Introduction aux objets en JavaScript

07/11/2024 23:01

Philippe Genoud

Philippe.Genoud@univ-grenoble-alpes.fr

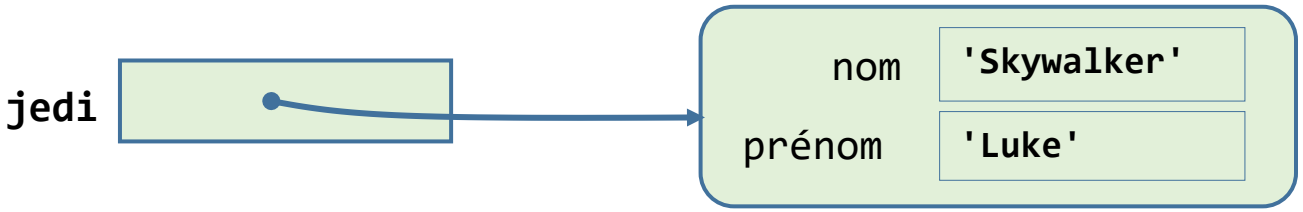
Types primitif / type objet

- Type primitif
  - Une variable de type primitif ne stocke qu'une simple valeur (chaîne de caractères, nombre, booléen…)
- Type objet
  - Un objet permet de définir des entités plus complexes en regroupant un ensemble de valeurs pouvant être soit des valeurs primitives soit d'autres objets.

```
let nom = 'Skywalker';
```

variable de type primitif
la variable stocke la valeur

```
let jedi = {
    nom : 'Skywalker',
    prenom: 'Luke';
}
```



variable de type objet
la variable stocke la référence (adresse) de l'objet

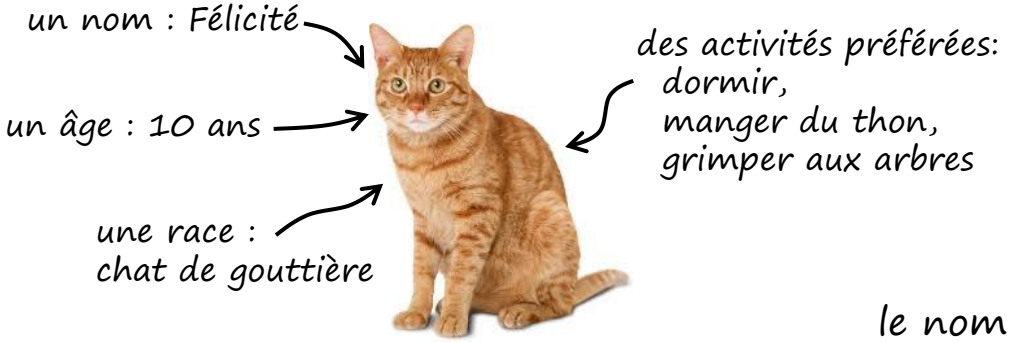
JavaScript et programmation orientée objet

- JavaScript permet l'utilisation d'objets et dispose d'objets natifs mais n'est pas à proprement parler un langage orienté objet au sens classique du terme
  - ne fournit pas d'éléments de langage pour supporter ce paradigme (par exemple pas de notion explicite de classe comme en Java, C++) mais émule certains de ses principes

- Deux types d'objets
  - Objets natifs
    - types prédéfinis : **`Array`**, **`String`**, **`Date`** …
    - objets liés à l'environnement d'exécution
      - **`window`** , **`document`** … dans le navigateur
      - **`process`** … dans NodeJs
  - Objets personnalisés
    - types définis par l'application

Syntaxe littérale

- objet JavaScript = une collection de valeurs nommées (propriétés ou attributs).
- sous sa forme littérale un objet est défini par :
  - un ensemble de couples *nom* : *valeur*, séparés par des , et délimité par { }



objet représentant ce chat

nom : "Félicité"
age : 10
race : "chat de goutière"
aime : [
"manger du thon",
"grimper aux arbres"
"dormir"]

code JavaScript : Littéral Objet

une variable référence

```
let felicite = {
    "nom": "Félicité",
    age: 10,
    race: "chat de gouttière",
    aime: ["manger du thon",
           "grimper aux arbres",
           "dormir"],
};
```

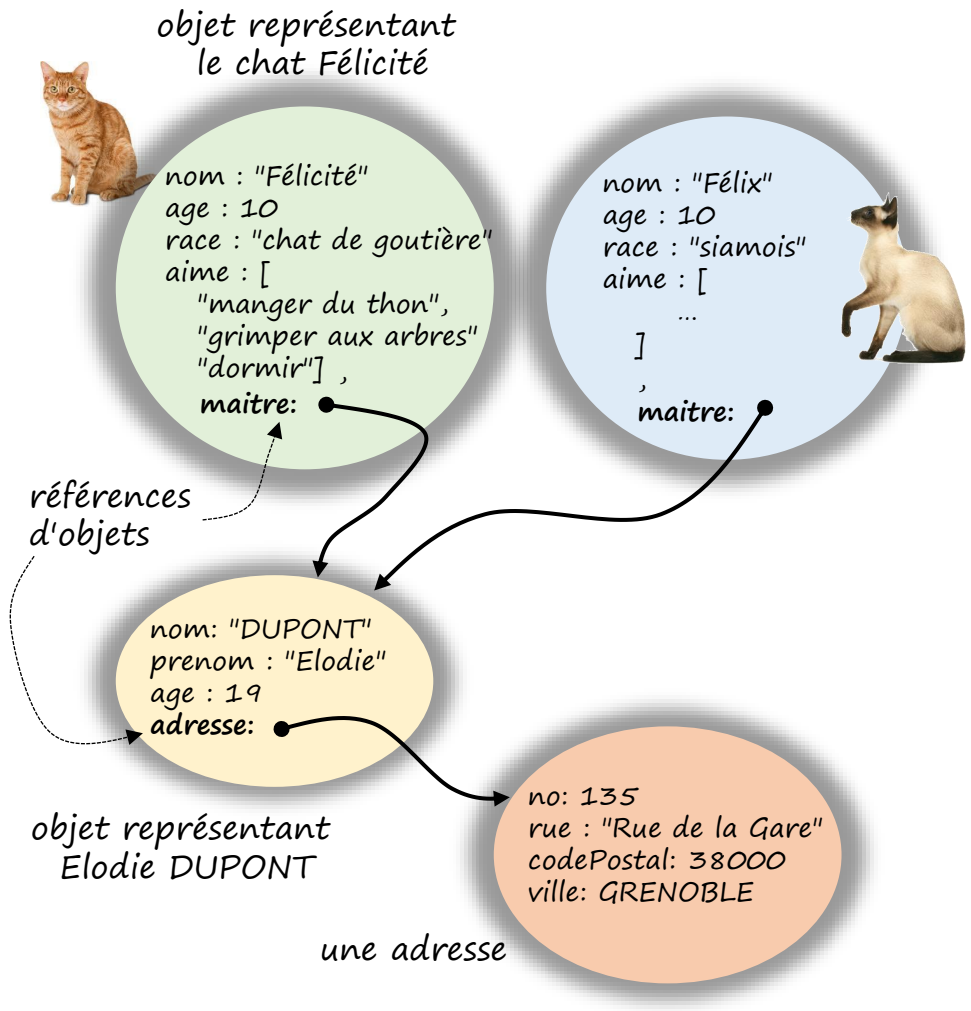
le nom d'une propriété peut être n'importe quelle chaîne de caractères (les guillemets sont facultatifs sauf si le nom est une chaîne qui ne correspond pas à un identificateur valide *)

trailing comma facultative
facilite ajout, suppression, déplacement de propriétés

