

Utiliser les fonctionnalités git et le débogage

Table des matières

I. Gérer ses projets git dans Visual Studio Code	3
A. Initialiser un projet ou récupérer un projet	3
B. Gérer son code	4
C. Visualiser les différences et résoudre les conflits de merge	10
D. Extensions git blame et gitlens	10
II. Exercices	11
A. Exercice : Quiz	11
III. Déboguer avec VSCode	11
A. Définir des points d'arrêt et commencer à debugger	12
B. Parcourir le code pas à pas avec les différents commandes d'exécution	12
IV. Exercice :	12
V. Auto-évaluation	13
A. Exercice :	13
B. Test	13
Solutions des exercices	15

I. Gérer ses projets git dans Visual Studio Code

- **Environnement de travail : avoir déjà installer Visual Studio Code, connaître git et savoir l'utiliser**
- **Pré-requis :**
 - Avoir des bases de connaissance en développement
 - Connaître GIT

Contexte

Dans la vie d'un développeur l'utilisation d'un logiciel de versioning du code comme GIT est très important. Cela permet d'avoir un historique du code et surtout de pouvoir revenir à une version précédente si besoin. Il peut alors être très utile de savoir comment l'éditeur Visual Studio Code nous permet de travailler avec GIT et nous aide à déboguer votre code sans avoir à faire une ligne de commande.

Objectifs

Utiliser git directement dans l'éditeur de code.

Contexte

L'éditeur de code gère nativement GIT. Vous pouvez également installer quelques extensions pour aller plus loin selon votre besoin. Vous allez pouvoir voir facilement les différences dans le code et ainsi gérer vos problèmes de conflits, visualiser facilement les modifications apportées dans le code par vous ou par un de vos collègues. Et bien sûr initialiser facilement un dépôt, cloner un dépôt ou bien créer des branches.

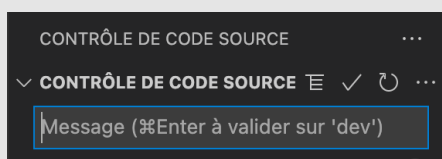
A. Initialiser un projet ou récupérer un projet


Avant de commencer à travailler avec GIT, vous devez initialiser un projet ou récupérer un projet distant.

Méthode

Initialiser un nouveau projet


1. Ouvrir le terminal de Visual Studio Code.
2. Créer un nouveau dossier.
3. Cliquer sur l'icône Git (barre de menu à gauche).
4. Cliquer sur « *initialiser un dépôt* ».
5. Si vous utiliser Github, Visual Studio Code vous permet de créer le repository directement depuis l'éditeur :
 - Indexer les fichiers
 - Ajouter un message de commit et cliquer sur le check pour valider



- Dans la barre en bas de l'éditeur, vous avez une icône :  qui permet de publier votre projet, vous pouvez cliquer dessus.
- Suivre les différentes étapes de connexion avec votre compte Github.

Félicitation, votre projet a été initialisé avec git et est présent sur votre dépôt Github !

Récupérer un projet distant

1. Ouvrir Visual Studio Code.
2. Cliquer sur l'icône git. 
3. Cliquer sur « *cloner un dépôt* ».
4. Vous pouvez entrer l'url de dépôt à cloner ou bien cliquer sur « *Cloner à partir de Github* » et choisir un de vos projets dans la liste affichée.
5. Sélectionner l'emplacement du dépôt en local.
6. Ouvrir le projet.

B. Gérer son code

Dans cette partie du cours, vous allez voir comment vous pouvez créer des branches et des tags sans avoir à utiliser le terminal. Vous allez aussi voir comment vous pouvez ajouter des modifications de fichier ou bien stasher du code assez facilement.

