Le développement côté client avec JavaScript

Module 02 – Notions complémentaires

eni

Objectifs

- Savoir utiliser les fonctions en Javascript
- Connaître quelques fonctions importantes
- Savoir stocker des informations sur le navigateur
- Appréhender des éléments de sécurité
- Connaître le format JSON

Fonctions

Fonctions

- Syntaxe

```
function nom([parametre[, ...]]) {
     instructions
     [return …]
}
```

- Exemple d'utilisation

```
function afficherBonjour(){
    console.log("Bonjour !");
}
afficherBonjour();
```

Bonjour !

Fonctions anonymes

Fonctions anonymes

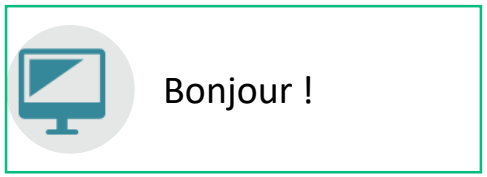
- Une fonction qui n’a pas de nom
- Souvent utilisée à un seul et même endroit
- Peut être stockée dans une variable
- Peut servir notamment comme paramètre ou retour dans d’autres fonctions

- Déclaration et stockage dans une variable

```
let afficherBonjour = function () {
    console.log("Bonjour !");
}
```

- Appel

```
afficherBonjour();
```



- Déclaration et auto-invocation

```
(function () {
    console.log("Bonjour !");
})();
```

- En passant un paramètre

```
(function (nom) {
    console.log("Bonjour " + nom);
})("Alex");
```

Fonctions fléchées (expression lambda)

Fonctions fléchées (ou expression lambda)

- Expression qui permet de créer de vraies fonctions
- Sucre syntaxique
- Ne possède pas son propre contexte d’exécution

- Déclaration et stockage dans une variable

```
let afficherBonjour = () => console.log("Bonjour !")
```

- Appel

```
afficherBonjour();
```

- En passant un paramètre, et en retournant une valeur

```
let  bonjour  = (nom) => "Bonjour " + nom;
```

- Équivalent à

```
let  bonjour  = (nom) => {  return "Bonjour " + nom };
```

- Appel

```
console.log(bonjour("Alex"));
```

Callback

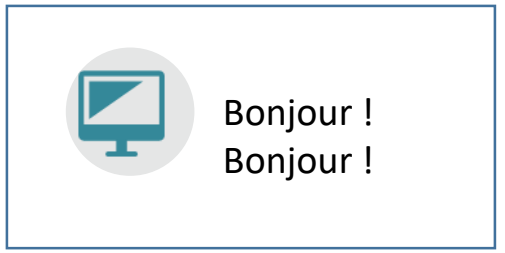
- Une callback est une fonction passée en paramètre d'une autre pour accomplir une action

- Exemple d'utilisation

```
function afficherBonjour(){
    console.log("Bonjour !");
}

function repeter(nombre, callback){

    for (let index = 0; index < nombre; index++) {
      callback();
    }
}

repeter(5, afficherBonjour);
```

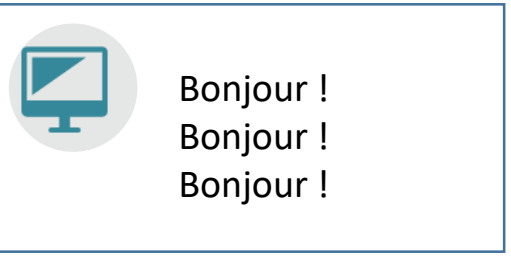
- En utilisant une fonction anonyme

```
function repeter(nombre, callback){

    for (let index = 0; index < nombre; index++) {
      callback();
    }
}

repeter(2, function(){
    console.log("Bonjour !");
});
```



- En utilisant une fonction fléchée

```
function repeter(nombre, callback){

    for (let index = 0; index < nombre; index++) {
      callback();
    }
}

repeter(3, () => console.log("Bonjour !"));
```



Démonstration

- setTimeout() est une fonction native qui permet d'exécuter une fonction passée en paramètre une et une seule fois, passé un certain temps

- setInterval() est une fonction native qui permet d'exécuter une fonction passée en paramètre de manière répétée à intervalle régulier

setTimeOut

- Syntaxe

delay en ms

paramètres additionnels passés à la callback

```
setTimeout(callback, delay [, param1, ...]);
```

- Exemple d'utilisation

```
function afficherBonjour(nom){
    console.log("Bonjour " + nom);
}

setTimeout(afficherBonjour, 5000, "Alex")
afficherBonjour("Marion")
```

Bonjour Marion
Bonjour Alex

setInterval

- Syntaxe

delay en ms

paramètres additionnels passés à la callback

```
setInterval(callback, delay [, param1, ...]);
```

- Exemple d'utilisation

```
function afficherBonjour(nom){
    console.log("Bonjour " + nom);
}

setInterval(afficherBonjour, 5000, "Alex");
afficherBonjour("Marion");
```

Bonjour Marion
Bonjour Alex
...
Bonjour Alex

clearInterval

- Arrêter l'exécution de l'intervalle

- setInterval() retourne un id correspondant à l'intervalle généré, la méthode clearInterval() permet de le stopper

Démonstration

TP

Mode strict

- Depuis ES5, il est possible d'activer le mode strict sur un fichier javascript
- Favorise l'optimisation du code par les navigateurs
- Interdit un certain nombre de mots-clés, comme nom de variables ou de fonctions
- Permet de lever des erreurs sur des fautes qui étaient précédemment autorisées

