On se propose de découvrir dans ce Tp, le langage de programmation ***JavaScript*** que l'on rajoutera au code *html* et *css* de nos pages web. Ce nouveau langage n'est pas très différent des autres langages. On pourra ainsi réutiliser tous vos acquis liés au langage ***Python***.

Pourquoi ne pourrait-on pas insérer directement du python dans nos pages web ? Et bien parce que l'évolution du web depuis une trentaine d'années, a fait que les navigateurs, qui permettent sur votre ordinateur de communiquer avec les serveurs pour demander et recevoir des pages webs, peuvent uniquement interpréter le *JavaScript* et pas un autre langage. *JavaScript* est apparu en 1995 et a évolué depuis en même temps que les différents navigateurs : Internet Explorer et Safari tout d'abord puis Firefox, Chrome, … .

Et le langage Php évoqué la semaine dernière sur le Tp des formulaires ? Le langage *Php* est le principal langage utilisé sur les **serveurs**, pour par exemple recevoir les données d'un formulaire qui a été envoyé et stocker ensuite ces données dans une base de données, qui se trouve toujours sur le serveur. Là aussi, historiquement dans l'évolution du web, c'est ce langage qui s'est imposé coté **serveurs**. Php a vu le jour bien avant *JavaScript* d'ailleurs, car à l'apparition du web, les premiers navigateurs IE et Safari se contentaient uniquement d'interpréter *html* et *css* pour afficher une page web. Ils n'exécutaient aucun code avec des structures *if* ou des boucles *for i in range(n)* par exemple. Lorsqu'il y avait de l'interactivité sur une page, ce type de code était exécuté en *php* **sur le serveur** …. d'où des temps de réponse très longs et souvent insupportables …. à l'époque.

L'avantage premier du *JavaScript* est ainsi d'avoir des lignes de codes qui sont exécutées **directement par le navigateur** sur **votre ordinateur**, ce qui permet d'obtenir une interactivité instantanée.

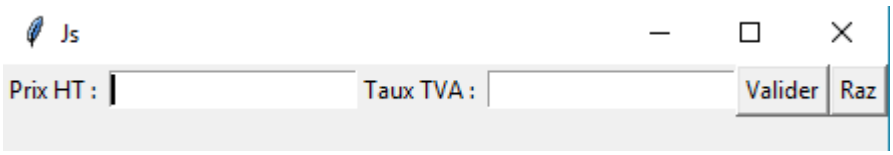
On se propose ici de voir tout cela en se fixant l'objectif modeste, de réaliser une page web qui calcule le prix TTC d'un produit, à partir du prix HT et du taux de Tva saisis par l'utilisateur.

1. Que va-t-on faire concrètement ?

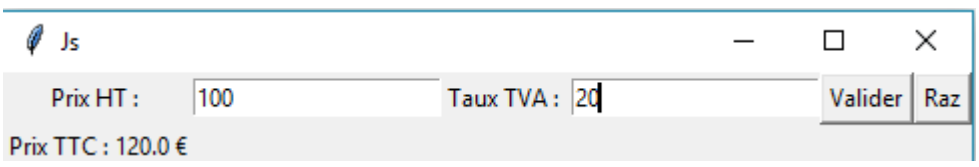
⇨ Télécharger le dossier js.zip : https://nsibranly.fr/documents/premiere/js.zip

⇨ Ouvrir ***Pyzo*** (à partir du raccourci bureau du lycée …) et exécuter le fichier *tva.py* :

- A l'ouverture la fenêtre *Tkinter* suivante s'affiche :

