Chapitre 2 :

Concepts de base de Javascript

Ce chapitre introduit les concepts de base de javascript en détaillant comment intégrer du javascript dans une page web, sa syntaxe de base, les principaux types de données et leur conversion, les opérateurs, les structures conditionnelles, les structures itératives et un aperçu de la déclaration des fonctions.

I. Intégration de Javascript dans une page web

Le code javascript peut être intégré soit dans la page html soit dans un fichier javascript externe.

I.1. Intégration de javascript dans la page html

Le script peut être intégré :

- Au niveau de l'entête de la page (head) à l'aide de la balise **<script>**

Exemple :

```
<head>
    <script>
        alert("Code déclaré dans l'entête");
    </script>
</head>
```

- Dans le corps de la page (body) à l'aide de la balise **<script>**

Exemple :

```
<body>
    <script>
        alert("Code déclaré dans le corps de la page");
    </script>
</body>
```

- Dans une URL au niveau de la balise href selon la syntaxe suivante

Syntaxe :

```
<a href="javascript:code javascript">Message</a>
```

Exemple :

```
<a href="javascript:alert('ISET Charguia')">Cliquer ici</a>
```

- Comme réponse à un événement

Syntaxe :

```
<balise …. onEvenement= "code javascript" />
```

Exemple :

```
<button onclick="alert('clic sur un bouton')"> Test </button>
```

I.2. Ecriture de javascript dans un fichier externe

Le script javascript peut être écrit dans un fichier externe, pour cela, il faut :

1. Ecrire le code javascript dans un fichier ayant pour extension **.js**
2. Lier le fichier .js à la page html au niveau de l'entête en précisant son chemin dans l'attribut **src** de la balise script.

Exemple :

Soit le fichier exemple.html qui fait appel à un script placé dans un fichier exemple.js

Code du fichier « Exemple.js »

```
alert('ISET Charguia') ;
```

Extrait du fichier « Exemple.html »

```
<head>
    <script src="exemple.js"></script>
</head>
```

II. Syntaxe de Javascript

La syntaxe de javascript est très proche de celle du langage C. Elle présente les spécificités suivantes :

- Javascript est sensible à la casse
- Dans le cas d'un bloc d'instructions, les instructions sont encadrées par des accolades **{ }** et séparées par un **;**
- Les commentaires s'écrivent :
  - Su une seule ligne //
  - Sur plusieurs lignes /* **….** */
- La déclaration des variables se fait avec les mots clés : **var**, **let** et **const**
- Javascript offre une panoplie de mots réservés dont il est possible d'en mentionner :

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

II.1. Les entrées/sorties en javascript

Il existe divers moyens, autres que les formulaires, pour récupérer ou renvoyer des données. Ils peuvent être classés comme suit :

II.1.1. Les entrées

Pour pouvoir introduire des données, il est possible d'utiliser les boîtes de dialogue suivantes :

- **prompt()**

Boîte de dialogue qui permet d'obtenir une entrée utilisateur récupérée en tant que chaîne de caractères. prompt accepte un à 2 arguments.

Syntaxe :

```
resultat = prompt("message", ["message d'invite facultatif "] );
```

Exemple :

```
let nombre = prompt("Donnez un nombre",
"Saisie d'un nombre");
```

- **confirm()**

Boîte de dialogue qui permet de formuler une question avec 2 boutons « OK » et « Annuler ».
La valeur récupérée est **true** si c'est le bouton « OK » qui est pressé, **false** sinon.

Syntaxe :

```
resultat = confirm ("question");
```

Exemple :

```
let result = confirm ("Voulez-vous continuer?");
```

II.1.2. Les sorties

L'affichage des données peut se faire via les boîtes de dialogue suivantes :

- **alert()**

Boîte de dialogue qui affiche un message

Syntaxe :

```
alert ("message");
```

Exemple :

```
alert(" Ceci est un test");
```

- **console.log()**

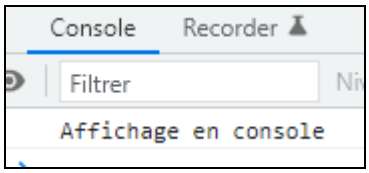
Cette fonction permet d'afficher dans la fenêtre console (accessible en tapant F12)

Syntaxe :

```
console.log("message");
```

Exemple :

```
console.log(" Affichage en console")
```



- **document.write()**

Cette fonction permet d'écrire l'argument passé dans la page html.

