

A decorative graphic on the left side of the slide. It consists of a blue parallelogram and a light green parallelogram, both tilted at an angle. The blue shape is in the foreground, and the green shape is partially behind it. The background of the entire slide is a dark navy blue with subtle, lighter blue geometric patterns.

Javascript



Avant-propos: Les algorithmes

- Suite d'instructions permettant de résoudre un problème
- Peuvent comporter des structures de contrôle:
 - boucles
 - conditions
- Peuvent avoir des données d'entrée et des données de sortie
- Ne sont pas spécifiques à l'informatique



Avant-propos: Les langages de programmation

- Permettent d'écrire des algorithmes qui seront exécutés par un ordinateur
- Un même algorithme peut être écrit dans tous les langages de programmation
- Un algorithme écrit dans un langage de programmation est un programme informatique
- Un programme informatique peut être assimilé à son code source, code source qui est écrit par le développeur puis traduit en langage machine exécutable par le processeur de l'ordinateur. C'est le processus de compilation.



Avant-propos: complexité des algorithmes

- Les ordinateurs, bien que maintenant capables d'effectuer plusieurs milliards d'opérations par seconde, ont des ressources limitées à la fois en capacité de calcul ainsi qu'en capacité mémoire
- La complexité des algorithmes peut être étudiée selon plusieurs critères:
 - Complexité d'exécution (occupation du temps de calcul du processeur)
 - Complexité en mémoire (occupation de la mémoire vive de l'ordinateur)
- Le temps d'exécution ainsi que l'occupation mémoire d'un algorithme caractérise son coût
- Deux algorithmes différents solutionnant un même problème peuvent avoir des coûts différents
- Cela fait partie du travail de développeur que d'être capable d'étudier la complexité d'un algorithme et éventuellement de l'optimiser pour réduire son coût



ECMAScript

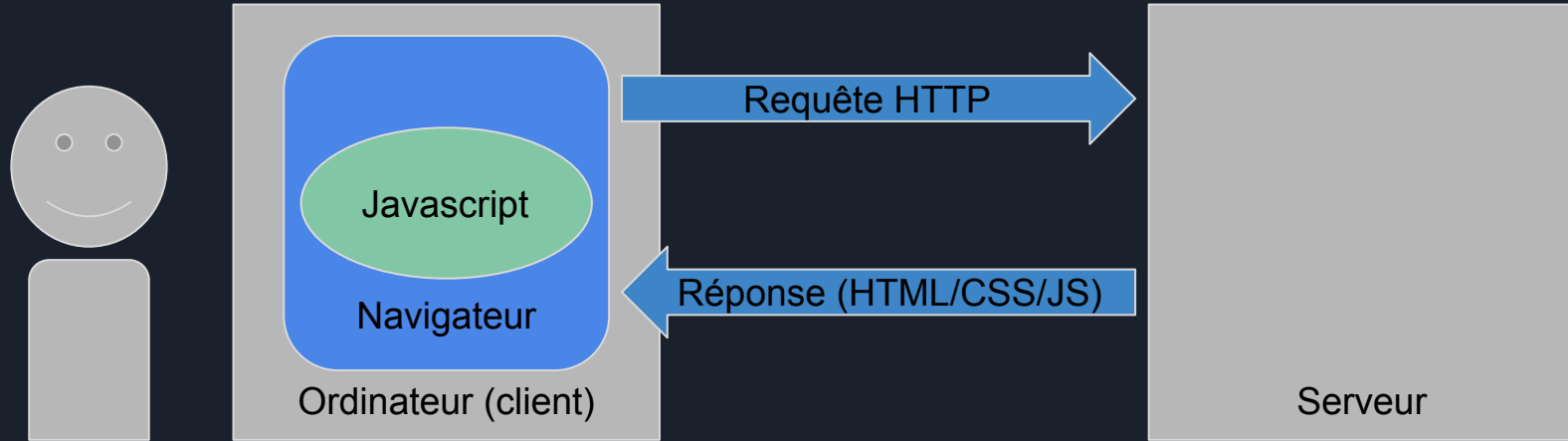
- Ecma International: organisme de standardisation dans l'informatique
- ECMAScript:
 - norme pour définir les fonctionnalités des langages de programmation
 - mise en oeuvre en Javascript et ActionScript
 - version actuelle: ES7
- Intérêt de la définition de normes en informatique:
 - permettre l'interopérabilité des systèmes informatiques
 - dans le cas plus spécifique de la norme ECMAScript: faire en sorte que les différents navigateurs interprètent le Javascript de la même manière



Qu'est-ce que Javascript ?

- Langage de programmation
- Interprété ou compilé à l'exécution
- Orienté prototype
- Classes introduites par la norme ES6
- À l'origine exécuté côté client par le navigateur pour rendre les pages web dynamiques

Qu'est-ce que Javascript ?





