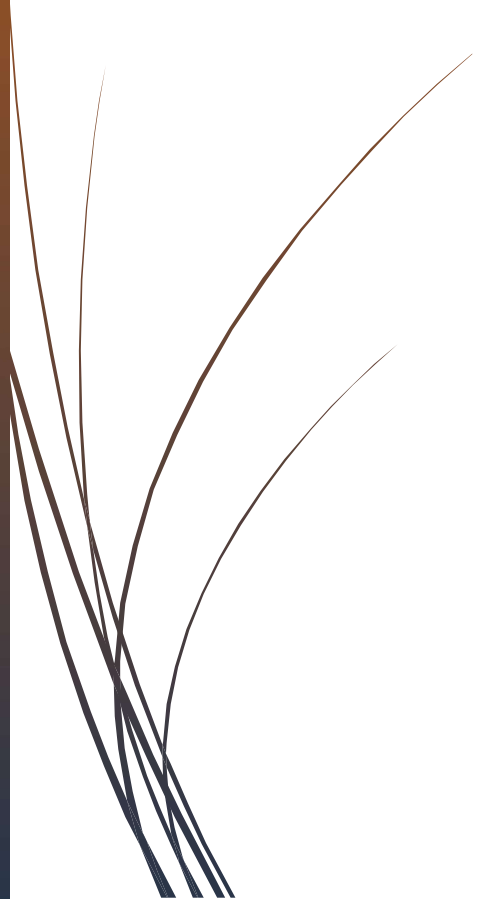
13/06/2023

JAVASCRIPT

Exercices de mise en application

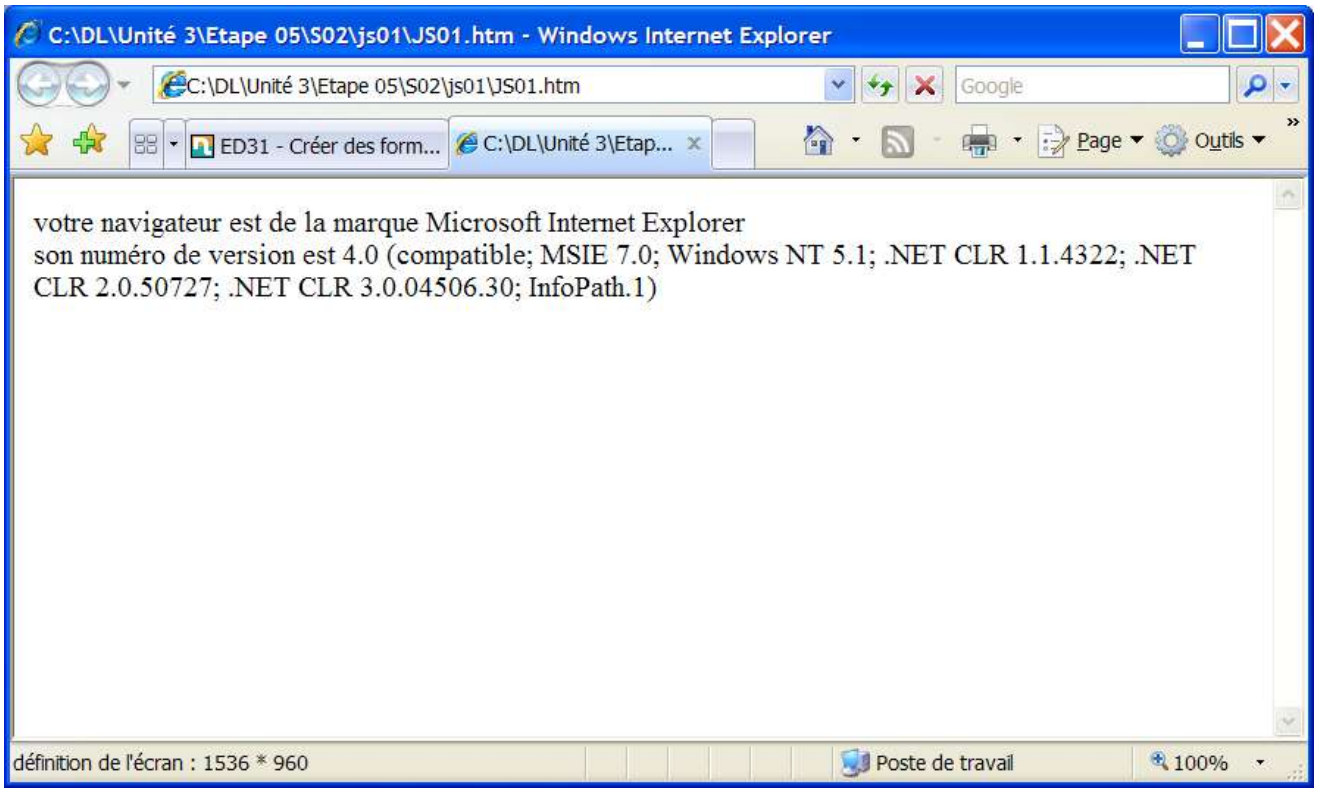


Rudy Lesur
SOFIP

Table des matières

I Affichage du navigateur

Affichez dans une page HTML, le nom de votre explorateur, son numéro de version, ainsi que les caractéristiques de définition de l'écran dans la zone status, comme suit :



Points abordés :

La méthode write() de l'objet document.
L'objet navigator et ses propriétés (appName, appVersion….)
L'objet window et sa propriété status.
L'objet screen et ses propriétés (width, height).

1/ Un peu de théorie

Window est l'objet JavaScript de plus haut niveau dans la hiérarchie d'une page HTML. C'est lui qui possède le plus de propriétés, de sous-objets et de méthodes. Il est le seul à autoriser une écriture simplifiée de ses propriétés et méthodes. Par exemple JS identifie le sous-objet *document* de *window* aussi bien avec l'écriture complète *window.document* qu'avec *document*. Cette particularité a un avantage sur la rapidité d'écriture.

