TP JS Notions avancées
Objet *event* – Iterators – Arrow Function

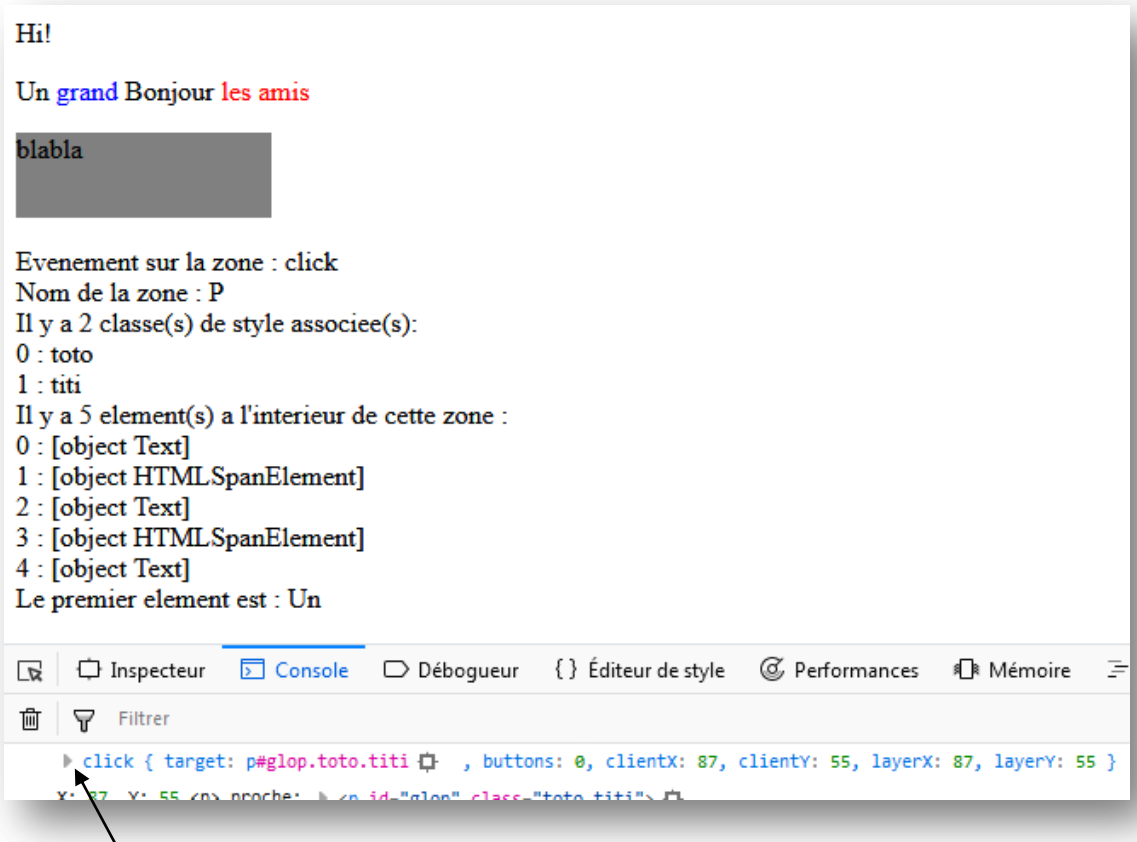
Objet event :

Se familiariser avec l'objet EVENT (Evènement) généré par la souris ou le clavier, ou tout autre dispositif générant des évènements sur le navigateur.
Cet objet contient une foule d'informations dont certaines seront utiles au programmeur.