Bases : les variables

- Possèdent un nom
- Permettent de stocker une donnée
- Ont une portée (Scope) : c'est la portion de code où la variable est accessible
- Sont déclarées avec les mots-clés **let**, **const**
- **let** : la variable a un scope de niveau bloc
- **const** : a variable a un scope de niveau bloc et ne peut pas être réaffectée
- De préférence, utiliser **const** en priorité, **let** si c'est nécessaire
- Il existe toujours le mot clé **var** , mais il est désuet, on lui préfère **let**



Bases : les types de données

Boolean	Deux valeurs possibles: true , false
Number	Nombres entiers et décimaux, négatifs ou positifs
String	Chaîne de caractères, encadrée par " ou '
Objects	Ensemble de propriétés
Undefined	Valeur assignée par défaut à une variable déclarée mais non initialisée
Null	Absence de valeur



Bases : les opérateurs arithmétiques

+	Addition (fonctionne aussi pour les Strings)
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste de la division)
++	Incrémentation (+1). C'est un opérateur unaire.
--	Décrémentation (-1). C'est un opérateur unaire.



Bases : les opérateurs d'affectation

=	Affectation
+=	Additionne la valeur de l'opérande gauche à la valeur de l'opérande droit et stocke le résultat dans l'opérande gauche
-=	Soustrait la valeur de l'opérande droit à la valeur de l'opérande gauche et stocke le résultat dans l'opérande gauche
/=	Divise la valeur de l'opérande gauche par la valeur de l'opérande droit et stocke le résultat dans l'opérande gauche
*=	Multiplie la valeur de l'opérande gauche par la valeur de l'opérande droit et stocke le résultat dans l'opérande gauche



Exemple

```
let age = 30;    // Number
age += 5;        // Ajout de 5 à age
age++;           // Ajout de 1 à age
console.log(age); // 36

let name = "Bertrand"; // String
name = name + " Bon";   // Concaténation
console.log(name);      // Bertrand Bon

let isDeveloper = true; // Boolean
isDeveloper = false;

let car;
console.log(car);        // undefined
car = null;
```

Exemple: Objets

```
let emptyObj = {}; // Objet vide
emptyObj.attr1 = "NotEmptyAnymore"; // Initialisation d'un nouvel attribut
console.log(emptyObj); // { attr1: "NotEmptyAnymore" }

const myAge = 30;
let person = {
  name: "Bertrand",
  age: myAge,
  languages: ["Javascript", "Java", "Python"], // Tableau (ensemble de valeurs)
  car: {
    manufacturer: "Renault",
    model: "Clio",
    km: 111945
  }
};

person.name = person.name + " Bon"; // Modification de l'attribut name
console.log(person.languages); // ["Javascript", "Java", "Python"]
console.log(person.car.manufacturer); // Renault
person.car.km++; // Incrémentation de l'année de la voiture
```



Bases : l'opérateur de déstructuration

- Extrait des valeurs d'objets ou de tableaux pour les stocker dans des variables distinctes
- Utilise une syntaxe semblable à la construction d'objets ou de tableaux

```
const tab = [1, 2, 3];  
const [one, two, three] = tab;  
console.log(two); // 2
```

```
const obj = { a: "valeur1", b: "valeur2" };  
const {a, b} = obj;  
console.log(b); // valeur2
```



Bases : l'opérateur "spread"

- Extrait les éléments d'un tableau (array) ou les attributs d'un objet
- Utilise la syntaxe ...

```
const tab1 = [1, 2];
const tab2 = [3, 4];
console.log([...tab1, ...tab2]); // [1, 2, 3, 4]

const obj1 = { a: "valeur1", b: "valeur2" };
const obj2 = { c: "valeur3", ...obj1};
const obj3 = { ...obj1 };
console.log(obj2); // { c: "valeur3", a: "valeur1", b: "valeur2" }
console.log(obj3); // { a: "valeur1", b: "valeur2" }
```



Bases : autres opérateurs

delete	Supprime un objet, un attribut d'un objet ou un élément d'un tableau. C'est un opérateur unaire.
typeof	Retourne une String exprimant le type d'un objet. C'est un opérateur unaire.

```
const obj = { a: "valeur1", b: "valeur2" };  
delete obj.a;  
console.log(obj); // { b: "valeur2" }  
console.log(typeof obj); // object
```




Bases : les commentaires

- `//` Pour entamer un commentaire sur la ligne courante
- `/*` Pour entamer un commentaire jusqu'à arriver au caractère de fermeture `*/`
- Les commentaires doivent aider à la compréhension du code par vous ou par les autres développeurs qui liront le code
- Quand on revient sur son propre code après plusieurs mois, on peut mettre du temps à comprendre le code qu'on a soi-même écrit
- Mon avis subjectif sur les commentaires:
 - Trop de commentaires rend le code illisible
 - Si il est bien structuré, découpé en fonctions, et si les variables et fonctions sont nommées avec des noms qui ont du sens, le code doit pouvoir généralement se passer de commentaires
 - Réserver les commentaires à ces situations:
 - Algorithmes complexes
 - Avant une fonction/classe pour expliquer ce qu'elle attend comme arguments, ce qu'elle en fait, et ce qu'elle retourne (documentation)
 - Point de code inhabituel