1) L'objet navigator

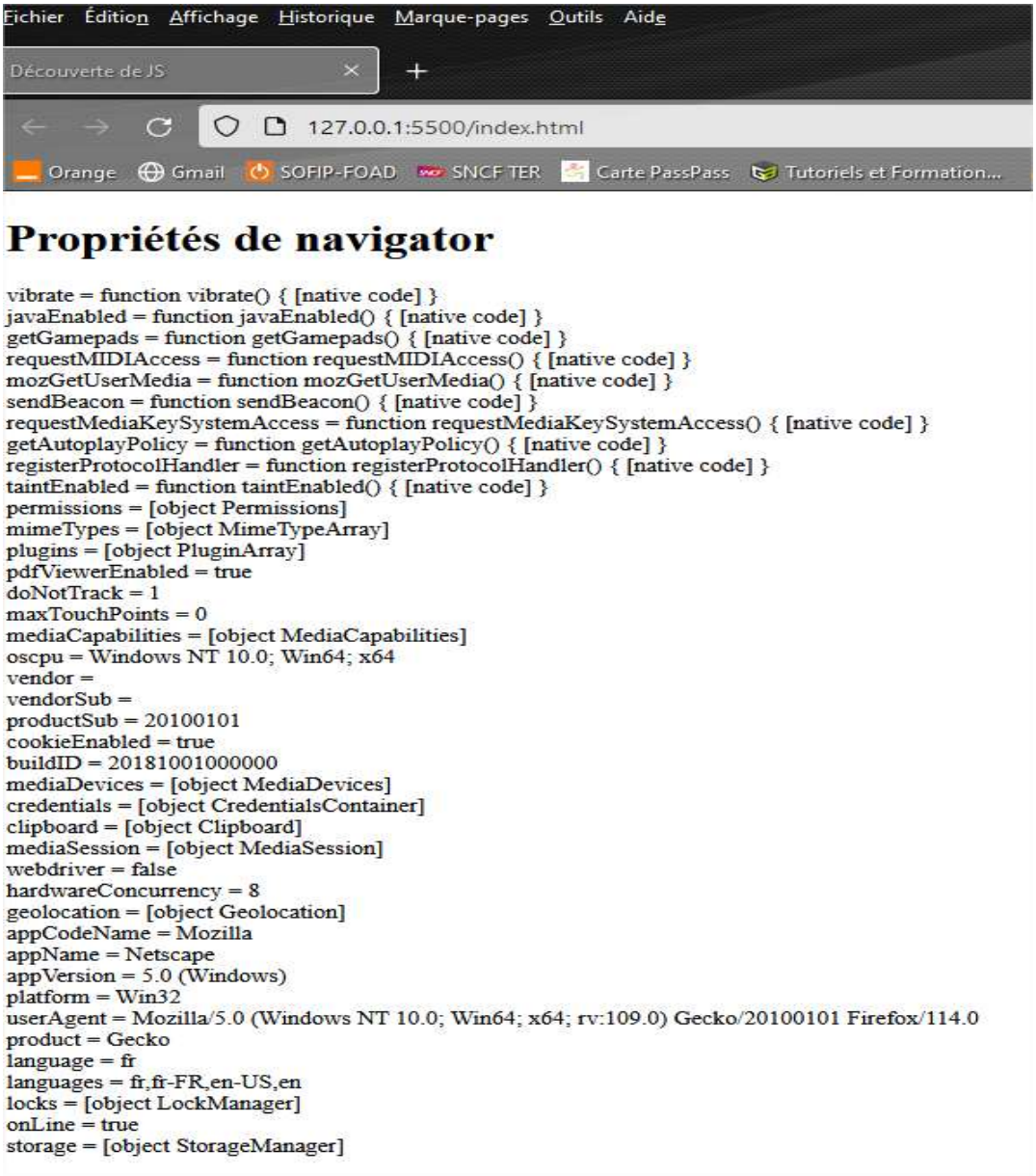
Il contient de nombreuses informations sur le navigateur en cours d'utilisation.
Il existe des différences de comportement entre navigateurs, il est donc difficile de repérer simplement l'application utilisée par le navigateur.

A l'aide de la boucle *for…in*, nous pouvons extraire toutes les propriétés de cet objet et leurs valeurs dans 2 navigateurs principaux :

```
<body>
  <p><h1> Propriétés de navigator</h1></p>
<script type="text/javascript">
  for (i in window.navigator) {
        document.write(i+" = "+window.navigator[i]+"<br/>");
        }
  </script>
  </body>
```

CHROME :

FIREFOX :



7 propriétés sont compatibles.
userAgent contient le plus d'informations. C'est cette propriété qui nous permettra au mieux d'identifier un navigateur.
Le nom de code de l'application donné par ***appCodeName*** est identique pour les 2 navigateurs !
La version du navigateur donnée par ***appVersion*** n'est pas sous la forme d'un nombre mais sous une chaine de caractères complexe.
La propriété ***language*** n'est pas reconnue par IE qui utilise 2 propriétés ***systemLanguage*** et ***userLanguage*** pour la remplacer. Il est possible de gérer un site multilingue avec ces propriétés et d'afficher directement la bonne page au visiteur en fonction de la langue de son navigateur.
platform indique le type de système d'exploitation. Elle peut être utile pour proposer des téléchargements adaptés.
cookieEnabled vaut true si les cookies sont activés.

2) L'objet screen

Le sous-objet *screen* lié à *window* possède des propriétés sur la résolution de l'écran du visiteur.

Propriétés :

screen.height et screen.width donnent le nombre de pixels affichés à l'écran en hauteur et en largeur.

screen.availWidth et screen.availHeight donnent le nombre de pixels disponibles pour les applications.

screen.colorDepth qui indique la profondeur de couleurs en bits (nombre de couleurs distinctes affichables)

math.pow(2,screen.colorDepth) donne le nombre de couleurs.

