

Chapitre 2

Concepts de base de JavaScript

Enseignante: **Amel TRIKI**

Année Universitaire 2024 - 2025

Plan

- Intégration du JavaScript dans une page web
- Syntaxe de JavaScript
- Les entrées/sorties des données
- Les variables et les constantes
- Principaux types de données
- Les opérateurs
- Structures conditionnelles
- Structures itératives
- Les fonctions

Intégration du JavaScript dans une page web

Le code JavaScript peut être placé:

Dans la page HTML

- Dans le **HEAD**:
`<script> </script>`
- Dans le **BODY**
`<script> </script>`
- Dans une **URL**
- Valeur d'un attribut pour répondre à un ***événement***

Dans un fichier externe



Fichier .js



Appel du script dans HEAD

Intégration de JS dans une page HTML (1/2)

<script> </script> dans le HEAD

```
<head>  
  <script>  
    alert('Script dans l\'entête');  
  </script>  
</head>
```

<script> </script> dans le BODY

```
<body>  
  <script>  
    alert('Script dans le corps de la page');  
  </script>  
</body>
```

Intégration de JS dans une page HTML (2/2)

Dans une URL

- Syntaxe: `Message`
- Exemple: `Lien `

Valeur d'un attribut pour répondre à un événement

- Syntaxe: `<Balise onEvenement= "code javascript" />`
- Exemple: ``

Intégration de JS dans un fichier externe

Etape 1

Ecrire le code dans un fichier ayant pour extension **.js**

// Contenu du fichier externe.js

```
alert('Script dans un fichier externe');
```

Etape 2

Lier le fichier .js avec la page html dans l'**entête** en précisant son chemin dans l'attribut **src** de la balise **<script>**

```
<head>
```

```
  <script src="externe.js"> </script>
```

```
</head>
```

Exemple complet

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" /> <title> Intégration JAVASCRIPT </title>
    <script src="exemple1.js"> </script>
    <script>
      alert('Script dans l'entête');
    </script>
  </head>
  <body>
    <script>
      alert('Script dans le corps');
    </script>

    <a href="javascript:alert('Script dans une url');"> Lien </a>

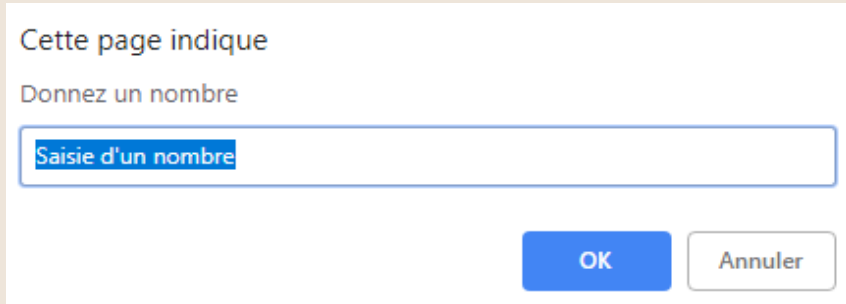
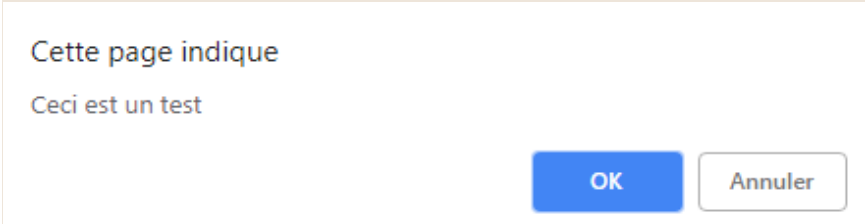
    