Syntaxe :

```
document.write ("message");
```

Exemple :

```
document.write(" Affichage dans la fenêtre");
```

- **print ()**

Cette fonction permet d'ouvrir une boîte de dialogue qui donne la possibilité d'imprimer le contenu actuel du document html

Syntaxe :

```
print();
```

II.2. Déclaration des variables

Une variable dans javascript:

- a un nom qui commence par une **lettre, _ ou $** suivie d'une chaîne quelconque de lettres, chiffres, _ et/ou $

Exemples :

Les noms de variable suivants sont corrects:
$surface, _abs, x1, _rayon$, ___longueur1, Ab$2_8Kj$_,…

- est sensible à la classe
- peut être déclarée (facultatif) avec **var** ou **let** qui ont des portées différentes :
  - **var** a une portée *globale* et peut être redéfinie
  - **let** a une portée *limitée au bloc* où elle est déclarée, elle ne peut pas être redéfinie.

Exemple :

```
<script>
    let x = 3;
    var y = 6;
    z = 1;
    if(x<y){
        let x = 10;
        z ++;
        var t = 2;
        u= 11;
        console.log("x= "+x+ " y= "+y + " z= "+z);     //x= 10 y= 6 z= 2
    }
    else{
        var y = 2;
        console.log("x= "+x+ " y= "+y + " z= "+z);
    }
    console.log("x= "+x+ " y= "+y + " z= "+z)          // x= 3 y= 6 z= 2
    console.log(" t = "+t+ " u= "+ u);                 // t = 2 u= 11
</script>
```

- Une variable déclarée, mais non initialisée a pour valeur ***undefined***

Remarque 1:

Javascript offre le mécanisme de **hoisting/hissage** qui déplace la déclaration d'une variable en haut de sa portée rendant ainsi la variable accessible avant sa déclaration.

Exemple :

On considère une variable x non déclarée et on examine 2 cas de figure

```
console.log(x); // Uncaught ReferenceError:        ⇨   console.log(x); // undefined
                // x is not defined                    var x =5;
```

Ainsi :

```
console.log(x);      Est transformé   ⇨   var x ;
var x =5;                                 console.log(x);
                                          x =5;
```

Remarque 2:

Il est possible d'utiliser le mode strict qui interdit l'utilisation de variables non déclarées en ajoutant la directive `"use strict";` ou `'use strict';`

Exemple :

```
<script>
        'use strict';
        x = 12;
        console.log(x); // Uncaught ReferenceError: x is not defined
</script>
```

II.3. Les constantes

Une constante est déclarée avec le mot clé **const**

Syntaxe :

```
const NOM_CONSTANTE = valeur;
```

Par convention, une constante est notée en majuscule.

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 TAUX = 19;
            console.log(TAUX);    // 19
        }
        console.log(TAUX);    // Uncaught ReferenceError : TAUX is not defined
```

III. Principaux types de données et conversion entre les types

Une variable a le type de la donnée qui lui est assignée.

III.1. Principaux types de données

Les principaux types de données sont :

Type	Description
Number	Type numérique: entier (entre -(2^53 − 1) et 2^53 – 1), nombre positif, négatif, scientifique, décimal (entre -2^1074 et 2^1024),…
String	Chaîne de caractères
Boolean	true / false
Undefined	Type d'une variable non assignée (aucune valeur ne lui a été affectée)
Null	Type qui précise explicitement que la valeur est nulle
Object	Collection non ordonnée de paires clé/valeur
Function	Type fonction

Il demeure qu'il existe d'autres types tels que *BigInt* ou *Symbol* qui sont moins fréquemment utilisés.

III.1.1. L'opérateur typeof

Pour identifier le type d'une variable, il est possible d'utiliser l'opérateur **typeof**

Syntaxe :

typeof opérande **typeof** (opérande)

Exemples :

