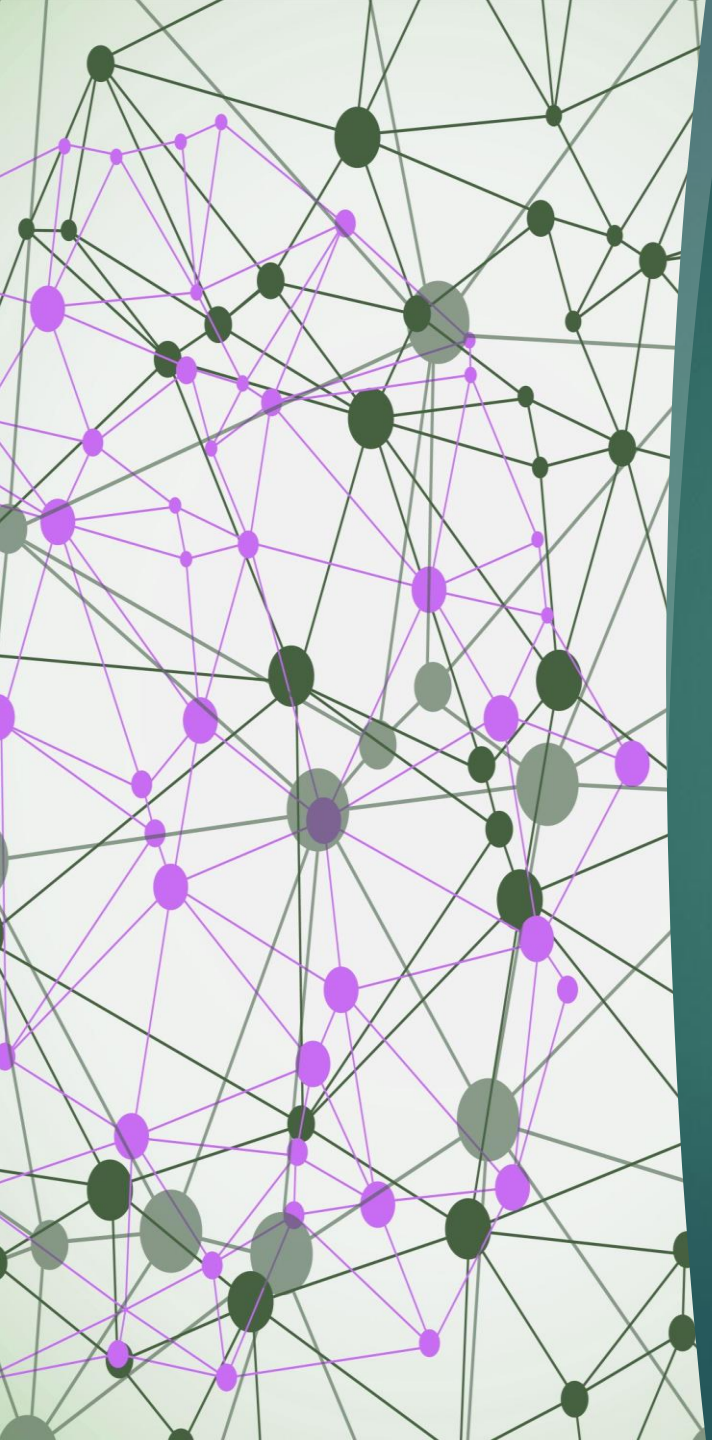


JavaScript

PARTIE 2

Manipuler les objets dans JavaScript

Manipuler les objets

- Un **objet** en JavaScript est un **conteneur** qui possède un ensemble de **propriétés** (variables) et de **méthodes** (fonctions) qui vont fonctionner ensemble.
- Nous pouvons créer des objets à l'aide d'un :
  - Objet littéral ;
  - Constructeur d'objet;

Manipuler les objets

- **Exemple d'objet littéral :**

```
// Déclaration de l'objet Pierre qui se compose de trois membres
let Pierre = {
    /* nom et age sont des propriétés de l'objet Pierre */
    nom : 'Jean',
    age : 29,
    //bonjour est une méthode de l'objet Pierre
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom + ', j\'ai ' + this.age + ' ans');
    }
};
```

Manipuler les objets

Objet littéral :

- Un objet est composé de différents couples de **''nom:valeur''** qu'on appelle **membres**.
- Chaque **nom** d'un **membre** doit être séparé de sa **valeur** par un caractère **deux-points** (:)
- **les différents membres** d'un objet doivent être **séparés** les uns des autres par **une virgule** (,)
- Les membres d'un objet qui stockent des données sont appelés des **propriétés**
- Les membres d'un objet qui contiennent des fonctions en valeur sont appelés des **méthodes**.
- le mot "**this**" remplace le nom de l'objet pendant sa création.

Manipuler les objets

Objet littéral :

- Pour utiliser l'objet littéral :
  1. On commence par écrire le **nom de l'objet** puis le caractère **point** (appelé aussi accesseur) puis enfin le **membre (propriété ou méthode)** auquel on souhaite accéder.
     - **Exemple :**

```
// Modifier le contenu d'une propriété
Pierre.age = 34;
//Afficher le contenu d'une propriété
alert("l'utilisateur : "+ Pierre.nom +" a "+ Pierre.age +" ans");
// Accéder à une méthode
Pierre.bonjour();
//cet appel affiche le message :  Bonjour, je suis Pierre, j'ai 34 ans
```

Manipuler les objets

Objet littéral :

- Pour utiliser l'objet littéral :
  2. on va considérer notre objet comme un tableau composés d'éléments dont les **clefs** sont les **noms des propriétés** et les **valeurs sont les valeurs associées**.
     - **Exemple :**

```
// Modifier le contenu d'une propriété
Pierre["age"] = 44;
//Afficher le contenu d'une propriété
alert("l'utilisateur : "+ Pierre["nom"] +" a "+ Pierre["age"] +" ans");
```

Manipuler les objets

Objet littéral :

Remarque :

- Il est clair qu'avec ce type de déclaration, il ne sera pas possible de réutiliser ce type de définition pour créer un objet de mêmes caractéristiques (ou proches).
- Cette méthode sera donc finalement peu (ou pas) utilisée car elle ne permet pas **l'héritage**.
- **Le principe de l'héritage** est de créer des objets (enfants) qui héritent les caractéristiques d'un autre objet (parent).