  </body>
</html>
```

Syntaxe de JavaScript

- ✓ Syntaxe proche du langage C
- ✓ JavaScript est sensible à la casse
- ✓ Dans le cas d'un bloc d'instructions, les instructions sont:
 - ✓ Encadrées par des `{ }`
 - ✓ Séparées par des `;`
- ✓ Les commentaires sur :
 - ✓ Une seule ligne: `//`
 - ✓ Plusieurs lignes: `/*`
.... `*/`


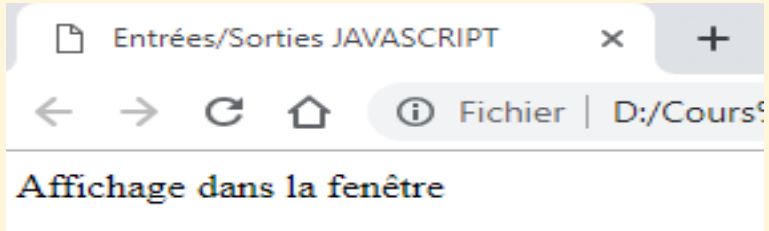
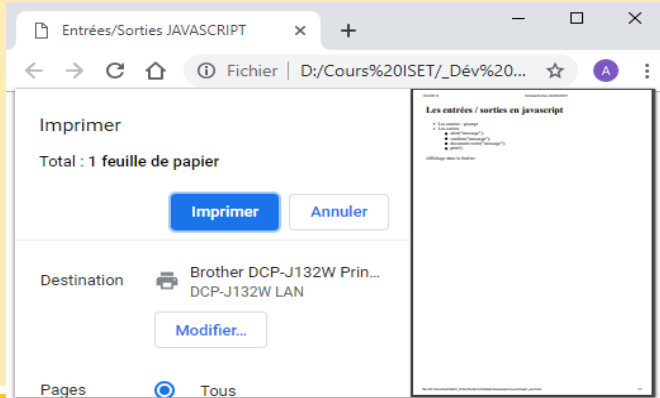
Les entrées/sorties des données (1/2)

Les entrées

Instruction	Résultat
prompt ("message", "message d'invite boîte facultatif "); ➡ prompt ("Donnez un nombre", "Saisie d'un nombre");	
confirm ("message"); ➡ confirm ("Ceci est un test");	

Les entrées/sorties des données (2/2)

Les sorties

Instruction	Résultat
<code>alert("message");</code> ➡ <code>alert(" Ceci est un test");</code>	
<code>document.write("message");</code> ➡ <code>document.write(" Affichage dans la fenêtre");</code>	
<code>print();</code> <code>/* Imprime le contenu de la fenêtre en question*/</code>	

Mots réservés dans JS

boolean	else	let	switch
break	extends	new	this
case	false	null	throw
catch	final	private	true
class	for	protected	try
const	function	public	typeof
continue	if	return	var
default	in	short	void
do	instanceof	static	while
delete	int	super	with

Les variables (1/2)

- ✓ Les variables sont **sensibles à la casse**
- ✓ Le nom d'une variable commence par
 - ✓ Une lettre, Un trait de soulignement `_` ou le signe dollar `$`
 - ✓ Peut comporter des lettres, des chiffres, `_` et des `$`
- ✓ La déclaration d'une variable **n'est pas obligatoire**
- ✓ Une variable peut être déclarée différemment pour avoir différentes portées:
 - ✓ **var**: globale ou locale à une fonction
 - ✓ **let**: bloc dans lequel la variable est déclarée

Exercice

Indiquer à chaque fois si la déclaration de la variable est correcte ou non

- `let $longueur;` 
- `long var;` 
- `var Var;` 
- `let _to?tal;` 
- `var in;` 
- `let __$;` 
- `let While;` 

Les variables (2/2)

	var	let
Déclaration	var nomVariable;	let nomVar;
Affectation	nomVariable = valeur;	nomVar = valeur;
Déclaration avec initialisation	var nomVariable = valeur;	let nomVar = valeur;
Redéclaration	possible	interdite
Exemples	var radian =5.2; var chaine= "message"; var nombre = 12;	let surface=5 *perimetre; let _val= surface*2; let \$find = true;

var / let

Example:

```
var a = 6;
let b = 3;
if(true)
{
    var a = 8;
    let b = 5;
    let c = 5;
    document.write(a);           // 8
    document.write("<br />" + b); // 5
}
document.write("<br />" + a);     // 8
document.write("<br />" + b);     // 3
document.write("<br />" + c);     // Uncaught Reference
                                // Error: C is not defined
```

Les constantes

- ✓ Une constante est déclarée avec le mot clé **const**
const NOM_CONSTANTE = valeur;
- ✓ Une constante a pour portée le **bloc dans lequel elle est définie**, pour qu'elle soit globale, il faut la déclarer hors toute fonction

Exemple:

```
if(true)
{   const CC = 10;
    console.log(CC);           // 10
}
console.log(CC);              // Erreur
```


Remarques (1/2)

- ✓ Une variable déclarée mais non initialisée a pour valeur **undefined**
- ✓ Une variable peut être utilisée sans être déclarée

Exemple:

