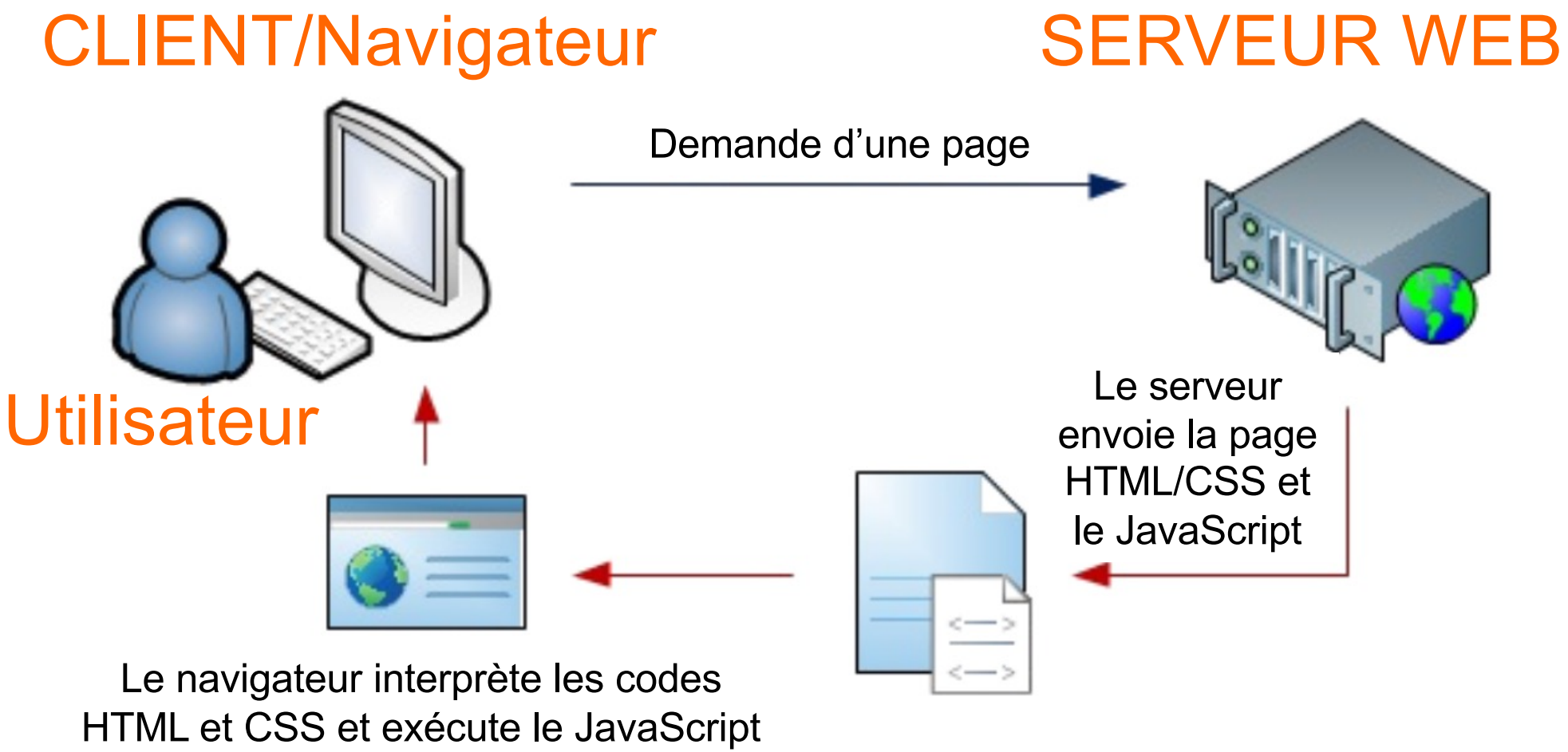
# Langage JavaScript


Jean-Christophe DUBOIS
jean-christophe.dubois@univ-rennes.fr

# Langage de programmation

- Langage de scripts orienté objet
- Langage interprété (*vs* compilé)
  - Code source du script utilisé pour l'exécution
    - Aucune analyse avant interprétation : erreur possible
  - Lisible
- Script client-side (*vs* server-side)
  - Interprété par le navigateur
    - Interpréteur propre à chaque navigateur
      *Chakra/Blink* (MS Edge), *SpiderMonkey* (Mozilla), *V8* (Chrome)
  - Entièrement chargé avec la page web
    - Ré-exécutable sans accès au serveur

