Initiation aux objets et aux classes

Table des matières

I. Contexte

Durée : 1 h

Environnement de travail : Repl.it

Pré-requis : Bases du JavaScript

Contexte

JavaScript est un langage de programmation de scripts. L'évolution de la norme ECMAScript en 2015 a ajouté au langage plus d'outils pour mettre en place un paradigme orienté objet (OO). JavaScript fait cependant partie de la famille des langages à **prototype**, l'autre famille étant celle des langages à **classes** comme Java et C++, bien que nous verrons que JavaScript a introduit la notion de classes en 2015 avec la version 6 de la norme d'ECMAScript.

Les langages basés sur les classes distinguent celles qui définissent une représentation abstraite d'un objet de la réalité (un plan d'architecte pour un immeuble, par exemple) et une instance de la classe (un des immeubles physiques construit avec ce plan, par exemple). Les langages basés sur les prototypes ne font pas cette distinction, et tous les objets héritent dans leur prototype des méthodes de la classe.

II. Qu'est-ce qu'une classe ?

Objectifs

- Comprendre l'intérêt des classes
- Déclarer une classe
- Instancier un objet

Mise en situation

Tout d'abord, rappelons, de manière succincte et simplifiée, ce qu'est la Programmation Orientée Objet.

Il s'agit d'une vision dans laquelle des éléments du monde réel sont décrits par des objets du monde programmatique. Ainsi, nous pouvons modéliser plus simplement les caractéristiques et les actions réalisables par ces objets.

Fondamental **La composition des objets**

Nous dirons que les objets contiennent des données, les **propriétés**, et qu'ils décrivent un comportement, les **méthodes**, que l'on désire leur appliquer. Ces propriétés et méthodes sont généralement stockées à l'intérieur de l'objet : on dit alors qu'elles sont **encapsulées**.

Tout ceci représente la définition d'un objet, et nous l'appelons **classe**.

Considérons le modèle objet suivant : à partir d'un programme, nous souhaitons afficher des informations concernant des voitures et leur conducteur.

Exemple **Jusqu'en ES5**

```
1 // Fonction décrivant une voiture
2 function Voiture() {
3   this.marque= 'Peugeot'
4 }
5
```

```
// Fonction décrivant un conducteur
function Conducteur() {
  this.name= 'Philippe'
}
```

Jusqu'à la version 5 de la norme ECMAScript, les classes ne sont que de simples fonctions. Elles sont appelées **fonction constructeur**.

Leur syntaxe est la même qu'une fonction JavaScript classique, à la seule différence que leur nom commence par une majuscule.

Exemple **À partir d'ES6**

```
class Voiture {
  constructor() {
    this.marque = 'Peugeot'
  }
}

class Conducteur {
  constructor() {
    this.name= 'Philippe'
  }
}
```

À partir de la version 6 de la norme ECMAScript, le mot-clé **`class`** a été introduit, ainsi que la notion de constructeur, afin de se rapprocher des langages de programmation orientée objet classiques.

Attention toutefois, il ne s'agit que d'une simplification syntaxique, les classes restent des fonctions spéciales.

Méthode **Instancier une classe**

Maintenant que nous avons défini nos classes, comment pouvons-nous les utiliser ? Nous allons les **instancier** grâce au mot-clé **`new`**.

```
let voiture = new Voiture()
let conducteur = new Conducteur()
```

Instancier un objet revient à utiliser le mot-clé `new` suivi du nom de la fonction constructeur que l'on souhaite utiliser, suivi d'éventuels paramètres ou propriétés que nous verrons plus tard.

Pour finir, il est tout à fait possible d'instancier plusieurs fois des objets.

```
let voiture1 = new Voiture()
let voiture2 = new Voiture()
```

Complément **D'autres façons de faire**

Il existe d'autres manières d'instancier un objet, par exemple en utilisant le constructeur `Object()`.

En JavaScript, tous nos objets sont en réalité des **Object**.

```
let personne1 = new Object({
  nom: 'Doe',
  prenom: 'John'
})
```

Dans ce cas, nous ne définissons pas de classe `Personne` : la classe `Object` est native à JavaScript, mais nous pouvons lui passer les paramètres dont nous avons besoin.