À propos des bonnes pratiques en développement

- Il est très important d'être rigoureux dans le nommage des fonctions et des variables, les membres d'un même projet doivent utiliser les même conventions pour que le code garde une forme cohérente, reste lisible et maintenable
- De préférence, on utilise l'anglais dans le code
- Un guide pour les conventions de style en Javascript:
https://www.w3schools.com/js/js_conventions.asp
- Dans un premier temps, veuillez à nommer vos fonctions et variables de la façon suivante:
 - le nom commence par une lettre minuscule
 - pas de tiret ou d'underscore dans le nom
 - si le nom est composé de plusieurs mots distincts, mettre une majuscule à la première lettre du mot, exemple: `const maSuperVariable = 1;`
- De manière générale, n'hésitez pas à bien découper votre code en fonctions. Il vaut mieux éviter d'avoir des fonctions trop longues et donc trop complexes



● Exercices

- Sauf mention contraire, utilisez un fichier par numéro d'exercice (exemple, pour l'exercice 1, créez un fichier `ex1.js`)
- Créez un répertoire dans lequel vous placerez vos fichiers d'exercices
- Dans ce répertoire, créez un fichier `main.html` avec le contenu suivant:

```
<html>
  <head>
    <meta charset="UTF-8">
    <script src="ex1c.js"></script>
  </head>
  <body>
    <input id="myInput" type="number">
    <button onclick="submitValue()">GO</button>
  </body>
</html>
```



● Exercice 1 : Manipulation de nombres

Exercice 1a:

1. Déclarez et initialisez une variable avec la valeur 256
2. Multipliez la par 3
3. Divisez la par 2
4. Incrémentez la de 1
5. Affichez le résultat : quel est-il ?

Exercice 1b:

6. Affichez l'opération suivante: $2 + 3 * 3$
7. Affichez l'opération suivante: $(2 + 3) * 3$
8. Affichez l'opération suivante: $4 / 2 + 2 * 3$
9. Affichez l'opération suivante: $4 / (2 + 2 * 3)$
10. Que déduire des priorités de calcul ?



● Exercice 1 : Manipulation de nombres

Exercice 1c: (dans un fichier distinct)

1. Ajoutez ce code au début de votre fichier:

```
function submitValue() {  
    number = document.getElementById("myInput").value;  
    console.log('Vous avez saisi: ', number);  
    // Commencez votre code ici  
}
```

2. Utilisez l'opérateur modulo (%) pour obtenir le reste de la division par 2
3. Exécutez plusieurs fois ce code en saisissant un entier différent à chaque fois
4. Quelle information le reste de la division par 2 nous donne sur le nombre saisi ?



● Exercice 2: Manipulation de strings

Exercice 2:

1. Déclarez et initialisez une variable avec la valeur "Hello world"
2. Affichez la taille de la string (attribut **length**)
3. Concaténez une string ", Hello Javascript" à la première
4. Affichez le résultat
5. Affichez la taille de la string
6. Affichez le dernier caractère (opérateur `[]`)



● Exercice 3: Manipulation d'objets

Exercice 3a:

1. Déclarez et initialisez une variable *personne* de type objet contenant les attributs suivants:
 - prenom: "Damien"
 - age : 27
2. Affichez l'âge
3. Incrémentez l'âge
4. Affichez l'âge
5. Affichez le premier caractère du prenom



● Exercice 3: Manipulation d'objets

Exercice 3b:

1. Ajoutez l'attribut suivant à votre objet *personne* :
 - `pere : { prenom: "Jean", age : 57 }`
2. Affichez votre objet *personne*
3. Modifiez l'âge du père à la valeur 60
4. Affichez le père

Exercice 3c:

1. Extraire l'âge et le prenom de *personne* en utilisant l'opérateur de déstructuration
2. Affichez les deux variables ainsi extraites



● Exercice 3: Manipulation d'objets

Exercice 3d:

1. Créez un nouvel objet *enfant* avec un prenom et age de votre choix
2. Ajoutez-lui un attribut *pere* dont la valeur sera définie par l'utilisation de l'opérateur spread sur l'objet *personne* défini dans les exercices précédents
3. Affichez votre objet *enfant*

Exercice 3e:

1. Utilisez l'opérateur delete sur *personne.pere*
2. Affichez *personne*



● Exercice 4: Undefined et Null

Exercice 4a:

1. Déclarez une variable sans l'initialiser
2. Affichez-la
3. Essayer d'accéder à l'attribut *length* de votre variable

Exercice 4b:

1. Modifiez l'exercice précédent en initialisant votre variable à null




● Exercice 5: typeof

Exercice 5a:

1. Déclarez et initialisez une variable avec pour valeur un nombre de votre choix
2. Affichez le type de votre variable avec l'opérateur typeof

Exercice 5b:

1. Déclarez et initialisez une variable de type objet
2. Affichez le type de votre variable avec l'opérateur typeof



Bases : les opérateurs de comparaison

==	Égal
!=	Différent
<=	Inférieur ou égal
>=	Supérieur ou égal
<	Inférieur
>	Supérieur
===	Strictement égal (vrai si même valeur et même type)
!==	Strictement différent (vrai si même type mais valeur différente, ou types différents)



● Exercice 6 : égalité, différence

Exercice 6a:

1. Affichez l'opération ("5" == 5)
2. Affichez l'opération ("5" === 5)
3. Affichez l'opération ("5" != 5)
4. Affichez l'opération ("5" !== 5)
5. Que déduire de cet ensemble de résultats ?

