LYCÉE ARISTIDE BERGÈS
SIO

Découvrir

110 - Découvrir JavaScript

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision
2023-11-29	Ajout de la partie gestion des événements.

Sommaire

A Introduction....1
A.1 Présentation....1
A.2 Prérequis....1
B Les instructions....2
B.1 Mode REPL....2
B.2 Première instruction....3
B.3 D'autres instructions....3
B.4 Instructions : à vous de jouer….4
C les variables simples....5
C.1 Règles de nommage....5
C.2 Variables : à vous de jouer….6
D Les variables complexes....7
D.1 Les tableaux....7
D.2 Les objets....8
D.3 Variables complexes : à vous de jouer....8
E Les commentaires....9
E.1 Utilité des commentaires....9
E.2 Syntaxe des commentaires....9
E.3 Commentaires : à vous de jouer….9
F Les opérations sur les variables....10
F.1 Opérations numériques....10
F.2 Opérations de chaînes....11
F.2.1 Chaîne....11
F.3 Opérations de variables : à vous de jouer….12
G Les fonctions....14
G.1 Fonction simple....14
G.2 Fonction avec paramètre·s....15
G.3 Fonctions : à vous de jouer….16
H Les conditions....17
H.1 Conditions générales (if… else)....17
H.2 Expressions conditionnelles (vide, inférieur, supérieur, égal)....18
H.3 Expressions Booléennes (et, ou, non et ou exclusif)....19
I les itérations (répétitions)....20
I.1 Itération numérique 'faire n fois' (boucle 'for')....20
I.2 Itération conditionnelle 'faire … tant que' (do… while)....22
I.3 Itération conditionnelle 'tant que' (do… while)....22
I.4 Condition et itération : à vous de jouer….23
J les tableaux....25
J.1 Ajout de données dans un tableau (push(), unshift() et splice())....25
J.2 Suppression de données dans un tableau (pop())....25
J.3 Rechercher une valeur dans un tableau (indexOf())....25
J.4 Code d'exemple des fonctions sur les tableaux....25
J.5 Tableaux : à vous de jouer....26
K les interactions avec la page HTML....28
K.1 récupérer position d'un objet DOM....28

K.2 Modifier un objet DOM à la position.....29
K.3 Créer un objet DOM à une position.....29
K.4 Créer un événement.....30
K.5 DOM et modification : à vous de jouer….....31

Nomenclature à supprimer :

- **Assimilation** : cours pur. Explication théorique et détaillée (globalement supérieur à 4 pages).
- **Décoder** : fiche de cours, généralement inférieure à 5 pages.
- **Découverte** : Travaux dirigés. Faisable sans matériel ou bien application encadrée (simple).
- **Explorer** : Travaux pratiques. Nécessite du matériel ou des logiciels.
- **Mission** : Projet encadré ou partie d'un projet.
- **Projet** : Projet en autonomie totale. Environnement ouvert.

A Introduction

La programmation en JavaScript nécessite uniquement un navigateur web et un éditeur de texte. Pourtant, JavaScript est l'un des langages les plus puissants et permet également de faire fonctionner des serveurs (Node.JS).

C'est donc un langage incontournable en BTS SIO.

A.1 Présentation

Pour être rapidement opérationnel en JavaScript, il faut suivre l'ensemble des fiches de cette découverte : la plupart des thèmes importants dans la programmation seront abordés :

- les instructions en direct
- les variables simples
- les opérations sur les variables
- les environnements de développement
- les fonctions
- les itérations (répétitions)
- les conditions
- les tableaux
- les interactions avec la page HTML

A.2 Prérequis

Aucun prérequis n'est nécessaire en termes de connaissance : une personne n'ayant jamais programmé peut comprendre les exemples à suivre.

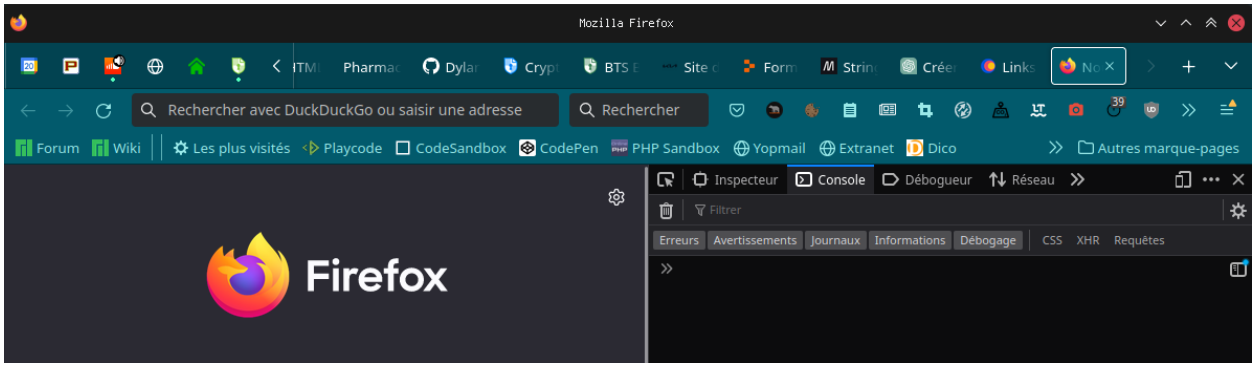
En revanche, il est recommandé de tester réellement les exemples fournis et même de tenter de les modifier (question en fin de fiches).

Ce document PDF vous permet de rédiger des réponses et des commentaires dans des zones appropriées : il est préférable de sauvegarder ce fichier sur votre clé USB après chaque écriture.

B Les instructions

Pour découvrir JavaScript, nous allons simplement ouvrir un nouvel onglet dans le navigateur de votre choix (FireFox, Chrome, Safari ou autre).

Nous allons accéder à l'outil le plus important pour les développeurs : la console de débogage. Pour cela, il suffit d'appuyer simultanément sur les touches [Ctrl], [Shift] et [i]. Si ce n'est pas déjà le cas, sélectionnez l'onglet "Console".



Cette opération sera fréquemment réalisée et même nécessaire pour tout développement : mémorisez donc ce raccourci-clavier.

B.1 Mode REPL

La console de débogage permet d'exécuter des instructions en direct. Cela signifie que le résultat s'affiche immédiatement. REPL (prononcer "rippeul") signifie "Read – Evaluate – Print – Loop".

Dans ce mode, le moteur JavaScript attend une instruction, la lit, l'évalue, affiche un résultat et attend à nouveau. Il affiche donc deux chevrons `>>` pour vous dire où écrire.

Essayez ceci : utilisez le mode REPL comme une calculatrice simple (appuyez sur la touche [Entrée] pour valider le calcul).

```
>> 2+2
← 4
```

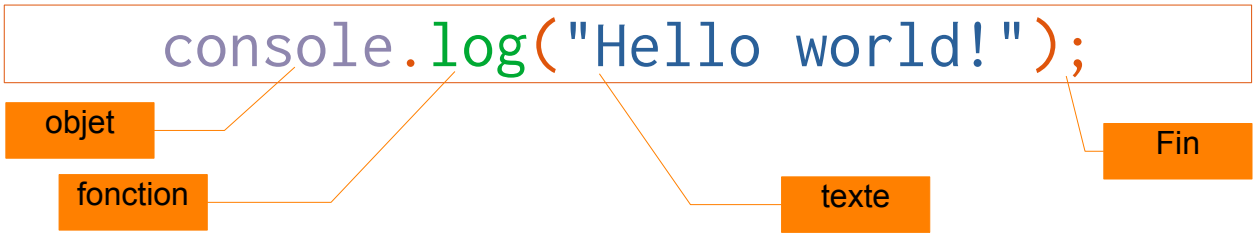
Comme vous pouvez le constater, JavaScript lit et évalue vos instructions.

Essayez ceci : nous allons demander à JavaScript d'écrire un texte dans la console. Un texte doit obligatoirement être délimité par des guillemets `"` ou de simple côte `'` ou encore des anti-côtes `` ` ``.

```
>> console.log("Hello world!");
  Hello world!
← undefined
```

B.2 Première instruction

La première instruction que vous venez d'écrire est un ensemble surprenant de mots anglais et de ponctuations diverses. Voici leur fonctionnement :



le mot clé `console` indique que l'action s'appliquera sur la console : c'est un **objet**.

