Les bases de la programmation

Les bases de JavaScript

Le JavaScript est un langage de programmation qui ajoute de l'interactivité à votre site web.

Par exemple l'affichage d'une animation quand on clique sur un bouton ou la validation en temps réel des données entrées dans un formulaire.

Le JavaScript (« JS » en abrégé) est un **langage de programmation dynamique complet** qui, appliqué à un document HTML, peut fournir une interactivité dynamique sur les sites Web.

Il a été inventé par Brendan Eich, co-fondateur du projet Mozilla, de la Mozilla Foundation et de la Mozilla Corporation.

Nous allons explorer les fonctionnalités de base de JavaScript pour que vous puissiez mieux comprendre comment il fonctionne.

Ces fonctionnalités sont communes à la plupart des langages de programmation, si vous comprenez ces éléments en JavaScript, vous êtes en bonne voie de pouvoir programmer à peu près n'importe quoi !

· Les variables

Les **variables** sont des conteneurs dans lesquels on peut stocker des valeurs. Pour commencer, il faut déclarer une variable avec le mot-clé let en le faisant suivre de son nom :

```
let myVariable;
let i;
let prenom;
let Prenom;
```

Un point-virgule en fin de ligne indique là où se termine l'instruction ; ce n'est impérativement requis que si vous devez séparer des instructions sur une même ligne. Toutefois il est de bonne pratique de les mettre à la fin de chaque instruction.

Une variable doit commencer par une lettre, un tiret bas (_) ou un symbole dollar ($). Les caractères qui suivent peuvent être des chiffres ou des lettres majuscules/minuscules .

Attention ! Javascript est sensible aux majuscules et minuscules. Une variable qui s'appellerait par exemple **myVariable** n'est pas identique à une autre qui s'appellerait **myvariable**.

· Les différents types de variables

Les **variables** sont essentielles à la programmation. Si les valeurs ne pouvaient pas changer, on ne pourrait rien faire de dynamique, comme par exemple personnaliser un message de bienvenue ou calculer le total des produits mis dans le caddie par un visiteur du site Web.

Type de variable	Description	Exemples
Chaîne de caractères	Une suite de caractères connue sous le nom de chaîne. Pour indiquer que la valeur est une chaîne, il faut la placer entre guillemets.	`let myVariable = 'Bob';` `let myVariable = 'Hello World!';`
Nombre	Un nombre. Les nombres ne sont pas entre guillemets.	`let myVariable = 10;` `let myVariable = -5;`
Booléen	Une valeur qui signifie vrai ou faux. **true/false** sont des mots-clés spéciaux en JS, ils n'ont pas besoin de guillemets.	`let myVariable = true;` `let myVariable = false;`

Tableau	Une structure qui permet de stocker plusieurs valeurs dans une seule variable.	`let myVariable = ['Bob','Bill'];`
		`myVariable[0] équivaut à Bob.`
		`myVariable[1] équivaut à Bill.`

Objet	Tout est un objet en JavaScript et peut être stocké dans une variable.	`myVariable` = document.getElementById(".conteneur"));;

· Les opérateurs

Un **opérateur** est un symbole mathématique qui produit un résultat en fonction de deux valeurs (ou variables). Le tableau suivant liste certains des opérateurs les plus simples.

Opérateur	Description	Exemples
Additionner **+**	Utilisé pour ajouter deux nombres ou concaténer (accoler) deux chaînes	`let myVariable = 6+9;` `let myVariable = 'Hello' + 'World';`

Soustraction, Multiplication, Division
- ***** **/**
	Les opérations mathématiques de base.	`let myVariable = 6-4;`
		`let myVariable = 2*2;`
		`let myVariable = 6/3;`

Assignation
=
	Affecte une valeur à une variable	`let myVariable = true;`
		`let myVariable = 6-4;`
		`let myVariable = 'Bill',`