*chaîne sans espaces ni caractères spéciaux (hormis $ et _) et ne commençant pas par un chiffre

Syntaxe littérale

- Objet javascript = un ensemble de propriétés (couples nom : valeur)
- Valeur d'une propriété peut être définie par n'importe quelle expression, et même depuis un autre objet littéral




```
let elodie = {
    nom: "DUPONT",
    prenom: "Elodie",
    age : 19,
    adresse:  {
        no: 135,
        rue: "Rue de la Gare",
        codePostal: 38000,
        ville: "GRENOBLE"
    } ,
};
```

objet littéral imbriqué

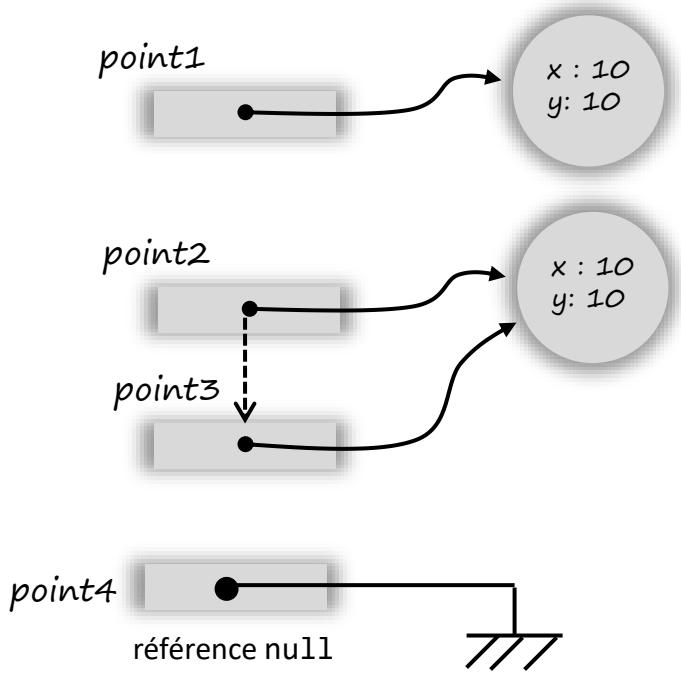
```
let felicite = {
   nom: "Félicité",
   age : 10,
   race : "chat de gouttière",
   aime : [
            "manger du thon",
            "grimper aux arbres",
            "dormir"
          ],
   maitre: elodie ,
};
```

référence

```
let felix = {
   nom: "Félix",
   age : 10,
   race : "Siamois",
   aime : [
            ...
          ],
   maitre: elodie ,
};
```

Références

- Pour désigner des objets on utilise des variables d'un type particulier : les **références**
- Une référence contient l'**adresse** d'un objet (elle pointe vers la structure de données correspondant aux propriétés (attributs) de l'objet)
- Affecter une référence à une autre référence consiste à recopier les pointeurs
- Une référence peut posséder la valeur `null` (aucun objet n'est accessible par cette référence)




```
let point1 = {
   x : 10,
   y : 10
};
```

```
let point2 = {
   x : 10,
   y : 10
};
```

```
let point3 = point2;
```

une référence n'est pas un objet, c'est un nom pour accéder à un objet

```
let point4 = null;
```

Accès à la valeur d'une propriété d'un objet

- notation pointée

`ref.nomDePropriété`

```
if (felicite.age > 2) {
   console.log("MAAOUU");
} else {
   console.log("miaou");
}
```

ne peut être utilisée que si le nom de la propriété est un identificateur valide (chaîne sans espaces ni caractères spéciaux (hormis `$` et `_`) et ne correspondant pas à un mot réservé du langage)

- notation crochets (*brackets*)

`ref[expressionDeTypeString]`

```
if (felicite["age"] > 2) {
 ...
```

une alternative à la notation pointée, indispensable si le nom de la propriété ne correspond pas à un identificateur valide

Accès à la valeur d'une propriété d'un objet

```
let myObject = {
    nom :  "attribut dont le nom est un identificateur",
    $nom : "autre attribut 'normal'",
    let : 'attribut dont le nom est un mot réservé',
    15 : 'attribut dont le nom est un entier',
    14 : 'attribut dont le nom est un entier',
    1.5 : "attribut dont le nom est un nombre flottant",
    "le prenom" : 'attribut dont le nom est un chaîne avec espaces',
    "5aaaaa" : "un attribut dont le nom n'est pas un identificateur valide"
};
```

nom de la propriété	notation pointée	notation crochets
nom	myObject.nom	myObject["nom"]
$nom	myObject.$nom	myObject["$nom"]
let	myObject.let	myObject["let"]
15	-	myObject["15"] mais aussi myObject[15]
14	-	myObject["14"] mais aussi myObject[14]
1.5	-	myObject["1.5"]
le prenom	-	myObject["le prenom"]
5aaaaa	-	myObject["5aaaaa"]

Obligatoire si le nom de la propriété n'est pas un identificateur valide

Accès à la valeur d'une propriété d'un objet

```
let elodie = {
   nom: "DUPONT",
   prenom: "Elodie",
   age : 19,
   adresse: {
      no: 135,
      rue: "Rue de la Gare",
      codePostal: 38000,
      ville: "Grenoble"
   } ,
};
```

```
let felicite = {
   nom: "Félicité",
   age : 10,
   race : "chat de gouttière",
   aime : [
            "manger du thon",
            "grimper aux arbres" ,
            "dormir"
           ],
   maitre: elodie
};
```

- accès à la valeur d'attributs d'objets imbriqués

  ex : accès au nom du maître de Félicité

  - notation pointée

    ```
    felicite.maitre.nom;
    ```

  - notation crochets (*brackets*)

    ```
    felicite["maitre"]["nom"]
    ```

  - possibilité de mélanger les différentes notations

    ```
    felicite["maitre"].nom
    ```

    ```
    felicite.maitre["nom"]
    ```

Accès à la valeur d'une propriété d'un objet

- avec la notation [ ] le nom de la propriété peut être calculé à l'exécution ou dépendre d'une valeur fournie par l'utilisateur

```
const readline = require('readline-sync');
const utils = require('./utils.js');

let felicite = {
   nom: "Félicité",
   age: 10,
   race: "chat de gouttière",
   aime: [
      "manger du thon",
      "grimper aux arbres",
      "dormir"
   ]
};

do {
   let nomProp = readline.question("nom de la propriété : ");
   console.log(`valeur de ${nomProp} : ${felicite[nomProp]}`);
} while (utils.encore('Voulez vous continuer ? '));
```

```
PS P:\M2CCI > node .\dynamicProp.js
nom de la propriété : nom
valeur de nom : Félicité
Voulez vous continuer ?  (O/N): o
nom de la propriété : race
valeur de race : chat de gouttière
Voulez vous continuer ?  (O/N): o
nom de la propriété : aime
valeur de aime : manger du thon,grimper aux arbres,dormir
Voulez vous continuer ?  (O/N): o
nom de la propriété : maitre
valeur de maitre : undefined
Voulez vous continuer ?  (O/N): n
PS P:\M2CCI >
```

renvoie la valeur **undefined** *lorsque l'on accède à une propriété qui n'est définie dans l'objet*

idem pour notation pointée

`felicite.maitre` → `undefined`

Accès à la valeur d'une propriété d'un objet

- Si on accède à un propriété qui n'existe pas la valeur retournée est `undefined`

```
let felicite = {
   "nom": "Félicité",
   age : 10,
   race : "chat de gouttière",
   aime : [
            "manger du thon",
            "grimper aux arbres",
            "dormir"
           ]
};
```

```
felicite.poids  →   undefined
```