Manipuler les objets

Constructeur d'objet :

- **Un constructeur d'objets** (fonction constructeur) est :
  - Un modèle pour créer des objets similaires
  - Un moyen de gagner du temps et de la clarté du code.
- **Un constructeur d'objets** définit les caractéristiques communes de nos objets et on pourra ensuite ajouter les propriétés/fonctions particulières à un objet.
- Pour créer des objets à partir d'un constructeur d'objet, il suffit de **définir la fonction constructeur** et **l'appeler** en utilisant le mot clé **new**.

Manipuler les objets

Constructeur d'objet :

- **Exemple :** fonction constricteur

```
function Utilisateur(a , b) {
    // nom et age sont des propriétés de l'objet Utilisateur
    this.nom = a;
    this.age = b;
    //bonjour est une méthode de l'objet Utilisateur
    this.bonjour = function(){
        alert('Bonjour, je suis ' + this.nom +', j\'ai ' + this.age + ' ans');
    }
};
```

- **Instanciation :** appel de la fonction

```
let pierre= new Utilisateur("Pierre",30);
let sophie= new Utilisateur("Sophie",24);
```

Manipuler les objets

Constructeur d'objet :

Remarque :

- On peut définir un constructeur sans paramètre.
- À intérieur du constructeur, on peut déclarer des variables locaux, qui ne sont pas accessible depuis l'extérieur, à l'aide des mots **let** et **var.**

Manipuler les objets

Constructeur d'objet :

- **Exemples d'utilisation :**

```
sophie.bonjour();
sophie.age=20;
alert("Maintenant, "+sophie["nom"]+" a "+sophie["age"]+" ans");
pierre.taille=160;
alert(`la taille de : ${pierre.nom} est ${pierre.taille} cm`);
```

- La propriété **taille** est exclusive à l'objet **pierre**
- De la même façon, on peut définir une méthode privée à un objet particulier

Manipuler les objets

Constructeur d'objet :

- la méthode **bonjour** sera créée pour chaque instance de l'objet **Utilisateur** ce qui prend de la place inutilement en mémoire.
- On peut corriger cela de la manière suivante:

```
function Utilisateur(a , b) {
    this.nom = a;
    this.age = b;
}
Utilisateur.prototype.bonjour = function(){
    alert('Bonjour, je suis ' + this.nom +', j\'ai ' + this.age + ' ans');
}
```

Manipuler les objets

Constructeur d'objet :

- Un **prototype** est un ensemble d'éléments (attributs/propriétés et méthodes) qui va être associé à un constructeur.
- À l'exécution quand une propriété/méthode d'objet sollicitée dans le code n'est pas trouvée dans le constructeur de l'objet en question, une recherche sera effectuée dans son prototype.

Manipuler les objets

Constructeur d'objet :

- **L'héritage** en JavaScript est obtenu par la copie des prototypes d'un constructeur d'objet dans un autre.
- **Exemple**

Manipuler les objets

Constructeur d'objet :

- **Exemple**

```
function Utilisateur() { };

Utilisateur.prototype.bonjour = function(){
    alert('Bonjour' );
}
Utilisateur.prototype.bonsoir = function(){
    alert('Bonsoir' );
}

function Personne(){}

let Jean= new Personne();

for(let i in Utilisateur.prototype)
Personne.prototype[i]=Utilisateur.prototype[i]

Jean.bonjour();
Jean.bonsoir();
```

Manipuler les objets

Constructeur d'objet :

Les limites de l'héritage dans JavaScript

- L'objet créé à partir du constructeur **enfant** n'est pas considéré comme un objet créé à partir du constructeur **parent**
- Si une méthode est rajoutée dans le prototype d'un type **parent** aprés que la copie des éléments entre le prototype **parent** et le prototype **enfant** a déjà été effectuée alors cette méthode n'est pas disponible dans les objets instanciés Il avec le type **enfant**.

Exercices

“Tableau”

Objet natif : Tableau

- **JavaScript donne deux syntaxes pour définir un tableau:**
  1. Syntaxe littérale :

```
let ages=[]; //Déclaration d'un tableau vide
ages = [29, 27, 29, 30]; //Remplissage du tableau
// Déclaration et initiation du tableau
let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
let produits = ['Livre', 20, 'Ordinateur', 5, ['Smartphones', 100]];
alert (prenoms[1] + ' possède 1 ' + produits[0]+'\n'
      + prenoms[1] + ' a ' + ages[1] + ' ans'+'\n'
      + produits[4][1] + ' ' + produits[4][0]+'\n');
```

Mathilde possède 1 Livre
Mathilde a 27 ans
100 Smartphones

Objet natif : Tableau

- **JavaScript donne deux syntaxes pour définir un tableau:**
  2. Syntaxe POO :

```
let ages = new Array(); //Déclaration d'un tableau vide
ages = [29, 27, 29, 30]; //Remplissage du tableau
// Déclaration et initiation du tableau
let prenoms =new Array('Pierre', 'Mathilde', 'Florian', 'Camille');
alert (prenoms[1] + ' possède 1 ' + produits[0]+'\n'
      + prenoms[1] + ' a ' + ages[1] + ' ans');
```

Pierre possède 1 Ordinateur
Mathilde a 27 ans

Objet natif : Tableau

- **Remplir un tableau par l'utilisateur**
  - Exemple :

```
let tableau=[];
let taille=parseInt(prompt("Quelle est la taille de ton tableau ?"));
for(let i=0; i<taille; i++){
    tableau[i]=prompt("entre l'élément "+(i+1)+" du tableau")
}
```

