TP RPG Textuel

Objectif

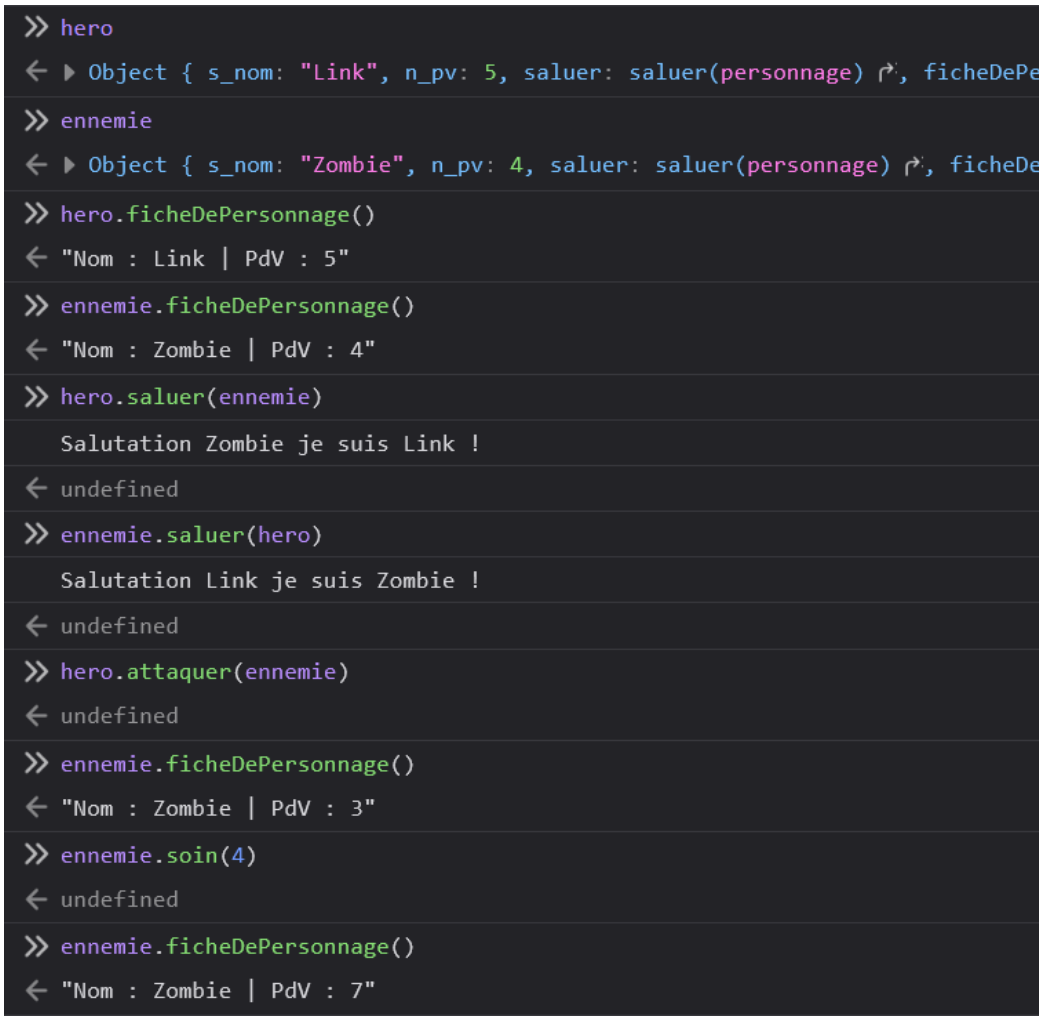
- Créer et Manipuler des objets.
- Créer des méthodes et des attributs d'objet
- Passage par référence.

Prérequis

- Déclarer et appelé une fonction
- Type de donnée

Résultat final

Au final j'aurais crée deux objets : *hero* et *ennemie* qui seront capable de saluer, attaquer, se soigner et fournir une ficher de personnage sous la forme d'une String.



Mise en place des fichiers nécessaires

Dans un dossier nommé *rpg_textuel*, créez un fichier *index.html* et *style.css* contenant les codes suivant.

index.html
```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

    <script src="script.js"></script>
</body>
</html>
```

script.js
```
console.log("Le script fonctionne.");
```

Si le message s'affiche correctement dans la console à l'ouverture de la page *index.html* vous pouvez avancer.

