étape par étape

Dans index.html, ajouter dans main une section id="explorer" avec h2 « Explorer librement », un paragraphe expliquant le caractère facultatif de l'expérience, trois boutons type="button" dotés de data-choix="observer", "relier" et "imaginer", un bouton id="retour-exploration" et un paragraphe id="texte-exploration" role="status" aria-live="polite". Conserver les sections existantes et leurs identifiants.

Préparation indispensable : dans la section #explorer, créer les trois boutons avec data-choix, un paragraphe Vous pouvez explorer librement. et un bouton <button type="button" id="retour-exploration">Revenir</button>. Pour un fonctionnement sans JavaScript, afficher aussi les trois propositions dans une liste HTML statique ; le script enrichit l'expérience mais ne doit pas rendre ces textes inaccessibles. Les identifiants doivent correspondre exactement à ceux du script. Placer le script dans script.js chargé avec defer, puis tester sans remplacer les autres interactions.

Extrait JavaScript à ajouter sans remplacer les fonctions déjà présentes :

```
const propositions = {
  observer: "Que remarquez-vous avant de chercher une explication ?",
  relier: "Quels liens inattendus pouvez-vous imaginer ?",
  imaginer: "Quelle autre possibilité pourrait exister ?"
};
const sortie = document.querySelector("#texte-exploration");
document.querySelectorAll("[data-choix]").forEach((bouton) => {
  bouton.addEventListener("click", () => {
    const cle = bouton.dataset.choix;
    if (Object.hasOwn(propositions, cle)) {
      sortie.textContent = propositions[cle];
    }
  });
});
document.querySelector("#retour-exploration").addEventListener("click", () => {
  sortie.textContent = "Vous pouvez explorer, attendre ou partir librement.";
});
```

Explication : l'objet propositions associe des identifiants à des textes ; dataset lit le choix ; l'événement click réagit au bouton ; textContent affiche une chaîne comme du texte, sans injecter de HTML. Object.hasOwn vérifie que la clé existe. Tester l'extrait uniquement après avoir ajouté les éléments HTML correspondants. En cas d'erreur, contrôler la console et les sélecteurs.

7. Mémoire locale facultative et effacement

Par défaut, l'exercice ne mémorise aucun choix : naviguer dans une expérience ne doit pas impliquer un enregistrement. Variante facultative : ajouter un bouton « Mémoriser cette préférence » et ne lancer localStorage.setItem("exploration-preference", cle) qu'après son activation. Un autre bouton exécute localStorage.removeItem("exploration-preference"). Prévoir try/catch : le stockage peut être désactivé. Ne jamais enregistrer de données sensibles ni utiliser localStorage comme preuve d'identité ou sauvegarde entre appareils.

L'apprenant doit pouvoir expliquer ce qui est conservé, où cela se trouve et comment l'effacer.

8. Accessibilité et qualité de l'expérience

Les choix sont des boutons HTML réels, activables avec Entrée et Espace. Le focus reste visible ; le texte de chaque action est compréhensible hors contexte ; les mises à jour sont annoncées sans interrompre excessivement la lecture. Vérifier la page sans JavaScript : les trois propositions doivent être accessibles en texte statique ou par une solution de repli. Respecter prefers-reduced-motion si une animation est ajoutée ; ne jamais rendre l'animation nécessaire pour comprendre le contenu.

Tester à 375, 768 et 1280 px, au zoom 200 %, au clavier, et après rechargement. Vérifier que les interactions du module 5 fonctionnent toujours.

9. Cinq travaux pratiques

TP 1 (20 min) — Rédiger la fiche d'intention et préciser trois libertés du visiteur. Livrable : six rubriques remplies.

TP 2 (25 min) — Dessiner les quatre états et les transitions, avec retour possible depuis chaque branche. Livrable : schéma lisible.

TP 3 (35 min) — Ajouter les boutons et l'affichage ; expliquer à voix haute chaque partie du JavaScript. Livrable : site fonctionnel.