Objet natif : Tableau

- **Parcourir un tableau avec trois façons :**
  1. Boucle **for classique**
  2. Boucle **for…of**
  3. Boucle **for…in**
- Exemple :

```
let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
//Parcourir le tableau avec la boucle for...of
for (let valeur of prenoms)
    console.log(valeur);
//Parcourir le tableau avec la boucle for...in
for (let index in prenoms)
    console.log(prenoms[index]);
```

Objet natif : Tableau

- **Propriété length**
  - la propriété **length** retourne le nombre d'éléments d'un tableau
  - Exemple:

```
let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
alert(prenoms.length);
```

Objet natif : Tableau

- **Méthodes**
  - La méthode **push(**arg_1, arg_2,…,arg_n**) ajoute des éléments à la fin** du tableau et **retourne sa nouvelle taille**. (arg_1, arg_2…arg_n : sont les éléments à ajouter)
  - La méthode **pop() supprime le dernier** élément du tableau et **retourne l'élément supprimé**.
- Ex :

```
let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
console.log("les éléments du tableau sont : " + prenoms+
"\nLa taille initiale du tableau est :"+prenoms.length+ //4
"\nLa taille après l'insertion est :" +prenoms.push('Sophie','Jean')+ //6
"\nLe dernier élément du tableau est:"+prenoms[prenoms.length-1]+ //Jean
"\nL'élément supprimé est :"+prenoms.pop()+ //Jean
"\nLa taille finale est :"+prenoms.length); //5
```

Objet natif : Tableau

- La méthode **unshift**(arg_1, arg_2…arg_n) **ajoute des éléments au début** du tableau et **retourne sa nouvelle taille**. (arg_1, arg_2,…,arg_n : sont les éléments à ajouter)
- La méthode **shift**() **supprime le premier élément** du tableau et **retourne l'élément supprimé**.
- Exemple :

```
let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
console.log(
"\nLa taille après l'insertion est :" +prenoms.unshift('Sophie','Jean')+ //6
"\nLe premier élément du tableau est:"+prenoms[0]+ //Sophie
"\nL'élément supprimé est :"+prenoms.shift()); //Sophie
```

Objet natif : Tableau

- La méthode **splice**(arg_1, arg_2, arg_3) permet **d'ajouter**, de **supprimer** ou de **remplacer** des éléments n'importe où dans un tableau.
  - **Arg_1**: la position de départ à partir d'où commencer le changement
  - **Arg_2** : le nombre d'éléments à remplacer
  - **Arg_3** : les éléments à ajouter au tableau.

Objet natif : Tableau

- **Arg_1**:
  - En précisant **la position de départ 0**, les changements seront effectués à **partir du début** du tableau. En précisant **la position 1**, ils se feront à partir du **deuxième élément** ...
  - En précisant une **position négative**, les changements seront faits en comptant **à partir de la fin** :
    - -1 pour commencer en partant du dernier élément ;
    - -2 pour commencer en partant de l'avant dernier élément...

Objet natif : Tableau

- **Arg_2** Si on précise 0 en nombre d'éléments à remplacer, alors aucun élément ne sera supprimé du tableau. Dans ce cas, il sera nécessaire de préciser des éléments à rajouter en dernier argument.
- **Arg_3** : Si on ne précise pas d'éléments à rajouter au tableau, le nombre d'éléments à remplacer tel quel précisé en deuxième argument seront supprimés du tableau à partir de la position indiquée en premier argument.

Objet natif : Tableau

- Exemple :

```
let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
alert(`L'élément supprimé : ${prenoms.splice(0,1)}
${prenoms.splice(-1,0,'Sophie')}
Le tableau après l'ajout : ${prenoms}
L'élément remplacé : ${prenoms.splice(1,1,'Anna','Jean')}
Le tableau final : ${prenoms}`);
```

L'élément supprimé : Pierre

Le tableau après l'ajout : Mathilde,Florian,Sophie,Camille
L'élément remplacé : Florian
Le tableau final : Mathilde,Anna,Jean,Sophie,Camille

Objet natif : Tableau

- La méthode **join**(arg) **retourne une chaine de caractères** créée **en concaténant les différentes valeurs** d'un tableau.
  - Le **séparateur utilisé** par défaut sera la **virgule** mais nous pouvons passer le séparateur de notre choix en argument.
  - Exemple **:**

```
let prenoms = ['Pierre', 'Mathilde', 'Florian', 'Camille'];
alert(`Le tableau en chaine de caractères : ${prenoms.join(' - ')}`)
```

le tableau en chaine de caractères : Pierre - Mathilde - Florian - Camille

Objet natif : Tableau

- La méthode **slice**(arg_1, arg_2) renvoie un tableau créé en découpant un autre tableau .
  - **Arg_1** : la position où doit commencer la découpe du tableau.
    - Si la position passée est un nombre négatif, alors le début de la découpe sera calculé à partir de la fin du tableau.
    - Si aucune position de départ n'est passée, la découpe commencera depuis le début du tableau.
  - **Arg_2** : la position où doit s'arrêter la découpe
    - Si la position passée est un nombre négatif, alors la fin de la découpe sera calculé à partir de la fin du tableau.
    - Si aucune position de fin n'est passée, alors on récupèrera le tableau de **Arg_1** jusqu'à la fin.