```
let a = 10;
console.log( typeof a );      // number
console.log(typeof 12.5);     // number
console.log(typeof (1/0));    // number
var x;
console.log(typeof x);        // undefined
console.log(typeof 'test');    // string
console.log( typeof(0 =="0")); // boolean
console.log(typeof {nom:'ISET', adresse: "Charguia" })  // object
function f(){ alert('Appel fonction');  }
console.log(typeof f);                          // function
console.log( typeof null);     // object
```

III.1.2. Le type Number

Le type Number accepte :

- Toutes les valeurs numériques : décimales ou non décimales
- Infinity, -Infinity
- NaN (Not a Number)
- Les valeurs hexadécimales : **0x**17, **0x**FE14, **0x**2A
- Les valeurs octales : **0o**627, **0o**25, **0o**431
- Les valeurs binaires : **0b**10, **0b**101, **0b**100001

Exemples :

```
alert (15/0);  //Infinity
alert (0x1A);  // 26
alert (00125);  // 85
```

III.1.3. Le type String

Une chaîne de caractères doit être écrite entre des guillemets (`""`) ou des apostrophes (`''`) comme il est possible d'utiliser les gabarits de chaînes.

La concaténation de 2 chaînes se fait à l'aide de l'opérateur « + »

Exemples :

```
console.log('ISET'+ " Charguia");      // ISET Charguia
console.log('ISET "Charguia"');        // ISET "Charguia"
console.log('ISET '"Charguia");        // Erreur de compilation
console.log('ISET \'Charguia');        // ISET 'Charguia
```

- ***Les gabarits de chaînes***

Les gabarits de chaînes ont été introduits avec ES6, ils permettent d'utiliser les accents graves (``` `` ```) afin de :

- définir des chaînes de caractères *multilignes*

Exemple :

```
console.log(` Voici une
        phrase sur
        3 lignes`);
```

⇨

```
Voici une
       phrase sur
       3 lignes
```

- Interpoler / intégrer des expressions

Syntaxe :

```
`...${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}`);
```

III.2. Conversion entre les types

Pour convertir d'un type à un autre, il existe des conversions implicites et d'autres explicites nécessitant l'utilisation d'une fonction prédéfinie.

III.2.1. Les conversions implicites

Il est possible de convertir certains types vers un autre type implicitement dans les cas suivants :

Conversion implicite	Cas	Exemple
Number → String	Cas d'une chaîne concaténée à un nombre	`alert("conversion"+1+2+3);` `// conversion123`
String → Number	Cas d'une opération de multiplication ou de division	`alert('opération'+'2'*'4');` `// opération8`
Null → 0	Cas d'une opération arithmétique	`alert(null+2+"-"+ null*3);` `// 2-0`

0, undefined, NaN →false	Les valeurs qui sont intuitivement vides sont converties en false	`alert(Boolean(null)); //false`
		`alert(Boolean(0)); //false`
		`var x; alert(Boolean(x)); //false`
		`alert(Boolean(NaN)); //false`

Autres valeurs → true	Les valeurs qui sont intuitivement non vides sont converties en true	`alert(Boolean(10.2)); //true`
		`alert(Boolean("a")); //true`

III.2.2. Les conversions explicites

Les conversions explicites par le biais d'un ensemble de fonctions prédéfinies décrites comme suit :

Fonction prédéfinie	Description	Exemple

parseFloat(string)	Transforme une chaîne en un nombre flottant après son analyse	`alert(parseFloat('87.3')); //87.3`
		`alert(parseFloat('1.2.5m')); //1.2`
		`alert(parseFloat('p5')); //NaN`

parseInt(string)	Transforme une chaîne en un entier après son analyse	`alert(parseInt('12.65')); // 12`
		`alert(parseInt('4dfj')); //4`
		`alert(parseInt('2.6.5')); //2`

Number(valeur)	Convertit en nombre, si le résultat obtenu n'est pas un nombre valide, la fonction renvoie NaN	`alert(Number('12')); // 12`
		`alert(Number('12.65')); // 12.65`
		`alert(Number('3gg')); //NaN`
		`alert(Number('4.68.5')); //NaN`

String(valeur), **Boolean**(valeur), …	Permet respectivement de convertir une expression en chaîne ou booléen	`alert (String (12.3)); // 12.3` `alert(Boolean (0)); //false`

isNaN(valeur)	Détermine si une expression est de type NaN ou non (ce n'est pas une fonction de conversion)	`alert(isNaN('12')); //false`
		`alert(isNaN(parseFloat('11.35t')));`
		`//false`
		`alert(isNaN(Number('11.35f')));`
		`//true`

IV. Les opérateurs

Javascript offre une panoplie d'opérateurs dont il est possible de citer principalement :

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

- **Les opérateurs == vs === et != vs !==**

Javascript distingue l'égalité == de l'égalité stricte === de sorte que :

- == convertit le type implicitement et renvoie si 2 expressions ont la même valeur
- ==== n'effectue aucune conversion et renvoie true si 2 expressions ont la même valeur et le même type de données

Le même principe s'applique à l'inégalité != et l'inégalité stricte !== où :

- != renvoie true si 2 expressions ont des valeurs différentes
- !== renvoie true si 2 expressions ont des valeurs différentes ou des types différents

Exemples :

