



Exos vus en cours

# Jour1

## **Somme.js**

Calculez la somme de 1 à 20.

## **Unshift.js**

Créez un tableau de 5 éléments de 0 à 4 le tableau contient ([0, 1, ... 4 ]).

Déplacez tous les éléments de 1 case, l'élément 0 se retrouve à la case d'index 1, l'élément 1 se retrouve à la case d'index 2 et ainsi de suite.

Puis ajoutez un élément à la case d'index 0.

# Jour1

## **copie\_tableau.js**

Copiez les éléments d'un tableau dans un autre tableau en utilisant 2 espaces mémoire séparés (lorsque je modifie mon premier tableau, je ne dois pas modifier mon 2ème tableau.

## **fibonacci.js**

Calculez l'index n de la suite de fibonacci. La suite de fibonacci est calculée comme suit.

$\text{Fibonacci}(0) = 0$

$\text{Fibonacci}(1) = 1$

$\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$

[https://fr.wikipedia.org/wiki/Suite\\_de\\_Fibonacci](https://fr.wikipedia.org/wiki/Suite_de_Fibonacci)

# Jour1

## FizzBuzz1.js

Ecrire une fonction `fizzBuzz` qui prend 2 entiers en paramètre 'start' et 'end' et qui pour chaque entier de 'start' à 'end' inclus effectue l'opération suivante

- Si le nombre est divisible par 3, on affiche 'Fizz'.
- Si le nombre est divisible par 5, on affiche Buzz
- Si le nombre est divisible par 3 et par 5, on affiche Fizzbuzz
- Sinon, on affiche le nombre passé en paramètre.

L'affichage dans la console, doit être sous cette forme  
<valeur de n>: <valeur de la fonction `fizzBuzz` pour n>

Pour faire un test, vous pouvez appeler la fonction avec les paramètres suivants  
`fizzBuzz(1, 30);`

# Jour2

## callBack.js

Affichez les éléments du tableau `tableau` en différé. Utilisez la propriété `timeout` de chaque élément pour afficher les éléments à l'écran au bout de `timeout` millisecondes. Par exemple pour le code ci-dessous, affichez le 1ère élément au bout de 730 ms, le 2ème au bout de 1425 ms, ...

```
class Signal {  
  constructor(name, timeout) { this.name = name; this.timeout = timeout; }  
}  
  
const tableau = [  
  new Signal("a", 730),  
  new Signal("b", 1425),  
  new Signal("c", 985),  
  new Signal("d", 3500),  
  new Signal("e", 2400),  
  new Signal("f", 812)  
];
```

# Jour2

## callBind.js

Utilisez la fonction `setTimeout` pour appeler la fonction `afficherNom` de l'instance `test` au bout de 2000 ms.

```
class Test {  
  constructor(nom) { this.nom = nom; }  
  
  afficherNom() {  
    console.log(this.nom);  
  }  
}  
  
let test = new Test("zebre");
```

# Jour2

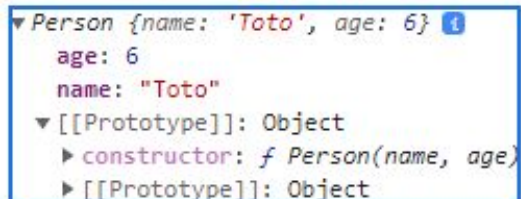
## fakePerson.js

Dans cette exercice, on se propose de simuler la fonction constructrice suivante

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

En utilisant une fonction classique `FakePerson`. Vérifiez bien que le constructeur est bien mis au niveau du Prototype.

