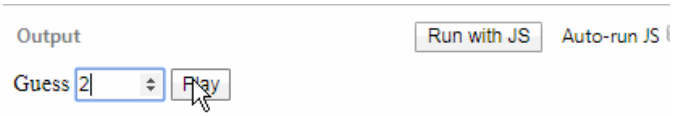
Interface web

😈Oublions l'absence de prompt pour la saisie dans pythonTutor[1] .

Mise au point d'un prototype

Je vous propose une solution minimaliste[2] du jeu (le nb secret est 9).



[1] Prompt in VS
[2] Notez l'absence d'interaction avec l'utilisateur pour le guider dans son choix.

Mise en place

Nous mettons en place un prototype. Son comportement est réduit au minimum.[3]

Nous allons créez un fichier  pour la mise en place de l'interface. Nous implémenterons la logique du code dans un fichier  . Nous finirons par implémenter le code de l'interaction avec l'utilisateur.

Code HTML

Créez un fichier index.html  pour implémenter **l'interface utilisateur**

index.html[4] 

1. <label>Guess</label>
2. <input type="number" **min=0 max=10** class="guessInput" value="0" />
3. <label htmlFor="guessPlay"></label>
4. <button class="guessPlay">Play</button>

Lig.2 : Notez l'utilisation des attributs min et max !

Code JS

Créez un fichier  guess.js pour implémenter **la logique**

 guess.js

---

[3] Les améliorations que vous pourrez apporter n'auront de limites que celles de votre imagination créatrice.
[4] Seul le code du body est ici indiqué. Je vous laisse compléter le code du fichier.

```
1.  const guessSecret = 9
2.      , guessLimit = 3;
3.  let guessCount = 1;
4.
5.  function checkGuess() {
6.
7.      let game_over = false;
8.      let userGuess = ?
9.
10.     if (userGuess === guessSecret) {
11.         game_over = true;
12.     } else if (guessCount === guessLimit) {
13.         game_over = true;
14.     } else {
15.         guessCount++;
16.     }
17.
18.     if (game_over) {
19.           // action
20.     }
21.}
```

🪛Principe pour nommer les variables :

Mot principal en tête : guess est le domaine ou le contexte, et Secret est la nature de la variable

🛟Remarquons dans le code de guess.js, qu'il manque quelque chose d'essentiel !

🚀Il manque simplement les appels à la fonction, sans cela, le programme ne va pas s'exécuter.

Il nous reste à ajouter l'interaction avec l'utilisateur.

DOM

Complétez un fichier guess.js pour implémenter **l'interaction utilisateur.**

Pour implémenter l'interaction avec l'utilisateur, nous allons définir un écouteur sur un élément de type bouton. Ainsi, dès que l'utilisateur cliquera sur ce bouton, la logique du jeu sera exécutée.

🕵🏽C'est génialement simple !

Il est donc important de comprendre que **l'utilisateur déclenche les étapes du jeu**, et **le navigateur réagit aux événements**, sans bloquer l'interface.

Voici donc le code guess.js complété.

les éléments de l'interaction sont mis en avant.

guess.js

```
1.  const guessSecret = 9
2.    , guessLimit = 3
3.    , buttonGuessPlay = document.getElementById('calcBtn')
4.    , guessInput = document.getElementById("guess");
5.
6.  let guessCount = 1;
7.
8.  function checkGuess() {
9.
10.    let gameOver = false;
11.    let userGuess = Number(guessInput.value);
12.
13.    if (userGuess === guessSecret) {
14.      guessInput.classList.add("win");
```

```
15.         gameOver = true;
16.     } else if (guessCount === guessLimit) {
17.         guessInput.classList.add("lost");
18.         gameOver = true;
19.     } else {
20.         guessCount++;
21.     }
22.
23.     if (gameOver) {
24.         setTimeout(function(){
25.             document.body.innerHTML = `<h1>Guess turns :
    ${guessCount}</h1>`;
26.         }, 500);
27.     }
28.}
29. buttonGuessPlay.addEventListener('click', checkGuess);
```

lig. 3-4 : La méthode getElementById() de Document renvoie un objet Element représentant l'élément <button id="calcBtn"> de la lig.4 du code HTML.

lig. 11 : La valeur saisie est convertie en nombre[5].

lig. 14 : La méthode classList.add(String [, String]): Ajoute les classes CSS spécifiées.

lig. 25 : Définir la valeur de innerHTML vous permet de remplacer aisément le contenu existant d'un élément par un nouveau contenu.