Il est également possible d'utiliser la méthode `create()` d'`Object`.

```
let personne1 = new Object({
  nom: 'Doe',
  prenom: 'John'
})

let personne2 = Object.create(personne1)
personne2.nom = 'Doe'
personne2.prenom = 'Jane'
```

Ici, `personne2` possède les mêmes propriétés que `personne1`, et l'on peut associer des valeurs.

Syntaxe **À retenir**

- Pour déclarer une classe en ES5, on utilise des fonctions constructeurs : `function MaClasse(){}`
- En ES6, il est possible utiliser le mot-clé `class` : `class MaClass(){}`, ce qui revient au même
- Les noms des fonctions ou classes doivent commencer par une majuscule
- Pour instancier un nouvel objet, il est nécessaire d'utiliser le mot-clé `new`

III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°1 p.17]

Déclarez à votre tour une classe représentant des humains.

Pour cela, utilisez les deux manières différentes d'écrire une classe (ES5 et ES6) et mettez en place une classe `Personne` et une classe `Humain` que vous instancierez.

Affichez votre instance dans la console.

IV. Les propriétés

Objectifs

- Définir des propriétés
- Accès aux propriétés

Mise en situation

Les **propriétés**, appelées aussi attributs, sont les variables d'un objet. Elles sont définies ou encapsulées dans la classe de l'objet. En JavaScript, nous pouvons ajouter ou supprimer des propriétés n'importe quand pendant l'exécution du programme.

1 https://repl.it/

Méthode **L'accès aux propriétés**

Pour l'accès aux propriétés, nous distinguons deux cas :

- Au niveau de la classe, l'accès se fait grâce au mot-clé **`this`**, qui fait référence à l'objet courant.
- En dehors de la classe, au niveau de l'objet, l'accès à une propriété se fait en appelant la variable dans laquelle il est instancié, suivie de " . " et du nom de la propriété souhaitée.

Exemple **L'objet Voiture en ES5**

```
// Version ES5
function Voiture() {
  this.moteur // Accès à la propriété moteur dans la classe
}

let voiture = new Voiture();
voiture.moteur // Accès à la propriété moteur en dehors de la classe
```

Exemple **L'objet Voiture en ES6**

```
class Voiture{
  constructor() {
    this.moteur
  }
}

let voiture = new Voiture();
voiture.moteur // Accès à la propriété moteur en dehors de la classe
```

Définir des propriétés

À ce niveau, nos propriétés ne possèdent pas de valeurs.

Nous pouvons leur en attribuer une soit de manière statique à l'intérieur de la classe, soit en passant un paramètre à notre fonction constructeur.

Exemple

```
// Version ES5
function Voiture(moteur, roues, carroserie) {
  this.moteur = moteur
  this.roues = roues
  this.carroserie = carroserie
  this.volant = 'sport'
}

let voiture = new Voiture('1.6L', 4, 'Berline')
```

```
// Version ES6
class Voiture{
  constructor(moteur, roues, carroserie) {
    this.moteur = moteur
    this.roues = roues
    this.carroserie = carroserie
    this.volant = 'sport'
  }
}

```

```
let voiture = new Voiture('1.8L', 4, 'Break')
```

Passer des paramètres à notre objet se fait lors de l'instanciation, de la même manière que nous passons des paramètres à une fonction classique.

Ils doivent être ensuite associés aux propriétés souhaitées à l'intérieur de la classe.

Remarque **Le typage des propriétés**

En JavaScript les propriétés ne sont pas typées, contrairement à d'autres langages de programmation orientée objet.

Complément **La propriété prototype**

Le JavaScript objet est basé sur les **prototypes**. Nous n'entrerons pas dans le détail, mais un objet JavaScript possède, comme nous venons de le voir, des propriétés spécifiques ainsi qu'une propriété globale nommée `prototype`[1].

Celle-ci va nous permettre de définir des propriétés partagées entre tous les objets d'une même classe.

Souvenons-nous de la propriété `this.volant` de notre classe `Voiture`. Nous lui avons attribué une valeur directement à l'intérieur de la classe : elle est donc identique à chaque objet.

