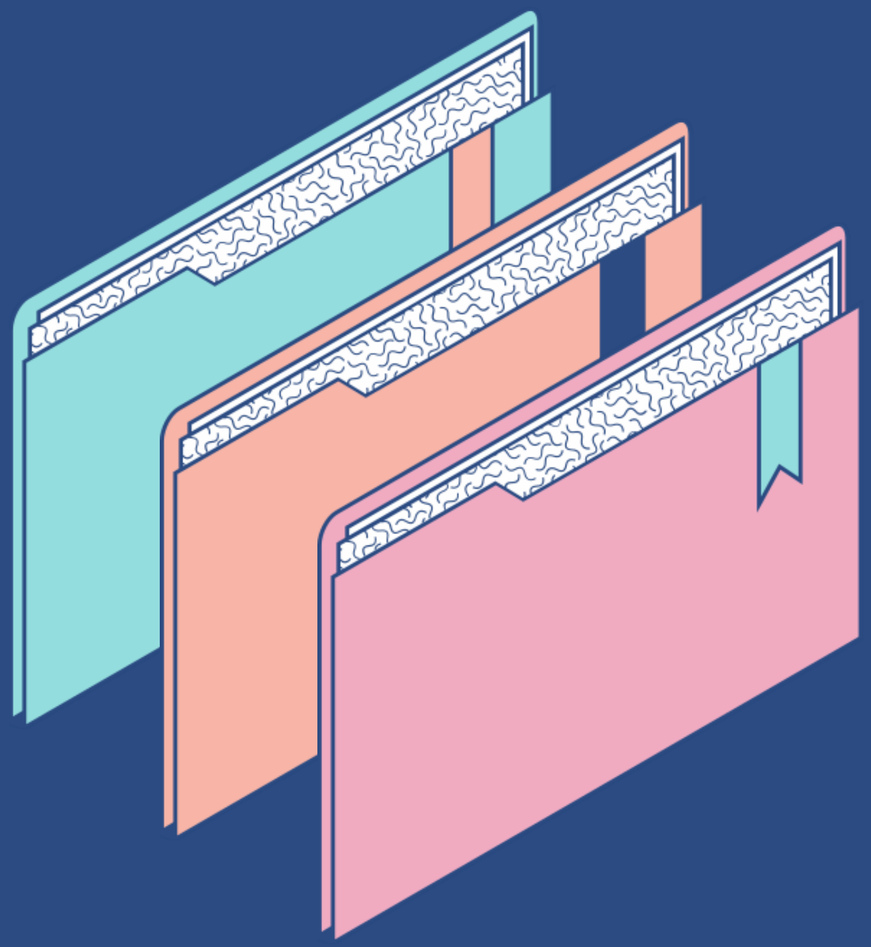
Javascript


JavaScript
<script>
if(age > 19){
alert("Adult");
} else{
alert("Teenager");
}
</script>

SOMMAIRE



Programmation Orientée Objets


```
class Chien{

    constructor(nom, age, race)
    {
        this.nom = nom;
        this.age = age;
        this.race = race;
    }

    toString()
    {
        return "Nom : " + this.nom + " - Age : " + this.age + " - Race : " + this.race;
    }
}

var c1 = new Chien("Pam",15,"Yorkshire");
var c2 = new Chien("Mina",5,"Yorkshire");
var c3 = new Chien("Hok",12, "Jack Russell");

console.log(c1.toString());
console.log(c2.toString());
console.log(c3.toString());
```

Présentation des objets

Un objet est un ensemble cohérent de données et fonctionnalités fonctionnant ensemble.
C'est un "conteneur" contenant des propriétés (variables) et des méthodes (fonctions).

La programmation orientée objet (POO) est une façon d'organiser son code en regroupant des éléments cohérents au sein d'objets.

Le JS possède des objets natifs, possédant des propriétés et méthodes que l'on pourra utiliser.

Présentation objets et POO

Il existe 4 manières différentes de créer des objets en JS :

- Créer un objet littéral
- Utiliser un constructeur Object()
- Utiliser une fonction constructeur personnalisée
- Utiliser la méthode create()

Objets littéraux

```
/*"pierre" est une variable qui contient un objet. Par abus de langage,
 *on dira que notre variable EST un objet*/
let pierre = {
    //nom, age et mail sont des propriétés de l'objet "pierre"
    nom : ['Pierre', 'Giraud'],
    age : 29,
    mail : 'pierre.giraud@edhec.com',

    //Bonjour est une méthode de l'objet pierre
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
};
```

Objets littéraux

Un objet est littéral si chacune de ses propriétés et de ses méthodes est définie lors de la création.