Méthode

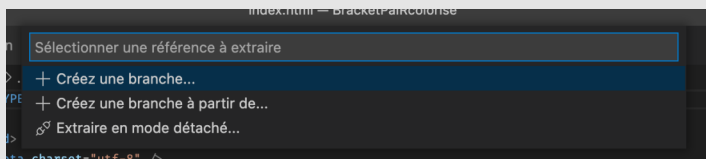
Créer des branches et tags

Pour créer une branche

1. Dans la barre en bas de l'éditeur, cliquez sur l'icône :

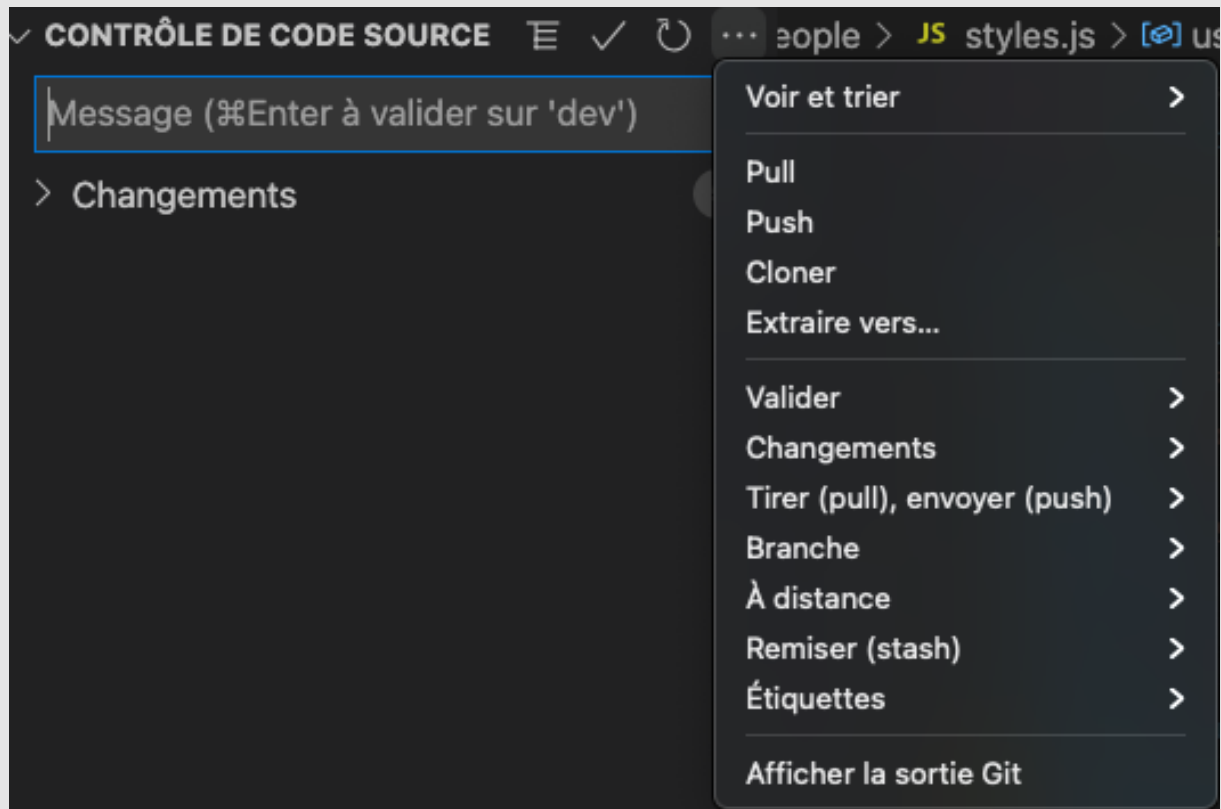


2. Créer une branche en cliquant sur « *Créer une branche* ».

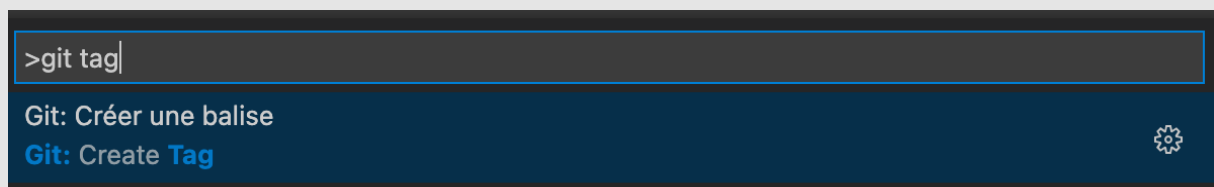


Pour créer un tag

1. Aller dans le menu de GIT

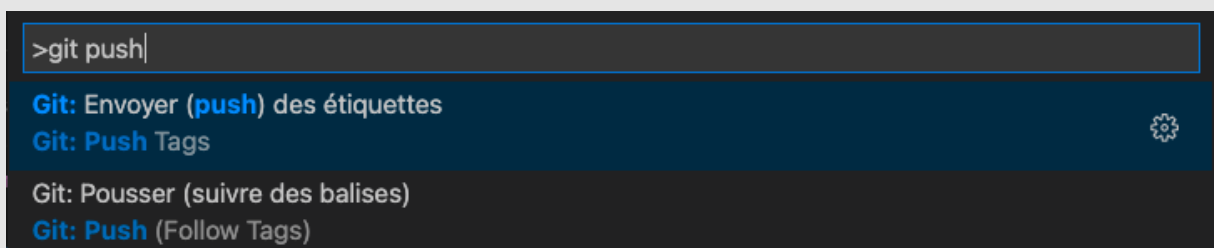


Ou utiliser la palette de commande (F1 ou ctrl + shift + p) et taper git tag (« Créer une balise / Create a tag »).



2. Ajouter votre tag et votre message.

Ps : pour pousser vos tags sur votre dépôt, vous pouvez utiliser la palette de commande (F1 ou ctrl + shift + p) et taper git push et choisir l'une des deux possibilités.



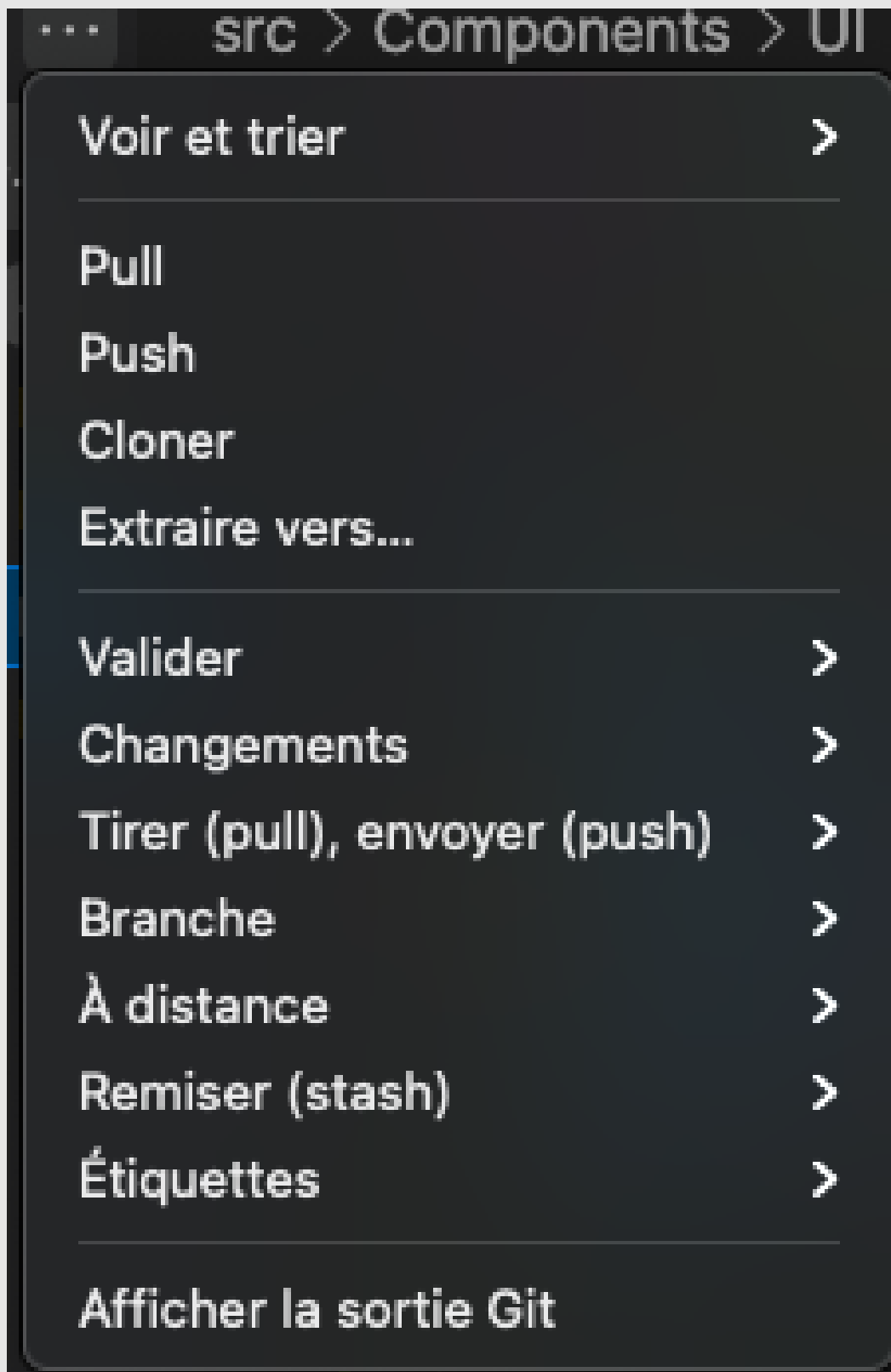
« Git: Push Tags », vas pousser vos tags et « Git: Push » va pousser vos modifications et vos tags.

