

Livret exercices - Bases de l'algo

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livres: Livret exercices - Bases de l'algo

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:53

Description

Premiers pas en algo avec JS : déclarations, instructions, répétitives et conditionnelles

Table des matières

1. Hello World

1.1. Correction

2. Bonus Adrien

3. Cube d'un nombre

3.1. Correction

3.2. Solution 2

4. Perimetre d'un cercle

4.1. Correction

5. Permutation

5.1. Correction

6. Pair ?

6.1. Correction

7. Maximum

7.1. Correction

7.2. Solution 2

8. Signe d'un nombre

8.1. Correction

9. Note

9.1. Correction

10. Signe d'un produit

10.1. Correction

11. Catégories

11.1. Solution 1

11.2. Solution 2

12. Reponse

12.1. Correction

13. L'impôt sur Zorglub

13.1. Correction

14. Factures photocopies

14.1. Correction

15. Assurance automobile

15.1. Correction

16. Somme de n entiers

16.1. Correction

17. Somme des n entiers pairs

17.1. Correction

18. Factorielle

18.1. Correction

19. Table de multiplication

19.1. Correction

20. Plusieurs sommes de n entiers

20.1. Correction

20.2. Solution 2 avec fonction

21. Saisie contrôlée d'un numéro de mois

21.1. Correction

22. FizzBuzz

22.1. Correction

23. Jeu du + et du -

23.1. Correction

24. Entre 1 et 3

24.1. Correction

25. Les 10 nombres suivants

25.1. Correction

26. Plus grand nombre

26.1. Correction

27. Plus grand nombre 2

27.1. Correction

28. Rendu de monnaie

28.1. Correction

1. Hello World

Ecrire un programme pour afficher "Bonjour" à la console.

Modifier le programme ci-dessous pour demander à l'utilisateur d'entrer son nom et et son **prénom** afficher "Bonjour **prénom** nom" concaténé au nom saisi.

Remarque :

Pour réaliser ce programme nous utiliserons une zone de saisie HTML avec le code suivant :

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">Hello World</h1>
<!-- -->
<label for="name">Nom (4 à 8 caractères):</label>
<input type="text" id="name" name="name" required minlength="4" maxlength="8" size="10">
<button id="button">Valider</button>
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur Le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //Affichage du message dans la zone paragraphe id message
    document.getElementById("message").innerHTML = "Bonjour " + document.getElementById("name").value;
  }

  //Fin de code Javascript
</script>
```

1.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP01 - Hello World</h1>
<!-- -->
<label for="name">Prénom (4 à 8 caractères):</label>
<input type="text" id="firstname" name="firstname" required minlength="4" maxlength="8" size="10">
<label for="name">Nom (4 à 8 caractères):</label>
<input type="text" id="name" name="name" required minlength="4" maxlength="8" size="10">
<button id="button">Valider</button>
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
    //Code Javascript
    //Ajout d'un evenement de type click sur le bouton
    document.getElementById("button").addEventListener("click",message);

    //Fonction
    function message() {
        //Affichage du message dans la zone paragraphe id message
        document.getElementById("message").innerHTML = "Bonjour " + document.getElementById("firstname").value + " " +
document.getElementById("name").value;
    }

    //Fin de code Javascript
</script>
```

2. Bonus Adrien

Réaliser un algorithme avec saisie utilisateur et bouton valider qui permet de récupérer selon l'année saisie le vainqueur du tournoi Wimbledon de Tennis par année, sachant que :

- 2018 à 2021 : Novak Djokovic
- 2017 : Roger Federer
- 2016 : Andy Murray
- 2014 à 2015 : Novak Djokovic
- 2013 : Andy Murray

3. Cube d'un nombre

Ecrire un programme qui affiche le cube d'un nombre entier saisi au clavier.

3.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP02 - Calcul d'un cube</h1>
<!-- -->
<label for="name">Saisir un nombre:</label>
<input type="text" id="nombre" name="nombre" required>
<button id="button">Valider</button>
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let nombre = document.getElementById("nombre").value;
    //variable cube
    let cube = nombre * nombre * nombre;
    //Affichage du message dans la zone paragraphe id message
    document.getElementById("message").innerHTML = "Le cube du nombre " + nombre + " = " + cube;
  }

  //Fin de code Javascript
</script>
```

3.2. Solution 2

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">Calcul d'un cube</h1>
<!-- -->
<label for="nombre">Saisir un nombre:</label>
<input type="text" id="nombre" name="nombre" value="2">
<label for="puissance">Saisir une puissance:</label>
<select id="puissance">
  <option value="2">2</option>
  <option value="5">5</option>
  <option value="10">10</option>
</select>
<button id="button">Valider</button>
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let nombre = document.getElementById("nombre").value;
    let puissance = document.getElementById("puissance").value;
    //variable cube
    //let cube = nombre * nombre * nombre;
    //let cube = Math.pow();
    let cube = Math.pow(nombre, puissance);
    //Affichage du message dans la zone paragraphe id message
    document.getElementById("message").innerHTML = "Le cube du nombre " + nombre + ", a la puissance " + puissance + " = " + cube;
  }

  //Fin de code Javascript
</script>
```

4. Perimetre d'un cercle

Écrire un programme qui calcule et affiche le périmètre d'un cercle à partir du rayon saisi par l'utilisateur.

Vous afficherez le résultat sous la forme : "Le cercle de rayon 5 a un périmètre égal à 31,4159".

Rappel : Pour utiliser la constante PI utilisez l'objet `Math.PI`. Pour arrondir un nombre utilisez `toFixed`, référez vous à la [documentation](#).

Formule du cercle :

```
2 * rayon * PI;
```

4.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP03 - Perimetre d'un cercle</h1>
<!-- -->
<label for="name">Saisir un rayon:</label>
<input type="text" id="rayon" name="rayon" required>
<button id="button">Valider</button>
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let rayon = document.getElementById("rayon").value;
    //variable cube
    let perimetre = 2 * rayon * Math.PI;
    //Affichage du message dans la zone paragraphe id message
    //perimetre.toFixed(4) - Pour arrondir un nombre à 4 décimal
    document.getElementById("message").innerHTML = "Le périmètre d'un cercle de rayon " + rayon + " est " + perimetre.toFixed(4);
  }

  //Fin de code Javascript
</script>
```

5. Permutation

Écrire un programme qui permute les valeurs de 2 variables c1 et c2 de type caractère.

Vous initialiserez les variables c1 et c2.

Pour vous permettre de vérifier, vous afficherez les valeurs de c1 et c2 avant et après la permutation.

5.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP04 - Permutation</h1>
<!-- -->
<label for="name">Valeur de C1 :</label>
<input type="text" id="c1" name="c1" value="Chien" disabled >
<label for="name">Valeur de C2 :</label>
<input type="text" id="c2" name="c2" value="Chat" disabled >
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable c1
    let c1 = document.getElementById("c1").value;
    let c2 = document.getElementById("c2").value;
    let c3 = c1;
    c1 = c2;
    c2 = c3;
    //Affichage du message dans la zone paragraphe id message
    //perimetre.toFixed(4) - Pour arrondir un nombre à 4 décimal
    document.getElementById("message").innerHTML = "Permutation C1 = " + c1 + " C2 = " + c2;
  }

  //Fin de code Javascript
</script>
```

6. Pair ?

Écrire un programme demande à l'utilisateur d'entrer un nombre entier et affiche "Pair" si le nombre saisi est pair.

A savoir : l'opérateur % (modulo) permet d'obtenir le reste de la division entière

6.1. Correction

```
<!-- code HTML -->
<h1 id="titre">TP05 - Pair - Réponse</h1>
<label for="name">Un nombre entier ? :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un événement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let reponse = document.getElementById("reponse").value;
    //Affichage du message dans la zone paragraphe id message
    if (reponse% 2 == 0)
    {
      message = "pair";
    }
    else
    {
      message = "impair";
    }

    document.getElementById("message").innerHTML = "Nombre " + message;
  }

  //Fin de code Javascript
</script>
```


7. Maximum

Demandez à l'utilisateur de saisir deux nombres. Afficher le nombre le plus grand.

7.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP06 - Maximum</h1>
<!-- -->
<label for="name">Entrer une valeur 1 :</label>
<input type="number" id="v1" name="v1" required>
<label for="name">Entrer une valeur 2 :</label>
<input type="number" id="v2" name="v2" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un événement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre, ici on est obligé de typer en number
    let v1 = document.getElementById("v1").value;
    let v2 = document.getElementById("v2").value;
    let message;
    //Affichage du message dans la zone paragraphe id message
    if (v1 > v2)
    {
      message = v1;
    }
    else
    {
      message = v2;
    }

    document.getElementById("message").innerHTML = "Max(" + v1 + ", " + v2 + ") = " + message;
  }

  //Fin de code Javascript
</script>
```

7.2. Solution 2

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP06 - Maximum</h1>
<!-- -->
<label for="name">Entrer une valeur 1 :</label>
<input type="number" id="v1" name="v1" required>
<label for="name">Entrer une valeur 2 :</label>
<input type="number" id="v2" name="v2" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre, ici on est obligé de typer en number
    let v1 = document.getElementById("v1").value;
    let v2 = document.getElementById("v2").value;

    document.getElementById("message").innerHTML = "Max(" + v1 + ", " + v2 + ") = " + (v1 > v2 ? v1 : v2);
  }

  //Fin de code Javascript
</script>
```

8. Signe d'un nombre

Étant donné un entier lu à la console, indiquer s'il est nul, positif ou négatif.

8.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP07 - Signe d'un nombre</h1>

<label for="name">Saisir un nombre :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let n = document.getElementById("reponse").value;

    if (n > 0)
    {
      document.getElementById("message").innerHTML = "Positif";
    }
    else if (n < 0)
    {
      document.getElementById("message").innerHTML = "Négatif";
    }
    else
    {
      document.getElementById("message").innerHTML = "Null";
    }
  }

  //Fin de code Javascript
</script>
```

9. Note

Ecrire un programme qui demande à l'utilisateur de saisir une note.
Dans cette version, on suppose que l'utilisateur entre une note valide entre 0 et 20.

Le programme affichera :

- "Admis" si la note est supérieure ou égale à 10
- "Rattrapage" si la note est supérieure ou égale à 8
- "Echec" si la note est inférieure à 8

9.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP08 - Note</h1>

<label for="name">Saisir une note :</label>
<input type="number" id="note">
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let note = document.getElementById("note").value;
    let message = "Echec";

    if (note >= 10) {
      message = "Admis";
    } else if (note >= 8) {
      message = "Rattrapage";
    }

    document.getElementById("message").innerHTML = message;
  }
  //Fin de code Javascript
</script>
```

10. Signe d'un produit

Écrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul).

Attention toutefois : on ne doit **pas** calculer le produit des deux nombres.