# Langage de programmation




*Source : J. Pardanaud & S. de la Marck*

# Langage de programmation

- Langage associé à HTML et CSS
  - HTML : description du contenu
  - CSS : description de la mise en forme
  - JavaScript : programmation d'actions
    - Interaction, animation, aide à la navigation, validation de formulaires, communication avec le serveur…

  en réponse à des événements
    - Chargement de la page, clavier, souris…

# Langage de programmation

- Standardisé sous le nom de ECMAScript
- Langages dérivés
  - JScript, JScript.NET, E4X
  - ActionScript utilisé pour les animations Flash
  - Attention Java n'est pas un langage de script et n'a pas de lien avec JavaScript !
- Dernière version ECMAScript15 (2024)
- Documentation en ligne :
  - https://developer.mozilla.org/fr/docs/Web/JavaScript (fr)
  - https://devdocs.io/javascript/ (en)

# Edition - Visualisation

- Edition
  - Éditeur de texte
    - BlueGriffon (Mac), Brackets, Sublime text, Atom, Visual Studio Code…
  - Logiciel de création de site web
    - Dreamweaver, KompoZer…
- Visualisation
  - Navigateur
    - Chrome, Firefox, Internet explorer, Safari, Opera…
- Test en ligne
  - https://jsfiddle.net/

1ère partie

# Bases du langage


Jean-Christophe DUBOIS
jean-christophe.dubois@univ-rennes.fr

# Règles d'écriture

- Syntaxe
  - Instructions séparées par le caractère ;
- Espace et indentation (tabulation)
  - Non obligatoire
    mais meilleure lisibilité, maintenance facilitée
- Insertion de commentaire

```
// Commentaire sur 1 ligne
/* Commentaire réparti sur
plusieurs lignes */
```

- Règle de nommage : camelCase

# Insertion du code

- Interne à la page
  - En HTML 5, dans l'en-tête <head>, dans le corps du texte

```
<script> … </script>
```

ou, mieux, en fin de page (avant </body>)

  - avec d'anciennes versions de HTML

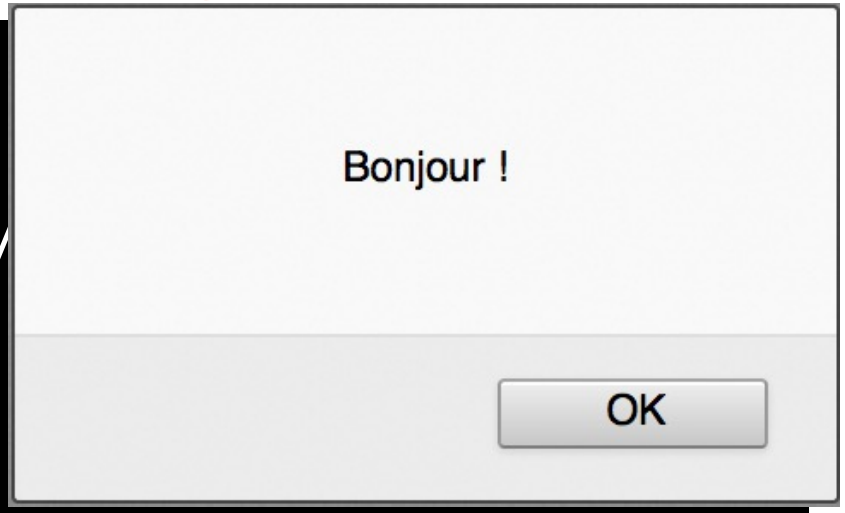
```
<script type="text/javascript" >
   <!-- poids1 > poids2;
   // -->
</script>
```

