XNATIS

ORSYS formation

L'héritage

Michael

XNATIS

XNATIS

ORSYS formation

Compétence visée :

Comprendre la factorisation de code grâce à l'héritage

Mettre en œuvre le principe de l'héritage

XNATIS

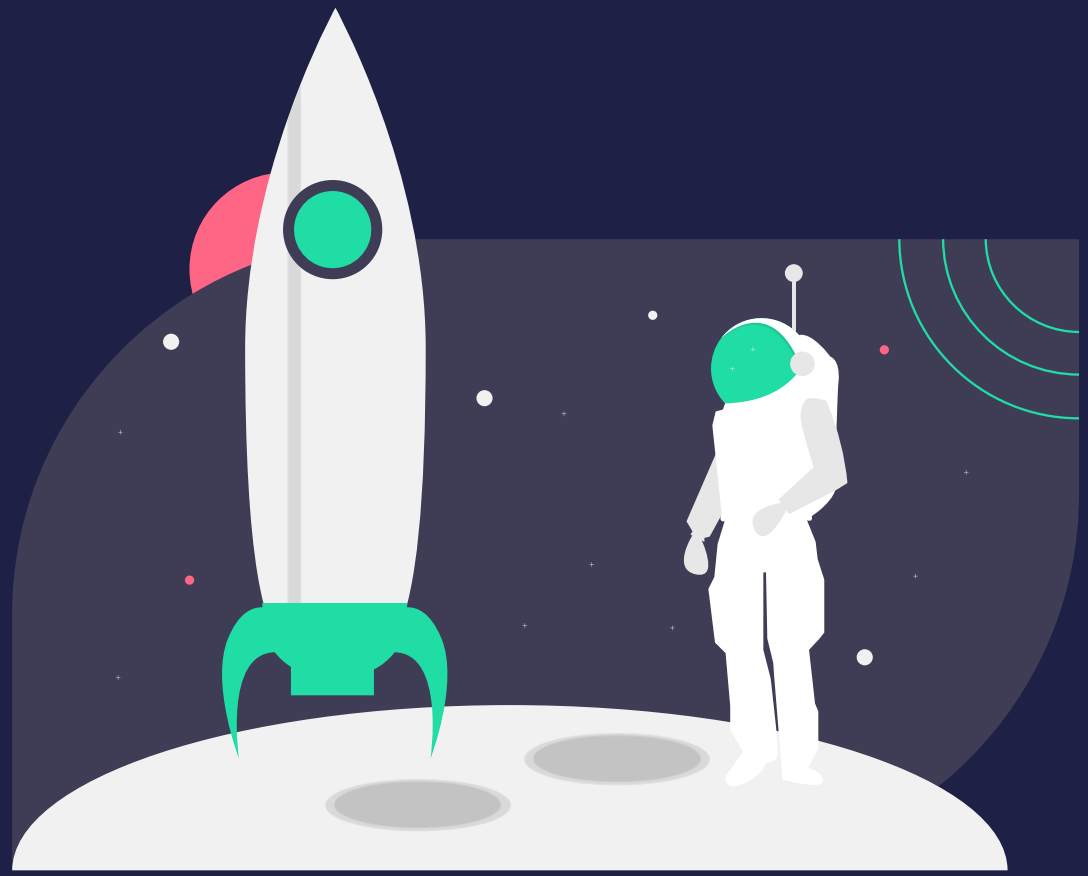
ORSYS formation

Le prototype
1. Prototype et __proto__
2. instanceof
3. Membre statique
4. Pratique (Création d'un singleton)

Cloner un objet
1. Le type Object
2. Spread operator & rest parameters
3. Serialisation & deserialisation
4. Object.assign

Programmation orientée prototypes
1. Définition
2. Héritage par portotype
3. dynamic dispatch

Héritage multiple
1. Définition
2. Mixins
3. Problème du diamant en JS
4. Pratique (namespace proxy avec des mixins)



Le prototype
1. Prototype et __proto__
2. instanceof
3. Membre statique
4. Pratique (Création d'un singleton)

Cloner un objet
1. Le type Object
2. Spread operator & rest parameters
3. Serialisation & deserialisation
4. Object.assign

Programmation orientée prototypes
1. Définition
2. Héritage par portotype
3. dynamic dispatch

Héritage multiple
1. Définition
2. Mixins
3. Problème du diamant en JS
4. Pratique (namespace proxy avec des mixins)

QUESTION à 1 million de faux dollars !