Déclarer l'objet *hero*

Dans le fichier *script.js*, déclarez un objet possédant 2 attributs *s_nom*[1] et *n_pv*.

script.js
```
let hero = {
      s_nom : "Link",
      n_pv : 10
} ;
```

> *Les variables contenu dans l'objet sont appelés "attributs" de l'objet. Les fonctions contenu dans un objet sont appelés « méthodes » de l'objet.*
>
> *La déclaration d'un objet se fait grâce au accolades qui vont contenir les attributs et les méthodes séparés par des virgules.*

Syntaxe de la déclaration d'un objet :

```
let identifiant = {
      attribut1 : value,
      attribut2 : value,
      méthode(param, ...){
            instructions
      }
};
```

Évaluez un objet et ses attributs

Évaluez les instructions suivante dans un *console.log_* et observé le résultat.

console
```
hero          // Décrit en détail l'objet dans la conseil
hero.n_pv     // => 10 : Number
```

1 Pour vous aidez à vous souvenir du type de vos variables, je précises les types **n_**variable pour les Number et **s_**variable pour les String.

```
hero.s_nom   // => « Link » : String
```

La méthode *saluer*

Déclarer la fonction *saluer* dans l'objet *hero*.

script.js
```
let hero = {
	s_nom : "Link",
	n_pv : 10,
	saluer(){
		console.log("Salutation je suis Link ! ");
	}
} ;
```

console
```
hero.saluer()	// => Salutation je suis Link !
```

Problématique

> *Le nom (Link) du hero est écrit en dur dans la fonction saluer, ce qui signifie que si la valeur de l'attribut s_nom change (*hero.s_nom = "Mario") *la méthode saluer sera erroné.*

Solution

script.js
```
let hero = {
	s_nom : "Link",
	n_pv : 10,
	saluer(){
		console.log("Salutation je suis "+ this.s_nom +" ! ");
	}
} ;
```

Après avoir corrigé le script, évaluez les instructions suivante dans un *console.log* et observez le résultat.

console
```
hero.s_nom = "Mario";
hero.saluer();
```

La variable *this*

> *A l'intérieur d'une fonction la variable this signifie « lui-même », pour comprendre remplacez dans votre esprit this par hero.*
>
> *this est obligatoire car sans elle la méthode saluer chercherait une variable global nommé s_nom déclarée quelque part en dehors de l'objet alors que nous parlons de l'attribut s_nom de l'objet.*

La méthode *ficheDePersonnage*

La méthode *ficherDePersonnage* permet de présenter le personnage.

```
script.js > hero
ficheDePersonnage(){
	// codez ici
}
```

Application :

Dans l'objet *hero*, déclarez une nouvelle méthode nommé *ficheDePersonnage* telle que :

- la méthode na pas de paramètres
- la méthode renvoi une string
- la string attendu ressemblera à ça pour un *hero* nommé Link à 5 point de vie
    - « Nom : Link | PdV : 5 »

Le résulat attendu sera :

```
console
>> console.log(hero.ficheDePersonnage())
<- « Nom : Link | PdV : 5 »
```

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution à la méthode *ficheDePersonnage*

script.js

```
let hero = {
	s_nom : "Link",
	n_pv : 10,
	saluer(){
		console.log("Salutation je suis "+ this.s_nom +" ! ");
	},
	ficheDePersonnage(){
		return "Nom : "+this.s_nom+" | PdV : "+this.n_pv;
	}
} ;
```

L'utilisation d'une structured string est possible et même recommandée dans ce genre de cas.

La méthode *soin*

La méthode soin permet d'augmenter le nom de point de vie du hero.

script.js > hero

```
soin(n_montantDuSoin){
	// codez ici
}
```

Application :

Dans l'objet *hero*, déclarez une nouvelle méthode nommé *soin* telle que :

- la méthode a un paramètre *n_montantDuSoin*, c'est un *Number* qui défini le nombre de point de vie que le personnage va récupérer.
- la méthode ne renvoi rien (donc *undefined* par défaut).

Le résulat attendu sera :

console

```
>> hero.n_pv
<- 5
>> hero.soin(15)
<- undefined
>> hero.n_pv
<- 20
```

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution à la méthode *soin*

script.js

```
let hero = {
	s_nom : "Link",
	n_pv : 10,
	saluer(){
		console.log("Salutation je suis "+ this.s_nom +" ! ");
	},
	ficheDePersonnage(){
		return "Nom : "+this.s_nom+" | PdV : "+this.n_pv;
	},
	soin(n_montantDuSoin){
		this.n_pv = this.n_pv + n_montantDuSoin;
		// équivalent à :
		// this.n_pv += n_montantDuSoin;
	},
};
```

Déclaration de l'objet ennemie

> *L'objet ennemie est similaire à l'objet hero à la différence du nom et du nombre de point de vie de départ*

script.js

```
let ennemie = {
	// coder ici
};
```

Application :

Déclarer un nouvelle objet nommé ennemie, il doit avoir les mêmes fonctionnalités que l'objet *hero*. Cependant son nom et ses points de vie doivent être différent du *hero*.