```
nb = 10;  
alert(nb);  
var nb;
```

// 10
// Hoisting (Hissage)

➡ Effet du Hoisting

Exemples	Affichage dans la fenêtre console
console.log(val);	Uncaught ReferenceError
console.log(val); var val;	undefined

Remarques (2/2)

En mode **strict** qui est sécurisé (identifiable par la directive **"use strict"** ou **'use strict'**), il est **interdit** d'utiliser des variables **non déclarées**

Exemples:

```
function test()
{
    val = 14;
    console.log(val);    // 14
}
```

```
function test()
{
    'use strict';
    val = 14;
    console.log(val);    // Reference Error
}
```

Principaux types de données

Une variable a le type de la donnée qui lui est assigné

Type	Description
Number	Type numérique: entier, nombre positif, négatif, scientifique, décimal,...
String	Chaîne de caractères
Boolean	true / false
Undefined	Type d'une variable déclarée mais non assignée
Null	Type qui précise explicitement que la valeur est nulle
Object	Collection non ordonnée de paires clé/valeur
Function	Type fonction

L'opérateur typeof (1/2)

L'opérateur **typeof** renvoie une chaîne de caractères indiquant le **type** d'une opérande

Syntaxe:

```
typeof opérande  
typeof (opérande)
```

Exemples:

```
console.log(typeof "11.25");           // string  
console.log(typeof 10.2);               // number  
console.log(typeof (5/0));              // number
```

L'opérateur typeof (2/2)

Exemples:

```
let a;
```

```
console.log(typeof(a));
```

// undefined

```
console.log(typeof (a==undefined));
```

// boolean

```
console.log(typeof null);
```

// object

```
function test() { alert('essai');}
```

```
let personne = {nom:'Ali', age:10}
```

```
console.log(typeof(personne));
```

// object

```
console.log(typeof(test));
```

// function

Le type number

Valeurs acceptées

- Toutes les valeurs numériques décimales ou non
- **Infinity**, **-Infinity** et **NaN** (Not a Number)
- Les nombres hexadécimaux: **0x17**, **0xFE14**, **0x2A**
- Les nombres octaux: **0o627**, **0o25**, **0o431**
- Les nombres binaires: **0b10**, **0b101**, **0b100001**

Exemples:

alert (15/0); **// Infinity**

alert (0x1A); **// 26**

alert (0o125); **// 85**

Conversion de types

Conversion implicite

- string → number
- number → string
- Null, false → 0
- 0, undefined, null et NaN → false
- Autres valeurs → true

Conversion explicite

Fonctions prédéfinies

- **parseFloat(chaine)**
- **parseInt(chaine)**
- **Number(..), String(..), Boolean(..)**
- **isNaN(valeur)**

Exemples de conversions implicites

Instruction

```
alert (" number to string: "+5+1+12);
```

```
alert ("string to number: "+3*"7");
```

```
alert (null *5 + " -- " + (null+5));
```

```
alert(Boolean(0));
```

```
var x; alert(Boolean(x));
```

```
alert(Boolean(NaN));
```

```
alert(Boolean("xyz"));
```

```
alert(Boolean(15));
```

Affichage

number to string: 5112

string to number: 21

0 -- 5

false

false

false

true

true

Exemples de conversions explicites

Instruction

Affichage

```
alert (parseInt("135.89En"));
```

135

```
alert (parseInt("e36"));
```

NaN

```
alert (parseFloat("137.58"));
```

137.58

```
alert (Number("15.34"));
```

15.34

```
alert (isNaN(parseInt("T1101")));
```

true

```
alert (isNaN(parseInt("14")));
```

false

```
let a = 10;
```

```
alert (eval("5*a" + "+a"));
```

60

Le type string (1/2)

- ✓ Une chaîne de caractères doit être écrite entre 2 guillemets " " ou 2 apostrophes ' '
- ✓ La **concaténation** se fait avec l'opérateur +

Exemple 1:

Donner le résultat de l'affichage:

alert("Bonjour");	// Bonjour
alert('Bonjour');	// Bonjour
alert("Bonjour 'TII0I' ");	// Bonjour 'TII0I'
alert('Bonjour "TII0I" ');	// Bonjour "TII0I"
alert("Bonjour 'TII0I' ');	// Erreur

Le type string (2/2)

Exemple 2:

Compléter le code pour obtenir l'affichage suivant

Affichage

Bonjour 'TII0I

Bonjour "TII0I"

Bonjour

TII0I

Bonjour \ 'TII0I

Bonjour TI I0I

Bonjour TI I0I

Code Correspondant