Un produit est positif si les 2 nombres sont positifs ou si les 2 nombres sont négatifs

10.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP09 - Signe d'un produit</h1>

<label for="name">Saisir un nombre 1 :</label>
<input type="text" id="n1" name="n1" required>
<label for="name">Saisir un nombre 2 :</label>
<input type="text" id="n2" name="n2" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let n1 = document.getElementById("n1").value;
    let n2 = document.getElementById("n2").value;

    //Signe du produit de 2 nombres
    //On ne traitera pas le cas du produit null
    //Un produit est positif si les 2 nombres sont positifs ou si les 2 nombres sont négatifs
    if ((n1 > 0 && n2 > 0) || (n1 < 0 && n2 < 0))
    {
      document.getElementById("message").innerHTML = n1 + " x " + n2 + " est positif";
    }
    else
    {
      document.getElementById("message").innerHTML = n1 + " x " + n2 + " est négatif";
    }
  }

  //Fin de code Javascript
</script>
```

11. Catégories

Écrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

- "Poussin" de 6 à 7 ans
- "Pupille" de 8 à 9 ans
- "Minime" de 10 à 11 ans
- "Cadet" après 12 ans

Peut-on concevoir plusieurs algorithmes équivalents menant à ce résultat ?

11.1. Solution 1

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP10 - Catégorie</h1>

<label for="name">Age de l'enfant ?:</label>
<input type="text" id="age" name="age" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let age = document.getElementById("age").value;
    let categorie = "Aucune catégorie";

    if (age == 6 || age == 7 )
    {
      categorie = "Poussin";
    }
    if (age == 8 || age == 9)
    {
      categorie = "Pupille";
    }
    if (age == 10 || age== 11)
    {
      categorie = "Minime";
    }
    if (age >= 12)
    {
      categorie = "Cadet";
    }

    document.getElementById("message").innerHTML = "La catégorie d'un enfant âgé de " + age + " ans est " + categorie;
  }

  //Fin de code Javascript
</script>
```

11.2. Solution 2

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP10 - Catégorie</h1>

<label for="name">Age de l'enfant ?:</label>
<input type="text" id="age" name="age" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let age = document.getElementById("age").value;
    let categorie = "Aucune catégorie";

    if (age >= 12)
    {
      categorie = "Cadet";
    }
    else if (age >= 10)
    {
      categorie = "Minime";
    }
    else if (age >= 8)
    {
      categorie = "Pupille";
    }
    else if (age >=6)
    {
      categorie = "Poussin";
    }
    else
    {
      categorie = "Aucune catégorie";
    }

    document.getElementById("message").innerHTML = "La catégorie d'un enfant âgé de " + age + " ans est " + categorie;

  }

  //Fin de code Javascript
</script>
```

12. Reponse

Écrire un programme qui demande à l'utilisateur de saisir un caractère.

Le programme devra afficher :

- "affirmatif" si le caractère est "o" (minuscule ou majuscule),
- "négatif" si le caractère est "n" (minuscule ou majuscule)
- "???" dans les autres cas.

12.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP11 - Réponse</h1>
<label for="name">Entrer une réponse (O/N) :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un événement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let reponse = document.getElementById("reponse").value;
    let message;
    //Affichage du message dans la zone paragraphe id message
    if (reponse == "o" || reponse == "O")
    {
      message = "Affirmatif";
    }
    else if (reponse == "n" || reponse == "N")
    {
      message = "Négatif";
    }
    else
    {
      message = "Ni Oui ni Non";
    }

    document.getElementById("message").innerHTML = message;
  }

  //Fin de code Javascript
</script>
```

13. L'impôt sur Zorglub

Les habitants de Zorglub paient l'impôt selon les règles suivantes :

- les hommes de plus de 20 ans paient l'impôt
- les femmes paient l'impôt si elles ont entre 18 et 35 ans
- les autres ne paient pas d'impôt

Le programme demandera donc l'âge et le sexe du Zorglubien, et se prononcera donc ensuite sur le fait que l'habitant est imposable.

13.1. Correction

```
<!-- code HTML -->
<h1 id="titre">TP12 - Zorglub</h1>
<!-- -->
<label for="genre">Quel est votre genre ? :</label>
<select id="genre">
  <option value="femme">Femme</option>
  <option value="homme">Homme</option>
</select>
<label for="age">Quel est votre age ? :</label>
<input type="number" id="age">
<button id="button">Valider</button>
<p id="message"></p>

<script>
  //Code Javascript
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let genre = document.getElementById("genre").value;
    let age = document.getElementById("age").value;
    let message = "non impossible";
    //Affichage du message dans la zone paragraphe id message
    if ((genre == "homme" && age > 20) || (genre == "femme" && (age >= 18 && age <= 35)))
    {
      message = "imposable";
    }
    /* Autre solution en deux étapes
    else if (genre == "femme" && (age >= 18 && age <= 35))
    {
      message = "imposable";
    }
    */
    document.getElementById("message").innerHTML = `L'individu ${genre} de ${age} ans est ${message}`;
  }

  //Fin de code Javascript
</script>
```


14. Factures photocopies

Un magasin de reprographie facture 0,10 € les dix premières photocopies, 0,09 € les vingt suivantes et 0,08 € au-delà. Écrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées et qui affiche la facture correspondante.

14.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP11 - Réponse</h1>
<!-- -->
<label for="name">Nombre de photocopies :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let nbPhotocopies = document.getElementById("reponse").value;
    let montantFacture = 0;

    //Affichage du message dans la zone paragraphe id message
    // calculer le montant de la facture
    if (nbPhotocopies <= 10) // calculer le montant pour 10 ou moins de photocopies
    {
      montantFacture = nbPhotocopies * 0.1; // nb_photocopies * 0.1
    }
    else if(nbPhotocopies <= 30 ) // calculer le montant pour 11 à 30 photocopies
    {
      montantFacture = 1 + (nbPhotocopies - 10) * 0.09; //10*0.1 + (nb_photocopies-10)*0.09
    }
    else
    {
      montantFacture = 2.8 + (nbPhotocopies - 30) * 0.08; // 10 * 0.1 + (nb_photocopies - 10) * 0.09
    }

    document.getElementById("message").innerHTML = "Facture : " + montantFacture + " €";
  }

  //Fin de code Javascript
</script>
```

15. Assurance automobile

Une compagnie d'assurance automobile propose à ses clients quatre familles de tarifs identifiables par une couleur, du moins au plus onéreux : tarifs bleu, vert, orange et rouge. Le tarif dépend de la situation du conducteur :

- un conducteur de moins de 25 ans et titulaire du permis depuis moins de deux ans, se voit attribuer le tarif rouge, si toutefois il n'a jamais été responsable d'accident. Sinon, la compagnie refuse de l'assurer.
- un conducteur de moins de 25 ans et titulaire du permis depuis plus de deux ans, ou de plus de 25 ans mais titulaire du permis depuis moins de deux ans a le droit au tarif orange s'il n'a jamais provoqué d'accident, au tarif rouge pour un accident, sinon il est refusé.
- un conducteur de plus de 25 ans titulaire du permis depuis plus de deux ans bénéficie du tarif vert s'il n'est à l'origine d'aucun accident et du tarif orange pour un accident, du tarif rouge pour deux accidents, et refusé au-delà
- De plus, pour encourager la fidélité des clients acceptés, la compagnie propose un contrat de la couleur immédiatement la plus avantageuse s'il est entré dans la maison depuis plus de cinq ans. Ainsi, s'il satisfait à cette exigence, un client normalement "vert" devient "bleu", un client normalement "orange" devient "vert", et le "rouge" devient orange.

Écrire l'algorithme permettant de saisir les données nécessaires (sans contrôle de saisie) et de traiter ce problème. Avant de se lancer à corps perdu dans cet exercice, on pourra réfléchir un peu et s'apercevoir qu'il est plus simple qu'il n'en a l'air (cela s'appelle faire une analyse !)

15.1. Correction

```
<h1 id="titre">TP14 - Assurance</h1>

<label for="age">Votre age :</label>
<input type="number" id="age" min="10" max="100"><br>
<label for="annee">Nombre d'années de permis :</label>
<input type="text" id="annee" size="4"><br>
<label for="accidents">Nombre d'accidents :</label>
<input type="text" id="accidents" size="4"><br>
<label for="anciennete">Nombre d'années d'ancienneté :</label>
<input type="text" id="anciennete" size="4"><br>

<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
    document.getElementById("button").addEventListener("click",message);

    function message() {
        //variables, type Number pour manipuler nombre (en l'etat type string)
        let age = Number(document.getElementById("age").value);
        let annee = Number(document.getElementById("annee").value);
        let accidents = Number(document.getElementById("accidents").value);
        let anciennete = Number(document.getElementById("anciennete").value);
        let points = 1;
        let contrat = "";

        if (age < 25){
            points++;
        }
        if(annee < 2){
            points++;
        }

        points += accidents;

        if(anciennete > 5 && points < 4){
            points--;
        }

        switch (points) {
            case 0: // equivalent points == 0
                contrat = "Bleu";
                break;//le break permet d'arreter la suite des instructions
            case 1:
                contrat = "Vert";
                break;
            case 2:
                contrat = "Orange";
                break;
            case 3:
                contrat = "Rouge";
                break;
            default:
                contrat = "Refusé";
        }

        document.getElementById("message").innerHTML = "Contrat : " + contrat;
    }
</script>
```

16. Somme de n entiers

Écrire un programme qui calcule la somme des n entiers inférieurs ou égaux à n.

Dans un première version, fixer $n = 5$.

Dans une seconde version, demander à l'utilisateur d'entrer la valeur de n.

16.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP15 - Somme de n entiers</h1>
<!-- -->
<label for="name">Valeur de n :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un événement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let n = document.getElementById("reponse").value;
    let somme = 0;
    let i = 1;

    while (i <= n)
    {
      somme = somme + i;
      i = i + 1; //ou i++
    }

    document.getElementById("message").innerHTML = "La somme des " + n + " entiers est égale à " + somme;
  }

  //Fin de code Javascript
</script>
```

17. Somme des n entiers pairs

Ecrire un programme qui calcule la somme des n premiers entiers pairs.
L'entier n sera saisi par l'utilisateur.

17.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP16 - Somme de n entiers pairs</h1>
<!-- -->
<label for="name">Valeur de n :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un événement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let n = document.getElementById("reponse").value;
    let somme = 0;
    let i = 1;

    while (i <= n)
    {
      if (i % 2 == 0) //i est pair
      {
        somme = somme + i;
      }
      i = i + 1; //ou i++
    }

    document.getElementById("message").innerHTML = "La somme des " + n + " entiers pairs est égale à " + somme;
  }

  //Fin de code Javascript
</script>
```


18. Factorielle

Écrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

NB : la factorielle de 8, notée 8 !, vaut

1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

18.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP17 - Factorielle </h1>
<!-- -->
<label for="name">Veuillez rentrer un nombre :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un événement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let n = Number(document.getElementById("reponse").value);
    let factorielle = 1;

    for (let i = 2; i <= n; i++)
    {
      factorielle = factorielle * i;
    }

    document.getElementById("message").innerHTML = n + " ! = " + factorielle;
  }

  //Fin de code Javascript
</script>
```

19. Table de multiplication

Écrire un algorithme qui demande à l'utilisateur d'entrer un nombre entier.
Le programme affichera la table de multiplication de ce nombre, présentée comme suit.

Exemple : Table de 7

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

...

7 x 10 = 70

Remarque : Il est possible d'auto concaténer avec `innerHTML`.

Votre message doit être dans une balise HTML `<pre>` si vous souhaitez utiliser le retour chariot `\n`.

19.1. Correction

Avec une boucle while

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP18 - Table de multiplication</h1>
<!-- -->
<label for="name">Table de n :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un événement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let n = document.getElementById("reponse").value;
    let i = 1;

    while (i <= 10)
    {
      //remarque, ici on utilise += pour auto concaténer le texte, et \n pour retour à la ligne dans la balise pre
      document.getElementById("message").innerHTML += n + " x " + i + " = " + n*i + "\n";
      i++;
    }

  }

  //Fin de code Javascript
</script>
```

Avec une boucle do ... while

```
do{
  //remarque, ici on utilise += pour auto concaténer le texte, et \n pour retour à la ligne dans la balise pre
  document.getElementById("message").innerHTML += n + " x " + i + " = " + n*i + "\n";
  i++;
}while (i <= 10)
```

Avec une boucle for

```
for(let i = 1;i <= 10;i++){
  //remarque, ici on utilise += pour auto concaténer le texte, et \n pour retour à la ligne dans la balise pre
  document.getElementById("message").innerHTML += n + " x " + i + " = " + n*i + "\n";
}
```

20. Plusieurs sommes de n entiers

Écrire un programme qui calcule la somme des n premiers entiers, n étant saisi à la console.
Le programme devra donner la possibilité à l'utilisateur de faire un nouveau calcul ou d'arrêter le programme.

```
let n = prompt("Entrer un nombre");
```

```
reponse = prompt(message + "Autre calcul (O) ?");
```

Remarque : Pour ce TP nous utiliserons la méthode [Prompt](#) qui permet d'interagir via une fenêtre de saisie avec l'utilisateur.

