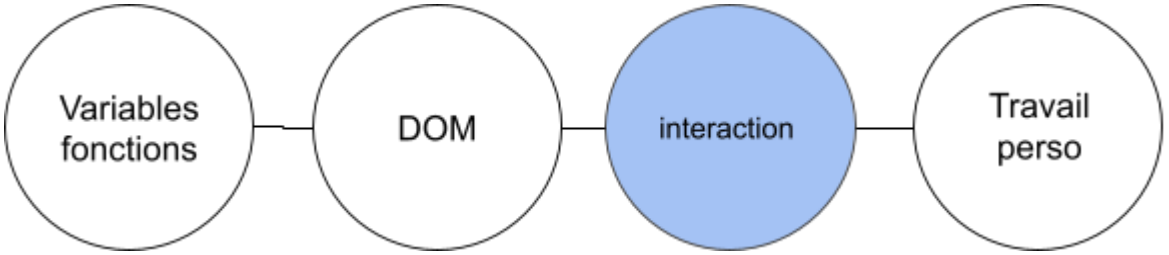
Calculette-2 : interaction 1



Définition des variables et fonctions.

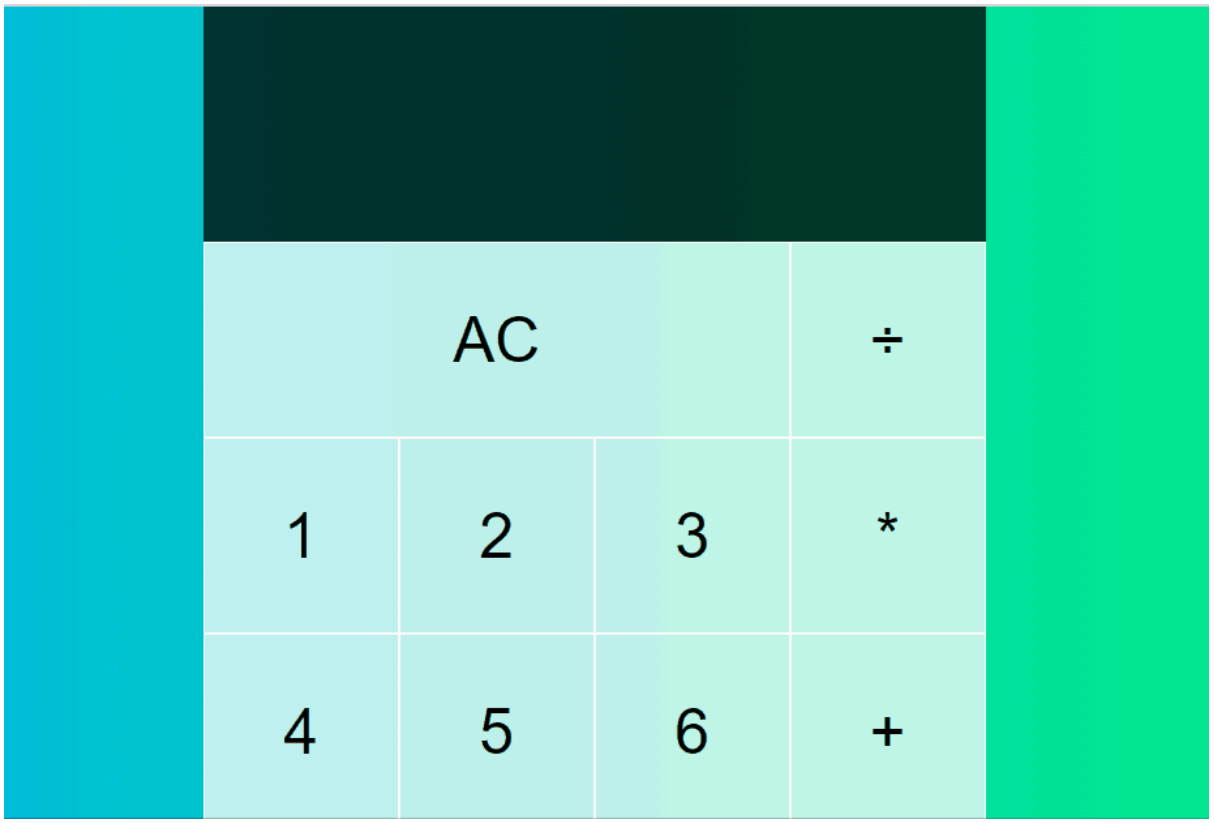
DOM

🚀Interaction

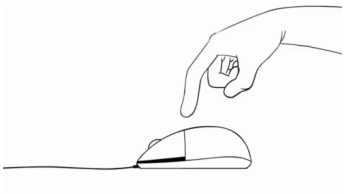
Améliorations

Ma calculette

Nous allons mettre en place le code JS qui permet l'interaction avec les différents boutons[1].