```
alert('Bonjour \ 'TII0I');
```

```
alert("Bonjour \" TII0I \");
```

```
alert("Bonjour\n TII0I");
```

```
alert('Bonjour \\\ 'TII0I');
```

```
alert('Bonjour' + 'TI I0I');
```

```
alert('BonjourTII'+ 0 + I);
```

Les gabarits de chaînes de caractères

Utilisation d'accents grave ` ` pour:



Définir des chaînes **multilignes**

Exemple: `console.log(`Phrase sur
2 lignes`);`



Phrase sur
2 lignes



Interpoler / **intégrer** des expressions `.....${expression} ...``

Exemple:

Soient $a = 15$ et $b = 23$, on veut afficher sous le format **15 + 23 = 38**

```
let a = 15, b = 23;  
alert(a + "+" + b + " = " + (a+b));
```



```
alert (`${a}+${b} = ${a+b}`);
```

Principaux opérateurs

- **Arithmétiques:** +, - , *, /, %, ++, --, -(négation), ** (puissance)
- **Affectation:** =, +=, -=, *=, /=, %=, **=, >>=, <<=, >>>=, |=, &=, ^=
- **Logiques:** &&, ||, !
- **Comparaison:** >, >=, < , <=, ==, !=, ===, !==, ?
- **Binaires:** &, |, ^ (XOR), ~ (NOT), >> , <<, >>>
- **Unaires:** typeof, delete, void
- **Concaténation:** + (entre les chaînes)

Les opérateurs == vs === et != vs !==

- == même valeur
- === même valeur **et même type**
- != valeurs différentes
- !== valeurs différentes **ou** types différents

Exemples:

```
console.log(1=='1');
```

true

```
console.log(1==true);
```

true

```
console.log("1"==true);
```

true

```
console.log(1.0==true);
```

true

```
console.log(null==undefined);
```

true

```
console.log(" "==false);
```

true



false avec l'opérateur
===

Les structures conditionnelles (1/2)

```
if(condition)
{   Bloc d'instructions   }
```

```
if(condition)
{   Bloc d'instructions   }
else
{   Bloc d'instructions   }
```

```
if(condition)
{   Bloc d'instructions   }
else if (condition)
{   Bloc d'instructions   }
....
else
{   Bloc d'instructions   }
```

Les structures conditionnelles (2/2)

```
switch(expression)
{
    case valeur1: instructions;
                break;
    case valeur2: instructions;
                break;
    ....
    default : instructions;
}
```

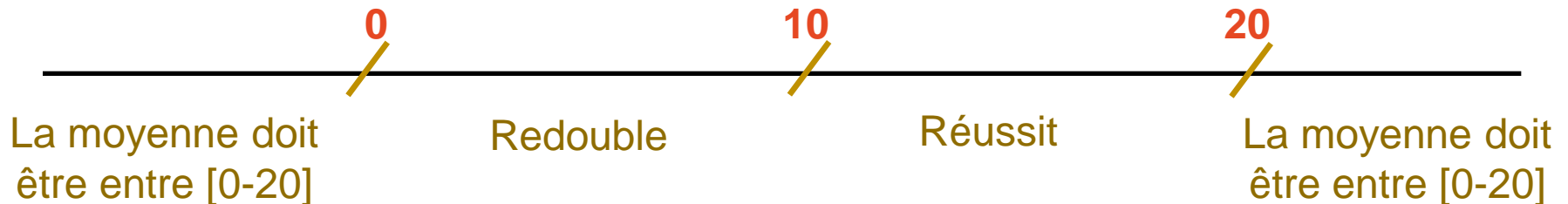
Structure ternaire

(condition) ? ...bloc du if : ...bloc du else

Exercice

Ecrire un script en JavaScript permettant de :

- Saisir une moyenne
- Afficher l'un des messages suivants selon la valeur de la moyenne



- Afficher un message d'erreur en cas de saisie incorrecte sous le format:
« n'est pas un réel »

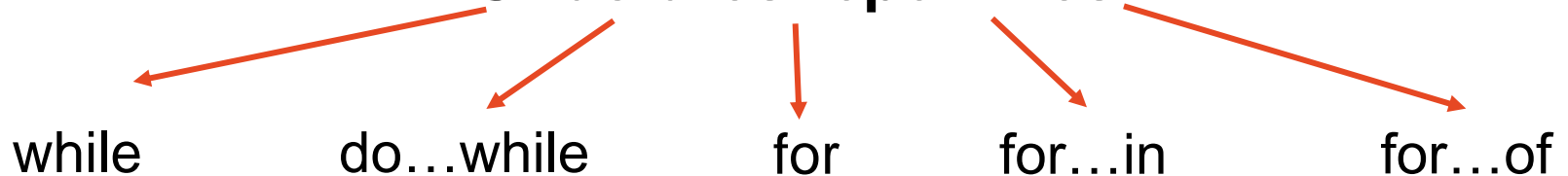
Corrigé