Le mot clé `log` avec des parenthèses signifie que c'est une **fonction**. À l'intérieur des parenthèses, on transmettra des **paramètres** à la fonction. Ici, `log()` reçoit une chaîne de texte `"Hello world!"`. Comme cette fonction écrit les différents contenus dans la parenthèse, elle affichera donc "`Hello world!`" dans la console.

Le point-virgule `;` à la fin, indique que l'instruction est terminée. En JavaScript, la plupart des interpréteurs JavaScript n'en ont pas besoin et savent détecter la fin d'une instruction. Dans d'autres langages, le symbole de fin est obligatoire (Java, C#, PHP).

B.3 D'autres instructions

Il n'y a pas beaucoup d'instructions très visibles en mode REPL, cependant en voici une qui aura un impact immédiat sur la page web :

```
>> document.write("Hello world!")
⚠ This page is in Quirks Mode....
← undefined
```

Le message en jaune dans la partie console est un avertissement : il n'y a pas eu d'erreur, mais le moteur JavaScript constate qu'il y a un risque de dysfonctionnement et qu'il faut donc vérifier l'instruction.

La page web doit normalement être une page blanche contenant la phrase

Hello world!

On peut décortiquer l'instruction comme pour la précédente :

`document.write("Hello world!")` demande à la fonction `write()` d'écrire `"Hello world!"` dans l'objet `document`.

B.4 Instructions : à vous de jouer…

Testez les instructions suivantes et donnez le résultat dans ce document.

```
console.log(8 + 5 * 10 -2 * 3-2)
```

Résultat

```
console.log(8+5*10-2*3-2)
```

Résultat

```
console.log("Bonjour","les","spaSIOnautes")
```

Résultat

```
console.log("9*8=",9*8)
```

Résultat

```
document.write("<br>","Bonjour<br>", "c'est <em>en HTML</em> !")
```

Résultat

```
document.bgColor = "chartreuse";
```

Résultat

Note : dans ce dernier cas, `bgColor` n'est pas une fonction mais une **propriété** de l'objet document.

```
window.location = "http://david.roumanet.free.fr/BTS-SIO"
```

Résultat

C les variables simples

L'intérêt majeur d'un programme est de pouvoir mémoriser des informations pour pouvoir les utiliser plus tard.

L'enregistrement de données, puis le traitement et l'affichage de ces données nécessite des zones de mémoire dans l'ordinateur. JavaScript permet de donner un nom à chaque zone réservée

Pour réserver de la mémoire, JavaScript utilise les instructions `let`, `const` et `var` suivi par le nom que l'on veut donner : ces zones de mémoire nommées s'appellent des **variables**.

Les variables peuvent contenir des nombres ou des textes (mais aussi des tableaux et en JavaScript, des objets et des fonctions).

Essayez ceci :

```
>> const nomProfesseur = "ROUMANET"
← undefined
```

Bien que l'interpréteur n'ait rien affiché, il a bien placé la chaîne de caractères (texte) "`ROUMANET`" dans la variable appelée `nomProfesseur`. Voici comment le vérifier :

```
>> console.log(nomProfesseur)
  ROUMANET
← undefined
```

C.1 Règles de nommage

Les variables peuvent prendre presque n'importe quel nom, à condition de respecter les règles suivantes :

- ne pas commencer par un chiffre
- ne pas contenir d'espace
- ne pas contenir de symboles (!:;.%$=+)

De plus, il existe des conventions de nommage qui précisent d'autres règles (par exemple, ne pas mettre d'accent dans les noms de variables).

Noms autorisés	Noms interdits
Nom nom unNom prixSP98 nom_etudiant nom_étudiant_BTS_SIO	12lettres titre+nom Un nom nom_étudiant_BTS-SIO

Enfin, JavaScript fait la différence entre les majuscules et minuscules : `nom` est différent de `Nom`.

C.2 Variables : à vous de jouer…

Utilisez les variables suivantes et découvrez les possibilités ou erreurs.

```
let nom_Etudiant="NORRIS"
console.log(nom_Etudiant)
```

Résultat

```
let nom_Etudiant="NORRIS"
```

Résultat

Note : ce message est normal, car vous avez déjà défini la variable `nom_Etudiant` juste avant. On ne doit définir une variable qu'une seule fois.

```
nom_Etudiant="Chuck NORRIS"
console.log(nom_Etudiant)
```

Résultat

Note : ici, nous ne redéfinissons pas le nom de la variable, mais son contenu, ce qui est normal.

```
let heures=14
let minutes=50
console.log(heures,":",minutes)
```

Résultat

```
console.log(Heures,":",minutes)
```

Résultat

Note : ce message est normal car `Heure` est différent de `heure`.

D Les variables complexes

Elles ne sont pas complexes à cause d'une difficulté, mais plutôt dans la manière de les stocker en mémoire.

D.1 Les tableaux

Ce type de variable contient plusieurs valeurs, pour un seul nom de variable. Comment stocker les températures mensuelles de Stockholm ?

Dans un tableau :

-1.8	-1.9	1	5.8	10.8	15.2	18.2	17.2	12.5	7	2.9	-0.2

Pour info, ce sont les véritables températures moyennes entre 1991 et 2020.

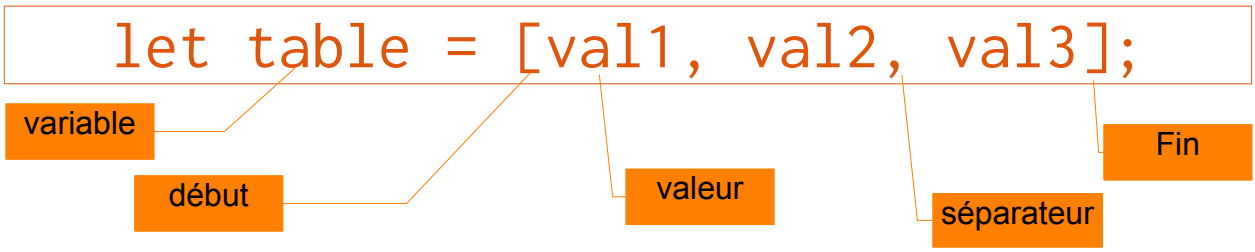
Il serait possible de placer chaque valeur dans une variable différente, mais ce serait difficile à utiliser :

```
let Stockholm_jan = -1.8
let Stockholm_fev = -1.9
// etc
```

JavaScript nous permet de stocker toutes les données dans un tableau :

```
let Stockholm_Temp = [-1.8, -1.9, 1, 5.8, 10.8, 15,2, 18.2, 17.2, 12.5, 7, 2.9, -0.2]
```

La syntaxe est la suivante :



Pour atteindre tout le tableau, il est possible d'afficher simplement la variable :

```
console.log(Stockholm_Temp)
```

Mais la grande force des tableaux, c'est de pouvoir accéder à chaque valeur individuellement, avec l'indice de la colonne du tableau : la première valeur a la place 0, la deuxième est à la place 1, etc.

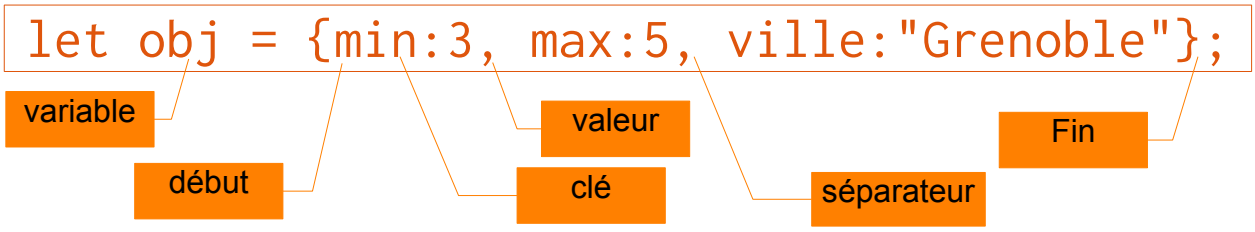
Donc la 12e case porte l'indice 11 (toujours -1).

