**Fiche Javascript n° 1 - *Variables - Opérateurs arithmétiques - Opérateurs logique - Constantes***

**1 - Variables: généralités**

A retenir

* **Une variable est un récipient (container), nommé, qui contient une valeur**. D'un point de vue un peu scientifique, on peut dire qu'une variable est une référence dont la valeur est une adresse mémoire qui elle-même est associée à une autre valeur.
* En javascript, **on déclare une variable avec le mot réservé var, ou let**.Vous ne **déclarez votre variable qu'une seule fois** (avec le mot réservé **var**) Par la suite vous accédez à votre variable directement par son nom, pour l'utiliser ou la modifier.
* Avant d'utiliser votre variable, vous devez l'**initialiser**, c'est à dire lui **affecter** une première valeur. Vous initialisez votre variable grâce à l'**opérateur d'affectation** '**=**'.

// Déclaration d'une variable sans initialisation (sans affectation de valeur initiale)

var pseudo;

// let pseudo;

// Utilisation d'une variable non-déclarée ou non-initialisée = Bad !

alert(pseudo);

console.log(pseudo); // undefined

// Si vous n'avez pas de valeur concrète pour l'initialisation, initialisez avec 'null'

var pseudo2 = null;

console.log(pseudo2); // null

//----------------------------------------------------------------------------

// Déclaration, affectation et utilisation

var score = 100;

console.log(score); // 100

// Modification et utilisation

score = 200;

console.log(score); // 200

**2 - Types de données (primitives, ou natives, ou scalaires)**

**!!! TODO ! Démo Abstraction. On apprend à différencier les attributs des entités**.

|  |  |
| --- | --- |
| types de données javascript | |
| **number** | 10 - 10.50 |
| **string** | 'Bob' - 'A' |
| **boolean** | true - false |
|  |  |
| **object** | Type de donnée Custom (perso, non natif) |

// Chaines de caractères (NB: quotes !)

var nom = "Bob";

// Nombres

var nombreEntier = 37;

var nombreDecimal = 37.1;

// Booleans

var ouvert = true; // ou false

// Affichage des valeurs et du type de donnée

console.log("nom ==> " + nom);

console.log(typeof nom);

console.log("nombreEntier ==> " + nombreEntier);

console.log(typeof nombreEntier);

console.log("nombreDecimal ==> " + nombreDecimal);

console.log(typeof nombreDecimal);

console.log("ouvert ==> " + ouvert);

console.log(typeof ouvert)

// ----------------------------------------------------

// Deux fonctions systèmes importantes:

var chaine1 = "37"; // string !

var chaine2 = "100.37"; // string !

console.log(chaine1 + chaine2); // concatène ! normal, ce sont des chaines de caractères

console.log(parseInt(chaine1) + parseFloat(chaine2)); // Deux fonctions système importantes

// On remarque que JS considère un type de données 'number' mais distingue néanmoins un entier d'un décimal

**!** On constate que JS considère un type de données 'number' mais on constate aussi qu'il distingue néanmoins un entier d'un décimal **!**

**!** Ayez bien conscience que ce qui vous arrive en réception d'une requète http est une série de chaine de caractères. Vous réceptionnez un nombre sous forme de string **!**

**3 - Opérateurs arithmétiques - Expressions**

**Un programme informatique est parfaitement ami avec les nombres, et est capable de les traiter avec les opérateurs arithmétiques classiques**.

// ----------------------------------------------------

// opérateur + (addition)

var total = 0;

console.log("total ==> " + total); // 0

var prix1 = 10;

var prix2 = 20;

total = prix1 + prix2; // NB ! ON AFFECTE A 'total' LE RESULTAT D'UNE EXPRESSION

console.log("total ==> " + total); // 30

total = total + 1;

// raccourcis !

// total += 1; // raccourci

// total++; // raccourci (uniquement avec 1)

// ++total; // raccourci (uniquement avec 1)

console.log("total ==> " + total); // 31

// ----------------------------------------------------

// opérateur - (soustraction)

var anneeNaissance = 1964;

var today = 2019;

var age = today - anneeNaissance; // NB ! ON AFFECTE A 'age' LE RESULTAT D'UNE EXPRESSION

console.log("age ==> " + (age - 10)); // on est coquet !

// ----------------------------------------------------

// opérateurs \* et / (multiplication et division)

var price = 100;

var rate = 20;

var tax = price \* rate / 100; // NB ! ON AFFECTE A 'tax' LE RESULTAT D'UNE EXPRESSION

console.log("ht ==> " + price); // 100

console.log("tva ==> " + tax); // 20

// NB ! On voit à quel point les parenthèses forcent le robot vers un calcul ou une concaténation

console.log("ttc ==> " + ( price + tax )); // 120

// ----------------------------------------------------

// opérateur modulo (partie restante d'une division)

var reste = 10 % 3;

console.log("reste ==> " + reste); // 1

// ----------------------------------------------------

// priorité (précédence) des opérateurs (comme en arithmétique)

var nb1 = 5 + 4 \* 2;

var nb2 = (5 + 4) \* 2;

console.log("nb1 ==> " + nb1 ); // 13

console.log("nb2 ==> " + nb2 ); // 18

**4 - Opérateurs de comparaison**

**Un programme informatique est parfaitement ami avec les nombres, et est capable de les comparer avec les opérateurs de comparaison. Ils sont identiques à ceux utilisés en arithmétique, excepté pour la stricte égalité, qui utilise ==, l'opérateur = étant l'opérateur d'affectation !** .

// Déclarations de variables

let value1 = 10;

let value2 = 100;

let value3 = 100;

let str1 = "Salut";

let str2 = "Salut";

// supérieur ?

console.log("value1 > value2 ==> " + (value1 > value2)); // false

// inférieur ?

console.log("value1 < value2 ==> " + (value1 < value2)); // true

// supérieur ou égal ?

console.log("value1 >= value2 ==> " + (value1 >= value2)); // false

// inférieur ou égal ?

console.log("value1 <= value2 ==> " + (value1 <= value2)); // true

// égal ?

console.log("value1 == value2 ==> " + (value1 == value2)); // false

console.log("str1 == str2 ==> " + (str1 == str2)); // true

// différent ?

console.log("value1 != value2 ==> " + (value1 != value2)); // true

// egal ET de même type ?

console.log("value2 === value3 ==> " + (value2 === value3)); // true

**5 - Opérateurs logiques**

**On utilise les opérateurs logiques (ou opérateurs booléens) pour effectuer des comparaisons sur une base de TRUE/FALSE (0/1)**.