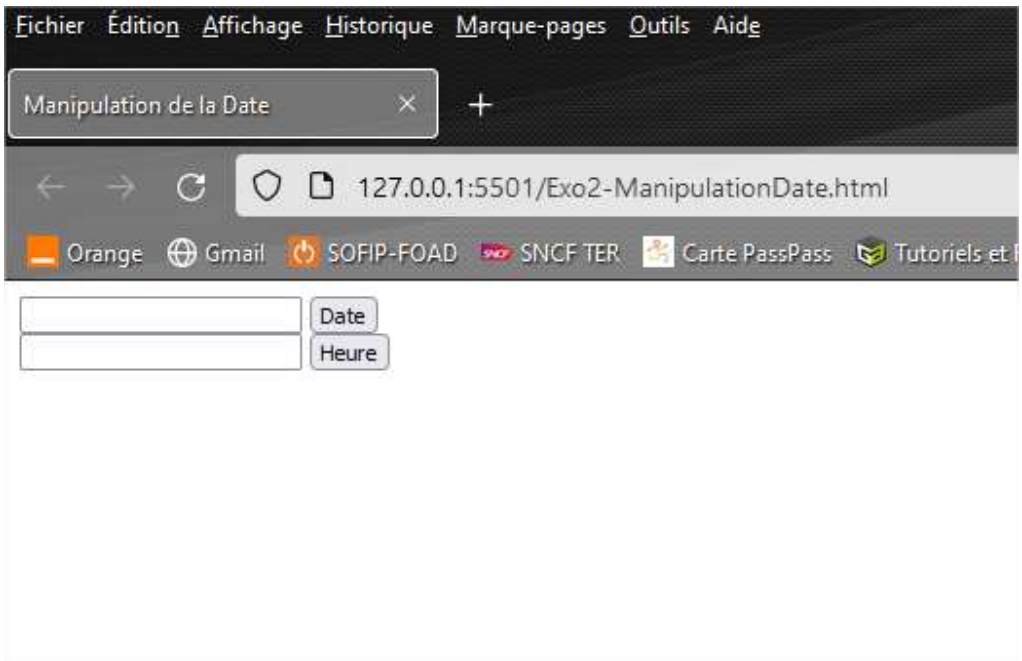
Cet objet permet donc de connaître la résolution de l'internaute. Grâce à ces informations, nous pouvons déterminer l'équipement de nos visiteurs et optimiser les informations à l'écran pour limiter au maximum les ascenseurs horizontaux.

II Manipulation de la date

Affichez la page HTML suivante.

Lorsque l'on clique sur le bouton date, la date système s'affiche dans le champ correspondant.

Lorsque l'on clique sur le bouton heure, l'heure système s'affiche dans le champ correspondant.



Points abordés :

L'objet Date et ses méthodes : getDate(), getMonth(), getHours()…….
La création de fonctions JavaScript.
Gestion d'événements : onClick.

1/ Un peu de théorie

1) La programmation événementielle

JavaScript est un langage événementiel, c'est-à-dire que son exécution peut être déclenchée par des événements qui surviennent sur la page. Ils consistent en un changement de l'état de la page. Il en existe plusieurs.

Parmi les plus courants nous trouvons :

- Le déplacement de la souris
- Un clic, sur un bouton ou un lien
- Le chargement d'une page
- Le survol de la souris au-dessus d'un élément HTML
- La modification d'une zone de saisie
- Un appui sur une touche du clavier

La plupart des événements sont déclenchés par une action de l'utilisateur à partir des périphériques de saisie connectés à son ordinateur (clavier, souris).
La programmation événementielle est idéale pour la gestion de l'interactivité mais elle impose des contraintes importantes au développeur. Le visiteur est complètement libre dans sa navigation et rien ne l'oblige à suivre l'ordre de saisie prévu. Il faut donc s'assurer que la fonctionnalité d'une page est bien remplie : le visiteur a-t-il bien respecté l'ordre logique, a-t-il oublié des étapes ou inversé des étapes ?