```
let signe = prompt("Quel est votre signe astrologique ?");

if (signe.toLowerCase() == "verseau") {
  console.log("Oh ? moi aussi je suis verseau :)");
}
```

Nous utiliserons cette méthode uniquement pour ce TP dans une logique de boucle. En effet, bien que compatible avec les navigateurs, dans les faits cette méthode présente des failles de sécurité, on préférera appliqué des fenêtres de type HTML/CSS, mais plus compliqué à mettre en oeuvre.

20.1. Correction

```
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<script>
  //Code Javascript
  let reponse;
  let message;
  do
  {
    let n = prompt("Entrer un nombre");

    let somme = 0;
    let i = 1;

    while (i <= n)
    {
      somme = somme + i;
      i = i + 1; //ou i++
    }
    message = "La somme des " + n + " entiers est égale à " + somme + "\n";
    document.getElementById("message").innerHTML = message;
    reponse = prompt(message + "Autre calcul (0) ?");

  }while (reponse == "0" || reponse == "o");
  //Fin de code Javascript
</script>
```

20.2. Solution 2 avec fonction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP19 - Somme de n entiers avec proposition de relancer le programme</h1>

<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript

  let reponse = "";

  do{
    let resultat = message();
    reponse = prompt(resultat + "Voulez vous continuer le programme ? (O/N)");
  }
  while (reponse == "O");

  //Fonction
  function message() {
    //variable nombre
    let n = prompt("Entrer un nombre");
    let somme = 0;
    let i = 1;

    while (i <= n)
    {
      somme = somme + i;
      i = i + 1; //ou i++
    }
    let resultat = "La somme des " + n + " entiers est égale à " + somme + "\n";
    document.getElementById("message").innerHTML += resultat;
    return resultat;
  }

  //Fin de code Javascript
</script>
```

21. Saisie contrôlée d'un numéro de mois

On souhaite réaliser la saisie d'un numéro de mois (compris entre 1 et 12) avec vérification.
Si la saisie est incorrecte, le programme demandera à l'utilisateur de saisir une nouvelle donnée.

Remarque : Utilisez une variable de type boolean pour cet exercice.

21.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP20 - Saisie contrôlée d'un numéro de mois</h1>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<script>
  //Code Javascript
  let valide = false;
  let mois;
  while (!valide)
  {
    mois = prompt("Entrer un nombre");
    valide = mois >= 1 && mois <= 12;
  }
  document.getElementById("message").innerHTML = "Bravo, vous avez saisi " + mois;
  //Fin de code Javascript
</script>
```

22. FizzBuzz

Etape 1 - Le challenge

Le but est de lister les nombres de 1 à 100.
Chaque nombre sera remplacé par un mot selon certaines conditions :

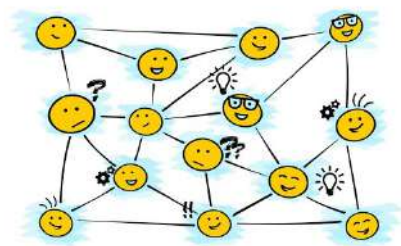
- Par le mot **Fizz** si le nombre est **multiple de 3** ;
- Par le mot **Buzz** si le nombre est **multiple de 5** ;
- Par le mot **FizzBuzz** si le nombre est **multiple de 3 et de 5** ;
- Le nombre sera affiché s'il n'est ni multiple de 3 ni multiple de 5.



% renvoie le modulo, le reste d'une division entière

A vous de jouer ! Codez ce programme en JS et testez-le !

Etape 2 - L'évaluation croisée entre pairs



Quand vous avez terminé, trouvez un collègue qui lui aussi terminé et échangez vos solutions.
Prenez le temps d'examiner vos solutions et faites un revue croisée.

- Les résultats sont conformes à ceux attendus ?
- Le code est-il bien structuré, indenté ?
- Les variables sont-elles clairement nommées

22.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP21 - FizzBuzz</h1>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<script>
  //Code Javascript
  let message;

  for (let i = 1; i <= 100; i++)
  {
    //Cette condition doit être en premier
    //autre option if (i % 15 == 0)
    if (i % 3 == 0 && i % 5 == 0)
    {
      message = "FizzBuzz";
    }
    else if (i % 3 == 0)
    {
      message = "Fizz";
    }
    else if (i % 5 == 0)
    {
      message = "Buzz";
    }
    else
    {
      message = i;
    }
    document.getElementById("message").innerHTML += message + "\n";
  }
  //Fin de code Javascript
</script>
```

Plus simple :

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">TP21 - FizzBuzz</h1>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<script>
  //Code Javascript
  let message;

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

    if (i % 3 == 0)
    {
      message += "Fizz";
    }
    if (i % 5 == 0)
    {
      message += "Buzz";
    }

    document.getElementById("message").innerHTML += message + "\n";
  }
  //Fin de code Javascript
</script>
```

23. Jeu du + et du -

Le programme tire au sort un nombre, entre 1 et 100.

L'utilisateur doit le deviner en s'approchant du nombre à trouver.

Quand l'utilisateur saisit un nombre, l'ordinateur affiche :

- **+** si le nombre saisi est **trop petit**
- **-** si le nombre saisi est **trop grand**
- **Bravo** si le nombre est **trouvé**

En plus, le programme affichera le nombre de coups pour trouver le nombre.

Utiliser la méthode `Prompt()` pour cet exercice.

A savoir : pour tirer un nombre aléatoire

```
function getRandomInt(max) {  
    return Math.floor(Math.random() * max);  
}  
  
console.log(getRandomInt(3));  
// Sortie possible: 0, 1 ou 2
```

23.1. Correction

```
<!-- code HTML -->
<h1 id="titre">TP07 - Signe</h1>
<pre id="message"></pre>
<script>
  //Code Javascript
  let message = "";
  let nbADeviner = 1 + Math.floor(Math.random() * 100); //Tire un nombre aléatoire entre [1,100]
  let nbSaisi = 0;
  let nbEssais = 0;
  do
  {
    nbEssais++;
    nbSaisi = prompt(message + "Entrer un nombre");

    if (nbSaisi > nbADeviner) {
      message = "Trop grand - ";
    } else if (nbSaisi < nbADeviner)
    {
      message = "Trop petit - ";
    }

  } while (nbSaisi != nbADeviner);

  document.getElementById("message").innerHTML = "Bravo, vous avez trouvé le nombre " + nbADeviner + " en " + nbEssais + " essais";

  //Fin de code Javascript
</script>
```

24. Entre 1 et 3

Écrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

24.1. Correction

```
<!-- code HTML -->
<h1 id="titre">TP23 - 1 à 3</h1>
<pre id="message"></pre>
<script>
  //Code Javascript
  let value = 0;
  do{

    value = prompt("Veuillez saisir un nombre :");

  }while (value < 1 || value > 3);

  document.getElementById("message").innerHTML = "Bravo, vous avez saisi le nombre " + value;

  //Fin de code Javascript
</script>
```

25. Les 10 nombres suivants

Écrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Écrivez deux versions de l'algorithme, une avec une boucle While et une avec une boucle For.

25.1. Correction

```
<h1>TP24 - 10 Nombres</h1>
<pre id="message"></pre>
<script>

  let nombre;
  let i = 0;
  nombre = prompt("Veuillez saisir un nombre");
  //typeof indique le type de la variable
  console.log(typeof nombre);
  //convertir la variable de type string en nombre
  nombre = Number(nombre);

  //les trois type de conversion
  //Number(variable); parseFloat(variable); parseInt(variable)
  //String(variable); variable.toString();
  document.getElementById("message").innerHTML += "Boucle While :\n";
  while(i < 10){

    i++;
    document.getElementById("message").innerHTML += nombre + i + "<br>";

  }
  document.getElementById("message").innerHTML += "Boucle For :<br>";
  //avec boucle for
  for (let n = 1; n <= 10; n++){

    document.getElementById("message").innerHTML += nombre + n + "<br>";

  }

</script>
```

26. Plus grand nombre

Écrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12

Entrez le nombre numéro 2 : 14

etc.

Entrez le nombre numéro 20 : 6

Le plus grand de ces nombres est : 14

Modifiez ensuite l'algorithme pour que le programme affiche de surcroît en quelle position avait été saisie ce nombre :

C'était le nombre numéro 2

26.1. Correction

```
<h1>TP25 - Plus Grand nombre</h1>
<pre id="message"></pre>
<script>

  let nombre;
  let max = 0;
  let i = 0;
  let counter = 0;

  while (i < 20){
    i++;
    nombre = prompt("Veuillez saisir un nombre");
    if(max < nombre){
      max = nombre;
      counter = i;
    }
  }

  document.getElementById("message").innerHTML += "Le plus grand nombre est " + max + ", saisie à la " + counter + "ème proposition.";
</script>
```

27. Plus grand nombre 2

Réécrire l'algorithme précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

27.1. Correction

```
<h1>TP26 - Plus Grand nombre</h1>
<pre id="message"></pre>
<script>

  let nombre;
  let max = 0;
  let i = 0;
  let counter = 0;

  do{
    i++;
    nombre = prompt("Veuillez saisir un nombre");
    if(max < nombre){
      max = nombre;
      counter = i;
    }

  }while (nombre != 0)

  document.getElementById("message").innerHTML += "Le plus grand nombre est " + max + ", saisie à la " + counter + "ème proposition.";

</script>
```

28. Rendu de monnaie

On souhaite réaliser un programme de gestion de caisse dans une superette :

- Permettre à un client de saisir le prix de différents articles qu'il met dans son panier (en euros entiers)
 - si prix = 0 mettre fin à la saisie des prix du client
- Indiquez le montant total de la facture
- Permettre au client qui n'a pas l'appoint d'entrer le montant qu'il donne au caissier (montant se terminant par zéro ou 5 €)
- Simuler la remise de la monnaie en affichant les textes "10 Euros", "5 Euros" et "1 Euro" autant de fois qu'il y a de coupures de chaque sorte à rendre.

28.1. Correction

```
<h1>TP27 - Rendu de monnaie</h1>

<pre id="factures"></pre>
<label>Montant versé par le client :</label>
<input id="montant">
<button id="button">Valider</button>
<pre id="monnaie"></pre>
<script>

    document.getElementById("button").addEventListener("click",montant);
    let factures = 0;
    let article = 0;

    do{

        article = Number(prompt("Veuillez saisir le prix de l'article"));
        factures += article;

    }while(article != 0);

    document.getElementById("factures").innerHTML = `La facture est de ${factures} €`;

    function montant(){

        let montant = Number(document.getElementById("montant").value);
        let message = "";
        let renduMonnaie = montant - factures;
        let nbBillet10 = 0;
        let nbBillet5 = 0;

        message += `La monnaie à rendre est de ${renduMonnaie} € \n`;
        message += `Rendu monnaie :\n`;

        while(renduMonnaie >= 10){
            nbBillet10++;
            renduMonnaie -= 10;
        }
        if(renduMonnaie >= 5){
            nbBillet5++;
            renduMonnaie -= 5;
        }

        message += `Nombre de billet de 10 € = ${nbBillet10}\n`;
        message += `Nombre de billet de 5 € = ${nbBillet5}\n`;
        message += `Nombre de pièce de 1 € = ${renduMonnaie}\n`;

        document.getElementById("monnaie").innerHTML += message;

    }

</script>
```

Livret exercices - Tableaux

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Livret exercices - Tableaux

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:54

Description

Pour comprendre les usages des tableaux et savoir les utiliser en JS

Table des matières

1. Maximum

2. Moyenne

2.1. Correction

3. Voyelles

3.1. Correction

4. Que fait ce programme ?

4.1. Correction

5. Somme de 2 tableaux

5.1. Correction

6. Schtroumpf

6.1. Correction

7. Tri à bulle

7.1. Vidéo

7.2. Solution 1

7.3. Solution 2

8. Recherche dichotomique

8.1. Aide

8.2. Correction

1. Maximum

A partir du tableau ci-dessous :

```
let tableau = [15, 1, 8, 50, 12, 33, 10, 46];
```

Afficher la valeur la plus grande de ce tableau. Pour vous aider inspirez vous du [TP_plus grand nombre](#).

2. Moyenne

A partir de l'exemple du cours "Calcul de la moyenne de 5 notes", remanier le programme pour permettre la saisie de nouvelles notes.

Afficher l'ensemble des notes et la moyenne.

Vous est il possible de saisir plusieurs nouvelles notes ?

2.1. Correction

```
<label for="name">Saisir une note :</label>
<input type="text" id="note">
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>

    //Porté en dehors de la fonction message
    let notes = [12, 14, 08, 13, 12];
    document.getElementById("button").addEventListener("click",message);

    //Fonction
    function message() {
        //variables
        let note = Number(document.getElementById("note").value);
        let moyenne = 0;

        notes.push(note);
        notes["toto"] = 20;

        //Calculer la moyenne
        notes.forEach(function(item, index, array) {

            document.getElementById("message").innerHTML += "Note[" + index + "] : " + item + "\n";
            moyenne += item;
            console.log(array);

        });

        //longeur du tableau notes.length
        moyenne = moyenne / notes.length;

        //Afficher le résultat
        document.getElementById("message").innerHTML += "La moyenne est : " + Math.round(moyenne) + "\n";

        //Solution avec une boucle For

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

            document.getElementById("message").innerHTML += "Note[" + i + "] : " + notes[i] + "\n";

        }

    }

</script>
```

3. Voyelles

Déclarer un tableau contenant les 6 voyelles de l'alphabet latin.

Afficher le contenu du tableau.

3.1. Correction

```
<pre id="message"></pre>
<script>

  let voyelles = ['a', 'e', 'i', 'o', 'u', 'y'];

  for (let i = 0; i < voyelles.length; i++) //Length = nombre d'éléments du tableau
  {
    document.getElementById("message").innerHTML += voyelles[i] + " , ";
  }

</script>
```

4. Que fait ce programme ?

Quel est le résultat de l'algorithme ci-dessous ?

