



DÉVELOPPEMENT INFORMATIQUE LOGIQUE & PROGRAMMATION

LANGAGE JAVAScript

Y. DELVIGNE

CH. LAMBEAU



INTRODUCTION AU CONCEPT DE FONCTION

qu'est ce c'est ?

Séquence (bloc de code, miniscript) **nommée**

pour quoi faire ?

- soit effectuer une **action** particulière
cfr. `alert()` ou `document.write()`, `console.log()`
- soit 'calculer' (renvoyer) une **valeur**
cfr. `prompt()` ou `parseInt()` ou `isFinite()`, `isNaN()`
- soit les deux ...

"écrire" à
l'utilisateur

respectivement :
"string"
"number"
"boolean"

Objectifs et avantages :

- structurer le code (découpage "procédural")
- réutilisabilité
- outil de la programmation "événementielle"
- outil de la programmation "objet" (méthodes)

comment faire ?

1. Déclaration : commence par le mot **function**

- **nom** (identificateur) suivi de ()
 - corps (**code** du script de la fonction) entre { };
- 👉 la déclaration est une description de ce qu'elle doit faire

2. Invocation

- provoque l'exécution
- 👉 l'invocation fait faire

```
// déclaration de la fonction
function hello() {
    console.log('hello world !');
}
// invocations
hello();
hello();
```

nom

code

faire
exécuter
le code

Fonctions & Variables

Les fonctions ont accès aux variables du script en lecture/écriture de telles variables sont qualifiées de **globales**

```
// déclaration variables globales
var quant, prixUnit, prixHtva, taxe, prixTvaC;
var tva = 0.06;

// déclaration fonctions
function saisieDonnees() {
    quant = prompt("quelle quantité ?");
    prixUnit = prompt("quel prix unitaire ?");
}

function calculPrix() {
    prixHtva = quant * prixUnit;
    taxe = prixHtva * tva;
    prixTvaC = prixHtva + taxe;
}

// le script : séquence ordonnée d'invocations
saisieDonnees();
calculPrix();
```

fonctions "procédurales" :
modifient les variables
globales par affectation

procédure = action
(affectation)

invocation (exécution)

Fonction = (généralisation de) valeur

Une fonction peut également avoir pour but de "calculer" une valeur et de renvoyer celle-ci

Les variables globales ne sont plus accédées qu'en lecture seule

La fonction renvoie la valeur "calculée" avec l'instruction **return**

```
// déclaration variables globales
var francs, euros, taux = 40.3399;

// déclaration fonctions
function saisieFrancs() {
    return +prompt("somme en francs ?");
}

function francs2euros() {
    return francs / taux;
}

// le script : séquence ordonnée d'invocations
francs = saisieFrancs();
euros = francs2euros();
```

'vraies' fonctions :
aucune modification
des variables globales

fonction :
valeur à affecter

fonction :
rvalue

PARAMÈTRES : SOUPLESSE ET RÉUTILISATION

Plutôt que d'accéder aux variables globales, une fonction peut recevoir leur valeur sous forme de paramètres; cela permet de réutiliser la fonction avec des jeux de données différents

1. Déclaration : commence par le mot **function**
 - **nom** (identificateur) suivi de ()
 - entre les () : un ou plusieurs **identificateurs** : les (noms des) **paramètres** (formels)
 - corps (**code** du script de la fonction) entre { } utilisant les paramètres comme variables
 - les paramètres sont considérés comme **variables locales**
2. Invocation
 - exécution : nom de la fonction en spécifiant entre () les **arguments** (paramètres effectifs)

PARAMÈTRES : SOUPLESSE ET RÉUTILISATION

nom de la fonction

nom du paramètre

utilisation du paramètre
dans le code

```
// déclaration de la fonction
function hello(nm) {
    alert('hello ' + nm + ' !');
}
// invocations
var w = 'world';
hello('Yves');
hello(w);
```

invocation avec passage
de valeur (littérale ou
contenu de variable)

fonction avec 2 paramètres

renvoi de valeur

```
// déclaration de la fonction
function min(x, y) {
    if (x < y)
        return x;
    else
        return y;
}
// invocations
var xx = 5, yy = -3;
var z = min(23, 15);
alert(min(xx, yy));
```


Fonction : rôle de **return**

L'instruction **return** entraîne :

- le **renvoi** d'une (seule) valeur au contexte qui l'a invoquée
 - la **fin** immédiate de l'exécution de la fonction (les instructions qui suivraient **return** ne sont pas exécutées)
- ➔ écrire les alternatives différemment dans une fonction

conventionnel, lourd ☹ ☹

```
// déclaration de la fonction
function min(x, y) {
  if (x < y)
    return x;
  else
    return y;
}
```

"pro" 😊 😊

```
// déclaration de la fonction
function min(x, y) {
  if (x < y) return x;
  return y;
}
```

très "pro" 😊 😊 😊

```
// déclaration de la fonction
function min(x, y) {
  return (x < y) ? x : y;
}
```

Fonction : variables locales

qu'est ce c'est ?

- variable déclarée à l'intérieur (au début) du code de la fonction
- donc 'privée' contrairement à une variable globale qui est 'publique'

pour quoi faire ?

