Les commentaires. Il est important de commenter vos programmes !
// Le reste de la **ligne** qui suit un // est un commentaire.
/* Est pris comme commentaire, tout ce qui suit jusqu'aux deux symboles : */

Définition d'une nouvelle variable :
var nom_de_variable = valeur par défaut**;** // commentaire indiquant sa signification.
var nouvel_objet = **new** objet**;** // pour une variable contenant un nouvel objet (pour plus tard)

Pour plus de clarté, une variable commence par des lettres en minuscules indiquant le type de variable, suivit d'un nom qui caractérise la variable et qui commence par une majuscule.
Exemples (en gras les minuscules indiquant le type de variable) :

- var **n**Entier = 17; // **n** indique que la variable contient un **entier**.
- var **v**Nombre_a_virgule = 3.14159265; // **v** pour un nombre à **virgule**.
- var **f**Boolean = true; // **f** pour une valeur **logique**, soit vraie, soit fausse.
- var **str**Chaine = "Ceci est un string"; // **str** pour une chaîne de caractères
- var **dt**Date = new Date(); // **dt** pour une date
- var **an**Tableau_d_entiers = new Array(2, 3, 5, 7, 11, 13, 17, 19, 23, 29); // **a** pour un **array**.
- var **astr**Jours = new Array("lundi", "mardi", "mercrdi", "jeudi", "vendredi", "samedi", "dimanche");
- var **o**Objet = new Nouvelle_objet; // **o** pour un **objet**;

Affectation ou assignation :
variable = expression**;**
Exemple :
```
vPerimetre = 3.1415 * 2 * vRayon;
```

Une **expression arithmétique** peut contenir des opérateurs et des fonctions.
Opérateurs arithmétiques :
+ - * / pour additionner, soustraire, multiplier, diviser.
% pour le reste de la division d'un nombre entier par un autre.
Exemple :
```
nReste = 26 % 7;
```
// nReste contiendra 5, car "26 divisé par 7" donne 3 avec 5 comme reste.

Opérateurs logiques pour des comparaisons.
var1 == var2 // test si var1 **est égale à** var2, retourne **true** si l'égalité est satisfaite, **false** sinon.
var1 **!**= var2 // test si var1 **est différent de** var2, retourne **true** si oui, **false** sinon.
var1 < var2 // test si var1 **est plus petit que** var2, retourne **true** si oui, **false** sinon.
var1 <= var2 // test si var1 **est plus petit ou égale à** var2, retourne **true** si oui, **false** sinon.
var1 > var2 // test si var1 **est plus grand que** var2 ...
var1 >= var2 // test si var1 **est plus grand ou égale à** var2 ...

Opérateurs logiques entre des grandeurs true (vraie) et false (faux).
fV1 **&&** fV2 // est **true** si fV1 **et** fV2 sont **true** tous les deux.
fV1 || fV2 // est **true** si fV1 **ou** fV2 **ou** les deux sont **true**.
// fV1 et fV2 peuvent être une comparaison vue ci-dessus, mise entre des parenthèses.
Exemple :
```
fT = (nNbr == 1)  || (nNbr % 2 == 0);
```
// fT est **true** si nNbr égale 1 ou est divisible par 2.

Opérateurs sur des chaînes de caractères (string)
```
var strS = "début" + " fin"; // strS contiendra "début fin".
```

Test : Si ... sinon.
if (*grandeur logique*) **{** *liste d'instructions séparées par des* ***;*** **}**
else { *autre liste d'instructions séparées par des* ***;*** **}** // le **else** est optionnel.
// La grandeur logique est généralement le résultat d'une opération logique, vue ci-dessus

Boucle for.
for (nVar=valeur_initiale**;** *opération_logique**;*** *modification de nVar***) {**
liste d'instructions séparées par des **; }**
Exemple :

```
nSum1 = 0;  nSum2 = 0;
for (nVar=1;  nVar <= 17;  nVar++) {
   nSum1 = nSum1 + nVar;
   nSum2 = nSum2 + nVar * nVar;
   }
```

// Fait varier nVar de 1 à 17 en ajoutant 1 à chaque passe et effectue deux instructions à chaque fois.

Boucle while.
while (*opération logique* **) {** *liste d'instructions séparées par des* ***;*** **}**
// Tant que l'opération logique donne un résultat **true**, effectue la liste d'instructions.
Exemple :

```
var nVar = 7;
while (nVar > 1) {
   if (nVar % 2 == 0)  { nVar = nVar / 2; }
   else  { nVar = 3*nVar + 1; }
   }
```

// Tant que nVar est supérieur à 1, effectue le test, qui modifiera la variable nVar.
// Attention, si l'opération logique donne toujours true, le programme restera dans la boucle
// indéfiniment et on dit dans ce cas que "le programme est bugué !"

Boucle do while
do { *liste d'instructions séparées par des* ***;*** **}**
while (*opération logique*);
// Répète la liste d'instruction, tant que l'opération logique donne un résultat **true**.
Exemple :

```
var nVar = 7;
do {
   if (nVar % 2 == 0)  { nVar = nVar / 2; }
   else  { nVar = 3*nVar + 1; }
   }
while (nVar > 1);
```

// Effectue les instructions, tant que nVar est supérieur à 1
// La boucle "do while" est très similaire à la boule "while".
// Elle effectue simplement le test à la fin des instructions et non au début.

