

# Javascript

Programmer objet

# Prototype

Javascript est un langage basé sur les prototypes

```
function Personne(nom, age) {  
  this.nom = nom;  
  this.age = age;  
}  
  
Personne.prototype.vieillir = function() { this.age++; }  
  
let toto = new Personne("Toto", 25);  
toto.vieillir();
```

[Le modèle objet JavaScript en détails](#)

# Classes

Introduites en ES6 pour simplifier l'écriture du modèle prototypal.

```
class Personne {  
  constructor(nom, age) {  
    this.nom = nom;  
    this.age = age;  
  }  
  vieillir() {  
    this.age++;  
  }  
}  
  
let toto = new Personne("Toto", 25);  
toto.vieillir();
```

# Propriétés/méthodes privées

Par convention on commence les noms de variables par "\_".

```
class MaClasse {  
  
    constructor() {  
        this._propPrivee = true;  
    }  
  
    _methodePrivee() {  
        /* ... */  
    }  
}
```



# Propriétés/méthodes privées

Avec ES2022, on utilise le préfixe "#"

```
class MaClasse {  
  #propPrivee  
  
  #methodePrivee() {  
    /* ... */  
  }  
}  
  
let monObjet = new MaClasse()  
monObjet.#propPrivee // Error
```

# Méthodes statiques

```
class Personne {  
  
  constructor(nom,age) {  
    this.nom = nom || "inconnu";  
    this.age = age || 0;  
  }  
  
  static test(obj) {  
    return obj instanceof Personne;  
  }  
}  
  
let toto = {nom:"Toto",age:26};  
  
Personne.test(toto); //false
```

# Héritage

```
class Personne {  
    constructor(nom, age) {  
        this.nom = nom;  
        this.age = age;  
    }  
    vieillir() {  
        this.age++;  
    }  
}  
  
class Homme extends Personne {  
    constructor(nom,age) {  
        super(nom,age); //appel du constructeur parent  
        this.sexe="M";  
    }  
}
```

# Surcharge d'une méthode

```
class Personne {  
    constructor(nom,age) {  
        this.nom = nom;  
        this.age = age;  
        this.memoire = 2000;  
    }  
    vieillir() {  
        this.age++;  
    }  
}  
  
class Alzheimer extends Personne {  
    vieillir() {  
        super.vieillir();  
        this.memoire--;  
    }  
}  
  
let totoMalade = new Alzheimer("Toto",60);  
totoMalade.vieillir();
```



# Surcharge d'une méthode de conversion

```
class Personne {  
  constructor(nom,age) {  
    this.nom = nom;  
    this.age = age;  
  }  
  toString() { return this.nom; }  
  toJSON() { return '{"nom":"' + this.nom + '"}'; }  
}  
  
let toto = new Personne("Toto",25);  
  
console.log("je m'appelle " + toto); //je m'appelle Toto  
console.log( JSON.stringify(toto) ); //'{"nom":"Toto}"'
```

# Accesseurs et mutateurs

```
class Personne {  
  constructor(nom) {  
    this.nom = nom;  
    this._age = 0;  
  }  
  
  get age() {  
    return this._age;  
  }  
  
  set age(val) {  
    if (typeof val !== "number") throw new TypeError("pas un nombre");  
    this._age = val;  
  }  
}  
  
let toto = new Personne("Toto");  
toto.age; // 0  
toto.age = 25;  
toto.age = "titi"; // TypeError
```

# Espaces de noms

Il n'y a pas d'espace de noms à proprement parler en javascript.  
On utilise les objets.

```
let MABIBLIO = {};  
  
MABIBLIO.maConstante = "toto";  
  
MABIBLIO.maFonctionGenerique = function() { /* ... */ }  
  
MABIBLIO.MaClasse = class {  
  
  constructor() {  
    this.maPropriete1 = "";  
  }  
  
  maMethode1() { /*...*/ }  
}  
  
let toto = new MABIBLIO.MaClasse();
```

# En résumé

```
(function(root) { "use strict";

  let BIBLIO = {};

  BIBLIO.MaClasse = class {

    constructor() {
      this.maProprietePublique = ""
      this._maProprietePrivee = ""
    }

    maMethodePublique() { /*...*/ }

    _maMethodePrivee() { /*...*/ }

    get maPropriete() { return this._maProprietePrivee; }
    set maPropriete(val) { this._maProprietePrivee = val; }

    static maMethodeStatique() { /*...*/ }
  }

  BIBLIO.maFonctionGenerique = function() { /*...*/ }

  BIBLIO.maConstante = 25;

  root.BIBLIO = BIBLIO; //on expose la bibliothèque à l'espace global
})(this)); //valable quel que soit l'environnement (navigateur ou autre)
```