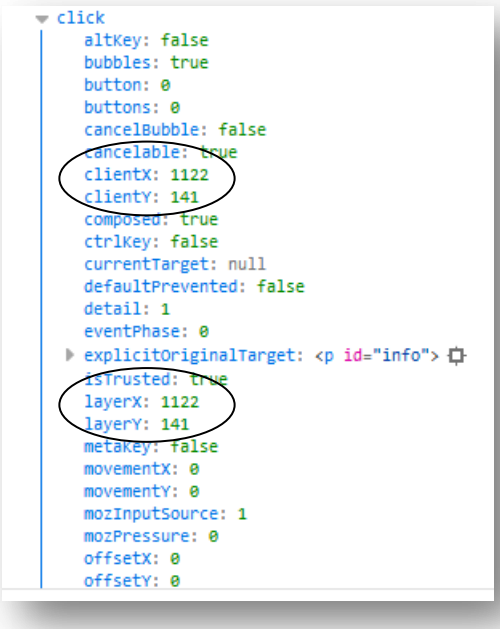
TRAVAIL 1 : Ci-joint le programme de test … à tester !!! en considérant les explications ci-dessous.

Pour « voir » l'objet EVENT en détail, il faut passer le navigateur en mode « inspection » pour accéder à sa console de *debug* (touche <F12>)

Si vous cliquez sur le mot « Bonjour », vous aurez ce résultat (navigateur Firefox) :



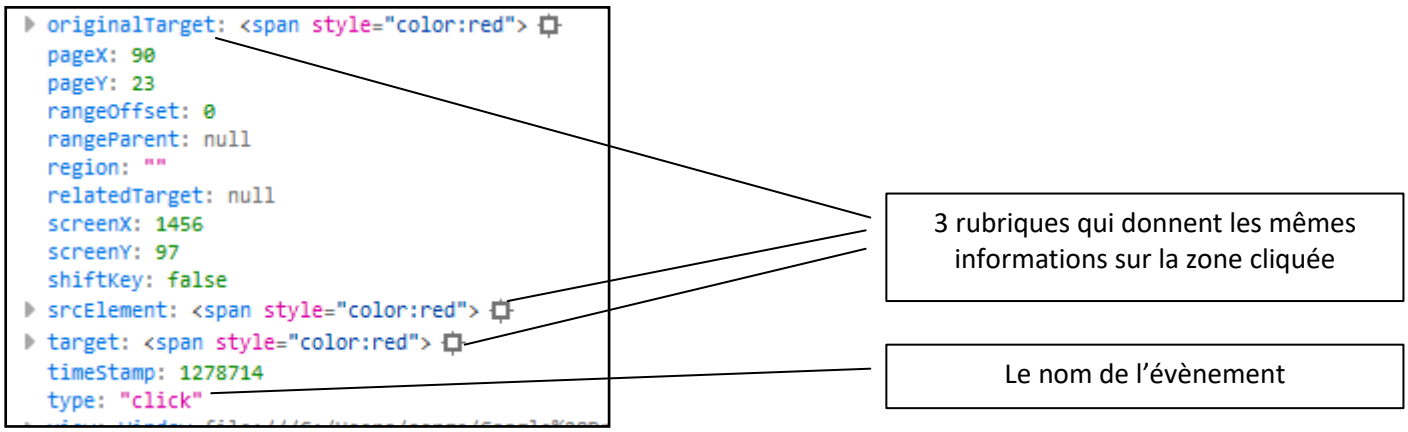
La console fournit un **résumé condensé** de l'évènement.
En cliquant sur le **triangle gris**, on constatera qu'il y a beaucoup plus d'informations !!!



On remarque par exemple des informations sur la position de la souris au moment du click :

NB : vous remarquez que les mêmes informations apparaissent à **plusieurs endroits** avec des noms de variables différents.
Cela peut correspondre soit à des nuances de contexte, soit à une compatibilité avec d'anciennes versions de JS).

Les informations suivantes sont utiles pour mieux connaitre la zone HTML sur laquelle on a cliqué :

