Les différentes fonctions en JavaScript

Fonction anonyme en JavaScript

Une fonction anonyme est une fonction sans nom, souvent utilisée comme argument pour d'autres fonctions ou assignée à une variable.

Exemple :

```
javascript
CopierModifier
// Fonction anonyme assignée à une variable
const saluer = function() {
  console.log('Bonjour!');
};

saluer(); // Affiche : Bonjour!
```

Fonction lambda en JavaScript

Les fonctions lambda, ou fonctions fléchées, sont une syntaxe concise pour définir des fonctions anonymes.

Exemple :

```
javascript
CopierModifier
// Fonction fléchée
const saluer = () => {
  console.log('Bonjour!');
};

saluer(); // Affiche : Bonjour!
```

Type générique

JavaScript est un langage faiblement typé et ne supporte pas directement les types génériques. Cependant, TypeScript, un sur-ensemble de JavaScript, offre cette fonctionnalité.

Exemple en TypeScript :

```
typescript
CopierModifier
function identite<T>(arg: T): T {
  return arg;
}
```

```
let sortie = identite<string>('Bonjour');
console.log(sortie); // Affiche : Bonjour
```

Closure en JavaScript

Une closure est une fonction qui a accès à son propre scope, au scope externe et au scope global, même après que la fonction externe ait terminé son exécution.

Exemple :

```
javascript
CopierModifier
function externe() {
  let compteur = 0;
  return function() {
    compteur++;
    console.log(compteur);
  };
}

const incrementer = externe();
incrementer(); // Affiche : 1
incrementer(); // Affiche : 2
```

Prototype et héritage en JavaScript

En JavaScript, chaque objet a une propriété interne appelée [[Prototype]] (accessible via Object.getPrototypeOf(obj)), permettant l'héritage.

Exemple :

```
javascript
CopierModifier
function Personne(nom) {
  this.nom = nom;
}

Personne.prototype.saluer = function() {
  console.log(`Bonjour, je suis ${this.nom}`);
};

const personne1 = new Personne('Alice');
personne1.saluer(); // Affiche : Bonjour, je suis Alice
```

Event Loop

L'Event Loop est le mécanisme qui permet à JavaScript d'exécuter des opérations non-bloquantes, en traitant les tâches en attente dans une file d'attente.

Exemple :

```
javascript
CopierModifier
console.log('Début');

setTimeout(function() {
  console.log('Timeout');
}, 0);

console.log('Fin');

// Affiche :
// Début
// Fin
// Timeout
```

== et ===

== compare les valeurs après conversion de type, tandis que === compare les valeurs et les types sans conversion.

Exemple :

```
javascript
CopierModifier
console.log(5 == '5');  // true
console.log(5 === '5'); // false
```

La portée des variables et des fonctions

La portée détermine où une variable ou une fonction est accessible.

Exemple :

```
javascript
CopierModifier
function test() {
  let x = 10;
  if (true) {
    let y = 20;
    console.log(x); // 10
    console.log(y); // 20
  }
  console.log(x); // 10
  console.log(y); // Erreur : y n'est pas défini
}

test();
```

await et async

async définit une fonction asynchrone, et await est utilisé à l'intérieur de ces fonctions pour attendre la résolution d'une promesse.

Exemple :

```
javascript
CopierModifier
async function obtenirDonnees() {
  let reponse = await fetch('https://api.exemple.com/donnees');
  let donnees = await reponse.json();
  console.log(donnees);
}

obtenirDonnees();
```

Promise de tableau

Promise.all() permet d'exécuter plusieurs promesses en parallèle et d'attendre leur résolution.

Exemple :

```
javascript
CopierModifier
let promesse1 = new Promise((resolve) => setTimeout(resolve, 1000,
'Premier'));
let promesse2 = new Promise((resolve) => setTimeout(resolve, 2000,
'Deuxième'));
```

```
Promise.all([promesse1, promesse2]).then((resultats) => {
  console.log(resultats); // Affiche : ['Premier', 'Deuxième']
});
```

Var et let

`var` a une portée fonctionnelle, tandis que `let` a une portée de bloc.

Exemple :

```
javascript
CopierModifier
function testVar() {
  if (true) {
    var x = 10;
  }
  console.log(x); // 10
}

function testLet() {
  if (true) {
    let y = 20;
  }
  console.log(y); // Erreur : y n'est pas défini
}

testVar();
testLet();
```

null et undefined

null est une valeur assignée représentant l'absence intentionnelle de valeur, tandis que undefined indique qu'une variable a été déclarée mais n'a pas encore été assignée.

Exemple :

```
javascript
CopierModifier
let a;
console.log(a); // undefined

let b = null;
console.log(b); // null
```