Javascript

Contrôle du flux d'instructions

Bloc

L'instruction la plus simple est l'instruction de bloc qui permet de regrouper des instructions. Un bloc est délimité par une paire d'accolades.

Les instructions de blocs sont souvent utilisées avec les instructions conditionnelles et itératives telles que `if`, `for`, `while`.

```
while (x < 10) {
  x++;
}
```

Instructions conditionnelles

Instruction if...else

```
if (condition) {
  //code exécuté si condition est évalué à true
}
```

```
if (condition) {
  //instruction1;
} else {
  //instruction2;
}
```

```
if (condition1) {
  //instruction1;
} else if (condition2){
  //instruction2;
} else {
  //instruction3;
}
```

Si la condition n'est pas un booléen, la conversion est automatique.

Opérateur conditionnel ternaire

Syntaxe

```
condition ? expr1 : expr2
```

Exemples

```
let autorisation = true;

console.log( autorisation ? "autorisé" : "interdit" ); //autorisé
```

```
let toto = { nom : "Toto", age : 25};

console.log( toto.age > 18 ? "majeur" : "mineur" ); //majeur
```

Valeurs équivalentes à false

Lors d'une conversion en booléen, les valeurs suivantes renvoient false (falsy values) :

- `false`
- `undefined`
- `null`
- `0`
- `NaN`
- `""`

Les autres valeurs, dont les objets, renvoient true.

```
Boolean(5); //true
Boolean(-1); //true
Boolean(0); //false
Boolean({}); //true
Boolean([]); //true
Boolean(5/0); // Infinity => true
Boolean( Number("toto") ); // NaN => false
Boolean("true"); //true
Boolean("false"); //true
```

Instruction switch

Syntaxe

```
switch (expression) {
  case label_1: instructions_1 [break;]
  case label_2: instructions_2 [break;]
  ...
  default: instructions_par_defaut [break;]
}
```

Exemple

```
function prixDesFruits(fruits) {

  let prix;

  switch (fruits) {
    case "Oranges" : prix = 0.59; break;
    case "Cerises" : prix = 3; break;
    case "Mangues" : case "Papayes" : prix = 2.79; break;
    default: console.log("Désolé, nous n'avons pas de " + fruits + ".");
  }

  return prix;
}
```

Instruction switch

C'est l'égalité stricte qui est testée.

Attention à ne pas oublier le `break` sans quoi toutes les instructions qui suivent seront également exécutées (c'est le mécanisme qui permet de regrouper les cas).

Boucles et itération

Instruction for

Syntaxe

```
for ([expressionInitiale]; [condition]; [expressionIncrément])
  instruction
```

Exemple

```
for (let pas=0 ; pas<5 ; pas++) {
  // Ceci sera exécuté 5 fois
  // la variable "pas" ira de 0 à 4
  console.log("Faire un pas vers l'est");
}
```

Instruction while

Syntaxe

```
while (condition) instruction;
```

ou

```
do instruction while (condition);
```

Dans le deuxième cas, l'instruction sera exécutée au moins une fois, même si la condition est fausse.

Exemple

```
let n = 0;
let x = 0;

while (n < 3) {
  n++;
  x+= n;
}
```

Instruction for...in

Itération sur les propriétés énumérables d'un objet.

Syntaxe

```
for (variable in objet) instruction
```

Exemple

```
function afficherProps(obj) {
  let result = "";
  for (let n in obj) result += `${n} = ${obj[n]} \n`;
  return result;
}

afficherProps({nom:"Toto",age:25});
/*
nom=Toto
age=25
*/
```

Instruction for...in

- parcourt les propriétés d'un objet dans un ordre arbitraire
- ne doit pas être utilisé pour parcourir un tableau
- ne parcourt que les propriétés énumérables

Instruction for...of

Boucle sur les valeurs des itérables (tableaux, chaînes, listes, etc)

Syntaxe

```
for (variable of iterable) instruction
```

Exemple

```
function enumere(obj) {
  let result = ""
  for (let value of obj) result += value + ";";
  return result;
}

enumere([1, 2, 3]);
// 1;2;3;

enumere("toto");
// t;o;t;o;
```

Instruction for...of

- ne fonctionne que sur les objets itérables
- parcourt les propriétés de manière ordonnée
- affecte la valeur à la variable et non la clé

break

Provoque la fin de l'instruction `while, do-while, for,` ou `switch` dans laquelle il est inscrit (la plus imbriquée).

```
let tab = ["toto","titi","tata"];

for (let i=0; i<tab.length; i++) {
  if (a[i] == "tata") break;
  console.log(a[i]);
}
```

continue

Termine l'itération courante de la boucle (la plus imbriquée) et passe à l'exécution de la prochaine itération.

```
let i = 0;
let n = 0;

while (i < 5) {
  i++;
  if (i == 3) continue;
  n += i;
}

n; //1 + 2 + 4 + 5
```

Gestion des exceptions

Gestion des exceptions

Problème

```
function dernierElement(tableau) {

  if (tableau.length === 0) {
    console.log("Le tableau est vide");
    return false;
  }

  return tableau.at(-1);
}
```

Ce code n'est pas très bon pour plusieurs raisons

- comment faire si je veux informer l'utilisateur que le tableau est vide, autrement que par la console ?
- Que se passe-t-il si le dernier élément du tableau contient la valeur `false` ?