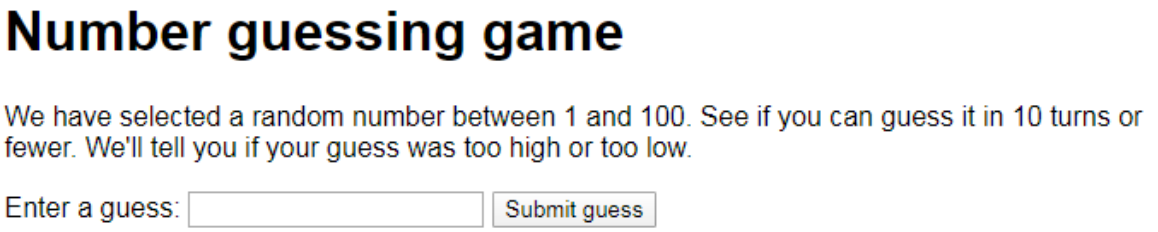
lig. 29 : checkGuess est un écouteur pour les événements click enregistrés à l'aide de addEventListener().

[5] Il faudrait ajouter des tests sur les valeurs saisies.

Amélioration de notre code

Mozilla à la rescousse !

Mais, sans plus attendre, je vous propose de regarder la solution avec une interface graphique proposée par MDN (lien)[6]



SuperDupont à la rescousse !

Je vous propose une solution, que je viens de rédiger pour vous, l'idée est de vous faire comprendre l'importance de l'organisation du code[7]. Vous pourrez à loisir ajouter une aide pour guider la saisie (valeur plus/loin grande).

🪛Cliquez pour jouer

[6] Vous lirez attentivement l'aide (lien)
[7] Peu importe si vous ne comprenez pas le code, il faut par contre comprendre qu'un effort dans l'organisation par type de code est mis en avant.

Étude du modèle MCV.

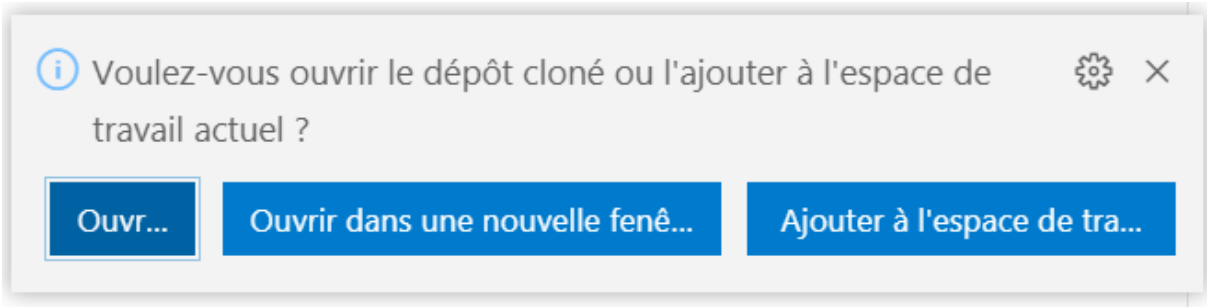
Vous allez télécharger le code sur github.

🪛Voici les instructions à suivre depuis Visual Studio Code.

Ouvrez Visual Studio Code puis téléchargez le dépôt GitHub de la formation :

1. Ouvrez la palette de commandes depuis le menu Afficher > Palette de commandes (raccourci clavier ⇧⌘P sur Mac ou Ctrl+Shift+P sous Windows/Linux) puis tapez git clone et validez avec ENTRÉE.
2. Si une erreur apparaît en bas à droite après avoir appuyé sur ENTRÉE il faut installer Git avant de continuer. Une fois que c'est fait suivez les instructions suivantes. 

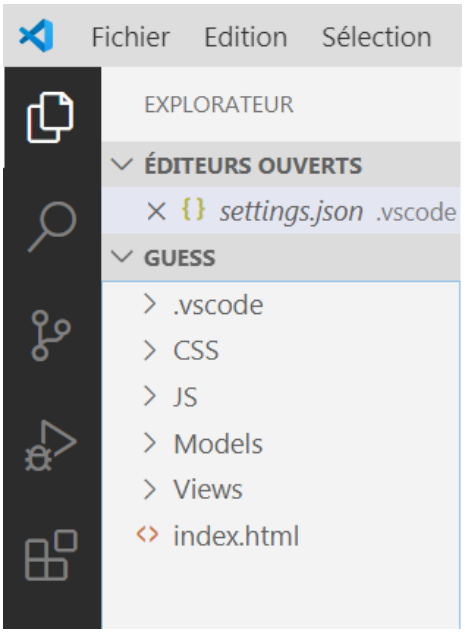
3. Copier-collez le lien du dépôt GitHub de la formation et validez : https://github.com/dupontdenis/MVC-Guess.git (attention à ne pas insérer d'espace en trop à la fin du lien !)
4. Visual Studio Code va vous demander dans quel répertoire vous voulez télécharger le dépôt GitHub du Jeu. Sélectionnez le répertoire de votre choix (par exemple Mes Documents), puis validez.
5. Une fois le téléchargement effectué vous aurez un nouveau répertoire Mes Documents/Guess. Visual Studio Code va vous demander dans une popup en bas à droite si vous voulez ouvrir le dépôt cloné : cliquez sur le bouton Ouvrir le dépôt.