**!** Vérifiez bien, selon la technologie utilisée, comment cette dernière considère le vrai et le faux **!**

var majeur = true;

var tenueCorrecte = true;

// 'AND' - '&&' les deux doivent être vrais

console.log("majeur ET tenueCorrecte ==> " + (majeur && tenueCorrecte)); // true

// 'OR' - '||' (l'un ou l'autre doit être vrai. les deux peuvent être vrais)

console.log("majeur OU tenueCorrecte ==> " + (majeur || tenueCorrecte)); // true

// 'XOR' - '^' (l'un doit être vrai. l'autre doit être faux)

console.log("majeur OU (exclusif) tenueCorrecte ==> " + (majeur ^ tenueCorrecte)); // false 0

// 'NOT' - '!'

console.log("pas majeur ==> " + (!majeur)); // false

// exemple avec not et xor

console.log("majeur OU (exclusif) tenueCorrecte ==> " + (!majeur ^ tenueCorrecte)); // true 1

**6 - Constantes**

**Une constante est un récipient pour une valeur, sauf que cette valeur ne peut pas changer suite à sa déclaration**. La valeur restera ... constante ..., d'où le nom.

**!** Les développeurs, parfois, n'ont pas à ce point le réflexe de prendre en compte une éventuelle constance de la valeur déclarée, et déclarent des variables. On perd dans ce cas beaucoup en termes de solidité et de rigueur **!**

const FORMULA = 111;

console.log(FORMULA);

// FORMULA = 100; // Interdit !

**20 - Exercice 1 - Enoncé**

exercice 1 Enoncé

Nous avons:

* un repas qui vaut 20€
* un pourboire d'un montant de 2€

SVP

* affichez le montant total (repas et pourboire)
* le repas vaut maintenant 30€ et le pourboire 1€
* affichez le montant total (repas et pourboire)
* affichez le montant total avec une taxe d'un taux de 20%
* affichez le montant total avec une remise de 5€
* *Merci et bon travail*

**21 - Exercice 1 - Solution**

exercice 1 Solution

// On déclare 2 variables

var repas = 20;

var pourboire = 2;

// Idéalement, on déclare une variable 'total' car on va la réutiliser

var total = null;

// Repas + pourboire

total = repas + pourboire;

console.log("Total = " + total); // 22

// On change les valeurs

repas = 30;

pourboire = 1;

// Repas + pourboire

total = repas + pourboire;

console.log("Total = " + total); // 31

// Taxe de 20%

var taux = 20;

total = total + (total \* taux / 100);

console.log("Total = " + total); // 37.2

// Remise

total = total - 5;

console.log("Total = " + total); // 32.2

**22 - Exercice 2 - Enoncé**

Exercice 2 Enoncé

Nous avons:

* 3 étudiants: 'Bob', 'Bernard', 'Monika'
* 3 notes pour Bob: 10, 5, 20
* 3 notes pour Bernard: 1, 1, 1
* 3 notes pour Monika: 18, 12, 13

SVP

* affichez la moyenne pour chaque étudiant
* affichez la moyenne générale
* *Merci et bon travail*

**23 - Exercice 2 - Solution**

Exercice 2 Solution

// 3 notes pour chacun des étudiants

var bobNote1 = 10;

var bobNote2 = 5;

var bobNote3 = 20;

var bernardNote1 = 1;

var bernardNote2 = 1;

var bernardNote3 = 1;

var monikaNote1 = 18;

var monikaNote2 = 12;

var monikaNote3 = 13;

// Calcul des moyennes de chaque étudiant

var moyenneBob = (10 + 5 + 20) / 3;

var moyenneBernard = (1 + 1 + 1) / 3;

var moyenneMonika = (18 + 12 + 13) / 3;

// Affichage

console.log("Moyenne de Bob " + moyenneBob);

console.log("Moyenne de Bernard " + moyenneBernard);

console.log("Moyenne de Monika " + moyenneMonika);

// Calcul de la moyenne générale

var moyenneGenerale = (moyenneBob + moyenneBernard + moyenneMonika) / 3;

// Affichage

console.log("Moyenne générale " + moyenneGenerale);

**Fiche Javascript n° 2 - *Tests - Conditions - (if - if else if - else)***

**1 - if**

On peut effectuer des tests sur des valeurs avec le mot réservé **if**

**Clairement, le robot écalue l'expression fournie entre parenthèses. Si le résultat est true, alors le robot exécute les instructions. Si le résultat est false, ici, il ne fait rien**.

En cas de succès du test (true), le robot exécute les instructions **délimitées par une accolade ouvrante { et une accolade fermante }**.

// ----------------------------------------------------

// Un simple test. On ne fait rien si le test est faux.

var age = 18;

if(age >= 18){

console.log("Majeur !");

}

**2 - if else**

En cas de d'échec du test (false), ont peut demander au robot de réagir grâce au mot réservé **else**

**Attention ! else ne fonctionne pas tout seul ! Il est forcément couplé à un if**.

Des accolades ouvrante et fermante **{}** délimitent les instructions que le robot exécute si le test échoue.

// Un simple test. On réagit également si le test est faux.

if(age >= 18){

console.log("Majeur !");

}

else{

console.log("PAS Majeur !");

}

**!** Pas de parenthèses après **else** ! Pas d'expression à évaluer. Le robot ira exécuter les instructions situées dans le bloc **else** uniquement si l'expression évaluée avec **if** a retourné false **!**

**3 - if - else if - (else)**

On peut effectuer plusieurs tests (idéalement sur des mêmes valeurs !) grâce au mot réservé **else if**

**Attention ! Un bloc else if ne fonctionne pas tout seul ! Il est forcément couplé à un bloc if**.

Des accolades ouvrante et fermante **{}** délimitent les instructions que le robot exécute si le test est en succès.

**Dès qu'une expression est vraie (true), le robot exécute les instructions, puis SORT du bloc ! Il ne teste pas les autres expressions**.

// ----------------------------------------------------

// Déclaration

var couleur = "#ff0000";

// Test de plusieurs valeurs sur la variable 'couleur'

if(couleur == "#ff0000"){

console.log("Rouge");

}

else if(couleur == "#00ff00"){

console.log("Vert");

}

else if(couleur == "#0000ff"){

console.log("Bleu");

}

else{

console.log("Couleur non acceptée !");

}

// ----------------------------------------------------

// Un autre test

// Appréciez la logique

// Remarquez les {} superflues si vous n'avez qu'une seule instruction. à votre appréciation.

if(age < 10) console.log("Kid !");

else if(age < 20) console.log("Ado !");

else if(age < 40) console.log("Adulte !");

else console.log("Sénior !");

**!** Vous pouvez omettre, si vous le souhaitez, les accolades ouvrante/fermante {} du bloc d'instruction, s'il n'y a qu'une seule instruction. A votre libre appréciation, mais sachez que c'est légal **!**

**4 - switch**

On peut également demander au robot de réagir pour une série de valeurs, vis à vis d'une variable avec le mot réservé **switch**

Un bloc **switch** ressemble à s'y méprendre à un bloc **if - else if**, sauf dans le fonctionnement. Le premier teste des expressions qui retournent true/false, pendant que le second teste des valeurs string ou nombres éventuellement égales à l'argument du switch (et dans ce cas exécute les instructions situées après le mot réservé **case:**)

Attention, dans un bloc un switch, lorsqu'il rencontre un cas vrai, et qu'aucun **break** ne l'interrompt, le robot considère tous les cas suivants comme vrai, et exécute leurs instructions ! Pour éviter cela, on utilise le mot réservé **break**. Lorsqu'un **case:** est vrai on peut donc si on le souhaite délimiter les instructions exécutées par le robot avec un **break**

Notez le mot réservé **default** (une sorte de else) que le robot exécutera automatiquement quand il le rencontre.

