

Javascript – Structure et Flux

1WEBD – Javascript Web Development



Sommaire

1. Fonctions.
2. Contrôle de Flux.
3. Objets et Tableaux.



0. Rappel

En Javascript, tout est un objet

1. Fonctions


1. Fonctions

Déclaration

- **function** déclare une fonction suivie de son nom et d'une éventuelle liste de paramètres (les parenthèses sont obligatoires même si elles ne contiennent rien)
- **return** désigne ce que la fonction renvoie (si elle renvoie quelque chose)
- le corps de la fonction est délimité par des accolades {}

1. Fonctions

Déclaration



```
function nomDeLaFonction(parametre1, parametre2) {  
    // Corps de la fonction  
    return parametre1 + parametre2;  
}
```


1. Fonctions

Expression

- on peut assigner une fonction à une variable

1. Fonctions


Expression



```
const maFonction = function(parametre1, parametre2) {  
    return parametre1 * parametre2;  
};
```


1. Fonctions

Appel



```
nomDeLaFonction(5, 3); // Appel de la fonction déclarée  
maFonction(5, 3);      // Appel de l'expression de fonction
```

1. Fonctions

Fonctions Flechées

- Introduit dans ES6
- Syntaxe plus concise
- **return** implicite si la fonction est sur une seule ligne
- Sinon, le corps de la fonction est défini par des accolades {}

1. Fonctions

Fonctions Flechées



```
const maFonctionFlechee = (parametre1, parametre2) => parametre1 + parametre2;
```

1. Fonctions

Infos Supplémentaires

- Une fonction peut renvoyer une fonction
- Une fonction peut recevoir en paramètre une autre fonction (ex: callback)

1. Fonctions

Infos Supplémentaires

```
function creerAddition(a, b) {  
  return function() {  
    return a + b  
  }  
}
```

```
function faireQuelqueChose(callback) {  
  callback();  
}
```

1. Fonctions



2. Contrôle de Flux

2. Contrôle de Flux

Conditions - Opérateurs

- `==` : Égalité (vérifie la valeur sans tenir compte du type)
- `===` : Égalité stricte (vérifie la valeur et le type)
- `!=` : Inégalité
- `!==` : Inégalité stricte
- `>` : Plus grand que
- `<` : Moins que
- `>=` : Plus grand ou égal
- `<=` : Moins ou égal

2. Contrôle de Flux

Conditions – Opérateurs Logiques

- **&&**: renvoie vrai si les deux conditions sont vraies
- **||**: renvoie vrai si une des deux conditions est vraie
- **!**: inverse la valeur de la condition (**!true** devient **false**)

2. Contrôle de Flux

Conditions – Opérateurs Logiques

```
let age = 25;
let permisDeConduire = true;

// Utilisation de l'opérateur ET (&&)
if (age > 18 && permisDeConduire) {
  console.log("Peut conduire une voiture");
} else {
  console.log("Ne peut pas conduire une voiture");
}

// Utilisation de l'opérateur OU (||)
if (age < 16 || !permisDeConduire) {
  console.log("Ne peut pas conduire légalement");
} else {
  console.log("Peut-être autorisé à conduire");
}

// Utilisation de l'opérateur NON (!)
if (!permisDeConduire) {
  console.log("Doit obtenir un permis de conduire");
} else {
  console.log("A déjà un permis de conduire");
}
```

2. Contrôle de Flux

Conditions

- **if**: Exécute un bloc de code si la condition spécifiée est vraie
- **else if**: Facultatif, pour tester une autre condition si la première est fausse
- **else**: Facultatif, exécute un bloc de code si toutes les conditions précédentes sont fausses

2. Contrôle de Flux

Conditions

```
if (condition) {  
    // Code à exécuter si 'condition' est vraie  
} else if (autreCondition) {  
    // Code à exécuter si 'autreCondition' est vraie  
} else {  
    // Code à exécuter si aucune des conditions précédentes n'est vraie  
}
```

2. Contrôle de Flux

Conditions - Switch

- Permet de comparer une expression à plusieurs valeurs
- Evite les forêts de if

2. Contrôle de Flux

Conditions - Switch

```
switch(expression) {  
    case valeur1:  
        // Code à exécuter si expression === valeur1  
        break;  
    case valeur2:  
        // Code à exécuter si expression === valeur2  
        break;  
    default:  
        // Code à exécuter si aucune correspondance n'est trouvée  
}  

```

2. Contrôle de Flux

Boucles – While

- Répète un bloc de code tant que la condition spécifiée est vraie

2. Contrôle de Flux

Boucles – While

```
while (condition) {  
    // Code à exécuter tant que la condition est vraie  
}
```


2. Contrôle de Flux

Boucles – Do...While

- Exécute un bloc de code une fois, puis répète l'exécution tant que la condition est vraie

2. Contrôle de Flux

Boucles – Do...While



```
do {  
    // Code à exécuter  
} while (condition);
```

2. Contrôle de Flux

Boucles - For

- Utilisé pour répéter un bloc de code un nombre défini de fois

2. Contrôle de Flux

Boucles - For

```
// Création d'un tableau de nombres
const nombres = [10, 20, 30, 40, 50];

// Initialisation d'une variable pour stocker la somme
let somme = 0;

// Utilisation d'une boucle for pour parcourir le tableau
for (let i = 0; i < nombres.length; i++) {
    somme += nombres[i]; // Ajoute l'élément courant à la somme
}

// Affichage de la somme totale
console.log("La somme des éléments du tableau est :", somme);
```