Gestion des exceptions

Solution

La fonction jette (ou lève) une exception quand elle rencontre un cas anormal.

```
function dernierElement(tableau) {

  if (tableau.length === 0) throw new Error("Le tableau est vide");

  return tableau.at(-1);
}
```

C'est à l'appel de la fonction que l'on décide comment gérer l'erreur.

```
try { dernierElement([]); }
catch(e) { console.log(e); }

//ou

try { dernierElement([]); }
catch(e) { window.alert(e); }
```

Gestion des exceptions

Exemple complet

```
function verifNom(nom) {
  if (typeof nom !== "string") throw new Error(typeof nom + ": type incorrect");
  if ( !/^[a-z]+$/i.test(nom) ) throw new Error(nom + ": nom incorrect");
}

function verifAge(age) {
  if (typeof age !== "number") throw new Error(typeof age + ": type incorrect");
  if (age < 0 || age > 120) throw new Error(age + ": âge incorrect");
}

function verifChamps(obj) {
  if (typeof obj !== "object") throw new Error("verifChamps attend un objet");
  verifNom(obj.nom);
  verifAge(obj.age);
}

try {
    verifChamps({ nom : "Toto", age : 25 });
    verifChamps({ nom : "Toto" });
    verifChamps({ nom : 25, age : "Toto" });
}
catch(e) { window.alert(e); }
```

Asynchronisme

Exemple

```
let i=0;

window.setTimeout(() => i++, 1000);

console.log(i); //?
```

Asynchronisme

évènements

```
let img = new Image();

img.src = "http://www.meteofrance.com/vigilance/mn.gif";

console.log(img.width); //0
//car l'image est chargée de manière asynchrone

//mais
img.addEventListener("load", () => console.log(img.width) /* 71 */);

//quand l'image est chargée, l'évènement "load" est déclenché
//et les fonctions liées à l'évènement sont exécutées.
```

Asynchronisme

Callback hell

```
//Supposons que nous voulions charger une série d'images,
//les unes après les autres, avec une pause de 5 secondes entre chaque
let img = new Image();
img.src = "http://monServeur/monImage1.png";
img.addEventListener("load", () => {
  window.setTimeout(() => {
    let img = new Image();
    img.src = "http://monServeur/monImage2.png";
    img.addEventListener("load", () => {
      window.setTimeout(() => {
        let img = new Image();
        img.src = "http://monServeur/monImage3.png";
        img.addEventListener("load", () => {
          //etc etc
        });
      },5000);
    });
  },5000);
});
```

Asynchronisme

Callback hell, tentative d'amélioration

```
function chargerImage(src,callback) {
  let img = new Image();
  img.src = "http://monServeur/"+src;
  img.addEventListener("load", () => window.setTimeout(callback, 5000));
}

chargerImage("monImage1.png", () => {
  chargerImage("monImage2.png", () => {
    chargerImage("monImage3.png", () => {
      //etc etc
    });
  });
});
```

Promesses

```
function chargerImage(src) {
  return new Promise(resolve => {
    let img = new Image();
    img.src = "http://monServeur/"+src;
    img.addEventListener("load", () => resolve(img));
  });
}

function attendre(s) {
  return new Promise(resolve => window.setTimeout(resolve,s*1000));
}

chargerImage("monImage1.png")
.then(() => attendre(5))
.then(() => chargerImage("monImage2.png"))
.then(() => attendre(5))
.then(() => chargerImage("monImage3.png"))
.then(() => attendre(5));
```

https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Utiliser_les_promesses

Promesses

Gestion des erreurs

```
function chargerImage(src) {
  return new Promise((resolve, reject) => {
    let img = new Image();
    img.src = "http://monServeur/"+src;
    img.addEventListener("load", () => resolve(img));
    img.addEventListener("error", () => reject(new Error("erreur de chargement")));
  });
}

chargerImage("monImage1.png")
.then(() => attendre(5))
.then(() => chargerImage("monImage2.png"))
.then(() => attendre(5))
// exécuté si une erreur se produit dans un des blocs précédents
.catch(e => console.error(e))
// exécuté dans tous les cas
.then(() => console.log("traitement terminé"));
```

Async / await

Façon la plus aboutie et la plus claire de gérer l'asynchronisme.

```
async function chargerTout() {
  await chargerImage("monImage1.png");
  await attendre(5);
  await chargerImage("monImage2.png");
  await attendre(5);
  await chargerImage("monImage3.png");
  await attendre(5);
}

chargerTout();
```

Async / await

Gestion des erreurs

```
async function chargerTout() {
  try {
    await chargerImage("monImage1.png");
    await attendre(5);
    await chargerImage("monImage2.png");
    await attendre(5);
    await chargerImage("monImage3.png");
    await attendre(5);
  } catch (e) {
    console.error(e);
  }
}

chargerTout();
```