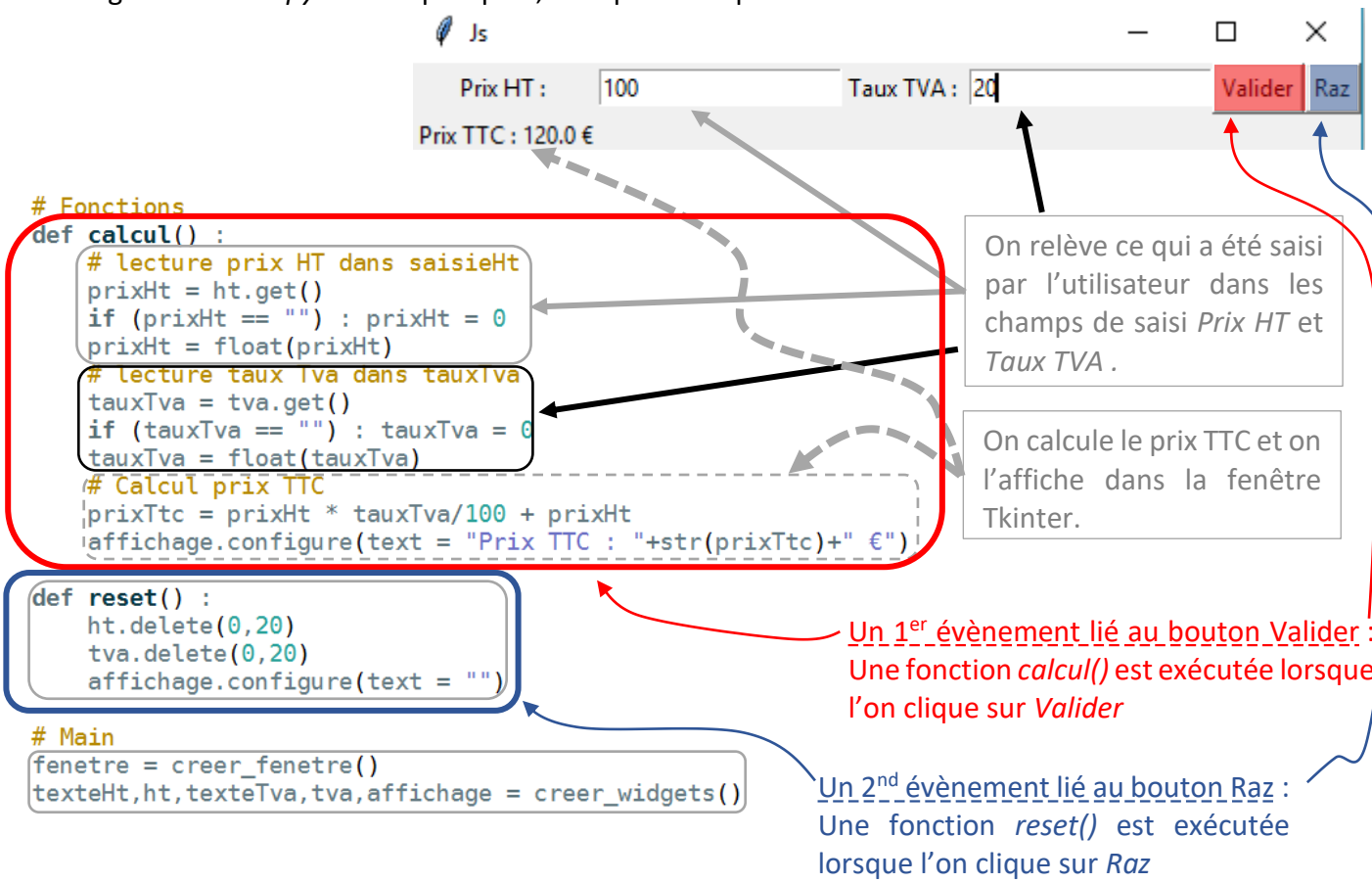


- On peut saisir un prix HT et un taux de TVA. En cliquant sur le bouton *Valider* le prix TTC est calculé et est affiché :

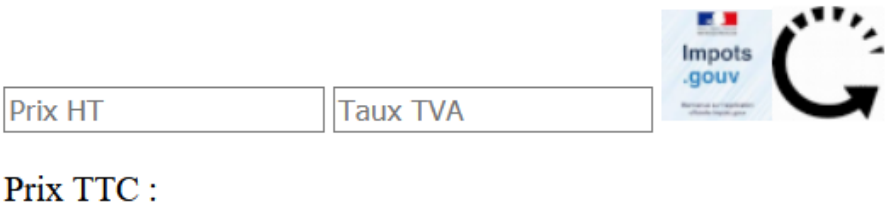


100 € + 20% de 100 € donnent : $100 + \frac{20}{100} \times 100 = 120$ €

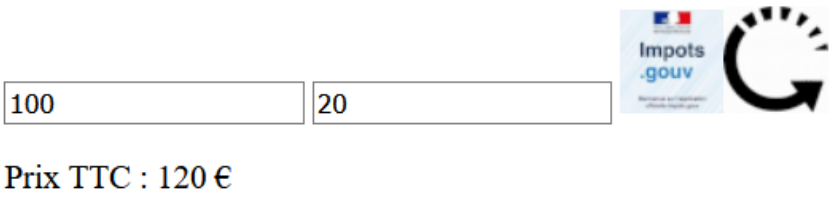
⇨ Si on regarde le code *python* de plus près, on a plusieurs parties distinctes :



On se propose dans ce Tp de faire pratiquement la même chose, **en remplaçant *pyzo* par votre navigateur** et le code ***tva.py*** par un code ***tva.html*** qui sera donc exécuté par votre navigateur. Ce code permettra d'obtenir la page suivante (pas de Css ici pour simplifier dans un 1er temps) :



Après saisie du prix HT et du taux de TVA, en cliquant sur l'image  on obtient la page ci-contre :



En cliquant sur l'image  on réinitialise les champs **sans avoir à recharger la page**.