- L'opérateur **`in`** permet de tester l'existence d'une propriété

```
"age" in felicite →   true
```

```
"poids" in felicite →   false
```

l'opérande gauche de **`in`** est une chaîne de caractères ou si c'est une expression d'un autre type elle sera convertie en chaîne de caractères

Accès à la valeur d'une propriété d'un objet

chainageOptionnel.js > ...

```
let chats = [
  {
    nom: "Félicité",
    age: 10,
    race: "chat de gouttière",
    aime: ["manger du thon", "grimper aux arbres", "dormir"],
    maitre: {
      nom: "DUPONT", prenom: "Elodie", age: 19,
      adresse: {
        no: 135,
        rue: "Rue de la Gare",
        codePostal: 38000,
        ville: "Grenoble",
      },
    },
  },
  {
    nom: "Felix",
    age: 12,
    race: "chartreux",
    aime: ["chasser les oiseaux", "dormir"],
  },
  {
    nom: "Eustache",
    age: 5,
    race: "chat siamois",
    aime: ["faire ses griffe sur le fauteuil", "dormir"],
    maitre: {
      nom: "DUPONT", prenom: "Mathieu", age: 19,
    },
  },
];
```

- chaîne de propriétés

```
for (let i = 0; i < chats.length; i++) {
  console.log(`${chats[i].nom}
      a pour maitre ${chats[i].maitre.prenom}
      habitant à ${chats[i].maitre.adresse.ville}`);
}
```

erreur pour le 2ème chat (Félix n'a pas de maître)

```
P:\M2CCI> node .\chainageOptionnel.js
Félicité
      a pour maitre Elodie
      habitant à Grenoble
chainageOptionnel.js:36
      a pour maitre ${chats[i].maitre.prenom}
                                           ^

TypeError: Cannot read properties of undefined (reading 'prenom')
    at Object.<anonymous> (P:\M2CCI\chainageOptionnel.js:36:39)
    ...

Node.js v17.7.2
P:\M2CCI>
```

Accès à la valeur d'une propriété d'un objet

chainageOptionnel.js > ...

```
let chats = [
  {
    nom: "Félicité",
    age: 10,
    race: "chat de gouttière",
    aime: ["manger du thon", "grimper aux arbres", "dormir"],
    maitre: {
      nom: "DUPONT", prenom: "Elodie", age: 19,
      adresse: {
        no: 135,
        rue: "Rue de la Gare",
        codePostal: 38000,
        ville: "Grenoble",
      },
    },
  },
  {
    nom: "Felix",
    age: 12,
    race: "chartreux",
    aime: ["chasser les oiseaux", "dormir"],
  },
  {
    nom: "Eustache",
    age: 5,
    race: "chat siamois",
    aime: ["faire ses griffe sur le fauteuil", "dormir"],
    maitre: {
      nom: "DUPONT", prenom: "Mathieu", age: 19,
    },
  },
];
```

- chaîne de propriétés
- opérateur **?.**

 + détails sur MDN

  - fonctionne de manière similaire à opérateur **.** mais si sur le chemin une référence est `undefined` ou `null` l'expression se court-circuite et renvoie la valeur `undefined`

```
for (let i = 0; i < chats.length; i++) {
  console.log(`${chats[i].nom}
      a pour maitre ${chats[i]?.maitre?.prenom}
      habitant à ${chats[i]?.maitre?.adresse?.ville}`);
}
```

```
P:\M2CCI> node .\chainageOptionnel.js
Félicité
      a pour maitre Elodie
      habitant à Grenoble
Felix
      a pour maitre undefined
      habitant à undefined
Eustache
      a pour maitre Mathieu
      habitant à undefined
P:\M2CCI>
```

Accès à la valeur d'une propriété d'un objet

- la boucle ***for … in*** permet d'énumérer toutes les propriétés d'un objet

```
for (let nomProp in unObjet) {
      // exécute le corps de la boucle pour chacun
      // des noms de propriété de l'objet unObjet
}
```

```
for (let prop in felicite) {
   console.log("ce chat a une propriété : " + prop);

   if (prop === "maitre") {
      console.log("Ce chat a pour maitre :");
      console.log(felicite[prop]);
   }

}
```

boucle for-in

la variable **prop** contient le nom de la propriété

c'est une chaîne de caractères

utilisation de la notation tableau pour accéder à la valeur de la propriété

```
λ node objectLiteraux2.js
ce chat a une propriété : nom
ce chat a une propriété : age
ce chat a une propriété : race
ce chat a une propriété : aime
ce chat a une propriété : maitre
Ce chat a pour maitre :
{
  nom: 'DUPONT',
  prenom: 'Elodie',
  age: 19,
  adresse: {
    no: 135,
    rue: 'Rue de la Gare',
    codePostal: 38000,
    ville: 'Grenoble'
  }
}
```

Accès à la valeur d'une propriété d'un objet

- dans quel ordre les propriétés de l'objet sont-elles accédées avec une boucle ***for … in*** ?

```
let myObject = {
    nom :  "attribut dont le nom est un identificateur",
    $nom : "autre attribut 'normal'",
    let : 'attribut dont le nom est un mot réservé',
    15 : 'attribut dont le nom est un entier',
    14 : 'attribut dont le nom est un entier',
    1.5 : "attribut dont le nom est un nombre flottant",
    "le prenom" : 'attribut dont le nom est un chaîne avec espaces',
    "5aaaaa" : "un attribut dont le nom n'est pas un identificateur valide"
};

for (let  prop in myObject) {
    console.log(prop + ' : ' + myObject[prop]);
}
```

```
14 : attribut dont le nom est un entier
15 : attribut dont le nom est un entier
nom : attribut dont le nom est un identificateur
$nom : autre attribut 'normal'
let : attribut dont le nom est un mot réservé
1.5 : attribut dont le nom est un nombre flottant
le prenom : attribut dont le nom est un chaîne avec espaces
5aaaaa : un attribut dont le nom n'est pas un identificateur valide
```

Les propriétés "entières" sont triées par ordre croissant et affichées en premier

Les autres propriétés apparaissent dans leur ordre de création

Modification des propriétés d'un objet

- changer la valeur d'une propriété
  - Par affectation directe

    ```
    felicite.race = "mélange Chartreux et " + felix.race;
    ```

  - Par opérateur

    ```
    felicite.age++;
    ```

  - Par appel d'une méthode (si la propriété est un objet ou un pseudo objet)

    ```
    felicite.aime.push("faire ses griffes sur le canapé");
    ```

- ajouter une propriété
  - si une affectation concerne une propriété non définie, une nouvelle propriété est crée

    ```
    felicite.poids = 3.5; // à partir de ce point l'objet félicité a une propriété poids
    ```

- supprimer une propriété
  - opérateur `delete` permet de supprimer un propriété d'un objet