```
<script>
        console.log("15" == 15);          // true
        console.log(12.35==="12.35");     // false
        console.log(1 == true);           // true
        console.log(1 === true);          // false
        console.log(0 != undefined);      // false
        console.log(null !== undefined); // true
</script>
```

V. Les structures conditionnelles

Les structures conditionnelles en javascript sont similaires à celles définies en langage C.

V.1. La structure « if »

Les différentes constructions de la structure « if » peuvent être résumées comme suit :

Structure de choix simple if	Structure à choix alternatif if … else	Structure à choix alternatif if… else …if

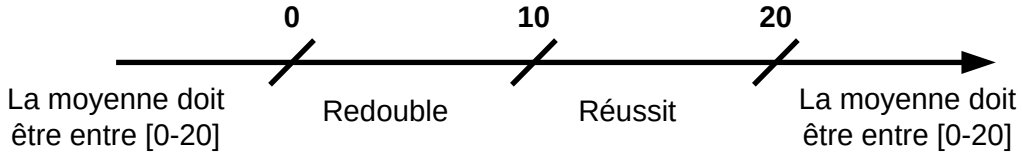
`if (condition) {`
`/* Bloc d'instructions*/`
`}`
	`if (condition) {`
	`/* Bloc d'instructions*/`
	`}`
	`else {`
	`/* Bloc d'instructions*/`
	`}`
		`if (condition) {`
		`/* Bloc d'instructions*/`
		`}`
		`else if(condition) {`
		`/* Bloc d'instructions*/`
		`}`
		`...`
		`else {`
		`/* Bloc d'instructions*/`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

A noter que la condition est une expression **booléenne**.

Exemple :

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 »

```
<script>
  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`);
</script>
```

V.2. La structure ternaire

La structure ternaire est similaire à celle du langage C.

Syntaxe :

```
(condition) ? ...bloc si condition vérifiée : ...bloc sinon
```

Exemple :

Afficher dans une boîte de dialogue si une chaîne saisie est vide ou non

```
let chaine = prompt("Donnez une chaine");
(chaine ===""){? alert('Chaîne vide'): alert('Chaîne non vide');
```

V.3. La structure switch

La structure à choix multiple « switch » a la syntaxe suivante.

Syntaxe :

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

VI. Les structures itératives

Javascript offre différentes structures répétitives : Les structures itératives for, while et do.. while qu'on retrouve dans la majorité des langages de programmation tel que le langage C ainsi que 2 autres structures for…of et for…in.

VI.1. Les boucles while, do…while et for

Les boucles while, do … while er for présentent une syntaxe analogue à celle du langage C comme le résume ce tableau

Boucle	Syntaxe

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

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

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

A noter que la condition représente une expression **booléenne**.

VI.2. La boucle for…of

La boucle **for…of** a été introduite avec ES6, elle permet de parcourir des objets itérables comme les tableaux.

Syntaxe :

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

Exemple :

```
let tabVilles = ['Tunis', 'Sfax', 'Nabeul', 'Sousse'];
for(let v of tabVilles)
  console.log(v);  // Affiche Tunis, puis Sfax puis Nabeul puis Sousse
```

VI.3. La boucle for…in

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

Syntaxe :

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

Exemples :

