# **Memo JS**

## Type Primitif

* *Imprécision des calculs à nombre flottant (0.01+0.02 != 0.03) : calculer sur des entier quand possible (puis /10 /100 etc…)*
* *Les objet et tableau sont passés par référence et les type primitif sont passé par valeur !*

|  |  |  |
| --- | --- | --- |
| **Scope : global** | **Scope : block** | **Scope : global | read ONLY** |
| **var Mavar = 10** | **let Mavar = 10** | **const Mavar = 10** |

Pour afficher le nom d’une variable dans une phrase facilement ont utilise les `` (ALT GR + 7) et on utilise ${nom}, ex :

**Let leNom = jean ; console.log( `${leNom} est gentil` ) ;**

## Tableau

*Permet de stocker des variables de façon ordonée ex :*

|  |  |
| --- | --- |
| *Let monTableau = [‘marc’, ‘claire’, ‘philipe’] ;* | *Let tabMulti = [ [0,1,5],[8,7,2] ]* |
| *monTableau[0] // renverra ‘marc’* | *tabMulti[1][2] // renverra 2* |
| *montab2 = [‘jack’,’bob’, … monTableau] // ajoute les valeurs du tableau monTableau dans montab2* | |

#### méhode et info sur tableau :

|  |  |
| --- | --- |
| *tab.lenght // renvois la taille du tableau* | *tab.push(value) // ajoute un élément à la fin du tableau* |
| *tab.unshift(value) // ajoute un élément en debut de tableau* | *tab.pop() // supprime le dernier elements* |
| *tab.forEach( function(element){ }) // element representera chaque element du tableau* | |
| *Const tab2 = tab.map( x => x\*2) // créer un nouveau tableau avec chaque valeur \*2* | |
| *Tab.sort( (a,b) => a-b ) // tri par ordre croissant, b-a pour ordre decroissant, par defaut sort() tri par caractere* | |

*forEach() ne retourne rien et est 60% moins rapide que map() qui lui renvois le tableau modifier*

## Objet

* En JavaScript tout est objet, par exemple, une chaine de caractère est une instance de String, et toute les instances de string « héritent » des méthodes de string.

|  |  |
| --- | --- |
| **« azdea ».length** // renverra la longueur de la chaine de caractères | **« salut ».toUpperCase()** // renverra SALUT |

* Les objet JavaScript sont écrit en JSON, ce sont des associations de paires clés/valeur, ex :

let myBook = {

title : ‘mon super titre’,

numberOfPage : 250

}

## Condition & boucle

**If( a == b ) { } else if { } else { }**

**While (a == b) { }**

**For ( var i = 0 ; i < 3 ; i++) { }** // **For ( var i = 0 ; i < tab.length  ; i++) { }** // simule un forEach

*for (let i in passengers) {console.log(« et voici maintenant : passengers[i] »)} ;*

*for (let passenger of passengers){ console.log(« et voici maintenant : passenger) ;*

try {} catch(error){}

## Fonction

*Les fonctions on accès au variable extérieur, si on les utilise on appel ces fonctions des closures, en revanche l’extérieur n’a pas accès au variable utilisé à l’intérieur des fonctions. Si on utilise une variable extérieure sans mettre var devant, on modifie la valeur de la variable globale, en revanche si on met var devant, elle n’est pas modifiée*

**Function demo(a) { return a\*3 ; }**

**Function demo( a = 3 ) { return a\*3 ; }** // valeur par défaut pour le paramètre a

**Var demo2 = function (a) { return 3\*a }** // on stock la fonction dans une variable

### Fonction flecher : les ligne suivante font pareil que **Var demo2 = function (a) { return 3\*a }**

Var demo2 = ( a => return a\*3) // si plusieurs arguments ou aucun mettre () si plusieur ligne mettre {}

Modifie le contexte de this : this ne renvois pas la fonction mais celui qui l’a appeler

## Hoisting

*Le JavaScript à une manière particulière de traiter les variables et les fonctions. Il déplace les déclarations en haut du contexte courant. Autrement dit il détecte toute les variable de nos fichier, et les déclare en haut du script sans les assignation qui ont lieu seulement là où on à affecter une valeur, idem pour les fonction qui sont appelable dès le début du fichier même si elle ne sont placer qu’à la fin, sauf si on les déclare tel que var x =* ***function ( ) { }*** *, une classe en revanche n’est pas remonter !*

## Classe

*On peut utiliser des classes en JS depuis ecmascript 2015 tel que :*

**class rectangle{** // on peut aussi utiliser le extends tel que : **class rectangle extends Forme {…}**

**constructor(hauteur, largeur){**

**this.hauteur = hauteur ;** // on peut override le constructeur tel que : **super()**

**this.largeur = largeur ;**

**}**

**get area() { return this.area ; }**

**calcArea() { return this.largeur \* this.hauteur ; }**

**static() {return « je suis static ! » ;}** // méthode de class !

**}**

Et pour instancier une classe : **var rect = new rectangle() ;** //

## this

*this est une sorte de super variable qui prendra differente valeur suivant le contexte dans lequelle elle est utilisé.*

*Par défaut this prend comme valeur la variable global (window sur navigateur, global sur nodejs).*

*Si une méthode est appelée depuis un objet, this prendra comme valeur l’objet, ex :*

***Var eleve = { maFonction : function() { return this } }*** *// eleve.maFonction() retournera l’instance de eleve{}*

*TIPS : This et eventListener dans une class : utiliser l’arrow function pour que this sois bind sur la class ex :*

*Machin.addEventListener( ‘click’, ()=>{this.methodeDeMaClasse}*

## L’objet window

Sur le navigateur l’objet global est l’objet window, qui contient un ensemble de méthode et de propriétés, ex :