Pousser / tirer / synchroniser une branche distance

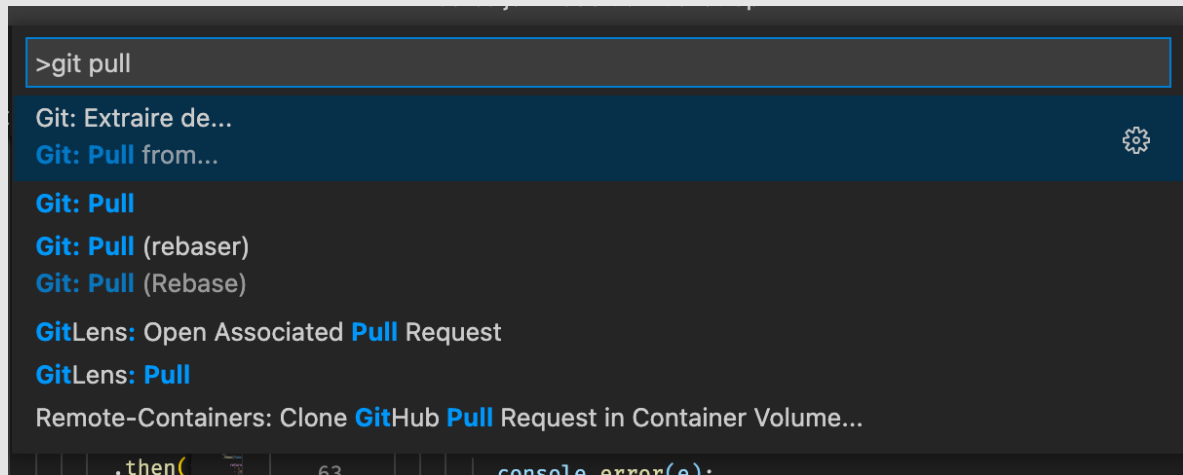
Dans cette partie du cours, vous allez voir comment vous pouvez créer des branches et des tags sans avoir à utiliser le terminal. Vous allez aussi voir comment vous pouvez pousser du code (push), tirer des modifications (pull) ou récupérer des modifications (fetch)

Astuce :

Vous pouvez faire quasiment toutes les actions (pousser, tirer et synchroniser) via le menu accessible avec les trois petits points.



Si vous préférez utiliser la palette de commande (F1 ou ctrl + shift + p) vous pouvez taper git + ce que vous souhaitez faire en français ou en anglais.



Astuce :

Vous pouvez facilement voir l'état de votre projet par rapport au dépôt distant dans la barre en bas de votre éditeur à droite du nom de votre branche (sur la capture branche de dev).

Sur l'image ci-après, le cercle avec des flèches vous propose ainsi de faire une synchronisation des changements et va ainsi pusher et tirer les commits.

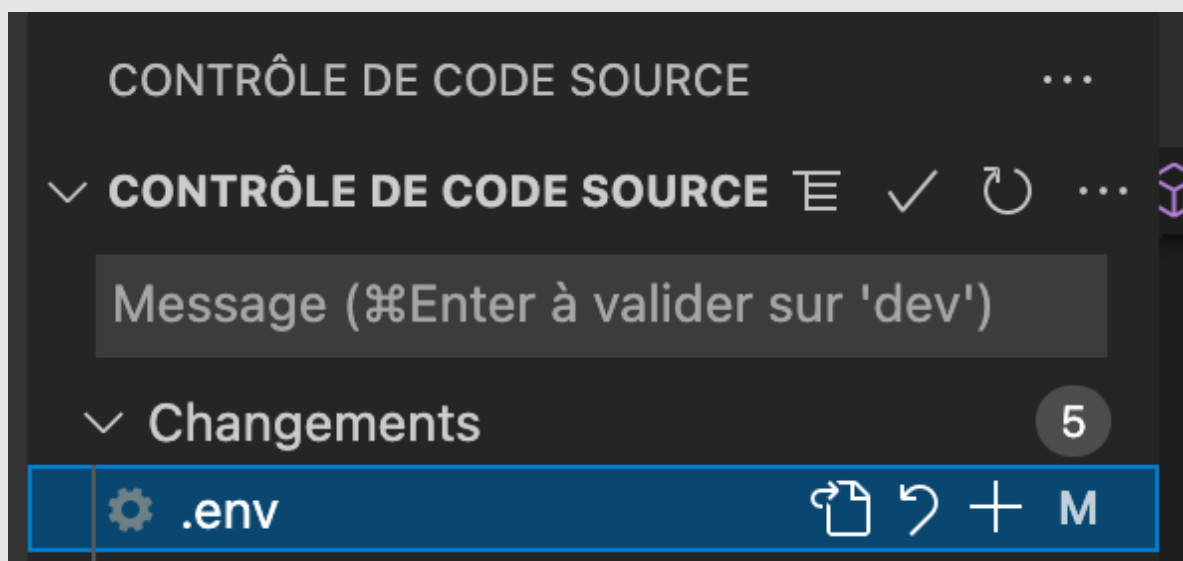
La flèche vers le bas, vous signale que vous êtes en retard d'un commit et que vous n'avez aucun commit à pousser.