```
let tabVilles = ['Tunis', 'Sfax', 'Nabeul', 'Sousse'];
for(let indice in tabVilles)
  console.log(indice);  // Affiche 0, 1, 2, 3
```

```
let personne= {nom:'Gharbi', prenom:'Ali', age:25, profession:'Ingénieur'}
for(let prop in personne)
  console.log(prop);  // Affiche nom, prenom ; age, profession
```

VI.4. L'instruction break

L'instruction **break**, outre son utilisation dans la structure *switch*, permet de quitter l'itération en cours et sortir de la boucle.

Exemple :

Ecrire un script permettant d'afficher dans une boîte de dialogue le premier nombre compris entre 100 et 200 divisible par 9

```
for (let i = 100; i <= 200; i++) {
  if (i % 9 == 0) {
    alert(i);
    break;
  }
}
```

VII. Les fonctions

Les fonctions qui sont des sous-programmes, peuvent être créées en javascript de 2 façons :

- Déclaration d'une fonction avec le mot clé **function**
- Définition d'une expression de fonction

VII.1. Les fonctions déclarées avec le mot clé « function »

VII.1.1. Déclaration et appel d'une fonction

Une fonction est déclarée avec le mot clé **function** comme suit.

Syntaxe :

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

A noter qu'une fonction peut avoir 0 ou plusieurs paramètres.

L'appel d'une fonction se fait comme suit :

Syntaxe :

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

Exemples :

```
function cube (x){
  alert(x*x*x);
}
```

⇨

```
cube(4);
```

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

⇨

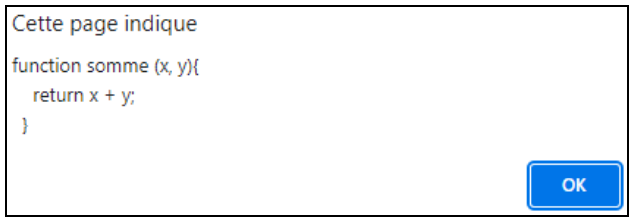
```
let s = somme(10, 13);
// Si on veut afficher la somme de 5 et 20
alert(somme(5, 20));
```

Remarque :

Attention, si lors de l'appel, les parenthèses sont oubliées, c'est le code de la fonction qui s'affiche.

Exemple :

```
alert(somme);
```



VII.1.2. Les paramètres et les arguments

Les **paramètres** sont les variables définies dans la signature de la fonction alors que les **arguments** désignent les valeurs effectivement passées lors de l'appel d'une fonction.

Exemple :

```
function somme (x, y){       // x et y sont les paramètres de la fonction
  return x + y;
}
let s = somme(10, 13);       // 10 et 13 sont les arguments passés
```

VII.1.3. Paramètres manquants & arguments supplémentaires

Dans une fonction, les paramètres :

- Peuvent être omis
- Ont la valeur *undefined* s'ils sont manquants
- Peuvent avoir des valeurs par défaut

Notons que les arguments supplémentaires sont **ignorés**.

Exemple :

```
function test (p1, p2, p3=5){
  alert(`${p1} -- ${p2} -- ${p3}`);
}
test();         // undefined -- undefined -- 5
test(1);        // 1 -- undefined -- 5
test(1,2);      // 1 -- 2 -- 5
test(1,2,3);    // 1 -- 2 -- 3
test(1,2,3,4); // 1 -- 2 -- 3
```

VII.1.4. Passage de paramètres

Le passage des paramètres se fait par valeur, lorsque le paramètre est un objet, il est passé par référence.

Exemple :

Passage par valeur	Passage par référence

`function incrementation(a) {`
`  a++;`
`}`
`let x = 10;`
`incrementation(x);`
`alert(x); // 10`
	`function increment(tab) {`
	`  tab[0]++;`
	`}`
	`let tab = [50, 70, 90];`
	`increment(tab);`
	`alert(tab[0]);  // 51`

VII.1.5. Tableau d'arguments

Les arguments d'une fonction peuvent être récupérés dans la fonction à travers un tableau nommé ***arguments*** dont la taille peut être récupérée à travers la propriété ***length***.

Exemple :