```
let tableau = [];  
  
for (let i = 0; i < 6; i++)  
{  
  tableau.push(i*i);  
}  
  
for (let i = 0; i < tableau.length; i++)  
{  
  document.getElementById("message").innerHTML += tableau[i] + "\n";  
}
```

Peut-on simplifier cet algorithme avec le même résultat ?

4.1. Correction

Résultat affiché

```
0 1 4 9 16 25
```

Simplification

Possible en une seule boucle

```
<pre id="message"></pre>

<script>

  let tableau = [];

  for (let i = 0; i < 6; i++)
  {
    tableau.push(i*i);
    document.getElementById("message").innerHTML += tableau[i] + "\n";
  }

</script>
```

5. Somme de 2 tableaux

Ecrire un programme qui fait la somme de 2 tableaux de même longueur préalablement remplis.

Exemple :

Tableau 1

4	8	7	9	1	5	4	6
---	---	---	---	---	---	---	---

Tableau 2

7	6	5	2	1	3	7	4
---	---	---	---	---	---	---	---

Résultat : Tableau 3, somme du tableau1 et du tableau 2

11	14	12	11	2	8	11	10
----	----	----	----	---	---	----	----

5.1. Correction

```
<pre id="message"></pre>

<script>

  let tab1 = [4, 8, 7, 9, 1, 5, 4, 6];
  let tab2 = [7, 6, 5, 2, 1, 3, 7, 4];
  let tab3 = [];

  for (let i = 0; i < tab1.length; i++)
  {
    tab3.push(tab1[i] + tab2[i]);
    document.getElementById("message").innerHTML += tab3[i] + ", ";
  }

</script>
```

6. Schtroumpf

Ecrire un programme qui calcule le schtroumpf de 2 tableaux.
Pour calculer le schtroumpf, il faut multiplier chaque élément du tableau 1 par chaque élément du tableau 2 et additionner le tout.

Tableau 1

4	8	7	12
---	---	---	----

Tableau 2

3	6
---	---

Résultat

$3 \cdot 4 + 3 \cdot 8 + 3 \cdot 7 + 3 \cdot 12 + 6 \cdot 4 + 6 \cdot 8 + 6 \cdot 7 + 6 \cdot 12 = 279$

6.1. Correction

```
<pre id="message"></pre>

<script>

  let tab1 = [4, 8, 7, 12];
  let tab2 = [3, 6];
  let tab3 = [];
  let somme = 0;
  let size = 0;

  for (let i = 0; i < tab1.length; i++)
  {
    for(let j = 0; j < tab2.length; j++){
      tab3.push(tab1[i] * tab2[j]);
      document.getElementById("message").innerHTML += "(" + tab1[i] + " * " + tab2[j] + ")[" + tab3[size] + "]" + " + ";
      somme += tab3[size];
      //ou plus simplement somme += tab1[i] * tab2[j];
      size++;
    }
  }
  document.getElementById("message").innerHTML += " = " + somme;

</script>
```

7. Tri à bulle

Principe

Il s'agit de **trier les éléments d'un tableau** en utilisant l'algorithme du **tri à bulle** (BubbleSort).

L'algorithme parcourt le tableau et compare les éléments consécutifs.

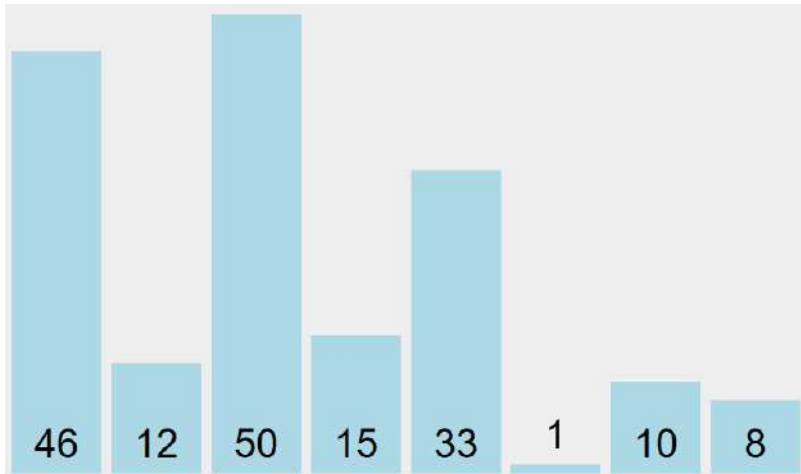
Lorsque **deux éléments consécutifs ne sont pas dans l'ordre**, ils sont **permutés**.

Après un premier parcours complet du tableau, le plus grand élément est forcément en fin de tableau, **à sa position définitive**. En effet, aussitôt que le plus grand élément est rencontré durant le parcours, il est mal trié par rapport à tous les éléments suivants, donc échangé à chaque fois jusqu'à la fin du parcours.

Le reste du tableau est en revanche **encore en désordre**. Il faut **reparcourir le tableau** jusqu'à qu'il n'y ait **plus de permutation**.

cet algorithme doit son nom au fait qu'il déplace rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide.

Un exemple



```
let tableau = [15, 1, 8, 50, 12, 33, 10, 46];
```

Conseils de réalisation :

- Déclarer un tableau d'entiers tel que dans l'image ci-dessus.
- Dans un premier temps, coder la première passe (le premier parcours du tableau) et constater que l'élément le plus grand est à la dernière position du tableau.
- Puis reparcourir le tableau autant de fois que nécessaire pour trier tous les éléments. Indiquer le nombre d'itération.
- Essayer d'obtenir un nombre d'itération le plus faible (optimisé)

7.1. Vidéo

7.2. Solution 1

36 itérations.

```
<pre id="message">

</pre>
<script>

    let tableau = [15, 1, 8, 50, 12, 33, 10, 46];
    let iteration = 0;

    restitutionResultat(false);

    for (let nbCaseAparcourir = tableau.length; nbCaseAparcourir > 0; nbCaseAparcourir--){

        for (let i = 0; i < nbCaseAparcourir - 1; i++){

            if(tableau[i] > tableau[i+ 1]){ //permutation

                let temp = tableau[i];
                tableau[i] = tableau[i+ 1];
                tableau[i+ 1] = temp;

            }
            iteration++;
        }
        iteration++;
    }

    restitutionResultat(true);

    function restitutionResultat(type){

        let message = `Tableau ${type ? '' : 'non '}trié :\n`;

        for (let i = 0; i < tableau.length; i++){
            message += `${tableau[i]}\n`;
        }
        document.getElementById("message").innerHTML += message;
        document.getElementById("message").innerHTML += `Nombre d'itération : ${iteration}\n`;
    }

</script>
```


7.3. Solution 2

26 itérations.

```
<pre id="message">

</pre>
<script>

    let tableau = [15, 1, 8, 50, 12, 33, 10, 46];
    let iteration = 0;
    let nbCaseAparcourir = tableau.length - 1;

    restitutionResultat(false);

    for (let j = 0; j < nbCaseAparcourir; j++){

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

            if(tableau[i] > tableau[i+ 1]){ //permutation

                let temp = tableau[i];
                tableau[i] = tableau[i+ 1];
                tableau[i+ 1] = temp;

            }
            iteration++;
        }
        iteration++;
        nbCaseAparcourir--;
    }

    restitutionResultat(true);

    function restitutionResultat(type){

        let message = `Tableau ${type ? '' : 'non '}trié :\n`;

        for (let i = 0; i < tableau.length; i++){
            message += `${tableau[i]}\n`;
        }
        document.getElementById("message").innerHTML += message;
        document.getElementById("message").innerHTML += `Nombre d'itération : ${iteration}\n`;
    }

</script>
```

8. Recherche dichotomique

Il s'agit de rechercher la place d'un élément donné dans un tableau.

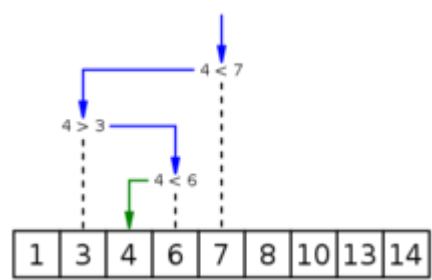
Pour cela, on déclarera un tableau d'entiers trié déjà par ordre croissant :

- 1, 3, 4, 6, 7, 8, 10, 13, 14

L'utilisateur entrera l'élément à chercher sur un champs input.

Le principe

Au lieu de rechercher séquentiellement du premier jusqu'au dernier, on compare l'élément à chercher à l'élément qui se trouve au milieu du tableau. Si c'est le même, on retourne le rang du milieu sinon on recommence sur la première moitié (ou la deuxième) si l'élément recherché est plus petit (ou plus grand).



Visualisation d'une recherche dichotomique, où 4 est la valeur recherchée.

Résultat attendu

On renvoie la place (l'index) de l'élément s'il est trouvé dans le tableau ou -1 lorsque l'élément n'est pas présent dans le tableau. Indiquer le nombre d'itération.

Remarque :

```
break; //permet de stopper un tableau

while(a <= b){
    if(x == y){
        //on a trouvé nombre
        //arrêt de la boucle
        break;
    }
}

Pour arrondir un nombre en entier réel utilisé Math.round(5/2); //3 au lieu de 2.5
```

8.1. Aide

Trop difficile ? Essayez de transposer le code suivant :

```
a = 0
b = len(t) - 1
while a <= b:
    m = (a + b) / 2
    if t[m] == v:
        //on a trouvé v
        break;
    elif t[m] < v:
        a = m + 1
    else:
        b = m - 1
```

8.2. Correction

```
<label>Cherche le nombre ?</label>
<input id="nombre" type="number">
<button id="chercher">Valider</button>

<pre id="message"></pre>
<script>

  let tableau = [1, 3, 4, 6, 7, 8, 10, 13, 14];

  document.getElementById("chercher").addEventListener("click", chercher);

  function chercher() {

    let nombre = Number(document.getElementById("nombre").value);
    let premierNombre = 0;
    let moitie = 0;
    let dernierNombre = tableau.length - 1;
    let iteration = 0;
    let message = "";

    while(premierNombre <= dernierNombre){

      iteration++;

      moitie = Math.round((premierNombre + dernierNombre) / 2);

      if(nombre == tableau[moitie]){
        message = `Ce nombre se trouve à l'index ${moitie}`;
        break;
      }else if(nombre > tableau[moitie]){

        premierNombre = moitie + 1;

      }else{

        dernierNombre = moitie - 1;

      }

    }

    if(premierNombre > dernierNombre){

      message = `Ce nombre n'est pas présent dans le tableau.`;

    }

    document.getElementById("message").innerHTML += `${message} après ${iteration} itération(s).\n` ;

  }

</script>
```

Livret exercices - Fonctions

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Livret exercices - Fonctions

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:54

Description

Réaliser des fonctions pour comprendre le passage de paramètres et leurs usages.

Table des matières

1. Bonjour

1.1. Correction

2. Appel de fonction

2.1. Table de multiplication

3. EstPair

3.1. Correction

4. Mention

4.1. Correction

5. Distance

5.1. Correction

6. OuiNon

6.1. Aide

6.2. Correction

7. Conversion en binaire

7.1. decimalToBinary

7.2. Correction

8. Inversion de chaine

8.1. Correction

9. Bissextile

9.1. Correction

10. Compression RLE

10.1. Correction

11. Code cesar

11.1. Correction

12. EstPremier

12.1. Correction - Etape 1

12.2. Correction - Etape 2

13. Factorielle

13.1. Solution

1. Bonjour

Ecrire une fonction pour dire bonjour à une personne.
Le nom de la personne sera passé en paramètre.
L'affichage se fera à la console.

1.1. Correction

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">Hello World</h1>
<!-- -->
<label for="name">Prénom (4 à 8 caractères):</label>
<input type="text" id="firstname" name="firstname" required minlength="4" maxlength="8" size="10">
<label for="name">Nom (4 à 8 caractères):</label>
<input type="text" id="name" name="name" required minlength="4" maxlength="8" size="10">
<button id="button">Valider</button>
<pre id="message"></pre>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",Message);

  //Fonction
  function Message() {
    let nom = document.getElementById("name").value;
    let prenom = document.getElementById("firstname").value;
    //appel fonction
    DireBonjour(nom, prenom);
    //appel fonction
    DireBonjour("DWM", "Algorithme");
  }

  //Fonction DireBonjour
  function DireBonjour(nom, prenom) {
    document.getElementById("message").innerHTML += "Bonjour " + nom + " " + prenom + "\n";
  }

  //Fin de code Javascript
</script>
```

2. Appel de fonction

- Etape 1

Créer 4 fonctions qui s'appelle :

- Martin
- Tom
- Paul
- Gerard

Utiliser un champs Input pour saisir l'un des noms de fonction ci-dessus.