Pour afficher la valeur du mois de juin (6e mois), on utilise `console.log(Stockholm_Temp[5])`. Pour changer la valeur, c'est la même chose, `Stockholm_Temp[5] = 15.2` !

D.2 Les objets

En JavaScript, tout est considéré comme un objet (pour ceux qui arrêteront le développement, vous aurez déjà vu cette notion). La description d'un objet utilise un format appelé **JSON** (JavaScript Object Notation).

La syntaxe est la suivante :



Pour accéder aux données de l'objet, il faut utiliser le nom de l'objet suivi du symbole `.` puis la clé choisie. Dans l'exemple ci-dessus, pour afficher la ville contenue par l'objet, il suffit de rédiger un code similaire à :

```
let obj = { min:3, max:5, ville:"Aix" } ;
console.log("Ville : ", obj.ville)
```

Un objet contient plusieurs valeurs, mais à la différence du tableau dans lequel les valeurs ont des indices, dans l'objet on nomme les valeurs.

```
let ville = { min:0, max:0, ville:"" }       // création d'un objet vide
ville.ville = Grenoble        // l'ordre des clé n'a pas d'importance puisqu'elle sont nommées
ville.min = -7
ville.max = +38
```

JavaScript acceptera l'affichage de l'objet complet (pour contrôler les valeurs) mais cette forme n'est pas exploitable dans un programme.

```
console.log(ville)
```

D.3 Variables complexes : à vous de jouer...

Créez un tableau `Amis` qui contient la liste de vos trois meilleurs amis et affichez uniquement le premier ami dans la liste du tableau.

Résultat

Créez un objet `moi`, qui contient votre `nom`, `taille` et `age`.

Résultat

E Les commentaires

E.1 Utilité des commentaires

Rédiger du code nécessite de décrire à l'ordinateur ce qu'il doit faire, mais aussi aux autres codeurs, ce que fait le programme.

Les commentaires sont donc essentiels dans un programme.

Note : parfois, on utilise les commentaires pour rendre inactif ("supprimer") du code : c'est souvent une source d'erreur et cela ne doit être que temporaire. Par exemple, pendant un débogage.

E.2 Syntaxe des commentaires

Pour écrire un commentaire, JavaScript utilise deux syntaxes :

- le double symbole `//` permet d'arrêter l'analyse de JavaScript jusqu'à la fin de la ligne
- le symbole `/*` permet de débuter une zone de commentaire (donc non lu par JavaScript) jusqu'à la rencontre du symbole `*/` qui ferme la zone.

La plupart des programmeurs utilisent `/*` et `*/` en entête de leur programme, pour préciser ce que fait le programme, la date de création, l'auteur et quelques renseignements simples.

Ils utilisent `//` à la suite d'une instruction pour décrire son action, lorsque cela peut paraître complexe ou illogique ou encore une exception.

Nous utiliserons dans cette introduction, les commentaires en fin de ligne, pour vous aider à comprendre les instructions, mais vous n'avez pas besoin de copier le commentaire.

E.3 Commentaires : à vous de jouer…

Tester l'emplacement des commentaires

```
console.log("Bonjour")                                    // Affichera bonjour dans la console
```

Résultat

```
console.log("Bonjour")   /* fait la même chose que précédemment */
```

Résultat

```
// console.log("Bonjour")  // n'affichera pas bonjour dans la console
```

Résultat

F Les opérations sur les variables

Les variables sont par définition, variables : leur contenu peut changer dans le temps et être modifié par les programmes.

Comme il est possible d'avoir plusieurs programmes différents qui fonctionnent en même temps, il faut s'assurer que les noms des variables soient bien choisis.

Une variable `i` est probablement le pire exemple, car c'est un nom très utilisé pour les comptages, par les programmeurs. Dans les exemples ci-dessous, nous faisons une exception pour gagner en rapidité d'écriture.

F.1 Opérations numériques

JavaScript permet d'utiliser le symbole +, -, *, / pour additionner, soustraire, multiplier ou diviser des nombres.

```
let a = 15     // n'oubliez pas, l'utilisation de variable à un caractère est dangereux
let b = 20
let c = 5
let d = 2
console.log(5 * (a + b - c) / d)
```

Dans ce programme, JavaScript commence par calculer a+b-c puis multiplie le résultat par 5 et enfin le divise par d. Le résultat est 75.

Les opérations sont utilisables dans une variable également :

```
let e = 5 * (a + b - c) / d
```

Pour les opérations plus complexes, il faut utiliser une bibliothèque appelée Math, qui contient de nombreuses opérations, comme les sinus, cosinus, puissance, log et d'autres fonctions.

```
let angle = Math.cos(3.1415926536)   // calcul le cosinus(PI)
let autreAngle = Math.sin(Math.PI)   // calcul le sinus de la constante PI
let autreAngle = Math.round(3.14159*100)/100   // Arrondir facilement un nombre à deux décimales
```

Note : nous avons déjà expliqué qu'une lettre, un mot ou une phrase doit être encadré par des guillemets ou de simples quotes, pour être considéré comme un texte. Cependant, Javascript essayera toujours de faciliter la vie du programmeur en effectuant des conversions automatiques. Le code suivant fonctionnera, alors que l'on multiplie une valeur numérique (le chiffre 2) avec le texte "21".

```
let vie = "21"
let reponseUniverselle = 2 * vie
console.log(reponseUniverselle)
```

F.2 Opérations de chaînes

Les programmeurs ont eu l'idée de traiter les phrases comme des blocs que l'on peut diviser en caractères ou bien concaténer ou encore séparer en plusieurs morceaux

F.2.1 Chaîne

Pour JavaScript, une chaîne est un tableau de caractères ASCII ou UTF-8.

indice	0	1	2	3	4	5	6
caractère	B	o	n	j	o	u	r
Code ASCII	66	111	110	106	111	117	114

On peut donc facilement retrouver une lettre ou modifier une lettre, en utilisant le nom de la variable et l'indice de la lettre recherchée : par exemple, lire la lettre à l'indice [3] de la variable s'écrit :

```
let mot = "Bonjour"
console.log(mot[3])
```

On peut ainsi créer un autre mot et concaténer des mots et des caractères :

```
let ours = "Mon ami" + " " + mot[2] + "ou" + mot[2] + mot[4] + mot[5] + mot[6] + "s"
```

Ce code renverra :

```
Mon ami nounours
```

La concaténation est l'action d'accoler des mots et des lettres les uns à la suite des autres.

Attention : il n'est pas possible de changer le contenu de la chaîne en essayant de modifier un caractère vie mot[3] = "c" par exemple.

Note : c'est l'occasion de tromper le moteur JavaScript sur nos besoins. En ajoutant `"91"` et `1`. Dans la console REPL, tapez le code suivant et regardez ce qu'il se passe.

```
>> "91"+1
← "911"
```

On n'obtient pas 92 mais "911" car JavaScript remarque une chaîne de caractères (texte) et un opérateur de concaténation : cette fois, il convertit automatiquement le chiffre 1 en texte "1" et fait l'ajout du texte à la fin du premier texte.

Rassurez-vous, le `+` est la seule opération ambiguë. D'ailleurs, pour convertir un nombre dans un texte, il est fréquent de le multiplier par 1 avant.

```
>> "91"*1+1
← 92
```

Pour la plupart des opérations sur les chaînes de caractères, il existe donc des fonctions toutes faites, dont voici les principales :

```
let phrase = "Bonjour, les spaSIOnautes du lycée"
```

petite précision : le bon terme est spationaute ; un spaSIOnaute étant un étudiant SIO en immersion dans mes cours 😇

fonction	action	Résultat sur 'phrase'
phrase.length	Donne le nombre de caractères	34
phrase.toLowerCase() phrase.toUpperCase()	Change tous les caractères en minuscules (change la casse) Change tous les caractères en majuscules (change la casse)	bonjour, les spasionautes du lycée BONJOUR, LES SPASIONAUTES DU LYCÉE
phrase.slice(13,25)	Récupère les caractères entre la position 13 et 25	spaSIOnautes
phrase.replace("Bonjour","Bonsoir") phrase.replaceAll(" ","-")	Remplace une partie de la chaîne par une autre chaîne Remplace tous les espaces par des tirets	Bonsoir, les spaSIOnautes du lycée Bonsoir,-les-spaSIOnautes-du-lycée
phrase.indexOf("du")	Indique la position de la chaîne recherchée à l'intérieur de la chaîne. Renvoie -1 si la position n'est pas trouvée.	26
phrase.split(" ")	Divise la chaîne en tableau de chaîne, en utilisant le séparateur " "	["Bonjour,", "les", "spaSIOnautes", "du", "lycée"]