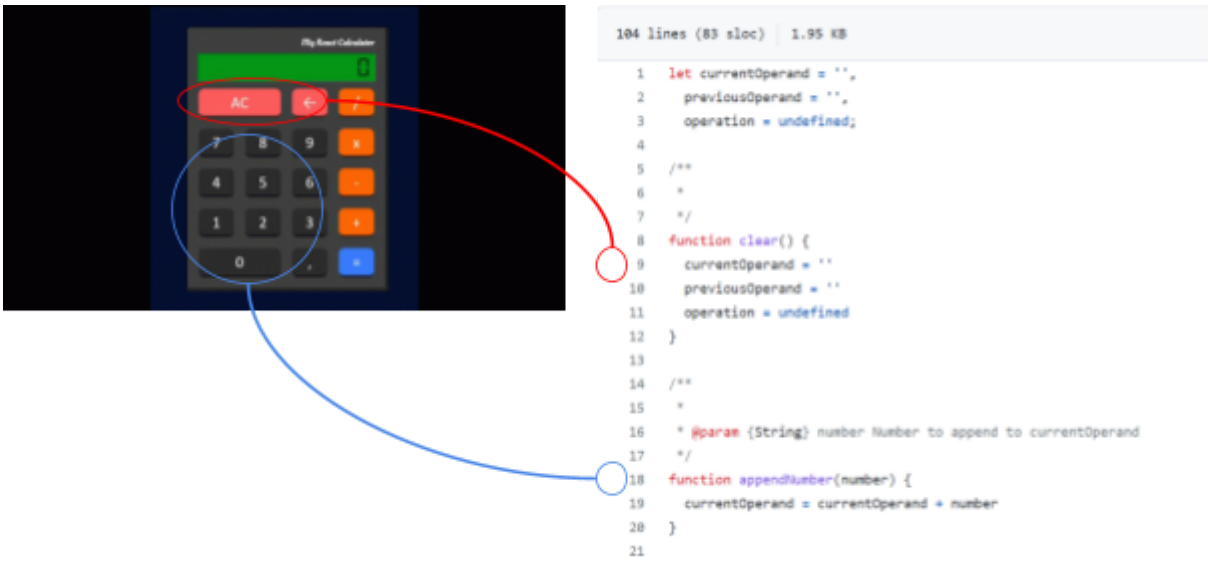


[1] Nous avons retiré le calcul des nombres décimaux !



Liens entre le DOM et les variables de la calculette

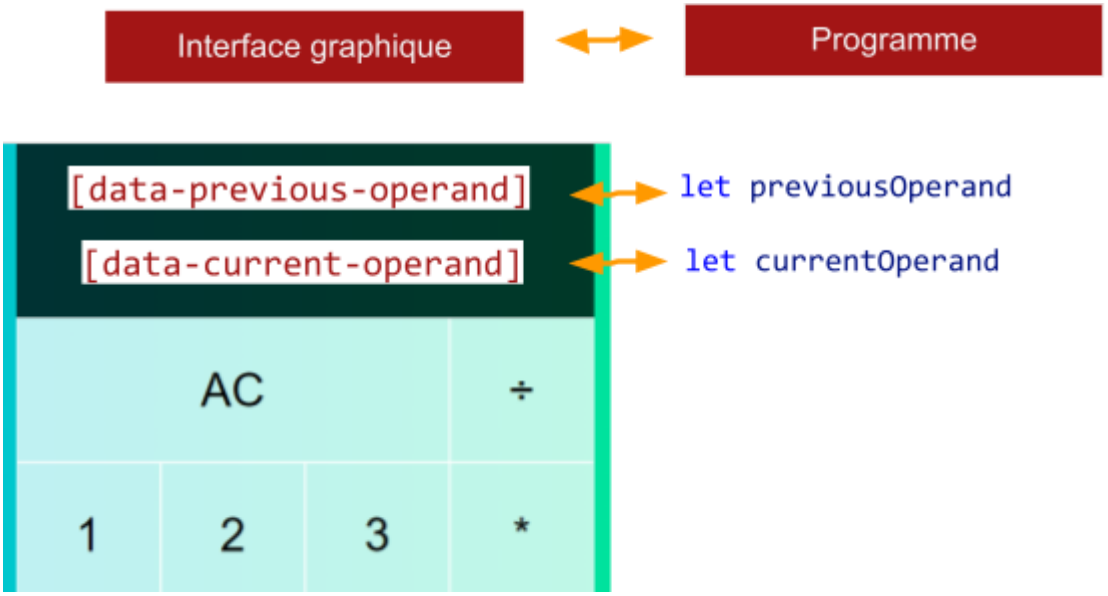
Il est temps de connecter les éléments de notre interface aux "objets" de notre programme.



updateDisplay

Une fonction `updateDisplay` crée un lien entre les variables de notre application et la vue. Cette fonction répercute les changements des variables de notre programme dans l'interface graphique de notre application.

Calculette-2 : interaction 3



Voici le code qui lie les objets entre eux :

```
1. function updateDisplay() {
2.   currentOperandTextElement.innerText = currentOperand;
3.   previousOperandTextElement.innerText = previousOperand;
4. }
```

Lig [2-3] : innerText permet de modifier le contenu textuel des éléments.