Exercice 6b:

1. Déclarez et initialisez une variable *obj1* de type objet avec un attribut de votre choix
2. Déclarez et initialisez une variable *obj2* de type objet, identique à *obj1*
3. Déclarez et initialisez une variable *obj3* avec la variable *obj1*
4. Comparez l'égalité (===) de *obj1* avec *obj2* et *obj1* avec *obj3*



Bases : les opérateurs de comparaison

- Dans les exercices précédents, nous avons principalement comparé des valeurs entre elles. En pratique, on comparera plutôt des variables entre elles, ou des variable avec des valeurs
- Préférez l'usage de l'égalité et inégalité stricte (`===` et `!==`) à celui de l'égalité et inégalité simple (`==` et `!=`)



Bases : les opérateurs logiques

&&	ET
	OU (inclusif)
!	NON



● Exercice 8 : true ou false

1. Affichez l'opération (`true && true`)
2. Affichez l'opération (`false && true`)
3. Affichez l'opération (`true || true`)
4. Affichez l'opération (`true || false`)
5. Affichez l'opération (`false || false`)
6. Affichez l'opération (`!true`)
7. Affichez l'opération (`!false`)
8. Affichez l'opération (`true || false && true`)
9. Affichez l'opération (`true && false || true && false`)
10. Affichez l'opération (`5 <= 6 && 5 > 4`)
11. Affichez l'opération (`5 <= 6 && 5 > 7`)
12. Affichez l'opération (`5 <= 6 || 5 > 7`)
13. Affichez l'opération (`5 <= 6 || 5 > 7 && 5 !== 3`)
14. Affichez l'opération (`5 <= 7 && !(5 !== 3)`)



Bases : les structures de contrôle, conditions

If, else

```
if (condition) {  
    instructions1;  
}  
  
if (condition) {  
    instructions1;  
} else {  
    instructions2;  
}
```

```
if (condition) {  
    instructions1;  
} else if (condition2) {  
    instructions2;  
} else {  
    instructions3;  
}
```



● Exercice 9 : If, else

Exercice 9a:

1. Déclarez et initialisez une variable avec la valeur 4
2. Déclarez et initialisez une variable avec la valeur 5
3. Copiez le code suivant à la suite:

```
if (/* CONDITION A TROUVER ICI */) {  
    console.log("Condition vraie");  
} else {  
    console.log("Raté !");  
}
```

4. Remplacez le commentaire du if par une comparaison entre les deux variables qui permette de rentrer dans le corps du if et afficher le texte "Condition vraie"



● Exercice 9 : If, else

Exercice 9b (dans un fichier distinct):

1. Réutilisez le code permettant de lire un nombre entier saisi sur la page (exercice 1c)
2. Écrire un ensemble de conditions testant l'entier saisi et permettant d'afficher les textes suivants:
 - Le nombre saisi est compris entre 0 et 10
 - Le nombre saisi est compris entre 11 et 100
 - Le nombre saisi a une autre valeur



Bases : les structures de contrôle, conditions

Switch : tester la valeur d'une variable

```
switch (variable) {  
    case value1:  
        instructions1;  
        break;  
    case value2:  
        instructions2;  
        break;  
    default:  
        // instructions par défaut  
}  

```

```
const maVariable = 5;  
switch(maVariable) {  
    case 2:  
        console.log("maVariable vaut 2");  
        break;  
    case 5:  
        console.log("maVariable vaut 5");  
        break;  
    default:  
        console.log("maVariable vaut autre chose");  
        break;  
}  

```



● Exercice 10 : switch

Exercice 10:

1. Réutilisez le code permettant de lire un nombre entier saisi sur la page (exercice 1c)
2. Ajoutez un switch après la lecture du nombre entier:
 - Si le nombre lu est 1, lui ajouter 3
 - Si le nombre lu est 2, le multiplier par 3
 - Dans tous les autres cas, lui soustraire 2
3. Affichez le nombre obtenu après le switch



Bases : les structures de contrôle, conditions

Ternaires

```
// Si condition est vrai, test vaudra value1, sinon test vaudra value2  
const test = condition ? value1 : value2;
```

- Forme compacte de if, else permettant d'affecter une variable ou de retourner une valeur

```
const a = -1;  
const test = a < 0 ? "Negatif" : "Positif";
```



● Exercice 11 : ternaire

Exercice 11:

1. Réutilisez le code permettant de lire un nombre entier saisi sur la page (exercice 1c)
2. Déclarez une nouvelle variable dont l'initialisation sera conditionnée par un ternaire:
 - Si le nombre est pair, alors on initialise la variable avec la string "pair"
 - Si le nombre est impair, alors on initialise la variable avec la string "impair"
3. Affichez la variable



Bases : les structures de contrôle, boucles

La boucle for : pour itérer un nombre de fois donné

```
for (let i = 0; i < 5; i++) {  
    instructions;  
}
```

```
for (let i = 0; i < 50; i++) {  
    console.log("Coucou");  
}
```




● Exercice 12 : boucle for

Exercice 12:

1. Déclarez et initialisez une variable avec la valeur 3
2. Créez une boucle for qui boucle 3 fois et qui multiplie la variable par 2 à chaque tour de boucle et stocke le résultat dans la variable
3. Affichez la variable