Pour indiquer au navigateur l'action à réaliser en fonction de l'événement, il faut utiliser les attributs de type *onEvenement* des balises HTML.
Par exemple, pour détecter un clic sur un lien :

```
<a href="#" onclick="alert('Bonjour') ">Cliquez sur le lien</a>
```

L'attribut *onclick* contient le code JavaScript à exécuter. Il est possible de définir plusieurs instructions pour le même événement en utilisant le séparateur d'instructions JavaScript à savoir le point-virgule.
Pour des raisons de lisibilité, il est préférable de regrouper les instructions au sein d'une fonction. Cela permettra de récupérer la fonction pour un autre événement et d'éviter des erreurs dans l'alternance des guillemets et des simples quotes.

```
<a href="#" onclick="message('Bonjour') ">Cliquez sur le lien</a>

<script>
        function message(texte) {
                alert(texte) ;
        }
</script>
```

2) Les différents événements :

onAbort : quand l'utilisateur appuie sur Echap

onBlur : quand un objet perd le focus

onChange : quand le curseur quitte la change et que son contenu a été modifié

onClick : quand on click sur l'élément

onContextMenu : avant l'apparition d'un menu contextuel avec clic droit

onDblClick

onDragDrop : détecte un glisser-déposer

onError : permet de capturer les erreurs JavaScript qui surviennent

onFocus : survient quand un élément reçoit le focus

onKeyDown : se produit quand l'utilisateur appuie sur une touche du clavier

onKeyPress : se déclenche quand l'utilisateur relâche la touche du sur laquelle il a appuyé

onKeyUp : se déclenche au moment où la touche est relâchée

onLoad : est déclenché quand une page est entièrement chargée

onMouseDown : déclenché au moment où le visiteur appuie sur l'un des boutons de la souris (gauche, droit ou les deux en même temps)

onMouseMove : déclenché en permanence pendant les mouvements de la souris

onMouseOut : déclenché quand le curseur de la souris quitte un objet

onMouseOver : déclenché quand le curseur de la souris survole un objet

onMouseUp : déclenché au moment où un bouton de la souris est relâché

onReset : avec la balise <form>. Se déclenche quand le formulaire est réinitialisé

onSelect : quand l'utilisateur sélectionne une zone de texte

onSubmit : avec la balise <form>. Déclenché à la soumission du formulaire

onUnload : quand la page courante est remplacée par une autre ou à la fermeture du navigateur.

Les fonctions

Les fonctions sont un moyen de coder une seule fois une fonctionnalité et de réutiliser ce code en cas de besoin. Les fonctions sont à la base du développement durable. L'idée est de se constituer un ensemble de fonctions et de les réutiliser dans chaque contexte où elles sont nécessaires.
JS nous permet d'écrire nos propres fonctions personnalisées.
La syntaxe est la suivante :

```
function nomDeLaFonction(param1, param2, …, paramN) {
    // instructions
}
```

Une fonction attend des paramètres. L'ensemble du code source composant le groupe d'instructions est appelé le corps de la fonction.

Pour exécuter une fonction dans un script, il suffit d'écrire son nom suivi, entre parenthèses, des différents paramètres nécessaires à son fonctionnement. Une fois définie, une fonction peut être appelée autant de fois que nécessaire.
Une fonction peut ne pas avoir de paramètre. Même dans ce cas, les parenthèses sont obligatoires, dans la définition de la fonction et dans son appel.

Une fonction ne retourne pas toujours de valeur. L'instruction *return* n'est pas obligatoire.

Conseil : il est toujours préférable de définir une fonction pour regrouper le code source présent à plusieurs endroits. La maintenance en est facilitée : vous ne modifiez qu'une seule fois le code. De plus, vous pourrez réutiliser votre fonction dans d'autres pages ou dans d'autres projets.