La propriété prototype peut être utile pour des fonctions de concaténation de chaînes de caractères ou de fonctions liées aux propriétés de l'objet.

Comme dans l'exemple ci-dessous

```
function Voiture(moteur, roues, carroserie) {
  this.moteur = moteur
  this.roues = roues
  this.carroserie = carroserie
  this.vitesse = 0

  // la fonction affiche() est rattachée à l'objet voiture
  // grâce au mot-clé this.
  this.affiche = function() {
    console.log(this.moteur,this.roues, this.carroserie, this.volant)
  }
}

// La propriété volant est partagée entre tous les objets Voiture.
Voiture.prototype.volant = 'Sport'

let voiture1 = new Voiture('1.6L', 4, 'Berline')
let voiture2 = new Voiture('1.8L', 4, 'Break')
voiture1.affiche() // 1.6L 4 Berline Sport
voiture2.affiche() // 1.8L 4 Break Sport


```

Résultat de l'exemple ci-dessus :

On parle dans ce cas précis d'**héritage**.

1 https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Object/prototype

Syntaxe **À retenir**

- À l'intérieur d'une classe, nous déclarons et accédons aux propriétés grâce au mot-clé `this` et on accède à la propriété de cette façon : `this.maPropriete`
- L'utilisation de `this` est importante, car les instances d'une classe disposent des mêmes propriétés : il va donc nous permettre d'utiliser des valeurs propres à chaque instance
- À l'extérieur d'une classe, nous accédons aux propriétés de la manière suivante : `monInstance.maPropriete`

V. Exercice : Appliquez la notion

Vous disposez de la classe `Personne` suivante :

```
// Version ES6
class Personne {
}

```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question 1

[solution n°2 p.17]

Définissez dans le constructeur de la classe les propriétés suivantes : `nom, prenom, age, sexe, adresse`.

Instanciez un nouvel objet et affichez les propriétés de celui-ci.

Question 2

[solution n°3 p.18]

Modifiez les valeurs de votre objet après l'avoir initialisé.

La nouvelle adresse sera "Rue des classes en JavaScript", tandis que l'âge vaudra désormais 41.

VI. Les méthodes

Objectifs

- Définir des méthodes
- Utilisation des méthodes

Mise en situation

À l'instar des propriétés, les méthodes sont aussi des variables d'un objet. Elles sont en tout cas stockées dans des variables, mais définies avec la syntaxe des fonctions. Les méthodes vont nous permettre de décrire le comportement de nos objets ou de définir des actions à réaliser.

1 https://repl.it/

Méthode **Définir et accéder aux méthodes**

Une méthode est encapsulée dans une classe et stockée dans une variable. On peut y accéder dans la définition de la classe avec le mo-clé `this`.

En dehors de la classe, l'utilisation d'une méthode se fait en appelant la variable dans laquelle est stockée notre instance, suivie de "`.`" et du **nom** de la méthode souhaitée, suivi des parenthèses `()`. Il s'agit ici de la syntaxe d'appel d'une fonction classique.

Exemple **Ajoutons des méthodes à notre objet Voiture**

Dans la vie réelle, une voiture est faite pour rouler, et donc accélérer : nous lui permettrons de le faire grâce à une méthode `accelere()`.

Il est tout à fait possible de lui passer des paramètres, une vitesse par exemple. Ajoutons donc à nos objets une propriété `vitesse`.

```
// Version ES5
function Voiture(moteur, roues, carroserie) {
  this.moteur = moteur
  this.roues = roues
  this.carroserie = carroserie
  this.volant = 'sport'
  this.vitesse = 0

  // Méthode permettant à une voiture de rouler
  this.accelere = function(vitesse) {
    this.vitesse += vitesse
    console.log(`Nous roulons à ${this.vitesse} Km/h.`)
  }

  // Méthode permettant d'afficher les caractéristiques d'une voiture
  this.affiche = function() {
    console.log(this.moteur,this.roues, this.carroserie, this.volant)
  }
}


let voiture = new Voiture('1.6L', 4, 'Berline')
voiture.affiche()
voiture.accelere(50)
```

```
// Version ES6
class Voiture{
  constructor(moteur, roues, carroserie) {
    this.moteur = moteur
    this.roues = roues
    this.carroserie = carroserie
    this.volant = 'sport'
    this.vitesse = 0
  }

  accelere(vitesse) {
    this.vitesse += vitesse
    console.log(`Nous roulons à ${this.vitesse} Km/h.`)
  }

  affiche() {
    console.log(this.moteur,this.roues, this.carroserie, this.volant)
```

```
  }
}

let voiture = new Voiture('1.8L', 4, 'Break')
voiture.affiche()
voiture.accelere(80)
```