Le code complet *html + Js* pourra alors être mis en ligne et pourra être exécuté par n'importe quel ordinateur connecté au web. C'est ici l'avantage par rapport à une programmation *python*.

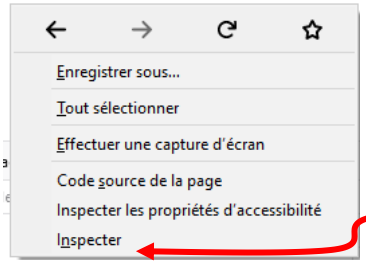
2. On commence ?

⇨ Vous pouvez fermer *pyzo* et ouvrir *Visual Studio Code* et votre *navigateur*.

⇨ Dans *Visual Studio Code*, créer un nouveau fichier à enregistrer sous le nom *tva.html* dans le répertoire contenant les images *gouv.png* et *raz.png* téléchargées avant. Faire exécuter par votre navigateur ce fichier *html*, qui est vide pour l'instant.

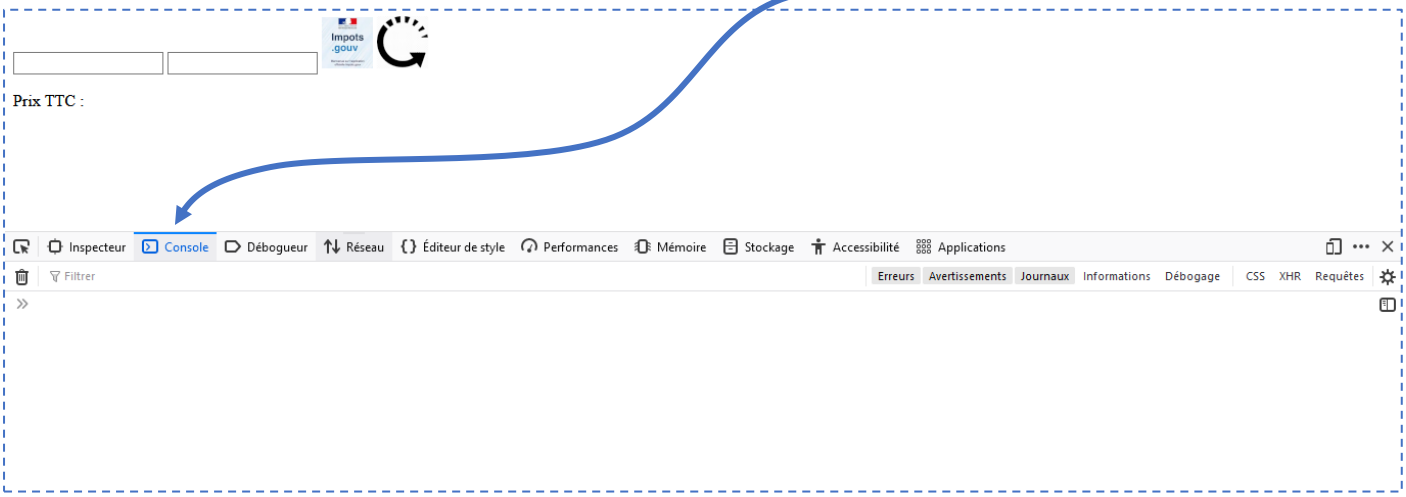
⇨ Ecrire le script *html* ci-contre qu'on peut qualifier de minimaliste et qui permet déjà d'obtenir les principaux éléments de notre page :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <input type="text">
        <input type="text">
        <img src="gouv.png">
        <img src="raz.png">
        <p>Prix TTC : </p>
    </body>
</html>
```



⇨ Ouvrir la ***console web*** de votre navigateur. Pour cela, le plus simple est de faire un click droit sur la page affichée par le navigateur et choisir : ***Inspecter***.

Une fenêtre apparait. Dans celle-ci, se placer dans l'onglet ***Console***.



⇨ Cette console est l'équivalent du shell de *pyzo*. Elle permet d'exécuter des lignes de codes écrites en langage *JavaScript*. Par exemple, tester les lignes JS ci-contre :

```
>> 3+5
<- 8
>> a = "super le js"
<- "super le js"
>> a
<- "super le js"
```

On continue à tester un peu le *JS* et on fait en même temps le parallèle avec *python*. On y va ….

3. Le JS c'est comme python .....

⇨ Tester les lignes suivantes :