3) L'objet DATE

L'objet *Date* est une structure de données qui contient à la fois une date et une heure précises.
L'année est exprimée sur chiffres.

Le mois est exprimé par un nombre compris entre 0 et 11. L'indice 0 correspond à janvier et 11 à décembre.

Le jour du mois varie de 1 à 31.

Le jour de la semaine est exprimé par un nombre compris entre 0 et 6. L'indice 0 correspond au dimanche et 6 au samedi.

L'heure varie de 0 à 23, les minutes et les secondes de 0 à 59.

Les millisecondes varient de 0 à 999.

Le fuseau horaire correspond au fuseau horaire défini dans les paramètres de l'horloge du poste client.

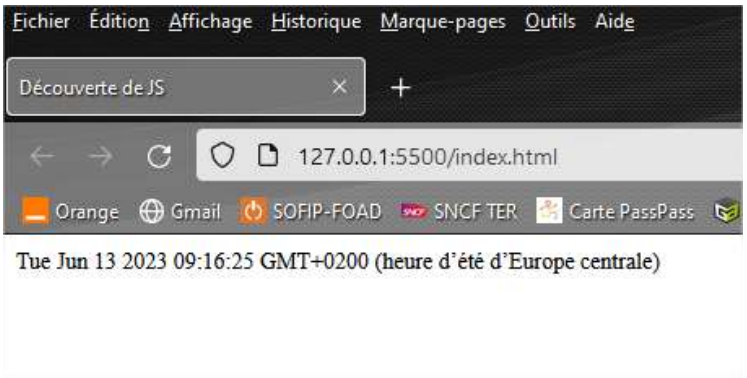
La création d'un objet *Date* est réalisée avec le constructeur *Date()* :

let dtJour = new Date() ;

Après exécution, *dtJour* est un objet *Date* contenant la date et l'heure de l'instant de création.
Le code suivant :

```
<body>
    <script type="text/javascript">
        let dtJour = new Date();
        document.write(dtJour);
    </script>
</body>
```

Affichera comme résultat :



Ce format d'affichage par défaut n'est pas très compréhensible.

L'objet *Date* ne possède pas de propriété. L'accès aux diverses informations se fait par une série de méthodes. Toutes ses méthodes portent le préfixe *get* dans leur nom pour montrer qu'elles retournent un résultat.

getDate()	Retourne le numéro du jour du mois (entre 1 et 31)
getDay()	Retourne le numéro du jour de la semaine (entre 0 et 6)
getFullYear()	Retourne l'année complète sur 4 chiffres. Remplace getYear() non prévu pour le passage à l'an 2000.
getHours()	Retourne l'heure de l'objet (entre 0 et 23)
getMilliseconds()	Retourne le nombre de millisecondes de l'objet (de 0 à 999)
getMinutes()	Retourne les minutes (de 0 à 59)
getMonth()	Retourne l'indice du mois (de 0 à 11)
getSeconds()	Retourne les secondes (de 0 à 59)
getTime()	Retourne le nombre de millisecondes qui séparent la date sur laquelle s'applique la méthode et le 1er janvier 1970. Utile pour déterminer l'écart entre deux dates.
getTimezoneOffset()	Retourne le nombre de minutes séparant l'heure locale et le fuseau horaire de référence.
getYear()	Retourne l'année mais il est conseillé d'utiliser getFullYear()

III Manipulation d'images

Affichez la page HTML suivante.

Lorsque l'on clique sur l'ampoule, l'image change (l'ampoule s'allume si elle est éteinte, s'éteint si elle est allumée).

Les images sont disponibles sous forme de fichiers GIF.