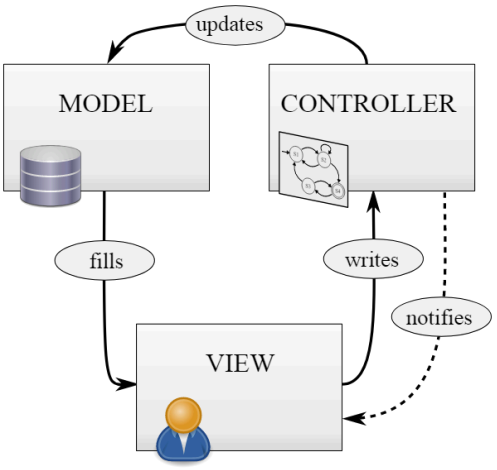


6. Si vous ne voyez pas la popup de l'étape précédente, utilisez le menu Fichier > Ouvrir puis sélectionnez le répertoire Documents/MVC-Guess. Sélectionnez bien le répertoire 📁MVC-Guess et pas 📁Documents avant de cliquer sur Ouvrir.

Je vous laisse découvrir le code, mais, observez principalement **la structure du programme.**

On recherche ici à approcher un modèle de conception classique en développement web : MVC[8]





Ainsi, si vous comparez les codes de Mozilla et de superDupont, vous observerez la différence notable de longueur du fichier index.html[9].

Comparaisons

MozillaDN : index.html	SuperDupont: index.html

<style>
html {
font-family: sans-serif;
}
 	
body {
width: 50%;
max-width: 800px;
	<label for="level">Choose your level:</label>
	<select name="guessLevel" id="guessLevel" class="guessLevel">
	<option

[8] https://fr.wikipedia.org/wiki/MVC
[9] Il n'y a pas moins de code, il est simplement mieux réparti.

```
      min-width: 480px;
      margin: 0 auto;
    }

    .lastResult {
      color: white;
      padding: 3px;
    }
  </style>
 </head>

 <body>
  <h1>Number guessing game</h1>

  <p>We have selected a random number between 1 and 100. See if you can guess it in 10 turns or fewer. We'll tell you if your guess was too high or too low.</p>

  <div class="form">
    <label for="guessField">Enter a guess: </label><input type="text" id="guessField" class="guessField">
    <input type="submit" value="Submit guess" class="guessSubmit">
  </div>

  <div class="resultParas">
    <p class="guesses"></p>
    <p class="lastResult"></p>
    <p class="lowOrHi"></p>
  </div>

  <script>
    let randomNumber = Math.floor(Math.random() * 100) + 1;
    const guesses = document.querySelector('.guesses');
    const lastResult = document.querySelector('.lastResult');
    const lowOrHi = document.querySelector('.lowOrHi');
    const guessSubmit = document.querySelector('.guessSubmit');
    const guessField = document.querySelector('.guessField');
    let guessCount = 1;
    let resetButton;
```

```
value="">--Choose an Level--</option>
   <option value="low">Low</option>
   <option value="high">High</option>
  </select>
  <label for="guessField">Enter a guess: </label>
  <input type="number" min=0 disabled id="guessField" class="guessField" placeholder="Set a Level First">
  <input type="submit" value="P" disabled class="guessSubmit disabled">
  <script src="JS/main.js" type="module"></script>
```

```
    function checkGuess() {
      let userGuess = Number(guessField.value);
      if (guessCount === 1) {
        guesses.textContent = 'Previous guesses: ';
      }

      guesses.textContent += userGuess + ' ';

      if (userGuess === randomNumber) {
        lastResult.textContent = 'Congratulations!
You got it right!';
        lastResult.style.backgroundColor = 'green';
        lowOrHi.textContent = '';
        setGameOver();
      } else if (guessCount === 10) {
        lastResult.textContent = '!!!GAME OVER!!!';
        lowOrHi.textContent = '';
        setGameOver();
      } else {
        lastResult.textContent = 'Wrong!';
        lastResult.style.backgroundColor = 'red';
        if(userGuess < randomNumber) {
          lowOrHi.textContent = 'Last guess was too
low!' ;
        } else if(userGuess > randomNumber) {
          lowOrHi.textContent = 'Last guess was too
high!';
        }
      }

      guessCount++;
      guessField.value = '';
      guessField.focus();
    }

    guessSubmit.addEventListener('click',
checkGuess);

    function setGameOver() {
      guessField.disabled = true;
      guessSubmit.disabled = true;
      resetButton =
document.createElement('button');
      resetButton.textContent = 'Start new game';
      document.body.appendChild(resetButton);
      resetButton.addEventListener('click',
resetGame);
```