Bases : les structures de contrôle, boucles

La boucle while : pour itérer tant qu'une condition est remplie

```
while (condition) {  
    instructions;  
}
```

```
let y = 0;  
while (y < 10) {  
    console.log("Bonjour");  
    y++;  
}
```



● Exercice 13 : boucle while

Exercice 13a:

1. Déclarez et initialisez une variable avec la valeur 3
2. Créez une boucle while qui, tant que la variable est inférieure à 1000, lui ajoute 2
3. Affichez le résultat après la boucle

Exercice 13b: (Attention, ne l'exécutez pas si votre ordinateur est déjà un peu fatigué, ou fermez rapidement la page une fois que vous visualisez le problème sur la console)

1. Déclarez et initialisez une variable avec la valeur 2
2. Créez une boucle while qui, tant que la variable est différente de 5, lui ajoute 2
3. Affichez la variable à chaque tour de boucle
4. (Si vous avez un problème, c'est bien le résultat attendu)



Bases : les structures de contrôle, boucles

La boucle `do...while` : pour itérer au moins une fois puis tant qu'une condition est remplie

```
do {  
    instructions;  
} while (condition);
```



Bases : les structures de contrôle, boucles

- break : sort de la boucle
- continue : passe à la prochaine itération de la boucle



● Exercice 14 : break

Exercice 14:

1. Déclarez et initialisez une variable avec la valeur 1
2. Créez une boucle while qui, tant que la variable est inférieure à 10, lui ajoute 1
3. Ajoutez une condition if dans le corps de la boucle, qui exécute un break si la variable vaut 5
4. Affichez la variable



Bases : les fonctions

- En Javascript, sont des objets
- Ensemble d'instructions réutilisable
- Peuvent être nommées ou anonymes
- Peuvent avoir entre 0 et X paramètres
- Peuvent retourner une valeur
- L'exécution d'une fonction s'arrête soit:
 - à la fin du bloc de fonction
 - au premier return rencontré à l'exécution
 - en cas d'exception (erreur) non capturée



Bases : les fonctions

Déclarer une fonction: fonction nommée

```
function nomDeLaFonction(parametre1, parametre2, ...) {  
    instructions;  
    return value; /* Optionnel mais les fonctions retournent généralement  
                   une valeur */  
}
```

```
function maFonction(param1, param2, param3) {  
    console.log("je suis dans maFonction");  
    return param1 + param2 + param3;  
}
```




Bases : les fonctions

Déclarer une fonction: fonction fléchée (anonyme)

```
const nomDeLaFonction = (parametre1, parametre2, ...) => {  
  instructions;  
  return value;  
}
```

```
const nomDeLaFonction = (parametre1, parametre2, ...) => value;
```

```
const maFonction = (param1, param2, param3) => {  
  console.log("je suis dans maFonction");  
  return param1 + param2 + param3;  
}
```



Bases : les fonctions

Appeler une fonction

```
maFonction(2, 5, "abc");
```

Appeler une fonction et stocker le résultat dans une variable

```
const resultat = maFonction(2, 5, "abc");
```



● Exercice 15 : fonctions

Exercice 15a:

1. Déclarez une fonction nommée prenant 2 paramètres
2. Dans le corps de la fonction, retournez la multiplication des deux paramètres
3. Appelez la fonction en stockant le résultat dans une variable
4. Affichez la variable

Exercice 15b:

1. Créez la même fonction en utilisant cette fois la syntaxe de fonction fléchée
2. Appelez la fonction en affichant le résultat directement (sans passer par une variable)



● Exercice 16 : fonctions et scope des variables

Exercice 16a:

1. Déclarez et initialisez une variable
2. Déclarez une fonction nommée ne prenant pas de paramètre
3. Dans le corps de la fonction, affichez la variable
4. Appelez la fonction



● Exercice 16 : fonctions et scope des variables

Exercice 16b:

1. Déclarez une fonction A ne prenant pas de paramètre
2. Dans le corps de la fonction A:
 - 2.1. Déclarez et initialisez une variable avec la valeur 3
 - 2.2. Déclarez une fonction B ne prenant pas de paramètre
3. Dans le corps de la fonction B:
 - 3.1. Déclarez et initialisez une variable de même nom que la précédente mais avec la valeur 2
 - 3.2. Affichez cette variable
4. De nouveau dans le corps de la fonction A:
 - 4.1. Appelez la fonction B
 - 4.2. Affichez la variable
5. Appelez la fonction A
6. Qu'est-ce qui est mis en évidence ?