Une paire d'accolades **{}** délimite la responsabilité du **switch**.

// ----------------------------------------------------

// Déclaration

var color = "#0000ff";

// Tests sur des valeurs bien précises pour 'color'.

// Des tests de stricte égalité.

switch(color){

case "#ff0000" :

console.log("Rouge");

break;

case "#00ff00" :

console.log("Vert");

break;

case "#0000ff" :

console.log("Bleu");

break;

default:

console.log("Couleur non acceptée !");

}

**5 - Opérateur ternaire (ternary operator)**

L'**opérateur ternaire** est une sorte de **if - else** court. Si le résultat de l'expression retourne true, alors le robot pointe sur la valeur à gauche des deux points **:**. Si le résultat retourne false, le robot pointe sur la valeur à droite des deux points **:**

**!** Prenez soin de faire quelquechose de cette valeur, de l'afficher, ou de l'affecter à une variable **!**

// ----------------------------------------------------

// Affectation d'une valeur à une variable d'après un test

var message = (age >= 18)?"Majeur":"Mineur";

console.log("message ==> " + message);

// Affichage d'une variable d'après un test

console.log((age > 180)?"Vieux":"Jeune, en tout cas moins vieux");

**20 - Exercice 1 - Enoncé**

Exercice 1 Enoncé

Nous avons:

* Un numéro gagnant (valeur à votre appréciation)
* Un numéro joué (valeur à votre appréciation)

SVP écrivez un programme capable de dire

* Si le numéro est inférieur, dîtes-le. Vous dîtes également combien il manque pour le bon numéro
* Si le numéro joué est égal au numéro gagnant, dîtes-le. Vous dîtes également 'bravo !'
* Si le numéro est supérieur, dîtes-le. Vous dîtes également de combien il dépasse le bon numéro
* Déclarez une variable 'win' et affectez la valeur 'Gagné' ou 'Perdu', selon. Contrainte: utilisez une seule instruction pour cela !
* Affichez cette variable 'win'
* *Merci et bon travail*

**22 - Exercice 2 - Enoncé**

Exercice 2 Enoncé

Nous avons:

* Un indice de satisfaction avec les valeurs:1(bien) - 2(passable) - 3(mauvais)

SVP Ecrivez un programme capable de dire

* Selon la satisfaction: 'bien', 'passable', 'mauvais'
* Contrainte: écrivez votre programme de deux manières différentes, notamment dans la manière de tester l'indice de statisfaction
* *Merci et bon travail*

**Fiche Javascript n° 3 - *Boucles - Répétitions***

**1 - for**

**Le mot réservé for permet de répéter des instructions un certain nombre de fois**.

Le certain nombre de fois est dépendant d'un compteur, et d'un test sur ce même compteur (qui va varier à chaque itération (à chaque passage dans la boucle)).

Fonctionnement **for(init compteur; test; modif compteur)**

* le robot initialise le compteur (une seule fois au début)
* le robot effectue un test avec ce compteur, et ce, à chaque itération. Si le test est vrai, il exécute les instructions du bloc (il effectue une itération). Dans le cas contraire, le robot sort du bloc
* le robot modifie la valeur du compteur (à la fin de chaque itération)

Notez le mot réservé **continue** qui permet de passer à l'itération suivante.

Notez le mot réservé **break** qui permet de sortir de la boucle (du bloc for).

**!** Attention avec la logique impliquée dans une boucle. Il y a un fort risque de boucler à l'infini, lorsqu'un test est toujours vrai notamment **!**

// ----------------------------------------------------

// On affiche 10 fois "Salut" et on numérote le salut

for(let i = 1; i <= 10; i++){

console.log("Salut n° " + i);

}

// ----------------------------------------------------

// Test avec 'continue'

for(let i = 1; i <= 100; i++){

if(i % 2 != 0) continue;

console.log(i + " est pair !");

console.log(i + " est pair, forcément, sinon le robot ne serait pas là !");

}

// ----------------------------------------------------

// Test avec 'break'

// On affiche 10 fois "Salut" et on numérote le salut, mais on sort de force au 3eme salut

for(let i = 1; i <= 10; i++){

console.log("Salut n° " + i);

if(i == 3) break;

}

// ----------------------------------------------------

// Boucles imbriquées. On affiche 5 fois les nombres de 1 à 10

// Cet exemple fait bien comprendre la forte localité des variables déclarées avec let

for(let i = 1; i <= 5; i++){

console.log("Boucle externe \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* " + i);

for(let i = 1; i <= 10; i++){

console.log("Boucle interne " + i);

}

}

**2 - while**

**Le mot réservé while permet de répéter des instructions un nombre inconnu de fois. Intéressant pour lire un fichier (tant qu'il y a des lignes), pour rejouer tant qu'on gagne ...**.

Fonctionnement **while(expression à tester)**

* si l'expression est vrai, le robot parcourt le bloc et exécute les instructions
* le test est toujours le même, et le robot boucle tant que ce test est vrai

**!** Attention avec la logique impliquée dans une boucle. Il y a un fort risque de boucler à l'infini, lorsqu'un test est toujours vrai notamment **!**

**!** Il y a aussi possibilité pour le robot de ne jamais entrer dans la boucle ! Notamment si le test est faux dès la première fois **!**

// ----------------------------------------------------

// On affiche la valeur de compteur tant qu'il est inférieur à 10

var compteur = 0;

while(compteur < 10){

console.log(compteur);

compteur++; // Watch !!!

}

// ----------------------------------------------------

// On joue tant qu'on gagne

while((Math.random()\*6) >= 3) {

console.log("You win");

}

**3 - do while**

**Le mot réservé do while permet de répéter des instructions un nombre inconnu de fois. La particularité est que le robot exécutera les instructions du bloc de toute façon une fois, même si le test est faux le début**.