2. Contrôle de Flux

Boucles - Autres

- on peut aussi utiliser les methodes `.forEach` disponible sur tous les tableaux

2. Contrôle de Flux

Boucles - Autres

```
● ● ●  
  
// Création d'un tableau de nombres  
const nombres = [1, 2, 3, 4, 5];  
  
// Utilisation de forEach pour parcourir le tableau  
nombres.forEach(function(element, index) {  
    console.log("Element à l'indice " + index + " : " + element);  
});
```

2. Contrôle de Flux



3. Objets et Tableaux

3. Objets et Tableaux

Objets

- Collection de paires clé-valeurs
- Utilisé pour stocker des données structurées
- Les valeurs peuvent être tout type d'objet en Javascript (rappel: tout est un objet)
- Pour accéder à une valeur, on utilise la clef associée
 - exemple.nom pour récupérer la valeur de la clef nom dans l'objet exemple

3. Objets et Tableaux

Objets

```
// Création d'un objet représentant une personne
const personne = {
  nom: "Dupont",
  prenom: "Jean",
  age: 30,
  metier: "Développeur Web",

  // Méthode de l'objet
  sePresenter: function() {
    console.log("Bonjour, je m'appelle " + this.prenom + " " + this.nom + " et je suis " + this.metier + ".");
  },

  // Méthode pour célébrer l'anniversaire
  feterAnniversaire: function() {
    this.age += 1;
    console.log("Joyeux anniversaire! Age maintenant : " + this.age);
  }
};

// Accès aux propriétés de l'objet
console.log(personne.nom); // Affiche "Dupont"
console.log(personne.age); // Affiche 30

// Appel des méthodes de l'objet
personne.sePresenter(); // Affiche "Bonjour, je m'appelle Jean Dupont et je suis Développeur Web."
personne.feterAnniversaire(); // Augmente l'âge de 1 et affiche le message d'anniversaire
```

3. Objets et Tableaux

Objets

- Une valeur dans un objet est appelée une propriété
- Une fonction dans un objet est appelée une méthode
- L'objet personne contient 4 propriétés et 2 méthodes

3. Objets et Tableaux

Objets - **this**

- fait référence au contexte d'exécution de la fonction
- dans un objet, **this** est l'objet
- dans un constructeur d'une classe, il fait référence à la nouvelle instance de cette classe (on en parlera plus tard)
- une fonction fléchée ne fait pas référence au contexte de la fonction, mais au contexte englobant (ici, la page du navigateur)

3. Objets et Tableaux

Tableaux

- collections ordonnées de valeurs, pouvant contenir différents types de données
- `.push` ajoute un element à la fin du tableau
- `.length` donne la longueur d'un tableau
- Contient aussi quelques fonctions pour chercher un element, filtrer etc.

3. Objets et Tableaux

Tableaux – Helper methods

- `.find`: execute une fonction pour chaque element et renvoie le premier pour qui la fonction renvoie **true**
- `.filter`: renvoie tous les éléments pour qui la fonction en parametre renvoie **true**
- `.map`: renvoie un nouveau tableau avec chaque element transformé par la fonction en parametre
- etc etc

3. Objets et Tableaux

Tableaux – Helper methods

```
// Création d'un tableau de nombres
const numeros = [1, 2, 3, 4, 5];

// Ajout d'éléments à un tableau
numeros.push(6); // Ajoute 6 à la fin du tableau
console.log("Après ajout :", numeros); // Affiche [1, 2, 3, 4, 5, 6]

// Accès à un élément du tableau
const troisiemeElement = numeros[2]; // Les indices des tableaux commencent à 0
console.log("Troisième élément :", troisiemeElement); // Affiche 3

// Modification d'un élément
numeros[0] = 10; // Modifie le premier élément
console.log("Après modification :", numeros); // Affiche [10, 2, 3, 4, 5, 6]

// Parcourir un tableau avec une boucle for
for (let i = 0; i < numeros.length; i++) {
  console.log(`Element à l'indice ${i} :`, numeros[i]);
}

// Utilisation de forEach pour parcourir un tableau
numeros.forEach((numero, index) => {
  console.log(`Element à l'indice ${index} :`, numero);
});

// Trouver un élément dans un tableau
const trouveCinq = numeros.find((numero) => numero === 5);
console.log("Trouvé le numéro 5 :", trouveCinq); // Affiche 5

// Filtrer un tableau
const numerosPairs = numeros.filter((numero) => numero % 2 === 0);
console.log("Nombres pairs :", numerosPairs); // Affiche les nombres pairs

// Transformation des éléments d'un tableau
const double = numeros.map((numero) => numero * 2);
console.log("Doubles des numéros :", double); // Affiche le double de chaque élément

// Réduction d'un tableau à une seule valeur
const somme = numeros.reduce((total, numero) => total + numero, 0);
console.log("Somme des numéros :", somme); // Affiche la somme de tous les éléments
```

3. Objets et Tableaux



