

# TP Noté : Programmation Web

17 mai 2024

Durée de l'examen : 90 minutes



## Quelques remarques :

- Documents et codes du cours/TP (ENSIIE) sont autorisés.
- Utilisation des appareils permettant la connexion internet est *interdite*.
- Utilisation de clé USB est *interdite*.
- Utilisation de l'ordinateur portable est *interdite*.
- *Attention 1* : ce sujet est en plusieurs pages et contient un « Cheat Sheet » pour vous aider !
- *Attention 2* : les fichiers complémentaires fournis pour les exercices de cet examen se trouvent sur *pydio.pedago* dans le répertoire du cours programmation web (*FISE\_PWRD12*), où vous trouverez *Prog\_web\_2024/Examen\_Final*
- Il vous faut créer un dossier (répertoire) à votre nom (sans espace, e.g., **NOM\_Prénom**) et mettre tous vos fichiers et dossiers de réponse dans ce dossier (répertoire)
- Une fois vous êtes prêt(e) à soumettre votre réponse aux questions, il faut compresser ce dossier (répertoire) et uploader sur <http://exam.ensiie.fr/> dans le dossier nommé : **prog\_web\_tp\_note\_mai\_2024**
- *Attention 3* : Il faut faire attention à ne pas soumettre de dossier vide, des faux fichiers, ou des fichiers endommagés !
- *Attention 4* : aucun dépôt après la fin du TP noté ne sera accepté.

## Exercice 1 : projet de vidéo

10+2 points

► Pour répondre à cet exercice, vous devez utiliser exclusivement HTML, CSS et Javascript (i.e., seulement HTML, CSS et Javascript (pas d'autre chose)).

Le but est de réaliser une page permettant d'afficher une vidéo en arrière-plan avec la possibilité de faire une pause (ON / OFF) sur la vidéo par un bouton.

Vérifiez l'encodage de votre page : attention à déclarer dans votre document HTML que l'encodage est UTF-8.

Bonus (2 points) : une image .gif pourrait être affichée avant que la vidéo commence.

Les codes CSS, la vidéo et l'image .gif ainsi que des guides sur les fichiers HTML et Javascript sont fournis (voir sur *pydio.pedago* dans le répertoire du cours programmation web (*FISE\_PWRD12*), où vous trouverez *Prog\_web\_2024/Examen\_Final*).

Voir aussi les figures suivantes pour avoir une meilleure idée sur la sortie de cet exercice.