**!** Attention avec la logique impliquée dans une boucle. Il y a un fort risque de boucler à l'infini, lorsqu'un test est toujours vrai notamment **!**

// ----------------------------------------------------

// Déclaration

var compteur2 = 10;

// Ne rentre jamais dans la boucle !

while(compteur2 < 10){

console.log("compteur2 dans while ==> " + compteur2);

compteur++;

}

// Rentre au moins une fois dans la boucle

do{

console.log("compteur2 dans do...while ==> " + compteur2);

}while(compteur2 < 10);

// ----------------------------------------------------

// On gagne au moins une fois, et on joue tant qu'on gagne

do {

console.log("You win");

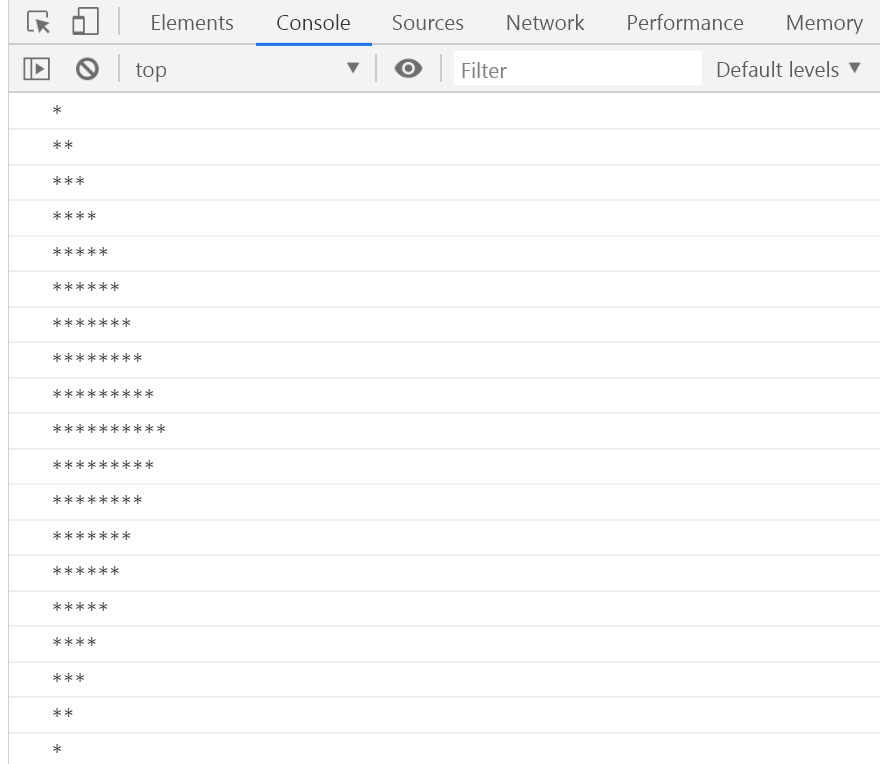
}while((Math.random()\*6) >= 3);

**20 - Exercice 1 - Enoncé**

Exercice 1 - Enoncé

SVP

* Ecrivez un programme qui, pour un nombre donné de lignes (ici 30), affiche des étoiles et ce avec la contrainte suivante
* La première moitié démarre à 1 étoile, et écris une étoile de plus à chaque ligne
* La seconde moitié écris une étoile de moins à chaque ligne
* *Merci et bon travail*



Exemple d'affichage de lignes d'étoiles (ici avec 30 lignes)

**21 - Exercice 1 - Solution**

Exercice 1 Solution

// déclaration nombre de lignes

var nbLines = 20;

// Motif

for(let i = 1; i <= nbLines; i++){

var stars = "";

// Première moitié

if(i <= nbLines/2){

for(let x = 0; x < i; x++){

stars += "\*";

}

}

else {

// Seconde moitié

for(let x = 0; x < nbLines - i; x++){

stars += "\*";

}

}

console.log(stars);

}

**Fiche Javascript n° 4 - *Fonctions***

**1 - Généralités - A retenir**

A retenir

* **Une fonction est bloc de code, (un amalgame, un regroupement de logique, d'instructions). Pour exécuter ce bloc de code, on appelle (call), ou encore on invoque (invoke) la fonction**.
* **Une fonction se déclare, s'implémente, puis s'utilise par la suite en invoquant son nom**.
* Une fonction peut avoir des paramètres, ou pas
* Une fonction peut retourner une valeur, ou pas
* On peut invoquer une fonction depuis une autre fonction
* En javascript, on déclare les fonctions de deux manières:
  + avec le mot réservé **function**, un nom, des parenthèses pour d'éventuels paramètres, et le corps de la fonction, ainsi que son implantation
  + on peut aussi affecter une variable avec une **inline function**, pas de nom (fonction anonyme), des parenthèses pour d'éventuels paramètres, et le corps de la fonction, ainsi que son implantation

**!** L'intérêt des fonctions est qu'**on utilise du code, écrit une seule fois, encore et encore**. Mieux, on peut dynamiser le fonctionnement d'une fonction, notamment grâce à ses paramêtres. Dans le cas de la gestion d'un évènement, la fonction (qui a dans ce cas un rôle de 'gestionnaire d'évènements') est absolument incontournable **!**

**!** Javascript est une **First Class Function Technology** (une technologie qui met la fonction au premier plan). En effet, on peut affecter une variable avec une fonction (inline function). On peut utiliser une fonction comme paramètre d'une autre fonction, on peut utiliser une fonction comme valeur de retour d'une autre fonction. Javascript est très 'copain' et ce d'une manière native, avec les fonctions. **!**

**!** Tentez de donner une et une seule responsabilité à vos fonctions. Principe SOLID **!**

**!** Si une fonction est membre d'un objet, on lui préfère dans ce cas l'appellation METHODE **!**

**2 - Fonction sans paramètre, qui ne retourne rien**

// Déclaration - Pas de paramètres - Pas de valeur de retour

function saluer(){

console.log("Salut !");

}

// Utilisation

saluer();

saluer();

**3 - Fonction avec paramètre(s), qui ne retourne rien**

// Déclaration - Avec paramètres - Pas de valeur de retour

// Avec 1 paramètre

function saluerPersonne(prenom){

console.log("Salut " + prenom + " !");

}

// Avec 2 paramètres

function saluerPersonne2(firstName, name ){

console.log("Salut " + firstName + " " + name + " !");

}

// utilisation

let nom = "Smith";

let prenom = "Bob";

saluerPersonne("Monika"); // arguments en dûr

saluerPersonne(prenom); // arguments dynamiques

saluerPersonne2("Monika", "Monroe"); // arguments en dûr

saluerPersonne2(prenom, nom ); // arguments dynamiques

**!** Les nom de l'argument est souvent le même que celui du paramètre, mais notez bien qu'ils pourraient avoir un nom différent. Le paramètre est local à la fonction, du coup, peu importe une éventuelle correspondance, ou pas **!**

**4 - Fonction avec paramètre(s), avec une valeur de retour**

// Déclaration - Avec paramètres - Avec valeur de retour

function applyTax(montant, taux){

return montant \* taux / 100;

}

// Métier et utilisation

var prixProduit = 1;

var rate = 20;

var tva = null;

// Pour des produits dont le produit va de 1 à 100

for(; prixProduit <= 100; prixProduit++ ){

// On calcule la tva

tva = applyTax(prixProduit, rate);

// On affiche les données

console.log("Produit HT ==> " + prixProduit );

console.log("TVA ==> " + tva + " (" + rate + "%)");

console.log("Produit TTC ==> " + (prixProduit+ tva));

console.log("\*\*\*");