**window.innerWidth** // renvois la largeur de la fenetre

**window.alert(‘ooops’)** // affiche une alerte avec le message ooops

**window.confirm(‘are u sur ?’)** // affiche une fenetre de confirmation

**window.prompt(‘entré votre nom’)** // affiche un champs qui permet de rentré une valeur

**window.setInterval( function() {…}, 1000)** // le code … sera executé toute les 1000ms

**var timerB = window.setTimeout(function() {…}, 3000** ) // le code … sera executé une fois au bout de 3000ms

**window.clearInterval(timerB)** // stop le timerB définie à la ligne au-dessus

## DOM

L’objet document permet de récupérer des élément HTML et de les manipuler, ex :

**document.body** // récupère l’élément body

**document.getElementById(‘demo’)** // sélectionne l’élément avec l’id démo

**document.querySelector(‘.demo’)** // sélectionne le premier élément correspondant au sélecteur CSS

**document.querySelectorAll(‘.demo’)** // sélectionne tous les élément correspondant au sélecteur CSS

**document.getElementsByClassName(‘demo’)** // sélectionne tous les éléments avec la classe démo

**document.getElementsByTagName(‘p’)** // sélectionne tous les éléments <p>

On peut récupérer les éléments et les parcourir comme un tableau :

**Var elements =** **document.querySelectorAll(‘.demo’)**

**For (var i = 0 ; i < elements.length ; i++) { var element = elements[i] }**

Une fois un élément récupérer on peut obtenir des informations :

**Element.getAttribut(‘attribut’)** // récupère la valeur d’un attribut

**Element.style** // récupère les styles associés à l’élément

**Element.classList** // permet de récupérer la liste des classe

**Element.offsetHeight** // donne la hauteur de l’élément

On peut également modifier un élément

**Element.setAttribute(‘href’, ‘http://monsite.fr’)**

**Element.style.fontSize = ‘24px’**

**Element.classList.add(‘red’)** // ajoute la classe red à l’élément

*Quand on récupère un élément dans la page, on récupère en fait un nœud dans le dom. Il est alors possible de naviguer dans l’arbre à partir d’un élément préalablement récupérer, ex :*

***Element.appendChild(enfant)*** *// ajoute un élément à un autre*

***Element.removeChild(enfant****) // supprime un enfant*

***Element.textContent = ‘Salut’*** *// change la valeur du nœud texte*

***Element.innerHtml*** *// renvoie le contenu HTML de l’élément*

***parentElement.insertBefore(nouvelElement, refElement)***

## évènement

Un évènement permet de déclencher une action dès lors qu’un type d’évènement se produit (clic, etc…). ex :

**Element.addEventListerner(‘click’, function(e) { window.alert(‘Vous avez cliqué sur le lien’)**

**e.preventDefault()** // annule l’évènement

**e.stopPropagation()** // empêche l’évènement de remonter vers les éléments parents

**e.target** // contient l’élément sur lequel on a cliqué

**e.currentTarget** // contient l’élément sur lequel on a greffé l’écouteur

**) }**

Supprimer un évènement :

**Element.removeEventListener(‘click’, next)**

## Canvas

La zone de canvas doit etre tout d’abord définis dans le html tel que :

**<canvas id= «canvas» width= «500» height= «500»></canvas>**

Puis nous recuperons en JS le node et enfin le contexte avec lequel on va travailler :

**Var ctx = document.getElementById(‘canvas’).getContext(‘2d’) ;**

A partir de la on peux dessiner des forme dans notre canvas :

**Ctx.fillStyle = ‘#FFF’ ;** // definie la couleur de remplissage

**Ctx.lineWidth = 1 ;** // definie la largeur du contour (stroke)

**Ctx.strokeStyle = ‘red’ ;** // définie la couleur du contour

**Ctx.setLineDash([3,3]) ;** // definie une ligne pointillé (3px de long, 3px d’espace), envoyer un [] pour réinitialisé

### Dessiner un carré ou un rectangle :

**Ctx.fillRect(x,y,w,h**) // créer un rectangle au coodonées (x,y) de longeur w et hauteur h

**Ctx.strokeRect(x,y,w,h)** // rectangle surligné

**Ctx.clearRect(x,y,w,h)** // rectangle vide qui efface les partie qu’ils couvre

**Ctx.rect(x,y,w,h)** // rectangle, doit faire appel a stroke() et fill() pour faire apparaitre

### Dessiner un triangle :

**Ctx.beginPath()** // indique qu’on commence un nouveau tracé

**Ctx.moveTo(100,100)** // point de départ du tracé

**Ctx.lineTo(150,100)** // trace une ligne du point de départ a 150,100

**Ctx.lineTo(150,150)** // trace une ligne du dernier point de la precedente ligne jusqu’à 150,150

**Ctx.closePath()** // clos le dessin (optionnel)

**Ctx.fill() ;** // remplis le dessins par la couleur de ctx.fillStyle

**Ctx.stroke()**// met les contours définis dans lineWidth et strokeStyle

### C:\Users\Modji\Desktop\img_canvas_arcto.pngForme courbé : arcto()

**Ctx.beginPath()** // indique qu’on commence un nouveau tracé

**Ctx.moveTo(20,120)** // point de départ du tracé

**Ctx.arcTo(20,30,160,120,20) ;** // arcTo(x1,y2,x2,y2,radius)

**Ctx.stroke()** // et on dessine les contours

### Dessiner un rond : arc()

**Ctx.beginPath()** // indique qu’on commence un nouveau tracé

**Ctx.arc(150,40,30,0,Math.PI\*2,true)** //arc(x,y,rayon,a1,a2,c) a1 angle début, a2 angle de fin et c = sens rotation

**Ctx.stroke()** // et on dessine les contours