F.3 Opérations de variables : à vous de jouer…

Comment afficher la longueur de la phrase ?

```
let unePhrase = "Ce serait si facile, de ne pas avoir à réfléchir..."
console.log(unPhrase.length)
```

Résultat

Comment afficher le dernier caractère de la phrase ?

```
let unePhrase = "Ce serait si facile, de ne pas avoir à réfléchir..."
console.log(unePhrase[???])
```

Résultat

Comment remplacer le mot "réfléchir" par le mot "penser" ?

```
let unePhrase = "Ce serait si facile, de ne pas avoir à réfléchir..."
console.log(unePhrase.???("réfléchir", "penser"))
```

Résultat

Comment trouver le mot "facile" dans la phrase ?

```
let unePhrase = "Ce serait si facile, de ne pas avoir à réfléchir..."
console.log(unePhrase.???("facile"))
```

Résultat

Comment mettre en majuscule le nom de l'utilisateur ?

```
let prenom = "Chuck"
let nom = "Norris"
console.log(prenom+" "+nom+" a déjà compté jusqu'à l'infini : deux fois")
```

Résultat

Comment inverser un mot quelconque de 5 lettres ?

```
let mot = "GANTS"     // on doit utiliser les [] pour ce challenge
console.log("Le mot inversé est "+???)
```

Résultat

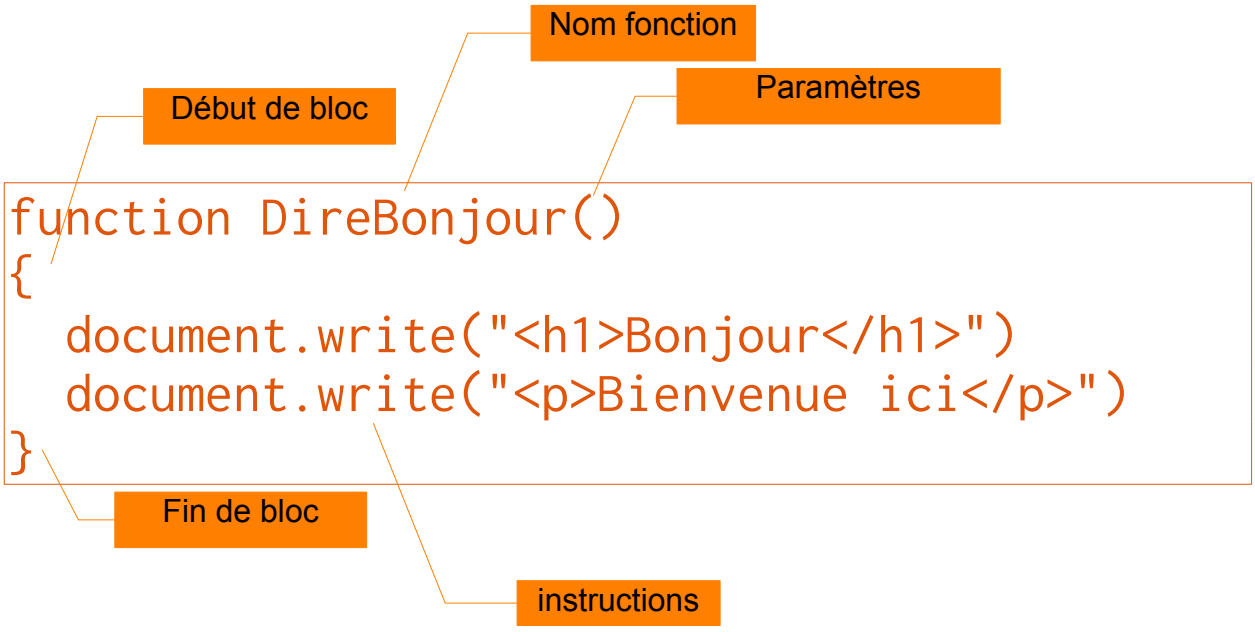
G Les fonctions

JavaScript permet de créer nos propres fonctions. C'est intéressant pour pouvoir utiliser plusieurs fois le même ensemble d'instructions. On peut comparer une fonction à une macro (si vous connaissez le terme) ou a une liste d'instruction cachée dans un bloc.

G.1 Fonction simple

C'est une fonction qui – de manière générale – ne gère que des choses statiques, comme afficher un message ou effacer et initialiser des données.

La syntaxe est la suivante :



Une fonction commence donc par le mot réservé function puis le nom qui résume ce que fait la fonction.

On ouvre ensuite des parenthèses qui pourront recevoir des choses (mais pas dans cette fonction, donc on laisse vide).

On ouvre ensuite un bloc de code avec l'accolade { et on ferme le bloc avec l'accolade }

On ne place qu'une instruction par ligne

Pour utiliser cette fonction, il suffit d'utiliser son nom (et les parenthèses).

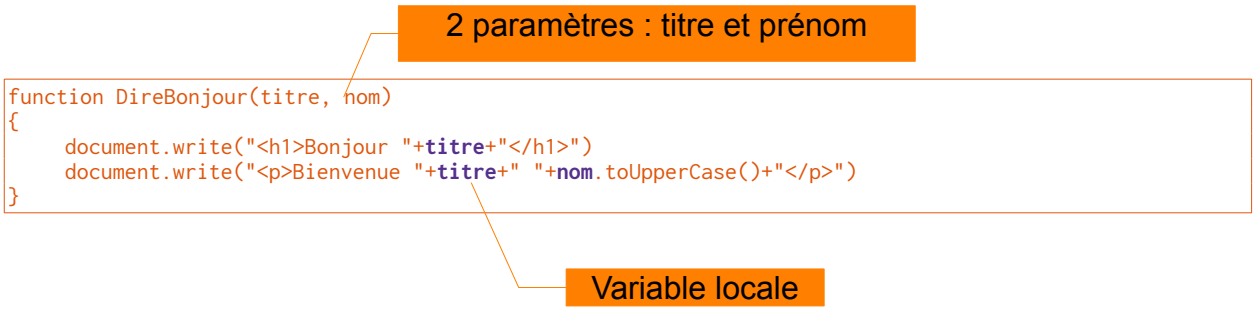
```
DireBonjour()
direBonjour()
direBonjour()
```

Ici, la fonction sera appelée 3 fois.

G.2 Fonction avec paramètre·s

C'est une fonction qui intègre des informations, transmises sous forme de paramètres.

La syntaxe est la suivante :



Il est possible de transmettre autant de paramètres que souhaité (et même zéro, tel que la précédente fonction) du moment qu'ils soient séparés par des virgules.

Important : à l'intérieur de la fonction, il faudra toujours utiliser le nom du paramètre (la variable locale).

Pour utiliser cette fonction, il suffit d'utiliser son nom (et les parenthèses).

```
let unNom = "Laprof"
DireBonjour("monsieur", "Norris")
direBonjour("professeur", "Einstein")
// exemple transmettant le contenu de la variable unNom dans la variable locale nom de la fontion
direBonjour("madame", unNom)
```

Fonction avec retour

La plupart du temps, l'usage d'une fonction ne doit pas afficher un résultat, mais effectuer un traitement. Par exemple, la fonction CalculerTVA(prixHT, taux) n'affichera pas la TVA sur un prix hors-taxes mais fournira le résultat du calcul. Pour cela, il faut utiliser le mot clé `return` suivi de la valeur à retourner.

```
function CalculerTVA(prixHT, taux)
{
    return Math.round(prixHT * taux) / 100
}

// test avec un vêtement à 100€ HT et un taux de 20% (doit renvoyer 20€ de TVA)
let prixVetement = 100
let tauxTVAMagasin = 20  // en réalité 20% devrait s'écrire 0.2 (soit 20 / 100)
let TVA = CalculerTVA(prixVetement, tauxTVAMagasin)
let prixVetementTTC = prixVetement + TVA
document.write("Votre article coute " + prixVetementTTC + " €")
```

On peut également appeler la fonction indépendamment :

```
CalculerTVA(66.6666, 20)
```

Le résultat est bien arrondi à deux chiffres après la virgule.