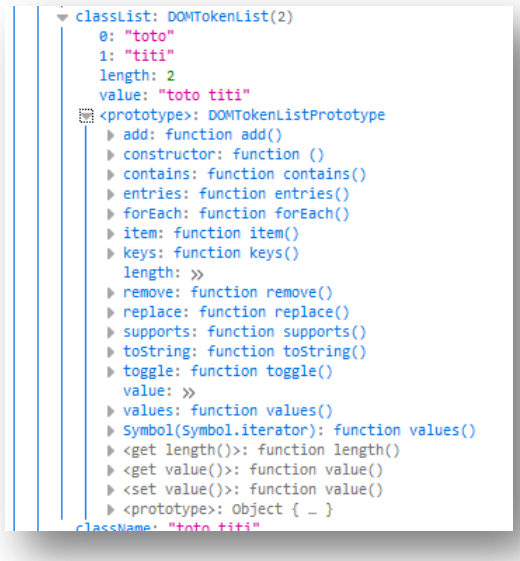


Nous avons déjà l'habitude d'utiliser les informations fournies par « target ». C'est un objet contenu dans l'objet Event. Dans l'exemple ci-dessus, il pointe une balise

En détaillant « target », on obtient toutes sortes d'informations utiles à un programmeur :

- *attributes* : objet tableau contenant la liste des attributs de la balise
- *classList* : objet tableau contenant uniquement la liste des classes de styles utilisés
- *childNodes* : objet tableau contenant une liste de sous éléments inclus dans la balise (sous-balises, texte, …)
- *firstChild.data* : texte contenu au début de la balise
- *parentElement* et *parentNode* : renseignements sur la balise parente (contenant notre balise)
- … etc …

Quand l'information est sous forme de variable complexe (tableau, objet tableau, …), la rubrique *prototype* indique les fonctions et méthodes utilisables par le programmeur (il suffit ensuite de consulter la documentation) :



Dans l'exemple ci-contre, la propriété « classList » est un objet de type tableau, et on remarque les méthodes :

- *forEach* : voir travail1.html
- *add* , *remove, replace, …*
- *contains* qui renvoi vrai ou faux en fonction d'une recherche :

```
if ( e.target.classList.contains('toto') )
      e.target.classList.replace("toto","nouveau");
else
      e.target.classList.add("nouveau");
```

Explication : Si la balise **contient** la classe « toto », on la **remplace** par la classe « nouveau », sinon on **ajoute** simplement la classe « nouveau ». Ce script permet de créer des effets dynamiques sur la déco.

Autre exemple : La méthode *closest(selectorCSS)* qui renvoi une référence à la balise parente la plus proche de la balise indiquée, sur condition fixée par un selecteur (même syntaxe que pour le CSS).
Exemple : `e.target.closest("p");` renvoi une référence à la balise <p> parente de l'emplacement cliqué (voir travail1.html)

Dernier exemple : Au fin fond des *prototype* de l'objet « target », on trouve la méthode *getBoundingClientRect( )* qui donne une information sur les dimensions de la zone autour de laquelle on clique.
Cette information est plus détaillée que les simples *e.target.offsetTop/Height/Left/Width* qu'on sait déjà utiliser.

Test : Afficher sur la console le contenu *getBoundingClientRect* de l'objet cliqué.

Iterators, Fonctions anonymes, Arrow Functions :

A FAIRE : Copier le fichier **travail1.html** en **travail2.html**

Ce fichier contient un exemple d'utilisation d'une **fonction anonyme** associée à un **iterator** :

```
e.target.classList.forEach( function (value, index) { msg += "<br/>" + index + " : " + value; } );
```

Comprendre cette ligne :

`e.target.classList` : représente un objet tableau (object Array).
`forEach` est un ***iterator*** de cet objet tableau
`function` est une fonction anonyme qui n'existe que dans le *forEach*

Un objet Tableau JS contient des méthodes spéciales nommées ***iterator*** (Voir le document JS ITERATOR dans le dossier de ce TP).

Un *iterator* (itération = boucle) effectue un traitement sur toutes les données d'un tableau.
Il génère automatiquement des variables qui suivent les étapes de l'itération : la valeur en cours de traitement, son index, et une référence globale au tableau (au cas où…).

On crée une fonction anonyme quand une fonction ne doit exister que dans un contexte particulier (puisqu'elle n'a pas de nom). Cela donne une écriture + compacte.

La fonction anonyme de l'exemple :

```
function (value, index) { msg += "<br/>" + index + " : " + value;}
```

Value et *index* sont des arguments automatiquement générés par l'itération *forEach*
La fonction ajoute dans une variable *msg* du texte html de la forme
 XX : YYYY pour chaque éléments de donnée de l'objet tableau.