Cette propriété est disponible en getter et setter. Nous rappelons que les propriétés sont soit en lecture seule soit en lecture/écriture.

Getter

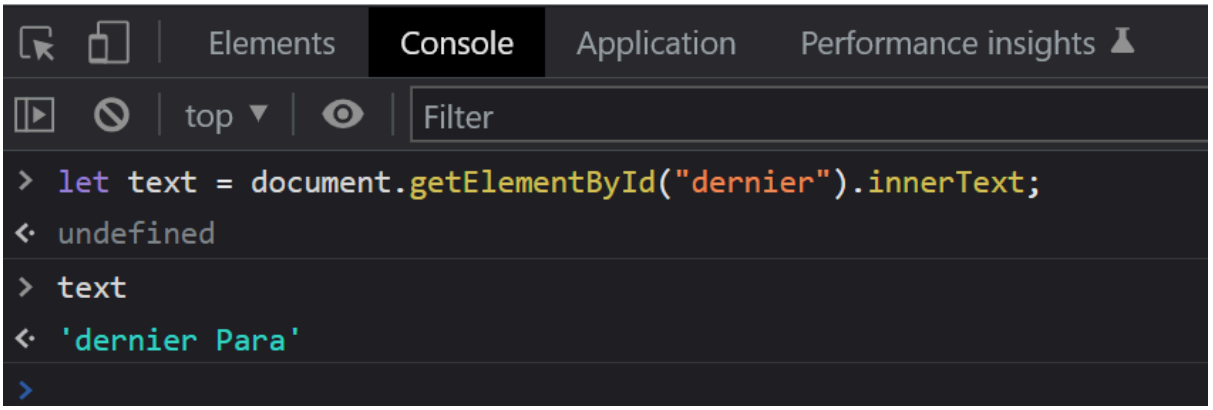
Exemple de récupération du texte du dernier paragraphe.

Dans la console du fichier test lancez le code :

let text = document.getElementById("dernier").innerText;

Calculette-2 : interaction 4

Vérifiez la valeur de la variable text en tapant simplement dans la console text ↩



Setter

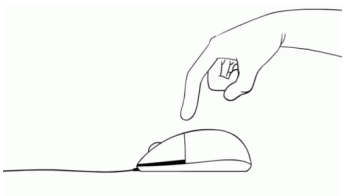
Exemple de modification du texte du dernier paragraphe.

Dans la console du fichier test lancez le code :

document.getElementById("dernier")[2].innerText = new Date().toLocaleDateString()

La date 📆d'aujourd'hui apparaît.

Événements



Il est tant d'ajouter les événements sur les différents boutons permettant de lancer des actions.

1. **numberButtons**.forEach(button => {
2. button.addEventListener('click', () => {
3. appendNumber(button.innerText)

[2] .getElementById("dernier") peut être remplacé par .querySelector("#dernier")

```
4.      updateDisplay()
5.    })
6.  })
```

Lig. 1 : On boucle sur tous les nombres.

Lig. 2 : On ajoute pour chaque nombre un écouteur sur le click.

Lig. 3-4 : On définit les actions sur le click. En particulier, lig. 3 on récupère la valeur du nombre que l'on passe en paramètre.

🪛Je vous laisse examiner le reste du code.

```
7.  operationButtons.forEach(button => {
8.    button.addEventListener('click', () => {
9.      chooseOperation(button.innerText)
10.    updateDisplay()
11.  })
12.})
13.
14.equalsButton.addEventListener('click', button => {
15.  compute()
16.  updateDisplay()
17.})
18.
19.allClearButton.addEventListener('click', button => {
20.  clear()
21.  updateDisplay()
22.})
```

Calculette-2 : interaction 6

Code en action

 https://github.com/dupontdenis/start-calculette/tree/tryJS

Recherche de bug

L'idée de cette partie de TD est de corriger l'application suivante.

Corrections

Revenons à notre application, Il semblerait que le calcul n'ait pas d'effet.