G.3 Fonctions : à vous de jouer…

Vérifiez et corrigez les propositions suivantes :

Que fait cette fonction ?

```
function Calculer(anNaissance) {
  let maintenant = new Date()
  let age = maintenant.getFullYear() - anNaissance
  return age
}
```

Résultat

Donnez un exemple d'utilisation pour afficher le résultat de la fonction précédente, dans la console JS.

Donnez un exemple d'utilisation pour afficher le résultat de la fonction précédente, dans la page web.

Sachant que la formule de conversion d'une température en fahrenheit vers Celsuis est la suivante :

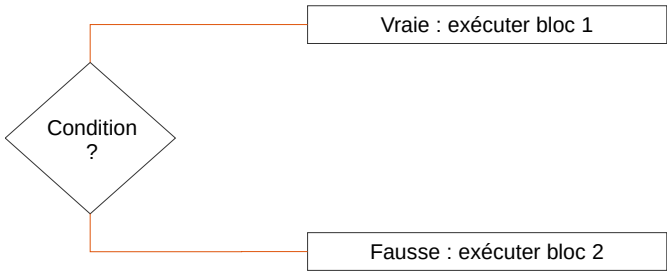
degreCelsius = (degreFahrenheit - 32) / 1.8000

Créez une fonction ConversionFversC à laquelle on transmet une température en °F et qui renvoie une température en °C

Enfin, quelle correction apporter à ce code pour obtenir une température arrondie à un chiffre après la virgule (exemple : 28,3541 devient 28,3)

H Les conditions

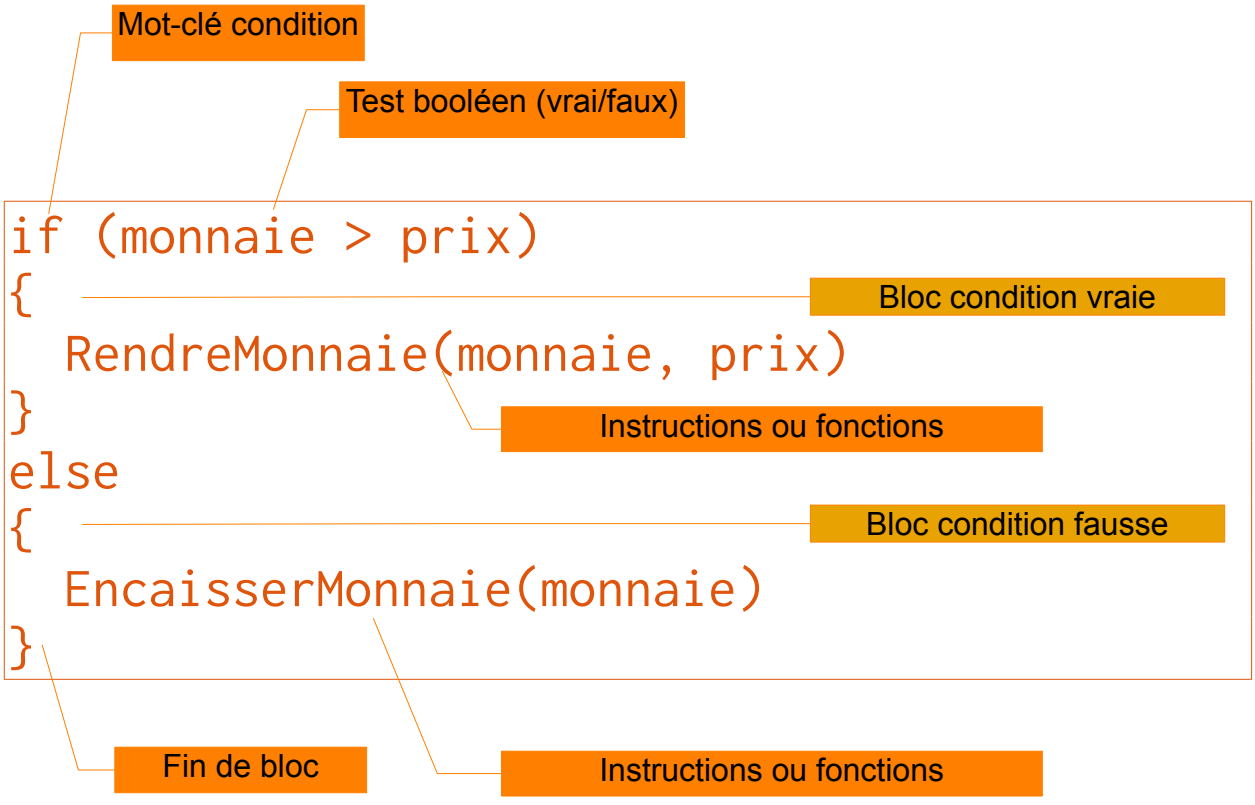
Un ordinateur peut prendre des décisions, de la même manière que les humains, en utilisant une logique de comparaison. Il ne s'agit pas d'intelligence, mais plutôt de logique binaire : les choix sont le résultat d'une assertion (affirmation) testée vraie ou fausse. Il n'y a donc que deux chemins possibles.



H.1 Conditions générales (if… else)

La notion de condition s'écrit comme en français, avec l'équivalent du 'si' en anglais : `if` !

Par exemple, si le client donne trop d'argent alors, il faut rendre la monnaie, sinon, encaisser sa monnaie.



On note qu'il y a deux blocs, puisqu'il y a deux choix possibles : il arrive cependant que le dernier choix n'ait aucune action, il pourra alors ne pas être écrit.

Voici un exemple de code contenant une suite de conditions :

```
let note = 5   // testez le code en modifiant la valeur de la note

if (note < 8) {
  console.log("Note très insuffisante")
}
else if (note < 11) {    // else if effectue un autre test avant d'utiliser le choix par défaut
  console.log("Note faible")
}
else if (note < 15) {
  console.log("Note moyenne mais encourageante")
}
else {    // choix par défaut (lorsque tous les autres tests ont échoué
  console.log("Note très correct ou excellente")
}
```

Plutôt que d'écrire des `if` sans liens, il utilise une structure "sinon si (condition)" qui permet de n'écrire que le commentaire voulu.

Note : le mot clé `else` sans `if` indique que c'est le bloc qui sera exécuté si aucun autre test n'a été satisfait.

H.2 Expressions conditionnelles (vide, inférieur, supérieur, égal)

Les expressions conditionnelles doivent donc renvoyer un résultat binaire. Voici quelques exemples simples :

Tester si une chaîne de caractère est vide	`if (chaine == "") {…}`
Tester si une chaîne de caractère n'est pas vide	`if (chaine != "") {…}`
Tester si un nombre est égal à un autre	`if (nombre1 == nombre2) {…}`
Tester si un nombre n'est pas égal à un autre	`if (nombre1 != nombre2) {…}`
Tester si un nombre est strictement inférieur à un autre	`if (nombre1 < nombre2) {…}`
Tester si un nombre est inférieur ou égal à un autre	`if (nombre1 <= nombre2) {…}`
Tester si un nombre est strictement supérieur à un autre	`if (nombre1 > nombre2) {…}`
Tester si un nombre est supérieur ou égal à un autre	`if (nombre1 >= nombre2) {…}`

Attention : Il existe une raison simple pour laquelle on teste une égalité avec un double égal `==` : avec un seul signe égal, JavaScript considérerait qu'il s'agit d'une affectation (nombre1 = nombre2), ce qui renvoie un résultat faux si nombre1 devient zéro et un résultat vrai dans tous les autres cas.

H.3 Expressions Booléennes (et, ou, non et ou exclusif)

L'algèbre Booléenne a été créée par Georges BOOLE autour du XIX^{e} siècle et s'appuie sur la logique binaire ;

Si nous avons 2 bits (a et b), nous pouvons les combiner avec les opérations ET (AND), OU (OR), OU Exclusif (XOR) ; l'opération NON (complément) ne nécessite qu'un seul bit.