<!-- isolation du code pour le validateur W3C -->

# Insertion du code

- Exemple d'insertion dans la page HTML

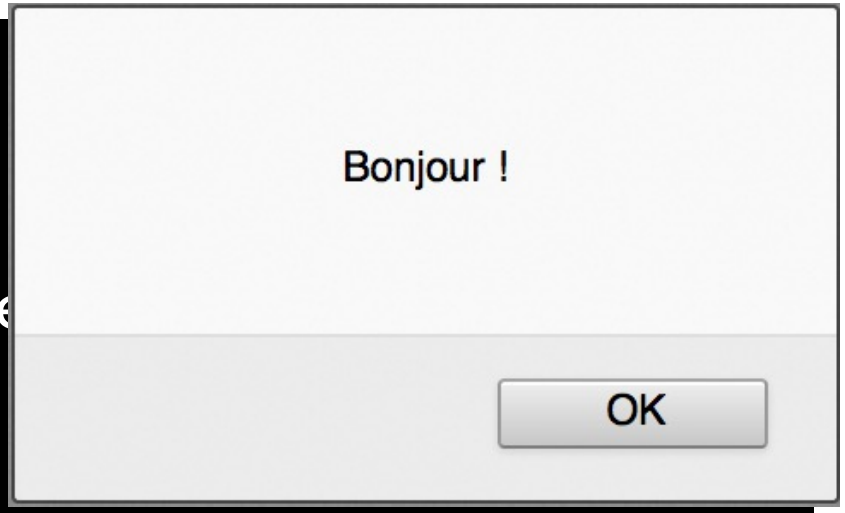
```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello World!</
   </head>
   <body>
      ....
      <script>
        alert("Bonjour !");      // fenêtre avec message
      </script>
      <noscript>
        JavaScript non accepté par le navigateur
      </noscript>
   </body>
</html>
```



# Insertion du code

- Externe à la page (conseillé car fichier chargé une fois)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Bonjour !</title
   </head>
   <body>
      ...
      <script src="fichierscript.js"></script>
   </body>
</html>
```




– Fichier fichierscript.js

```
alert('Bonjour !');
```

# Les variables

- Variable : espace permettant de stocker des données - chaîne de caractères, nombre, valeur booléenne ou autres plus complexes…
- Déclaration : réservation d'un espace de stockage
- Nom (explicite !) :
  - 0..9 (sauf pour le 1er caractère)
  - a..z et A..Z
  - $ et _
  - Différent d'un mot réservé, utilisé par le code JS
  - Sensible à la casse (minuscule/majuscule)

# Les variables

- Type de données : déclaration d'une donnée avec let
  Le typage est dynamique, il est inutile de préciser le type de contenu de la variable
- Type numérique (number) :

```
let maVariable = 3.14;
let maVariable = 1.412e+6;          // 1.412.000
```

- Chaîne de caractères (string) :

```
let maVariable = 'texte';        let maVariable = "texte";
let maVariable = "c\'est une exception";        // échapnt
let maVariable = '8';                          // caractère 8 !
```

- Booléen (boolean) : 2 valeurs, true (vrai) et false (faux)

```
let estVrai = true;                    let estFaux = false;
```

# Les variables

- Déclaration avec let ou avec var *(portée différente)*

```
let maVariable;                    var maVariable;
```

- Affectation avec =

```
maVariable = 2021;
```

- Déclaration et affectation avec let/var et =

```
let maVariable = 2021;     var maVariable = 2021;
```

- Déclaration et affectation multiples avec ,

```
let maVariable1, maVariable2, maVariable3 = 4;
maVariable1 = maVariable2 = 2;
```

# Les variables

- Vérification du type de données

Opérateur typeof pour connaître le type d'une variable

```
let maVariable = 3.14;
console.log(typeof maVariable);          // number
```

Variable inexistante