Objet natif : Tableau

- Exemple :

```
let prenoms = ['Pierre', 'Mathilde','Sofhie', 'Florian', 'Camille'];
alert(`Couper les deux premiers éléments : ${prenoms.slice(0,2)}
Couper les deux derniers éléments : ${prenoms.slice(-2)}`)
```

Couper les deux premiers éléments : Pierre,Mathilde
Couper les deux derniers éléments : Florian,Camille

Objet natif : Tableau

- La méthode **concat**(arg_1, arg_2, …arg_n) **fusionne** des tableaux entre eux pour **créer un nouveau** qu'elle renvoie.
  - arg_1, arg_2, …arg_n : sont des tableaux que l'on souhaite concaténer à un **premier de départ**.
  - Exemple **:**

```
let prenoms1 = ['Pierre'];
let prenoms2 = ['Mathilde','Sofhie', 'Florian'];
let prenoms3 = ['Camille'];
let prenoms = prenoms1.concat(prenoms2,prenoms3);
alert(`le résultat est : ${prenoms}`)
```

le résultat est : Pierre,Mathilde,Sofhie,Florian,Camille

Objet natif : Tableau

- La méthode **includes**(arg) renvoie **True** si le tableau contient la valeur passée en argument, et **False** dans le cas contraire.
  - La méthode **includes**(arg) est sensible à la casse (Majuscule/Minuscule).
  - Exemple :

```
let prenoms = ['Pierre', 'Mathilde','Sofhie', 'Florian', 'Camille'];
alert(`Pierre ${prenoms.includes('Pierre')?" est ":" n'est pas "} dans le tableau
pierre ${prenoms.includes('pierre')?" est ":" n'est pas "} dans le tableau`);
```

```
Pierre  est  dans le tableau
pierre  n'est pas  dans le tableau
```

Objet natif : Tableau

- La méthode **reverse() inverse** l'ordre des valeurs dans d'un tableau.
  - Exemple :

```
let prenoms = ['Pierre', 'Mathilde','Sofhie', 'Florian', 'Camille'];
alert(`les éléments du tableau  : ${prenoms}
les éléments inversés du tableau : ${prenoms.reverse()}`)
```

```
les éléments du tableau  : Pierre,Mathilde,Sofhie,Florian,Camille
les éléments inversés du tableau : Camille,Florian,Sofhie,Mathilde,Pierre
```

Objet natif : Tableau

- La méthode **sort(**arg**) trie** les éléments d'un tableau, dans ce même tableau, et renvoie le tableau.
  - Arg est facultatif
  - En absence de l'**arg,** le tri s'effectue sur les éléments du tableau convertis en chaînes de caractères.
  - L'**arg** est une fonction de rappel définissant l'ordre de tri. Elle prendra deux arguments : le premier élément à comparer **(a)** et le deuxième élément à comparer **(b)**.

Objet natif : Tableau

- Si **fonctionComparaison(a, b)** renvoie une valeur inférieure à 0, donc **a** sera classé avant **b**
- Si **fonctionComparaison(a, b)** renvoie la valeur 0, on laisse **a** et **b** inchangés l'un par rapport à l'autre, mais triés par rapport à tous les autres éléments.
- Si **fonctionComparaison(a, b)** renvoie une valeur supérieure à 0, donc **b** sera classé avant **a**.
- Exemple 1:

```
let nombres = [4, 20, 5, 2, 13];
nombres.sort();
alert(nombres);
```

13,2,20,4,5

Objet natif : Tableau

- Exemple : 2

```
let nombres = [4, 20, 5, 2, 13];
nombres.sort(
    function comparaison(a, b) {
    if (a < b )
        return -1;
    if (a > b )
        return 1;
    // a doit être égal à b
    return 0;
  });
alert(nombres);
```

2,4,5,13,20

```
let nombres = [4, 20, 5, 2, 13];
nombres.sort(
    function comparaison(a, b) {
    if (a < b )
        return 1;
    if (a > b )
        return -1;
    // a doit être égal à b
    return 0;
  });
alert(nombres);
```

20,13,5,4,2

Objet natif : Tableau

- **Tableau à deux dimensions**
  - Syntaxes :

```
//Syntaxe littérale
let tab2=[[],[],[]];
tab2 = [["a", "b", "c"], [1, 2, 3, 4], ["@", "_", ")"]];
alert(tab2[0][0]); //a

//Syntaxe POO
let tableau2 = new Array(3);
//remplissage du tableau
for (let i = 0; i < tableau2.length; i++) {
    //chaque ligne est composée de 4 colonnes
    tableau2[i] = new Array(4);
    for (let j = 0; j < tableau2[i].length; j++)
        tableau2[i][j] = prompt("Entrer la valeur["+i+"]["+j+"]");
}
```

Exercices

Objet natif : Date

- Il y a deux façons pour afficher la date actuelle :
  - Sous forme **littérale**, on utilise **Date**().
  - Sous forme de **nombre** (le nombre de **millisecondes** écoulées depuis le **1er janvier 1970**), on utilise la méthode **Date.now**().
  - Exemple :

```
alert(`avce Date() ===> ${Date()}
avec Date.now() ===> ${Date.now()}`)
```

```
avce Date() ===> Mon Mar 07 2022 11:47:11 GMT+0100 (UTC+01:00)
avec Date.now() ===> 1646650031237
```

Objet natif : Date

- On utilise le constructeur **Date**(arg) avec le mot clef **new**, pour créer et manipuler des dates particulières.
- L'argument du constructeur **Date**() peut être :
  - Vide.
  - Une date sous forme d'un **nombre** (le temps écoulé depuis 1 janvier 1970 en milliseconde)
  - Une Chaine de caractères représentant une date selon la norme **ISO 8601**
  - Des nombres représentent, dans l'ordre :

Objet natif : Date

- L'année;
- Le mois entre 0 (pour janvier) et 11 (pour décembre) ;
- Le jour du mois entre 1 et 31 ;
- L'heure entre 0 et 23 ;
- Les minutes entre 0 et 59 ;
- Les secondes entre 0 et 59 ;
- Les millisecondes entre 0 et 999.