Faire en sorte que chaque fonction retourne dans une balise HTML le message "je suis la fonction 'xxx'" : xxx étant le nom de la fonction.

- Etape 2

Ajouter un champs de saisie Nom.

Créer quatre nouvelles fonctions "Dupont", "Durand", "Couder", "Dupond" qui retourne un nom.

Afficher le résultat sous la forme d'une concatenation de deux fonctions, pour obtenir le message suivant "Bonjour, Tom Couder" par exemple.

2.1. Table de multiplication

Nous souhaiterions créer 4 fonctions :

- addition, multiplication, division, soustraction
- chaque fonction doit avoir en argument une variable nommé number
- les fonctions sont des tables de 10
- appelez chacune de ces fonction avec la valeur 4
- affiché les tables de 10 correspondante.

3. EstPair

Ecrire et tester une fonction de type `Return` qui, à partir d'un nombre entier passé en paramètre, renvoie un booléen égal à `true` si le nombre est pair et `false` sinon.

3.1. Correction

```
<!-- code HTML -->
<h1 id="titre">TPfunction - Pair</h1>
<label for="name">Un nombre entier ? :</label>
<input type="text" id="reponse" name="reponse" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable nombre
    let reponse = document.getElementById("reponse").value;
    //Affichage du message dans la zone paragraphe id message

    document.getElementById("message").innerHTML = "Nombre " + EstPair(reponse);
  }

  function EstPair(nombre){

    if (nombre%2 == 0)
    {
      message = "pair";
    }
    else
    {
      message = "impaire";
    }

    return message;
  }

  //Fin de code Javascript
</script>
```

Ecriture plus condensée avec une ternaire `a == b ? True : False;`

```
function EstPair(nombre){

  //avec une ternaire
  return nombre%2 == 0 ? "pair" : "impaire";

}
```

4. Mention

Ecrire une fonction Mention qui va renvoyer la mention associée à une note donnée en paramètre.

Règles d'attribution des mentions :

- 16 ou plus : Très Bien
- ≥ 14 et < 16 : Bien
- ≥ 12 et < 14 : Assez Bien
- ≥ 10 et < 12 : Passable
- inférieur à 10 : Echec

Vous appellerez cette fonction à partir d'un tableau de notes à tester.

4.1. Correction

```
<label for="name">Saisir une note :</label>
<input type="text" id="note">
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>

    //Porté en dehors de la fonction message
    let notes = [12, 14, 08, 13, 12];
    document.getElementById("button").addEventListener("click",message);

    //Fonction
    function message() {
        //variables
        let note = Number(document.getElementById("note").value);
        let moyenne = 0;

        if(note >= 0 && note <= 20){
            notes.push(note);
        }

        //Calculer la moyenne
        notes.forEach(function(note, index, array) {

            document.getElementById("message").innerHTML += "Note[" + index + "] : " + note + " - Mention " + Mention(note) + "\n";
            moyenne += note;

        });

        //longueur du tableau notes.length
        moyenne = moyenne / notes.length;

        //Afficher le résultat
        document.getElementById("message").innerHTML += "La moyenne est : " + Math.round(moyenne) + "\n";
    }

    function Mention(note){

        let mention = "Echec";

        switch (true){

            case (note > 16):
                mention = "Très Bien";
                break;
            case (note >= 14):
                mention = "Bien";
                break;
            case (note >= 12):
                mention = "Assez Bien";
                break;
            case (note >= 10):
                mention = "Passable";
                break;

        }

        return mention;

    }

</script>
```

Avec une ternaire :

```
function Mention(note){

    return note > 16 ? "Très Bien" : note >= 14 ? "Bien" : note >= 12 ? "Assez Bien" : note >= 10 ? "Passable" : "Echec";

}
```

5. Distance

Ecrire et tester une fonction qui calcule la distance AB entre 2 points A (x1,y1) et B (x2,y2).

Vous vérifierez notamment que la distance entre A et B est la même que la distance entre B et A.

Rappel de la formule :

La distance entre les points $A(x_1, y_1)$ et $B(x_2, y_2)$ est donnée par la formule suivante :

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



[Math.Sqrt](#) calcule la racine carrée d'un nombre

5.1. Correction

```
<!-- Un élément de paragraphe HTML destiné à restituer les résultats -->
<pre id="message"></pre>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>

    let x1 = 0;
    let y1 = 1;
    let x2= -2;
    let y2 = 4;

    document.getElementById("message").innerHTML += "Distance entre les points A(" + x1 + "," + y1 + ") et B(" + x2 + "," + y2 + ") = " +
Distance(x1, y1, x2, y2) + "\n";
    document.getElementById("message").innerHTML += "Distance entre les points O(0,0) et C(1,1) = " + Distance(0, 0, 1, 1) + "\n";

    function Distance (xA, yA, xB, yB){

        return Math.sqrt(Math.pow(xB - xA, 2) + Math.pow(yB - yA, 2));

    }

</script>
```

6. OuiNon

Nous souhaitons réaliser un questionnaire :

La question doit être paramétrable (Ajout de question, Utiliser un tableau de question).

- Etape 1 : Proposer une saisie de questions avec un formulaire HTML
- Etape 2 : Créer un bouton lancer questionnaire
- Etape 3 : Enregistrer les réponses OUI/NON

Ecrire une fonction demandant à l'utilisateur d'entrer une réponse de type Oui/Non.

La fonction centralise la question "Oui/Non ?" et renverra la réponse qui sera forcément Oui ou Non.

- Etape 4 : Enregistrer les réponses dans un Tableau
- Etape 5 : Proposer un bouton Résultat, qui affichera le questionnaire et les réponses

Rappel : Utilisez la commande prompt pour OUI/NON, voir [TP_PlusouMoins](#).

6.1. Aide

Votre interface doit ressembler à ceci :

Ajouter question :

Liste des questions :

Question 1 - Etes-vous marié ?

Question 2 - Avez-vous des enfants ?

Question 3 - Avez vous le permis ?

Question 4 - Avez vous 18 ans ?

Lancer questionnaire :

Résultats du questionnaire :

Question 1 - Réponse : Non

Question 2 - Réponse : Non

Question 3 - Réponse : Non

Question 4 - Réponse : Oui

Vous devez réaliser un minimum de 5 fonctions :

-
-
-
-
-

6.2. Correction

```
<h1>TPF05 - Questionnaire</h1>
<label>Ajouter question :</label>
<input type="text" id="question">
<button id="ajoutQuestion">Valider</button>
<button id="lancerQuestion">Lancer questionnaire</button>

<pre id="message"></pre>
<script>

    //initialisation des variables tableau
    let questions = new Array();
    let reponses = new Array();

    //Evenement de type click sur bouton HTML
    document.getElementById("ajoutQuestion").addEventListener("click", ajoutQuestion);
    document.getElementById("lancerQuestion").addEventListener("click", lancerQuestion);

    //permet d'ajouter la question dans le tableau questions
    function ajoutQuestion() {

        let question = document.getElementById("question").value;
        questions.push(question);
        message(`Question ${questions.length} ajouté.`);

    }

    //parcour les questions de notre tableau questions
    function lancerQuestion() {

        questions.forEach(function (value, index, array) {
            //reponses.push(ouiNon(value));
            //solution plus secure si question ne suit pas l'ordre de l'index
            reponses[index] = ouiNon(value);
        });

        resultat();

    }

    /**
     * Boucle tant que reponsse n'est pas oui OU non
     * @param question, issu du tableau questions
     * @returns {string} la réponse oui/non
     */
    function ouiNon(question) {

        let reponse;
        do {

            reponse = prompt(question + "(Oui/Non)");

        } while (reponse.toLowerCase() != "oui" && reponse.toLowerCase() != "non");

        return reponse;

    }

    function resultat(){

        message("Résultat :");
        questions.forEach(function(question, index){

            message(question);
            message(reponses[index]);

        });

    }

    //restituer l'information
    function message(msg) {

        document.getElementById("message").innerHTML += msg+"\n";

    }

</script>
```

7. Conversion en binaire

Ecrire et tester une fonction qui convertit un nombre décimal en binaire.

Exemples :

- 2 --> 10
- 8 --> 1000
- 9 --> 1001
- 36 --> 100100

La conversion d'un nombre décimal en binaire est réalisée par une série de divisions entières par 2
Pour obtenir le résultat, on prend chaque reste en partant de la dernière division

Exemple :
// 36 | 2
// 0 | 18 | 2
// 0 | 9 | 2
// 1 | 4 | 2
// 0 | 2 | 2
// 0 | 1 | 2
// 1 | 0

Ici vous allez manipuler des nombre et des strings. Vous devrez utiliser [parseInt\(\)](#).

Ensuite, écrire et tester une fonction qui transforme un nombre binaire en décimal.

Remarque : Dans cet exercice vous ne devez pas utiliser une variable de porté global (hors de la fonction).

Il est possible de parcourir un string comme un tableau :

```
//let i = '36';  
  
i[0] //3  
  
i[1] //6
```

[En savoir plus sur conversion Binaire to decimal.](#)

Votre programme doit appeler les fonctions sous la forme anonyme :

Ajouter nombre :

Binary To Decimal

Decimal To Binary

```
//Mode 1 = BinaryToDecimal ; Mode 2 = DecimalToBinary ;  
document.getElementById("binaire").addEventListener("click", function(){Resultat(1)});  
document.getElementById("decimal").addEventListener("click", function(){Resultat(2)});
```

7.1. decimalToBinary

```
<h1>TPF06 - Binaire</h1>
<label>Ajouter Nombre :</label>
<input type="text" id="nombre">
<button id="decimalToBinary">Decimal To Binary</button>
<button id="binaryToDecimal">Binary To Decimal</button>
<pre id="resultat"></pre>
<script>

document.getElementById("decimalToBinary").addEventListener("click", function(){resultat(1)});
document.getElementById("binaryToDecimal").addEventListener("click", function(){resultat(2)});

function resultat(mode) {

    let nb1 = document.getElementById("nombre").value;
    let nb2 = mode == 1 ? decimalToBinary(nb1) : binaryToDecimal(nb1);

    document.getElementById("resultat").innerHTML += `${nb1} = ${nb2}`;
}

function decimalToBinary(nombre){

    let reste;
    let resultat = "";
    do{
        debugger;
        reste = nombre % 2;
        resultat = reste + resultat;
        nombre = parseInt(nombre / 2);

    }while(nombre != 0);

    return resultat;
}

</script>
```

7.2. Correction

```
<!-- code HTML -->
<label>Ajouter nombre :</label>
<input type="text" id="number">
<button id="binaire">Binary To Decimal</button>
<button id="decimal">Decimal To Binary</button>
<pre id="resultat"></pre>

<script>

//Mode 1 = BinaryToDecimal ; Mode 2 = DecimalToBinary ;
document.getElementById("binaire").addEventListener("click", function(){Resultat(1)});
document.getElementById("decimal").addEventListener("click", function(){Resultat(2)});

function Resultat(mode){

    let nb1 = document.getElementById("number").value;
    let nb2 = mode == 1 ? BinaryToDecimal(nb1) : DecimalToBinary(nb1);

    document.getElementById("resultat").innerHTML += nb1 + " = " + nb2 + "\n";

}

//Chaque bit est élevé à la puissance de 2 qui correspond à son rang
function BinaryToDecimal(nb1)
{
    let resultat = 0;
    let exposant = 0;
    for (let i = nb1.length - 1; i >= 0; i--)
    {
        if (nb1[i] == '1')
        {
            resultat += Math.pow(2, exposant); //resultat = resultat + Math.Pow(2, exposant)
        }
        exposant++;
    }
    return resultat;
}

function DecimalToBinary(nb1)
{
    let resultat = "";
    let reste = 0;
    do
    {
        reste = nb1 % 2;
        resultat = reste + resultat;
        nb1 = parseInt(nb1 / 2);
    }
    while (nb1 != 0);

    return resultat;
}

</script>
```

8. Inversion de chaine

Ecrire et tester une fonction `Inverse` qui prend en paramètre une chaine de caractère et renvoie son inverse.