Tout objet possède des membres (propriétés ou méthodes) que l'on ne voit pas par Object.getOwnPropertyNames.

Alors où sont-elles ?

Soit … ce sont des membres :

- Internes : déclarés par des symbols et non accessibles sauf si on connait ces symbols
- Hérités : déclarés dans un autre objet connecté par la propriété __proto__

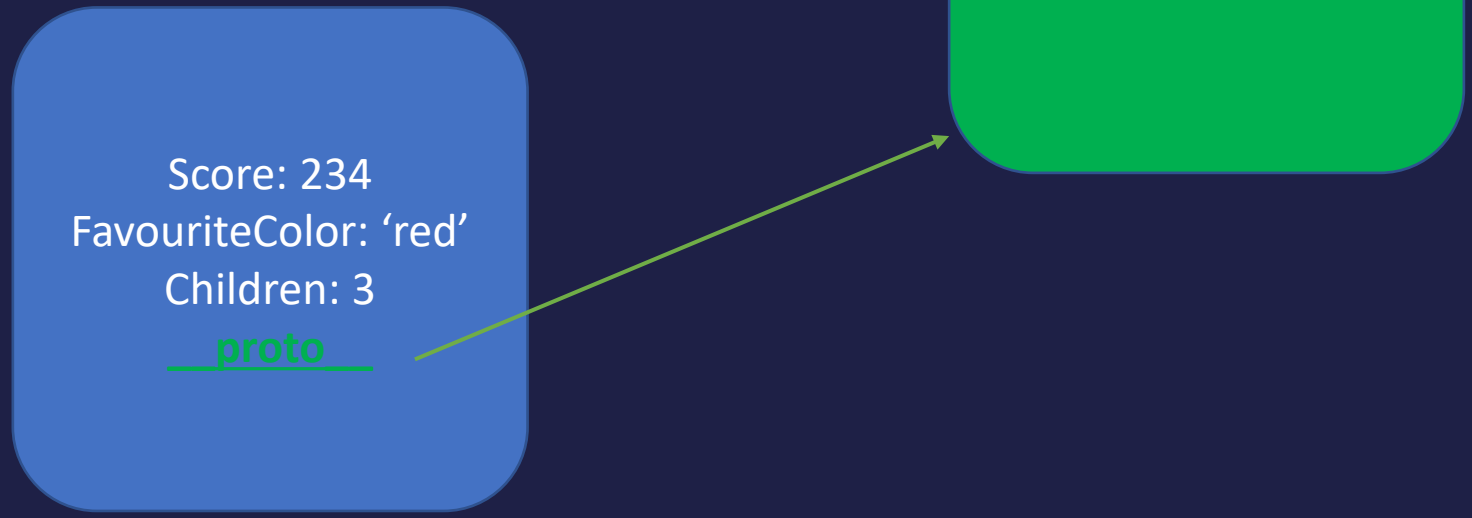
Le prototype
1. Prototype et __proto__
2. instanceof
3. Membre statique
4. Pratique (Création d'un singleton)

Cloner un objet
1. Le type Object
2. Spread operator & rest parameters
3. Serialisation & deserialisation
4. Object.assign

Programmation orientée prototypes
1. Définition
2. Héritage par portotype
3. dynamic dispatch

Héritage multiple
1. Définition
2. Mixins
3. Problème du diamant en JS
4. Pratique (namespace proxy avec des mixins)

Pour chaque objet créé par une fonction constructeur, une association automatique se fait, entre la propriété __proto__ de l'objet et l'objet prototype de la fonction constructeur