Exemple :

```
let date1 = new Date();
let date2 = new Date(Date.now());
let date3 = new Date('March 7, 2022 13:00:00');
let date4 = new Date(2022, 2, 7, 13, 30);

alert(`date 1 = ${date1}
date 2 = ${date2}
date 3 = ${date3}
date 4 = ${date4}`);
```

```
date 1 = Mon Mar 07 2022 12:57:19 GMT+0100 (UTC+01:00)
date 2 = Mon Mar 07 2022 12:57:19 GMT+0100 (UTC+01:00)
date 3 = Mon Mar 07 2022 13:00:00 GMT+0100 (UTC+01:00)
date 4 = Mon Mar 07 2022 13:30:00 GMT+0100 (UTC+01:00)
```

Objet natif : Date

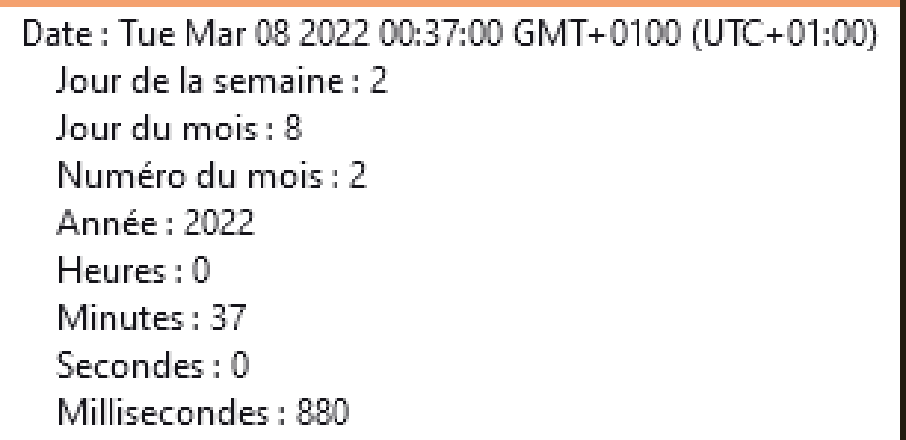
- L'objet **Date** possède des méthodes **getters** qui nous permettre de récupérer un composant de date selon l'heure locale :
  - getDay() renvoie le jour de la semaine sous forme de chiffre (avec 0 pour dimanche, 1 pour lundi et 6 pour samedi);
  - getDate() renvoie le jour du mois en chiffres;
  - getMonth() renvoie le numéro du mois de l'année (avec 0 pour janvier, 1 pour février, 11 pour décembre);
  - getFullYear() renvoie l'année en 4 chiffres;

Objet natif : Date

- getHours() renvoie l'heure en chiffres;
- getMinutes() renvoie les minutes en chiffres;
- getSeconds() renvoie les secondes en chiffres;
- getMilliseconds() renvoie les millisecondes en chiffres.

- **Exemple :**

```
let date1 = new Date();
alert(`Date : ${date1}
    Jour de la semaine : ${date1.getDay()}
    Jour du mois : ${date1.getDate()}
    Numéro du mois : ${date1.getMonth()}
    Année : ${date1.getFullYear()}
    Heures : ${date1.getHours()}
    Minutes : ${date1.getMinutes()}
    Secondes : ${date1.getSeconds()}
    Millisecondes : ${date1.getMilliseconds()}`
    );
```



Objet natif : Date

- L'objet **Date** fournit également des getters équivalents qui permettent de renvoyer un composant de date selon **l'heure UTC** :
  - getUTCDay() - getUTCDate() - getUTCMonth() - getUTCFullYear() - getUTCHours() - getUTCMinutes() - getUTCSeconds() - getUTCMilliseconds()
  - Exemple :

```
let date1 = new Date();
alert(`Date : ${date1}
    Heures locale: ${date1.getHours()}
    Heures UTC : ${date1.getUTCHours()}
    `);
```

```
Date : Tue Mar 08 2022 00:50:24 GMT+0100 (UTC+01:00)
    Heures locale: 0
    Heures UTC : 23
```

Objet natif : Date

- Les **setters** de l'objet **Date** définissent les composants d'une date donnée. Ces setters correspondent exactement aux getters vus précédemment.
  - setDate() et setUTCDate() définissent le jour du mois;
  - setMonth() et setUTCMonth() définissent le mois de l'année
  - setFullYear() et setUTCFullYear() définissent l'année
  - setHours() et setUTCHours () définissent l'heure;
  - setMinutes() et setUTCMinutes() définissent les minutes;
  - setSeconds() et setUTCSeconds() définissent les secondes;
  - setMilliseconds() et setUTCMilliseconds() définissent les millisecondes.

Objet natif : Date

- Exemple :

```
let date1 = new Date();
alert(`Date : ${date1}
${date1.setDate(1),
date1.setMonth(0),
date1.setFullYear(2022),
date1.setHours(0),
date1.setMinutes(0),
date1.setSeconds(0),
date1.setMilliseconds(0)}
Date : ${date1}`);
```

Date : Tue Mar 08 2022 18:46:57 GMT+0100 (UTC+01:00)
1640991600000
Date : Sat Jan 01 2022 00:00:00 GMT+0100 (UTC+01:00)

Objet natif : Date

- La méthode **toLocaleDateString**(arg_1,arg_2) renvoie la partie « **jour-mois-année** » d'une date, formatée en fonction de l'arg_1,arg_2.
- La méthode **toLocaleTimeString**(arg_1,arg_2) renvoie la partie « **heures-minutes-secondes** » d'une date, formatée en fonction de l'arg_1,arg_2
- La méthode **toLocaleString**(arg_1,arg_2) renvoie la **date complète**, formatée en fonction de l'arg_1,arg_2

Objet natif : Date

- Ces trois méthodes prennent deux arguments :
  1. La **locale** sert à définir la langue dans laquelle la date doit être formatée. Exemple : fr-FR, en-EN, ar-AR.
  2. Les **options** modifient le format d'affichage :

Option	représente	Valeur possible
weekday	Le jour de la semaine	**narrow**, **short** et **long**
day	Le jour du mois	**numeric**, **2-digit**
month	Le mois	**numeric**, **2-digit**, **narrow**, **short** et **long**
year	L'année	**numeric** et **2-digit**
hour	Les heures	**numeric** et **2-digit**
minute	Les minutes	**numeric** et **2-digit**
second	Les secondes	**numeric** et **2-digit**