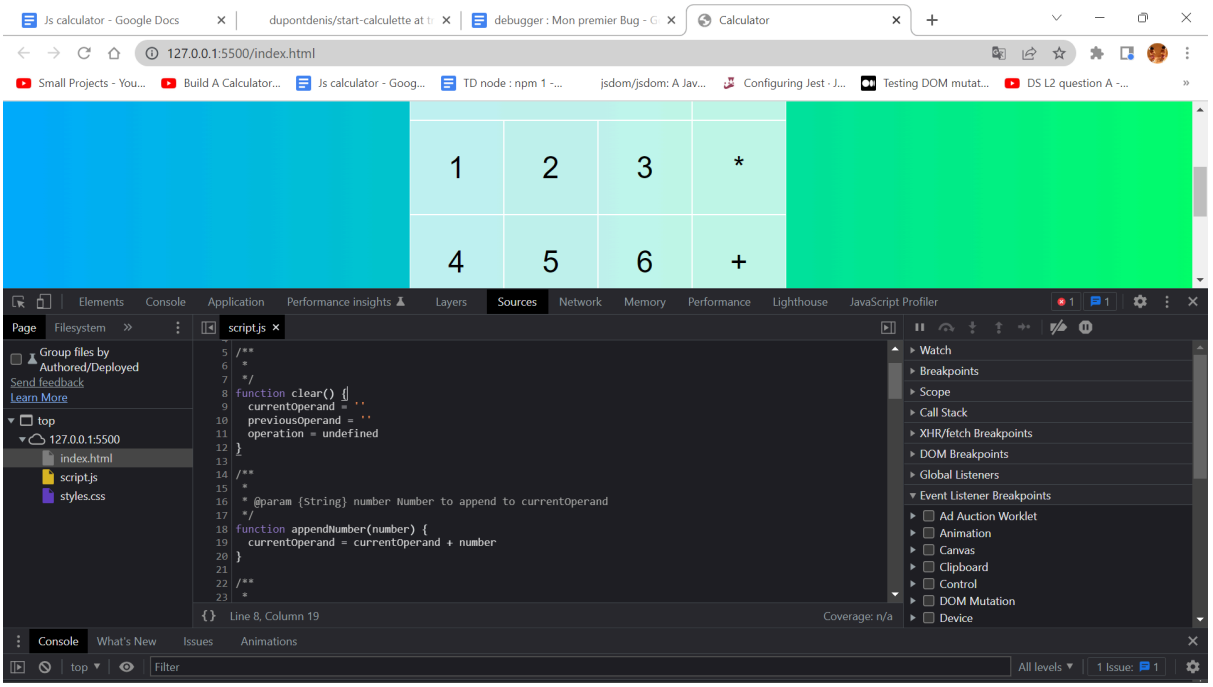
Il nous faut Inspecter le code et en particulier les événements.

Voici ce qu'il faut faire étape par étape lors de l'inspection.

Commençons par ouvrir l'inspecteur F12 puis cliquez sur l'onglet Sources.

Recherchez le fichier script.js.

Calculette-2 : interaction 8

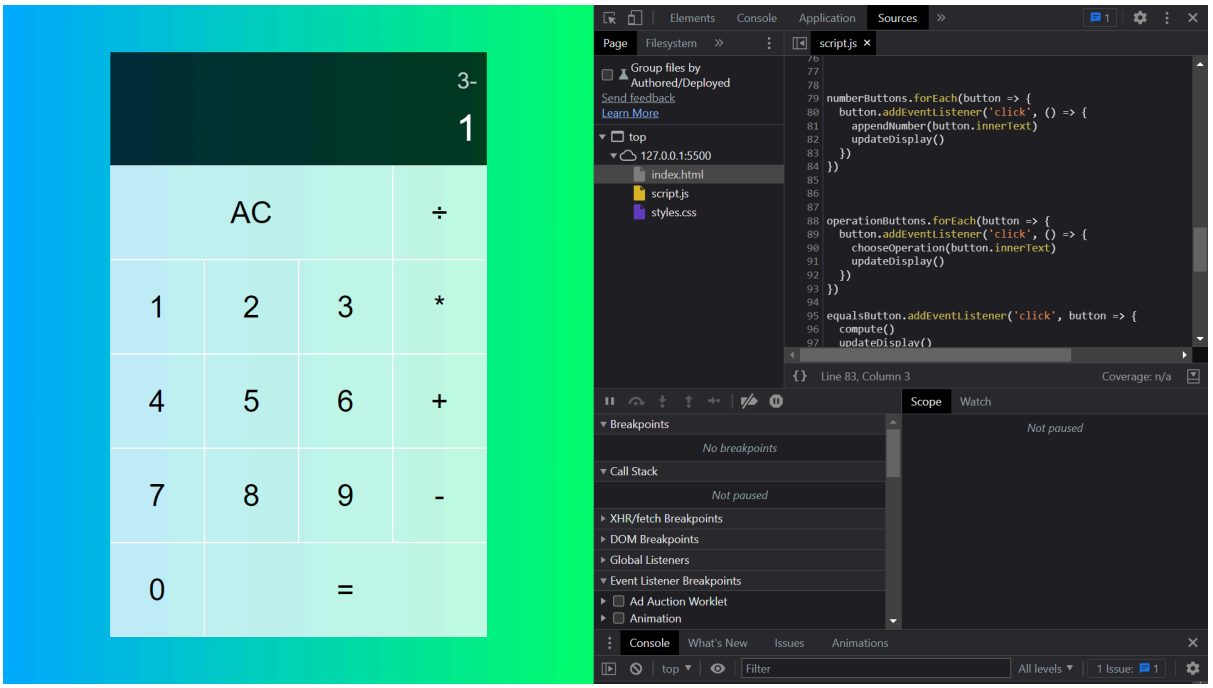


Rechercher ensuite l'onglet ▸ Event Listener Breakpoints et ouvrir 🔻 Mouse

Mettre ensuite un point d'arrêt sur le click.