Pour synthétiser, **les méthodes sont des fonctions qui permettent à un objet de réaliser une action**. Elles peuvent utiliser les propriétés de l'objet en lecture ou en écriture.

Résultat de l'exemple précédent :

Remarque **La syntaxe en ES6**

En ES6, la syntaxe de définition d'une méthode diffère un peu. Il est possible de nommer directement une méthode sans l'associer à une variable dans la classe et s'affranchir de l'utilisation de `this`.

Complément **Définir ses méthodes dans un prototype**

Si l'on crée deux objets :

- `voiture1 = new Voiture('1.6L', 4, 'Berline')`
- `voiture2 = new Voiture('1.8L', 4, 'Break')`

Ceux-ci contiennent les propriétés et les méthodes suivantes :

```
// Objet voiture1
{
  moteur: '1.6L',
  roues: 4,
  carroserie: 'Berline',
  volant: 'Sport',
  vitesse: 0,

  accelere: function(vitesse) {
    this.vitesse = vitesse
    console.log(`Nous roulons à ${this.vitesse} Km/h.`)
  }
}

//Objet voiture2
{
  moteur: '1.8L',
  roues: 4,
  carroserie: 'Break',
  volant: 'Sport',
  vitesse: 0,

  accelere: function(vitesse) {
    this.vitesse = vitesse
    console.log(`Nous roulons à ${this.vitesse} Km/h.`)
  }
}
```

Contrairement aux propriétés, les méthodes sont identiques pour chaque objet. Pourtant, en l'état, on constate qu'elles sont redéfinies à chaque construction d'objet.

Pour remédier à cela, il est possible de définir une méthode directement au sein du `prototype`.

```
Voiture.prototype.accelere = function(vitesse) {
  this.vitesse = vitesse
  console.log(`Nous roulons à ${this.vitesse} Km/h.`)
}
```

Syntaxe **À retenir**

- En ES5, une méthode est définie comme une fonction anonyme stockée dans une variable `this.maMethode = function() {}`
- En ES6, nous pouvons nommer directement notre méthode dans notre classe `maMethode() {}`
- Depuis un objet, une méthode est appelée ainsi : `monObjet.maMethode()`

VII. Exercice : Appliquez la notion

Vous disposez de la classe `Personne` suivante :

```
// Version ES6
class Personne {
  constructor(nom, prenom, age, sexe, adresse) {
    this.nom = nom
    this.prenom = prenom
    this.age = age
    this.sexe = sexe
    this.adresse = adresse
  }
}

```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question 1

[solution n°4 p.18]

Créez la méthode `identite()`, qui affichera dans la console la première lettre du prénom, suivie du nom.

Indice :

Le code suivant vous permettra de valider la bonne exécution des éléments mis en place :

```

let personne1 = new Personne('Doe', 'John', 31, 'M', 'Rue du JavaScript')
let personne2 = new Personne('Doe', 'Ann', 28, 'F', 'Avenue des classes et objets')

personne1.identite()
personne2.identite()
```

1 https://repl.it/

Question 2

[solution n°5 p.18]

Créez la méthode `gender()`, qui affichera dans la console "Monsieur" ou "Madame", suivie du nom.

Indice :

Le code suivant vous permettra de valider la bonne exécution des éléments mis en place :

```
1
2 let personne1 = new Personne('Doe', 'John', 31, 'M', 'Rue du JavaScript')
3 let personne2 = new Personne('Doe', 'Ann', 28, 'F', 'Avenue des classes et objets')
4
5 personne1.gender()
6 personne2.gender()
```

Question 3

[solution n°6 p.19]

Créez la méthode `changeAge()`, qui permettra de modifier l'âge d'une personne. Cette dernière méthode sera instanciée dans un prototype.