Objet natif : Date

- Exemple :

```
let date1 = new Date();
let dateLocale = date1.toLocaleString('fr-FR',{
    weekday: 'long',
    year: 'numeric',
    month: 'long',
    day: 'numeric',
    hour: 'numeric',
    minute: 'numeric',
    second: 'numeric'
});
alert(dateLocale)
```

mardi 8 mars 2022, 19:18:28

Objet natif : Math

- Certaines propriétés de l'objet Math :
  - **Math.PI** a pour valeur pi;
  - **Math.SQRT1_2** a pour valeur la racine carrée de;
  - **Math.SQRT2** a pour valeur la racine carrée de 2.

```
Math.PI = 3.141592653589793
Math.SQRT1_2 = 0.7071067811865476
Math.SQRT2 = 1.4142135623730951
```

Objet natif : Math

- Certaines méthodes de l'objet Math :
  - La méthode **Math.floor**(nb) arrondit la valeur passée en argument à l'entier immédiatement inférieur (ou égal) à cette valeur.
  - La méthode **Math.ceil**(nb) arrondit la valeur passée en argument à l'entier immédiatement supérieur (ou égal) à cette valeur.
  - la méthode **Math.trunc**(nb) ignore la partie décimale d'un nombre et retourne sa partie entière.

Objet natif : Math

- La méthode **Math.round**(nb) arrondit la valeur passée en argument à l'entier le plus proche.
  - Si la partie décimale de la valeur passée en argument est supérieure ou égale à 0,5, la valeur sera arrondie à l'entier supérieur. Dans le cas contraire, la valeur sera arrondie à l'entier inférieur.
- Exemple :

```
let nb=4.556
alert(`Nombre : ${nb}
floor : ${Math.floor(nb)}
ceil : ${ Math.ceil(nb)}
round : ${ Math.round(nb)}
trunc : ${Math.trunc(nb)}` );
```

Nombre : 4.556
floor : 4
ceil : 5
round : 5
trunc : 4

Objet natif : Math

- La méthode **Math.random**() génére un nombre décimal compris entre 0 (inclus) et 1 (exclu) de manière aléatoire.
- Pour obtenir un nombre aléatoire compris dans une intervalle de notre choix, On va multiplier le résultat de cette méthode par un autre nombre.
- Exemple :

```
alert(
    'Nombre [0 , 1[ ==> ' + Math.random() +
    '\nNombre [0 , 10[ ==> ' + Math.random()*10 +
    '\nNombre [0 , 100[ ==> ' + Math.random()*100 );
```

```
Nombre [0 , 1[ ==> 0.8090321577419934
Nombre [0 , 10[ ==> 6.339771302843385
Nombre [0 , 100[ ==> 69.40336064379922
```

Objet natif : Math

- La méthode **Math.min**(nb1,nb2…..) renvoie le plus petit nombre d'une série de nombres passés en arguments.
- La méthode **Math.max**(nb1,nb2…..)renvoie le plus grand nombre d'une série de nombres passés en arguments.
- La méthode **Math.abs**(nb) renvoie la valeur absolue d'un nombre passé en argument.
- Autres méthodes : **Math.cos(nb)**, **Math.sin(nb)**, **Math.tan(nb)**, **Math.exp(nb)** et **Math.log(nb)**.
  - Si l'une des valeurs fournies en **argument n'est pas un nombre** et ne peut pas être convertie en nombre, alors ces méthodes renverront la valeur **NaN**.

Objet natif : Math

- Exemple:

```
alert(
    'Math.min(2,3,5,6,9) ==> ' + Math.min(2,3,5,6,9) +
    '\nMath.max(2,3,5,6,9) ==> ' + Math.max(2,3,5,6,9) +
    '\nMath.max(2,3,5,6,"9") ==> ' + Math.max(2,3,5,6,"9") +
    '\nMath.max(2,3,5,6,"u") ==> ' + Math.max(2,3,5,6,"u") );
```

```
Math.min(2,3,5,6,9) ==> 2
Math.max(2,3,5,6,9) ==> 9
Math.max(2,3,5,6,"9") ==> 9
Math.max(2,3,5,6,"u") ==> NaN
```

Objet natif : Number

- **Propriétés** de l'objet **Number** :
  - Les propriétés **Number.MIN_VALUE** et **Number.MAX_VALUE** sont respectivement la plus petite valeur positive et plus grande valeur représentables en JavaScript ;
  - Les propriétés **Number.NEGATIVE_INFINITY** et **Number.POSITIVE_INFINITY** sont respectivement l'infini côté négatif et côté positif ;
  - La propriété **Number.NaN** représente une valeur qui n'est pas un nombre et est équivalente à la valeur **NaN**.

Objet natif : Number

- Exemple :

```
alert(   'MIN_VALUE : ' + Number.MIN_VALUE
    + '\n MAX_VALUE : ' + Number.MAX_VALUE
    + '\n NEGATIVE_INFINITY : ' + Number.NEGATIVE_INFINITY
    + '\n POSITIVE_INFINITY : ' + Number.POSITIVE_INFINITY
    + '\n NaN : ' + Number.NaN);
```

```
MIN_VALUE : 5e-324
 MAX_VALUE : 1.7976931348623157e+308
 NEGATIVE_INFINITY : -Infinity
 POSITIVE_INFINITY : Infinity
 NaN : NaN
```

Objet natif : Number

- Quelques méthodes de l'objet **Number :**
  - La méthode **Number.isFinite**(nb) détermine si la valeur nb est un nombre fini.
  - La méthode **Number.isInteger**(nb) détermine si la valeur nb est un entier.
  - La méthode **Number.isNaN**(nb) détermine si la valeur nb est la valeur **NaN**.

