

Module 10

Compléments sur les fonctions

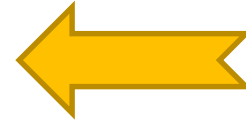
Technologies web

HENALLUX — IG2 — 2016-2017

Compléments sur les fonctions

➤ Gestion des paramètres de fonctions

- Paramètres optionnels, valeurs par défaut
- Surcharge
- Paramètres en nombre indéterminé
- Déstructuration




➤ Le mot-clef "this"

➤ Fonctions comme objets de premier ordre

➤ Pour aller plus loin : les closures

Gestion des paramètres

- 
- **Rappels**
 - **Paramètres optionnels**
 - **Surcharge**
 - **Paramètres en nombre indéterminé**
 - **Déstructuration**

Ensuite : *Le mot-clef "this"*

Rappels

- Sur les paramètres : **aucun contrôle de type**
 - Aucune déclaration de type non plus
 - *Comment surcharger une fonction ?*
- Sur les paramètres : **aucun contrôle de nombre**
 - On peut appeler une fonction avec "pas assez" ou "trop" d'arguments.
 - **Pas assez** : on complète avec des valeurs undefined.
 - **Trop** : on ignore les paramètres supplémentaires.

Paramètres optionnels

- Cas #1 : **paramètre optionnel simple**

```
function créePara (texte, classe) {  
    let code = "<p";  
    if (classe) code += " class='" + classe + "'";  
    code += ">" + texte + "</p>";  
    return code;  
}
```

- Conversion de "classe" en booléen (`undefined` → `false`)
 - *Rappel* : donne "true" sauf pour les valeur falsy
 - *Valeurs falsy* : `false`, `null`, `undefined`, `0`, `NaN`, `""`
- **Attention** : uniquement si aucune valeur falsy n'est valide !
- Protection : tester `if (classe === undefined)`

Paramètres optionnels

- Cas #2 : **paramètre optionnel avec valeur par défaut (1/2)**

```
function saluer (nom) {  
  if (nom) {  
    alert("Hi, " + nom + "!");  
  } else {  
    alert("Hi, anonyme!");  
  }  
}
```

```
function saluer (nom) {  
  nom = nom || "anonyme";  
  alert("Hi, " + nom + "!");  
}
```

- Conversion de « nom » en booléen (`undefined` → `false`)
 - *Rappel* : donne "true" sauf pour les valeurs falsy
 - *Valeurs falsy* : `false`, `null`, `undefined`, `0`, `NaN`, `""`
- **Attention** : uniquement si aucune valeur falsy n'est valide !

Paramètres optionnels

- Cas #2 : **paramètre optionnel avec valeur par défaut (2/2)**

```
function salue (nom = "anonyme") {  
    alert("Hi, " + nom + "!");  
}
```

- Seul `undefined` déclenche le calcul et l'utilisation de la valeur par défaut (sans ça, la valeur par défaut n'est même pas évaluée).
- On peut utiliser les paramètres précédents comme valeur par défaut :

```
function créeLien (url, texte = url) {  
    return `\${texte}</a>`;   
}
```

Surcharge

- Si on définit deux fois une fonction (même nom), seule la dernière définition compte.

- Il faut donc **traiter tous les cas en une seule fonction** !

```
function retardTrain (retard) {  
    if (typeof retard == "number")  
        return retard + " minute(s)";  
    if (typeof retard == "string")  
        return retard;  
}
```

- [Clean Code] Rien ne vous empêche de définir plusieurs fonctions de noms différents pour séparer les cas.

```
function retardTrain (retard) {  
    if (typeof retard == "number") return retardNum(retard);  
    if (typeof retard == "string") return retardStr(retard);  
}
```


Nb de paramètres variable

- Fonction qui peut être appelée avec un **nombre quelconque d'arguments** (exemple : `Math.min`)

```
function moyenne (...valeurs) {  
  let somme = 0;  
  for (let valeur of valeurs) somme += valeur;  
  return somme / valeurs.length;  
}
```
- Le **"rest" parameter** `...valeurs` rassemblent tous les autres arguments en un tableau.
- Il ne peut y avoir qu'un seul paramètre "rest" par fonction et celui-ci doit se trouver en dernière position !
- *Ancienne méthode (pré-ES6)* : dans la fonction, utiliser la variable locale prédéfinie "arguments" = tableau des paramètres effectifs.

Nb de paramètres variable

- Le pendant du "rest" parameter est le "**spread operator**".
 - Rest : conversion liste d'arguments -> tableau
 - Spread : conversion tableau -> liste d'arguments
- Exemples :


```
let tempSemaine1 = [15, 17, 16, 14, 13, 12, 15];  
moyenne(...tempSemaine1);  
  
moyenne(16, ...tempSemaine1);  
  
let tempSemaine2 = [18, 15, 13, 14, 11, 16, 12];  
moyenne(...tempSemaine1, ...tempSemaine2);
```
- Peut également s'utiliser dans un littéral de tableau :

```
let temp = [...tempSemaine1, ...tempSemaine2];
```
- Peut s'utiliser sur n'importe quel objet itérable.

Déstructuration

- La **déstructuration** permet d'accéder directement aux composantes d'une valeur structurée.
 - *Valeur structurée* = objets et tableaux (ou autres éléments similaires comme les collections).
 - *Accéder directement* = les associer à un nom de variable
- Exemples (tableaux) : `afficheHeure([13, 37, 30]);`

```
function afficheHeure (tHeure) {  
  console.log `${tHeure[0]}:${tHeure[1]}:${tHeure[2]}`;  
}
```



Pattern pour un tableau

```
function afficheHeure ([h, m, s]) {  
  console.log `${h}:${m}:${s}`;  
}
```


Faire correspondre une valeur avec un pattern = "pattern matching"

Déstructuration

- Exemples (objets) :

```
afficheHeure({heure: 13, min: 37, sec: 30});
```

```
function afficheHeure (oHeure) {  
  console.log `${oHeure.heure}:${oHeure.min}:${oHeure.sec}`;  
}
```



Pattern pour un objet

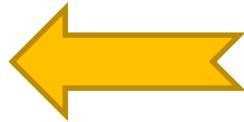
```
function afficheHeure ({heure, min, sec}) {  
  console.log `${heure}:${min}:${sec}`;  
}
```

Compléments sur les fonctions

➤ Gestion des paramètres de fonctions

- Paramètres optionnels, valeurs par défaut
- Surcharge
- Paramètres en nombre indéterminé
- Déstructuration

➤ Le mot-clef "this"



➤ Fonctions comme objets de premier ordre

➤ Pour aller plus loin : les closures