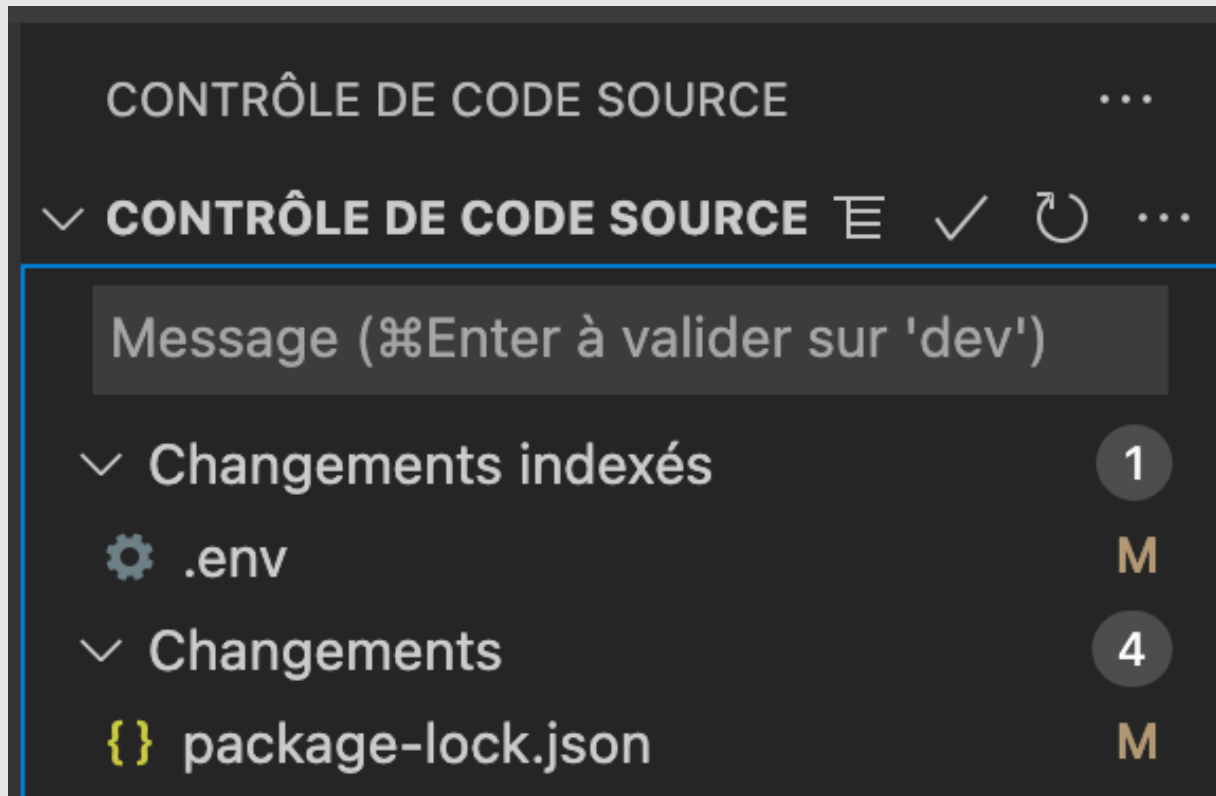
Commiter, stager du code

Pour commiter vos modifications :

1. Indexez votre ou vos fichiers en cliquant sur l'icône

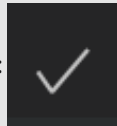


Les fichiers indexés sont facilement visibles dans la partie « *Changements indexés* ».



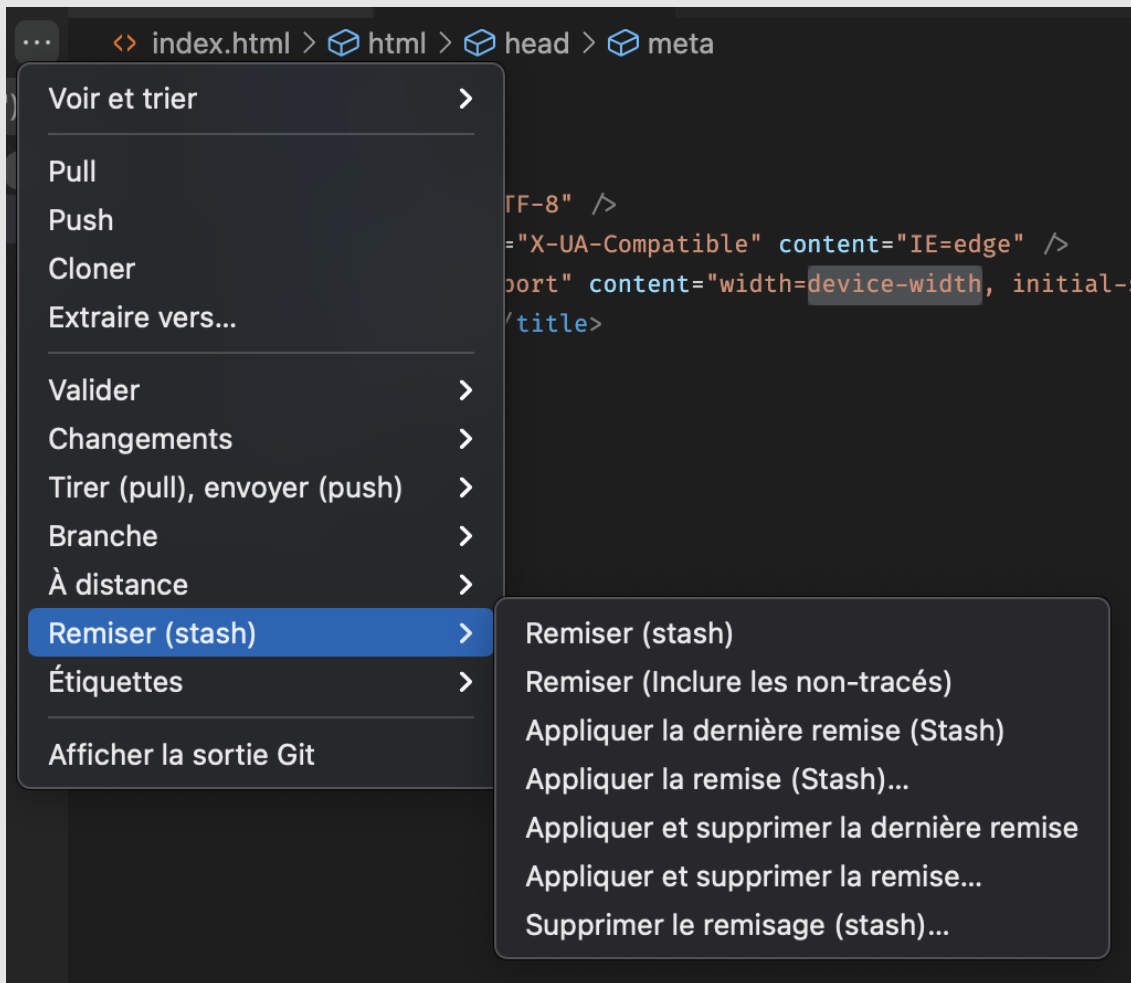
2. Ajoutez un Message dans l'input message.

3 - Validez votre commit en cliquant sur l'icône :