}

**8 - Conception d'une fonction dont le rôle est de retourner un nombre au hasard entre une valeur min et max**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// myRandom(min, max)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

function myRandom(min, max){

// On utilise la méthode random() de l'objet Math

let nombre = Math.random();

// On multiplie par le nombre de valeurs distinctes souhaitées

nombre \*= (max - min + 1);

// On arrondit (par en-dessous)

nombre = Math.floor(nombre);

// On se décale (éventuellement) du 0

nombre += min;

return nombre;

// One shot

//return Math.floor( (Math.random() \* (max - min + 1)) + min );

}

**20 - Exercice 1 - Enoncé**

Exercice 1 - Enoncé

SVP

* Ecrivez un programme qui, pour un nombre quelconque de lignes, affiche un motif quelconque et ce avec la contrainte suivante
* La première ligne écrit une fois le motif, la deuxième deux fois le motif ... jusqu'à la moitié du dessin
* La seconde moitié écris un motif de moins à chaque ligne
* *Merci et bon travail*

**22 - Exercice 2 - Enoncé**

Exercice 2 - Enoncé

SVP

* Ecrivez un fonction dont la responsabilité est de retourner si une personne est majeure (ou pas)
* Testez votre fonction
* Ecrivez un fonction dont la responsabilité est de retourner le périmètre d'un cercle à partir de son rayon
* Testez votre fonction
* Ecrivez un fonction dont la responsabilité est de retourner si un nombre est divisible par un autre
* Testez votre fonction
* *Merci et bon travail*

**Fiche Javascript n° 5 - *Objets***

**1 - Un objet c'est quoi ?**

**!** Javascript est un peu *à part ...* vis à vis de la COO (et du coup de la poo), car il ne connait pas la **class** (qui est considérée comme l'élément de base de l'orienté objet) **!**

**!** Par contre, (ouf !) il connait l'**Object** (qui est clairement le résultat de l'instanciation d'une classe (qui sert dans ce cas de modèle)) **!**

A retenir

* **Un objet est un conteneur, un récipient, une enveloppe, une sorte de super-variable, qui contient**.
  + des attributs, ou encore propriétés, qui sont ce que l'objet connait, ce dont il est intrinsèquement constitué
  + des méthodes, qui sont sont clairement des fonctions à l'intérieur d'un objet, et qui sont ce que l'objet sait faire, comment il va se comporter
* il y a 3 manières de déclarer un objet.
  + déclaration inline: l'objet est unique, en ce sens où il est lui-même son propre modèle, mais peut être cloné
  + déclaration avec une fonction constructrice prototypale: c'est ce qui ressemble le plus aux classes des langages comme Java, C++, C#, Php 7. Particulièrement intéressant car c'est un modèle pour un nombre indéfini d'objets. C'est clairement une fonction normale, à part que son implantation se sert systématiquement du pointeur conceptuel **this**. Dans ce cas, on peut parler pour les objets qui se servent de la fonction constructrice, d'instance de classe, même si c'est un peu abusif, Javascript ne connaissant pas les classes, au sens pur du terme
  + déclaration avec le mot réservé **class**: c'est un leurre ! Javascript ne connait pas les classes. C'est un emballage agréable de la fonction constructrice prototypale, qui a pour intérêt de posséder un constructeur, et de faciliter l'écriture des méthodes statiques, pour ne citer que cela. D'autre part, cela plait aux développeurs qui viennent des langages plus 'rigoureux', même s'ils ne sont pas dupes

**!** On se permettra de parler de classe lorsque l'on parle de la fonction constructrice prototypale, et d'objet, lorsque l'on parle de ses instances **!**

**2 - Objets inline -   
A - Création**

// On déclare un objet

var clientModel = {

// Attributs (ou propriétés)

nom : "Bob",

scoreFid : 100,

//Méthodes

afficher : function(){

console.log("Nom ==> " + this.nom + " Score ==> " + this.scoreFid);

},

addPoints : function(points){

this.scoreFid += points;

}

}

// On affiche les données (l'état de l'objet (object state))

clientModel.afficher();

// On donne 10 points à Bob

clientModel.addPoints(10);

// On affiche les données (l'état de l'objet (object state))

clientModel.afficher();

**3 - Objets inline -   
B - Clone de l'objet initial**

// suite (B)

// On instancie un objet en prenant comme modèle un autre objet

var client2 = Object.create(clientModel);

// 'client2' est un clone de clientModel en termes de propriétés et de valeurs !

// Mais ce n'est pas le même objet

console.log("Monika");

client2.afficher(); // !!! Un clone !!!

// affectation du nom et don de points fid à Monika

// va permettre de bien distinguer les 2 objets

client2.nom = "Monika";

client2.addPoints(10);

// Il y a bien deux objets

client2.afficher();

clientModel.afficher();

**4 - Objets inline -   
C - Ajout de propriétés à la volée**

// suite (C)

// Monika a hérité, a pris pour modèle Bob. Elle est 'enfant' de Bob, qui lui est le 'parent'

// Ajout d'une propriété 'on the fly' au modèle (au parent)

clientModel.pseudo = "Pas de pseudo";

console.log(clientModel.pseudo);

// L'enfant hérite automatiquement de la nouvelle propriété du parent

console.log(client2.pseudo);

client2.pseudo = "Nini";

// Les deux objets avec chacun leurs valeurs

console.log(clientModel.pseudo);

console.log(client2.pseudo);

// Ajout d'une propriété 'on the fly' à l'enfant

client2.beaute = true;

console.log(client2.beaute); // true

// Le parent n'hérite pas d'une nouvelle propriété de l'enfant. L'enfant peut grandir indépendamment du parent.

// Alors que le parent transmet son évolution

console.log(clientModel.beaute); // undefined

**!** On *flirte* ici littéralement avec le **concept de généralisation / spécialisation** **!**

**5 - Objets instanciés à l'aide d'une fonction prototypale constructrice -   
A - Définition du modèle - Instanciation d'objets**

// Fonction prototypale constructrice

function Player(nom){

// Propriétés (ce que l'objet connait)

this.nom = nom;

this.score = 0;

// Méthodes (ce que l'objet sait faire)

this.afficher = function(){

console.log("Nom ==> " + this.nom + " Score ==> " + this.score);

}

}

// Utilisation

// On instancie deux joueurs (on crée deux objets de type 'Player')

var player1 = new Player("Bob");

var player2 = new Player("Monika");

player1.afficher(); // Dans la méthode afficher(), this représente Bob

player2.afficher(); // Dans la méthode afficher(), this représente Monika

**6 - Objets instanciés à l'aide d'une fonction prototypale constructrice -  
B - Ajout de propriétés généralistes hors modèle**

// suite (B)

// On rajoute une propriété 'on the fly' (dynamiquement, à la volée, hors modèle)

Player.prototype.pseudo = "Pas de pseudo";

// On rajoute une méthode 'on the fly' (dynamiquement, à la volée, hors modèle)

Player.prototype.addBonus = function(bonus){

this.score += bonus;

}

// On utilise cette propriété

player1.addBonus(100);

player2.addBonus(200);

// On affiche l' états des objets

player1.afficher();

player2.afficher();

// On regarde un peu les pseudos

player1.pseudo = "Bobby";

console.log(player1.pseudo);

console.log(player2.pseudo);

**7 - Objets instanciés à l'aide d'une fonction prototypale constructrice -  
C - Ajout de propriétés spécifiques hors modèle**

// suite (c)

// On peut attribuer des spécificités à un objet

player1.force = true;

player2.beaute = true;

// Monika est belle ?

console.log(player2.beaute); // true

// Bob est beau ?

console.log(player1.beaute); // Aucune idée !