La création d'objet littéral se fait grâce aux accolades { } qui indiquent la création de l'objet.

Très généralement, les objets littéraux sont stockés au sein d'une variable.

On peut accéder aux propriétés d'un objet avec la syntaxe "objet.propriété"

Propriétés des objets

```
/*"pierre" est une variable qui contient un objet. Par abus de langage,
 *on dira que notre variable EST un objet*/
let pierre = {
    //nom, age et mail sont des propriétés de l'objet "pierre"
    nom : ['Pierre', 'Giraud'],
    age : 29,
    mail : 'pierre.giraud@edhec.com',

    //Bonjour est une méthode de l'objet pierre
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
};

/*On accède aux propriétés "nom" et "age" de "pierre" et on affiche leur valeur
 *dans nos deux paragraphes p id='p1' et p id='p2'.
 *A noter : "document" est en fait aussi un objet, getElementById() une méthode
 *et innerHTML une propriété de l'API "DOM"!*/
document.getElementById('p1').innerHTML = 'Nom : ' + pierre.nom;
document.getElementById('p2').innerHTML = 'Age : ' + pierre.age;

//On modifie la valeur de la propriété "age" de "pierre"
pierre.age = 30;

document.getElementById('p3').innerHTML = 'Nouvel âge : ' + pierre.age;

/*On accède à la méthode "bonjour" de l'objet "pierre" qu'on exécute de la même
 *même façon qu'une fonction anonyme stockée dans une variable*/
pierre.bonjour();
```

Accéder à la valeur d'une propriété :
objet.propriété

Présentation POO

La programmation orientée objet est une façon de coder basée sur le concept d'objets.

La POO est utile dans des projets de grande envergure, permettant de créer de multiples objets semblables de manière dynamique.

La POO va permettre de créer un "schéma" à partir duquel l'on pourra créer des objets similaires.

Fonctions constructeurs d'objets

Une fonction constructeur d'objets est un fonction permettant de créer des objets semblables.

Cette fonction pourra elle même définir un ensemble de propriétés et de méthodes.

L'appel à cette fonction constructeur s'effectuera à l'aide du mot clé "new".

Fonctions constructeurs d'objets

```
//Utilisateur() est une fonction constructeur
function Utilisateur(n, a, m){
    this.nom = n;
    this.age = a;
    this.mail = m;

    this.bonjour = function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
}

let pierre = new Utilisateur(['Pierre', 'Giraud'], 29, 'pierre.giraud@edhec.com');
let mathilde = new Utilisateur(['Math', 'Ml'], 27, 'math@edhec.com');
let florian = new Utilisateur(['Flo', 'Dchd'], 29, 'flo.dchd@gmail.com');
```

Classes JS

```
class Ligne{
    /*Nous n'avons pas besoin de préciser "function" devant notre constructeur
     *et nos autres méthodes classe*/
    constructor(nom, longueur){
        this.nom = nom;
        this.longueur = longueur;
    }

    taille(){document.getElementById('p1').innerHTML +=
        'Longueur de  ' + this.nom + ' : ' + this.longueur + '<br>'};
}

let geo1 = new Ligne('geo1', 10);
let geo2 = new Ligne('geo2', 5);
geo1.taille();
geo2.taille();
```

Exercice d'application n°4 :
POO

Créez une classe Complexe construisant les attributs reel et imaginaire,
Créez le constructeur de la classe, ainsi que les methods d'accès (getters et setters)
Ajoutez une méthode toString(), qui retourne le nombre sous la forme a+b*i,.
Ajoutez les methods ajouter(Complexe), soustraire(Complexe), multiplier(Complexe), diviser(Complexe)

Ecrivez un script permettant de tester la classe Complexe.

```
le premier nombre complexe est :
3+1*I
le deuxieme nombre complexe est :
1+2*I
L'Addition des deux nombres renvoie :
4+3*I
La soustraction des deux nombres renvoie :
2-1*I
La multiplication des deux nombres renvoie :
1+7*I
La division des deux nombres renvoie :
1-0.2*I
```

Addition : $z1+z2=(a+x)+i*(b+y)$
Soustraction : $z1-z2=(a-x)+i*(b-y)$
Produit : $z1*z2= (a*x)-(b*y)+i*(x*b+a*y)$
Division : $z1/z2=((a*x+b*y)/(x^2+y^2)) +i *((b*x-a*y)/(x^2+y^2))$