- permettre à une fonction de disposer de variables temporaires pour lui permettre d'effectuer son 'métier'
- éviter d'alourdir le script avec des variables inutiles
- éviter les conflits entre noms de variables 'privées' et 'publiques'

```
// combien de racines à l'équation du second degré ?  
function nbRacines(a, b, c) {  
    var delta; // variable locale  
    delta = b * b - 4 * a * c;  
    switch(true) {  
        case (delta < 0) : return 0;  
        case (delta == 0) : return 1;  
        default          : return 2;  
    }  
}
```


PARAMÈTRES + VARIABLES LOCALES : EXEMPLE (1)


```
// script
var cote1, cote2, moyenne;

cote1 = +prompt('quelle est la cote (0..20) ?');
while (cote1 < 0 || cote1 > 20) {
    cote1 = +prompt('incorrect, encore (0..20) ?');
}

cote2 = +prompt('quelle est la cote (0..20) ?');
while (cote2 < 0 || cote2 > 20) {
    cote2 = +prompt('incorrect, encore (0..20) ?');
}

moyenne = (cote1 + cote2) / 2;
```

 **redondance**

 **beaucoup trop de valeurs littérales dans le code !**
et si la cotation passait sur 30 ?

PARAMÈTRES + VARIABLES LOCALES : EXEMPLE (2)

```
// script
var min = 0, max = 20;
var msg1 = 'quelle est la cote (' + min + '...' + max + ') ?';
var msg2 = 'incorrect, encore (' + min + '...' + max + ')';
var cote1, cote2, moyenne;

cote1 = +prompt(msg1);
while (cote1 < min || cote1 > max) {
    cote1 = +prompt(msg2);
}

cote2 = +prompt(msg1);
while (cote2 < min || cote2 > max) {
    cote2 = +prompt(msg2);
}

moyenne = (cote1 + cote2) / 2;
```

 redondance

PARAMÈTRES + VARIABLES LOCALES : EXEMPLE (3)

```
// déclaration fonction
function getCote(min, max) {
  // fournit un nombre entre min et max
  // variables locales
  var c;
  var msg1 = 'quelle est la cote (' + min + '..' + max + ') ?';
  var msg2 = 'incorrect, encore (' + min + '..' + max + ')';
  // validateur
  c = +prompt(msg1);
  while (c < min || c > max) {
    c = +prompt(msg2);
  }
  // renvoi : c est entre min et max !
  return c;
}

// script
var moyenne = (getCote(0, 20) + getCote(0, 20)) / 2;
```

PARAMÈTRES + VARIABLES LOCALES : EXEMPLE (4)

```
// déclaration fonction
function getCote(min, max) {
  // fournit un nombre entre min et max
  var c;
  var msg1 = 'quelle est la cote (' + min + '..' + max + ') ?';
  var msg2 = 'incorrect, encore (' + min + '..' + max + ')';
  c = +prompt(msg1);
  while (c < min || c > max) {
    c = +prompt(msg2);
  }
  return c;
}

// script
var total = 0, moyenne;
var nbCotes = +prompt('combien de cotes ? ');
for(var i = 1; i <= nbCotes; i++) {
  total += getCote(0, 20);
}
moyenne = total / nbCotes;
```

PARAMÈTRES : RÉUTILISATION, **GETTERS**, **SETTERS**

```
// déclaration fonction
function setElem(id, val){ // setter (procédure)
// écrit la valeur val dans l'élément d'identifiant id
    document.getElementById(id).innerText = val;
}
function getElem(id){ // getter (fonction)
// renvoie le contenu de l'élément d'identifiant id
    return document.getElementById(id).innerText;
}

// script
var tauxTva = 0.21, prxUnit = 22.5;
var prxHtva, prxTvaC;
var quantite = +prompt('quelle quantité ?');
if (isNaN(quantite)) {
    setElem('msg', 'erreur, pas un nombre !'); }
else {
    prxHtva = quantite * prxUnit;
    prxTvaC = prxHtva * (1 + tauxTva);
    setElem('prxH', prxHtva);
    setElem('prxC', prxTvaC);
}
```

OÙ METTRE LES DÉCLARATIONS DE FONCTIONS (1) ?

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
function setElem(id, val){ // setter
    // écrit la valeur val dans l'élément d'identifiant id
    document.getElementById(id).innerText = val;
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<!-- contenu html (balises) -->
```

```
<script>
```

```
var tauxTva = 0.21, prxUnit = 22.5, prxHtva, prxTvaC;
var quantite = +prompt('quelle quantité ?');
if (isNaN(quantite)) {
    setElem('msg', 'erreur, pas un nombre !'); }
else {
    prxHtva = quantite * prxUnit; prxTvaC = prxHtva * (1 + tauxTva);
    setElem('prxH', prxHtva);      setElem('prxC', prxTvaC);
}
```

```
</script>
```

```
</body>
```

```
</html>
```

• DANS LE HEAD ! ← DÉCLARATIONS

• DANS LE BODY ! ← INVOCATIONS

OÙ METTRE LES DÉCLARATIONS DE FONCTIONS (2) ?

```
function setElem(id, val){ // setter  
// écrit la valeur val dans l'élément d'identifiant id  
    document.getElementById(id).innerText = val;  
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <script src=JS/myScripts.js></script>
```

```
  </head>
```

```
  <body>
```

```
    <!-- contenu html (balises) -->
```

```
    <script>
```

```
      var tauxTva = 0.21, prxUnit = 22.5, prxHtva, prxTvaC;
```

```
      var quantite = +prompt('quelle quantité ?');
```

```
      if (isNaN(quantite)) {
```

```
        setElem('msg', 'erreur, pas un nombre !'); }
```

```
      else {
```

```
        prxHtva = quantite * prxUnit; prxTvaC = prxHtva * (1 + tauxTva);
```

```
        setElem('prxH', prxHtva);      setElem('prxC', prxTvaC);
```

```
      }
```

```
    </script>
```

```
  </body>
```

```
</html>
```

• DANS LE HEAD ← FICHER EXTERNE

• DANS LE BODY ! ← INVOCATIONS