TP 4 (25 min) — Par défaut, vérifier qu'aucun choix d'exploration n'est mémorisé. Variante facultative, uniquement si les boutons de mémorisation et d'effacement ont été implémentés : tester l'enregistrement après action explicite, la suppression ciblée et l'indisponibilité du stockage (par simulation contrôlée ou avec l'aide du formateur). Ne pas attribuer un test réussi à une variante non implémentée. Livrable : observation du comportement par défaut et, si la variante existe, trois observations documentées.

TP 5 (25 min) — Effectuer une recette clavier/mobile/sans script et réparer une régression éventuelle. Livrable : tableau « test / attendu / observé / correction ».

10. Autoévaluation : dix questions

1. Quelle différence entre interface et expérience ? 2. Qu'est-ce qu'un état ? 3. Pourquoi proposer un retour ? 4. Que représente une transition ? 5. Pourquoi éviter un score imposé ? 6. Que lit dataset ? 7. Pourquoi utiliser textContent ? 8. Pourquoi la mémorisation doit-elle être facultative ? 9. Que vérifier au clavier ? 10. Comment démontrer l'absence de régression ?

11. Corrigé synthétique

1. L'interface permet des actions ; l'expérience inclut la manière dont elles sont vécues et interprétées. 2. Une situation courante de l'interface. 3. Pour éviter un parcours captif. 4. Le passage entre deux états. 5. Il impose une hiérarchie qui peut contredire l'intention. 6. Les attributs data-*

d’un élément. 7. Pour afficher sans interpréter du HTML. 8. Pour préserver le choix et la confidentialité. 9. Ordre de tabulation, activation, focus et retour. 10. Retester les fonctionnalités déjà validées et conserver un journal de recette.

12. Épreuve d’autonomie et validation

En 45 minutes, sans génération de code par IA, modifier le texte d’une proposition, ajouter un quatrième bouton cohérent, expliquer la transition correspondante et tester le retour. Barème sur 20 : intention (4), fonctionnement (5), accessibilité (4), confidentialité (3), tests et explication (4). Seuil indicatif : 12/20, sans défaut bloquant de navigation. Conserver un commit Git identifié et une capture des tests. Transition : le module 12 rassemble conception, développement, publication, contrôle et soutenance.

MODULE 12 — PROJET INTÉGRATEUR ET SOUTENANCE

8 heures • Bloc C — Réalisation autonome • Prérequis : modules 1 à 11

Édition pédagogique V2.2 — relecture en cours — Achèvement des 120 heures

1. Finalité et résultat attendu

Réaliser une version complète et maintenable de « Mon espace de découverte » : site HTML/CSS/JavaScript accessible, responsive, doté d’une interaction créative, suivi par une méthode de versionnement identifiable et publié via un hébergeur adapté, lorsque l’accès le permet. L’apprenant doit pouvoir présenter les décisions, les limites, les vérifications et les apports réels de l’IA.

Livrables : dépôt GitHub identifié si l’accès est disponible ; sinon, dossier local versionné ou dossier de simulation explicitement identifié, sans prétendre disposer d’un dépôt distant ; URL publique HTTPS si la publication est possible, ou dossier de simulation clairement identifié en cas d’absence d’accès ; fichiers index.html, style.css, script.js et README.md ; fiche de conception ; journal de tests ; journal d’assistance IA ; courte soutenance. Aucune donnée personnelle réelle n’est nécessaire.

2. Répartition des huit heures

Séquence A — Cadrage final et critères d’acceptation (1 h).

Séquence B — Intégration et corrections ciblées (2 h).

Séquence C — Tests UX, accessibilité, sécurité, SEO et performance (1 h 30).

Séquence D — Versionnement, publication et vérification sur l’hébergeur choisi (1 h).

Séquence E — Documentation et préparation de la soutenance (1 h).

Séquence F — Démonstration, évaluation et bilan (1 h 30).