    ```
    delete felicite.poids; // à partir de ce point félicité.poids est undefined
    ```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);
```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);
```

```
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ]
}
```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);

// modification d'une propriété
felicite.race = "mélange Chartreux + X";
felicite.age++;
felicite.aime.push("faire ses griffes sur le canapé");
console.log(felicite);
```

```
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ]
}
```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);

// modification d'une propriété
felicite.race = "mélange Chartreux + X";
felicite.age++;
felicite.aime.push("faire ses griffes sur le canapé");
console.log(felicite);
```

```
{
  nom: 'Félicité',
  age: 11,
  race: 'mélange Chartreux + X',
  aime: [
    'manger du thon',
    'grimper aux arbres',
    'dormir',
    'faire ses griffes sur le canapé'
  ]
}
```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);

// modification d'une propriété
felicite.race = "mélange Chartreux + X";
felicite.age++;
felicite.aime.push("faire ses griffes sur le canapé");
console.log(felicite);

// ajout d'une propriété
felicite.poids = 3.5;
console.log(felicite);
```

```
{
  nom: 'Félicité',
  age: 11,
  race: 'mélange Chartreux + X',
  aime: [
    'manger du thon',
    'grimper aux arbres',
    'dormir',
    'faire ses griffes sur le canapé'
  ]
}
```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);

// modification d'une propriété
felicite.race = "mélange Chartreux + X";
felicite.age++;
felicite.aime.push("faire ses griffes sur le canapé");
console.log(felicite);

// ajout d'une propriété
felicite.poids = 3.5;
console.log(felicite);
```

*à partir de ce point l'objet félicité a une propriété **poids***

```
{
  nom: 'Félicité',
  age: 11,
  race: 'mélange Chartreux + X',
  aime: [
    'manger du thon',
    'grimper aux arbres',
    'dormir',
    'faire ses griffes sur le canapé'
  ],
  poids: 3.5
}
```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);

// modification d'une propriété
felicite.race = "mélange Chartreux + X";
felicite.age++;
felicite.aime.push("faire ses griffes sur le canapé");
console.log(felicite);

// ajout d'une propriété
felicite.poids = 3.5;
console.log(felicite);

// supression d'une propriété
delete felicite.race;
console.log(felicite);
console.log(felicite.race);
```

```
{
  nom: 'Félicité',
  age: 11,
  race: 'mélange Chartreux + X',
  aime: [
    'manger du thon',
    'grimper aux arbres',
    'dormir',
    'faire ses griffes sur le canapé'
  ],
  poids: 3.5
}
```

Modification des propriétés d'un objet

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de gouttière",
    aime : [
              "manger du thon",
              "grimper aux arbres" ,
              "dormir"
             ]
};

console.log(felicite);

// modification d'une propriété
felicite.race = "mélange Chartreux + X";
felicite.age++;
felicite.aime.push("faire ses griffes sur le canapé");
console.log(felicite);

// ajout d'une propriété
felicite.poids = 3.5;
console.log(felicite);
```

*à partir de ce point l'objet félicité a une propriété **poids***

```
// supression d'une propriété
delete felicite.race;
console.log(felicite);
console.log(felicite.race);
```

*à partir de ce point **félicité.race** est **undefined***

```
{
  nom: 'Félicité',
  age: 11,
  aime: [
    'manger du thon',
    'grimper aux arbres',
    'dormir',
    'faire ses griffes sur le canapé'
  ],
  poids: 3.5
}
undefined
```

Objets en paramètres de fonction

- Un paramètre de fonction peut être une référence d'objet
  - déclaration

Le paramètre objet est défini comme n'importe quelle autre paramètre par un simple identifiant

accès aux propriétés de l'objet

```
function miauler(unChat) {
   if (unChat.age > 2) {
      console.log(unChat.nom + " dit MAAOUU");
   } else {
      console.log(console.log(unChat.nom + " dit miaou");
   }
}
```

  - appel

appel de la fonction avec la référence de l'objet sur lequel l'appliquer.

```
miauler(felicite);
```

```
Félicité dit MAAOUU
```

```
miauler( { nom: 'felix', age: 1.5} );
```

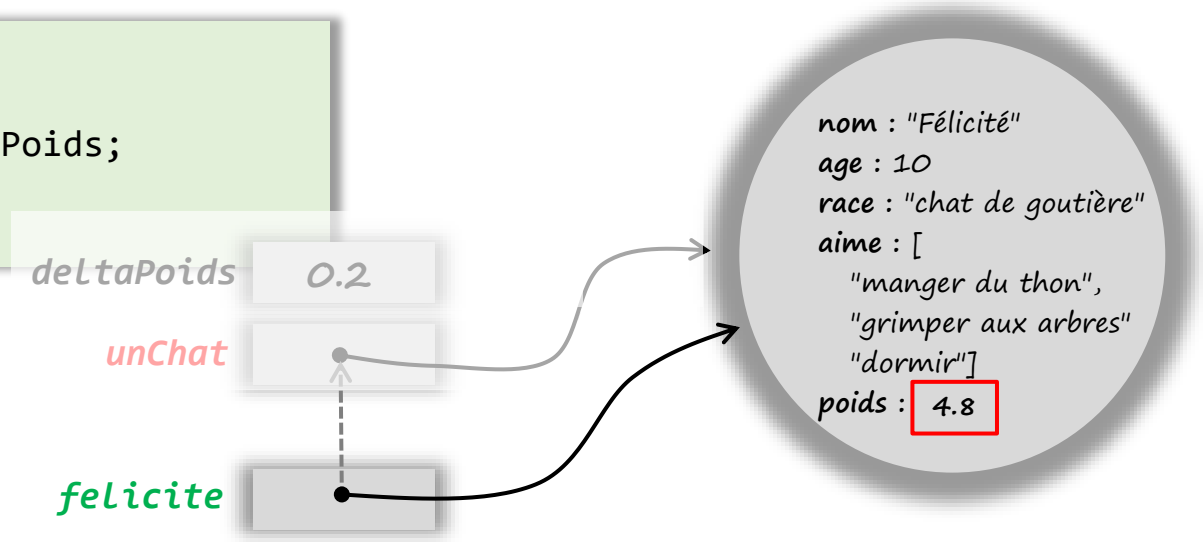
```
Félix dit miaou
```

appel de la fonction avec un littéral objet

Objets en paramètres de fonction

- comme en Java le passage de paramètres de fonctions est un passage par valeurs
  - la fonction dispose d'une copie de la variable, toute modification de cette copie ne sera pas visible en dehors de la fonction.
  - si la variable est une référence, les propriétés de l'objet référencé peuvent elles être modifiées.

```
function maigrir(unChat, deltaPoids) {

    unChat.poids = unChat.poids - deltaPoids;

}
```