Égalité
== **===**
	Teste si deux valeurs sont égales et renvoie un booléen `true`/`false` comme résultat.	`let a = 2;`
		`let b = 2;`
		`if (a ===  b) { alert ('Égaux'); }`

Négation , N'égale pas
! **!==**
	Renvoie la valeur logique opposée à ce qu'il précède ; il change true en false, etc. Utilisé avec l'opérateur d'égalité, l'opérateur de négation teste que deux valeurs ne sont pas égales.	`let a = 2;`
		`let b = 3;`
		`if (a !== b) { alert ('Pas égaux'); }`
		`if ( !(a === b)) { alert ('Pas égaux'); }`

Exercices

Ouvrez le CodePen suivant: https://codepen.io/xsccsx/pen/ExbYaXj, faites un fork puis essayez de résoudre les problèmes suivants.

a) Addition

- Déclarez les variables **a**, **b** et **c**
- Assignez le valeur **5427** à la variable **a**
- Assignez le valeur **4572** à la variable **b**
- Assignez la somme de **a** et **b** à la variable **c** et affichez sa valeur le #conteneur

b) Concatenation

- Déclarez les variables **a**, **b** et **c**
- Assignez le valeur **Bon** à la variable **a**
- Assignez le valeur **Johnson** à la variable **b**
- Créez une variable **c** pour afficher **Bob Johnson** dans le #conteneur

c) Pointure et anniversaire

- Assignez votre taille de chaussures à la variable **a**
- Assignez la multiplication de **a * 5** à la variable **b**
- Assignez la somme de **b + 50** à **d**
- Assignez la multiplication de **d * 20** à la variable **e**
- Assignez le résultat de **e + 2022 - 1000** à la variable **f**
- Soustrayez votre **année de naissance** à **f** et assignez le résultat à la variable **g**
- Affichez la valeur de **g** dans le #conteneur (Vous devriez voir votre âge et taille de chaussures)

Solution: https://codepen.io/xsccsx/pen/podxWMO

d) Aléatoire

- Rajoutez la fonction suivante :

```
/* Retourne un nombre entier aléatoire < max */
function get_random_int(max) {
  return Math.floor(Math.random() * max);
}
```

- Déclarez les variables **a**, **b** et **c**
- Assignez un nombre aléatoire < 10 à la variable **a**
- Assignez un nombre aléatoire < 100 à la variable **b**
- Assignez le résultat de la multiplication de **a** par **b** à la variable **c** et affichez sa valeur le #conteneur

d) Toujours 1

- Changez l'exercice précédent pour que le résultat soit toujours **1**

· Les structures conditionnelles

Les structures conditionnelles sont des éléments du code qui permettent de **tester si une expression est vraie ou non** et d'exécuter des instructions différentes selon le résultat. La structure conditionnelle utilisée la plus fréquemment est *if ... else.*

```
let ice_cream = 'chocolat';

/* Attention ! Pour comparer deux valeurs on utilise == ou === */
if (ice_cream == 'chocolat') {
    alert("J'adore la glace au chocolat! ");
} else {
    alert("Je n'aime pas la glace " + ice_cream + ". Je préfère la glace au chocolat! ");
}
```

L'expression contenue dans `if ( ... )` correspond au test — Ici, on utilise l'opérateur d'égalité (décrit plus haut) pour comparer la variable `ice_cream` avec la chaîne de caractères `chocolat` pour voir si elles sont égales.

Si cette comparaison renvoie true, le premier bloc de code sera exécuté. Sinon, le premier bloc est sauté et c'est le code du second bloc qui est exécuté.

· Les fonctions

Les fonctions sont un moyen de compacter des fonctionnalités en vue de leur réutilisation. Quand vous avez besoin de la procédure, vous pouvez appeler une fonction au lieu de ré-écrire la totalité du code chaque fois.