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution à la déclaration de l'objet *ennemie*

script.js
```
let ennemie = {
	s_nom : "Zombie",
	n_pv : 4,
	saluer(){
		console.log("Salutation je suis "+ this.s_nom +" ! ");
	},
	ficheDePersonnage(){
		return "Nom : "+this.s_nom+" | PdV : "+this.n_pv;
	},
	soin(n_soin){
		this.n_pv = this.n_pv + n_soin;
	}
};
```

Modifier la méthode saluer

> *La méthode saluer permet au héros de donner son nom. Nous voulez maintenant que le hero salut l'ennemie et se présente. Pour ceci il nous faut accéder au nom de l'ennemie depuis la fonction saluer du héro.*

La méthode saluer actuel :

script.js > hero
```
saluer(){
	console.log("Salutation je suis "+ this.s_nom +" ! ");
}
```
console
```
>> hero.saluer()
<- Salutation je suis Link !
```

La méthode saluer après modification
script.js > hero
```
saluer(autrui){
	console.log("Salutation "+autrui.s_nom+" je suis "+ this.s_nom +" ! ");
}
```
console
```
>> hero.saluer(ennemie);
<- Salutation Zombie je suis Link !
```

Le passage par valeur d'une variable

En JavaScript il est possible de passer en paramètre d'une fonction la valeur d'une variable.

console
```
>> let eleve = "Maxime";
<- string
>> console.log(eleve);
<- Maxime
```
Ici la fonction log à reçu la valeur de la variable eleve via un *passage par valeur*.

Le passage par référence d'un objet

Il est impossible de passer par valeur un objet car cette opération n'aurait pas de sens, un objet n'a pas de « *valeur* ». C'est un ensemble de plusieurs attributs et méthode ce qui est plus complexe qu'une simple variable contenant une valeur.

Si je veut pouvoir accéder au attribut d'un objet dans une fonction il va falloir que je passe la référence de l'objet dans la fonction. La référence d'une objet est son adresse dans la mémoire, voyez cela comme un raccourci. Si je connais la référence d'un objet je peut m'en servir pour accéder aux attributs et méthodes de l'objet.

Démonstration :

```
let adidas = {
       n_prix: 70,
       s_nom: «Superstar »,
       s_marque : «Adidas»
} ;
function convertirEnConverse(objet){
       // objet contient la reference de l'objet passer en parmètre
       // Dans les lignes suivantes on découvre que objet va être la reference de l'objet
       // adidias
       console.log(objet.n_prix);
       objet.s_nom = "Classic";
       objet.s_marque = "Converse";
}

convertirEnConverse(adidas); // Passage par reference de adidas dans la variable objet
/* Le nom et la marque de l'objet adidias ont été modifiés **/
```

La méthode Attaquer

La méthode attaquer de l'objet hero permet de faire baisser la vie de l'adversaire.

script.js > hero

```
attaquer(autrui){
       // coder ici
}
```

Application :

Déclarer une nouvelle méthode de l'objet hero nommé attaquer :

- Attaquer de renvoi rien (undefined par défaut)
- Attaquer prend en paramètre la référence d'un objet autrui similaire au hero.
- Attaquer fait baisse la vie de l'objet autrui de 1.

Le résulat attendu sera :

console

```
>> ennemie.ficheDePersonnage()
<- « Nom : Zombie | PdV : 5 »
>> hero.attaquer(ennemie)
<- undefined
>> ennemie.ficheDePersonnage()
<- « Nom : Zombie | PdV : 4 »
```

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution à la déclaration de l'objet *ennemie*

script.js

```
let ennemie = {
	s_nom : "Zombie",
	n_pv : 4,
	saluer(){
		console.log("Salutation je suis "+ this.s_nom +" ! ");
	},
	ficheDePersonnage(){
		return "Nom : "+this.s_nom+" | PdV : "+this.n_pv;
	},
	soin(n_soin){
		this.n_pv = this.n_pv + n_soin;
	},
	attaquer(ennemie){
		ennemie.n_pv--;
	}
};
```

Le poliche

Ajouter des console.log dans les méthodes **soin** et **attaquer** pour décrire ce qu'il se passe. Comme par exemple :

```
>> ennemie.soin(10);
<- « Zombie c'est soigné de 10 pdv, il à maintenant 15 pdv !";
```