Indice :

Le code suivant vous permettra de valider la bonne exécution des éléments mis en place :

```
1
2 let personne1 = new Personne('Doe', 'John', 31, 'M', 'Rue du JavaScript')
3 let personne2 = new Personne('Doe', 'Ann', 28, 'F', 'Avenue des classes et objets')
4
5 personne1.changeAge(41)
```

VIII. Essentiel

IX. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°7 p.19]

Exercice

À l'intérieur d'une classe, on peut retrouver...

- ☐ Des propriétés
- ☐ Des méthodes
- ☐ Des événements

Exercice

Le mot-clé `class` a été introduit avec la norme...

- ○ ECMAScript 5
- ○ ECMAScript 6
- ○ Il n'existe pas

Exercice

Le nom d'une classe doit toujours commencer par...

- ○ Une minuscule
- ○ Une majuscule
- ○ Un underscore

Exercice

Quel mot-clé permet d'accéder à une propriété à l'intérieur d'une classe ?

- ○ new
- ○ Object
- ○ this
- ○ prototype

Exercice

En JavaScript, les propriétés d'un objet sont des éléments...

- ○ Typés
- ○ Non typés

Exercice 7

[solution n°8 p.20]

Exercice

Parmi ces syntaxes, lesquelles sont correctes pour instancier un objet ?

- ☐ let reponse = Reponse()
- ☐ let reponse = new Reponse()
- ☐ let reponse = new Object()
- ☐ let reponse2 = Object.create(reponse1)

Exercice

Parmi ces classes, laquelle n'est pas correctement définie ?

- ○
```
function Reponse(prop) {
  this.prop = prop
}
```
- ○
```
class Reponse(prop) {
  constructor(prop) {
    this.prop = prop
  }
}
```
- ○
```
class Reponse {
  constructor(prop) {
    this.prop = prop
  }
}
```

Exercice

Comment pourrait-on créer une propriété partagée à l'ensemble des objets `Reponse` ?

- Reponse.prop = 'ma réponse'
- Reponse.prototype.prop = 'ma réponse'
- Reponse.this.prop = 'ma réponse'

Exercice

Parmi ces appels, lesquels sont des appels de méthodes corrects ?

- [ ] this.afficheReponse()
- [ ] reponse.afficheReponse
- [ ] reponse.changeReponse(2)
- [ ] reponse->changeReponse(2)

Exercice

Instanciez simplement un objet de cette classe avec le mot-clé `new`.

```
class Customer {
  constructor() {
  }
}
```

B. Exercice : Défi

Pour cet exercice final, vous allez devoir créer un petit jeu mettant en scène trois joueurs.

Chaque joueur possède un capital de départ de points, le but étant de faire perdre des points à ses adversaires en les attaquant. Le joueur gardant le plus de points à la fin de la partie gagne.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question 1

[solution n°9 p.21]

Procédons par étapes.

Dans un premier temps, implémentez deux classes, que vous intitulerez `Partie` et `Joueur`.

Un `Joueur` dispose d'un `nom`, d'un `prénom`, d'un `nombre de points` valant `100` par défaut. On doit également pouvoir connaître son `tour` de jeu, valant 0 par défaut.

Une `Partie` comporte des joueurs, un `joueurVainqueur` que l'on ne connaît pas par défaut, et d'un `tour`.

Initialisez une partie, ainsi que 3 nouveaux joueurs.

1 https://repl.it/

Question 2

[solution n°10 p.22]

Mettons en place les éléments suivants :

Modifiez la classe `Joueur` afin d'implémenter les méthodes suivantes :

- Une méthode permettant de retourner l'identité du joueur (son nom et son prénom).
- Une méthode permettant d'afficher les points du joueur, qui appellera la méthode précédente et affichera le nombre de points du joueur.
- Une méthode permettant à un joueur d'attaquer : une attaque peut faire perdre aléatoirement entre 1 et 50 points à un adversaire. Si l'adversaire n'a plus de points, son total ne peut être négatif, il vaudra simplement zéro. Un joueur dispose de 3 tours, chaque attaque augmente le nombre de tours consommés.