● Exercice 16 : fonctions et scope des variables

Exercice 16c:

1. Déclarez une fonction nommée ne prenant pas de paramètre
2. Dans le corps de la fonction, déclarez et initialisez une variable
3. Appelez la fonction
4. Affichez la variable



● Exercice 17 : Fonction récursive

- Il est tout à fait possible qu'une fonction s'appelle elle-même (récursivité)
- En pratique, plutôt rare
- Cas typique: calcul de la fonction factorielle:
 - $1! = 1$
 - $2! = 1 * 2$
 - $3! = 1 * 2 * 3$
 - ...
- Attention à la récursivité infinie (stack overflow): condition d'arrêt



● Exercice 17 : Fonction récursive

Exercice 17a:

1. Déclarez une fonction *factorielle* prenant un paramètre n
2. Dans le corps de la fonction, retournez: $\text{factorielle}(n-1) * n$
3. Affichez l'appel de la fonction *factorielle* avec 3 en paramètre

Exercice 17b:

1. Avant l'instruction de retour, ajouter une condition if:
 - a. Si n est égal à 1, retournez 1



Bases : exceptions, contrôle d'erreurs

Exceptions : se déclenchent lorsque le programme rencontre une erreur exceptionnelle

```
let obj;      // obj est undefined  
obj.a;        // provoque une exception
```



Bases : exceptions, contrôle d'erreurs

Try, catch, finally

```
try {  
    instructions; // instructions susceptibles de provoquer une exception  
} catch (exception) {  
    instructions; // Gestion de l'erreur  
} finally {  
    instructions; // Bloc exécuté dans tous les cas (exception ou non)  
}
```



Bases : exceptions, contrôle d'erreurs

Try, catch, finally

```
try {  
  let str = "Au cas ou ce code provoquerait une exception, je l'ai encadré avec try";  
  console.log(str);  
} catch (exception) {  
  console.log(exception); // On affiche l'erreur  
}  
console.log("Meme si il y a une erreur, on continue l'exécution ici");
```



● Exercice 18 : Exceptions

Exercice 18:

1. Ouvrez un bloc try
2. Dans le corps de ce bloc, provoquez une erreur en appelant un attribut sur une variable undefined
3. Ouvrez le bloc catch associé
4. Dans le corps de ce bloc, affichez l'exception
5. Après ce bloc, affichez un texte quelconque
6. Qu'est-ce qu'a permis cette capture d'exception ?



Bases : les structures de données : Array

- Est un objet
- Ensemble de valeurs ordonnées accessibles par leur index
- L'index du premier élément est 0
- Peut contenir des éléments de nature différente (ex: des number et des strings)
- Documentation : [Array - JavaScript](#)

```
const monTableau = []; // Initialise un tableau vide
const monTableau2 = ["valeur1", "valeur2", "valeur3"];
console.log(monTableau2[0]); // Affiche le premier élément du tableau
console.log(monTableau2.length); // Affiche la taille du tableau
```



Bases : les structures de données : Array

Itérer sur un Array

```
const monTableau2 = ["valeur1", "valeur2", "valeur3"];

for (let i = 0; i < monTableau2.length; i++) {
  console.log(monTableau2[i]);
}

monTableau2.forEach(element => { console.log(element); });
```



Bases : les structures de données : Array

Ajouter / retirer / modifier un élément

```
const monTableau2 = ["valeur1","valeur2","valeur3"];

monTableau2.push("valeur4"); // Ajoute l'élément "valeur4" à la fin

console.log(monTableau2.pop()); // Retire puis affiche le dernier élément

monTableau2[0] = "valeur1-2"; // Modifie le premier élément du tableau
```



Bases : les structures de données : Array

La méthode `map` : itère sur un Array, effectue une opération sur chaque élément et retourne un nouvel Array en résultat

```
const monTableau2 = [1, 2, 3];  
  
// Ajoute 1 à chaque élément du tableau  
const newTab = monTableau2.map(element => element + 1);  
  
console.log(monTableau2); // Affiche [1, 2, 3]  
console.log(newTab); // Affiche [2, 3, 4]
```




Bases : les structures de données : Array

Array à 2 dimensions : création

```
const monTableau = [];  
  
for (let i = 0; i < 4; i++) {  
    monTableau[i] = [];  
    for (let j = 0; j < 4; j++) {  
        monTableau[i][j] = '[' + i + ', ' + j + ''];  
    }  
}
```



Bases : les structures de données : Array

Array à 2 dimensions : affichage du contenu

```
for (let i = 0; i < monTableau.length; i++) {  
  for (let j = 0; j < monTableau[i].length; j++) {  
    console.log(monTableau[i][j]);  
  }  
}
```



● Exercice 19 : Array

Exercice 19:

1. Déclarez et initialisez un tableau d'entiers contenant 5 entiers dans cet ordre: `9, 1, 3, 6, 2`
2. Affichez la longueur
3. Affichez le dernier élément
4. Ajoutez l'entier 4 au tableau
5. Itérez sur le tableau avec une boucle for pour ajouter 2 à chaque élément
6. Modifiez l'élément à l'index 3 pour lui ajouter 1
7. Affichez le tableau
8. Déclarez une variable *somme* et initialisez la à 0
9. Itérez sur le tableau avec une boucle forEach pour ajouter la valeur de chaque élément à la variable *somme*
10. Affichez la variable *somme*
11. Appelez la méthode `sort()` sur le tableau
12. Affichez le tableau



Bases : les structures de données : Set

- Est un objet
- Ensemble de valeurs dont l'unicité dans le set est garantie
- Documentation : [Set - JavaScript](#)

```
let monSet = new Set();

monSet.add(1);           // [1]
monSet.add(2);           // [1, 2]
monSet.add(2);           // [1, 2]

monSet.has(1);           // true
monSet.has(3);           // false

monSet.size;             // 2
monSet.delete(2);        // supprime 2 du set
```