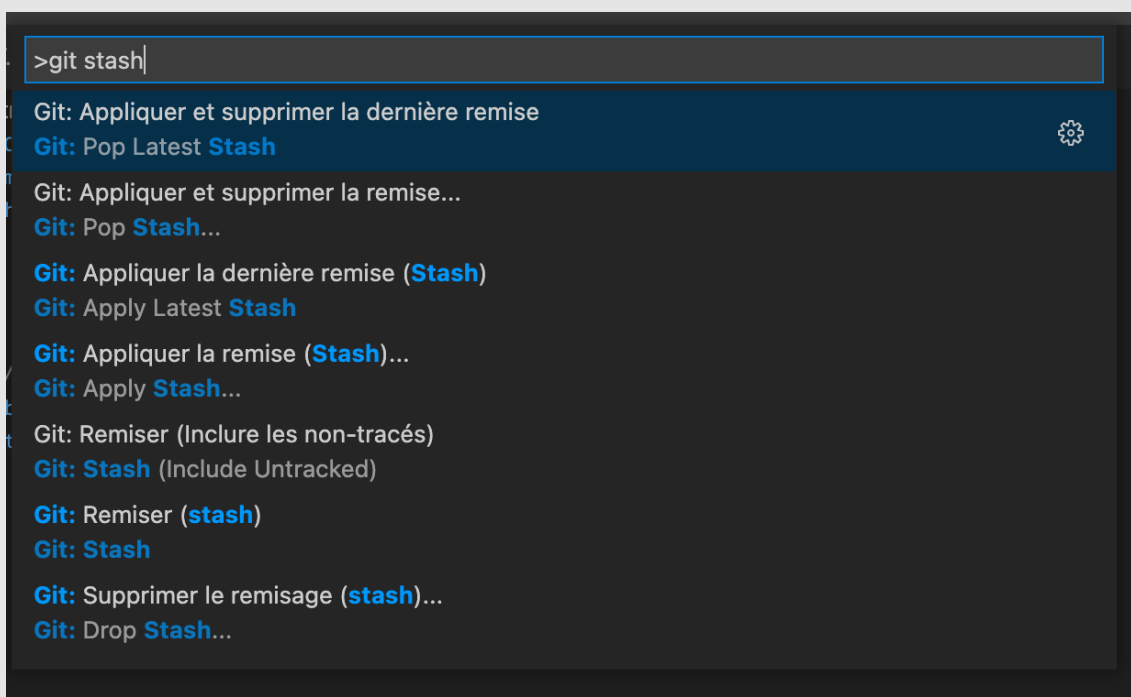
Pour stager des fichiers :

1. Ouvrez le menu.



2. Cliquez sur remiser (stash).

Vous pouvez également utiliser la palette de commande (F1 ou ctrl + shift + p).



Et cliquez sur « *Git Remiser* (stash).

Comme vous pouvez le voir dans les images précédentes, vous pouvez utiliser le même schéma pour récupérer vos fichiers stager (git stash pop) en cliquant sur « *Appliquer et supprimer la dernière remise* » par exemple si vous souhaitez récupérer le dernier stash et supprimer celui-ci de la liste des stash.

C. Visualiser les différences et résoudre les conflits de merge

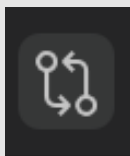
Tout au long du développement de votre projet, vous serez sûrement amenée à aller voir votre ancien code et ainsi voir les différences qui sont peut-être la cause d'un bug. Ou tout simplement voir les différences entre le nouveau et l'ancien code afin de valider une pull request. Et si vous travaillez en équipe, de gérer des conflits de merge.

Méthode

Pour visualiser les différences vous avez plusieurs possibilités soit :

- Possibilité 1 :

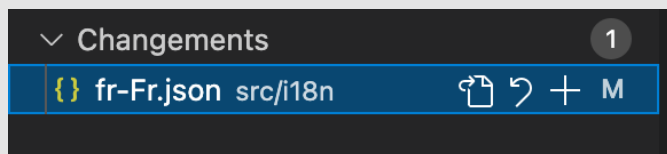
Cliquez sur l'icône :



en haut à droite de votre fichier.

- Possibilité 2 :

Cliquez sur un fichier dans la partie des fichiers modifiés.



- Possibilité 3 :

Allez dans la sidebar à gauche, dans la partie « *Chronologie* », cliquer sur un commit.



Quand vous avez des conflits, Visual Studio Code va vous aider à facilement voir le code qui change, avec la possibilité de valider les nouvelles modifications, garder le code actuel ou bien accepter les deux (garder le code actuel et ajouter les modifications).

D. Extensions git blame et gitlens

Visual Studio Code est personnalisable à souhait. Il existe beaucoup d'extension liées à git. L'extension à absolument avoir est **Git Lens**. C'est une extension assez complète qui possède les fonctionnalités de Git blame et Git history par exemple. Pour information l'extension Git history, va vous permettre d'accéder facilement à l'historique du projet.

Git blame peut être utile si vous ne souhaitez pas installer Git Lens. L'extension va vous permettre de savoir quand et par qui le bout de code a été modifié. Ce qui peut être pratique si vous souhaitez demander des compléments d'information sur le code à une personne.


Git Lens possède beaucoup de fonctionnalités, comme le blame. Sur chaque ligne vous avez la date de modification du code ainsi que la personne qui l'a commité. Vous pouvez également avoir accès à la différence entre deux portions de code.

II. Exercices

A. Exercice : Quiz


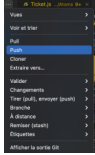
Question 1

Si vous souhaitez commiter du code, quelles sont les étapes à faire :

- ☐ Ajouter un message dans le code 2- Valider le message.
- ☐ Cliquer tout simplement sur l'icône :  pour commiter.
- ☐
 1. Indexer le / les fichiers à commiter.
 2. Ajouter un message.
 3. Valider le message.

Question 2

Que faut-il faire pour pousser ce commit sur le dépôt Github.

- ☐ Cliquer sur 
- ☐ Aller dans le menu (...) et cliquer sur push 
- ☐ Aller dans la palette de commande de Visual Studio Code et taper « *Git + push* ».
- ☐ Toutes les solutions sont bonnes.

Question 3

Par défaut sur Visual Studio Code, il est possible de voir les différences entre le code actuel en local (sur votre machine) et le code sur le dépôt github.

- ☐ Vrai
- ☐ Faux

III. Débuguer avec VSCode

Objectifs

Débuguer directement dans l'éditeur

Contexte

Dans la vie de développeur, vous serez amené à rencontrer des bugs et à déboguer votre code.

L'éditeur de code propose sans installer d'extension un outil pour vous aider à déboguer vos projets.

Il existe différentes méthodes pour déboguer du code. La méthode qu'on va voir ici est la méthode qui consiste à ajouter des points d'arrêts dans le code et de naviguer dans celui-ci.

A. Définir des points d'arrêt et commencer à debugger

Vous l'avez compris, les points d'arrêts sont les clés pour déboguer avec l'outil intégré dans l'éditeur.

Si vous êtes adepte des `console.log`, vous pouvez facilement les remplacer par un point d'arrêt afin de visualiser la valeur d'une variable. Mais pas que, vous pouvez aussi avoir accès à toutes les autres variables qui sont visibles dans le fichier sur lesquels vous êtes arrêtés.

B. Parcourir le code pas à pas avec les différentes commandes d'exécution

Vous pouvez ajouter autant de points d'arrêts que vous souhaitez, mais vous pouvez aussi ajouter un seul point d'arrêt et naviguer le long de l'exécution de votre programme. Vous pouvez facilement voir les étapes de l'exécution du projet du début à la fin. Vous avez aussi la possibilité de rentrer à l'intérieur d'une fonction afin de pouvoir connaître les différentes étapes et valeurs possibles des variables.

IV. Exercice :

Le code ci-après utilise des `console.log`. Le but de votre exercice est d'enlever les `console.log`, pour les remplacer par des points d'arrêts pour analyser le code. Pour cela vous devez créer un dossier « *exo_debug* » dans lequel vous allez créer un fichier « *test_debug.js* ». Ouvrez le dossier dans Visual Studio Code et commencez l'exercice.