Objet natif : Number

- La méthode **Number.parseFloat**(str) convertit la chaîne str en un nombre décimal.
  - L'analyse de la chaîne s'arrête dès qu'un caractère qui n'est pas +,-, un chiffre, un point ou un exposant est rencontré. Ce caractère et tous les suivants vont alors être ignorés.
  - Si le premier caractère de la chaîne ne peut pas être converti en un nombre, **parseFloat**() renverra la valeur **NaN**

Objet natif : Number

- La méthode **Number.parseInt**(str,base) convertit la chaine str en un entier selon une base et renvoie ce nombre en base 10.
- Exemple :

```
let str= "100" ;
alert( Number.parseInt(str,2) );
//résultat : 4
```

Objet natif : Number

- La méthode **toString**() transforme un nombre en une chaine de caractères.
  - Elle renvoie une chaine de caractères représentant notre nombre dans la base passée en argument.
- Exemple :

```
let nb=8;
alert( nb.toString(2) );
//résultat : 1000
```

Objet natif : Number

- La méthode **variable.toFixed(**nb_décimal**)** garde un nombre de chiffres après la virgule.
- Exemple :

```
let nb=20.1270;
alert( nb.toFixed(2) );
//Résultat: 20.13
```

Objet JSON

- **JSON** (JavaScript Object Notation) est une syntaxe pour stocker et échanger des données avec le serveur.
- **JSON** est un format de texte, indépendant de tout langage, permettant d'organiser des données.
- Les objets JSON sont écrits entre accolades. Ces objets peuvent contenir plusieurs paires des clés/valeurs qui peuvent s'imbriquer ou non.

Objet JSON

- **Exemple :**

```
{
"nom" :"Nour" ,
"prenom" : "Said",
"age":20,
"interets"  :null,
"experience"  : ["CSS","JS","HTML"],
"addresse" : {"Rue" : "hassan 2","Ville": "Tiznit"}
}
```

Objet JSON

- Pour traiter et afficher les données JSON dans les pages web, on a souvent besoin de les convertir en objets javaScript et vice versa.
- En JavaScript, les méthodes utilisées sont :
  - JSON.parse permet de convertir JSON vers un objet javascript.
  - JSON.stringify permet de convertir des objets javascript vers des données JSON

Objet JSON

- Exemple : 1

```
//Création d'un string JSON
let Data =
`{
"nom" :"Nour" ,
"prenom" : "Said",
"age":20,
"interets"  :null,
"experience"  : ["CSS","JS","HTML"],
"addresse" : {"Rue" : "hassan 2","Ville": "Tiznit"}
}`
```

```
//Convertir JSON vers Javascript
let Objet = JSON.parse(Data);
console.log(Objet.nom + "\n"
+ Objet.prenom+'\n'
+ Objet.experience[2]+'\n'
+ Objet.addresse.Rue);
```

Nour
Said
HTML
hassan 2

Objet JSON

- Exemple : 2

```
//Création d'un objet javaScript
let objet =
{
"nom" :"Nour" ,
"prenom" : "Said",
"age":20,
"interets"  :null,
"experience"  : ["CSS","JS","HTML"],
"addresse" : {"Rue" : "hassan 2","Ville": "Tiznit"}
}
```

```
//Convertir l'objet javascript vers json
let data = JSON.stringify(objet);
console.log(data);
```

{"nom":"Nour","prenom":"Said","age":20,"interets":null,"experience":["CSS","JS","HTML"],"addresse":{"Rue":"hassan 2","Ville":"Tiznit"}}

Objet String

Objet natif : String

- Le constructeur String() possède :
  - la propriété **length** : permet d'obtenir la longueur d'une chaine de caractères.

```
let str = 'Bonjour tout le monde.';
console.log(`Cette phrase contient ${str.length} caractères`);
//Cette phrase contient 22 caractères
```

Objet natif : String

- La méthode **includes**() : permet de déterminer si une chaine de caractères est inclue dans une autre.

```
let str = 'Bonjour tout le monde.', ch = 'tout';
console.log(`Cette phrase ${str.includes(ch)? "contient":"ne contient pas"} la chaine "${ch}"`);
//Cette phrase contient la chaine "tout"
```

```
let str = 'Bonjour tout le monde.', ch = 'tout', pos=10 ;
console.log(`Cette phrase ${str.includes(ch,pos)? "contient":"ne contient pas"} "${ch}" à partir de la position ${pos}`);
//Cette phrase ne contient pas "tout" à partir de la position 10
```

Objet natif : String

- La méthode **startsWith**() / **endsWith**() détermine si une chaine **commence** / **se termine** par une autre chaine.

```
let str = 'Bonjour tout le monde.', ch = 'bon';
console.log(`Cette phrase ${str.startsWith(ch)? "commence":"ne commence pas"} par "${ch}"`);
//Cette phrase ne commence pas par "bon"
```

```
let str = 'Bonjour tout le monde.', ch = 'tout', pos=8;
console.log(`Cette phrase ${str.startsWith(ch,pos)? "commence":"ne commence pas"} par "${ch}" à la position ${pos}`);
//Cette phrase commence par "tout" à la position 8
```

Objet natif : String

- La méthode **indexOf**() / **LastIndexOf**() détermine la position de la **première** / **dernière** occurrence d'une chaine dans une autre. (ou -1 si elle n'est pas trouvée)

```
let str = 'Bonjour tout le monde.';
console.log(`Le premier \"ou\" se trouve à la position: ${str.indexOf("ou")}`);
//Le premier "ou" se trouve à la position: 4
```

```
let str = 'Bonjour tout le monde.', pos=8;
console.log(`Le premier \"ou\", après la position ${pos}, se trouve à la position: ${str.indexOf("ou",pos)} `);
//Le premier "ou", après la position 8, se trouve à la position: 9
```

Objet natif : String

- **La méthode replace()** nous permet de rechercher une expression dans une chaine et de la remplacer par une autre.

```
let str = 'Bonjour tout le monde.';
console.log(`la phrase devient: ${str.replace("Bonjour","Bonsoir")}`);
//la phrase devient: Bonsoir tout le monde.
```

Objet natif : String

- La méthode **substring(deb,fin)** retourne une sous-chaîne de la chaîne courante, entre un indice de début **deb** et un indice de fin **fin**.

```
let str = 'Bonjour tout le monde.', deb=8 ;
console.log(`Nous avons extrait "${str.substring(deb)}" de la phrase`);
//Nous avons extrait "tout le monde." de la phrase
```

```
let str = 'Bonjour tout le monde.', deb=8, fin=12 ;
console.log(`Nous avons extrait "${str.substring(deb, fin)}" de la phrase`);
//Nous avons extrait "tout" de la phrase
```

```
let str = 'Bonjour tout le monde.', deb=12, fin=8 ;
console.log(`Nous avons extrait "${str.substring(deb, fin)}" de la phrase`);
//Nous avons extrait "tout" de la phrase
```

Objet natif : String

- **La méthode toLowerCase()** retourne une chaine de caractères en minuscules.
- **la méthode toUpperCase()** retourne une chaine de caractères en majuscules
- **La méthode trim()** supprime les espaces et les retours à la ligne en début et en fin de chaîne.