// Monika est forte ?

console.log(player2.force); // Aucune idée !

// Bob est fort ?

console.log(player1.force); // true !

**8 - Composition (aggrégation)**

**On parle de COMPOSITION lorsqu'un objet possède une propriété qui elle-même est un objet. Un objet dans un objet !** .

**L'objet qui possède l'autre objet (qui aggrège !) est le COMPOSITE**.

**L'objet qui est possédé par l'autre (qui est aggrégé !) est le COMPOSANT**.

//---------------------------------------

// Déclaration d'un objet (inline)

var livre = {

titre : 'Les Misérables',

auteur : {

prenom : 'Victor',

nom : 'Hugo'

}

}

//---------------------------------------

// Utilisation de cet objet

// SVP ! Observez !

console.log(livre);

console.log('Titre: ' + livre.titre);

console.log('Auteur: ' + livre.auteur.prenom + " " + livre.auteur.nom);

//---------------------------------------

// Déclaration d'une fonction constructrice

function Book(titre, auteur){

this.titre = titre;

this.auteur = auteur;

}

function Auteur(p, n){

this.prenom = p;

this.nom = n;

}

//---------------------------------------

// Instanciation d'objets

var auteur1 = new Auteur("Victor", "Hugo" );

var book1 = new Book("Les Misérables", auteur1);

var book2 = new Book("Notre-Dame", auteur1);

// SVP ! Observez !

console.log(book1);

console.log('Titre: ' + book1.titre);

console.log('Auteur: ' + book1.auteur.prenom + " " + book1.auteur.nom);

**20 - Exercice 1 - Enoncé**

Exercice 1 - Enoncé

* Soit un modèle (logique) nommé MyImage avec les propriétés (attributs) suivantes
  + fichier
  + categorie
  + legende

SVP

* Implantez ce modèle (codez-le), à l'aide d'une fonction constructrice, et enregistrez-le dans un fichier autonome que vous nommez MonImage.js.
* Testez votre modèle. Instanciez 2 ou 3 objets de type 'MyImage'.
* *Merci et bon travail*

**Fiche Javascript n° 6 - *Arrays (Tableaux, Listes, Collections)***

**1 - A retenir**

A retenir

* Un array (tableau, liste, collection) est un ensemble d'éléments, accessibles avec un même nom
* On accède à l'élément d'un tableau avec sa clé (ou index)
* La clé est numérique et automatique, ET COMMENCE A 0 ! Dans le cas des arrays Javascript, on ne peut créer des clés 'custom'
* Un tableau peut contenir des éléments de tout type (y compris des éléments qui sont des objets, ou qui sont d'autres tableaux). Dans un même tableau, il est plus confortable d'avoir affaire à un type unique (genre un tableau de strings, ou encore de nombres, ou de clients

**Le format JSON est une représentation de données structurées. Attendez-vous à trouver de nombreuses déclarations d'arrays, (avec les crochets '[]'), dans une string json** .

|  |  |
| --- | --- |
| propriétés et méthodes importantes de array | |
| **Propriétés** | --- |
| **length** | le nombre d'éléments du tableau |
|  |  |
| **Méthodes** | --- |
| **push()** | ajoute un élément à la fin |
| **pop()** | retire un élément à la fin |
| **unshift()** | ajoute un élément au début |
| **shift()** | retire un élément au début |

**2 - Déclaration - Accès aux éléments**

// NB ! Un tableau contient des éléments.

// NB ! Les éléments d'un tableau sont accessibles avec leur CLE ou INDEX

// NB ! C'est le robot qui indexe numériquement la clé en commençant à 0 et en incréméntant de 1 pour l'élément suivant

//----------------------------------------------------------------

// Déclaration simple

var monTableau = [];

// Ecriture gentiment 'méprisée' par certains. Javascript déclare qu'il 'préfère' la déclaration à base de crochets

// var monTableau = new Array();

// Déclaration avec éléments

var listeVilles = ["Paris", "Nice", "Le Caire"];

var notesEtudiant = [15, 12, 11.2, 14.5, 9];

// Affichage d'élément

console.log(listeVilles[0]); // Paris

console.log(listeVilles[4]); // Elément inexistant !

// Modification d'un élément

listeVilles[0] = "Los Angeles";

console.log(listeVilles[0]); // Los Angeles

// Lecture de chaque élément avec une boucle classique

for(let i = 0; i < listeVilles.length; i++){

console.log(listeVilles[i]);

}

for(let i = 0; i < notesEtudiant.length; i++){

console.log(notesEtudiant[i]);

}

// Plus moderne (ES6) (méthode forEach qui pour chaque élément, lui applique un traitement ( grâce à la fonction de rappel (callback function) ) )

// NB !!! NOTEZ ICI UN COMPORTEMENT IMPORTANT (ET MODERNE):

// CE N'EST PLUS VOUS QUI DICTEZ LA MANIERE DE PARCOURIR LES ELEMENTS !!!

listeVilles.forEach(function(ville) {

console.log(ville + " " + ville.length);

});

notesEtudiant.forEach(function(note) {

console.log(note);

});

// Avec un lambda

listeVilles.forEach((ville) => {

console.log("lambda " + ville);

});

**3 - Ajout et suppression d'éléments (fin de liste et début de liste)**

//--------------------------------------

// Ajout d'élément(s) en fin de liste

listeVilles.push("Paris");

notesEtudiant.push(5.5, 5.0);

// Affichage (on vérifie)

listeVilles.forEach(function(v) {

console.log(v);

});

notesEtudiant.forEach(function(n) {

console.log(n);

});

//--------------------------------------

// Suppression d'élément(s) en fin de liste

listeVilles.pop();

notesEtudiant.pop();

notesEtudiant.pop();

notesEtudiant.pop();

// Affichage (on vérifie)

listeVilles.forEach(function(v) {

console.log(v);

});

notesEtudiant.forEach(function(n) {

console.log(n);

});

//--------------------------------------

// Ajout d'élément(s) en début de liste

listeVilles.unshift("Paris");

notesEtudiant.unshift(20, 19);

listeVilles.forEach(function(v) {

console.log(v);

});

notesEtudiant.forEach(function(n) {

console.log(n);

});