```
>> temperature = 17
<- 17
>> if (temperature  > 25){humeur = "content"} else {humeur = "mécontent"}
<- "mécontent"
>> humeur
<- "mécontent"
>> temperature = 30
<- 30
>> if (temperature  > 25){humeur = "content"} else {humeur = "mécontent"}
<- "content"
>> humeur
<- "content"
```

.... Finalement c'est proche du python. En JS, la structure *if* ... s'écrit par contre un peu différemment.

Mais écrire les lignes dans la console n'est pas très pratique. On va à présent les écrire dans le fichier *tva.html* entre 2 balises *<script>* et *</script>*.

Ces balises *<script>* sont de préférence placées en fin de fichier afin de permettre au navigateur de lire en priorité la partie *html*. Cela permet d'afficher le contenue de la page en 1er lorsqu'elle commence à être lue.

⇨ Compléter le fichier *tva.html* en rajoutant ce script écrit en JS :

L'équivalent python de ce script est donné ci-dessous. On voit que c'est très proche :

```python
temperature = float(input("Saisir la température : "))
if temperature > 25 :
    humeur = "content"
else :
    humeur = "mécontent"
print(humeur)
```

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <input type="text">
        <input type="text">
        <img src="gouv.png">
        <img src="raz.png">
        <p>Prix TTC : </p>
    </body>
    <script>
        let temperature = prompt('Saisir la temperature');
        let humeur;
        if (temperature  > 25){
            humeur = "content";
        }else{
            humeur = "mécontent";
        }
        console.log(humeur);
    </script>
</html>
```

A RETENIR pour l'instant : en JavaScript,

- A la fin de chaque instruction, on met un **;**
- Les instructions dans la structure *if* sont entre ***{}***
- Pour afficher un résultat dans la console web du navigateur, on utilise l'instruction : ***console.log()***
- Lorsque l'on utilise une variable pour la première fois, on écrit le mot clé ***let*** juste devant.

⇨ Tester ce script en rechargeant votre page *tva.html* dans votre navigateur.

⇨ Tester à présent, de la même façon, le script JS ci-contre qui comprend une structure *for* et dont on donne aussi un équivalent python :

```python
for i in range(30) :
    temperature = "Oggi sono "+str(i)+ " gradi"
    print(temperature)
```

```html
<script>
    for(let i = 0 ; i < 30 ; i++){
        let temperature = "Oggi sono "+i+" gradi";
        console.log(temperature);
    }
</script>
```

4. Si c'est pour refaire du python, c'est pas la peine ….. mais non, avec *JS* on va pouvoir utiliser les éléments graphiques *html* de la page web et ça va tout changer.

⇨ Effacer le code JS du fichier *tva.html* afin de revenir au script basique ci-contre :

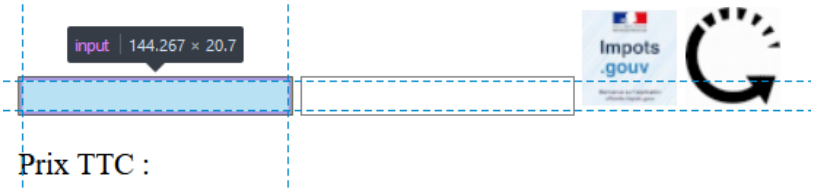
```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <input type="text">
        <input type="text">
        <img src="gouv.png">
        <img src="raz.png">
        <p>Prix TTC : </p>
    </body>
    <script>

    </script>
</html>
```

Le JS va nous permettre d'interagir avec les éléments *html* de la page web. Pour cela, on utilise les méthodes *querySelector()* et *querySelectorAll()* appliquée à votre document html …..

⇨ Dans la console, exécuter les lignes suivantes les unes après les autres (ne pas recharger la page entre temps, car cela réinitialiserait toutes les variables).

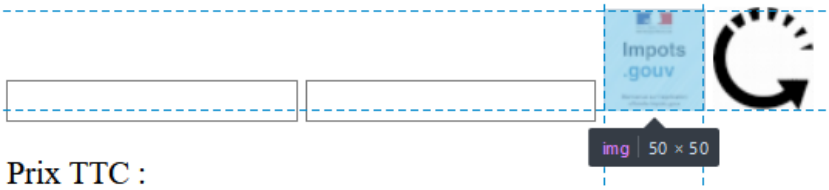
On sélectionne le 1er <input>

```
» a = document.querySelector("input")
← ▸ <input type="text">
» a
← ▸ <input type="text">
```



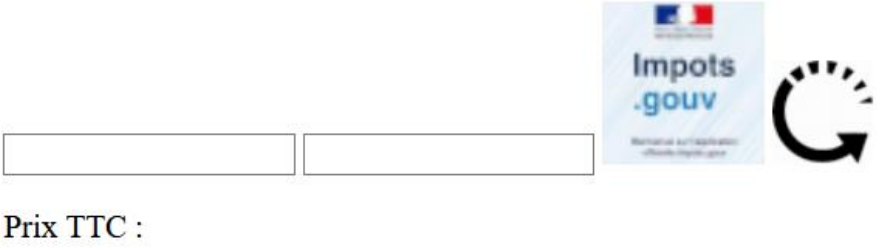
On sélectionne le 1er <img>

```
» a = document.querySelector("img")
← ▸ <img src="gouv.png">
» a
```



On modifie le Css de cet <img>

```
» a.style.width="80px"
← "80px"
```



On peut ainsi agrandir l'image depuis la console. Par contre en rechargeant la page, on revient à la taille initiale.

