

Javascript

partie 4

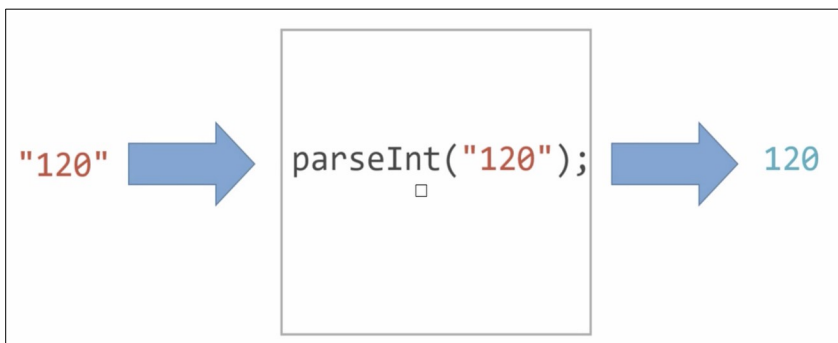
Introduction aux fonctions

1) Un peu de lexique :

Qu'est-ce qu'une fonction?

- Une fonction est un bloc de code qui effectue une tâche ou un calcul (une instruction)
- Une fonction JavaScript est aussi **un objet**
- Les fonctions peuvent prendre des valeurs de variables extérieures à la fonction et renvoyer une nouvelle valeur (plus de détails ci dessous)

Exemple de fonction :



Dans notre exemple, on le fourni une entrée à la **fonction parseInt** (dans ce cas le string « 120 ») et la fonction nous fournit une sortie (le number 120)

Lexique d'une fonction

En ES5

Pour créer une fonction, nous avons besoin de 3 choses: le **mot-clé fonction**, suivi du **nom de la fonction** (en camelCase et sans des caractères spéciaux), des **parenthèses** et l'instruction écrite entre les accolades :


```
function nom_de_la_fonction() {  
    // code  
}
```



Les parenthèses sont impératifs

L'instruction de la fonction ne s'exécutera que si la fonction est appelée :

```
function nom_de_la_fonction() {  
  // code  
}  
nom_de_la_fonction();
```



On appelle la fonction
nom_de_la_fonction
Et seulement à ce moment,
l'instruction à l'intérieur
sera exécutée

PS : Il existe plusieurs façons d'appeler
une fonction, nous verrons plus ci-dessous

En ES6

Après l'introduction d'ES6 en 2015, il a apporté un changement de syntaxe aux fonctions JavaScript. Nous supprimons simplement le mot-clé « fonction » et le remplaçons par (=>)

```
let name = () => {  
  // code  
}
```

Cette syntaxe est connue sous le nom de **fonctions fléchées**.

ES5	ES6
<pre>let name = function() { // code }</pre>	<pre>let name = () => { // code }</pre>

Les paramètres et les arguments

Tout d'abord, un petit schéma pour définir la différence entre un paramètre et un argument :

- **Paramètres** : paramètres que l'on déclare au moment de la déclaration de la fonction
- **Les arguments** : les valeurs que l'on passe au moment de l'exécution de la fonction.

Exemple 1 :

```
function monTacos(epice,viande)
{
  console.log(epice); //
  retourne "oignons"
  console.log(viande); //
  retourne "steack"
}
monTacos("ciboulette","steack")
;
```

```
>> function myTacos(epice,viande){
  console.log(epice); // retourne "oignons"
  console.log(viande); // retourne "steack"
}
myTacos("ciboulette","steack");
ciboulette
steack
← undefined
```

1) on définit mes **paramètres**. **Au moment de l'exécution de la fonction**, les **paramètres** seront remplacés par les **arguments** et soumis aux actions définies par les instructions à l'intérieur de la fonction

Exemple 2 :

```
let tonScore = 0;
function calculerScore(points)
{
  tonScore += points;
  console.log("Ton score " +
tonScore);
}



calculerScore(1);
calculerScore(5);
calculerScore(1);
calculerScore(10);
```

- 1) On crée la fonction « calculerScore » et on le passe le paramètre nommé « **points** »
- 2) Dans les instructions de la fonction, nous additionnons « points » à la variable tonScore (qui est initialisée à 0) et on fait un console.log de cette variable.
- 3) Nous exécutons la fonction en « **remplaçant** » le paramètre points par des **arguments qui comportent des valeurs**

```
>> ▶ let tonScore = 0;
  function calculerScore(points) {
    tonScore += points;
    console.log("Ton score " + tonScore);
  }...
Ton score 1
Ton score 6
Ton score 7
Ton score 17
← undefined
```

Fonctions anonymes

Nous pouvons affecter la fonction à une variable:

ES5	ES6
 <pre>let name = function() { // code } name();</pre>	 <pre>let name = () => { // code } name();</pre>

Dans ce cas, nous disons qu'il s'agit d'une **expression de fonction** : une **fonction anonyme** affectée à une variable

Pour que cette fonction anonyme soit exécutée, nous l'appelons par le **nom de la variable suivi de ()** ; Ces parenthèses sont des parenthèses dites « **appelantes** » car elles servent à exécuter la fonction qui les précède. Le résultat peut être stocké dans une autre variable.