3. Cahier des charges minimal

Le site comporte un titre principal, une navigation interne, trois cartes de contenu, une section « Explorer » à choix libres, un pied de page, des styles adaptés aux écrans étroits et une interaction JavaScript compréhensible. Les liens doivent fonctionner. Les textes restent lisibles sans animation et, autant que possible, sans script.

Le site n'a pas besoin de compte visiteur, de collecte de données, d'API externe, de formulaire opérationnel ni de service payant. Toute fonctionnalité supplémentaire doit être justifiée et testée ; la simplicité est une qualité, non un manque.

Critères d'acceptation : pour une publication réelle, ouverture de la page publique, HTTPS actif et vérification du commit déployé ; dans tous les cas, absence de ressource manquante dans le projet testé, fonctionnement au clavier, lisibilité mobile et absence d'erreur JavaScript bloquante. Pour une simulation, documenter les paramètres de déploiement et expliquer comment revenir à une version antérieure, sans présenter la simulation comme une publication effective.

4. Préparer le projet et les preuves

Arborescence recommandée :

mon-premier-projet/

index.html

style.css

script.js

README.md

images/ (facultatif)

docs/

conception.md

recette.md

journal-ia.md

Dans conception.md : public, objectif, fonctionnalités, exclusions, accessibilité et confidentialité. Dans recette.md : date, appareil ou largeur, action, résultat attendu, résultat observé, correction et nouveau test. Dans journal-ia.md : demande formulée, réponse utile ou rejetée, modifications réellement retenues, vérification effectuée.

Ne jamais placer de mot de passe, jeton d'accès ou clé API dans ces fichiers.

5. Intégration sans régression

Partir du dernier commit fonctionnel. Faire une modification cohérente à la fois, inspecter le diff et vérifier qu'aucune section antérieure n'a disparu. Si une correction concerne l'exploration, ne pas réécrire inutilement la navigation ou le stockage du module 5.

Méthode : constater → isoler → proposer → modifier → tester → comparer → enregistrer. Lorsqu'une suggestion IA est utilisée, demander une modification ciblée et une explication des conséquences. Refuser toute correction non comprise ou non testée.

Exemple d'incident : un bouton « Effacer la préférence d'exploration » supprime toutes les données localStorage. Correction attendue : removeItem avec la clé exacte, plutôt que clear. Vérifier les autres fonctions mémorisées.

6. Recette qualité en six familles

Fonctionnel : boutons, retour, liens, affichage sans erreur et suppression ciblée.

Responsive : contrôles à 375, 768 et 1280 px, puis zoom 200 % sans débordement gênant.

Accessibilité : structure de titres, libellés, alternatives d'images, focus visible, navigation clavier, absence d'information portée par la seule couleur.

SEO : titre de page spécifique, meta description utile, canonical uniquement si l'URL de référence est connue et cohérente, sitemap uniquement si nécessaire ; l'indexation n'est jamais garantie.

Performance : images dimensionnées et compressées, absence de bibliothèques inutiles, chargement raisonnable sur réseau lent.

Sécurité et confidentialité : aucun secret dans le dépôt ; aucune donnée sensible dans le navigateur ; permissions des applications limitées ; aucune publication non autorisée.

Pour chaque famille, conserver au moins un test avec résultat observé. Une grille remplie sans test réel n'est pas une preuve.

7. Publier et contrôler

Vérifier la branche et le commit à publier. Déployer avec la méthode disponible (transfert manuel ou dépôt connecté) sur un hébergeur adapté ; Netlify et GitHub constituent un exemple et non une obligation. Pour un site statique simple, confirmer le dossier contenant index.html et les paramètres de publication adaptés au projet ; ne pas inventer de commande de compilation. Consulter le résultat du déploiement et les éventuels messages d'erreur.

Ouvrir l'URL HTTPS dans une fenêtre privée et sur un téléphone si possible. Vérifier liens, CSS, JavaScript et chemins sensibles aux majuscules. Tester un lien depuis un QR code si le projet en utilise un. Si la version publiée échoue, identifier le dernier déploiement valide et utiliser la procédure de restauration disponible dans l'interface, puis vérifier à nouveau le site public.