Bases : les structures de données : Set

Itérer sur un Set

```
monSet.forEach(element => console.log(element));  
  
for (let element of monSet) {  
    console.log(element);  
}
```



Bases : les structures de données : Set

Conversions entre Set et Array

```
const monSet = new Set([1, 2, 3]);  
const monTableau = Array.from(monSet);
```



● Exercice 20 : Set

Exercice 20a:

1. Déclarez et initialisez un Set
2. Ajoutez les strings suivantes dans le set: Javascript, Java, Node.js
3. Affichez la taille du set
4. Ajoutez la string Javascript
5. Affichez le set
6. Convertissez le set en tableau (Array) et affichez-le

Exercice 20b:

1. Déclarez et initialisez un Set
2. Ajoutez deux objets ayant un seul attribut identique (ex: { id: 2 });
3. Affichez le set



Bases : les structures de données : Map

- Est un objet
- Ensemble de valeurs accessibles par leur clé, une clé étant unique
- Documentation : [Map - JavaScript](#)

```
let maMap = new Map();

maMap.set("key1", "value1");
maMap.set("key2", "value2");

maMap.size; // 2

maMap.get("key1"); // retourne "value1"
```




Bases : les structures de données : Map

Itérer sur une Map

```
maMap.forEach((valeur, cle) => {  
  console.log(cle + ' => ' + valeur);  
});
```



● Exercice 21 : Map

Exercice 21:

1. Déclarez et initialisez une Map
2. Ajoutez une nouvelle entrée avec l'entier 1 comme clé, et comme valeur un objet avec les attributs suivants:
 - nom: Damien
 - age: 29
3. Ajoutez une nouvelle entrée avec l'entier 2 comme clé, et comme valeur un objet avec les attributs suivants:
 - nom: Laurent
 - age: 51
4. Ajoutez 2 à l'age de l'objet ayant la clé 1
5. Affichez la map



Bases : les structures de données : Intérêt Set / Map / Array

- **Array:**
 - Structure simple pour stocker un ensemble d'élément
 - Si on cherche un élément spécifique dans le tableau, il faut le parcourir en entier
- **Set:**
 - Ensemble d'éléments dont l'unicité est garantie
 - On peut vérifier directement la présence d'un élément (has)
- **Map:**
 - Structure de forme clé - valeur. L'unicité de la clé est garantie.
 - On peut accéder directement à un élément si on connaît la clé (get)



Bases : Template strings

- String au sein de laquelle il est possible d'évaluer des expressions
- Délimitées par le caractère ` (accent grave)
- Multiligne
- Plus élégant que la concaténation de Strings

```
const a = 1;
const b = 2;
console.log("est-ce que "+a+" est égal à "+b+" ? "+ (a === b)); // concatenation
console.log(`est-ce que ${a} est égal à ${b} ? ${a === b}`); // template string
```



● Exercice 22 : Template string

Exercice 22:

1. Déclarez et initialisez un tableau contenant 2 objets ayant chacun les attributs suivants:
 - *nom* avec une chaîne de caractères de votre choix
 - *age* avec un nombre de votre choix
2. Itérez sur le tableau et pour chaque élément, utilisez une template string pour afficher la phrase: "Cet élève s'appelle *[nom]* et a *[age]* ans" (*[nom]* et *[age]* doivent être remplacés par les valeurs de ces attributs pour l'élément courant)



Bases : les expressions régulières

- Est un objet
- Outil puissant présent dans tous les langages de programmation
- Sert à rechercher des motifs dans des Strings
- Documentation : [RegExp - JavaScript](#)
- [Pour tester les Regexp](#)

```
const maRegExp = /\w+\s/g;  
const str = "hello javascript world";  
  
const matchResult = str.match(maRegExp);  
  
console.log(matchResult); // ["hello ", "javascript "]
```



Bases : les expressions régulières

```
const regex = new RegExp('abc', 'g');  
const str = "testabcjavabc";  
  
const result = str.replace(regex, '');  
  
console.log(result); // "testjav"
```



Bases : programmation orientée objet (POO)

- À la base, le Javascript permet déjà de créer des objets avec le mécanisme de prototypage
- Les classes ont été ajoutées à posteriori par le standard ES6 pour s'aligner sur les autres langages de programmation
- Une classe est un modèle permettant de définir une structure pour ensuite instancier (créer) des objets qui appartiendront à cette classe
- Chaque classe a un constructeur permettant d'instancier des objets de la classe
- Le mot-clé *new* permet d'appeler le constructeur
- Les attributs sont des variables appartenant à l'objet
- Les méthodes d'instance sont des fonctions appartenant à l'objet
- Le mot-clé *this* désigne l'objet
- Par convention, le premier caractère du nom de la classe est écrit en majuscule

Bases : programmation orientée objet (POO)

```
class Person {  
  constructor(nom, age) {  
    this.nom = nom;  
    this.age = age;  
  }  
}
```

```
let person1 = new Person("Damien", 27);  
let person2 = new Person("Laurent", 42);
```

```
let person = { nom: "Damien", age: 27 };
```

```
let person2 = Object.create(person);  
person2.nom = "Laurent";  
person2.age = 42;
```