Points abordés :

Gestion des événements : onClick
Ecriture d'une fonction
Utilisation de variables
Tests conditionnels : if

1/ Un peu de théorie Les variables

Tous les langages de programmation s'appuient sur des variables pour stocker et manipuler des informations.
Une variable est une zone mémoire contenant une donnée. Les variables ont des noms nous permettant de les manipuler.
En JS, une variable est définie avec le mot clé *var*.
Nous pouvons déclaration une variable avec ou sans initialisation.

Ex : **var compteur = 0 ;**
var text1 ;
var text2, text3 = ''bonjour'' ;

JS n'est pas très strict : nous pouvons utiliser une variable sans qu'elle ait été déclarée. Cela s'appelle une déclaration implicite.

Le nom d'une variable peut être composé des 26 lettres de l'alphabet en majuscules et en minuscules, des 10 chiffres et du caractère underscore _ (touche 8). Le caractère d'espacement est interdit et un nom ne peut pas commencer par un chiffre.

JS distingue la casse c'est-à-dire les majuscules et les minuscules.

JS est pauvrement typé. Une variable peut contenir indifféremment tous les types de données au sein du même script.
JS possède une instruction qui nous permet de connaître le type de la donnée contenue dans une variable : *typeof*.
Le fait de déclarer une variable sans lui affecter de valeur initiale rend la variable indéfinie. Elle contient la valeur *undefined* et a le type *undefined*.

Le type **number** : concerne toutes les variables contenant un nombre, entier, à virgule, positif ou négatif, petit ou grand.
Le type **string** : concerne toutes les variables contenant une chaine de cractères. Une chaine de caractères est délimitée par les guillemets (‘’)
Le type **boolean** : peut contenir 2 valeurs : « vrai » ou « faux » soit *true* ou *false*.

1) Les tests conditionnels

Un programme est une succession d'instructions exécutées les unes à la suite des autres. Les instructions conditionnelles orientent l'exécution vers une partie du programme selon le résultat des tests.

Exemple :

```
if (prixTotal > 50)
        {
        prixTransport = 10 ;
        }
else
        {
        prixTransport = 20 ;
        }
```

L'expression du test s'écrit obligatoirement entre parenthèses.
Le ELSE n'est pas obligatoire.

2) Les opérateurs de tests :

==	égal
!=	différent
>	strictement supérieur
<	strictement inférieur
>=	supérieur ou égal
<=	inférieur ou égal

3) Les combinaisons de tests

il est possible de combiner des tests ensemble pour créer des conditions plus complexes.
Le test logique ET est symbolisé par &&. Le test logique OU est symbolisé par ||.

Les tests multiples

Pour comparer une variable à plusieurs valeurs, nous pouvons utiliser l'instruction **switch**.

Exemple :

```
var nbTour = 2 ;
switch (nbTour) {
        case (1) :    document.write(« Premier tour ») ;
                      break ;
        case (2) :    document.write(« deuxième tour ») ;
                      break ;
        default :     document.write("je ne sais pas");
}
```

nbTour est comparée aux différentes valeurs. Chaque comparaison de valeur est définie par un « *case* » suivi de la valeur à comparer entre parenthèses.

Si le test d'égalité entre cette valeur et la variable à comparer du switch est vérifié, le groupe d'instructions suivant les : est exécuté.

L'instruction **break** marque la fin de ce groupe et arrête toutes les comparaisons définies dans le switch. L'exécution du script reprend immédiatement après l'accolade fermante de l'instruction switch.

Si aucun des tests n'est vérifié, les instructions qui suivent **default** sont exécutées.

Il est impossible de donner plus d'une valeur de comparaison à chaque *case.*

4) L'opérateur ternaire

Il existe pour les tests d'égalité une écriture rapide en une seule ligne qui retourne une valeur si le test est vérifié et une autre valeur sinon.

Syntaxe :

test ? valeur si test vrai : valeur si test faux

exemple :

```
var txtAccueil = heure < 18 ? ''Bonjour'' : ''Bonsoir'' ;
```

cet opérateur permet d'affecter une valeur en évitant l'écriture d'un test conditionnel *if else*.