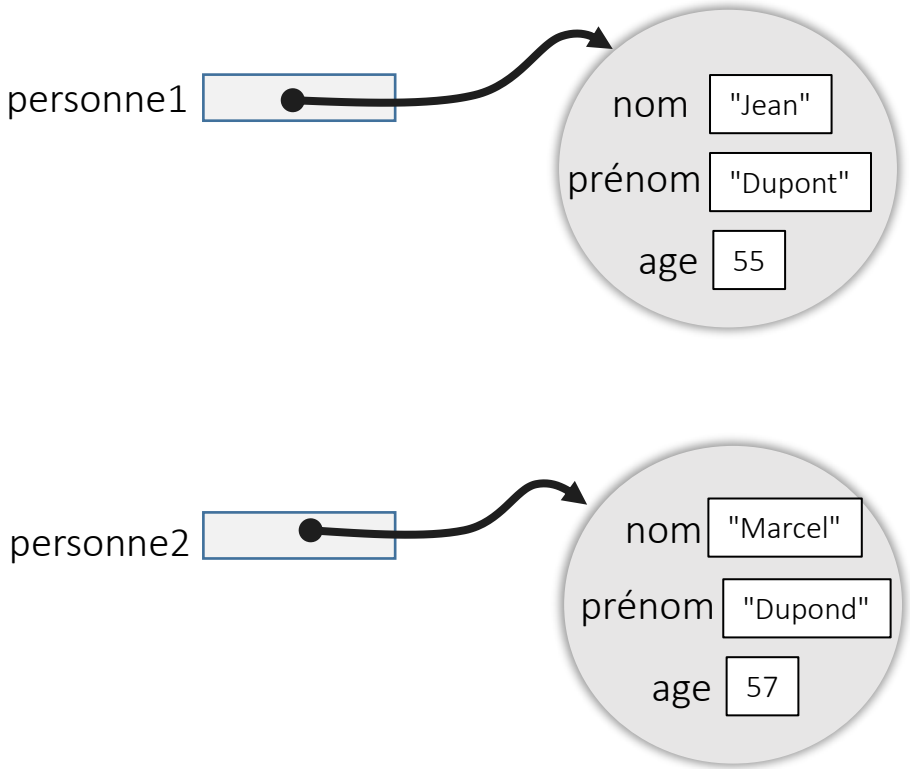

```
let felicite = { nom: "Félicité", age:10, poids: 5.0, …};
maigrir(felicite, 0.2);
console.log("Félicité pèse  " + felicite.poids + "  kg");
```

```
Félicité pèse  4.8  kg
```

Modification des objets `const`

- un objet déclaré avec `const` peut être modifié

```
1  const personne1 = {
2      prénom: "Jean",
3      nom: "Dupont",
4      age : 55,
5  };
6
7  let personne2 = {
8      prénom: "Marcel",
9      nom : "Dupond",
10     age : 57
11 };
12
13 personne1.age = 56;
14
15 console.log(personne1.age);
16
17 // personne1 = personne2;
18
19 personne2 = personne1;
```

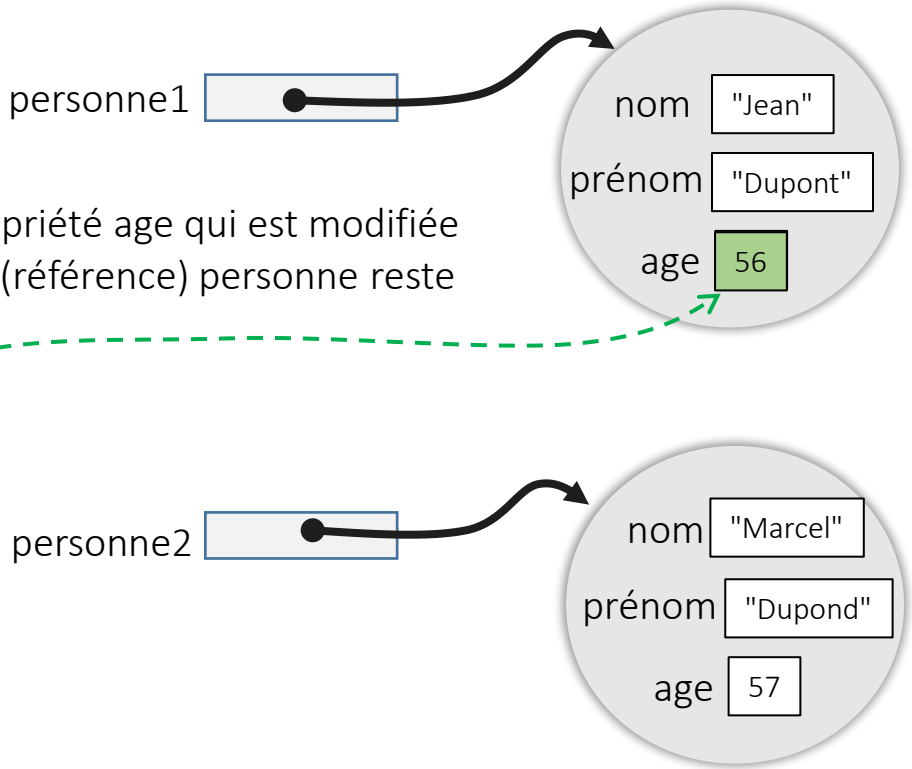


Modification des objets `const`

- un objet déclaré avec `const` peut être modifié

```
1  const personne1 = {
2      prénom: "Jean",
3      nom: "Dupont",
4      age : 55,
5  };
6
7  let personne2 = {
8      prénom: "Marcel",
9      nom : "Dupond",
10     age : 57
11 };
12
13 personne1.age = 56;
14
15 console.log(personne1.age);
16
17 // personne1 = personne2;
18
19 personne2 = personne1;
```

56



Modification des objets `const`

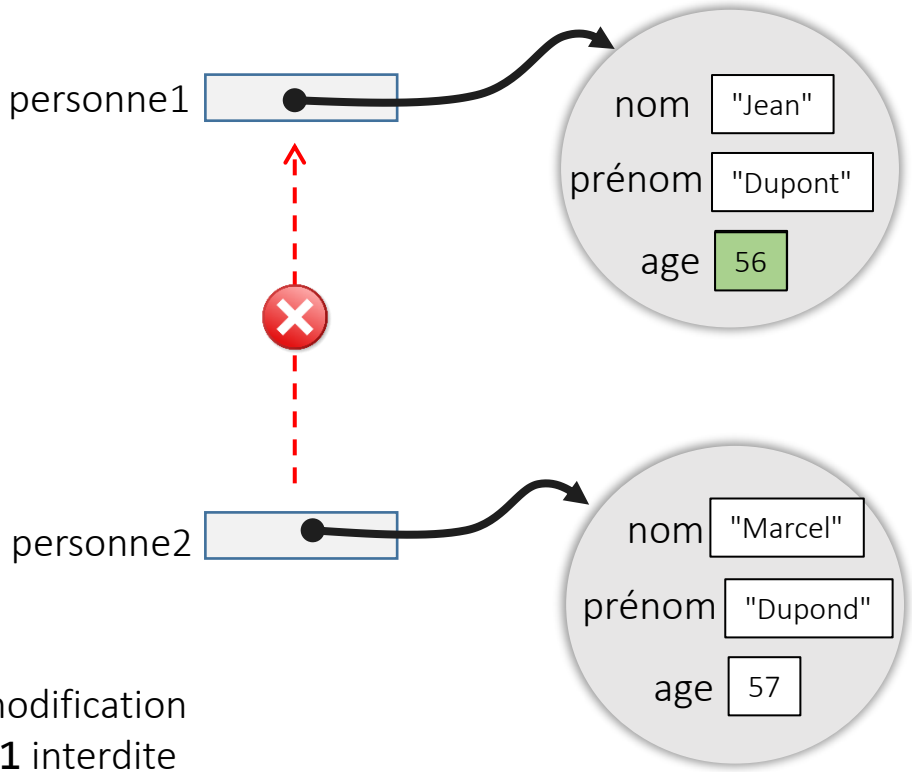
- un objet déclaré avec `const` peut être modifié

```
1  const personne1 = {
2      prénom: "Jean",
3      nom: "Dupont",
4      age : 55,
5  };
6
7  let personne2 = {
8      prénom: "Marcel",
9      nom : "Dupond",
10     age : 57
11 };
12
13 personne1.age = 56;
14
15 console.log(personne1.age);
16
17 // personne1 = personne2;
18
19 personne2 = personne1;
```