On sélectionne le 2nd <img> et on change son Css

```
a = document.querySelector("img[src='raz.png']")
<img src="raz.png">
a.style.border="1px solid red"
"1px solid red"
```

Prix TTC :

On sélectionne les 2 éléments <input> pour les stocker dans une liste a. On modifie le contenue du 2nd <input>

```
a=document.querySelectorAll('input')
NodeList [ input , input ]
a[0]
<input type="text">
a[1]
<input type="text">
a[1].value="trop fort"
"trop fort"
```

trop fort

Prix TTC :

On sélectionne le paragraphe <p> et on modifie son contenu

```
a = document.querySelector("p")
<p>
a.textContent = "Comment on appelle un chien sans patte ? "
"Comment on appelle un chien sans patte ? "
```

Comment on appelle un chien sans patte ?

En résumé :

- la méthode ***querySelector()*** appliquée à l'objet ***document*** permet de sélectionner n'importe quel élément *html* d'un page web. Cette méthode retourne un **objet**.

Syntaxe

```
element = document.querySelector(sélecteurs);
```

Nom quelconque que l'on donne à **l'objet** JS relatif à l'élément html sélectionné.

Même sélecteurs que ceux utilisés pour repérer les éléments html dans le Css. Exemples :

- Repérage par le type d'élément *html* : *''p''* , *''input''* , *''input[src='raz.png']''* , ….
- Repérage par un sélecteur *id* unique : *''#toto''*
- Repérage par un sélecteur de *class* : *''.toto''*

Si plusieurs éléments *html* correspondent au sélecteur :

- avec *querySelector(),* seul le 1er rencontré est sélectionné
- avec *querySelertorAll()* la variable retournée sera une liste [] indexée qui les contiendra tous.

- on peut alors en changer le Css en modifiant la propriété *style* de cet **objet**,
- on peut alors changer son contenu texte en modifiant la propriété *textContent* de cet **objet**,
- avec la propriété *value* d'un **objet** de type *<input type= 'text'>*, on accède à son contenu saisi.

5. Pour réaliser notre page web interactive on devra utiliser des fonctions. Est-ce-possible en JS ?

Pas de problème, là encore c'est très proche du python …. la syntaxe est par contre assez différente :

```python
def une_fonction(a) :
    b = float(b) + 1
    c = a + b
    return c

b = "2"
nombre = une_fonction(7)
```

L'équivalent JavaScript de ce bout de code python est :

```javascript
function une_fonction(a) {
    b = parseFloat(b) + 1;
    let c = a + b;
    return c;
}
let b = "2";
let nombre = une_fonction(7);
```

En JS, le mot clé python ***def*** est remplacé par ***function***, le bloc d'instructions n'est plus repéré par l'indentation, mais par des accolades { }. Les instructions ne sont plus séparées par un retour à la ligne, mais par des points-virgules **;**.

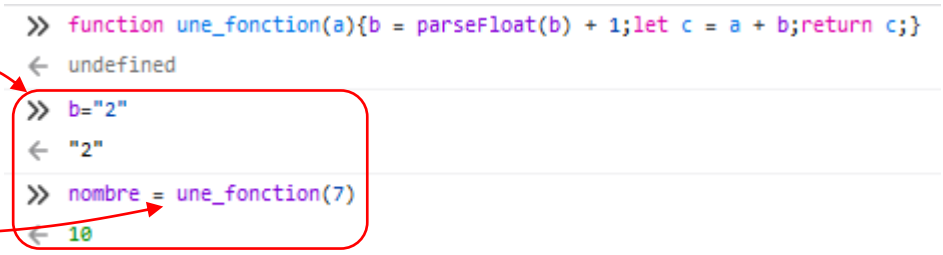
Ainsi, le code de la fonction précédente aurait pu tenir en 1 seule ligne :

```javascript
function une_fonction(a){b = parseFloat(b) + 1;let c = a + b;return c;}
```

⇨ Pour en être certain, faire un copié-collé de cette ligne d'instructions réécrite ci-dessous, dans la console du navigateur :

```
function une_fonction(a){b = parseFloat(b) + 1;let c = a + b;return c;}
```

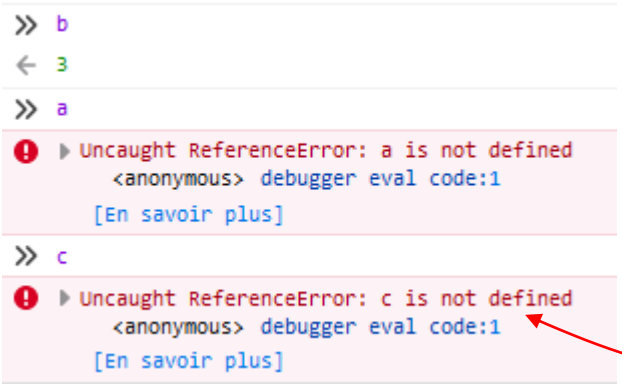
⇨ Définir ensuite la valeur de la variable nommée ***b*** (la chaine de caractère 2) et affecter à la variable ***nombre*** le retour de la fonction nommée *une_fonction()* qui vient d'être définie.