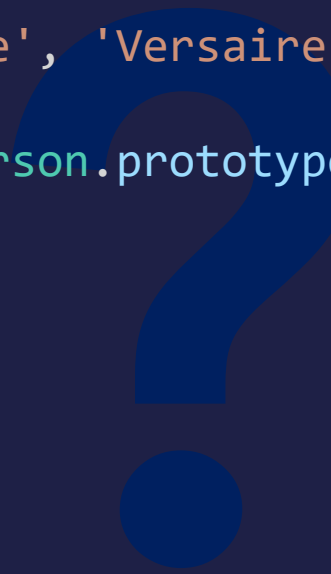

Score: 234
FavouriteColor: ‘red’
Children: 3
__proto__
Age: 16
Firstname: Annie
Lastname: Versaire

XDNATIS

ORSYS formation

```
function Person(age, firstname, lastname) {
    this.age = age;
    this.firstname = firstname;
    this.lastname = lastname;
}
const obj = new Person(16, 'Annie', 'Versaire');

console.log(obj.__proto__ === Person.prototype);
```



XNATIS

ORSYS formation

```
function Person(age, firstname, lastname) {
    this.age = age;
    this.firstname = firstname;
    this.lastname = lastname;
}

Person.prototype = {
    score: 234,
    favouriteColor: 'red'
};

const obj = new Person(16, 'Annie', 'Versaire');
const obj2 = new Person(21, 'Pierre', 'Carre');

console.log(obj.score);
console.log(obj2.score);
```

Le prototype
1. Prototype et __proto__
2. instanceof
3. Membre statique
4. Pratique (Création d'un singleton)

Cloner un objet
1. Le type Object
2. Spread operator & rest parameters
3. Serialisation & deserialisation
4. Object.assign

Programmation orientée prototypes
1. Définition
2. Héritage par portotype
3. dynamic dispatch

Héritage multiple
1. Définition
2. Mixins
3. Problème du diamant en JS
4. Pratique (namespace proxy avec des mixins)

L'opérateur instanceof permet de tester si un objet possède, dans sa chaîne de prototype, la propriété prototype d'un certain constructeur

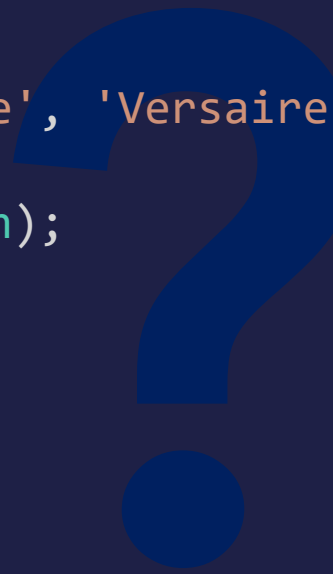
XNATIS

ORSYS formation

```
function Person(age, firstname, lastname) {
    this.age = age;
    this.firstname = firstname;
    this.lastname = lastname;
}

const obj = new Person(16, 'Annie', 'Versaire');

console.log(obj instanceof Person);
```



XNATIS

ORSYS formation

```
function Person(age, firstname, lastname) {
    this.age = age;
    this.firstname = firstname;
    this.lastname = lastname;
}

function Voiture() {

}

const obj = new Person(16, 'Annie', 'Versaire');

obj.__proto__ = Voiture.prototype;

console.log(obj instanceof Voiture);
```

Un membre statique est un membre appartenant au prototype et non à l'objet. Les méthodes statiques sont souvent des fonctions utilitaires car elles ne dépendent d'aucun contexte.

Le prototype
1. Prototype et __proto__
2. instanceof
3. Membre statique
4. Pratique (Création d'un singleton)

Cloner un objet
1. Le type Object
2. Spread operator & rest parameters
3. Serialisation & deserialisation
4. Object.assign

Programmation orientée prototypes
1. Définition
2. Héritage par portotype
3. dynamic dispatch

Héritage multiple
1. Définition
2. Mixins
3. Problème du diamant en JS
4. Pratique (namespace proxy avec des mixins)

PRATIQUE

Les variables auxquelles est attribuée une valeur non primitive reçoivent une référence à cette valeur. Cette référence pointe vers l'emplacement de l'objet en mémoire. Les variables ne contiennent pas réellement la valeur.