```
console.log(typeof uneVariable);         // undefined
```

Variable déclarée mais sans affectation

```
let uneVariable;
console.log (typeof uneVariable);        // undefined
```

console.log() : Envoi du message sur la console du navigateur

# Les constantes

- Déclaration d'une constante avec const

```
const TVA;                          // erreur, TVA doit avoir
                                    // une valeur initiale
const TVA = 20;
console.log(typeof TVA);                    // number
const PAYS = "FRANCE";
console.log(typeof PAYS);                   // string
```

- Réassignation d'une constante impossible

```
const TVA = 20;
TVA = 10;                                   // erreur !
```

# Opérateurs arithmétiques

- Opérateurs simples
  - `+` Addition
  - `-` Soustraction
  - `*` Multiplication
  - `/` Division
  - `%` Modulo (reste de la division)
- Opérations simples

```
let res1, res2 = 6;
res1 = 11 % 4;                  // res1 vaut 3
res2 = res2 / res1;             // res2 vaut 2
```

```
let res1, res2;
res2 = res1 = (10 + 5) * 2;     // res1 et res2 valent 30
```

# Opérateurs arithmétiques

- Incrémentation d'une variable

```
let compteur = 3;
compteur = compteur + 2;    // compteur vaut 5
```

- Ecriture simplifiée d'un calcul

```
let compteur = 3;
compteur += 2;                    // compteur vaut 5
```

Opération et assignation possibles avec d'autres opérateurs :

-=  *=  /=  %=

```
let compteur = 9;
compteur %= 2;                    // compteur vaut 1
```

# Opérateurs arithmétiques

- Incrémentation d'une variable

```
let compteur = 0;
compteur++;                          // compteur vaut 1
compteur--;                          // compteur vaut 0
let resultat1 = ++compteur;          // resultat1 vaut 1
                    // incrémentation avant affectation
let resultat2 = compteur--;          // resultat2 vaut 1 !!!
                    // décrémentation après affectation
```

Attention à l'ordre des opérateurs !

# Opérations sur les chaînes

- Concaténation de 2 chaînes de caractères

```
let debutPhrase = "Bonnes ", finPhrase = "vacances";
let maPhrase = debutPhrase + finPhrase;
    // maPhrase contient "Bonnes vacances"
    // ne pas oublier l'espace  pour séparer les mots
```

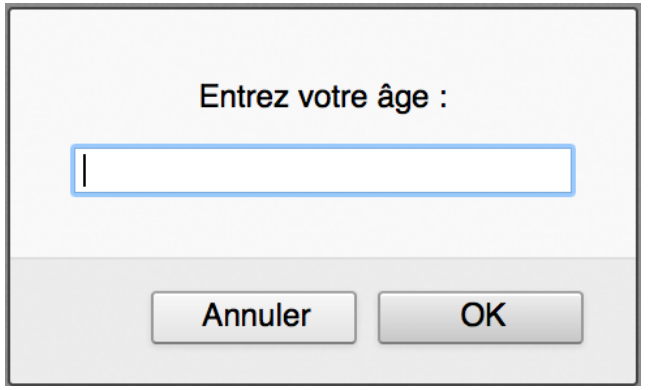
- Concaténation d'une chaîne et d'un nombre

```
let monAge = 20;
let text1 = "Vous avez ", text2 = " ans";
let maPhrase = text1+ monAge + text2;
    // maPhrase contient "Vous avez 20 ans"
```

# Opérations sur les chaînes

- Saisie d'une chaîne de caractères au clavier :

Fonction prompt()




```
let monAge = prompt("Entrez votre âge");
    // monAge contient les caractères saisis "20"
let maPhrase = "Vous avez " + monAge + " ans";
console.log(maPhrase);          // "Vous avez 20 ans"
```

# Opérations sur les chaînes

- Interpolation d'expression – template strings

  Définition d'une chaîne de caractères entre ` (accent grave)

  - Insertion d'une variable insérée avec ${ }.
    Elle est remplacée par sa valeur au moment de l'exécution.