```
let moyenne = prompt("Donnez une moyenne");

if (!isNaN(moyenne)) {
  moyenne = Number(moyenne);
  if (moyenne < 0 || moyenne > 20)
    alert("La moyenne doit être entre [0-20]");
  else if (moyenne < 10)
    alert("Redouble");
  else
    alert("Réussit");
}
else
  alert(`${moyenne} n'est pas une nombre valide`);
```

Les structures itératives (1/3)

Structures répétitives



```
while(condition)
{   Bloc d'instructions   }
```

```
do
{   Bloc d'instructions   }
while(condition de répétition);
```

```
for(initialisation ; condition; incrémentation)
{   Bloc d'instructions   }
```

Les structures itératives (2/3)

for...of: permet de parcourir les éléments d'un objet itérable et renvoie la liste des valeurs de l'objet

```
for(variable of objetItérable)
{   Bloc d'instructions   }
```

Exemple:

```
let tab=[15, 87, 93];
```

```
for(let val of tab)
```

```
alert(val);
```

```
// affiche 15 puis 87 puis 93
```

Les structures itératives (3/3)

for...in: permet d'itérer sur les propriétés énumérables d'un objet et renvoie la liste des clés de l'objet

```
for(variable in objet)
{ Bloc d'instructions }
```

Exemple:

```
let tab=[15, 87, 93];
```

```
for(let indice in tab)
```

```
    alert(indice);
```

```
// affiche 0 puis 1 puis 2
```

Utilisation de break

break; permet de quitter l'itération en cours et sortir de la boucle

Exemple:

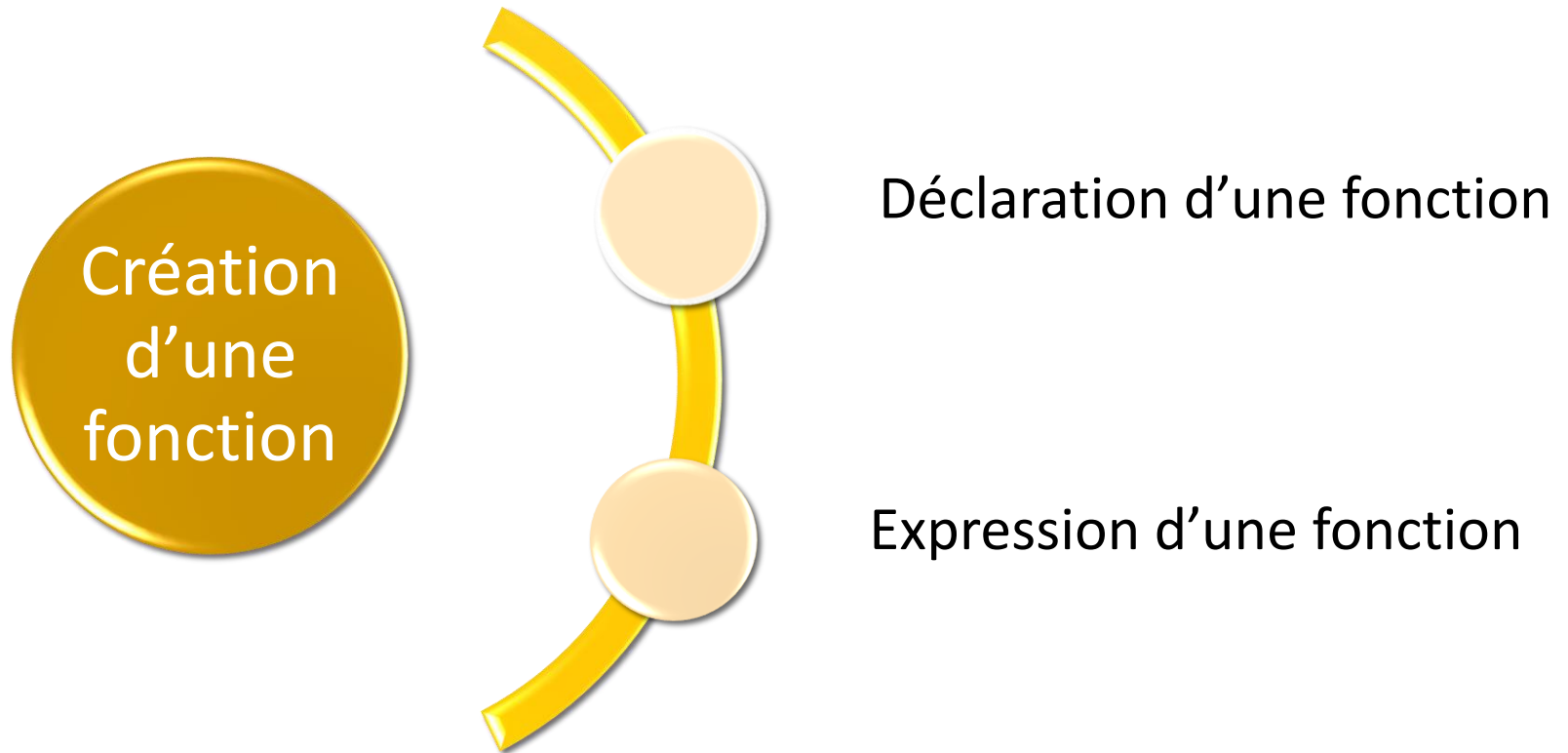
Afficher le premier nombre divisible par 17 compris entre 300 et 400