On ne peut pas cloner un objet par une simple affectation !

XDNATIS

ORSYS formation

```
const obj = {
    id: 4
};
let copie = obj;
copie.id = 3;
console.log(obj === copie); // prints true
```

XNATIS

ORSYS formation

Comment faire ? ☹

1ère fausse bonne idée : Utiliser new Object

Le constructeur Object new Object crée un wrapper d'objet pour la valeur donnée.

Si la valeur est null ou undefined, il créera et retournera un objet vide, sinon, il retournera un objet du Type qui correspond à la valeur donnée.

Si la valeur est déjà un objet, le constructeur retournera cette valeur.

Ce n'est pas un copy constructor !

XNATIS

ORSYS formation

```
const obj = {};
const clone = new Object(obj);
console.log(obj === clone); // prints true
```

Lorsqu'il n'est pas appelé dans un contexte constructeur, Object() se comporte de façon identique à new Object().

2ème fausse bonne idée : Utiliser le spread operator

La syntaxe de décomposition (spread operator) permet d'étendre un itérable en lieu et place.

Les paramètres du reste (rest parameters) permet de représenter un nombre indéfini d'arguments sous forme d'un tableau.

Utilisons les pour cloner des objets !

XDNATIS

ORSYS formation

```
function Voiture(price) {
    this.price = price;
}

const obj = new Voiture(234);
const copie = {...obj};
console.log(obj instanceof Voiture); // prints true
console.log(copie instanceof Voiture); // prints false
```

3ème fausse bonne idée : Utiliser la sérialisation et la désérialisation

La méthode JSON.stringify() convertit une valeur JavaScript en chaîne JSON.

La méthode JSON.parse() analyse une chaîne de caractères JSON et construit la valeur JavaScript ou l'objet décrit par cette chaîne.

XNATIS

ORSYS formation

```
function Voiture(price) {
    this.price = price;
}
const obj = new Voiture(34);
const copie = JSON.parse(JSON.stringify(obj));
console.log(obj instanceof Voiture); // prints true
console.log(copie instanceof Voiture); // prints false
```

4ème fausse bonne idée : Utiliser Object.assign({}, ...)

La méthode Object.assign() est utilisée afin de copier les valeurs de toutes les propriétés directes (non héritées) d'un objet qui sont énumérables sur un autre objet cible. Cette méthode renvoie l'objet cible.

Cette méthode déclenche les accesseurs (getters) et les mutateurs (setters) des objets passés en argument. Il s'agit donc ni d'une construction réelle, mais bien d'une modification.

XNATIS

ORSYS formation

```
function Voiture(price) {
    this.price = price;
}

const obj = new Voiture(34);
const copie = Object.assign({}, obj);

console.log(obj instanceof Voiture); // prints true
console.log(copie instanceof Voiture); // prints false
```

XNATIS

ORSYS formation

Comment cloner correctement un objet ?

XDNATIS

ORSYS formation

```
function Voiture(price) {
    this.price = price;
}
const obj = new Voiture(34);
const copie = Object.assign(new obj.constructor, obj);
console.log(obj instanceof Voiture); // prints true
console.log(copie instanceof Voiture); // prints true
```

La programmation basée sur les prototypes est un style de programmation orientée objet dans lequel la réutilisation du comportement (appelée héritage) est effectuée via un processus de réutilisation d'objets existants qui servent de prototypes.

Ce modèle peut également être connu sous le nom de programmation prototypique, orientée prototype, sans classe ou basée sur des instances.

En somme :

La programmation basée sur des prototypes utilise des objets généralisés, qui peuvent ensuite être clonés et étendus.