```
let monAge = prompt("Entrez votre âge");
let maPhrase = `Vous avez ${monAge} ans`;
console.log(maPhrase);          // "Vous avez 20 ans"
```

  - Insertion de portions de code

```
let maPhrase = `Année de naissance ${2021-monAge}`;
console.log(maPhrase);   // "Année de naissance 2001"
```

# Opérations sur les chaînes

- Interpolation d'expression – template strings
  - Insertion du résultat d'une fonction
    Exécution de la fonction appelée et insertion de son résultat dans la chaîne de caractères.

```
let maPhrase
        = `Vous avez ${prompt("Entrez votre âge")} ans`;
console.log(maPhrase);          // "Vous avez 20 ans"
```

  - Définition de chaînes de caractères multilignes

```
let maPhrase  = `Vous avez
                 20
                 ans`;
```

# Opérations sur les chaînes

- Conversion d'une chaîne de caractères en nombre :

  Fonction parseInt()

```
let textAge = prompt("Entrez votre âge");
let nbreAge = parseInt(textAge);
console.log(typeof textAge);                    // string
console.log(typeof nbreAge);                    // number
```

```
let nbreAge = parseInt("20");
nbreAge += 1;
console.log(nbreAge);                           // 21
```

  Quel est le résultat sans parseInt() ?

# Opérateurs de comparaison

- Les opérateurs de comparaison

| | | | |
|---|---|---|---|
| == | contenu égal | != | différent |
| > | supérieur | < | inférieur |
| >= | supérieur ou égal | <= | inférieur ou égal |
| === | contenu **et** type égal | !== | contenu **ou** type différent |

```
let nbre1 = nbre2 = 20, nbre3 = "20";
let resultat1 = nbre1 != nbre2;        // false
let resultat2 = nbre1 == nbre3;        // true,  20 == "20"
let resultat3 = nbre1 === nbre3;
    // false, nbre1 est un number et nbre3 un string
let resultat = "false" == false;       // false
```

# Condition if…else

- if (condition) {
  Instruction(s) si la condition est vérifiée;
  } else {
  Instruction(s) si la condition n'est pas vérifiée;
  }

```
let age = 20;
if (age >= 18) {                    // age >= 18 est vérifiée
    alert("Vous êtes majeur");// le message est affiché
} else {
    alert("Vous n'êtes pas encore majeur");
}
```

- La partie else est optionnelle

# Conditions if…else imbriquées

- if (condition1) {
  Instruction(s) si la condition1 est vérifiée;
  } else if (condition2) {
  Instruction(s) si la condition2 est vérifiée;
  } else {
  Instruction(s) si la condition2 n'est pas vérifiée;
  }

```
let age = 20;
if (age < 18) {
    alert("Vous êtes mineur·e");
} else if (age >= 65) {
    alert("Bientôt la retraite !");
} else {
    alert("Au boulot ! ☺ ");
}
```

# let versus var

- La portée de let se limite au bloc

```
let age = 20;
let majeur = 0;
if (age >= 18) {                    // age >= 18 est vérifiée
    let majeur = 1;
    alert("Vous êtes majeur·e : "+ majeur);  // 1
}
alert("Etes-vous majeur·e ? " + majeur);   // 0 est affiché
```

- Les 2 variables majeur sont différentes

# let versus var

- La portée de var est plus grande (fonction)

```
let age = 20;
var majeur = 0;
if (age >= 18) {                    // age >= 18 est vérifiée
    var majeur = 1;
    alert("Vous êtes majeur·e : "+ majeur);  // 1
}
alert("Etes-vous majeur·e ? " + majeur);   // 1 est affiché
```

Attention : La modification de majeur au sein du if affecte la variable majeur définie à l'extérieur !

# let versus var

- La portée de let est plus limitée, est descendante mais est plus facile à comprendre et donc à utiliser