Question

Ps : Suivez bien les explications de l'éditeur dans la configuration du debug et de la création du fichier `launch.json`. Vous devez sélectionner comme environnement de debug `node.js`.

```

1 utiliserPointsdarret();
2 function utiliserPointsdarret() {
3   let total = 0;
4
5   while (total < 10) {
6     total++;
7     console.log(total);
8   }
9
10  do {
11    total++;
12    console.log(total);
13  } while (total < 10);
14
15  console.log("total final", total);
16  return total;
17 }
18
```

Indice :

Si vous n'avez pas réussi à configurer launch.json (à mettre dans un dossier .vscode)

```
1 {
2   // Utilisez IntelliSense pour en savoir plus sur les attributs possibles.
3   // Pointez pour afficher la description des attributs existants.
4   // Pour plus d'informations, visitez : https://go.microsoft.com/fwlink/?linkid=830387
5   "version": "0.2.0",
6   "configurations": [
7     {
8       "type": "node",
9       "request": "launch",
10      "name": "Launch Program",
11      "skipFiles": ["<node_internals>/**"],
12      "program": "${file}"
13    }
14  ]
15 }
```

V. Auto-évaluation**A. Exercice :**

Récupérer un projet et débbugger le fichier debug.js

Question 1

Étape 1 : Cloner le projet github¹ sur votre machine.

Question 2

Étape 2 : Changer l'url d'origine du projet pour héberger le projet dans votre github.

Question 3

Étape 3 : Créer une branch dev

Question 4

Étape 4 : Résoudre le bug en utilisant les points d'arrêt.

Indice :

Dans le fichier debug.js, je ne comprends pas pourquoi la fonction retourne la valeur :

Montant total : 1 740, quantité total : 10 !!!!!!!!!!!

Vous devez réparer le code pour retourner la valeur :

Montant total : 1 740, quantité total : 10

Question 5

Étape 5 : Ajouter un commit de votre fix et pousser la modification sur votre dépôt Github.

B. Test**Exercice**

Question 1

Quelles extensions vu dans le cours vous permet de visualiser le nom et la date des dernières modifications faites sur un fichier ?

¹ https://github.com/Studi-Github/exo_git_debug

- ☐ git lens
- ☐ git blame
- ☐ git history

Question 2

Est-ce possible de commiter, pusher, tirer du code dans Visual Studio Code sans installer d'extension ?

- ☐ Non il faut installer Git lens.
- ☐ Non, on peut seulement commiter du code.
- ☐ Oui, c'est une fonctionnalité par défaut.

Question 3

Où pouvez-vous voir l'historique du projet ?

- ☐ Ce n'est pas possible, il faut aller sur Github.
- ☐ Dans la sidebar à gauche, dans l'onglet « *Chronologie* ».

Question 4

Que représente la lettre M à droite d'un fichier ?

- ☐ Modification
- ☐ Manipulation

Question 5

Que représente la lettre C à droite d'un fichier ?

- ☐ Changement
- ☐ Conflit
- ☐ Chargement

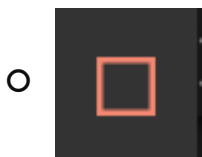
Question 6

Pour déboguer, que faut-il ajouter dans le fichier pour que l'éditeur s'arrête sur la ligne 8 du code ?

- ☐ Ajouter des points d'arrêts.
- ☐ Ajouter un console.log .
- ☐ Ajouter un console.stop .

Question 7

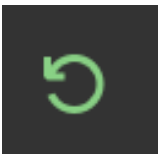
Quelle icône permet d'entrer à l'intérieur d'une fonction ?



☐☐

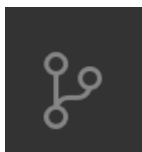
Question 8

Quelle icône permet de relancer le programme ?

☐☐☐

Question 9

Comment lancer le débogueur sur Visual Studio Code ?

☐ Cliquer sur l'icône☐ Cliquer sur l'icône☐ Lancer dans le terminal la commande `vscode start debug`.

Question 10

Quel fichier devez-vous absolument configurer pour lancer votre programme en mode débogage ?


☐ `launch.json`☐ `start.json`☐ `debug.json`**Solutions des exercices**

Exercice p. 11 Solution n°1

Question 1

Si vous souhaitez commiter du code, quelles sont les étapes à faire :

- ☐ Ajouter un message dans le code 2- Valider le message.

- ☐ Cliquer tout simplement sur l'icône :  pour commiter.

- ☒ 1. Indexer le / les fichiers à commiter.
2. Ajouter un message.
3. Valider le message.

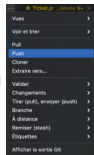
Question 2

Que faut-il faire pour pousser ce commit sur le dépôt Github.

- ☐ Cliquer sur



- ☐ Aller dans le menu (...) et cliquer sur push



- ☐ Aller dans la palette de commande de Visual Studio Code et taper « *Git + push* ».
- ☒ Toutes les solutions sont bonnes.

Question 3

Par défaut sur Visual Studio Code, il est possible de voir les différences entre le code actuel en local (sur votre machine) et le code sur le dépôt github.

- ☒ Vrai
- ☐ Faux

Exercice p. Solution n°2

Afin de visualiser le résultat, vous devez mettre un point d'arrêt à côté de chaque « *total++* » et un à côté du *return total*.