XDNATIS

ORSYS formation

```
const obj = {
    age: 16,
    firstname: 'Annie',
    lastname: 'Versaire'
};
const obj2 = Object.assign(Object.create(obj), {
    score: 234,
    favouriteColor: 'red',
    children: 3
});
console.log(obj2.age);
```

XEBIA NATIS

ORSYS formation

```
const obj = {
    age: 16,
    firstname: 'Annie',
    lastname: 'Versaire'
};
const obj2 = Object.assign(Object.create(obj), {
    score: 234,
    favouriteColor: 'red',
    children: 3
});
console.log(obj2.__proto__ === obj);
```

XNATIS

ORSYS formation

Mais .. Quel est le prototype d'un objet de base ?

XEBIA NATIS

ORSYS formation

```
const obj = {
    age: 16,
    firstname: 'Annie',
    lastname: 'Versaire'
};
console.log(obj.__proto__);
```

?

En informatique, le dynamic dispatch est le processus de sélection de l'implémentation d'une opération (méthode ou fonction) à appeler au moment de l'exécution.

Le prototype
1. Prototype et __proto__
2. instanceof
3. Membre statique
4. Pratique (Création d'un singleton)

Cloner un objet
1. Le type Object
2. Spread operator & rest parameters
3. Serialisation & deserialisation
4. Object.assign

Programmation orientée prototypes
1. Définition
2. Héritage par portotype
3. dynamic dispatch

Héritage multiple
1. Définition
2. Mixins
3. Problème du diamant en JS
4. Pratique (namespace proxy avec des mixins)

L'opération a exécuter est donc ciblé au moment de l'exécution (runtime) en fonction de la chaine de prototypes.

XNATIS

ORSYS formation

```
function Person(age, firstname, lastname) {
    this.age = age;
    this.firstname = firstname;
    this.lastname = lastname;
}
Person.prototype = {
    score: 432
}

function Voiture() {}
Voiture.prototype = {
    score: 234
}

const obj = new Person(16, 'Annie', 'Versaire');
obj.__proto__ = Voiture.prototype;

console.log(obj.score);
```

XNATIS

ORSYS formation

REFLEXION : Quels sont les dangers d’une telle implémentation ?

XDNATIS

ORSYS formation

L'héritage multiple est une caractéristique de certains langages de programmation informatique orientés objet dans lesquels un objet peut hériter des caractéristiques et fonctionnalités de plusieurs objets parent.

Il est différent de l'héritage simple, où un objet ne peut hériter que d'un objet en particulier.

XNATIS

ORSYS formation

REFLEXION : En JS, l'héritage multiple est-il possible ?

En JS, l'héritage multiple est impossible,
car il n'y a qu'un seul __proto__ par objet !

XDNATIS

ORSYS formation

Quelle solution possible ?

Les mixins ! ☺

Le moyen le plus simple d'implémenter un mixin en JavaScript est de créer un objet avec des méthodes utiles, afin que nous puissions facilement les fusionner dans un prototype de n'importe quelle classe.

XNATIS

ORSYS formation

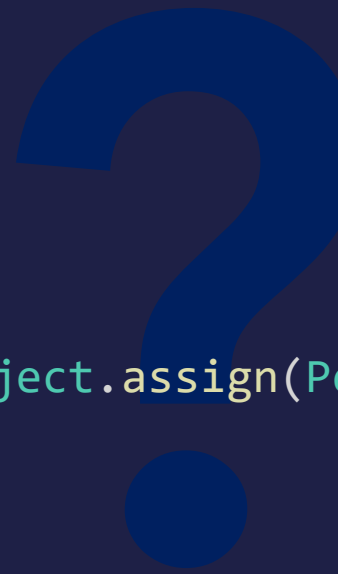
```
let mixin1 = {
    sayHello: function() {
        console.log('hello');
    }
}

let mixin2 = {
    sayBye: function() {
        console.log('bye bye!');
    }
}

function Person(name) {
    this.name = name;
}
[mixin1, mixin2].forEach(x => Object.assign(Person.prototype, x));

obj = new Person('Sophie');

obj.sayHello();
obj.sayBye();
```