ET

a	b	R
0	0	0
0	1	0
1	0	0
1	1	1

OU

a	b	R
0	0	0
0	1	1
1	0	1
1	1	1

OU EXC.

a	b	R
0	0	0
0	1	1
1	0	1
1	1	0

NON

a	R
0	1
0	1
1	0
1	0

Les opérations booléennes renvoient un résultat binaire (vrai ou faux) ce qui permet de les utiliser dans les conditions.

Vous utilisez inconsciemment cette algèbre, par exemple, "si (il fait chaud ET que j'ai de l'argent) alors (j'irais à la piscine)" : vous n'irez à la piscine seulement si vous avez de l'argent pour acheter un billet et qu'il fait chaud.

Note : la syntaxe dépend du langage, nous traiterons ici le langage JavaScript.

a ET b	a OU b	a OU Exclusif b	NON a
`a && b`	`a || b`	`a ^ b`	`~a`

Pour représenter notre test piscine en JavaScript, nous écrirons :

```
let chaud = 1
let argent = 0

if (chaud == true && argent == true)    // condition utilisant une opération booléenne
{
  let piscine = true
} else
{
  let piscine = false
}
console.log(piscine)
```

I les itérations (répétitions)

Les langages de programmation servent principalement à deux choses :

- réaliser des opérations complexes sans erreurs (un être humain pouvant se tromper)
- réaliser des opérations répétitives sans erreurs (un être humain pouvant se lasser)

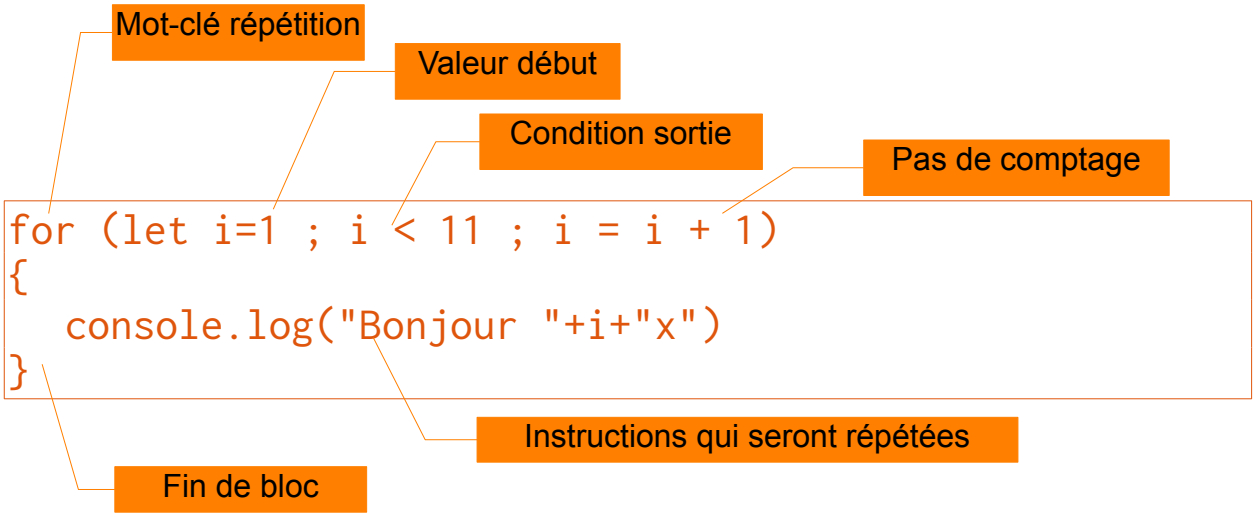
L'utilisation de fonctions répond à la première phrase, et les itérations répondent à la seconde.

Ainsi, tous les langages de programmation proposent des instructions pour répéter des instructions, de manière définie (exemple, "5 fois" ou "de 12 à 24" ou encore "dans l'intervalle de 7 à 19") ou terminer sur une condition réalisée (exemple "jusqu'à ce que le résultat soit plus égal à 1").

I.1 Itération numérique 'faire n fois' (boucle 'for')

Javascript propose une syntaxe un peu longue, mais permettant de choisir le début, la fin et le pas de comptage. Les paramètres sont séparés par des `;` à l'intérieur des parenthèses.

La syntaxe est la suivante :



Si vous essayez ce code dans la console, vous verrez apparaître

```
Bonjour 1x
Bonjour 2x
Bonjour 3x
Bonjour 4x
Bonjour 5x
Bonjour 6x
Bonjour 7x
Bonjour 8x
Bonjour 9x
Bonjour 10x
```

Le langage va commencer son comptage à partir de 1 et ne pas dépasser 11 (strictement inférieur à 11, c'est-à-dire, 10). Il comptera de 1 en 1, car il incrémente la variable i de 1 à chaque passage dans la boucle.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 1 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 2)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	I = 2 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 3)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 3 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 4)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 4 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 5)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 5 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 6)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 6 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 7)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 7 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 8)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 8 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 9)
	`i` est inférieure à 11, on relance.

`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 9 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 10)
	`i` est inférieure à 11, on relance.

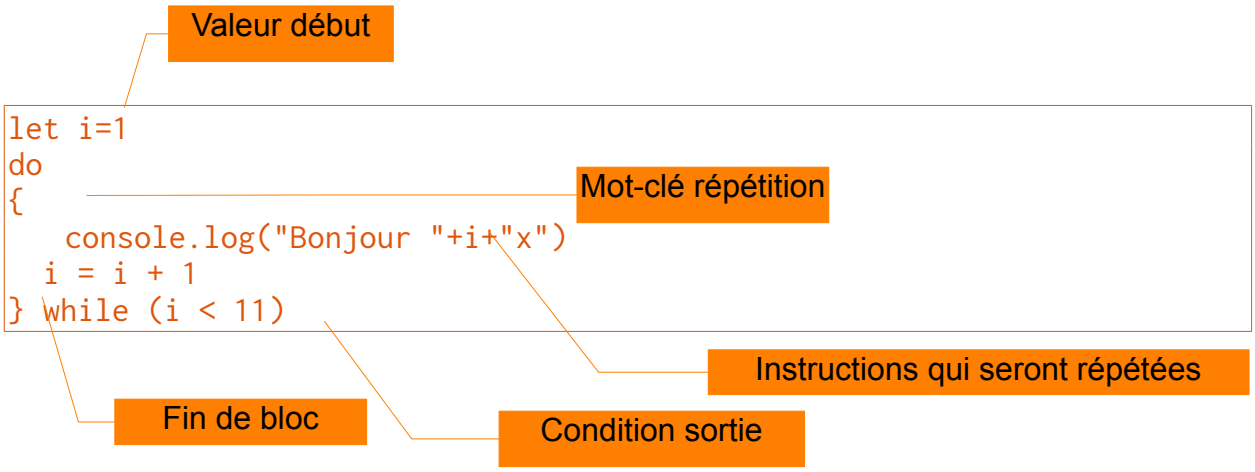
`for (let i=1 ; i < 11 ; i = i + 1)`
`{`
`  console.log("Bonjour "+i+"x")`
`}`
	i = 10 : Afficher "Bonjour " et la variable `i` et "x"
	on incrémente `i`. (la variable `i` passe à 11)
	`i` est égale à 11 (condition fausse), on s'arrête.

Ce qu'il faut retenir, c'est que la boucle for incrémente la variable après la dernière instruction et teste le résultat juste après. Nous allons voir la même boucle avec un compteur manuel, dans la prochaine section.

I.2 Itération conditionnelle 'faire … tant que' (do… while)

Cette itération est la version moins automatisée de l'itération `for`.