On vient d'exécuter ici, en 3 lignes dans la console, le code ci-contre.

```javascript
function une_fonction(a) {
    b = parseFloat(b) + 1;
    let c = a + b;
    return c;
}
let b = "2";
let nombre = une_fonction(7);
```

Dans celui-ci, on constate que la variable **b** a été définie dans le programme principal et que **b** est ensuite utilisée à l'intérieur de la fonction sans la passer en argument. Son contenu y est ensuite modifié. Avant appel de la fonction on a b = ''2'' , et après appel de la fonction : b = 3.



⇨ Réaliser les tests ci-contre. Vous constatez que **b** a bien pris la valeur 3 et 3 est ici un nombre (pas de guillemets). On constate par contre que les variables **a** et **c** sont, elles, inaccessibles à l'extérieur de la fonction.

En résumé : Les fonctions en python ou JS, c'est presque pareil. Les différences essentielles sont :

- Différences de syntaxe : **python** : def + indentation + retour ligne // **JavaScript** : function + { } + ;
- Différences sur **la PORTEE des variables** :
  - Python :
    - Les variables définies dans le programme principale sont accessibles en lecture uniquement dans les fonctions, pas en écriture.
    - Les variables définies dans une fonction ont une portée locale, elles sont inaccessibles en dehors de cette fonction.
  - JavaScript : Même chose, mais les variables définies dans le programme principal sont accessibles en lecture **et en écriture**.

6. Pour avoir de l'interactivité, il faut mettre en place des évènements souris, clavier, …. est-ce-possible en JS ?

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <input type="text">
        <input type="text">
        <img src="gouv.png">
        <img src="raz.png">
        <p>Prix TTC : </p>
    </body>
    <script>

    </script>
</html>
```

⇨ Recharger votre page *tva.html* qui ne comporte toujours pour l'instant que les éléments html.

⇨ Exécuter une à une les 3 instructions suivantes :

```
>> a = document.querySelector("img")
<- <img src="gouv.png">
>> function fait_cela(){a.style.border="1px solid red";a.setAttribute("src","raz.png");}
<- undefined
>> a.addEventListener("mouseover",fait_cela)
<- undefined
```

a est l'objet JS qui repère la 1ère image

fait_cela() est une fonction définie dans les { }

On créé un évènement lié à l'objet **a** : si on survole l'élément l'image repérée par **a**, la fonction *fait_cela()* s'exécute.

Pour revenir à la page d'origine sans recharger la page, on peut créer un deuxième évènement lié à la fin du survol de l'image par la souris :

```
>> function revient(){a.style.border="none";a.setAttribute("src","gouv.png");}
<- undefined
>> a.addEventListener("mouseout",revient)
<- undefined
```

On a ainsi créé 2 évènements. L'action de rajouter ou d'enlever la bordure rouge aurait pu être gérée par du Css avec la propriété *:hover*. Par contre l'action de modifier le nom du fichier image lié à l'image affichée dans la page ne peut être réalisée qu'en utilisant la méthode *setAttribute()* proposée en JS.

En résumé, ce qu'il faut retenir ici : Pour créer un évènements, on utilise la méthode ***addEventListener()*** :