→ 56

tentative de modification de **`personne1`** interdite

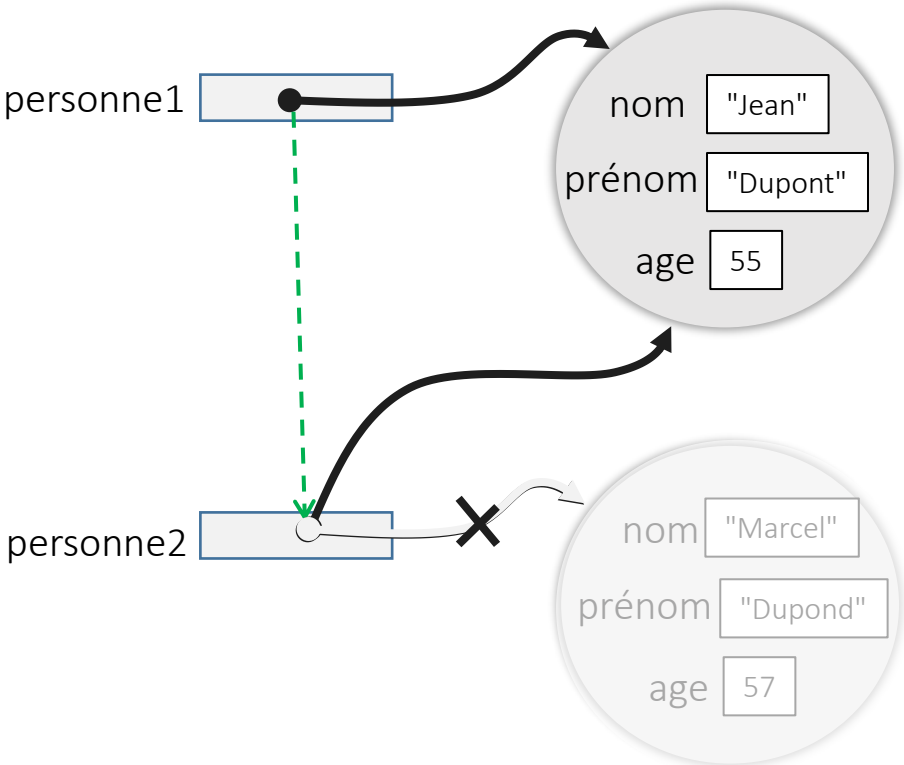
```
personne1 = personne2;
          ^

TypeError: Assignment to constant variable.
```



Modification des objets `const`

- un objet déclaré avec `const` peut être modifié

```
1  const personne1 = {
2      prénom: "Jean",
3      nom: "Dupont",
4      age : 55,
5  };
6
7  let personne2 = {
8      prénom: "Marcel",
9      nom : "Dupond",
10     age : 57
11 };
12
13 personne1.age = 56;
14
15 console.log(personne1.age);
16
17 // personne1 = personne2;
18
19 personne2 = personne1;
```



recopie de la valeur de la variable **personne1** dans la variable **personne2**

Si il n'y a pas d'autres références sur l'objet il ne pourra plus être accédé. La mémoire allouée sera libérée automatiquement par l'interpréteur JavaScript (*garbage collection*)

voir MDN

Méthodes

- Les objets ne sont pas qu'un regroupement de valeurs, les propriétés peuvent être aussi des fonctions (les objets peuvent aussi avoir un comportement).

```
let felicite = {
   nom: "Félicité",
   age : 10,
   race : "chat de gouttière",
   aime : [
            "manger du thon",
            "grimper aux arbres",
            "dormir"
           ],
   poids: 3.5 ,
   miauler: function () {
          console.log("Miaou ! Miaou !");
   }
};
```

une fonction anonyme est affectée à une propriété de l'objet

ce type de fonction est généralement appelé **méthode** de l'objet

- Invocation d'une méthode

```
felicite.miauler();
```

envoi du **message** `miauler` à l'objet référencé par `felicite`

Le mot clé `this`

- comment accéder aux attributs d'un objet dans une méthode ?

```
let felicite = {
    nom: "Félicité",
    age : 10,
    race : "chat de goutière",
    aime : [
              "manger du thon",
              "grimper aux arbres",
              "dormir"
            ],
    poids: 3.5,
    miauler: function () {
        console.log("Miaou ! Miaou !");
    },
     maigrir: function (deltaPoids) {

        this.poids = this.poids - deltaPoids;
     }

};
```

```
function maigrir(unChat, deltaPoids) {

   unChat.poids = unChat.poids - deltaPoids;

}
```

```
maigrir(felicite, 0.2);
```

transformer cette fonction en une méthode de l'objet.

pour désigner l'une des propriétés de l'objet, le mot clé **`this`** ***doit*** *être utilisé*

appel de la méthode

```
felicite.maigrir(0.2);
```

Dans une méthode **`this`** *désigne l'objet qui reçoit le message.*

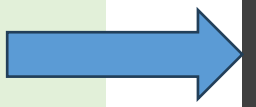
Le mot clé `this`

```
let felicite = {
    nom: "Félicité",
    age: 10,
    race: "chat de gouttière",
    aime: ["manger du thon", "grimper aux arbres", "dormir"],
    poids: 3.5,

    miauler: function () {
        console.log("Miaou ! Miaou !");
        console.log(`j'ai ${this.age} ans`);
    },

    maigrir: function (deltaPoids) {
        this.poids -= deltaPoids;
    },
};

console.log(felicite);
felicite.miauler();
felicite.maigrir(0.2);
console.log(felicite);
```

```
> node .\13_methode2.js
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ],
  poids: 3.5,
  miauler: [Function: miauler],
  maigrir: [Function: maigrir]
}
Miaou ! Miaou !
j'ai 10 ans
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ],
  poids: 3.3,
  miauler: [Function: miauler],
  maigrir: [Function: maigrir]
}
```

Le mot clé `this`

```
let felicite = {
    nom: "Félicité",
    age: 10,
    race: "chat de gouttière",
    aime: ["manger du thon", "grimper aux arbres", "dormir"],
    poids: 3.5,

    miauler: () => {
        console.log("Miaou ! Miaou !");
        console.log(`j'ai ${this.age} ans`);
    },

    maigrir:  (deltaPoids) => {
        this.poids -= deltaPoids;
    },
};

console.log(felicite);
felicite.miauler();
felicite.maigrir(0.2);
console.log(felicite);
```



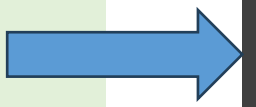
???

```
> node .\13_methode2.js
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ],
  poids: 3.5,
  miauler: [Function: miauler],
  maigrir: [Function: maigrir]
}
Miaou ! Miaou !
j'ai undefined ans
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ],
  poids: 3.5,
  miauler: [Function: miauler],
  maigrir: [Function: maigrir]
}
```

Le mot clé `this`

Attention : le contexte global n'est pas le même dans un environnement du navigateur et dans node

```
let felicite = {
    nom: "Félicité",
    age: 10,
    race: "chat de gouttière",
    aime: ["manger du thon", "grimper aux arbres", "dormir"],
    poids: 3.5,

    miauler: () => {
        console.log("Miaou ! Miaou !");
        console.log(`j'ai ${this.age} ans`);
    },

    maigrir:  (deltaPoids) => {
        this.poids -= deltaPoids;
    },
};

console.log(felicite);
felicite.miauler();
felicite.maigrir(0.2);
console.log(felicite);
```



```
> node .\13_methode2.js
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ],
  poids: 3.5,
  miauler: [Function: miauler],
  maigrir: [Function: maigrir]
}
Miaou ! Miaou !
j'ai undefined ans
{
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  aime: [ 'manger du thon', 'grimper aux arbres', 'dormir' ],
  poids: 3.5,
  miauler: [Function: miauler],
  maigrir: [Function: maigrir]
}
```

???