//--------------------------------------

// Suppression d'élément(s) en début de liste

listeVilles.shift();

notesEtudiant.shift();

listeVilles.forEach(function(v) {

console.log(v);

});

notesEtudiant.forEach(function(n) {

console.log(n);

});

// Tri

listeVilles.sort();

notesEtudiant.sort(); // // A améliorer !!!

listeVilles.forEach(function(v) {

console.log(v);

});

notesEtudiant.forEach(function(n) {

console.log(n);

});

**Fiche Javascript n° 7 - *DOM (Document Object Model)***

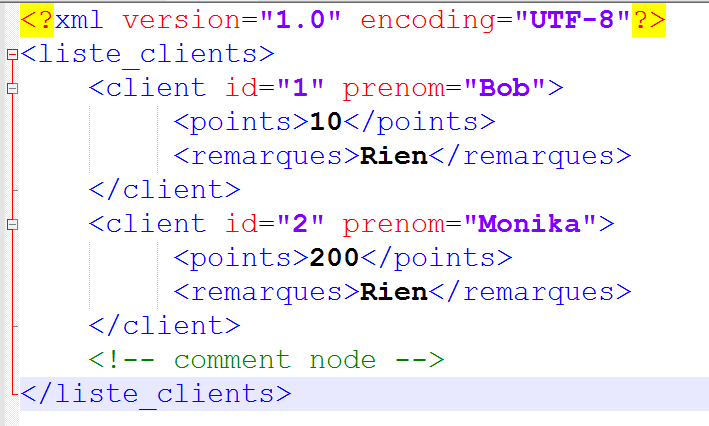
**1 - A retenir**

[Très bonne doc et synthèse du HTML DOM sur le W3Schools](https://www.w3schools.com/js/js_htmldom.asp)

* DOM (Document Object Model) est une logique **QUI TRANSFORME LES MARQUEURS HTML, OU XML, EN OBJETS (logiques)**. Dans le cas du DOM HTML, c'est le navigateur qui est responsable de l'exécution de cette logique
* Le dom doit avoir complètement (!) terminé son exécution, avant de pouvoir être utilisé. C'est une de ses contraintes: il doit connaitre l'ensemble du document
* Ensuite, attendez-vous à trouver des méthodes homogènes pour le manipuler, quelquesoit le langage utilisé
* Dans le cas de Javascript, c'est son objet ***document*** qui permet de communiquer avec le DOM (la page web composée d'objets, pas de marqueurs), et donc de manipuler cette dernière

Le DOM est une technique généraliste de transformation de marqueurs en objets. Attendez-vous à trouver une, ou des API pour le manipuler dans la plupart des langages de programmation (java, php, javascript ...)

**2 - Hiérarchie et types de nœuds dans l'arbre DOM**



structure d'un fichier xml

**Hiérarchie**

* ***parents/enfants***
* ***ascendants/descendants***
* ***frères/sœurs***

**Xml bien formé / Xml valide**

* ***xml bien formé (well-formed)***
  + pour tout tag ouvrant correspond un tag fermant de même nom
  + les balises doivent être correctement imbriquées
  + les valeurs des attributs sont entre quotes (att='value')
  + ...
* ***xml valide***
  + le document xml est couplé à un **XML Schema** et la structure du document XML respecte les règles et définitions déclarées dans le XML Schema

La plupart des parsers XML s'attendent à trouver un premier élément qui contient tous les autres ! On nomme cet élément ***root-element***, néanmoins, c'est un élément comme les autres en termes de propriétés

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| types de nœuds importants | | | | | |
| **Type** | **nodeType** | **nodeName** | **nodeValue** | **Description** | **Enfants** |
| **Document** | 9 | #document | null | Représente le document entier (le nœud-racine de l'arbre DOM) | Element (max un !), |
| **Element** | 1 | nom de l'élément | null | Représente un élément | Element, Text |
| **Attribute** | 2 | nom de l'attribut | valeur de l'attribut | Représente un attribut | Text |
| **Text** | 3 | #text | contenu (text + elem) du node | Représente du contenu textuel dans un élément ou dans un attribut | Text |
| **Comment** | 8 | #comment | contenu textuel du node | Représente un commentaire (non parsé) | non |
| **CDATASection** | 4 | #cdata-section | contenu textuel du node | Représente une section du document prise en compte par le parser comme de simples caractères. Le parser n'applique pas dans cette section les règles XML | non |

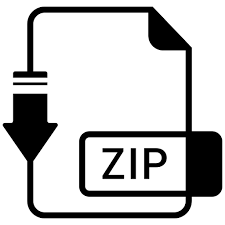
**3 - Propriétés et Méthodes importantes de l'objet *Document***

|  |  |
| --- | --- |
| propriétés et méthodes importantes de document | |
|  | ***Propriétés*** |
| **body** | sélectionne l'élément 'body' du document web |
| **forms** | retourne une collection contenant tous les éléments  Haut du formulaire  de la page  Bas du formulaire |
| **images** | retourne une collection contenant tous les éléments de la page |
| **readyState** | retourne l'état d'avancement de la requète (envoi, traitement, réception) |
|  |  |
|  | ***Méthodes*** (sélection d'élément(s)) |
| **getElementById('xxx')** | sélectionne l'élément de la page web, qui possède un attribut 'id' avec la valeur 'xxx' |
| **getElementsByName('xxx')** | sélectionne le ou les éléments de la page web, qui possède un attribut 'name' avec la valeur 'xxx' (retourne une *NodeList* !) |
| **getElementsByTagName('xxx')** | sélectionne le ou les éléments de la page web, dont le nom du tag est 'xxx' (retourne une *NodeList* !) |
| **getElementsByClassName('xxx')** | sélectionne le ou les éléments de la page web, qui possède(nt) un attribut 'class' avec la valeur 'xxx' (retourne une *NodeList* !) |
| **querySelectorAll("x y z");** | sélectionne le ou les éléments de la page web, qui répondent au sélecteur css 'x y z' (retourne une *NodeList* !) |
|  | ***Méthodes*** (manipulations d'éléments) |
| **createElement("xxx");** | crée un nœud de type *element* nommé 'xxx' |
| **createTextNode("xxx");** | crée un nœud de type *text* nommé 'xxx' |