```
let age = 20;
let majeur = 0;
if (age >= 18) {                    // age >= 18 est vérifiée
    majeur = 1;
    alert("Vous êtes majeur·e : "+ majeur);  // 1
}
alert("Etes-vous majeur·e ? " + majeur);   // 1 est affiché
```

- Utilisation de let recommandée pour déclarer des variables

# Opérateurs logiques

- condition1 && condition2 : "et" logique
  - condition1 et condition2 doivent être vraies toutes les 2 pour que la condition générale soit vérifiée
- condition1 || condition2 : "ou" logique
  - L'une ou l'autre des conditions doit être vraie pour que la condition générale soit vérifiée
- !condition : "non" logique
  - Inverse la valeur logique de la condition

```
let age = prompt("Saisir votre âge");
if ((age > 18) && (age < 65))  {
    alert("Au boulot !");
} else {
    alert("L'école ou la retraite, la belle vie quoi  !");
}
```

# Valeurs booléennes

- Tests conditionnels particuliers

  Le contenu d'une variable peut être évalué comme une valeur booléenne et être utilisé dans une condition

  – Un nombre égal à 0 → faux
  – Une chaîne de caractères vide → faux
  – Une variable indéfinie → faux

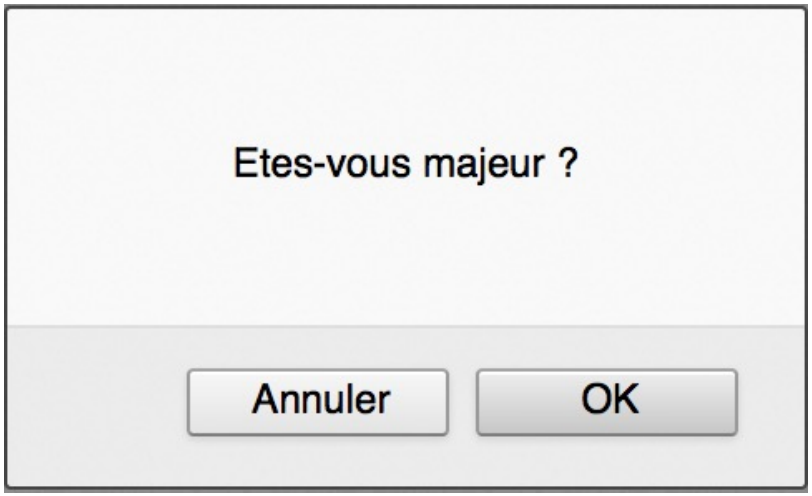
```
let prenom= prompt("Saisir votre prénom");
if (prenom)  {
    alert("Bonjour " + prenom);
} else {
    alert("Vous n'avez pas saisi de prénom");
}
```

Remarque : valeurs Falsy ou Truthy précisées dans les documentations

# Valeurs booléennes

- Demande de confirmation à l'utilisateur

La fonction confirm() contient en paramètre la question posée à l'utilisateur.
La fonction retourne true si l'utilisateur clique sur OK ou false sinon.




```
if (confirm("Etes-vous majeur ?")) {
    alert("Vous pouvez vous inscrire pour voter");
}
else {
    alert("Désolé, il faut attendre vos 18 ans pour voter");
}
```

# Opérateur ternaire ?:

- condition ? expression1 : expression2

Si la *condition* est vérifiée, *expression1* est retournée, dans le cas contraire, c'est *expression2*

```
let val1= 10, val2 = 20;
let reponse= (val1 > val2) ? val1 : val2;
alert("La plus grande valeur est : " + reponse);
```

Équivalent à :

```
let val1= 10, val2 = 20;
if (val1 > val2)
    alert("La plus grande valeur est : " + val1);
else
    alert("La plus grande valeur est : " + val2);
```

# Condition switch

```
switch (choix) {
    case val1:
                    instruction(s);
    break;
    case val2:
                    instruction(s);
    break;
    case …

    default :
                    instruction(s);
}
```

val1, val2… représentent les différentes valeurs que peut prendre la variable choix.

Une seule condition est effectuée grâce au break qui permet de sortir du switch.

default est l'option réalisée par défaut si aucun autre choix n'a été effectué auparavant.

# Boucle while

- while (condition) {
  instruction(s);
  }
- Boucle itérative : "tant que" la condition est vérifiée, les instructions sont réitérées.
  Les instructions peuvent ne pas être exécutées.