Une fonction « classique » (séparée et nommée) aurait donné ce code :

```
e.target.classList.forEach( action (value, index));

function action (value, index) {
        msg += "<br/>" + index + " : " + value;
}
```

Arrow Function : C'est une nouvelle façon (norme ES6) d'écrire une fonction (anonyme). Cela peut sembler relever de la torture psychologique mais en fait, cette notation est très utilisée en mode objet JS. L'écriture est encore + compacte. De plus une *arrow function* règle notamment le problème du contexte d'exécution pour la variable *this*.
(Cours en ligne avec exercices : http://78.122.136.24)

```
function (value, index) { msg += "<br/>" + index + " : " + value; }
```

devient avec les *arrow function* :

```
(value, index)  =>   msg += "<br/>" + index + " : " + value;
```

TRAVAIL 2 :
Modifier *travail2.html* en remplaçant les fonctions anonymes par des *arrow functions*.

JS Objet : Créer une classe JS utilisant *arrow function* et *iterator*

Pour cette partie, vous devez savoir manipuler les objets JS (ex : TP-JS-OBJ-HORLOGE)

TRAVAIL 3 : Une fenêtre « modal » intelligente.

On appelle « modal » une fenêtre de type « pop-up » qui contient des instructions ou un formulaire.

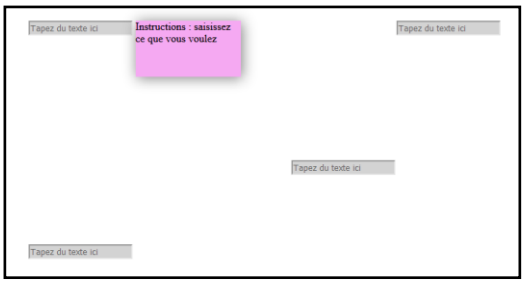
Utiliser le fichier **travail3.html**. La fenêtre modal est une gérée par la classe maClasseModal
La fenêtre modal est pour l'instant cachée (CSS : display :none).

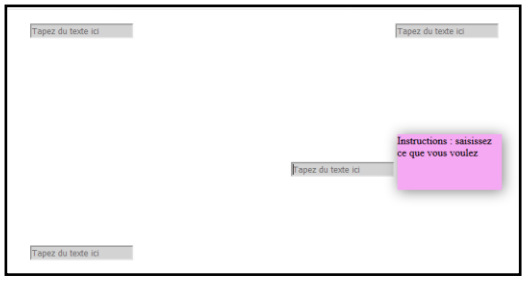
Pour l'afficher par script : `this.modal.style.display = 'block';`
L'évènement « click » est associé à chaque balise <input> (lignes 29-31 du script).

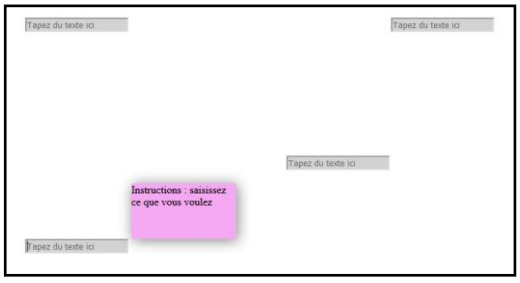
Compléter la classe pour que la « modal » s'affiche à côté de l'<input> cliqué, selon les règles suivantes :

En priorité, s'il y a la place, la « modal » s'affiche à droite de l'input.
Sinon, elle s'affiche côté gauche (s'il y a la place)

En priorité, elle s'affiche centrée verticalement sur l'<input>.
Sinon au dessus ou en dessous suivant la place disponible :







TRAVAIL 4 :
En HTML, remplir la « modal » avec un tableau de chiffre (<table><tr><td> …)
Modifier la classe : quand on clique sur un chiffre, on inscrit la valeur dans l'<input>

Le modal est géré par la classe ; il n'y a pas de script supplémentaire dans la zone « Programme Principal »