Nous avons déjà utilisé des fonctions plus haut, par exemple :

```
// Retourne un nombre entier aléatoire < max.
function get_random_int(max) {
  return Math.floor(Math.random() * max);
}
```

Si la fonction contient un *return*, la valeur est retournée par la fonction et peut être assigné à une variable. Une fonction ne doit pas nécessairement retourner une valeur.

```
// Assigne la valeur retournée par la fonction à la variable a.
let a = get_random_int (100);

// Affiche une notification avec la valeur de a.
alert (a);
```

https://codepen.io/xsccsx/pen/GROYZGj

Les noms des fonctions suivent les mêmes règles que ceux des variables. Vous pouvez donc donner le nom que vous voulez à votre fonction du moment que celui-ci commence par une lettre, ne contient pas d'espace ni de caractères spéciaux et n'est pas déjà pris nativement par le JavaScript.

Une fonction peut renvoyer une valeur en utilisant un *return* à l'intérieur du corps de la fonction.

```
// Fonction qui additionne deux nombres.
function additionner (a, b)
{
    // Retourne la somme de a+b.
    return a + b;
}

// Assigne la valeur retournée par la fonction additionner à la variable x.
let x = additionner (10, 5);

// Affiche 15.
alert (x);
```

Une fonction ne doit pas nécessairement renvoyer une valeur.

```
// Fonction qui affiche un message.
function display (message)
{
    alert(message);
}

// Fait appel à la fonction display pour afficher un message.
display ('Hello World');
```

Une fonction peut recevoir plusieurs paramètres.

```
// Fonction qui fait le total de 5 nombres donnés.
function somme (a,b,c,d,e,f)
{
    return a+b+c+d+e+f;
}

// Assigne le total des valeurs à la variable total.
let total = somme (1,22,333,4444,55555,66666);
```

Une fonction ne doit pas nécessairement recevoir des paramètres.

```
// Retourne la valeur de PI.
function get_pi ()
{
   return Math.PI;
}

// Affiche PI.
alert (get_pi());
```

On peut aussi imbriquer les fonctions les unes dans les autres.

```
// Retourne la valeur de PI.
function get_pi ()
{
   return Math.PI;
}

// Calcule la circonférence d'un cercle à partir du diamètre.
function calculer_circonference_avec_diametre (diametre)
{
    // Circonférence= π × diamètre
    return get_pi() * diametre;
}

// Calcule la circonférence d'un cercle à partir du rayon.
function calculer_circonference_avec_rayon (rayon)
{
    // Diamètre = 2 x rayon.
    return calculer_circonference_avec_diametre (2 * rayon);
}
```

Exercices

Ouvrez le CodePen suivant: https://codepen.io/xsccsx/pen/RwjeaYR, faites un fork puis essayez de résoudre les problèmes suivants.

a) Addition

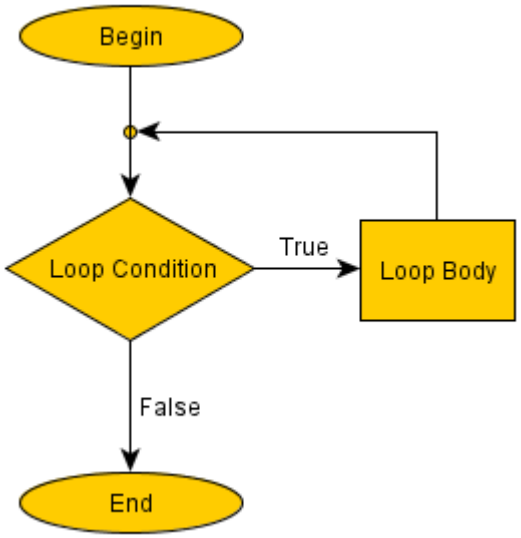
- Déclarez une fonction **additionner** qui accepte deux paramètres **x** et ***y***
- Déclarer une variable **z** à l'intérieur de cette fonction
- Assignez la somme de **x + y** à **z**
- Retournez la valeur de **z** à l'aide d'un **return** et assignez le résultat à la variable **c**
- Afficher la valeur de **c** dans le #conteneur

b) Comparaison

- Déclarez les variables **x** et **y** et assignez un nombre entier a chacune de ces variables
- Créez un programme qui utilise des conditions pour comparer les deux variables
- Selon la valeur de **x** et **y**, le programme doit afficher un des trois cas possibles sous la forme suivante :
    - 5 > 2 (si x = 5 et y = 2)
    - 10 < 100 (si x = 10 et 100 = 100)
    - 50 = 50 (si x = 50 et y = 50)