Exemple :

```
let monCarre = (nombre) => {  
  return nombre * nombre  
};  
  
let x = monCarre(4); // on exécute la fonction et on stocke le  
résultat dans la variable x. Ici x reçoit la valeur 16 (4 x 4)  
  
console.log(x); // retourne 16
```

Auto-invoquer une fonction anonyme

La deuxième façon d'exécuter une fonction anonyme va être de créer une fonction anonyme qui va s'auto-invoquer elle-même dès sa création.

Pour créer une fonction auto-invoquée à partir d'une fonction anonyme, on ajoute des **parenthèses autour de la fonction** et un **second parenthèse après la fonction.**

Exemple :

```
//Fonction anonyme auto-invoquée  
(function(){alert('Alerte exécutée par une fonction anonyme')}}  
());
```

Nous avons vu précédemment que le couple de parenthèses suivant le nom de notre variable stockant notre fonction anonyme servait à lancer l'exécution de la fonction.

De la même manière, le couple de parenthèses après la fonction va faire en sorte que la fonction s'appelle elle-même.

La notion d'auto-invocation n'est pas réservée qu'aux fonctions anonymes : on va tout à fait pouvoir auto-invoquer une fonction qui possède un nom. Cependant, en pratique, cela n'aura souvent pas beaucoup d'intérêt (puisque si une fonction possède un nom, on peut tout simplement l'appeler en utilisant ce nom).

```
//Fonction nommée auto-invoquée  
(function bonjour(){alert('Bonjour !')}}());
```

```
//Fonction nommée invoquée par son nom  
function bonjour(){alert('Bonjour !')};  
bonjour();
```



Lorsqu'on auto-invoque une fonction, la fonction s'exécute immédiatement et on n'a donc pas de flexibilité par rapport à cela : **une fonction auto-invoquée s'exécutera toujours juste après sa déclaration.**

Les arguments dans les fonctions :

// Exécution 1 :
Multiples arguments peuvent être passés à une fonction, entre les () et en les séparant par des virgules

```
function calcRect(longueur, largeur) {  
    let result = longueur * largeur;  
    console.log(result);  
}  
  
calcRect(6, 5);
```

Sortie : 30

<p>// Exécution 2</p> <p>Nous avons ajouter un troisième argument pour inclure l'unité de mesure de la superficie : comme « mètre carré ».</p> <pre>function calcRect(longueur, largeur, unite) { var result = longueur * largeur; console.log(result + unite); } calcRect(6, 5, "m²");</pre>	<p>Sortie : 30m²</p>
<p>// Exécution 3</p> <p>Quand on appelle un fonction avec moins d'arguments que paramètres, l'argument manqué retourne un undefined</p> <pre>function calcRect(longueur, largeur, unite) { var result = longueur * largeur; console.log(result + unite); } calcRect(6, 5);</pre>	<p>Sortie : 30 undefined</p>
<p>// Exécution 4</p> <p>Ici on utilise l'opérateur ternaire pour mettre « mètres carrés » comme la valeur par défaut de l'argument "unite".</p> <pre>function calcRect(longueur, largeur, unite) { unite = (unite === undefined) ? "m²" : unite; var result = longueur * largeur; console.log(result + unite); } calcRect(6, 5);</pre>	<p>Sortie : 30m²</p>
<p>// Exécution 5</p> <p>Si nous définissons une unité de mesure, la valeur par défaut de l'argument "unite" n'est plus prise en compte. Les arguments optionnels doivent être placés à la fin de la liste de paramètresde sorte qu'ils peuvent être omis.</p> <pre>function calcRect(longueur, largeur, unite /*optionnel*/) { unite = (unite === undefined) ? "m²" : unite; var result = longueur * largeur; console.log(result + unite); } calcRect(6, 5, "cm²");</pre>	<p>Sortie : 30m²</p>

<pre>// Exécution 6 Les paramètres d'une fonction sont itérables (on peut les parcourir), semblables à un tableau. function calculerSomme(a, b) { var somme = 0; for(var i = 0; i < arguments.length; i++) { somme += arguments[i]; } console.log(somme); } calculerSomme(1, 2, 3);</pre>	<p>Sortie : 6</p>
<pre>// Exécution 7 Vous pouvez utiliser l'objet « arguments », non pas seulement avec les boucles for, mais vous pouvez aussi extraire les paramètres basés sur leur index directement, comme un tableau. function calcRect(longueur, largeur, unite /*optionnel*/) { arguments[2] = (arguments[2] === undefined) ? "m²" : arguments[2]; var result = arguments[0] * arguments[1]; console.log(result + arguments[2]); } calcRect(6, 5, "cm²");</pre>	<p>Sortie : 30m²</p>

L'instruction return :

L'instruction return est utilisé dans le corps d'une fonction pour renvoyer le résultat d'un traitement.