La syntaxe est la suivante :

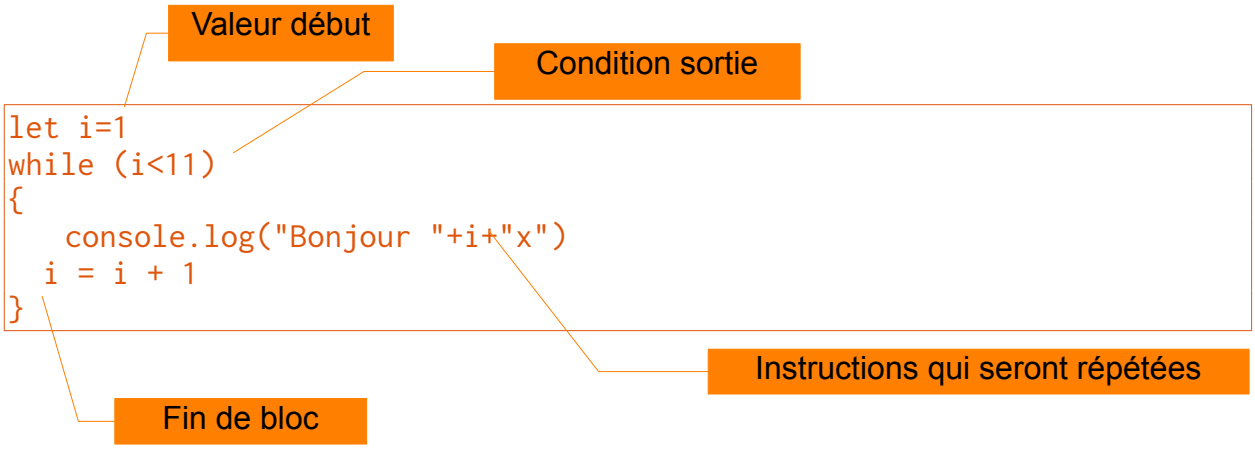


Ici, il faut d'abord déclarer une variable de comptage et l'initialiser, ensuite il faut mettre l'instruction d'addition à la fin de la boucle et enfin, tester le résultat.

Note : on parcourt cette boucle au moins une fois.

I.3 Itération conditionnelle 'tant que' (do… while)

Dans cette itération, on teste la condition avant d'exécuter le code contenu dans l'itération, pour éviter de lire un fichier vide, par exemple. Voici sa syntaxe :



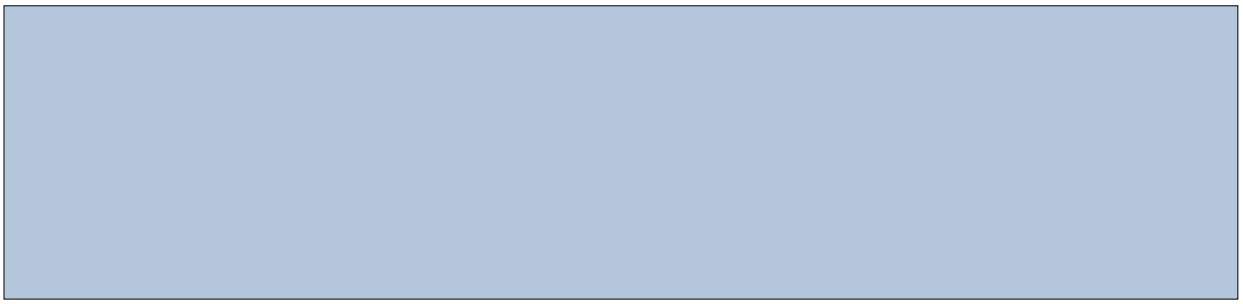
Ici, il faut d'abord déclarer une variable de comptage et l'initialiser, ensuite il faut mettre l'instruction d'addition à la fin de la boucle et enfin, tester le résultat.

Note : on parcourt cette boucle au moins une fois.

I.4 Condition et itération : à vous de jouer…

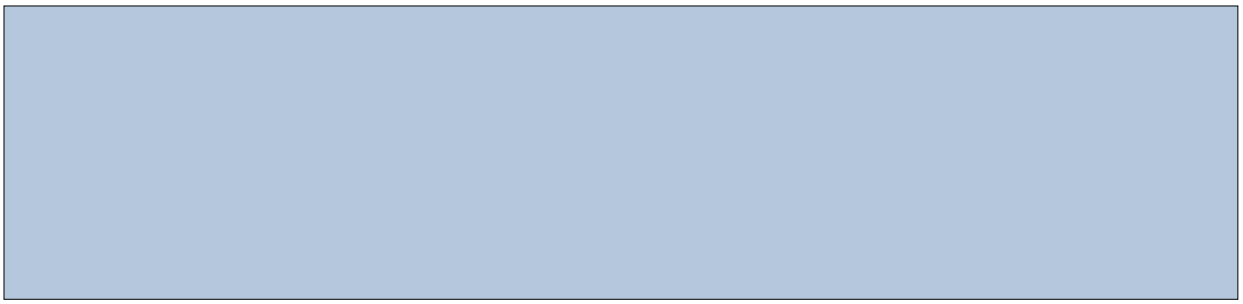
Rédigez une condition qui affiche un message s'il faut rendre de la monnaie, suite à un achat :

```
PrixAchat = 70
MonnaieAchateur = 100
```



Affichez dans la console, la table de multiplication de 5, en commençant à 1x5 et terminant à 10x5 :

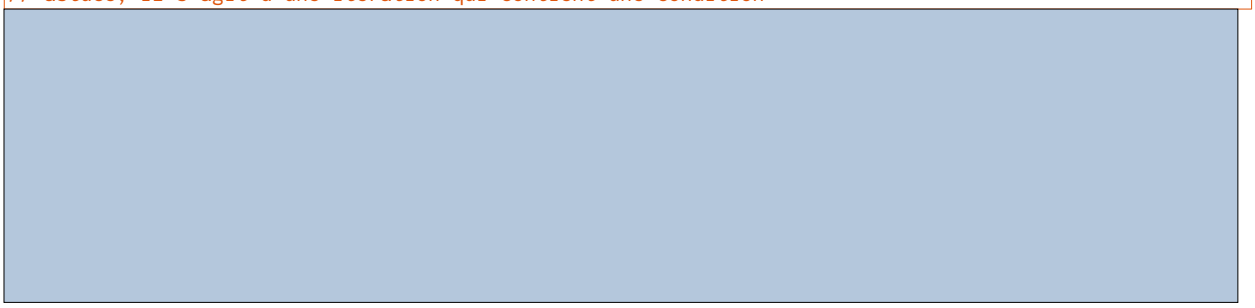
```
let multiplicateur = 5
let debut = 1
let fin = 10
```



Affichez une liste de nombre, en indiquant à côté de chaque nombre, s'il est pair (divisible par 2) ou impair.

Exemple : 1 est impair, 4 est pair, 53 est impair…

```
let liste = [5, 18, 36, 39, 42, 42, 42, 63, 100]
// astuce, il s'agit d'une itération qui contient une condition
```



J les tableaux

Il existe quelques fonctions permettant de faciliter l'utilisation des tableaux.

J.1 Ajout de données dans un tableau (push(), unshift() et splice())

Pour ajouter un élément à la fin d'un tableau, il suffit d'utiliser la fonction `push(valeur)` de JavaScript. Ainsi, le langage s'occupe de trouver le dernier élément actuel et de créer un nouvel élément juste après (le tableau s'agrandit).

Pour ajouter un élément avant tous les autres, il faut décaler les éléments vers la droite, ce que fait la fonction `unshift(valeur)`.

Il est possible d'insérer une valeur à partir d'un index en utilisant `splice(position, 0, valeur)`.

J.2 Suppression de données dans un tableau (pop())

Pour supprimer le dernier élément d'un tableau, JavaScript propose la fonction `pop()`.

Pour supprimer le premier élément d'un tableau, JavaScript propose la fonction `shift()`.

Pour supprimer un élément du tableau, en utilisant son index (sa position), la fonction `splice(position, 1)` permet de le faire (elle supprime 1 élément à partir de la position indiquée). Pour supprimer plusieurs élements, il suffit de remplacer 1 par le nombre d'éléments : `splice(position, nombre)`.

J.3 Rechercher une valeur dans un tableau (indexOf())

La fonction `indexOf(valeur)` permet de retrouver l'index de la première valeur trouvée.