}
 	
function resetGame() {
guessCount = 1;
const resetParas =
document.querySelectorAll('.resultParas p');
for(let i = 0 ; i < resetParas.length ; i++) {
resetParas[i].textContent = '';
}
 	
resetButton.parentNode.removeChild(resetButto
n);
guessField.disabled = false;
guessSubmit.disabled = false;
guessField.value = '';
guessField.focus();
lastResult.style.backgroundColor = 'white';
randomNumber = Math.floor(Math.random()
* 100) + 1;
}
</script>	

solution Mozilla	Ma* solution

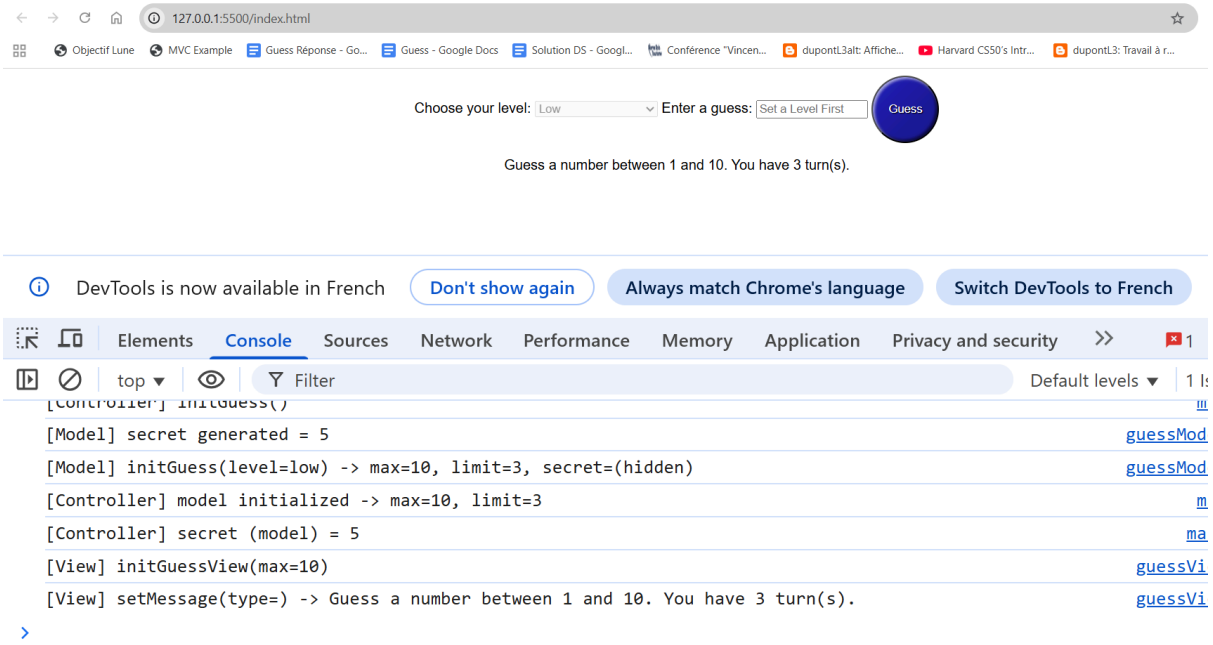
Le modèle MVC s'est ainsi rapidement imposé.

C'est un des principaux pattern utilisé dans le WEB.