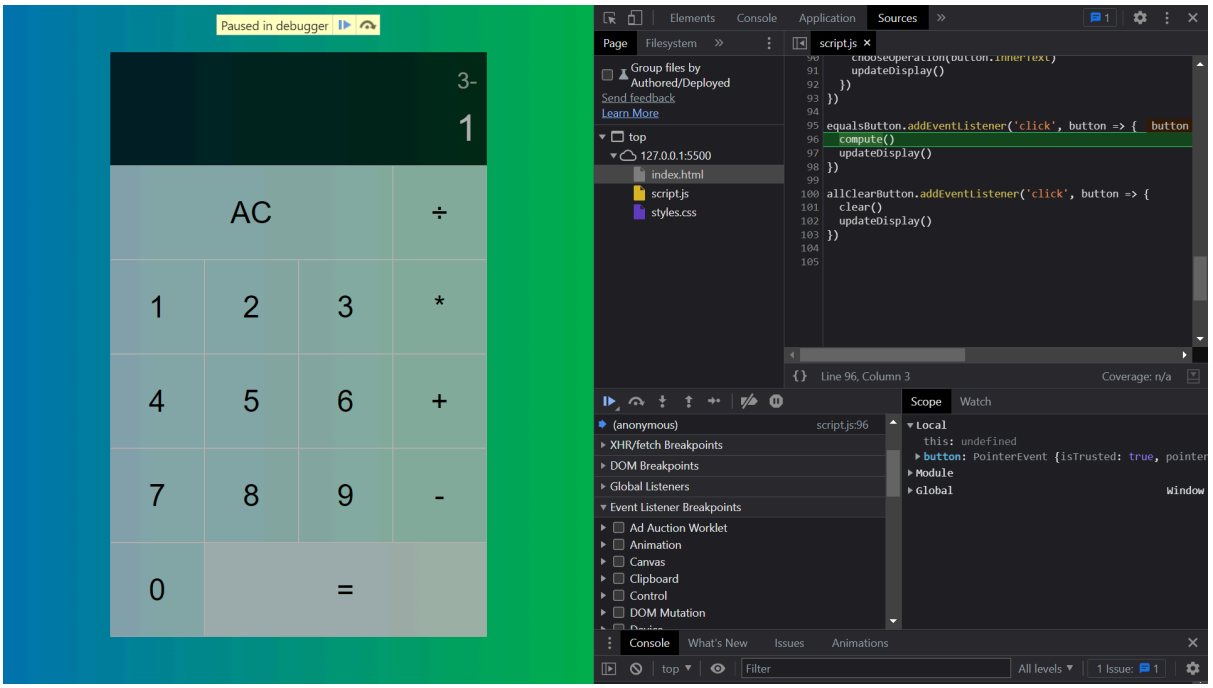
🔪Lancez le calcul 3-1 en cliquant sur la vue.

Voici la situation avant de cliquer sur le signe d'égalité qui lance le calcul.



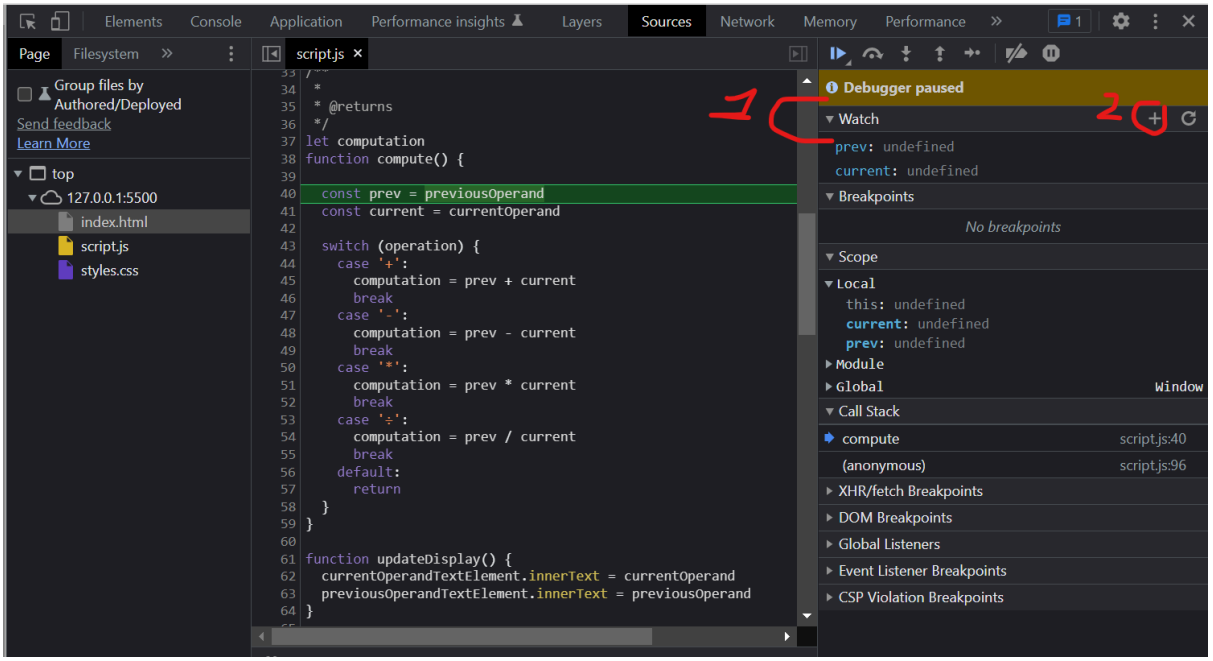
Calculette-2 : interaction 9

Cliquez sur le signe = lance à nouveau le débugger



🥷Soyez attentif à chaque valeur de la fonction "`compute`"