Exemple :

WINDOWS --> SWODNIW
KAYAK --> KAYAK

Bonus :

Ecrire et tester une fonction `EstPalindrome` qui renvoie un booléen égal à `true` si la chaine est un palindromme.

[C'est quoi un palindrome ?](#)

8.1. Correction

```
<!-- code HTML -->
<label>Ajouter un mot :</label>
<input type="text" id="mot">
<button id="inverser">Inverser</button>
<pre id="resultat"></pre>

<script>

  //Boutton Evenement
  document.getElementById("inverser").addEventListener("click", function(){

    let mot = document.getElementById("mot").value;

    document.getElementById("resultat").innerHTML += Inverser(mot) + "\n";

    if (EstPalindrome(mot))
    {
      document.getElementById("resultat").innerHTML += mot + " est un palindrome de " + mot.length + " lettres";
    }
  });

  function Inverser(mot){

    let resultat = "";
    for (let i = mot.length - 1; i >=0 ; i--)
    {
      resultat = resultat + mot[i];
    }
    return resultat;

  }

  //Une chaine est un palindrome si est égale à la chaine inverse
  //Exemple : KAYAK
  function EstPalindrome(mot)
  {
    return mot == Inverser(mot);
  }

</script>
```

9. Bissextile

Ecrire une fonction qui renverra un booléen pour indiquer si l'année d'une date passée en paramètre est bissextile ou non.

Une année est bissextile si elle est divisible par 4 et non divisible par 100 ou si elle est divisible par 400.

Exemple : 2000 et 2020 sont bissextiles. 1900 et 2005 ne le sont pas.

Vous testerez cette fonction en demandant à l'utilisateur d'entrée son année de naissance par exemple.

9.1. Correction

```
<!-- code HTML -->
<label>Ajouter une année :</label>
<input type="text" id="annee" value="">
<button id="bissextile">Bissextile</button>
<pre id="resultat"></pre>

<script>

  //Bouton Evenement
  document.getElementById("bissextile").addEventListener("click", function(){

    let annee = document.getElementById("annee").value;
    //On passe le message en majuscule - .toUpperCase()
    document.getElementById("resultat").innerHTML += "L'année " + annee + (Bissextile(annee) ? " est " : " n'est pas ") +
    "Bissextile\n";

  });

  function Bissextile(annee){

    let resultat = false;

    if ((annee % 4 === 0 && annee % 100 > 0) || (annee % 400 === 0)) {
      resultat = true;
    }

    return resultat;

  }

</script>
```

10. Compression RLE

Ecrire et tester une fonction de compression.

Run-Length Encoding/**RLE**) est un algorithme de compression de données sans perte qui repose sur l'idée de comprimer des plages de valeurs identiques en signalant le nombre de fois qu'une valeur donnée devrait être répétée.

On remplace plusieurs caractères consécutifs de la manière suivante :

- AAAaa --> A3a2
- ABC --> A1B1C1
- AAAAABBBCCddd --> A5B3C2d3

Rappel : Une chaîne de caractère est considérée comme un tableau.

```
let mot = "abcd";

for (let i = 0; i < mot.length ; i++) {
  console.log(mot[i]);
}
```

10.1. Correction

```
<!-- code HTML -->
<label>Ajouter une chaine :</label>
<input type="text" id="mot" value="AAAaaBbbCcCC">
<button id="compresser">compresser</button>
<pre id="resultat"></pre>

<script>

  //Boutton Evenement
  document.getElementById("compresser").addEventListener("click", function(){

    document.getElementById("resultat").innerHTML += Compression(document.getElementById("mot").value) + "\n";

  });

  function Compression(mot){

    let resultat = "";
    let compteurLettre = 1;

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

      if(mot[i] == mot[i + 1]){
        compteurLettre++;
      }else{
        resultat += compteurLettre + mot[i];
        compteurLettre = 1;
      }

    }
    return resultat;
  }

</script>
```

11. Code césar

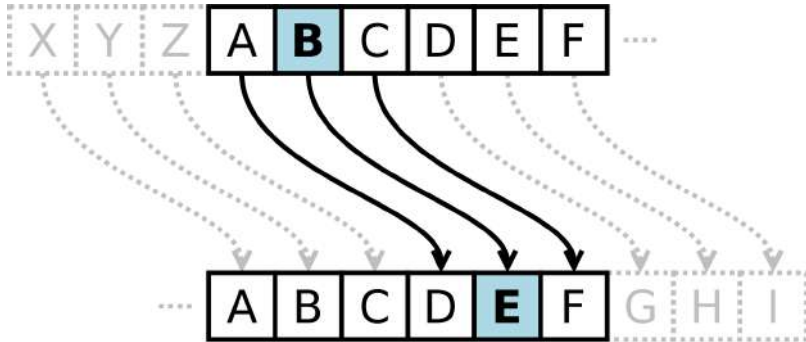
Principe

Le **chiffrement par décalage**, connu comme le **chiffre de César** ou le **code de César**, est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes.

Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières lettres (dans le cas d'un décalage à droite), on reprend au début.

Par exemple avec un décalage de 3 vers la droite, A est remplacé par D, B devient E, et ainsi jusqu'à w qui devient z, puis x devient A etc. Les caractères espace du texte en clair sont laissés tels quels dans le texte chiffré.

La longueur du **décalage**, 3 dans l'exemple évoqué, constitue **la clé du chiffrement** qu'il suffit de transmettre au destinataire pour que celui-ci puisse déchiffrer le message.



Ecrire et tester une méthode qui permet de chiffrer un message avec le code de César.

Vous devez prendre en considération Majuscule, Minuscule et caractère spéciaux comme "!? .#()"

Aidez vous des méthodes `var.toLowerCase()` ou `var.toUpperCase()`;



[Chiffrement vs cryptage : quelles différences ?](#)

11.1. Correction

```
<!-- code HTML -->
<label>Ajouter une chaine :</label>
<input type="text" id="message" value="Ceci est un message secret !">
<button id="chiffre">chiffre</button>
<pre id="resultat"></pre>

<script>

  //Bouton Evenement
  document.getElementById("chiffre").addEventListener("click", function(){

    let cle = 3;
    //On passe le message en majuscule - .toUpperCase()
    document.getElementById("resultat").innerHTML += CodeCesar(document.getElementById("message").value, cle) + "\n";

  });

  function CodeCesar(message, cle){

    let resultat = "";
    let alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    let special = "!? .#()";
    let reference = alphabet + special + alphabet.toLowerCase();

    //Parcours du message en clair pour le chiffrer
    for (let i = 0; i < message.length ; i++)
    {

      //Recherche la position de chaque caractère dans l'alphabet
      let j = 0;
      while (j < reference.length && message[i] != reference[j])
      {
        j++;
      }
      //Calcul de la position du caractère chiffré
      j = (j + cle);
      if (j >= reference.length)
      {
        j = j % reference.length;
      }
      if (j < 0)
      {
        j = reference.length + j;
      }
      resultat += reference[j];
    }
    return resultat;
  }

</script>
```

12. EstPremier

Etape 1

Ecrire et tester une fonction qui retournera un booléen pour indiquer si un nombre entier passé en paramètre est premier.

Enfin, vous afficherez tous les nombres premiers inférieurs à 100.



Un nombre est premier s'il n'a que 2 diviseurs : 1 et lui-même.

Etape 2

Ecrire et tester une fonction qui ajoute dans un tableau des nombres saisis par l'utilisateur. Lister ensuite l'ensemble de ces nombres et déterminer si ils sont premiers ou non.

12.1. Correction - Etape 1

```
//Affiche les nombres premiers inférieurs ou égal à 100
for (let i = 1; i <= 100; i++)
{
    if (EstPremier(i))
    {
        console.log(i);
    }
}

function EstPremier (nombre)
{
    premier = true;

    //Un nombre premier est divisible seulement par 1 et par lui-même
    let i = 2;
    while (i < nombre && premier) //arrêt de la boucle dès qu'on a un diviseur
    {
        if (nombre % i == 0)
        {
            premier = false;
        }
        i++;
    }

    return premier;
}
```

12.2. Correction - Etape 2

```
<h1>TPF12 - EstPremier</h1>
<label>Saisir un nombre</label>
<input id="nombre" type="text">
<button id="valider">Valider</button>
<button id="chercher">Chercher</button>
<pre id="message"></pre>
<script>

    let nombreTeste = new Array();

    document.getElementById("valider").addEventListener("click", ajout);
    document.getElementById("chercher").addEventListener("click", chercher);

    function ajout(){

        let nombre = document.getElementById("nombre").value;
        nombreTeste.push(nombre);
        afficher(`Nombre ${nombre} ajouté.`);

    }

    function chercher(){
        for (let i = 0; i < nombreTeste.length; i++)
        {
            if (EstPremier(nombreTeste[i]))
            {
                afficher(`Nombre ${nombreTeste[i]} est premier.`);
            }else{
                afficher(`Nombre ${nombreTeste[i]} n'est pas premier.`);
            }
        }
    }

    function EstPremier (nombre)
    {
        premier = true;

        //Un nombre premier est divisible seulement par 1 et par lui-même
        let i = 2;
        while (i < nombre && premier) //arrêt de la boucle dès qu'on a un diviseur
        {
            if (nombre % i == 0)
            {
                premier = false;
            }
            i++;
        }

        return premier;
    }

    function afficher(message){

        document.getElementById("message").innerHTML += message + '\n';

    }

</script>
```

13. Factorielle

La factorielle d'un nombre est le produit de tous les entiers compris entre 1 et lui-même. Il y'a trois façon pour trouver une factorielle d'un nombre donné, en utilisant la boucle for, la récursivité, ou en créant une fonction sur une plage allant de 1 à X(nombre entré par l'utilisateur). Exemple :

```
0! = 1
1! = 1
2! = 2 * 1
3! = 3 * 2 * 1
4! = 4 * 3 * 2 * 1
5! = 5 * 4 * 3 * 2 * 1
6! = 6 * 5 * 4 * 3 * 2 * 1
```

Etape 1 : Calculer la factorielle en utilisant la boucle for.

Etape 2 : Calculer la factorielle en utilisant la récursivité (la fonction s'appelle elle même.)