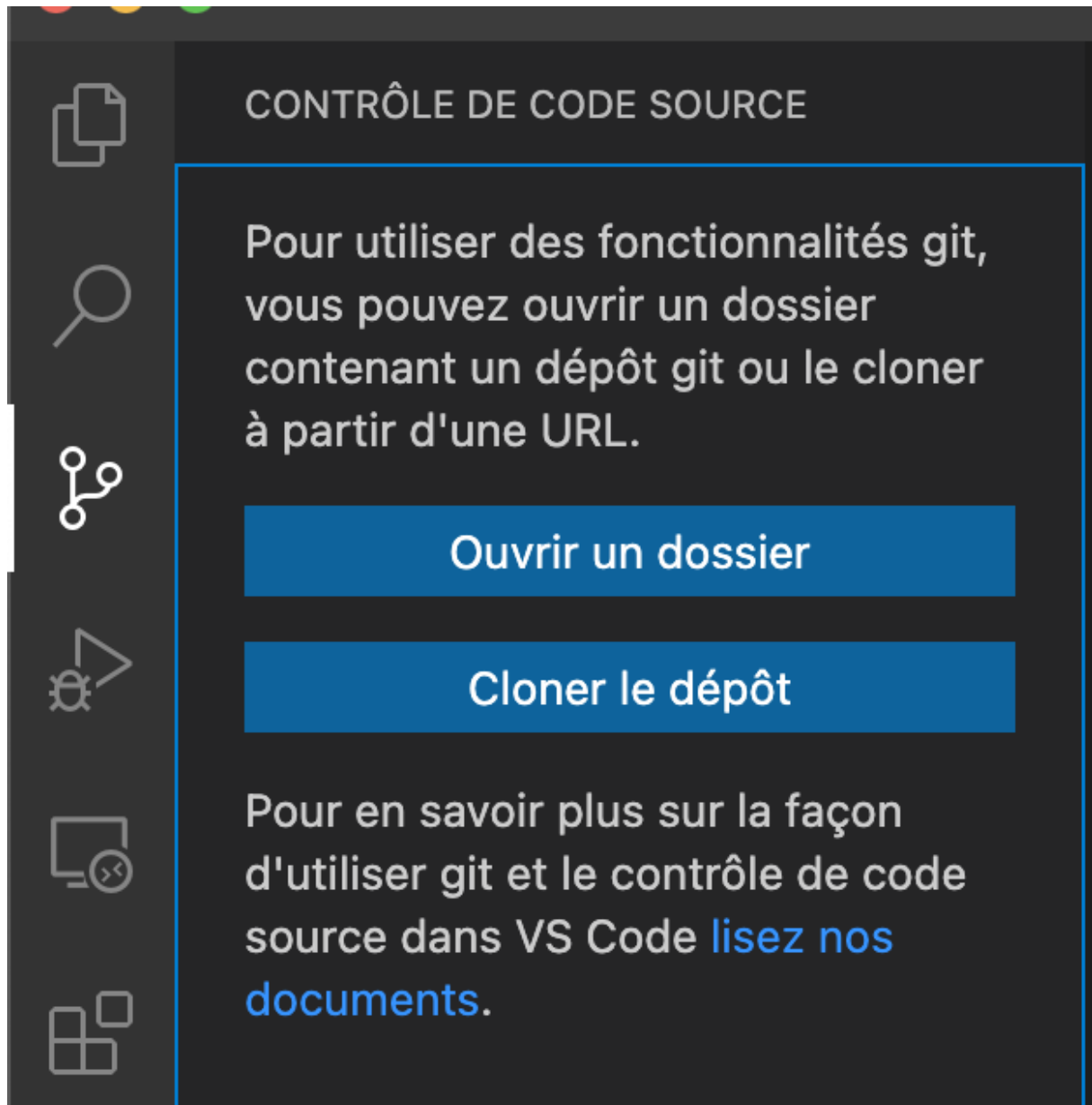
```

JS test_debug.js > ...
1  utiliserPointsdarret();
2  function utiliserPointsdarret() {
3      let total = 0;
4
5      while (total < 10) {
6          | total++;
7      }
8
9      do {
10         | total++;
11     } while (total < 10);
12
13     return total;
14 }
15

```

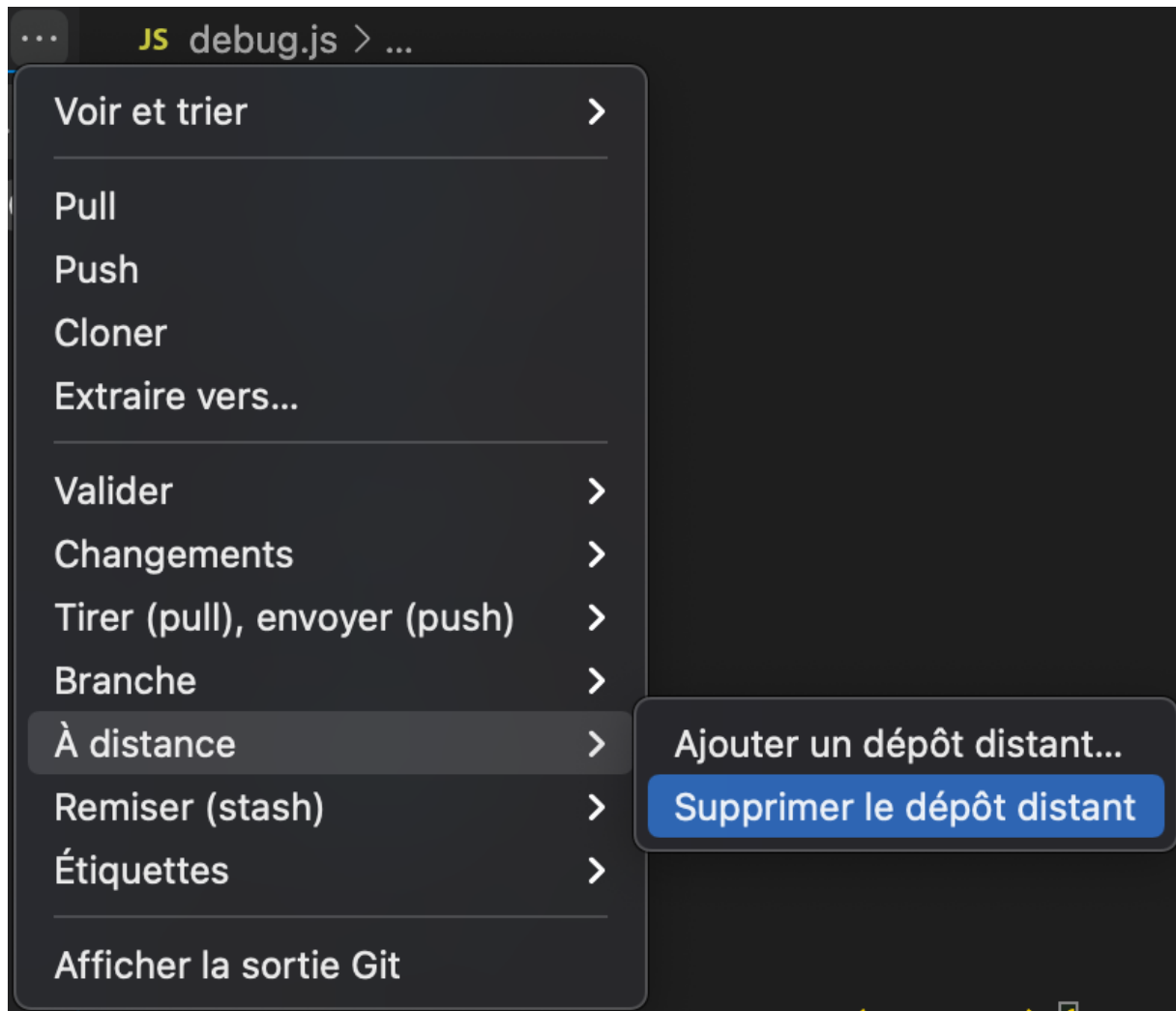
Exercice p. Solution n°3

Ouvrir une nouvelle fenêtre sur Visual Studio Code, et cloner le dépôt en copiant l'url ci-dessous.