XNATIS

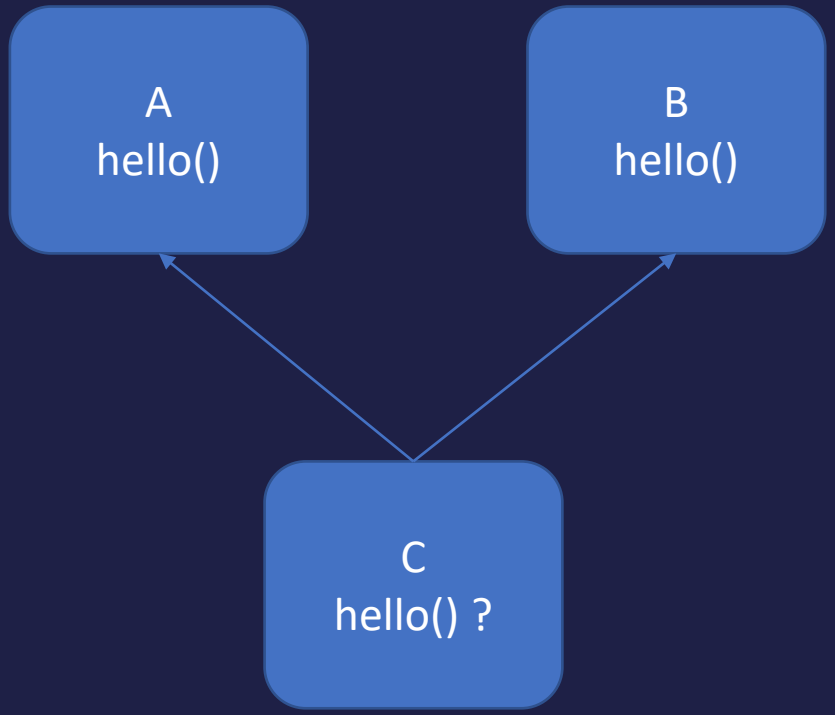
ORSYS formation

Attention, cependant au diamond problem !

XNATIS

ORSYS formation

Si A a une fonction hello
Et B une fonction hello également,
Quelle est la fonction hello de C ?



XEBIA NATIS

ORSYS formation

REFLEXION : En JS, le diamond problem est-il vraiment un problème ?

En JS, le diamond problem ne se pose pas, à cause du shadowing

XNATIS

ORSYS formation

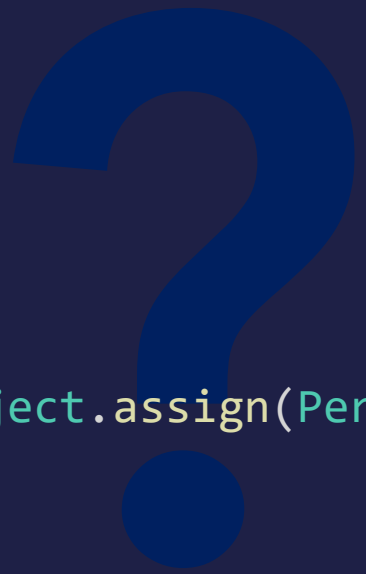
```
let mixin1 = {
    sayHello: function() {
        console.log('hello');
    }
}

let mixin2 = {
    sayHello: function() {
        console.log('bye bye!');
    }
}

function Person(name) {
    this.name = name;
}
[mixin1, mixin2].forEach(x => Object.assign(Person.prototype, x));

obj = new Person('Sophie');

obj.sayHello();
```



Le prototype
1. Prototype et __proto__
2. instanceof
3. Membre statique
4. Pratique (Création d'un singleton)

Cloner un objet
1. Le type Object
2. Spread operator & rest parameters
3. Serialisation & deserialisation
4. Object.assign

Programmation orientée prototypes
1. Définition
2. Héritage par portotype
3. dynamic dispatch

Héritage multiple
1. Définition
2. Mixins
3. Problème du diamant en JS
4. Pratique (namespace proxy avec des mixins)

PRATIQUE

XNATIS

ORSYS formation

Pour aller plus loin

https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/Inheritance