**4 - Propriétés et Méthodes importantes de l'objet *Element***

|  |  |
| --- | --- |
| propriétés et méthodes importantes de Element | |
|  | ***Propriétés*** |
| **attributes** | retourne une collection d'attributs du nœud (NamedNodeMap) |
| **childNodes** | retourne une collection de nœuds, enfants du nœud spécifié (NodeList) |
| **firstChild** | retourne le premier nœud, enfant du nœud spécifié (Node) |
| **id** | pointe vers l'attribut id de l'élément (permet de lire/écrire sa valeur) |
| **innerHTML** | pointe vers le contenu de l'élément (permet de lire/écrire du contenu) |
| **lastChild** | retourne le dernier nœud, enfant du nœud spécifié |
| **nextSibling** | retourne le prochain nœud, frère (soeur) du nœud spécifié |
| **nodeName** | retourne le nom de l'élément (ou la valeur de l'attribut) |
| **nodeType** | retourne le type de nœud de l'élément |
| **nodeValue** | retourne la valeur du nœud de l'élément |
| **parentNode** | retourne le nœud, parent du nœud spécifié (Node) |
| **previousSibling** | retourne le précédent nœud, frère (soeur) du nœud spécifié |
|  |  |
|  | ***Méthodes*** |
| **addEventListener('event', handler)** | abonne l'élément à l'évènement |
| **appendChild(node)** | ajoute un nœ à l'élément |
| **getAttribute('id')** | retourne la valeur d'un attribut (nom spécifié en paramètre) d'un élément |
| **getElementsByTagName('xxx')** | sélectionne le ou les éléments de la page web, dont le nom du tag est 'xxx' (retourne une *NodeList* !) |
| **getElementsByClassName('xxx')** | sélectionne le ou les éléments de la page web, qui possède(nt) un attribut 'class' avec la valeur 'xxx' (retourne une *NodeList* !) |
| **hasAttributes()** | retourne true ou false pour l'existence, ou pas d'attributs pour un élément |
| **hasChildNodes()** | retourne true ou false pour l'existence, ou pas d'éléments enfants et/ou descendants pour un élément |
| **querySelectorAll("x y z");** | sélectionne le ou les éléments enfants/descendants de l'élément, qui répondent au sélecteur css 'x y z' (retourne une *NodeList* !) |
| **removeChild(node)** | retire un nœud à l'élément |

**5 - Manipulation d'éléments difficiles**

[](http://itformations.com/campus/medias/fiches/JS/ELEMENTS_DIFFICILES.zip)

Exemples pour manipuler les boutons radio, cases à cocher, listes déroulantes

**10 - Exemples**

// Page web

// exemple.js

// Evènement 'click' sur le bouton

document.getElementById('bt1').onclick = processClick;

//document.getElementById('bt1').addEventListener('click', processClick) ;

//----------------------------

//----------------------------

function processClick(){

// On met du contenu dans le titre

document.getElementById("out").innerHTML = "Salut !";

// On affiche en console le contenu du titre

var titre = document.getElementById("out").innerHTML;

console.log(titre);

//console.log(document.getElementById("out").innerHTML);

}

// ----------------------------------------------------

// Affichage de la taille de l'écran en direct

onresize = infosSize;

function infosSize(){

document.getElementById("out").innerHTML = "Width " + innerWidth + " Height " + innerHeight;

}

## Fiche Javascript n° 8 - Dom Events

### 1 - Objet Event

|  |  |
| --- | --- |
| propriétés et méthodes importantes de Event | |
|  | **Propriétés** |
| **currentTarget** | retourne l'élément dont le listener a été déclenché (est souvent le même que pour 'target') |
| **target** | retourne l'élément émetteur de l'évènement |

### 2 - Evènements importants

|  |  |
| --- | --- |
| Evènements DOM importants | |
| **blur** | quand l'utilisateur quitte un champ texte |
| **change** | quand l'utilisateur change l'option sélectionnée dans une liste déroulante |
| **click** | quand un élément est cliqué (souvent un bouton) |
| **dblclick** | quand un élément est double-cliqué (souvent une option) |
| **drag, dragend, dragenter, dragleave, dragover, dragstart, drop** | évènement de cliquer/glisser/lacher (drag and drop) |
| **focus** | quand l'utilisateur rentre dans un champ texte |
| **keydown, keypress, keyup** | évènement liés au clavier |
| **load** | page chargée (Window), ou média chargé (Image, Video, Sound) |
| **mousedown, mouseenter, mouseleave, mousemove, mouseover, mouseout, mouseup** | évènement liés à la souris |
| **resize** | quand la page est redimensionnée |
| **scroll** | quand la barre de défilement d'un élément est manipulée (Window, Element) |
| **submit** | quand un formulaire est envoyé |
| **touchcancel, touchend, touchmove, touchstart** | évènement liés au écrans tactiles |

**Fiche Javascript n° 20 - *TP ' Jour/Nuit '***

**1 - Enoncé**

exercice DayLight

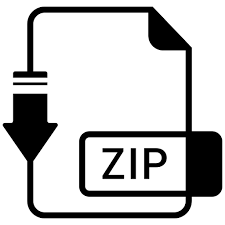
Nous avons:

* une page web (intelligente) avec les composants suivants
  + un titre qui dit s'il fait jour ou nuit
  + une image qui représente une ambiance lumineuse (pour le jour) ou une ambiance sombre (pour la nuit)
  + une ambiance globale lumineuse (pour le jour) ou une ambiance globale sombre (pour la nuit)
  + un bouton qui permet de switcher à volonté entre le jour et la nuit. Le libellé du bouton annonce l'action ('faire le jour' quand il fait nuit, 'faire la nuit' quand il fait jour



écran jour et écran nuit

**2 - Solution**

[](http://itformations.com/campus/medias/fiches/JS/JOUR_NUIT.zip)

TP 'DayLight' - Solution (matériel + sources html et js)

**Fiche Javascript n° 22 - *TP Formulaire de Pizza***

**1 - Enoncé**

exercice Pizza Form

Soit un formulaire de commande de pizza:

* un nom (min 10 caractères - max 20 caractères)
* un mot de passe (entre 5 et 10 caractères)
* une taille (aucun bouton n'est 'pre-checked')
* des ingrédients (min 3)
* un moyen de paiement (une option 'SVP, sélectionnez ...' est 'pre-selected')

SVP

* Validez-le formulaire selon les critères mentionnés ci-dessus. Le bitmap ci-dessous montre le formulaireavec tous les champs en erreur. Si le formulaire est valide, on le 'submit'. S'il ne l'est pas, on indique les champs erronés
* Merci et bon travail !



le formulaire de commande de pizza, avec tous ses champs en 'erreur' (non-valides)

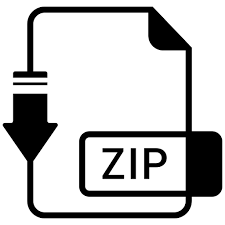
**Hints & Tips**

* obtenir la valeur d'un bouton radio (valeur unique): boucler sur les boutons et vérifier leur état *checked*. Si checked: prendre la valeur.
* obtenir la valeur de cases cochées (valeurs multiples): boucler sur les cases et vérifier leur état *checked*. Si checked: prendre la valeur.
* obtenir la valeur d'un select (valeur unique ou valeurs multiples): boucler sur les options et vérifier leur état *selected*. Si selected: prendre la valeur.

**Hints & Tips**

Envoyer le formulaire ou pas ? Décider d'envoyer de toute façon, et 'annuler' cette décision au moindre mécontentement ? ... pourquoi pas ...

**2 - Solution**

[](http://itformations.com/campus/medias/fiches/JS/PIZZAFORM.zip)

TP Pizza - Solution (matériel + sources html et js)