```
for (let i = 300; i <= 400; i++)  
{  
    if(i % 17 == 0)  
    {  
        alert(i);  
        break;  
    }  
}
```

Les fonctions

- Déclaration d'une fonction
- Expression d'une fonction
- Paramètres de fonctions et arguments
- Paramètres manquants/supplémentaires
- Passage de paramètres
- Tableau d'arguments
- Les fonctions fléchées
- Quelques fonctions prédéfinies

Création d'une fonction



Déclaration d'une fonction

Déclaration

```
function nomFonction ([liste des paramètres] )  
{  
    Bloc d'instructions  
    // return sauf si la fonction renvoie un résultat  
}
```

Appel Explicite

```
nomFonction ([liste des arguments] );
```

Exemples

Déclaration

```
function affichage()  
{ alert ("Ma 1ère fonction"); }
```

```
function carre(x)  
{ return x*x }
```

Appel

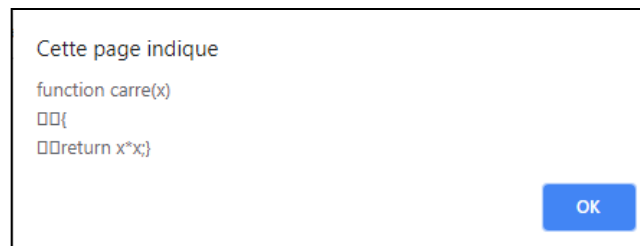
```
affichage();
```

```
let c= carre(6); // 1er appel  
alert(carre(7)); // 2ème appel
```

Remarque:

Si on ne met pas de (), c'est le code de la fonction qui est renvoyé


```
alert(carre);
```



Paramètres de fonctions et arguments


Paramètres vs Arguments

```
function somme(x,y)  
{ return x+y;}
```



paramètres

```
let c= somme(6, 12);
```



arguments

Paramètres manquants /supplémentaires

Paramètres	Peuvent être omis
	Ont la valeur undefined s'ils sont manquants
	Les arguments supplémentaires sont ignorés
	Les paramètres peuvent avoir des valeurs par défaut

Exemple

```
function maFonction(p1, p2, p3=2)
{ alert(p1 + " -- " + p2 + " -- " + p3); }
```

```
maFonction(10); // 10 -- undefined -- 2
```

```
maFonction(15, 12, 24, 58, 74); // 15 -- 12 -- 24
```

Passage de paramètres

- ✓ Le passage des paramètres se fait par **valeur**
- ✓ Lorsque l'argument est un **objet** ➡ passage par **référence**

Exemples:

```
function inc(x)
{
    x++;
}
```

```
let x = 10;
inc(x);
alert(x);    // 10
             // Passage par valeur
```

```
function incrementation(tab)
{
    tab[0]++;
}
```

```
let tab = [10, 20, 30];
incrementation(tab);
alert(tab[0]); // 11
              // Passage par référence
```

Tableau d'arguments

- ✓ Les arguments d'une fonction peuvent être récupérés dans la fonction dans un tableau nommé **arguments**
- ✓ La taille du tableau est définie dans la propriété **length**

Exemple:

```
function concat()  
{  
  let ch =arguments[0];  
  ch += " "+arguments[1];  
  ch += " "+arguments[2];  
  alert (ch + " taille = " + arguments.length);  
}  
concat("iset","ti", "tc", "info", "aa");    // iset ti tc taille = 5
```

Exercice

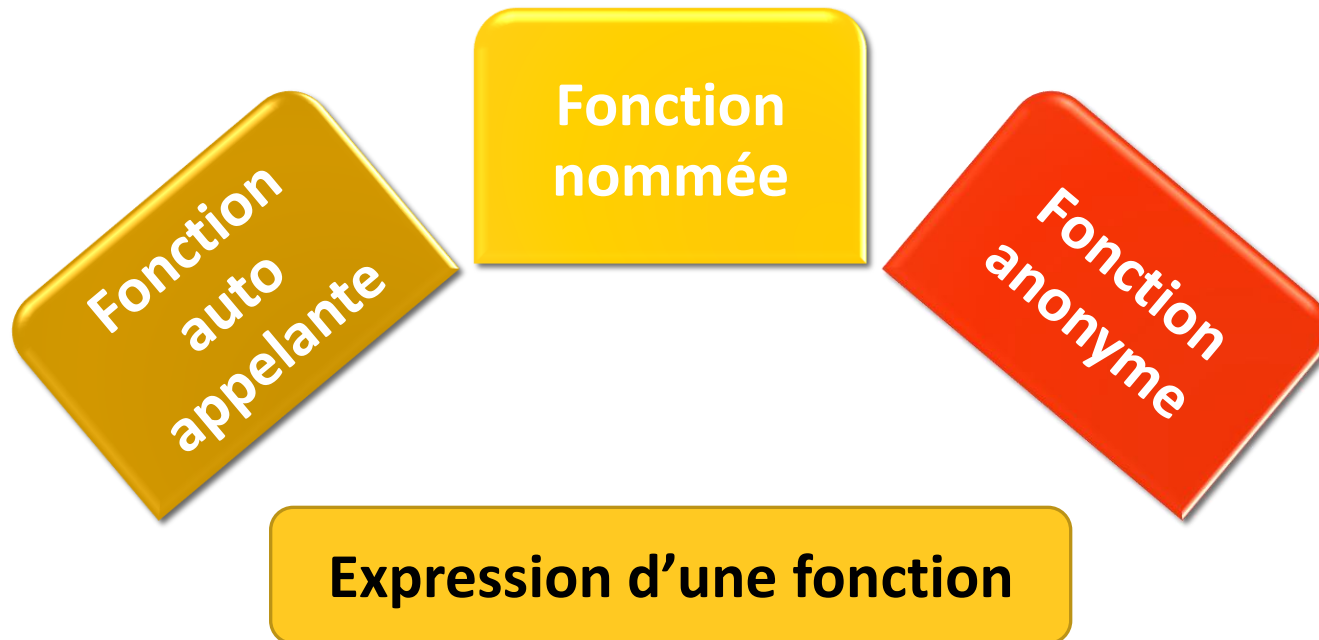
Qu'affiche ce programme?

```
function concatenation(x, y )  
{  
    let ch =x + " " +y;  
    for(let i=0; i< arguments.length; i++)  
        ch += " " +arguments[i];  
    alert (ch + " nombre d'éléments " + arguments.length);  
}
```

```
concatenation("1", "2", "3", "4", "5", "6");
```

// 1 2 1 2 3 4 5 6 nombre d'éléments 6

Expression d'une fonction



Fonction auto-appelante

Fonction

Avec ou sans nom

Appelée automatiquement

Syntaxe

```
( function([paramètres]) { /* code */ } ) ([arguments]);
```

Exemple

```
( function() { alert('Cours Web2');}) ();
```

Fonction nommée

Fonction **Avec** un nom

Déclarée dans une expression

Syntaxe

```
let myVar = function nomFonction([paramètres]) { /* code */ }
```

Exemple

```
let c= function carre(x) {return x*x;}  
alert(c(4));           // 16  
alert(typeof c);       // function
```

Fonction anonyme

Fonction **Sans** nom

Déclarée dans une expression

Syntaxe

```
let myVar = function ([paramètres]) { /* code */ }
```

Exemple

```
let msg= function() {return 'Cours Web2;');  
alert(msg());
```

Exemple (1/2)

Indiquez à chaque fois si l'expression est correcte ou non et donnez l'affichage correspondant

```
let f = function(x){return x+3; };
```



```
const g = function display{alert("TI101")};
```



```
const h = function calcul(x){alert(x-5)};
```



```
(function(a){alert(a*2)})(5);
```



//10

```
calcul(2);
```



```
h(10);
```



//5

```
alert(f(10));
```



//13


Exemple (2/2)

Préciser si c'est correct ou non et donner soit la correction soit le résultat