Vous disposez de la méthode suivante :

```
rand(nb) {
  return Math.floor(Math.random() * Math.floor(nb));
}
```

Question 3

[solution n°11 p.22]

Considérons désormais ces éléments supplémentaires :

Définissez, pour chaque joueur, une méthode lui permettant d'effectuer une "super attaque". Une super attaque est une attaque faisant perdre 10 points de plus.

Modifiez la méthode `attaque()` afin qu'elle accepte un second paramètre "bonus". Le bonus sera cumulé à la valeur initiale de l'attaque.

Question 4

[solution n°12 p.23]

Il est temps d'implémenter la méthode permettant d'indiquer le vainqueur pour une partie et de tester le bon déroulement de notre jeu

Écrivez une méthode `vainqueur()` qui parcourra l'ensemble des joueurs de la partie afin de déterminer lequel dispose du plus grand nombre de points. Affichez alors l'identité du vainqueur !

Voici le déroulement de la partie, on considère qu'un joueur peut effectuer une attaque et une super attaque lors de chaque tour.

```
//Déroulement de la partie
for (let tour=0; tour<partie.tour; tour++) {
  joueur1.attaque(joueur3)
  joueur1.superAttaque(joueur2)

  joueur2.attaque(joueur1)
  joueur2.superAttaque(joueur3)

  joueur3.attaque(joueur2)
  joueur3.superAttaque(joueur1)
}

joueur1.affichePoints()
joueur2.affichePoints()
joueur3.affichePoints()

partie.vainqueur()
```

Solutions des exercices

p. 5 Solution n°1

```
// Version ES5
      function Personne(nom, prenom, age, sexe, adresse) {

       this.nom = nom
       this.prenom = prenom
       this.age = age
       this.sexe = sexe
       this.adresse = adresse


   }

// Version ES6
   class Humain {
     constructor(nom, prenom, age, sexe, adresse) {
       this.nom = nom
       this.prenom = prenom
       this.age = age
       this.sexe = sexe
       this.adresse = adresse

     }


   }

   let personne1 = new Personne()
   console.log(personne1)

   let personne2 = new Humain()
   console.log(personne2)
```

p. 8 Solution n°2

```
// Version ES6
class Personne {
  constructor(nom, prenom, age, sexe, adresse) {
    this.nom = nom
    this.prenom = prenom
    this.age = age
    this.sexe = sexe
    this.adresse = adresse
  }
}

let personne = new Personne('Doe', 'John', 31, 'M', 'Rue du JavaScript')

console.log(personne.nom, personne.prenom, personne.age, personne.sexe, personne.adresse)

```

p. 8 Solution n°3

```
personne.age = 41
personne.adresse = 'Rue des classes en JavaScript'

console.log(personne.nom, personne.prenom, personne.age, personne.sexe, personne.adresse)
```

p. 11 Solution n°4

```
// Version ES6
class Personne {
  constructor(nom, prenom, age, sexe, adresse) {
    this.nom = nom
    this.prenom = prenom
    this.age = age
    this.sexe = sexe
    this.adresse = adresse
  }

  identite() {
    let identite = `${this.prenom.substring(0,1)}. ${this.nom}`
    console.log(identite)
  }
}


let personne1 = new Personne('Doe', 'John', 31, 'M', 'Rue du JavaScript')
let personne2 = new Personne('Doe', 'Ann', 28, 'F', 'Avenue des classes et objets')

personne1.identite() // J. Doe
personne2.identite()// A Doe

```

p. 12 Solution n°5

```
// Version ES6
class Personne {
  constructor(nom, prenom, age, sexe, adresse) {
    this.nom = nom
    this.prenom = prenom
    this.age = age
    this.sexe = sexe
    this.adresse = adresse
  }

  gender() {
    if (this.sexe === 'M') {
      console.log(`Monsieur ${this.nom}`)
      }else {
        console.log(`Madame ${this.nom}`)
    }

}