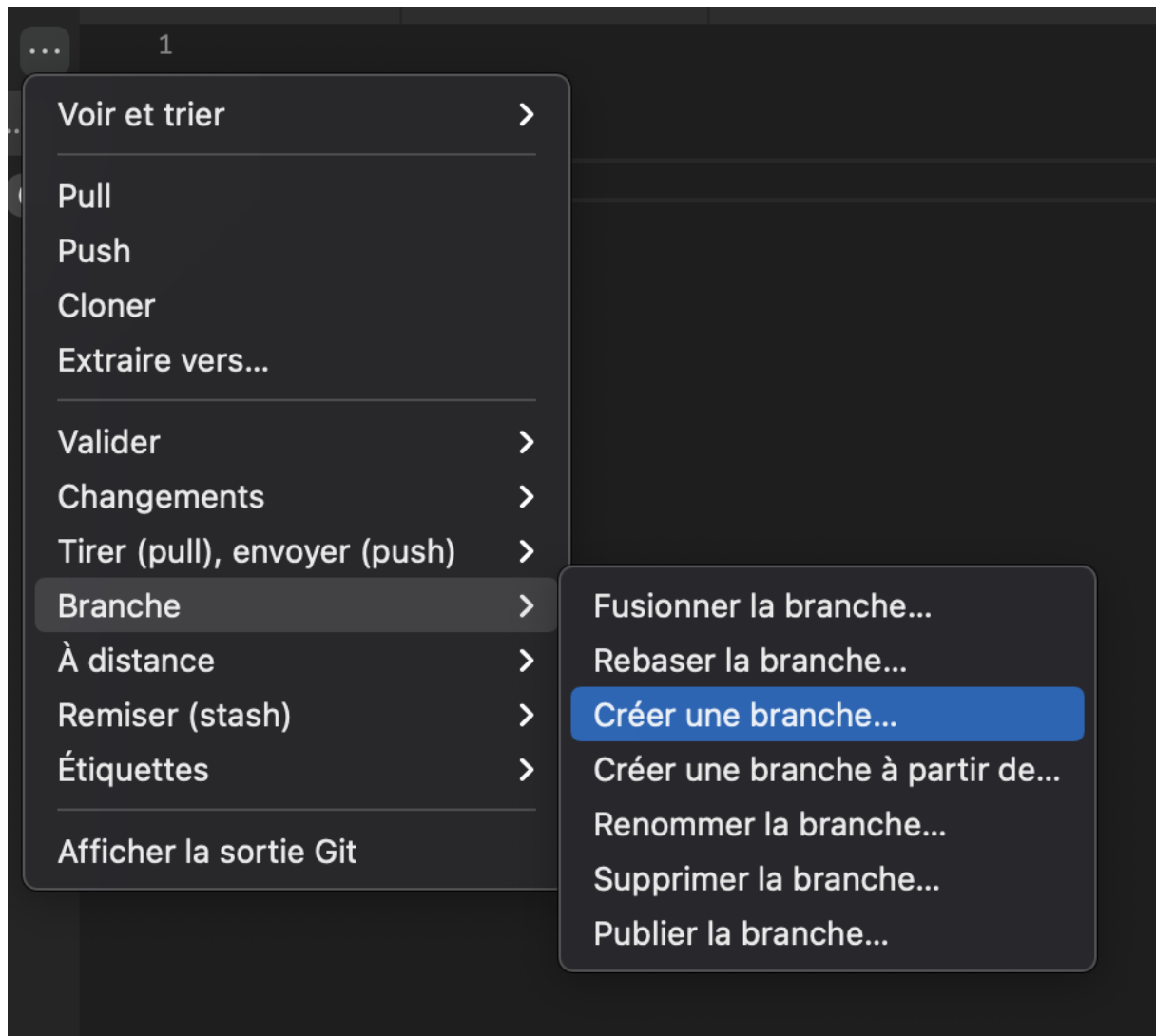
Exercice p. Solution n°4

Vous devez supprimer le dépôt existant et l'origine « *origin* », pour ensuite en créer un nouveau sur votre compte Github



Exercice p. Solution n°5

Créer une branche et nommer la dev



Exercice p. Solution n°6

Mettre des points d'arrêt où vous le souhaitez, mais il en faut une ligne 26 afin de savoir la valeur retournée par la fonction. Pour réparer le bug, il suffit de supprimer les points d'exclamation.

Exercice p. Solution n°7

1. Indexer le fichier debug.js
2. Ajouter un message de commit et valider
3. Cliquer sur push

Exercice p. 13 Solution n°8

Question 1

Quelles extensions vu dans le cours vous permet de visualiser le nom et la date des dernières modifications faites sur un fichier ?

- ☒ git lens
- ☒ git blame
- ☐ git history

Question 2

Est-ce possible de commiter, pusher, tirer du code dans Visual Studio Code sans installer d'extension ?

- ☐ Non il faut installer Git lens.
- ☐ Non, on peut seulement commiter du code.
- ☒ Oui, c'est une fonctionnalité par défaut.

Question 3

Où pouvez-vous voir l'historique du projet ?

- ☐ Ce n'est pas possible, il faut aller sur Github.
- ☒ Dans la sidebar à gauche, dans l'onglet « *Chronologie* ».

Question 4

Que représente la lettre M à droite d'un fichier ?

- ☒ Modification
- ☐ Manipulation

Question 5

Que représente la lettre C à droite d'un fichier ?

- ☐ Changement
- ☒ Conflit
- ☐ Chargement

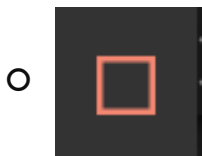
Question 6

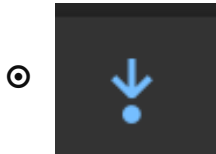
Pour déboguer, que faut-il ajouter dans le fichier pour que l'éditeur s'arrête sur la ligne 8 du code ?

- ☒ Ajouter des points d'arrêts.
- ☐ Ajouter un console.log .
- ☐ Ajouter un console.stop .

Question 7

Quelle icône permet d'entrer à l'intérieur d'une fonction ?





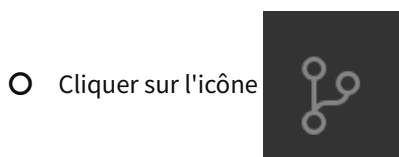
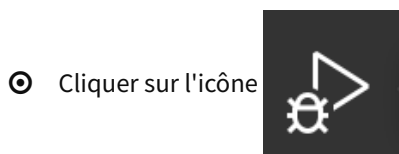
Question 8

Quelle icône permet de relancer le programme ?



Question 9

Comment lancer le débogueur sur Visual Studio Code ?



☐ Lancer dans le terminal la commande `vscode start debug`.

Question 10

Quel fichier devez-vous absolument configurer pour lancer votre programme en mode débogage ?

- ☒ `launch.json`
- ☐ `start.json`
- ☐ `debug.json`