```
function calculer(){return arguments[1]+ arguments[2];}  
alert(calculer(10,"45",5));
```

 // 455


```
let f= function add(x){x++;}  
add(5);
```

 // f(5)


```
(function add(x,y){alert(x+y); }));
```



```
const fonc = function{alert("TI101")};  
fonc(25);
```



```
const fct = function(){alert(parseFloat("25.32ti101"))}  
fct();
```

 //25.32

```
(function(x){alert(typeof x)}})();
```

 //undefined

Exercice

Qu'affiche à chaque fois le programme?

```
let x=5;  
(function(){alert(x*x*x);})();
```

 // 125 (x est une variable globale)

```
let valeur= function(x){x--;}  
let a=10;  
valeur(a);  
alert(a);
```

 /* 10 (passage de paramètres par valeur) */

```
let car = function(y){return y*y;}  
function aff(f, x) {alert(f(x)*5);}  
aff(car,4);
```

 /* 80 (une fonction peut être passée en paramètre) */

Fonction fléchée

Fonction

Déclarée dans une expression souvent anonyme

Offre une syntaxe concise et optimale

Syntaxe

(param1,param2,...)=> { instructions}

(paramètres)=> expression

Paramètre => expression

Exemples

```
const calc= (x,y) =>{ x++; return x*y;}  
alert(calc(5,4));
```

```
const pdt= (x,y)=> x*y;  
alert(pdt(6, 4));
```

```
let aff = ch => 'Bonjour '+ch;  
alert(aff('TI'));
```

Ecritures d'une fonction fléchée

ES5

```
const total = function(x,y) {return x+y;}
```

ES6

```
const total = (x,y) => {return x+y;}
```



```
const total = (x,y) => x+y;
```

// Accolades inutiles
car 1 seule instruction

Remarques

- ✓ Les parenthèses `()` des paramètres ne sont **pas** obligatoires si on a 1 seul paramètre
- ✓ Si on renvoie 1 seule instruction, le mot clé **return** et les accolades `{ }` peuvent être supprimées
- ✓ On ne peut **pas** utiliser le tableau ***arguments*** dans une fonction fléchée comme c'est le cas pour les fonctions « classiques »

Exercice

Réécrire ces fonctions en utilisant les fonctions fléchées

```
const test = function(ch)
{return "Bonjour " + ch;}
```

```
const val = function(a)
{  if (a>0)
      a++;
  return a;}
```

```
(function() {alert("TI 101");})();
```

```
const test= ch =>"Bonjour " + ch;
```

```
const val = a => { if (a>0)
                    a++;
                  return a;
                }
```

```
((()=>alert("TI 101"))());
```

Quelques fonctions prédéfinies

- `eval`
- `isFinite` // convertit en entier et vérifie si le nombre est fini
- `isNaN`
- `parseFloat`
- `parseInt`
- `String`
- `Number`
- `Boolean`
- ...

Exercice

Qu'affiche chaque instruction?

alert(Number("145.34e2")); // 14534

alert(isFinite(10/0)); // false

alert(isFinite("TI")); // false

alert(isFinite("145e3")); // true

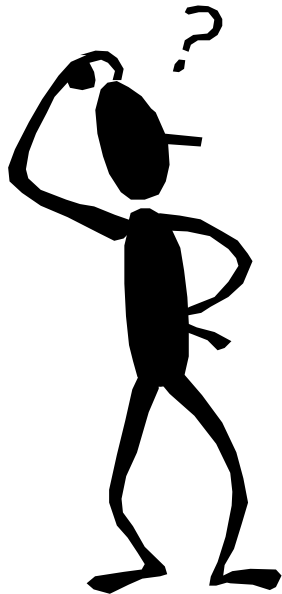
alert(isNaN(null)); // false

alert(isFinite(null)); // true

alert(isFinite(Infinity)); // false

let a = 10; alert (eval("5*a" + "+a")); // 60

Des questions?



Références

- <https://developer.mozilla.org/fr/docs/Web/JavaScript>
- <https://javascript.info/first-steps>
- <http://2ality.com/2013/04/quirk-implicit-conversion.html>
- <https://www.w3schools.com/js/default.asp>
- <https://www.pierre-giraud.com/javascript-apprendre-coder-cours/presentation-fonction/>
- <https://www.xul.fr/ecmascript/fonctions-predefinies.php>