```
let age = false;
while (!age) {
    age = prompt("Veuillez saisir votre âge");
}
```

- La condition doit pouvoir être modifiée pour éviter une boucle infinie.

# Boucle do while

- do {
  instruction(s);
  } while (condition);
- Boucle itérative : "faire" les instructions "tant que" la condition est vérifiée.
  Les instructions sont effectuées au moins une fois.

```
let age;
do {
    age = prompt("Veuillez saisir votre âge");
} while (!age);
```

- La condition doit pouvoir être modifiée pour éviter une boucle infinie.

# Boucle for

- for (initialisation; condition; incrémentation) {
  instruction(s);
  }
- Boucle itérative effectuée un nombre fini de fois
  - *Initialisation* du compteur à la première boucle
  - *Condition* testée avant chaque boucle
  - *Incrémentation* du compteur après chaque fin de boucle

  La boucle se termine lorsque la condition n'est plus vérifiée

```
for (let cptBoucle=0; cptBoucle <5; cptBoucle ++) {
    alert("Boucle n°  "+ cptBoucle);
    // boucle répétée 5 fois pour cptBoucle de 0 à 4
}
```

# Interruptions break et continue

- break;

Vue lors de la présentation d'une structure switch, la commande break peut également être utilisée pour interrompre une boucle for, do..while ou while

- continue;

D'utilisation plus rare, continue met un terme à l'itération en cours mais laisse les itérations restantes de la boucle s'effectuer si nécessaire.

# Les fonctions

- function mafonction(argument1, argument2, …) {
  instruction(s);
  }
- Une fonction a pour objectif de regrouper des instructions qui pourront ainsi être exécutées facilement lors de chaque appel.
- Peut accepter un ou plusieurs arguments, placés lors de l'appel entre les parenthèses.
- Retourne éventuellement, en fin de son exécution, une valeur (une seule) avec l'instruction return.
- prompt(), alert() ou confirm() sont des fonctions natives du langage (*enfin, pas tout à fait mais on y reviendra…*).
- Le code d'une fonction non appelée n'est jamais exécuté.

# Variable locale *vs* globale

- Une variable locale est définie dans une fonction. Elle n'existe pas en dehors de la fonction où elle est définie.
- Une variable globale est définie hors d'une fonction. Son contenu est accessible à tout moment.

```
var taux = 0.2, prixHT = 100;              // variables globales
function calculPrixTTC ( ) {
    let prixTTC = prixHT * (1+taux);       // prixTTC var. locale
    alert("Prix TTC :" + prixTTC);         // prixTTC vaut 120
}
calculPrixTTC( );
alert("Prix TTC :" + prixTTC);             // prix TTC indéfini
```

# Variable locale *vs* globale

- Une variable locale est prioritaire sur une variable globale.

```
var prixHT = 100;                          // variable globale
function calculPrixTTC ( ) {
    let prixHT = 200;                      // variable locale
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);         // prixTTC vaut 240
}
calculPrixTTC( );
                        // La valeur 200 est utilisée dans le calcul
```

- En l'absence de variable locale, une variable globale est recherchée.

# Variable locale *vs* globale

- L'environnement local est prioritaire sur le global :

```
var prixHT = 100;                                   // prixHT vaut 100
function calculPrixTTC () {          // fonction externe
    let prixHT = 200;                               // prixHT vaut 200
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);          // prix TTC vaut 240
    function calculPrixLuxe () {  // fonction interne
        let prixTTC = prixHT * 1.5;
        alert("Prix TTC :" + prixTTC);
    }
    calculPrixLuxe();
}
calculPrixTTC();
```

- La visibilité des variables est appelée portée lexicale

# Argument d'une fonction

- Lors du passage d'un argument, une variable est créée en début de fonction pour contenir la valeur, la variable ou le résultat d'une autre fonction passé en paramètre.
- Toutes les variables créées et initialisées en début de fonction sont détruites à la fin de la fonction.
- Valeur passée en argument