- 1) Si une valeur est fournie, elle sera renvoyée à l'appelant de la fonction.
- 2) Si l'expression définissant la valeur de retour de la fonction est absente, la valeur undefined sera renvoyée.

```
function carre(x) {
  return x * x;
}
demo = carre(3);
console.log(demo) //retourne 9

demo = carre();
console.log(demo) //retourne undefined
```



Il est interdit d'avoir un caractère de fin de ligne entre l'instruction return et l'expression :

Not good	Good
<pre>return; a + b;</pre>	<pre>return a + b;</pre>

Lorsqu'une instruction return est utilisée dans une fonction, l'exécution de la fonction se termine.

Exemple : dans cet exemple nous allons nous contenter de retourner la valeur, au lieu d'utiliser l'instruction break.

```
function obtenirJour(nbr) {  
  switch (nbr) {  
    case 0: return "Lundi";  
    case 1: return "Mardi";  
    case 2: return "Mercredi";  
    case 3: return "Jeudi";  
    case 4: return "Vendredi";  
    case 5: return "Samedi";  
    case 6: return "Dimanche";  
  }  
}
```

```
var jour = obtenirJour(2); // retourne Mercredi
```


L'instruction return : retourner plusieurs valeurs d'une fonctions

L'instruction return retournera uniquement une valeur dans une fonction. Après return, la fonction s'arrêtera. Si une expression est trouvée après le return (exemple, un deuxième return), javascript, nous enverra un warning.

Exemple 1 : ici, nous demandons à la fonction de retourner 2 valeurs : la somme et la différence :

<pre>function add_soust(valeur1, valeur2) { let somme = valeur1 + valeur2; let difference = valeur1 - valeur2; return somme; return difference; } let resultat = add_soust(10, 5); console.log(resultat); // retourne uniquement la somme (15), car après le return, les instructions suivantes ne sont plus lues</pre>	<pre>>> ► function add_soust(valeur1, valeur2) { let somme = valeur1 + valeur2; let difference = valeur1 - valeur2; return somme; return difference;... } 15 ⚠ unreachable code after return statement [En savoir plus] ⚠ unreachable code after return statement [En savoir plus] ← undefined</pre>
--	--



Warning: unreachable code after return statement (Firefox)

Ce warning a deux origines :

- Soit une instruction a été utilisée après le return :

```
function f() {
  let x = 3;
  x += 4;
  return x; // return permet de finir la fonction sur le champ
  x -= 3; // Cette ligne ne sera jamais lue donc exécutée
}
```

- Soit le return a été utilisé sans point virgule et suivi par une expression avec un passage à la ligne.

```
function f() {
  return // Cette instruction est traitée `return;`
  3 + 4; // La fonction termine et cette ligne n'est jamais traitée
}
```

Lorsqu'une instruction existe après le return, ce warning alerte qu'une portion du code ne peut pas être atteinte et ne sera donc jamais lue et exécutée.

Donc, comment retourner plusieurs valeurs dans une fonction ?

L'une des façons pour retourner plusieurs valeurs d'une fonction, et de retourner un tableau de valeurs, et puis accéder à ces valeurs par leur indice.

Exemple 2 :

<pre>function add_soust(valeur1, valeur2) { let somme = valeur1 + valeur2; let difference = valeur1 - valeur2; return [somme, difference]; } let resultat = add_soust(10, 5); console.log("Somme: " + resultat[0]); console.log("Différence: " + resultat[1]);</pre>	<pre>>> ► function add_soust(valeur1, valeur2) { let somme = valeur1 + valeur2; let difference = valeur1 - valeur2; return [somme, difference]; }...</pre> <div>Somme: 15</div> <div>← undefined</div> <div>Différence: 5</div>
---	---

L'autre façon pour retourner plusieurs valeurs d'une fonction est de retourner un objet littéral

Exemple 3 :

<pre>function add_soust(valeur1, valeur2) { let somme = valeur1 + valeur2; let difference = valeur1 - valeur2; return {somme: somme, difference: difference}; } let resultat = add_soust(10, 5); console.log("Somme: " + resultat.somme); console.log("Différence: " + resultat.difference);</pre>	<pre>>> ► function add_soust(valeur1, valeur2) { let somme = valeur1 + valeur2; let difference = valeur1 - valeur2; return {somme: somme, difference: difference}; }...</pre> <div>Somme: 15 debut</div> <div>← undefined</div> <div>Différence: 5 debut</div>
---	--