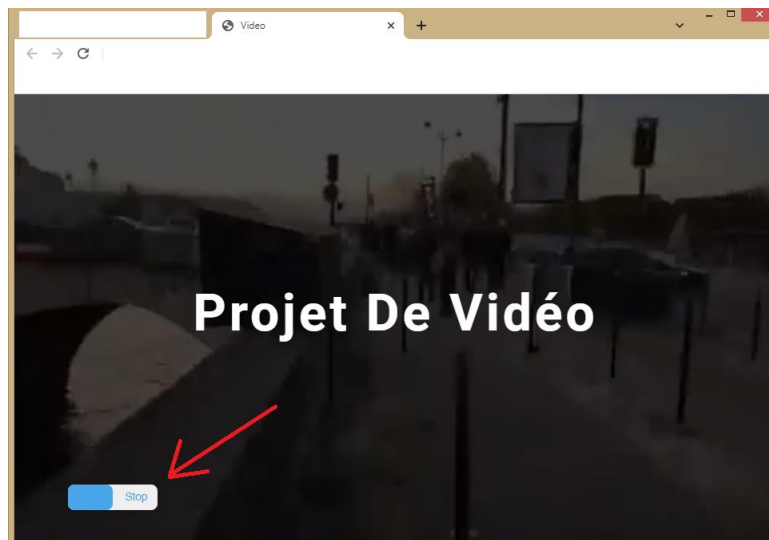


Figure : la flèche indique le bouton

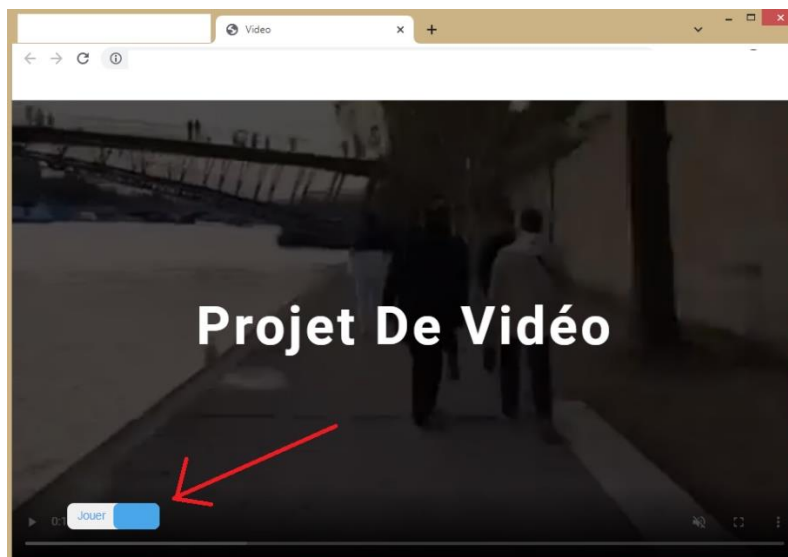


Figure : la flèche indique le bouton

## Exercice 2 : un jeu simple

**10 points**

► Pour répondre à cet exercice, vous devez utiliser exclusivement HTML, CSS et Javascript (i.e., seulement HTML, CSS et Javascript (pas d'autre chose)).

Le but est de réaliser une page permettant de jouer à un jeu simple. Plus précisément, il y a un bouton, un compteur du temps (initié à 10 secondes) et un compteur des points cumulés. En cliquant sur le bouton un nombre aléatoire entre 0 et 10 est généré qui est ensuite ajouté aux points cumulés du joueur. Le joueur a 10 secondes du temps pour cliquer autant de fois qu'il peut sur le bouton afin de générer des points. Dès que du temps 10 secondes est terminé, un message est affiché indiquant les points totaux (cumulés) du joueur et le bouton est désactivé.

Vérifiez l'encodage de votre page : attention à déclarer dans votre document HTML que l'encodage est UTF-8.

Voir les figures suivantes pour avoir une meilleure idée sur le jeu et la sortie de cet exercice.

## Jeu de Clique

Vous avez 10 secondes pour cliquer sur le button autant de fois que vous pouvez !

Cliquer !

Points : 0

Il vous reste (secondes): 10

## Jeu de Clique

Vous avez 10 secondes pour cliquer sur le button autant de fois que vous pouvez !

Cliquer !

Points : 43

Il vous reste (secondes): 7

Cette page indique

Fin du jeu ! Vous avez cumulé 62 Points !

OK

## Jeu de Clique

Vous avez 10 secondes pour cliquer sur le bouton autant de fois que vous pouvez !

Cliquer !

Points : 62

Il vous reste (secondes): 1

## Jeu de Clique

Vous avez 10 secondes pour cliquer sur le bouton autant de fois que vous pouvez !

Cliquer !

Points : 62






Il vous reste (secondes): 0

# Cheat Sheet

Voici quelques consignes qui peuvent vous aider !

- Returns a random integer from 0 to 9:  
$$\text{Math.floor}(\text{Math.random}() * 10)$$
- Un moment donné vous aurez éventuellement besoin de désactiver quelque chose, voici son attribue :  
... `.disabled = true`

## Browser Support

Element					
<button>	Yes	Yes	Yes	Yes	Yes

## Attributes

Attribute	Value	Description
<a href="#">autofocus</a>	autofocus	Specifies that a button should automatically get focus when the page loads
<a href="#">disabled</a>	disabled	Specifies that a button should be disabled
<a href="#">form</a>	form_id	Specifies which form the button belongs to

## HTML DOM Document addEventListener()

[< Previous](#)[< Document Object Reference](#)[Next >](#)

### Examples

Add a click event to the document:

```
document.addEventListener("click", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World";
}
```

[Try it Yourself »](#)

Simpler syntax:

```
document.addEventListener("click", function(){
  document.getElementById("demo").innerHTML = "Hello World";
});
```

[Try it Yourself »](#)

# HTML DOM Element classList

[< Previous](#)[< Element Object Reference](#)[Next >](#)

## Example

Add a "myStyle" class to an element:

```
const list = element.classList;  
list.add("myStyle");
```

[Try it Yourself »](#)

Remove the "myStyle" class from an element:

```
const list = element.classList;  
list.remove("myStyle");
```