Bases : programmation orientée objet (POO)

```
class Person {  
  constructor(nom, age) { // constructeur  
    this.nom = nom; // définit un attribut "nom"  
    this.age = age; // définit un attribut "age"  
  }  
  
  salutations() { // Ceci est une méthode d'instance  
    return `Bonjour, je suis ${this.nom} et j'ai ${this.age} ans`;  
  }  
}  
  
let person1 = new Person("Damien", 27); // Instanciation d'un objet de la classe Person  
let person2 = new Person("Laurent", 42); // Instanciation d'un objet de la classe Person  
person1.nom // Accès à un attribut  
person1.salutations(); // Appel de méthode (Bonjour, je suis Damien et j'ai 27 ans)
```



Bases : POO : héritage

- Les classes peuvent hériter d'autres classes (mot-clé *extends*)
- On parle de classe parente et de sous-classe
- Une sous-classe hérite des attributs de la classe parente
- Une sous-classe hérite des méthodes d'instance de la classe parente
- Une sous-classe peut définir ses propres attributs
- Une sous-classe peut définir ses propres méthodes d'instance
- Une sous-classe peut redéfinir les méthodes d'instance de la classe parente
- Une sous-classe peut appeler une méthode parente avec le mot-clé *super*

Bases : POO : héritage

```
class Person {
    constructor(nom, age) {
        this.nom = nom;
        this.age = age;
    }
    salutations() {
        return `Bonjour, je suis ${this.name} et j'ai ${this.age} ans`;
    }
}

class Eleve extends Person { // La classe Eleve hérite de la classe Person, c'est une sous-classe
    constructor(nom, age, etablissement) {
        super(nom, age); // Appel du constructeur de la classe parente (Person)
        this.etablissement = etablissement; // Attribut spécifique à la classe Eleve
    }
    salutations() { // méthode d'instance de la classe Person redéfinie
        return `Bonjour, je suis ${this.name}, j'ai ${this.age} ans et j'étudie à ${this.etablissement}`;
    }
    secherLesCours() { } // méthode d'instance spécifique à Eleve
}
```



● Exercice 23 : le module objet

Exercice 23:

1. Définissez une classe *Vehicule* qui possède:
 - un attribut *nom*
 - un attribut *vitesseMax*
 - une méthode *seDeplacer* qui affiche le texte "se déplace"
2. Définissez une classe *Voiture* qui hérite de la classe *Vehicule* et qui possède:
 - un attribut *immatriculation*
 - une méthode *affichage* qui retourne le texte "Mon immatriculation est " suivi de l'immatriculation
 - redéfinit la méthode *seDeplacer* qui affiche le texte "roule"
3. Instanciez un objet de la classe *Vehicule*
4. Appelez la méthode *seDeplacer* de cet objet
5. Instanciez un objet de la classe *Voiture*
6. Appelez et affichez les méthodes *seDeplacer* et *affichage*
7. Modifiez l'immatriculation de la voiture



● Exercice 24 : Pour tout combiner...

1. Définissez une classe *Personne* qui possède:
 - un attribut *nom*
 - un attribut *age*
 - une méthode *vieillir* qui incrémente l'age de 1
 - une méthode *categorie* qui retourne "enfant" si l'age est inférieur à 11, "adolescent" si l'age est compris entre 11 et 17 inclus, "jeune" si l'age est compris entre 18 et 29 inclus, "fleur de l'age" entre 30 et 49 inclus, "senior" sinon.
 - une méthode *anneesAvantRetraite* qui retourne 70 - age si l'age est inférieur à 70, sinon 0
 - une méthode *presentation* qui retourne le texte: "Bonjour, je suis [*nom*] et j'ai [*age*] ans, plus que [*anneesAvantRetraite*] ans et je serai enfin peinard"
2. Définissez une classe *Eleve* qui hérite de la classe *Personne* et qui possède:
 - un attribut *etablissement*
 - un attribut *notes* (qui est initialisé avec un tableau vide dans le constructeur)
 - une méthode *recevoirNote* qui prend en paramètre un entier et qui l'ajoute au tableau des notes
 - une méthode *moyenne* qui calcule et retourne la moyenne des notes de l'élève
 - redéfinit la méthode parente *presentation* en appelant la méthode parente avec *super* puis en lui ajoutant le texte ", j'étudie à [*etablissement*]"



● Exercice 24 : Pour tout combiner...

Exercice 24 (suite):

1. Instanciez une personne proche de la retraite
2. Instanciez une personne à la retraite
3. Appelez et affichez le résultat de la méthode *presentation* pour ces deux personnes
4. Instanciez un élève (âge raisonnable, pas encore à la retraite)
5. Ajoutez-lui 5 notes différentes (des entiers)
6. Appelez et affichez le résultat des méthodes *moyenne* et *presentation*
7. Utilisez une boucle *while* et la méthode *vieillir* pour envoyer l'élève à la retraite, il a assez travaillé
8. Définissez une fonction *comparer* qui prend deux personnes en paramètres et qui retourne *true* si les deux personnes sont identiques, *false* sinon. Deux personnes sont identiques si:
 - Elles ont le même nom et le même âge
9. Créez deux nouvelles personnes identiques et appelez la fonction *comparer*