13.1. Solution

```
function fact(nbr){
  var i, nbr, f = 1;

  for(i = 1; i <= nbr; i++)
  {
    f = f * i;    // ou f *= i;
  }
  return f;
}

console.log(fact(3));
```

Fonction récursive :

```
function fact(nbr)
{
  // Si nbr = 0 la factorielle retournera 1
  if (nbr === 0)
  {
    return 1;
  }
  // appelez à nouveau la procédure récursive
  return nbr * fact(nbr-1);
}

console.log(fact(3));
```

Jeu de la vie

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Jeu de la vie

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:55

Description

Réaliser en Javascript, HTML, CSS le jeu de la vie

Table des matières

1. TP01 - Grille

1.1. Solution

2. TP02 - Event

2.1. Solution

3. TP03 - CSS et JS

3.1. Solution

4. TP04 - Reverse

4.1. Solution

5. TP05 - Array

5.1. Solution

6. TP06 - Algo

6.1. Solution

7. TP07 - Function

7.1. Solution

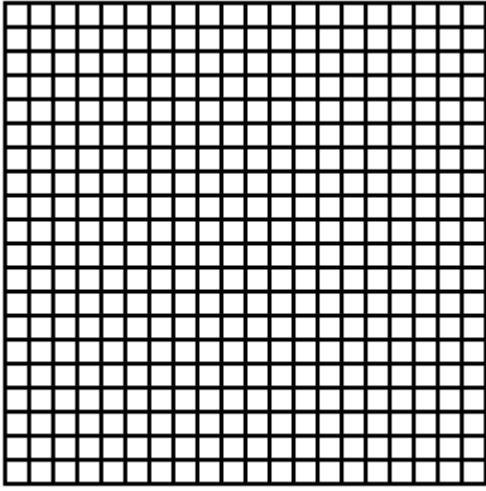
7.2. Solution 2

1. TP01 - Grille

Réaliser une grille de 10 x 10. On utilisera :

- 100 champs de type <div>id de 01 à 100, value 0 ou 1 (noir, blanc)
- réaliser le css pour former une grille

1.1. Solution



```
<style>
div{
  background-color: white;
  border: 1px solid black;
  width: 10px;
  height: 10px;
  display: inline-block;
}

#grille{

  border:1px solid black;
  height: 240px;
  width:240px;
  margin: 100px;

}
</style>
<div id="grille">

</div>

<script>
let grille = 20;
//horizontale
for (let i = 0; i < grille ; i++){
  //vertical
  for (let n = 0; n < grille ; n++) {
    document.getElementById("grille").innerHTML += "<div id='c"+ i + "," + n + "'></div>";
  }
}

</script>
```

2. TP02 - Event

Créer un bouton evenement de type click sur chaque cellule :

- la fonction appelé doit afficher avec `console.log()` les coordonnées de la cellule (x,y)

2.1. Solution

```
let grille = 20;
let id = 0;

//horizontale
for (let x = 0; x < grille ; x++){
  //vertical
  for (let y = 0; y < grille ; y++) {
    document.getElementById("grille").innerHTML += "<div class='cellule' id='"+ id + "' onclick=cellule(\"+x+\",\"+y+\")></div>";
    id++;
  }
}

function cellule(x, y){
  console.log("click");
  console.log(x + "_" + y);
}
```

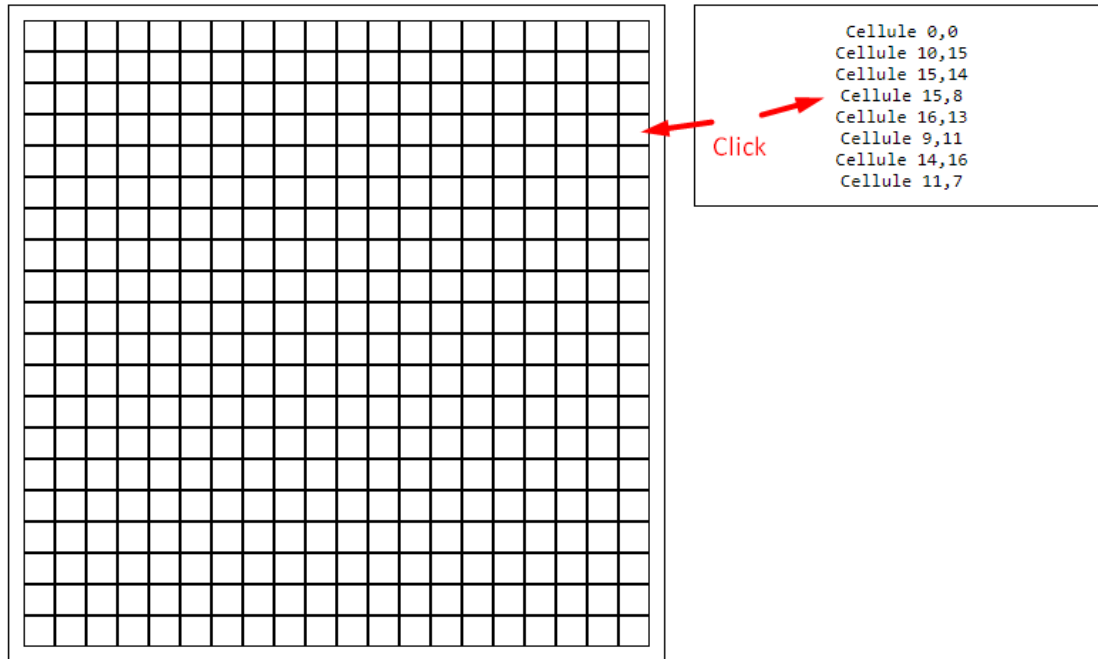
3. TP03 - CSS et JS

Nous souhaitons gérer notre grille css de manière dynamique lié avec les variables JS :

- le CSS doit être géré par la taille des cellules
- la taille de grid

Nous souhaitons afficher les résultats de notre fonction cellule dans un élément HTML (à droite de la grille).

Jeu de la vie en Javascript - DWWM01



3.1. Solution

```
<style>
  .cellule {
    background-color: white;
    border: 1px solid black;
    display: inline-block;
  }
  #container{
    width: 800px;
    text-align: center;
    margin: 0 auto;
  }
  #grid, #message {
    border: 1px solid black;
    padding: 10px;
    float: left;
    margin: 0 10px;
  }
  #message{
    width: 270px;
  }
</style>
<div id="container">
  <h2>Jeu de la vie en Javascript - DWWMM01</h2>
  <div id="grid">

  </div>
  <pre id="message"></pre>
</div>

<script>
  const CELBORDER = 2;
  let grid = 20;
  let id = 0;
  let celluleSize = 20;

  //CSS
  let gridCss = document.getElementById("grid");
  gridCss.style.width = (grid * (celluleSize + CELBORDER)).toString();
  gridCss.style.height = (grid * (celluleSize + CELBORDER)).toString();

  //génération de la grille
  // colonne
  for (let x = 0; x < grid; x++) {
    //ligne
    for (let y = 0; y < grid; y++) {
      document.getElementById("grid").innerHTML += `<div style="width:${celluleSize}px;height:${celluleSize}px" class="cellule"
id="${id}" onclick="cellule(${x},${y})"></div>`;
      id++;
    }
  }

  function cellule(x, y) {
    document.getElementById("message").innerHTML += "Cellule " + x + ", " + y + "\n";
  }
</script>
```

4. TP04 - Reverse

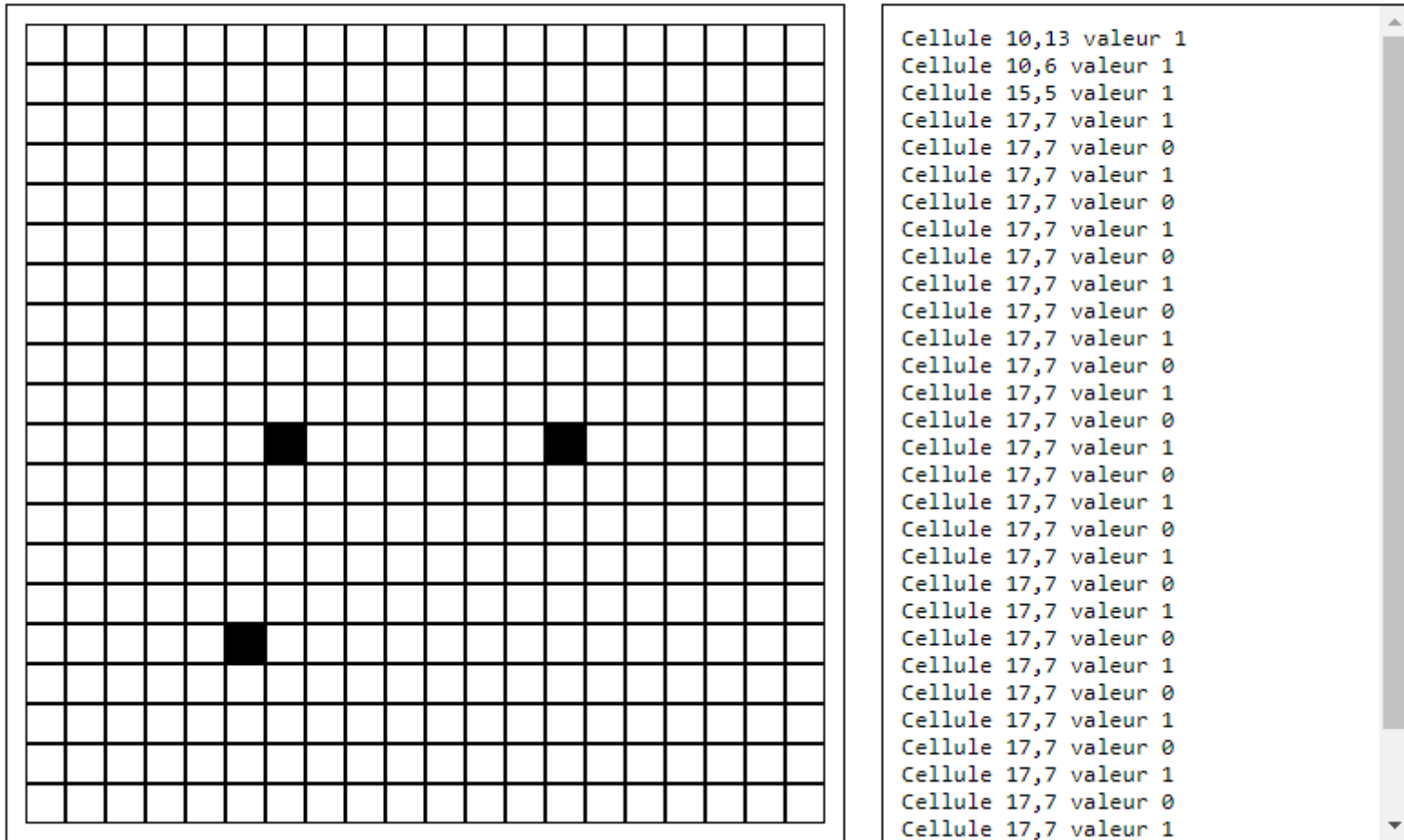
Nous souhaitons que lorsque la cellule est cliquée elle change de couleur (noir/blanc) :

- Utiliser une variable cellule active valeur 1 pour noir ou 0 pour blanc que vous afficherez dans la zone prévu.