· Les boucles

Les langages de programmation sont très utiles pour effectuer des tâches répétitives, allant de calculs basiques à à peu près n'importe quelle autre situation où vous avez un certain nombre d'actions similaires à répéter.



Les boucles de programmation ne font que faire la même action encore et toujours – ce qui se traduit par **itérer** en langage de programmeur.

A chaque itération de la boucle (loop), on vérifie si la condition est vraie (true). Si tel est le cas, le code compris dans la boucle (loop body) est executé à nouveau, et ceci jusqu'à ce que la condition soit fausse (false).

Exemple :

```
// Affiche les valeurs entre 0 et 5 dans la console.
for (let i = 0; i <= 5; i++) {
  console.log(i);
}
```

- La boucle standard

```
for (initialisation; condition de sortie; expression finale) {
  // Code à exécuter
}
```

La boucle standard commence par le mot-clé for, suivi par des parenthèses.

A l'intérieur des parenthèses, on a trois parties :

L'initialisation — il s'agit souvent d'une variable initialisée à une certaine valeur, qui est incrémentée afin de compter le nombre de fois où la boucle s'est exécutée. On peut également la nommer **compteur**.

La condition de sortie — cette condition définit le moment où la boucle doit arrêter de s'exécuter. C'est généralement une expression contenant un opérateur de comparaison, c'est-à-dire un test pour voir si la condition de sortie est atteinte.

L'expression finale — cette expression est exécutée à chaque fois que la boucle a effectué une itération complète. Cela sert souvent à incrémenter (ou dans certains cas décrémenter) le compteur, pour le rapprocher de la valeur de la condition de sortie.

Si nous reprenons l'exemple précédent :

```
// Affiche les valeurs entre 0 et 5 dans la console.
for (let i = 0; i <= 5; i++) {
  console.log(i);
}
```

`let i = 0;`

On initialise la variable i avec la valeur 0.

`i <= 5;`

Notre condition définit que le code contenu entre les accolades sera executé tant que i <= 5. Dès lors que la variable i aura atteint la valeur 6, la condition n'est plus remplie et l'exécution de la boucle s'arrêtera.

`i++`

Pour éviter que la boucle s'exécuté indéfiniment, la valeur de i doit à un moment donné être plus grand que 5 pour que la condition i <= 5 ne soit plus remplie (false). L'expression finale sera à incrémenter i à chaque exécution de la boucle.

Quitter une boucle avec break

Si vous voulez quitter une boucle avant que toutes les itérations aient été terminées, vous pouvez utiliser l'instruction **break**. Un **break** quittera immédiatement la boucle et fera passer le navigateur sur n'importe quel code qui suit la boucle.

Exemple:

```
for (let i = 0; i <= 5; i++) {
  if (i == 3)
  {
      break;
  }
  console.log(i);
}
```

Résultat :

```
0
1
2
```

Pourquoi est-ce-que les chiffres 3,5 et 5 ne sont-ils pas affichés ?

Passer des itérations avec continue

L'instruction **continue** fonctionne d'une manière similaire à break, mais au lieu de sortir complètement de la boucle, elle passe à l'itération suivante de la boucle.

Exemple:

```
for (let i = 0; i <= 5; i++) {
  if (i == 3)
  {
      continue;
  }
  console.log(i);
}
```

Résultat :

```
0
1
2
4
5
```

Pourquoi est-ce-que le chiffre 3 n'est-il pas affiché ?