## classList Properties and Methods

Name	Description
<u><a href="#">add()</a></u>	Adds one or more tokens to the list
<u><a href="#">contains()</a></u>	Returns true if the list contains a class
<u><a href="#">entries()</a></u>	Returns an Iterator with key/value pairs from the list
<u><a href="#">forEach()</a></u>	Executes a callback function for each token in the list
<u><a href="#">item()</a></u>	Returns the token at a specified index
<u><a href="#">keys()</a></u>	Returns an Iterator with the keys in the list
<u><a href="#">length</a></u>	Returns the number of tokens in the list
<u><a href="#">remove()</a></u>	Removes one or more tokens from the list
<u><a href="#">replace()</a></u>	Replaces a token in the list
<u><a href="#">supports()</a></u>	Returns true if a token is one of an attribute's supported tokens
<u><a href="#">toggle()</a></u>	Toggles between tokens in the list
<u><a href="#">value</a></u>	Returns the token list as a string
<u><a href="#">values()</a></u>	Returns an Iterator with the values in the list

# HTML Audio/Video DOM Reference

[< Previous](#)

## HTML Audio and Video DOM Reference

The HTML5 DOM has methods, properties, and events for the `<audio>` and `<video>` elements.

### HTML Audio/Video Methods

Method	Description
<a href="#"><code>addTextTrack()</code></a>	Adds a new text track to the audio/video
<a href="#"><code>canPlayType()</code></a>	Checks if the browser can play the specified audio/video type
<a href="#"><code>load()</code></a>	Re-loads the audio/video element
<a href="#"><code>play()</code></a>	Starts playing the audio/video
<a href="#"><code>pause()</code></a>	Pauses the currently playing audio/video

### HTML Audio/Video Events

Event	Description
<a href="#"><code>abort</code></a>	Fires when the loading of an audio/video is aborted
<a href="#"><code>canplay</code></a>	Fires when the browser can start playing the audio/video
<a href="#"><code>canplaythrough</code></a>	Fires when the browser can play through the audio/video without stopping for buffering
<a href="#"><code>durationchange</code></a>	Fires when the duration of the audio/video is changed
<code>emptied</code>	Fires when the current playlist is empty
<a href="#"><code>ended</code></a>	Fires when the current playlist is ended
<a href="#"><code>error</code></a>	Fires when an error occurred during the loading of an audio/video
<a href="#"><code>loadeddata</code></a>	Fires when the browser has loaded the current frame of the audio/video
<a href="#"><code>loadedmetadata</code></a>	Fires when the browser has loaded meta data for the audio/video
<a href="#"><code>loadstart</code></a>	Fires when the browser starts looking for the audio/video
<a href="#"><code>pause</code></a>	Fires when the audio/video has been paused
<a href="#"><code>play</code></a>	Fires when the audio/video has been started or is no longer paused

<a href="#"><u>playing</u></a>	Fires when the audio/video is playing after having been paused or stopped for buffering
<a href="#"><u>progress</u></a>	Fires when the browser is downloading the audio/video
<a href="#"><u>ratechange</u></a>	Fires when the playing speed of the audio/video is changed
<a href="#"><u>seeked</u></a>	Fires when the user is finished moving/skipping to a new position in the audio/video
<a href="#"><u>seeking</u></a>	Fires when the user starts moving/skipping to a new position in the audio/video
<a href="#"><u>stalled</u></a>	Fires when the browser is trying to get media data, but data is not available
<a href="#"><u>suspend</u></a>	Fires when the browser is intentionally not getting media data
<a href="#"><u>timeupdate</u></a>	Fires when the current playback position has changed
<a href="#"><u>volumechange</u></a>	Fires when the volume has been changed
<a href="#"><u>waiting</u></a>	Fires when the video stops because it needs to buffer the next frame

## Window setTimeout()

[< Previous](#)
[< Window Object Reference](#)
[Next >](#)

### Examples

Wait 5 seconds for the greeting:

```
const myTimeout = setTimeout(myGreeting, 5000);
```

[Try it Yourself »](#)

Use clearTimeout(myTimeout) to prevent myGreeting from running:

```
const myTimeout = setTimeout(myGreeting, 5000);

function myStopFunction() {
  clearTimeout(myTimeout);
}
```

[Try it Yourself »](#)