```
a.addEventListener("mouseover",fait_cela)
```

Objet JavaScript sur lequel est appliqué l'évènement.

Type d'évènement :
''*mouseover*'' : survol de la souris
''*mouseout*'' : arrêt de ce survol
''*click*'' : click gauche
''*dblclick*'' : double click

Nom de la fonction à exécuter, sans parenthèses.
Possible d'utiliser la syntaxe :
a.addEventListener(''mouseover'' , function(){ })
script à exécuter à écrire entre les { }

7. Bon on devrait avoir tout vu là. On ne pourrait pas terminer notre script à présent ?

Oui, oui, allons-y ….. on procède étapes par étapes :

7.1. On affecte des sélecteurs *id* aux différents éléments html de la page afin de pouvoir les sélectionner plus facilement :

⇨ Modifier le code html comme indiqué ci-dessous :

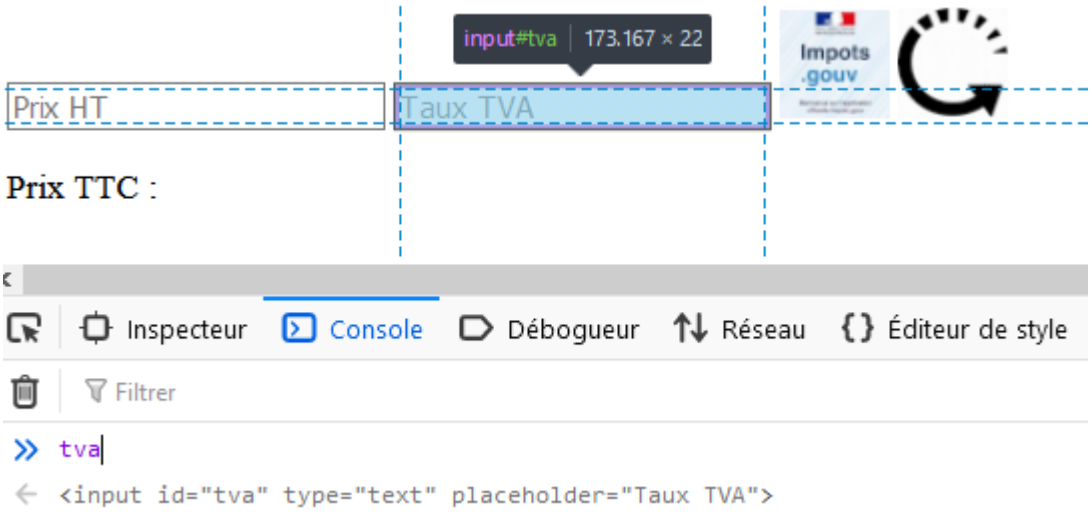
La balise <*span*> est une balise universelle de type *inline*. Elle a été rajoutée ici pour pouvoir définir précisément l'endroit où sera affiché le résultat du calcul du prix TTC.

```
<body>
    <input type="text" placeholder="Prix HT" id="ht">
    <input type="text" placeholder="Taux TVA" id="tva">
    <img src="gouv.png" id="im">
    <img src="raz.png" id="raz">
    <p>Prix TTC : <span id="affichage"></span></p>
</body>
```

⇨ Dans la partie <script>, créer les objets JS qui repèrent les éléments *html* de la page sur lesquels on agira ensuite pour obtenir le fonctionnement souhaité.

```
<script>
    // Main
    // Création des objets repérants les éléments html
    let ht = document.querySelector("#ht");
    let tva = document.querySelector("#tva");
    let affichage = document.querySelector("#affichage");
    let im = document.querySelector("#im");
    let raz = document.querySelector("#raz");
</script>
```

⇨ Ecrire le nom d'un de ces objets dans la console, ….. vous constaterez que l'élément associé sera aussitôt sélectionné par le navigateur sur la page affichée :



7.2. On crée la fonction *reset()* qui réinitialise la page sans avoir à la recharger

⇨ Ecrire le script de la fonction *reset()*

⇨ Appel de la fonction au chargement de la page pour vider les champs …

⇨ Créer un événement lié au *click* sur l'image *raz*.

```
<script>
    // Fonctions
    function reset(){
        ht.value = "";
        tva.value = "";
        affichage.textContent = "";
    }
    // Main
    // Création des objets repérants les éléments html
    let ht = document.querySelector("#ht");
    let tva = document.querySelector("#tva");
    let affichage = document.querySelector("#affichage");
    let im = document.querySelector("#im");
    let raz = document.querySelector("#raz");
    reset();
    // Evènements
    raz.addEventListener('click',reset);
</script>
```

⇨ Tester le bon fonctionnement du script : remplir les champs des 2 éléments input. En cliquant sur l'image , ces 2 champs se réinitialisent.

On continue notre marathon ……

7.3. On créé la fonction *calcul()* qui calcule enfin le prix TTC :

⇨ Rajouter le script ci-contre de la fonction *calcul()* qui permet de calculer et d'afficher le prix TTC au bon emplacement. Ne pas copier bêtement, donner du sens aux différentes instructions.