🚀 Code en action

https://dupontdenis.github.io/MVC-Guess/

🥷Pensez à ouvrir la console pour observer les interactions du MVC



🪛Code

https://github.com/dupontdenis/MVC-Guess.git

TO-DO

Analysez le code pour ajouter la liste des coups joués !

index.html

```
22    <button id="restartBtn" class="restartBtn hidden" disabled>Restart</button>
23    <div id="message" class="message" role="status" aria-live="polite"></div>
24    <div id="history" class="history" aria-live="polite" aria-atomic="true"></div>
```

guessView.js

```
5   const message = document.getElementById("message");
6
7   const historyEl = document.getElementById('history');
```

```
15
16  function renderHistory(items = []){
17    historyEl.innerHTML = items.length
18      ? items.map((g,i)=>`<div class="item">${i+1}. ${g}</div>`).join('')
19      : '';
20    console.log('[View] renderHistory()', items);
21  }
```

```
55  function resetView() {
56      renderHistory([]);
57    guessSubmit.disabled = true;
58    guessSubmit.classList.add("disabled");
```

```
70  const View = {
71    getLevel() {
72      return guessLevel.options[guessLe
73    },
74    initGuessView,
75    renderHistory,
76    play
```

guessModel.js

```
32    get secret() {
33      return this._secret;
34    },
35    // list of previous guesses (in order)
36    previousGuesses: [],
37    addGuess: function (g) {
38      this.previousGuesses.push(g);
39      console.log(`[Model] addGuess -> ${g}`);
40    },
41    clearGuesses: function () {
42      this.previousGuesses = [];
43      console.log('[Model] clearGuesses()');
44    },
45    count: 1,
```

main.js

```
4   function initGuess() {
5     console.log("[Controller] initGuess()");
6     Guess.initGuess(View.getLevel());
7     Guess.clearGuesses();
8     console.log(
9       `[Controller] model initialized -> max=${Gue
10    );
11    console.log(`[Controller] secret (model) = ${G
12    View.initGuessView(Guess.max);
13    View.setMessage(
14      `Guess a number between 1 and ${Guess.max}.
15    );
16    View.renderHistory(Guess.previousGuesses);
17  }
```

```
47    // incorrect guess
48    if (Guess.count >= Guess.limit) {
49      View.lost();
50      View.setMessage(`Out of turns. The secret was $
51      View.gameOver();
52      console.log("[Controller] out of turns", userGu
53      return;
54    }
55    // record guess
56    Guess.addGuess(userGuess);
57    View.renderHistory(Guess.previousGuesses);
58
59    Guess.count++;
```

Style.css

```
}

.history{
    margin-top: .5rem;
    font-size: .95rem;
    color: #222;
}

.history .item{margin: .2rem 0}

```

🪛Modifiez l'affichage avec

```
function renderHistory(items = []) {
  // Render as reversed, arrow-joined string (most recent first)
  if (!historyEl) return;
  const reversed = items.slice().reverse();
  historyEl.textContent = reversed.length ? reversed.join(" \u2190 ") : "";
  console.log("[View] renderHistory()", reversed);
}
```

et pour CSS

```
.history {
    margin-top: .5rem;
    font-size: .95rem;
    color: #222;
    display: inline-block;
    padding: 0.25rem 0.5rem;
    background: rgba(0, 0, 0, 0.03);
    border-radius: 4px;
}
```

Have-fun

🪛Formatez le code pour avoir une liste des coups sous cette forme

Correct! The secret was 11.

34↘ 22↘ 10↗ 15↘ 12↘

Out of turns. The secret was 49.

45↗ 55↘ 88↘ 50↘ 66↘ 40↗ 45↗ 46↗ 47↗

Main.js

```
// record guess with direction
 const dir = userGuess < Guess.secret ? "low" : "high";
 Guess.addGuess(userGuess, dir);
 View.renderHistory(Guess.previousGuesses);
```

Model.js

```
previousGuesses: [],
 addGuess: function (g, dir = '') {
   this.previousGuesses.push({ value: g, dir });
   console.log(`[Model] addGuess -> ${g} (${dir})`);
 },
 clearGuesses: function () {
   this.previousGuesses = [];
   console.log('[Model] clearGuesses()');
 },
```

View.js

```
function renderHistory(items = []) {
  // Render in chronological order (oldest first). Use arrow-only markers:
  // too-low -> ↗ (U+2197), too-high -> ↘ (U+2198)
  if (!historyEl) return;
  const formatted = items.map((entry) => {
    const { value, dir } = entry || {};
    if (!dir) return `${value}`;
    if (dir === "low") return `${value}\u2197`;
    if (dir === "high") return `${value}\u2198`;
    return `${value}`;
  });
```

ou une version plus compacte !

```
const formatted = items.map(({ value, dir }) =>
  dir === "low"  ? `${value}\u2197` :
  dir === "high" ? `${value}\u2198` :
  `${value}`
);
  historyEl.textContent = formatted.length ? formatted.join("  ") : "";
  console.log("[View] renderHistory()", formatted);
}
```

CSS

```
.history {
    margin-top: .5rem;
    font-size: .95rem;
    color: #222;
```

```
    display: inline-block;
    padding: 0.25rem 0.5rem;
    background: rgba(0, 0, 0, 0.03);
    border-radius: 4px;
}
```