Le nom de domaine personnalisé reste facultatif ; un domaine peut nécessiter un achat et une configuration DNS indépendante de la formation.

8. README : documenter pour une autre personne

Le README.md doit répondre à : (1) quel est le projet ? (2) comment l'ouvrir localement ? (3) quels fichiers ont quels rôles ? (4) comment tester l'interaction ? (5) où trouver la version publiée ? (6) quelles données sont conservées ? (7) comment signaler une anomalie ? (8) quelle version constitue la référence ?

Exemple : « Ouvrir index.html dans un navigateur. Certaines fonctions de stockage peuvent varier en file:// ; utiliser un serveur local de développement si nécessaire. Aucun compte visiteur requis. »

Une documentation utile décrit le comportement réel, pas seulement les intentions.

9. Soutenance courte : démontrer et expliquer

Présentation de 6 à 8 minutes suivie de questions. Plan : intention (1 min), architecture des fichiers (1 min), démonstration du site et de l'exploration (2 min), exemple d'erreur corrigée et preuve de test (1 min), versionnement/publication et retour arrière (1 min), limites et pistes d'évolution (1 min).

Questions possibles : que se passe-t-il si JavaScript est désactivé ? Pourquoi le stockage est-il facultatif ? Comment vérifier un lien QR ? Que fait le commit ? Comment savoir si la version publique est la bonne ? Qu'avez-vous accepté ou refusé parmi les propositions de l'IA ?

Le formateur évalue la compréhension et les preuves, pas l'effet spectaculaire.

10. Barème final sur 100 points

Conception et adéquation au besoin : 20 points.

HTML, CSS, JavaScript et qualité du code : 25 points.

Usage critique et traçable de l'IA : 20 points.

Publication, accessibilité et tests : 20 points.

Documentation et soutenance : 15 points.

Seuil pédagogique proposé : 60/100, avec remédiation obligatoire pour tout défaut bloquant de sécurité ou d'accessibilité de base. Ce barème est interne à la formation et ne constitue pas une certification officielle.

11. Autoévaluation : dix questions

1. Quel est le livrable principal ? 2. Pourquoi conserver un README ? 3. Que vérifie un test de non-régression ? 4. Pourquoi un site publié n'est-il pas forcément indexé ? 5. Quelle preuve permet de relier une version à GitHub ? 6. Quel est le rôle de HTTPS ? 7. Que faire avant une modification générée par IA ? 8. Comment vérifier une restauration ? 9. Pourquoi conserver un journal de recette ? 10. Que doit montrer la soutenance ?

12. Corrigé synthétique

1. Un site complet et documenté, publié si les accès sont disponibles ou accompagné d'une simulation de publication explicitement identifiée. 2. Pour permettre sa reprise et sa maintenance. 3. Les fonctions antérieures après modification. 4. Les moteurs décident de l'exploration et de l'indexation. 5. L'identifiant du commit. 6. Chiffrer la communication, sans garantir la fiabilité du contenu. 7. Définir la portée et garder un état stable. 8. Ouvrir et tester la version réellement servie. 9. Pour conserver des preuves et faciliter le diagnostic. 10. L'intention, les choix, les tests, les limites et l'autonomie.

13. Bilan des compétences et poursuite

Au terme des 120 heures, l'apprenant a rencontré l'ensemble de la chaîne : cadrage, assistance IA au choix, HTML, CSS, JavaScript, gestion de versions, développement assisté, publication web, connecteurs facultatifs, UX, accessibilité, SEO et expérience interactive. L'acquisition réelle dépend des exercices effectués et des évaluations ; le volume horaire seul ne prouve pas la maîtrise.

Pistes d'approfondissement : tests automatisés, design system, gestion de contenu, JavaScript avancé, RGPD, performance et accessibilité approfondies. Une amélioration future doit commencer par une intention et un critère de réussite, puis être soumise à la vérification.

FIN DU PARCOURS — 120 HEURES / 12 MODULES