J.4 Code d'exemple des fonctions sur les tableaux

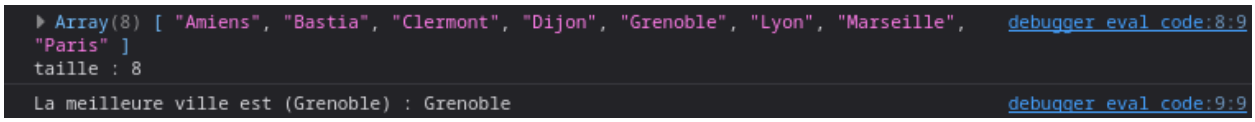
Voici un code utilisant toutes les fonctions précédentes :

```
let mesVilles = ["Bastia", "Clermont", "Dijon", "Grenoble", "Kiev", "Lyon"]

mesVilles.push("Paris")
mesVilles.unshift("Amiens")
mesVilles.splice(7, 0, "Marseille")     // ajout de Marseille
mesVilles.splice(5, 1)                  // suppression de Kiev (pas en France)

console.log(mesVilles, "taille : "+mesVilles.length)
console.log("La meilleure ville est (Grenoble) : "+mesVilles[4])
```

Vous devez obtenir le même affichage que ci-dessous :



J.5 Tableaux : à vous de jouer...

Soit le tableau suivant :

```
let lesMois = ["Février", "March", "Avril", "Mai", "Juillet", "Août", "Septembre", "Octobre"]
```

Insérez le mois de Janvier, Juin, Novembre et Décembre.

Modifiez le mois de Mars (changer son nom anglais par la version française). Attention, il faut le supprimer puis insérer sa nouvelle version.

Indiquez l'instruction pour retrouver la position du mois de Juillet

Combien y'a-t-il de mois dans le tableau (donnez l'instruction) :

Enfin, affichez la valeur de l'index 4 :

K les interactions avec la page HTML

Nous avons vu que document.write() permet d'écrire dans une page web vierge. Cependant, la plupart du temps, les programmes en Javascript s'appuieront sur une page existante : pour ne pas la détruire, il faut donc utiliser d'autres fonctions JavaScript, qui pourront connaître la valeur d'un objet de la page web, modifier ses valeurs ou même créer de nouvelles valeurs. Cette partie s'appuie sur la connaissance HTML (principalement le DOM, les balises `<div>` et les propriétés `id="nom???"`)

Rappel : la page web est un document (un objet nommé 'document').

K.1 récupérer position d'un objet DOM

Il y a deux possibilités en JavaScript :

- Récupérer un seul élément par son ID → `getElementById(nomID)`
- Récupérer des choses, dont un élément ayant un ID → `querySelector(nomID)`

```
<!DOCTYPE html>
<html>
<body>
  <h1>Lire le contenu d'une page web</h1>
  <p id="prenom">Chuck</p>
  <p id="nom">NORRIS</p>

  <script>
    console.log(document.querySelector("#prenom"));
    console.log(document.getElementById("nom"));
  </script>
</body>
</html>
```

L'affichage dans la console est le suivant :

JavaScript affiche... l'objet complet (au format JSON). Si nous voulons retrouver la propriété utile (par exemple, "Chuck") il faut déplier l'objet et chercher quelle propriété contient "Chuck".

Il suffit ensuite de modifier l'appel à la fonction correspondante et d'ajouter `.` Puis le nom de la propriété.

```
console.log(document.querySelector("#prenom").innerText);
```

En faisant défiler toutes les propriétés, on trouvera notamment le style pour changer la couleur du texte ou du fond, les propriétés pour les interactions (onclick(), onfocus(), onmouseover()).

Note : pour querySelector, le paramètre est précédé par un `#` pour un ID ou un `.` Pour une classe. Pour getElementByID, le paramètre étant forcément un ID, il n'y a pas de symbole devant.

K.2 Modifier un objet DOM à la position

Le principe est le même, simplement on met une égalité pour indiquer à JavaScript que l'on affecte une valeur à un attribut. L'attribut sur l'objet est `.innerHTML`. On peut aussi utiliser `.value` sur un champ.

Dans le code suivant, l'affichage n'affichera pas "Chuck NORRIS" mais "Edy LAMARR".

```
<!DOCTYPE html>
<html>
<body>
  <h1>Lire le contenu d'une page web</h1>
  <p id="prenom">Chuck</p>
  <p id="nom">NORRIS</p>

  <script>
    // écrire une valeur
    document.querySelector("#prenom").innerText = "Edy"
    document.getElementById("nom").innerHTML = "<b>LAMARR</b>"

    // afficher la valeur (note : c'est la valeur indiqué sur la page après le script)
    console.log(document.querySelector("#prenom"));
    console.log(document.getElementById("nom"));
  </script>
</body>
</html>
```

K.3 Créer un objet DOM à une position

JavaScript peut aussi créer des éléments HTML, en utilisant `.innerHTML` ou bien `createElement()` :

```
<!DOCTYPE html>
<html>
<body>

    <h1>Création d'objet dans le DOM</h1>
    <p>Comment créer un objet et le modifier</p>

    <script>
        const titre = document.createElement("H1");     // création d'un objet de type balise de titre

        let textNode = document.createTextNode("Un titre rouge");
        titre.appendChild(textNode);
        titre.setAttribute('style', 'color: red');

        document.body.appendChild(titre);               // affichage de l'objet
    </script>

</body>
</html>
```

Note : on peut ainsi ajouter des choses à l'élément créé, grâce à `.appendChild(proprieteDOM)` et aux fonctions `.createTextNode()`, `.createAttribute()` ou `.setAttribute()`.

L'utilisation de `.innerHTML` permet de créer directement des lignes de code HTML :

```
let contenuVariable = "<div id='surnom'>l'homme qui a compté jusqu'à l'infini</div>"
document.getElementById("oEmplacement").innerHTML = contenuVariable
```

Permet de créer un objet de type `<div>` ayant pour ID, "surnom".

K.4 Créer un événement

JavaScript permet de placer des points de surveillance sur un objet : cela signifie qu'il peut lancer une fonction lorsqu'un événement survient sur l'objet : `click`, `change`, `mouseover`, `keyup`, etc.

Lorsqu'on a le nom de l'objet et la fonction à exécuter, il suffit de les associer ensemble :

```
objet.addEventListener('click', fonctionEnExecution)
```

Nom de l'événement, entre guillemets

Nom de la fonction à exécuter, sans parenthèses

oObjet = document.getElementById("objet")

On peut ainsi facilement agir sur des éléments de la page web.

Voici un exemple de compteur qui s'incrémente lorsqu'on clique sur le bouton dans la page.

```
<!DOCTYPE html>
<html>

<body>

  <h1>Compteur de clic</h1>
  Vous avez cliqué <span id="idNombreTxt"></span> fois sur le bouton<br>
  <input type="text" id="idNombreChamp" value="0" readonly>
  <input type="button" id="idCompter" value="Compter">

  <script>
    let compteur = 0
    const oNombreChamp = document.getElementById("idNombreChamp")
    const oNombreTxt = document.getElementById("idNombreTxt")
    const oCompter = document.getElementById("idCompter")
    oCompter.addEventListener('click', AjoutCompteur)

    function AjoutCompteur() {
      compteur = compteur + 1   // ou compteur++
      oNombreTxt.innerHTML = "<strong>" + compteur + "</strong>"
      oNombreChamp.value = compteur
    }

  </script>

</body>

</html>
```

Remarquez que nous utilisons `.value` pour écrire à l'intérieur d'un champ de formulaire alors qu'il faut utiliser `.innerHTML` (ou `.innerText`) pour écrire dans une balise `<div>` ou `<span>`.

K.5 DOM et modification : à vous de jouer…

L'exercice consiste à insérer plusieurs balises en utilisant JavaScript, à l'intérieur d'une balise ayant pour id="listeEtudiants". Les balises insérées auront un style de bordure solide, noire.

Complétez les éléments du code ci-dessous (vous pouvez le tester en sauvant le fichier avec l'extension .HTML et en le testant dans un navigateur)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Liste des élèves concernés :</h1>
    <div id="listeEleves">
    </div>

    <script>
        listeEleves = ["Jérémy", "Florian", "Natasha", "Ulysse", "Alix"]
        oListeEleves = document.getElementById("                    ")
        // insérez le code pour créer un bloc div contenant le nom de
        // chaque élève de la liste (utilisation d'une boucle)
        let oDiv
        for (let t=0; t<listeEleves.length; t++) {
            oDiv = document.createElement("DIV")
            oDiv.setAttribute('            ', 'border: solid black')
            oDiv.innerText = listeEleves[      ]
                document.body.appendChild(              )
        }
    </script>
</body>
</html>
```