```

```
let personne1 = new Personne('Doe', 'John', 31, 'M', 'Rue du JavaScript')
let personne2 = new Personne('Doe', 'Ann', 28, 'F', 'Avenue des classes et objets')

personne1.gender()// Monsieur Doe
personne2.gender() // Madame Doe
```

p. 12 Solution n°6

```
// Version ES6
class Personne {
  constructor(nom, prenom, age, sexe, adresse) {
    this.nom = nom
    this.prenom = prenom
    this.age = age
    this.sexe = sexe
    this.adresse = adresse
  }
}

Personne.prototype.changeAge = function(age) {
  this.age = age
    console.log(`${this.prenom} a ${this.age} ans`)
}

let personne1 = new Personne('Doe', 'John', 31, 'M', 'Rue du JavaScript')
let personne2 = new Personne('Doe', 'Ann', 28, 'F', 'Avenue des classes et objets')


personne1.changeAge(41)
```

Exercice p. 12 Solution n°7

Exercice

À l'intérieur d'une classe, on peut retrouver...

- [x] Des propriétés
- [x] Des méthodes
- [ ] Des événements

Exercice

Le mot-clé `class` a été introduit avec la norme...

- ○ ECMAScript 5
- ◉ ECMAScript 6
- ○ Il n'existe pas

Exercice

Le nom d'une classe doit toujours commencer par...

- ○ Une minuscule
- ◉ Une majuscule
- ○ Un underscore

Exercice

Quel mot-clé permet d'accéder à une propriété à l'intérieur d'une classe ?

- ○ new
- ○ Object
- ◉ this
- ○ prototype

Exercice

En JavaScript, les propriétés d'un objet sont des éléments...

- ○ Typés
- ◉ Non typés

Exercice p. 13 **Solution n°8**

Exercice

Parmi ces syntaxes, lesquelles sont correctes pour instancier un objet ?

- ☐ let reponse = Reponse()
- ☑ let reponse = new Reponse()
- ☑ let reponse = new Object()
- ☑ let reponse2 = Object.create(reponse1)

Exercice

Parmi ces classes, laquelle n'est pas correctement définie ?

- ○
```
function Reponse(prop) {
 this.prop = prop
}
```
- ◉
```
class Reponse(prop) {
 constructor(prop) {
  this.prop = prop
 }
}
```
- ○
```
class Reponse {
 constructor(prop) {
  this.prop = prop
 }
}
```

Exercice

Comment pourrait-on créer une propriété partagée à l'ensemble des objets `Reponse` ?

- ○ Reponse.prop = 'ma réponse'
- ◉ Reponse.prototype.prop = 'ma réponse'
- ○ Reponse.this.prop = 'ma réponse'

Exercice

Parmi ces appels, lesquels sont des appels de méthodes corrects ?

- ☑ this.afficheReponse()
  Correct dans le contexte d'une classe.
- ☐ reponse.afficheReponse
- ☑ reponse.changeReponse(2)
- ☐ reponse->changeReponse(2)

Exercice

Instanciez simplement un objet de cette classe avec le mot-clé `new`.

```
class Customer {
  constructor() {
  }
}
```

new Customer()

p. 14

Solution n°9

```
class Partie {
  constructor(tour, joueurs) {
    this.tour = tour
    this.joueurs = joueurs
    this.joueurVainqueur = null
  }
}

class Joueur {
  constructor(nom, prenom, points, tour) {
    this.nom = nom
    this.prenom = prenom
    this.points = 100
    this.tour = 0
  }
}

let joueur1 = new Joueur('Joueur', '1', 100, 0)
let joueur2 = new Joueur('Joueur', '2', 100, 0)
let joueur3 = new Joueur('Joueur', '3', 100, 0)

let partie = new Partie(3, [joueur1, joueur2, joueur3])
```

p. 15 Solution n°10

```
class Joueur {
  constructor(nom, prenom, points, tour) {
    this.nom = nom
    this.prenom = prenom
    this.points = 100
    this.tour = 0
  }

  rand(nb) {
    return Math.floor(Math.random() * Math.floor(nb));
  }

  attaque(adversaire) {
    if (this.tour < 3) {
      adversaire.points -= this.rand(51)
      if (adversaire.points < 0) {
        adversaire.points = 0
      }
    adversaire.affichePoints()
      ++this.tour
    }
  }

  identite() {
    return `${this.nom} ${this.prenom}`
  }

  affichePoints() {
    console.log(`${this.identite()} possède ${this.points} points`)
  }

}
```

p. 15 Solution n°11

```
Joueur.prototype.superAttaque = function(adversaire) {
  this.attaque(adversaire, 10)
}

class Joueur {
  // ...
  attaque(adversaire, bonus = 0) {
    if (this.tour < 3) {
      adversaire.points -= this.rand(51) + bonus
      if (adversaire.points < 0) {
        adversaire.points = 0
      }
    adversaire.affichePoints()
      ++this.tour
    }
  }
  // ...
}