```
function test(){
    for(let i= 0; i < arguments.length; i++)
        alert(arguments[i]);
    }
    test(101, "ti", 106, "Charguia"); // Affiche : 101 puis ti puis 106
                                      // puis Charguia
```

VII.2. Les expressions de fonctions

Une expression de fonction est une fonction stockée dans une variable.

Le moteur Javascript traite les déclarations de fonctions (vues précédemment) et les expressions de fonctions différemment. En effet :

- Pour la déclaration de fonction : elle est totalement chargée dans la mémoire du navigateur même si la fonction n'est pas appelée
- Pour une expression de fonction : elle n'est chargée que si elle est appelée

Une expression de fonction peut prendre les formes suivantes:

- ❖ Fonction auto-appelante
- ❖ Fonction nommée
- ❖ Fonction anonyme

Remarque :

Il est possible de créer une fonction à l'aide de l'objet **Function**, mais ce ne sera pas abordé au niveau du cours.

VII.2.1. Fonction auto-appelante

Une fonction auto-appelante :

- Est définie et appelée immédiatement /automatiquement
- Peut avoir ou non un nom

Syntaxe :

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

Exemples :

```
(function(){alert('test')})();   // Affiche test
(function produit(x,y){alert(x*y)})(17,2) ; // Affiche 34
let val = (function(x,y){return(x*y)})(10,2) ; // val contient la valeur 20
```

VII.2.2. Fonction nommée

Une fonction nommée :

- Possède un nom
- Peut être déclarée dans une expression

Syntaxe :

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

Exemples :

```
let a = function afficher(x){alert(x+5)};
a(4); // 9
let c = function calcul(x){return x * 3};
alert(c(5));  // 15
```

VII.2.3. Fonction anonyme

Une fonction anonyme :

- N'a pas de nom
- Peut être déclarée dans une expression

Syntaxe :

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

Exemples :

```
let message = function(){ return 'Fonction anonyme'};
alert(message());
let somme = function(x,y){return x+y};
alert(somme(10,2));
```

VII.2.4. Les fonctions fléchées

Une fonction fléchée est une notation plus concise d'une expression de fonction (généralement anonyme). Elle ne possède pas ses propres valeurs pour *this* et *arguments*.

Syntaxe :

```
([param1, ..., parami])=> {instructions}
```

Sachant que :

- Les parenthèses **()** des paramètres ne sont **pas** obligatoires si on a 1 seul paramètre
- Les accolades **{}** et le mot clé **return** peuvent être omis si la fonction renvoie 1 seule instruction

Exemples :

```
let somme = (x,y)=>x+y ;
alert(somme(10,15)); //25

let affichage= ()=>alert('Fonction fléchée');
affichage(); // Fonction fléchée

let parite = elt => { if (elt % 2 == 0)  alert('pair');
                      else alert('impair') }
parite(17); //impair
```

VII.3. Les fonctions prédéfinies

Javascript offre une multitude de fonctions prédéfinies dont certaines ont déjà été présentées auparavant. Voici d'autres fonctions.

Fonction	Rôle	Exemple

isNaN()	Détermine si une valeur est NaN (Not a Number)	`isNaN("10.2.3.6"); // true`
		`isNaN("143.56"); // false`
		`isNaN(undefined); // true`

isFinite()	Convertit une valeur en un nombre et renvoie s'il est fini ou non	`isFinite(10/0); // false`
		`isFinite("10.3"); // true`
		`isFinite(undefined);// false`
		`isFinite("Bonjour"); // false`

Number(), parseInt(), parseFloat(), …	Fonctions permettant la conversion de type telles que définies précédemment (dans III.2)	`Number("135.4t"); // NaN`
		`parseInt("135.4t"); // 135`
		`parseFloat("135.4t"); // 135.4`

eval(string)	Evalue du code js présenté sous forme de valeur	`let x = 3;`
		`alert(eval("(x+2)"+"*(x-1)"));`
		`//10`

void(fonction)	Evalue une expression et renvoie *undefined*. Elle permet d'annuler la valeur de retour d'une expression	`function f(){`
		`  return 10;`
		`}`
		`alert(void f()); //undefined`