Exercices

Objet natif : RegExp

- les **expressions régulières** nous permettent de vérifier la présence de certains caractères ou suites de caractères dans une chaine de caractères.
- Nous disposons de deux façons de créer des expressions régulières :
  - déclarer les expressions régulières de manière littérale :

```
let express1= /Pierre/;
```

  - appeler le constructeur **RegExp**() :

```
let express2 = RegExp("Pierre");
```

Objet natif : RegExp

- les méthodes qui utilisent les **expressions régulières** sont:
  - De l'objet string
    - La méthode **match**(arg)
    - La méthode **search**(arg)
    - La méthode **replace**(arg1, arg2)
    - La méthode **split**(arg)
  - De l'objet RegExp
    - La méthode **exec(**arg**)**
    - La méthode **test()**

Objet natif : RegExp

- La méthode **match**(arg) renvoie un **tableau** contenant les caractères qui satisfont à l'expression régulière.
  - **match**() renvoie **null** si aucun résultat n'est pas trouvé.
  - **match**() renvoie la première occurrence trouvée.
  - **match**() renvoie toutes les occurrences, si l'option **g** est utilisée.
  - Exemple :

```
let chaine = 'Bonjour, je m\'appelle Pierre et vous ?';
let express1 = /[A-Z]/;
let express2 = /[A-Z]/g;
alert(`${chaine.match(express1)}
${chaine.match(express2)}`);
```

B

B,P

Objet natif : RegExp

- La méthode **search**(arg) retourne la **position** de la première occurrence trouvée de l'expression régulière recherchée ou **-1** si rien n'est trouvé.
  - Exemple :

```
let chaine = 'Bonjour, je m\'appelle Pierre et vous ?';
let express1 = /e/;
let express2 = /en/;
alert(`${chaine.search(express1)}
${chaine.search(express2)}`);
```

10
-1

Objet natif : RegExp

- La méthode **replace**(arg1, arg2) **remplace** les occurrences trouvées d'une expression régulière par d'autres caractères.
  - Cette méthode renvoie **une nouvelle chaine de caractères** avec les remplacements effectués s'ils existent.
  - La chaine de départ reste inchangée.
  - Exemple :

```
let chaine = 'Bonjour, je m\'appelle Pierre et vous ?';
let express1 = /e/;
let express2 = /ou/g;
alert(`${chaine.replace(express1, 'E')}
${chaine.replace(express2, 'OU')}`);
```

Bonjour, jE m'appelle Pierre et vous ?
BonjOUr, je m'appelle Pierre et vOUs ?

Objet natif : RegExp

- La méthode **split(arg) divise** la chaine de caractères en fonction des séparateurs fournis en argument.
  - Cette méthode **retourne un tableau** de **sous chaines** créé à partir de la chaine de départ.
  - Exemple :

```
let chaine = 'Bonjour, je m\'appelle Pierre et vous ?';
let express1 = /[,?]/;
let sousChaine = chaine.split(express1);
alert(`${sousChaine[0]}
${sousChaine[1]}`)
```

Objet natif : RegExp

- La méthode **exec(**arg**)** recherche les occurrences d'une expression régulière dans une chaine de caractères.
  - Cette méthode renvoie la première occurrence trouvée, Sinon elle renvoie la valeur **null**.
  - Exemple :

```
let chaine = 'Bonjour, je m\'appelle Pierre et vous ?';
let express1 = /ou/;
let express2 = /en/;
alert(`${express1.exec(chaine)}
${express2.exec(chaine)}`)
```

ou
null

Objet natif : RegExp

- La méthode **test()** recherche des occurrences d'une expression régulière dans une chaine de caractères
  - Cette méthode renvoie **true** si au moins une occurrence a été trouvée ou **false** dans le cas contraire.
  - Exemple:

```
let chaine = 'Bonjour, je m\'appelle Pierre et vous ?';
let express1 = /Pierre/;
if(express1.exec(chaine)){
    alert('"Pierre" trouvé dans la chaine');
}
```

"Pierre" trouvé dans la chaine

Objet natif : RegExp

- **Options**
  - Les **options**, encore appelées **modificateurs**, sont des caractères qui vont nous permettre d'ajouter des options à nos expressions régulières.

Option	Description
g	Permet d'effectuer une recherche globale
i	Rend la recherche insensible à la casse
m	Permet de tenir compte des caractères de retour à la ligne et de retour chariot et fait que ^ et $ vont pouvoir être utilisés pour chercher un début et une fin de ligne
s	Cette option permet au méta-caractère . (point) de remplacer n'importe quel caractère y compris un caractère de nouvelle ligne

Objet natif : RegExp

- Exemples :

```
let chaine = 'Bonjour, je suis Pierre\n et mon no. est le [06-36-65-65-65]';
let express1 = /pierre/; //Cherche "pierre" exactement
let express2 = /pierre/i; //Cherche "pierre", "PIERRE", "PiErRe"...
let express3 = /e$/; //Cherche "e" en fin de chaine
let express4 = /e$/m; //Cherche "e" en fin de chaque ligne
let express5 = /./gs; //Cherche tout caractère et effectue une recherche globale
```

Exercices