Dans une fonction fléchée, **this** ne fait pas référence à l'objet dans lequel la fonction est définie.
Dans une fonction fléchée **this** fait référence à l'objet **this** du contexte englobant (ici le **contexte global**)

Le mot clé `this`

- Dans une fonction fléchée, **this** ne fait pas référence à l'objet dans lequel la fonction est définie.
- Dans une fonction fléchée **this** fait référence à l'objet **this** du contexte englobant

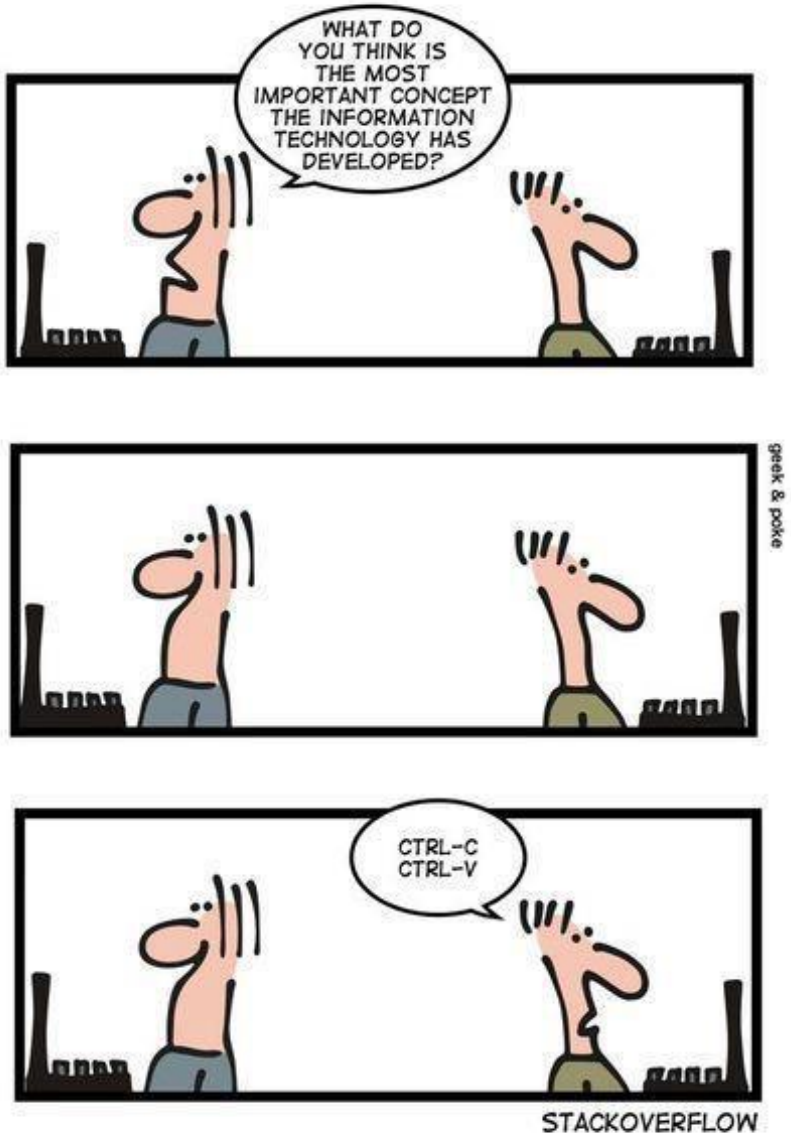
- l'objet this du contexte global est différent selon l'environnement d'exécution
  - Node.js : `{ }`
  - Browser : `window`

Créer plusieurs objets du même type ?


```
let felicite = {
   nom: "Félicité",
   age : 10,
   race : "chat de gouttière"
   aime : [
            "manger du thon",
            "grimper aux arbres"
            "dormir"
           ],
   poids: 3.5,

   miauler: function () {
     console.log("Miaou ! Miaou !");
   },
   maigrir : function(deltaPoids) {
      this.poids -= deltaPoids;
   }
};
```

```
let felix = {
   nom: "Félix",
   age : 6,
   race : "siamois"
   aime : [
            "se lécher",
            "manger des croquettes",
            "dormir"
           ],
   poids: 3.,

    miauler: function () {
      console.log("Miaou ! Miaou !");
    },
    maigrir : function(deltaPoids) {
       this.poids -= deltaPoids;
    }
};
```

Créer plusieurs objets du même type ?



Dupliquer du code est souvent (toujours ?) une mauvaise idée

- Source potentielle d'erreurs
- Difficulté de mises à jour
- Taille du code
- Lisibilité

Constructeur

Utiliser une fonction de création: **constructeur**

```
var felicite = {
   nom: "Félicité",
   age : 10,
   race : "chat de gouttière"
   aime : [
            "manger du thon",
            "grimper aux arbres"
            "dormir"
           ],
   poids: 3.5,
   miauler: function () {
        console.log("Miaou ! Miaou !");
   },
   maigrir : function(deltaPoids) {
      this.poids -= deltaPoids;
   }
};
```

une fonction comme une autre par convention débute par Majuscule

Souvent les paramètres définissent les valeurs des propriétés que l'on souhaite initialiser à la création de l'objet