```

p. 15 **Solution n°12**

Vous avez dû écrire la méthode suivante :

```
class Partie {
  // ...
  vainqueur() {
    let pts = 0
    this.joueurs.forEach(joueur => {
      this.joueurVainqueur = joueur.points > pts ? joueur : this.joueurVainqueur
      pts = joueur.points > pts ? joueur.points : pts
    })

    if (this.joueurVainqueur === null) {
      console.log('Oups personne n\'a gagné !')
      return;
    }

    console.log(`${this.joueurVainqueur.identite()} gagne la partie avec ${this.joueurVainqueur.points} points`)
  }
}
```

Et voici le code complet :

```
class Partie {
  constructor(tour, joueurs) {
    this.tour = tour
    this.joueurs = joueurs
    this.joueurVainqueur = null
  }

  vainqueur() {
    let pts = 0
    this.joueurs.forEach(joueur => {
      this.joueurVainqueur = joueur.points > pts ? joueur : this.joueurVainqueur
      pts = joueur.points > pts ? joueur.points : pts
    })

    if (this.joueurVainqueur === null) {
      console.log('Oups personne n\'a gagné !')
      return;
    }

    console.log(`${this.joueurVainqueur.identite()} gagne la partie avec ${this.joueurVainqueur.points} points`)
  }
}

class Joueur {
  constructor(nom, prenom, points, tour) {
    this.nom = nom
    this.prenom = prenom
    this.points = 100
    this.tour = 0
  }

  rand(nb) {
    return Math.floor(Math.random() * Math.floor(nb));
  }
```

```
35
36    attaque(adversaire, bonus = 0) {
37      if (this.tour < 3) {
38        adversaire.points -= this.rand(51) + bonus
39        if (adversaire.points < 0) {
40          adversaire.points = 0
41        }
42      adversaire.affichePoints()
43        ++this.tour
44      }
45    }
46
47    identite() {
48      return `${this.nom} ${this.prenom}`
49    }
50
51    affichePoints() {
52      console.log(`${this.identite()} possède ${this.points} points`)
53    }
54
55  }
56
57  Joueur.prototype.superAttaque = function(adversaire) {
58    this.attaque(adversaire, 10)
59  }
60
61  let joueur1 = new Joueur('Joueur', '1', 100, 0)
62  let joueur2 = new Joueur('Joueur', '2', 100, 0)
63  let joueur3 = new Joueur('Joueur', '3', 100, 0)
64
65  let partie = new Partie(3, [joueur1, joueur2, joueur3])
66
67  //Déroulement de la partie
68  for (let tour=0; tour<partie.tour; tour++) {
69    joueur1.attaque(joueur3)
70    joueur1.superAttaque(joueur2)
71
72    joueur2.attaque(joueur1)
73    joueur2.superAttaque(joueur3)
74
75    joueur3.attaque(joueur2)
76    joueur3.superAttaque(joueur1)
77  }
78
79  joueur1.affichePoints()
80  joueur2.affichePoints()
81  joueur3.affichePoints()
82
83  partie.vainqueur()
84
85  /* Voici ce qu'on peut trouver dans la console :
86  Joueur 3 possède 65 points
87  Joueur 2 possède 42 points
88  Joueur 1 possède 76 points
89  Joueur 3 possède 7 points
90  Joueur 2 possède 40 points
91  Joueur 1 possède 59 points
92  Joueur 3 possède 2 points
```

```
Joueur 1 possède 36 points
Joueur 2 possède 40 points
Joueur 1 possède 36 points
Joueur 2 possède 40 points
Joueur 3 possède 2 points
Joueur 2 gagne la partie avec 40 points
*/

```