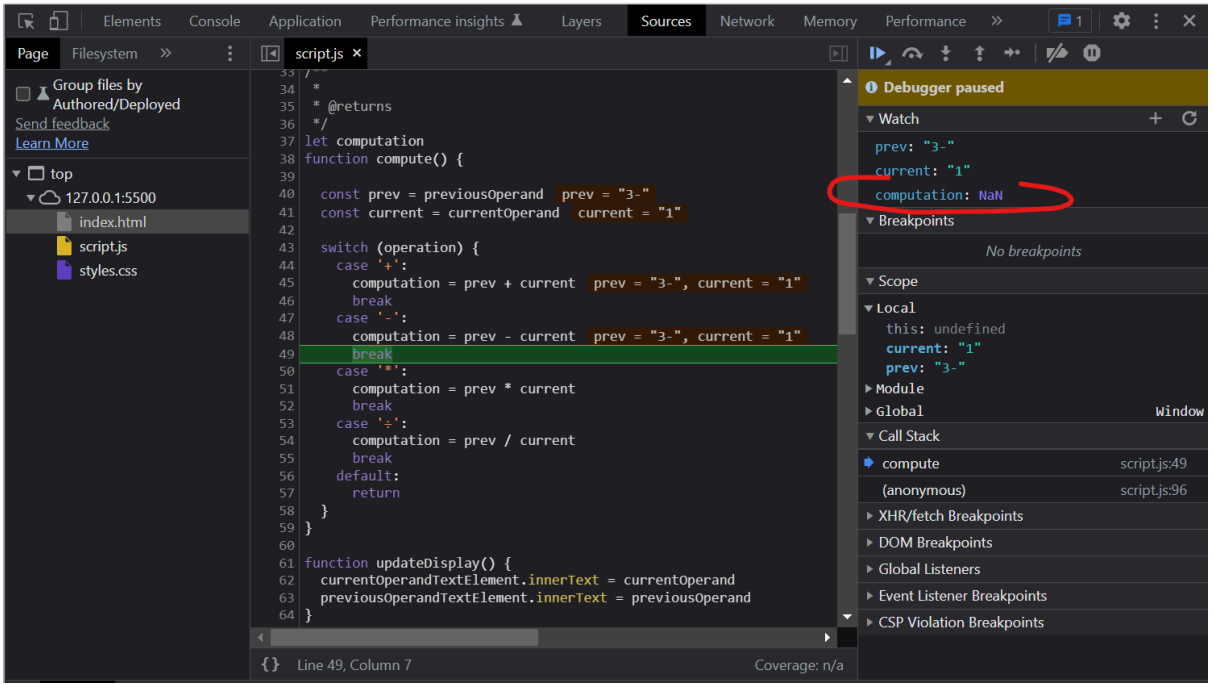
🔪Ajoutez dans watch les valeur des variables **prev**, **curent** et **computation** avec le signe +



Calculette-2 : interaction 10

😈Nous remarquons que la variable `computation` n'est pas égale à 2 !

La variable `computation` a la valeur Not a Number (NAN).




```
/**
 *
 * @returns
 */
let computation
function compute() {

  const prev = previousOperand  prev = "3-"
  const current = currentOperand  current = "1"

  switch (operation) {
    case '+':
      computation = prev + current  prev = "3-", current = "1"
      break
    case '-':
      computation = prev - current  prev = "3-", current = "1"
      break
    case '*':
      computation = prev * current
      break
    case '÷':
      computation = prev / current
      break
    default:
      return
  }
}

function updateDisplay() {
  currentOperandTextElement.innerText = currentOperand
  previousOperandTextElement.innerText = previousOperand
}
```

Analyse :

Avons nous remarqué le type des variables `prev` et `current`.

💥Rien avoir avec les nombres, le type de ces deux variables est **String**.

🆘Correction

Modifions la fonction compute !

Un formatage des opérandes est obligatoire. Il nous faut convertir les variables en Flottant à l'aide de l'opérateur parseFloat.