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids ;
   this.miauler = function () {
      console.log("Miaou ! Miaou !");
   };
   this.maigrir = function(deltaPoids) {
      this.poids -= deltaPoids;
   };
}
```

initialisation des propriétés de l'objet avec les valeurs des paramètres

Définition des méthodes

un constructeur est invoqué par l'opérateur **new**

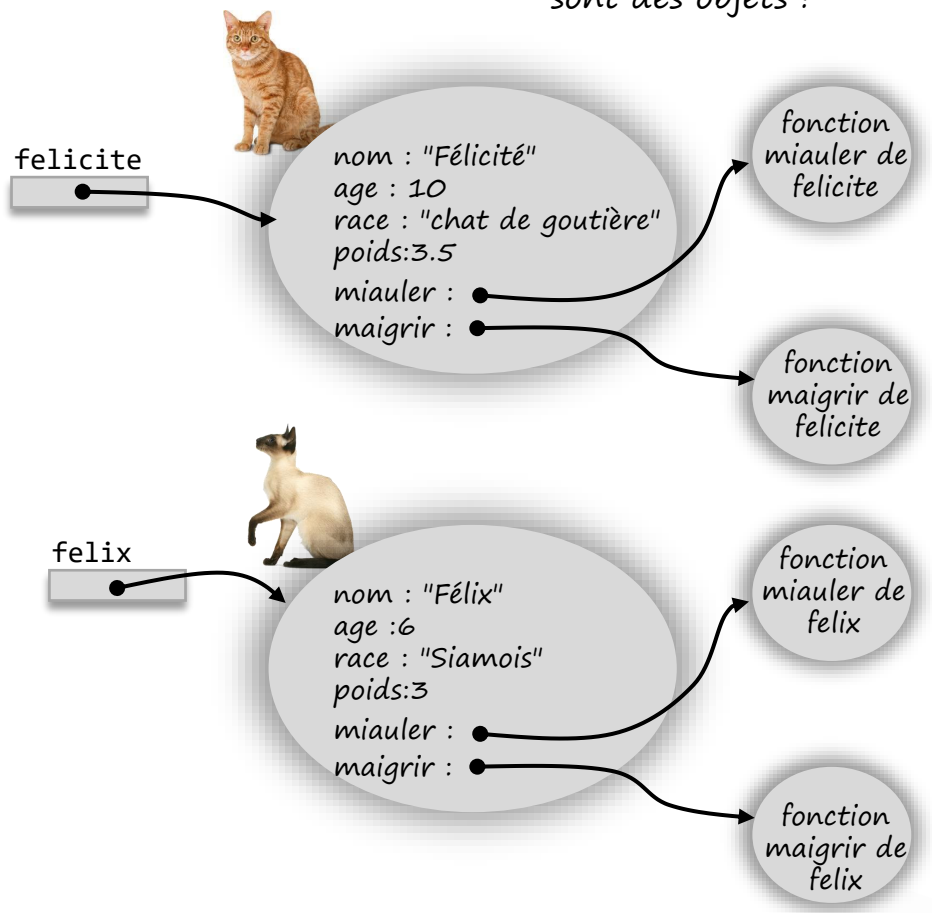
```
let felicite = new Chat("Félicité",10,"chat de gouttière", 3.5);
let felix    = new Chat("Felix",6,"siamois",3);
```

les objets référencés par `felicite` *et* `felix` *sont des '**instances**' de* **Chat**.

Constructeur et méthodes d'un objet

En JavaScript les fonctions sont des objets !

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids ;
   this.miauler = function () {
     console.log(this.nom + "-> Miaou ! Miaou !");
   };
   this.maigrir = function(deltaPoids) {
      this.poids -= deltaPoids;
   };
}

let felicite = new  Chat("Félicité",10,"chat de gouttière", 3.5);
let felix    = new  Chat("Felix",6,"siamois",3);
felicite.miauler();   ⟹  Felicité -> Miaou ! Miaou !
felix.miauler();      ⟹  Felix -> Miaou ! Miaou !
console.log("felicite.miauler === felix.miauler --> " +
             (felicite.miauler === felix.miauler));   ⟹  false
```



`felicite.miauler` *et* `felix.miauler` *référencent deux objets fonctions différents (même si ils font la même chose)*

Constructeur et méthodes d'un objet : prototype

- Les fonctions sont des objets
- Elles ont une propriété `prototype` :
  - liste de propriétés attachée à un constructeur (initialement vide)
  - une propriété rajoutée sur le prototype du constructeur devient disponible sur tous les objets créés à l'aide de ce constructeur (*fallback*)

Utiliser `prototype` du constructeur pour partager les méthodes

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids ;
}

Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Miaou ! Miaou !");
};
Chat.prototype.maigrir = function(deltaPoids) {
      this.poids -= deltaPoids;
};
let felicite = new  Chat("Félicité",10,"chat de gouttière", 3.5);
let felix    = new  Chat("Felix",6,"siamois",3);
felicite.miauler();   ⟹  Felicité -> Miaou ! Miaou !
felix.miauler();      ⟹  Felix -> Miaou ! Miaou !
console.log("felicite.miauler === felix.miauler --> " +
              (felicite.miauler === felix.miauler));   ⟹  true
```

Prototype pour définir des propriétés

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids ;
}

Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Miaou ! Miaou !");
};

Chat.prototype.maigrir = function(deltaPoids) {
     this.poids -= deltaPoids;
};

Chat.prototype.famille = "Felidés";
```

les propriétés rajoutées sur le prototype d'un constructeur ne se limitent pas à des fonctions méthodes elles peuvent être de n'importe quel type.

Une propriété définie sur le prototype d'un constructeur est accessible pour tous les objets (instances) créés via ce constructeur

```
let felicite = new  Chat("Félicité",10,"chat de gouttière", 3.5);
let felix    = new  Chat("Felix",6,"siamois",3);
console.log(felicite),
felicite.miauler();
console.log(felicite.famille);
console.log(felix.famille);
```

```
Chat {
  nom: 'Félicité',
  age: 10,
  race: 'chat de gouttière',
  poids: 3.5
}
```

Felicité -> Miaou ! Miaou !

Félidés

Félidés

Redéfinition d'une méthode

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids ;
}

Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Miaou ! Miaou !");
};

Chat.prototype.maigrir = function(deltaPoids) {
      this.poids -= deltaPoids;
};

let felicite = new  Chat("Félicité",10,"chat de gouttière", 3.5);
let felix    = new  Chat("Felix",6,"siamois",3);

felicite.miauler();
felix.miauler();
```

Felicité -> Miaou ! Miaou !

Felix -> Miaou ! Miaou !

```
felix.miauler = function () {
    console.log(this.nom + "-> Meaow ! Meaow !");
};
```

redéfinition de la méthode miauler pour felix

```
felix.miauler();
felicite.miauler();
```

Felix -> Meaow ! Meaow !

Felicité -> Miaou ! Miaou !

Une propriété du prototype peut être redéfinie sur une instance.

Dans ce cas la redéfinition ne concerne que l'instance

Modification du prototype du constructeur

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids ;
}

Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Miaou ! Miaou !");
};

Chat.prototype.maigrir = function(deltaPoids) {
     this.poids -= deltaPoids;
};

let felicite = new  Chat("Félicité",10,"chat de gouttière", 3.5);
felicite.miauler();
```

Félicité -> Miaou ! Miaou !

```
Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Meaow ! Meaow !");
};
```

modification de la méthode miauler du prototype du constructeur

```
felicite.miauler();
```

Félicité -> Meaow ! Meaow !

```
let felix  = new  Chat("Felix",6,"siamois",3);
felix.miauler();
```

Felix -> Meaow ! Meaow !

Une modification du prototype est immédiate pour les instances déjà existantes. Le *fallback* se fait à l'exécution (*runtime*) au moment de l'accès à la propriété.

Modification du prototype du constructeur

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids ;
}

Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Miaou ! Miaou !");
};

Chat.prototype.maigrir = function(deltaPoids) {
     this.poids -= deltaPoids;
};

let felicite = new  Chat("Félicité",10,"chat de gouttière", 3.5);
felicite.miauler();
felicite.ronronner();
```

Félicité -> Miaou ! Miaou !

Ce message provoquerait une erreur la méthode n'est pas définie

```
Chat.prototype.ronronner = function () {
    console.log(this.nom + "-> Rrr... Rrr...");
};
```

ajout de la méthode ronronner au prototype du constructeur

```
felicite.ronronner();
```

Félicité -> Rrr... Rrr...

De la même manière qu'une modification, un ajout au prototype est immédiatement actif et s'applique à toutes les instances déjà créées