- Exemple d'utilisation

```
message = "Bonjour !";

console.log(message);
```

Bonjour !

```
'use strict';

message = "Bonjour !";

console.log(message);
```

Uncaught ReferenceError: assignment to undeclared variable message

Boîte de dialogue

- 3 fonctions natives qui permettent d'interagir de manière simple avec l'utilisateur
- Ouvre une popup ou fenêtre modale sur le navigateur
- alert() / prompt() / confirm()

alert()

- Permet d'informer l'utilisateur
- Syntaxe

```
alert(message);
```

- Exemple d'utilisation

```
alert("Bonjour !")
```

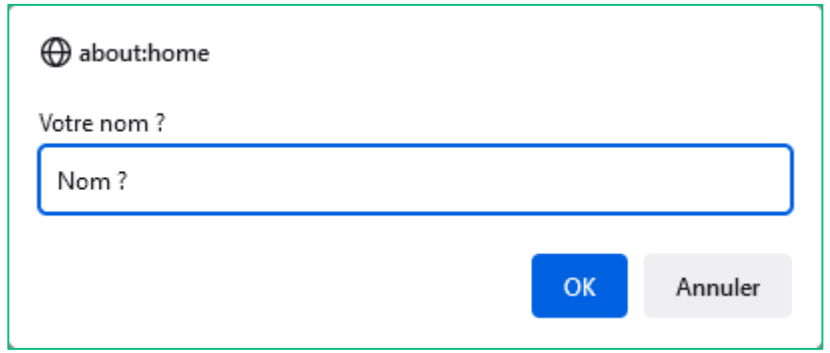
prompt()

- Permet une saisie utilisateur
- Syntaxe

```
prompt(message, default);
```

- Exemple d'utilisation

```
let nom = prompt("Votre nom ?", "Nom ?");
console.log(nom);
```



confirm()

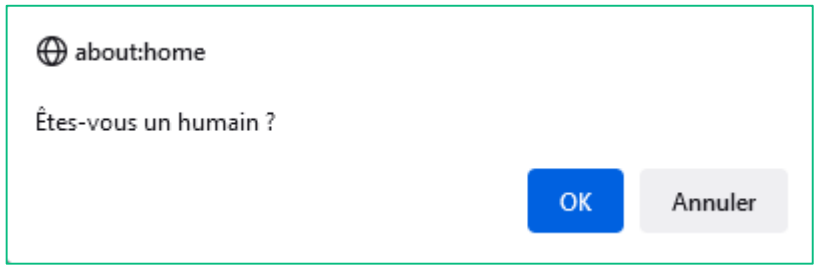
confirm

- Renvoie un booléen en fonction du bouton cliqué
- Syntaxe

```
confirm(message);
```

- Exemple d'utilisation

```
let reponse = confirm("Êtes-vous un humain ?");
console.log(reponse);
```



Démonstration

Stockage

- Permet de stocker des informations sur le navigateur sous forme de chaines de caractères

Cookies

- Limite de stockage de 4 ko par cookie
- Les données sont envoyées systématiquement au serveur à chaque requête HTTP

```
//Ajouter un cookie
document.cookie = "message=bonjour;max-age=86400;secure;"

//Récupérer les cookies
let cookies = document.cookie
```

max-age = durée de vie du cookie, par défaut c'est la session

secure = transmet les cookies uniquement si le protocole est sécurisé (https)

Stockage local

- Limite de stockage de 5 mo
- Accessible depuis la même origine
- Système de clé-valeur

sessionStorage = les données sont stockées le temps de la session

```
//Ajouter une entrée
localStorage.setItem('nom', 'Alex');

//Récupération
let nom = localStorage.getItem('nom');
```

```
//Suppression de l'entrée
localStorage.removeItem('nom');

//Supprimer tous les éléments
localStorage.clear();
```

Démonstration

Sécurité
Failles XSS

- Cross-site scripting
- Permet à un attaquant d'injecter du code (HTML, Javascript) malveillant dans un site web au travers d'une URL, d'un input…

Sécurité
Obfuscation

- Le code source Javascript est accessible depuis le navigateur
- Rendre le code difficilement décryptable pour un utilisateur malveillant
- Permet de ralentir ou décourager, la récupération de code ou les tentatives d'attaque

Démonstration

JSON

- JavaScript Object Notation
- Permet de structurer des données sous forme de chaines de caractères
- Facile à écrire et à lire pour les humains, comme pour les machines
- Format populaire d'échanges de données

- Système de clé-valeur
- Structure :

```
{
    "poisson" : "bar",
    "legume" : {"nom" : "carotte", "espece" :  "Daucus carota"},
    "fruits": [
        {"nom" : "pomme", "couleur" : "rouge"},
        {"nom" : "banane", "couleur" : "jaune"}
    ]

}
```

- Récupérer les données d'un fichier JSON

```
fetch('data.json')
.then(response => response.json())
.then(data => afficherData(data))
```

Nom du fichier où sont stockés les données

Méthode appelée une fois les données extraites

```
function afficherData(data){
    console.log(data);
}
```

Démonstration

TP

Objectifs

- Vous savez créer et utiliser des fonctions en javascript
- Vous savez stocker des informations sur un navigateur
- Vous connaissez des éléments de sécurité
- Vous connaissez le format JSON