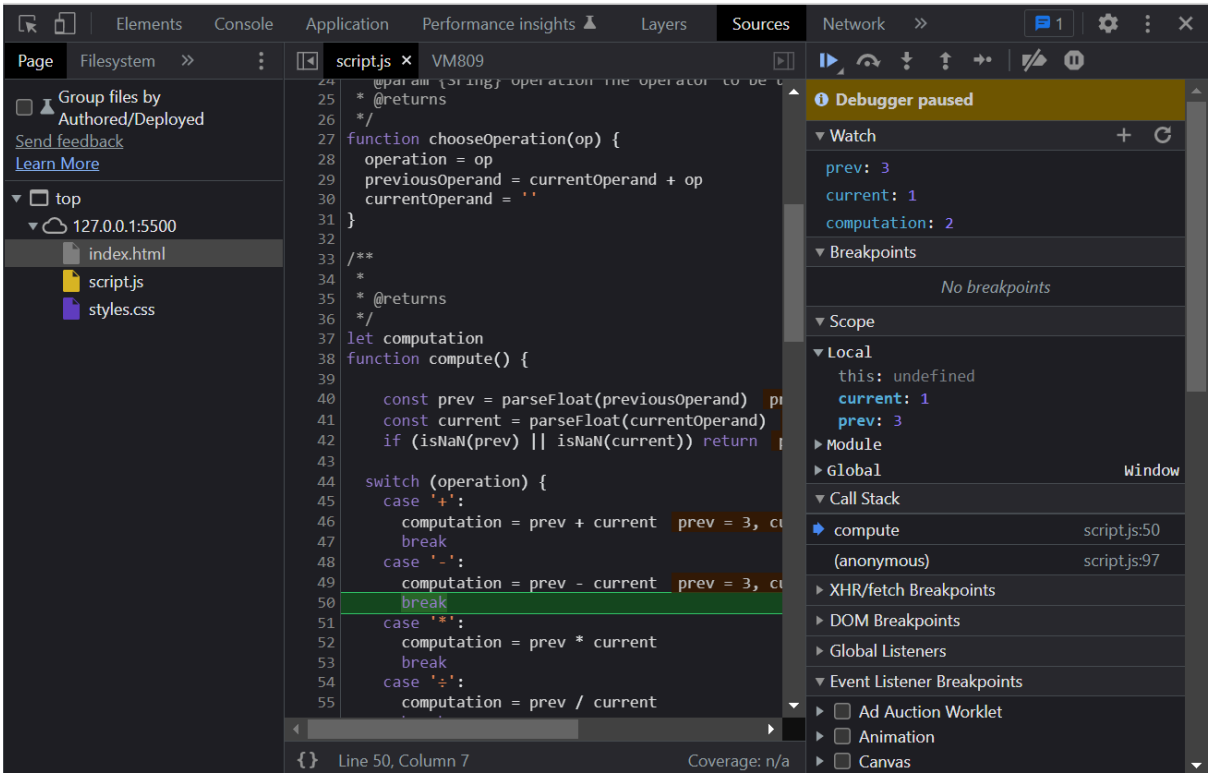
1. const prev = **parseFloat**(previousOperand)
2. const current = **parseFloat**(currentOperand)

Calculette-2 : interaction 11

3. if (isNaN(prev) | | isNaN(current)) return

Lig. 3 : On teste si les variables sont des nombres.

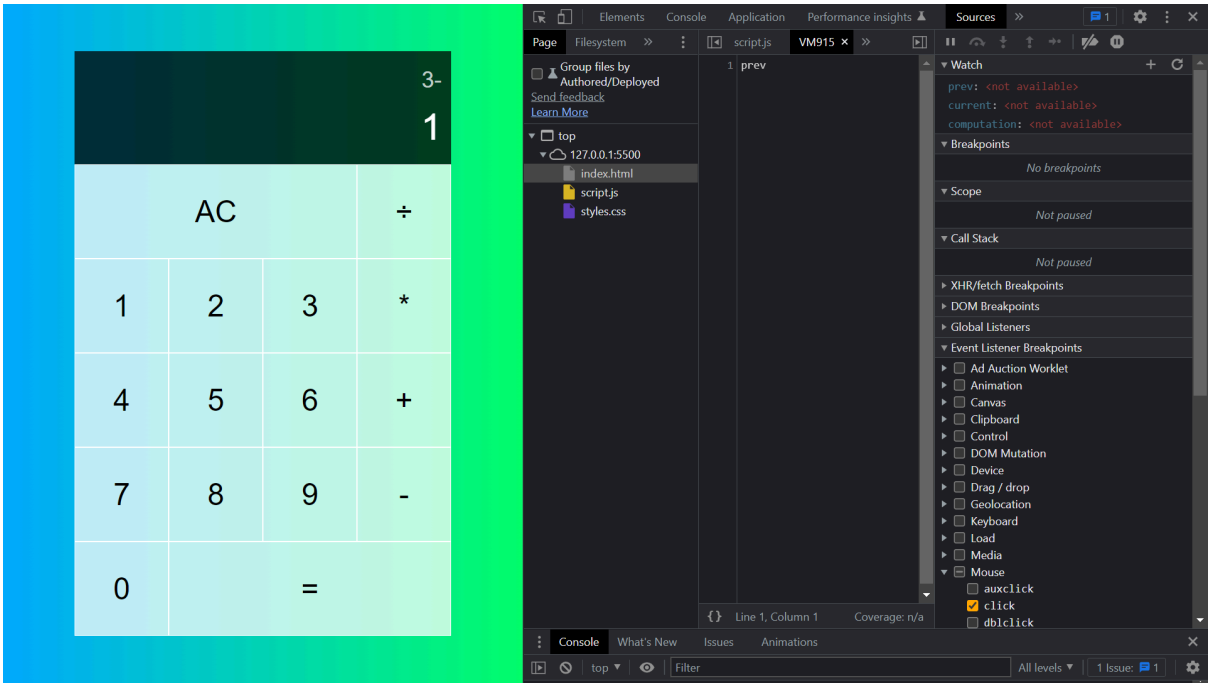
🪛Relancez le debugger avec toujours 3 - 1 et regardez à nouveau les valeurs des variables de la fonction compute.