Vous devez utiliser les fonctions `Object.create` et `Object.assign`

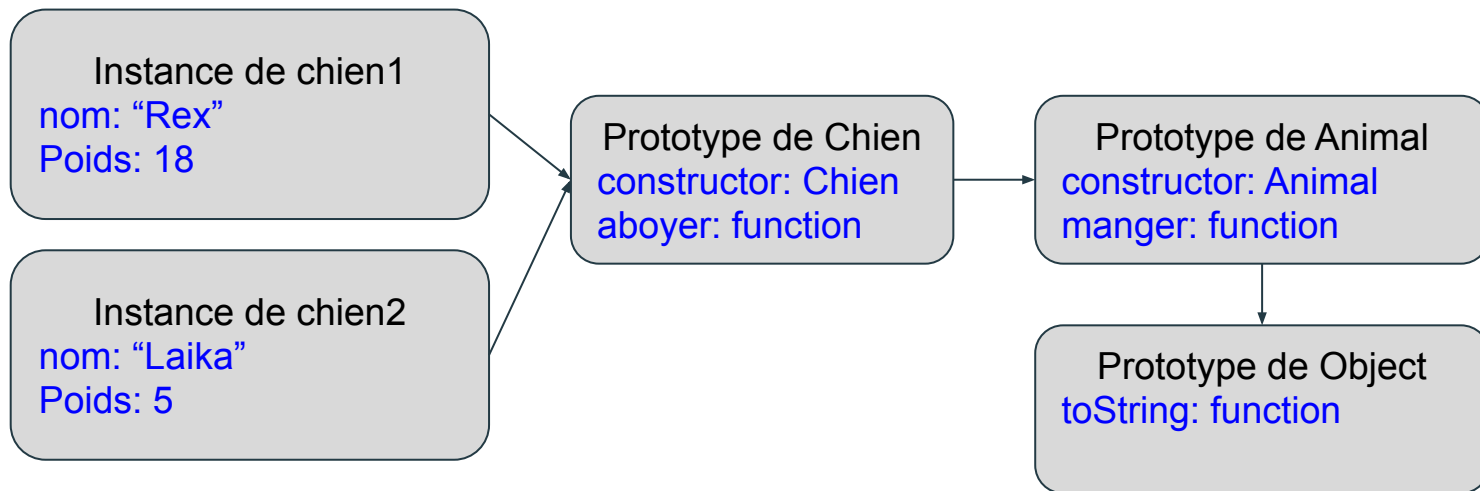


```
▼ Person {name: 'Toto', age: 6} ⓘ  
  age: 6  
  name: "Toto"  
  ▼ [[Prototype]]: Object  
    ► constructor: f Person(name, age)  
    ► [[Prototype]]: Object
```

# Jour 2

## protoInheritance.js & inheritanceClass1.js

Écrivez le code qui crée l'héritage de prototype suivant, puis créez les instances chien1 et chien2.

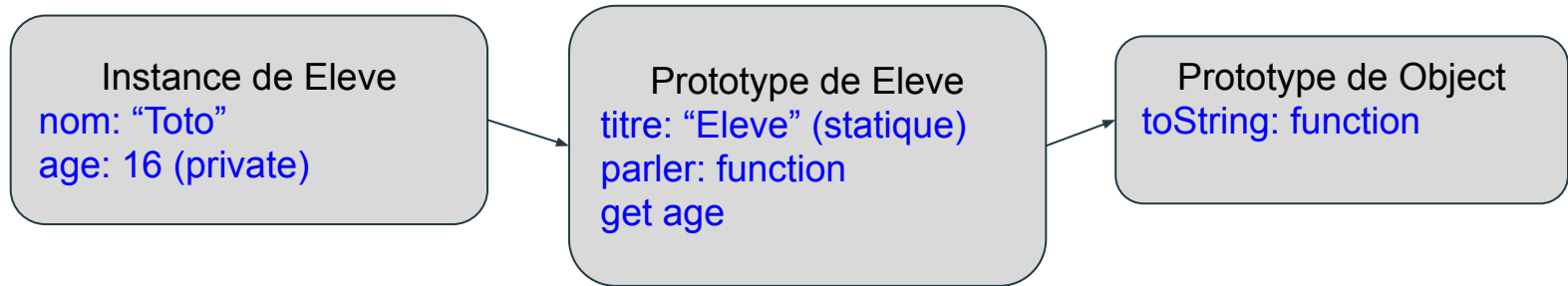




# Jour 2

## inheritanceClass2.js

Simuler en utilisant le mot clé 'class' l'héritage de prototypes suivant



# Jour 2

## mapFilter.js

Créer un tableau de taille 32 dont chaque élément est égale à sa position dans le tableau.

[0,1,2,3,...31]

Pour chaque élément du tableau, on se propose de réaliser les opérations suivantes.

1. Transformer chaque élément du tableau en chaîne de caractère représentant le nombre binaire de l'élément modifié (3 => "11").
2. On concatène "0" à la fin de la chaîne de caractère précédemment obtenue. ("11" => "110").
3. On ne garde que les chaînes de caractère dont la longueur est inférieure ou égale à 4.
4. On transforme toutes les chaînes de caractère représentant un chiffre binaire en nombre. ("110" => 6).

On affiche chaque élément du tableau ainsi obtenu.

Transformation d'un nombre en une chaîne binaire: `num => num.toString(2)`

Transformation d'une chaîne binaire en nombre: `str => parseInt(str, 2)`

# Jour 2

## mapReduce.js

Dans cet exercice on se propose de prendre la chaîne de caractère suivante

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit,  
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam
```

On doit transformer cette phrase en une chaîne de caractères contenant la première lettre de chaque mot. Pour ce faire nous allons utiliser les fonction map & reduce.

Jour3