```
function calcul()  {
    // lecture prix HT dans input
    let prixHt = ht.value;
    if (prixHt == ""){prixHt = 0;}
    prixHt = parseFloat(prixHt);
    // Lecture taux Tva dans input
    let tauxTva = tva.value;
    if (tauxTva == ""){tauxTva = 0;}
    tauxTva = parseFloat(tauxTva);
    // Calcul prix TTC
    let prixTtc = prixHt * tauxTva/100 + prixHt;
    // Affichage du prix dans le span
    affichage.textContent = prixTtc+" €";
}
```

⇨ Rajouter l'évènement qui permet l'exécution de la fonction *calcul()*

```
// Evènements
raz.addEventListener('click',reset)
im.addEventListener('click',calcul)
```

⇨ Tester le bon fonctionnement du script complet.

Voilà, on y est arrivé ……

8. CONCLUSION

On a réalisé pas mal de choses durant cette activité. Si on résume les points essentiels, on pourrait avoir :

- Le **JavaScript** est exécuté localement par le navigateur de mon ordinateur.
- C'est un langage de programmation qui a de nombreux points communs avec ***python.*** Si on compare le code **JS** écrit avec celui écrit en **python** et donné en page 2, on retrouve à peu près les mêmes choses. Ainsi les acquis obtenus en étudiant un langage particulier pourront être transposées à n'importe quel langage. **C'est un point extrêmement encourageant ….. même s'il reste une montagne de choses à apprendre.**

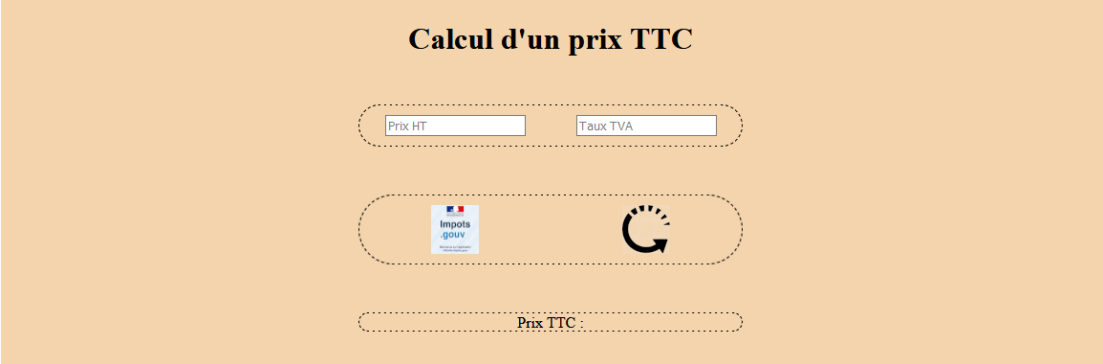
- On peut exécuter du JS dans la console de mon navigateur.
- Pour créer de l'interactivité, on crée un objet JS pour repérer les éléments html de la page.
  On utilise : *document.querySelector(" ")*
- Pour définir une fonction on utilise : ***function + { } + ;***
- Pour définir un évènement qui exécutera une fonction, on utilise :
  objetJS.addEventListener(" ", fonction)

9. On améliore encore un peu le tout :

⇨ On rajoute des effets du type :hover pour améliorer le rendu, mais ici en rajoutant des évènements JS.

```
// Evènements
im.addEventListener('mouseover',function(){im.style.borderRadius="10px";im.style.border="1px solid black"})
im.addEventListener('mouseout',function(){im.style.borderRadius="0px";im.style.border="none";})
im.addEventListener('click',calcul)
raz.addEventListener('mouseover',function(){raz.style.borderRadius="10px";raz.style.border="1px solid black
raz.addEventListener('mouseout',function(){raz.style.borderRadius="0px";raz.style.border="none";})
raz.addEventListener('click',reset)
```

⇨ On rajoute du Css en créant un fichier *tva.css* et en modifiant légèrement *l'html* pour créer des blocs qui permettent d'obtenir la page ci-dessous.

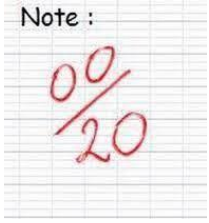



```
<head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="tva.css">
</head>

<body>
    <h1>Calcul d'un prix TTC</h1>
    <div class = "entrees">
        <input type="text" placeholder="Prix HT" id="ht">
        <input type="text" placeholder="Taux TVA" id="tva">
    </div>
    <div class = "images">
        <img src="gouv.png" id="im">
        <img src="raz.png" id="raz">
    </div>
    <p>Prix TTC : <span id="affichage"></span></p>
</body>
```

```
body {
    background-color: rgb(243, 212, 172);
}
h1 {
    text-align: center;
}
.entrees , .images {
    width:400px;
    max-width:100%;
    margin:50px auto;
    display:flex;
    flex-direction: row;
    justify-content:space-around;
    border:1px dashed black;
    border-radius: 50px;

}
input , img {
    margin:10px;
}
p {
    margin:auto;
    width:400px;
    text-align : center;
    border:1px dashed black;
    border-radius: 50px;
}
span {
    color:red;
}
```

10. Rendu de votre travail :

⇨ Transférez votre travail (*fichiers tva.html, tva.css + images gouv.png et raz.png*) dans votre espace web sur *nsibranly.fr*. Créer un lien vers ce travail dans votre fichier *index.html* déjà sur le site, afin que je puisse l'évaluer.



Le code pour le transfert est toujours le même : *web* .