Parfait, nous avons bien la variable computation qui vaut 3-1 = 2.

Mais il est clair que le résultat ne s'affiche pas !

Calculette-2 : interaction 12



SOS Correction

Il nous faut afficher le résultat du calcul.

😈Nous avons déclaré la variable computation à l'extérieur de la fonction compute.

Modifions encore une fois la fonction `compute.`

```
1.  function compute() {
2.    let computation;
3.    const prev = parseFloat(previousOperand)
4.    const current = parseFloat(currentOperand)
5.    if (isNaN(prev) || isNaN(current)) return
6.    switch (operation) {
7.      case '+':
8.        computation = prev + current
9.        break
10.     case '-':
```

Calculette-2 : interaction 13

```
11.      computation = prev - current
12.      break
13.    case '*':
14.      computation = prev * current
15.      break
16.    case '÷':
17.      computation = prev / current
18.      break
19.    default:
20.      return
21.  }
22.  currentOperand = computation
23.  operation = undefined
24.  previousOperand = ''
25.}
```

Lig. 22-24 : on met à jour les variables.

Résumé des valeurs des variables pour le calcul de 3-1 :

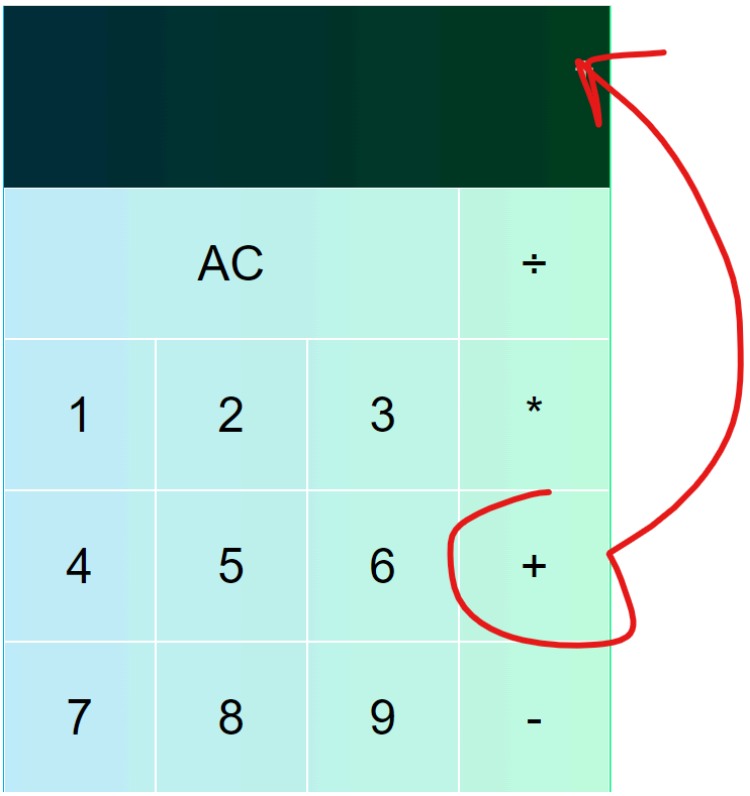
	previousOperand	currentOperand	operation
avant	3-	1	-
après	''	2	''

Nous allons corriger notre dernier bug !

Calculette-2 : interaction 14

🆘Correction

Nous avons observé que lorsque l'on clique sur un opérateur celui-ci s'affiche alors qu'il n'y a pas d'opérande.



🪛Trouvez les conditions de test dans la fonction `chooseOperator` pour éviter ce comportement.

🆘help-correction3

AméliorationS

Modifier la fonction `appendNumber` ainsi

```
/**
 * @param {String} number Number to append to currentOperand
 */
function appendNumber(number) {
```

Calculette-2 : interaction 15

```
6.    currentOperand = currentOperand.toString() + number.toString()
7.  }
```

---

help-correction3

```
1.  /**
2.   *
3.   * @param {Sring} operation The operator to be used by the cacul
4.   * @returns
5.   */
6.  function chooseOperation(op) {
7.
8.    if (currentOperand === '' || previousOperand !== '') return
9.
10.  operation = op
11.  previousOperand = currentOperand + op
12.  currentOperand = ''
13.}
```

Lig.8 : currentOperand === '' teste si nous n'avons pas la seconde opérande et previousOperand !== "" teste que nous avons déjà un opérateur

suite