```
function calculPrixTTC (prixHT) {
                // prixHT est créée et initialisée à 100
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);
                // prixHT est supprimée
}
calculPrixTTC(100);       // Appel de la fonction
```

# Argument d'une fonction

- Variable passée en argument

```
function calculPrixTTC (prixHT) {        // prixHT = montant
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);
}
let montant = 100;
calculPrixTTC(montant);
```

- Retour d'une fonction passé en argument

```
function calculPrixTTC (prixHT) {
// prixHT vaut la valeur retournée par la fonction prompt()
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);
}
calculPrixTTC(prompt("Prix HT du produit ?"));
```

# Argument par défaut

- Valeur par défaut d'un argument

Initialisation de la valeur d'un argument par simple affectation.
En cas d'absence de l'argument lors de l'appel de la fonction, la valeur définie par défaut est utilisée.

```
function calculPrixTTC (prixHT, TVA = 0.2  ) {
    let prixTTC = prixHT * (1 + TVA);
    alert("Prix TTC :" + prixTTC);
}
let montant = 100;
calculPrixTTC(montant, 0.5);          // Prix TTC = 150
calculPrixTTC(montant);               // Prix TTC = 120
calculPrixTTC(montant,      );        // Prix TTC = 120
calculPrixTTC(     , montant);        // Erreur !
```

# Paramètre rest

- Fonction avec un nombre d'arguments variable

  JavaScript stocke les arguments dans un tableau dont le nom est mentionné après l'opérateur de décomposition …

  Les valeurs peuvent être utilisées isolément dans la fonction

```
let val1 = 1, val2 = 2, val3 = 3;
function somme(...nombres){
    let maSomme = 0;
    for (let nombre of nombres){
    maSomme += nombre;
  }
  return maSomme;
}
console.log(somme(val1, val2));          // appel avec 2 arg.
console.log(somme(val1, val2, val3));    // appel avec 3 arg.
```

# Retour d'une fonction

- L'instruction return permet d'indiquer la valeur que l'on souhaite retourner.
- return met fin à l'exécution de la fonction

```
function calculPrixTTC (prixHT) {
    let prixTTC = prixHT * 1.2;
    return prixTTC;
            // La valeur de prixTTC est retournée
}
let prixProduitHT = prompt("Montant du produit HT ?");
let prixProduit = calculPrixTTC(prixProduitHT);
            // prixProduit reçoit la valeur retournée
alert("Montant du produit TTC : " + prixProduit);
```

# Retour d'une fonction

- Plusieurs valeurs de retour grâce à la décomposition (appelée aussi destructuring)

```
function calculPrixTTC (prixHT) {
    let TTCBase = prixHT * 1.2;
    let TTCLuxe = prixHT * 1.33;
    return { TTCBase, TTCLuxe};
            // Les 2 valeurs de prix TTC sont retournées
}
let { TTCBase, TTCLuxe} = calculPrixTTC(100);
            // Les 2 valeurs retournées sont récupérées
console.log(TTCBase);          // TTCBase vaut 120
console.log(TTCLuxe);          // TTCLuxe vaut 133
```

# Retour d'une fonction

- Plusieurs valeurs de retour grâce à la décomposition

```
function calculPrixTTC (prixHT) {
    let TTCBase = prixHT * 1.2;
    let TTCLuxe = prixHT * 1.33;
    return { TTCBase, TTCLuxe};
            // Les 2 valeurs de prix TTC sont retournées
}
let {TTCLuxe} = calculPrixTTC(100);
            // 1 seule valeur retournée est récupérée
console.log(TTCLuxe);          // TTCLuxe vaut 133
```