Les fonctions.
function nom_de_fonction (paramètre_1**,** paramètre_2**,** ... **,** paramètre_N) {
//==
liste d'instructions séparées par des **;**
return un_résultat; // optionnel
} // nom_de_fonction

Les fonctions permettent de regrouper une liste d'instructions que **l'on désire exécuter plusieurs fois**, avec des paramètres différents.
Fréquemment, elles retournent un résultat avec l'instruction : **return**.

Les variables déclarées dans la liste d'instructions de la fonction ne sont pas utilisables à l'extérieur de la fonction et n'influence aucune variable à l'extérieure de celle-ci. On dit que ces **variables sont locales**.

Exemples :

```
function Puissance(vNb, nPuissance) {
//==================================
// Retourne vNb à la puissance nPuissance
// nPuissance doit être un entier

var nn = 0;     // pour boucler
var vPuis = 1;  // vNb mis à la puissance nPuissance

for (nn=1; nn<=nPuissance; nn++) {
  vPuis = vPuis * vNb;
  }
return vPuis;
}  // fin de Puissance

// utilisation de la fonction.
var vNb = 2;    var nn = 0;
for (nn = 0;  nn <=10;  nn++) { Display(vNb + " ^" + nn + " = " + Puissance(vNb, nn)); }
//-------------------------------------------------------


function TestPremier(nTest) {
//==========================
// Test si le nombre  nTest  donné, est premier ou non.
// Retourne  1  s'il est premier,  0 sinon.

var nn = 2;         // facteur possible de nTest
var fPremier = 1;   // indique si nTest est premier,
                    // par défaut, au départ on suppose qu'il est premier.

// Boucle pour tester si  nTest  est divisible par  nn.
// S'arrête si on a testé tous les nombres jusqu'à la racine carrée de  nTest.
// S'arrête si le nombre n'est pas premier.
while ((nn*nn <= nTest) && (fPremier == 1))  {
  if ((nTest % nn) == 0) fPremier = 0;
  nn++;  // même signification que : nn = nn + 1;
  }
return fPremier;  // retroune l'indication si nTest est premier ou non.
}  // fin de TestPremier

// utilisation de la fonction.
for (nn = 2;  nn <25;  nn++) {
  if (TestPremier(nn) == 1) Display(nn + "  est premier");
  else Display (nn + "  n'est pas premier");
   }
```

Quelques fonctions standards, utiles :
parseInt(une chaîne de caractères) donne le nombre entier représenté par la chaîne de caractères.
parseFloat(une chaîne de caractères) donne le nombre à virgule représenté par la chaîne de caractères.
Math.sqrt(nombre) donne la racine carrée du nombre.
Math.sin(angle en radians) **Math.cos**(angle en radians) **Math.tan**(angle en radians)
Math.asin(nombre) donne l'arcsinus du nombre, qui doit être entre -1 et 1.
Math.acos(nombre) et **Math.atan**(nombre) sont similaires.
Math.log(nombre) donne le logarithme naturel du nombre.
Math.ceil(nombre) donne la partie entière du nombre.
Tapez **Math.** et vous verrez beaucoup de choix dans Netbeans.

Quelques instructions javascript très utiles pour faire le lien avec le code HTML.

```
var idObjet = document.getElementById("id_du_style_de_l_objet");
```
// Récupère une variable qui contient de l'information sur l'objet, telle que :

```
idObjet.value
```
// sa valeur, typique pour un champ texte
```
idObjet.style.left
```
// la gauche de sa position
```
idObjet.style.top
```
// le haut de sa position
```
idObjet.style.width
```
// sa largeur
```
idObjet.style.height
```
// sa hauteur
```
idObjet.style.zIndex
```
// lit ou modifie la valeur du champ z-index
```
idObjet.style.visibility
```
// = "visible" ou = "hidden"

```
idObjet.style.xxx
```
// xxx est un champ du style. Il y en a beaucoup.

```
document.getElementById("divObjet").innerHTML = "nouveau code HTML";
```
// Permet de changer le code HTML d'une section ...

Quelques tests d'événements :
onclick = "action à effectuer si on click sur l'objet"
ondblclick = "action à effectuer si on double click sur l'objet"
onmouseover = "action à effectuer si la souris arrive sur l'objet"
onmouseout = "action à effectuer si la souris part de l'objet"
onmousemove = "action à effectuer si la souris bouge sur l'objet"
Liste d'autres événements :
onabord ; onblur ; onchange ; onclick ; ondblclick ; ondragdrop ; onerror ; onfocus ; onkeydown ; onkeypress ; onkeyup ; onload; onmousedown ; onmousemove ; onmouseout ; onmouseover ; onmouseup ; onreset ; onresize ; onselect ; onsubmit ; onunload.

Le **timer** (minuteur) est utile pour répéter une action régulièrement.
c.f. http://www.juggling.ch/zgisin/javascript_20140130_timer_simple.html pour un exemple.
```
nTimerID = setInterval("Compteur()", 200);
```
// pour démarrer le Timer
// A intervalle de toutes les 200 millisecondes, la fonction "Compteur()" sera appelée.

```
clearInterval(nTimerID);
```
// pour arrêter le Timer
// arrête l'appelle régulier à la fonction "Compteur()".