Astuce : Vous pouvez pour changer de couleur utiliser une class 'black ou White' par exemple et utiliser les methodes JS :

`.classList.contains` et `.classList.replace`
<https://developer.mozilla.org/fr/docs/Web/API/Element/classList>

Jeu de la vie en Javascript - DWWM01



4.1. Solution

```
<style>
  .cellule {
    background-color: white;
    border: 1px solid black;
    display: inline-block;
  }

  #container {
    width: 800px;
    text-align: center;
    margin: 0 auto;
  }

  #grid, #message {
    border: 1px solid black;
    padding: 10px;
    float: left;
    margin: 0 10px;
    text-align: left;
  }

  #message {
    width: 270px;
    overflow: auto;
  }

  .white {
    background-color: white;
  }

  .black {
    background-color: black;
  }
</style>
<div id="container">
  <h2>Jeu de la vie en Javascript - DWM01</h2>
  <div id="grid">

    </div>
    <pre id="message"></pre>
</div>

<script>
  //Taille border cellule 2px
  const CELBORDER = 2;
  //Grille de 20 x 20
  let grid = 20;
  //Taille cellule 20px * 20px
  let celluleSize = 20;
  let id = 0;

  //Adapter CSS
  let gridCss = document.getElementById("grid");
  let messageCss = document.getElementById("message").style.height = (grid * (celluleSize + CELBORDER)).toString();
  gridCss.style.width = (grid * (celluleSize + CELBORDER)).toString();
  gridCss.style.height = (grid * (celluleSize + CELBORDER)).toString();

  //génération de la grille, double boucle
  // colonne
  for (let x = 0; x < grid; x++) {
    //ligne
    for (let y = 0; y < grid; y++) {
      document.getElementById("grid").innerHTML += `<div style="width:${celluleSize}px;height:${celluleSize}px" class="cellule white"
id="${id}" onclick="cellule(${x},${y},${id})"></div>`;
      id++;
    }
  }

  /**
   * Function click cellule color White/Black
   * @param x number coord X
   * @param y number coord Y
   * @param id number
   */
  function cellule(x, y, id) {
    let value = 0;
    if (document.getElementById(id).classList.contains('white')) {
      document.getElementById(id).classList.replace("white", "black");
      value = 1;
    } else {
      document.getElementById(id).classList.replace("black", "white");
    }
    document.getElementById("message").innerHTML += `Cellule ${x},${y} valeur ${value} \n`;
  }
</script>
```


5. TP05 - Array

Vous devez stocker les caractéristiques des cellules sur un tableau. L'index du tableau se réfère à l'ID de la cellule et sa valeur sera 1 (noir, vivant) ou 0 (blanc, mort) :

- Créer une fonction `grille` pour la génération de la grille
- Générer un tableau et l'initialiser à la création des cellules
- Mettre à jour le tableau à l'index correspondant avec sa valeur sur la fonction `clickCellule`

5.1. Solution

```

<style>
    .cellule {
        background-color: white;
        border: 1px solid black;
        display: inline-block;
    }

    #container {
        width: 800px;
        text-align: center;
        margin: 0 auto;
    }

    #grid, #message {
        border: 1px solid black;
        padding: 10px;
        float: left;
        margin: 0 10px;
        text-align: left;
    }

    #message {
        width: 270px;
        overflow: auto;
    }

    .white {
        background-color: white;
    }

    .black {
        background-color: black;
    }
</style>
<div id="container">
    <h2>Jeu de la vie en Javascript - DWM01</h2>
    <div id="grid">

        </div>
        <pre id="message"></pre>
</div>

<script>
    //Taille border cellule 2px
    const CELBORDER = 2;
    //Grille de 20 x 20
    let grid = 20;
    //Taille cellule 20px * 20px
    let celluleSize = 20;
    let id = 0;
    let cellules = [];

    //Adapter CSS
    let gridCss = document.getElementById("grid");
    let messageCss = document.getElementById("message").style.height = (grid * (celluleSize + CELBORDER)).toString();
    gridCss.style.width = (grid * (celluleSize + CELBORDER)).toString();
    gridCss.style.height = (grid * (celluleSize + CELBORDER)).toString();

    //lancement des fonctions
    grille();

    function grille() {
        // colonne
        for (let x = 0; x < grid; x++) {
            //ligne
            for (let y = 0; y < grid; y++) {
                document.getElementById("grid").innerHTML += `<div style="width:${celluleSize}px;height:${celluleSize}px" class="cellule
white" id="${id}" onclick="clickCellule(${x},${y},${id})"></div>`;
                id++;
                cellules[id] = 0;
            }
        }
    }

    /**
     * Function click cellule color White/Black
     * @param x number coord X
     * @param y number coord Y
     * @param id number
     * @param {string} id
     */
    function clickCellule(x, y, id) {
        let value = 0;
        let cellule = document.getElementById(id).classList;
        if (cellule.contains('white')) {
            cellule.replace("white", "black");
            value = 1;
        } else {
            cellule.replace("black", "white");
        }
        document.getElementById("message").innerHTML += `Cellule ${x},${y} valeur ${value} \n`;
        cellules[id] = value;
    }

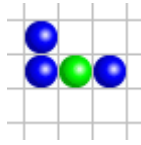
```

</script>

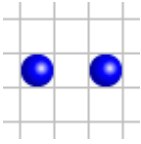
6. TP06 - Algo

Créer une fonction `jeuDelavie(id)` qui change la valeur de la cellule 1 (vivant) 0 (mort) en fonction des règles de gestion suivantes :

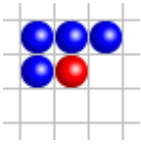
- si une cellule a exactement trois voisines vivantes, elle est vivante à l'étape suivante.



- si une cellule a exactement deux voisines vivantes, elle reste dans son état actuel à l'étape suivante.



- si une cellule a strictement moins de deux ou strictement plus de trois voisines vivantes, elle est morte à l'étape suivante.



Procédez aux étapes ci-dessous :

- Créer une fonction `jeuDelavie(id)` qui retourne dans le container message l'une des informations suivante :
 - Cette cellule a "exactement trois voisines vivantes"
 - Cette cellule a "exactement deux voisines vivantes"
 - Cette cellule a "strictement moins de deux ou strictement plus de trois voisines vivantes"
- La fonction `jeuDelavie(id)` est appelé à partir de la fonction `clickCellule`

6.1. Solution

```
/**
 *
 * @param id
 */
function jeuDelavie(id) {

    //  0   1   2
    //  3   x   4
    //  5   6   7
    let celTab = [];
    let voisine = 0;

    celTab[0] = id - grid - 1;
    celTab[1] = id - grid;
    celTab[2] = id - grid + 1;
    celTab[3] = id - 1;
    celTab[4] = id + 1;
    celTab[5] = id + grid - 1;
    celTab[6] = id + grid ;
    celTab[7] = id + grid + 1;

    //total voisine
    for (let c = 0; c < celTab.length ; c++){
        if((celTab[c] >= 0) && (celTab[c] <= grid * grid) && (cellules[celTab[c]] == 1)){
            voisine++;
        }
    }
    console.log(id + ">" + voisine);

    //si trois cellule voisines
    if(voisine == 3){
        cellulesTmp[id] = 1;
        //document.getElementById("message").innerHTML += "3 voisines\n";
        document.getElementById("message").innerHTML += "Cellule " + id + " update vivante\n";
        //sinon, si cellule inférieur à 2 ou supérieur à 3, alors meurt
    }else if(voisine < 2 || voisine > 3){
        cellulesTmp[id] = 0;
        //document.getElementById("message").innerHTML += "moins de 2 voisines et plus de 3 voisines\n";
        document.getElementById("message").innerHTML += "Cellule " + id + " update mort\n";
    }else{
        cellulesTmp[id] = cellules[id];
        document.getElementById("message").innerHTML += "Cellule " + id + " etat egal " + cellules[id] ? "vivant\n" : "mort\n";
    }
}
```

7. TP07 - Function

Adapter la fonction `grille()` pour permettre la gestion de trois types de mode. Utiliser un `Switch`:

- Initialisation de la grille
- Lancer `jeuDeLaVie()` pour chaque cellule
- Régénérer la grille en analysant chacune des cellules

Créer une fonction `run()` qui se lancera à partir d'un bouton et qui permettra le rafraichissement de la grille sur 30 itérations.

7.1. Solution


```

<style>
    .cellule {
        background-color: white;
        border: 1px solid black;
        display: inline-block;
    }

    #container {
        width: 800px;
        text-align: center;
        margin: 0 auto;
    }

    #grid, #message {
        border: 1px solid black;
        padding: 10px;
        float: left;
        margin: 0 10px;
        text-align: left;
    }

    #message {
        width: 270px;
        overflow: auto;
    }

    .white {
        background-color: white;
    }

    .black {
        background-color: black;
    }
</style>
<div id="container">
    <h2>Jeu de la vie en Javascript - DWWMM01</h2>
    <div id="grid">

        </div>
        <pre id="message"></pre>
</div>
<button id="button">Run</button>

<script>
    //Taille border cellule 2px
    const CELBORDER = 2;
    //Grille de 20 x 20
    let grid = 20;
    //Taille cellule 20px * 20px
    let celluleSize = 20;
    let cellules = [];
    let cellulesTmp = [];

    //Adapter CSS
    let gridCss = document.getElementById("grid");
    let messageCss = document.getElementById("message").style.height = (grid * (celluleSize + CELBORDER)).toString();
    gridCss.style.width = (grid * (celluleSize + CELBORDER)).toString();
    gridCss.style.height = (grid * (celluleSize + CELBORDER)).toString();

    //event Button
    document.getElementById("button").addEventListener("click",run);

    //lancement des fonctions
    grille(1);

    /**
     * Fonction génération grille
     * @param mode number - 1 init, 2 jeu de la vie, 3 update,
     */
    function grille(mode) {
        let id = 0;
        // colonne
        for (let x = 0; x < grid; x++) {
            //ligne
            for (let y = 0; y < grid; y++) {
                switch (mode) {
                    case 1:
                        document.getElementById("grid").innerHTML += `<div style="width:${celluleSize}px;height:${celluleSize}px"
class="cellule white" id="${id}" onclick="clickCellule(${x},${y},${id})"></div>`;
                        cellules[id] = 0;
                        cellulesTmp[id] = 0;
                        break;
                    case 2:
                        jeuDelavie(id);
                        break;
                    case 3:
                        cellules[id] = cellulesTmp[id];
                        if(cellules[id] == 0){
                            document.getElementById(id).classList.replace("black", "white");
                        }else{
                            document.getElementById(id).classList.replace("white", "black");
                        }
                        break;
                }
            }
        }
    }

```

```

    }
    id++;
  }
}

/**
 * Function click cellule color White/Black
 * @param x number coord X
 * @param y number coord Y
 * @param id number
 */
function clickCellule(x, y, id) {
  let value = 0;
  let cellule = document.getElementById(id).classList;
  if (cellule.contains('white')) {
    cellule.replace("white", "black");
    value = 1;
  } else {
    cellule.replace("black", "white");
  }
  document.getElementById("message").innerHTML += `Cellule ${x},${y} valeur ${value} (${id}) \n`;
  cellules[id] = value;
  //jeuDelavie(id);
}

/**
 *
 * @param id
 */
function jeuDelavie(id) {

  //  0   1   2
  //  3   x   4
  //  5   6   7
  let celTab = [];
  let voisine = 0;

  celTab[0] = id - grid - 1;
  celTab[1] = id - grid;
  celTab[2] = id - grid + 1;
  celTab[3] = id - 1;
  celTab[4] = id + 1;
  celTab[5] = id + grid - 1;
  celTab[6] = id + grid ;
  celTab[7] = id + grid + 1;

  //total voisine
  for (let c = 0; c < celTab.length ; c++){
    if((celTab[c] >= 0) && (celTab[c] <= grid * grid) && (cellules[celTab[c]] == 1)){
      voisine++;
    }
  }
  console.log(id + ">" + voisine);

  //si trois cellule voisines
  if(voisine == 3){
    cellulesTmp[id] = 1;
    //document.getElementById("message").innerHTML += "3 voisines\n";
    document.getElementById("message").innerHTML += "Cellule " + id + " update vivante\n";
    //sinon, si cellule inférieur à 2 ou supérieur à 3, alors meurt
  }else if(voisine < 2 || voisine > 3){
    cellulesTmp[id] = 0;
    //document.getElementById("message").innerHTML += "moins de 2 voisines et plus de 3 voisines\n";
    document.getElementById("message").innerHTML += "Cellule " + id + " update mort\n";
  }else{
    cellulesTmp[id] = cellules[id];
    document.getElementById("message").innerHTML += "Cellule " + id + " etat egal " + cellules[id] ? "vivant\n" : "mort\n";
  }
}

function run() {
  console.log("run");

  for (let i = 0; i < 10; i++){

    //timer pour actualiser l'affichage
    setTimeout(function() {
      grille(2);
      grille(3);
      document.getElementById("message").innerHTML += `Iteration ${i}\n`;
    }, 1000);

  }
}

```

</script>

7.2. Solution 2

```
<canvas id="gameCanvas" class="grid" onclick="gridClick(event)">
</canvas>
```

```
<p id="debug"></p>
```

```
<script>
```

```
/**
 *
 * @type {HTMLCanvasElement}
 */
let canvas = document.getElementById("gameCanvas");
let ctx = canvas.getContext("2d");
let sizeOfCells = 15;
let sizeOfGridSides = 150;
let length = sizeOfCells * sizeOfGridSides;
let gridArray = [];
canvas.style.border = "solid black 2px"

canvas.style.height = length;
canvas.style.width = length;
canvas.height = length;
canvas.width = length;

let paused = true;
let nextStep = new Map();
drawGridLines();

document.addEventListener("keydown", onKeyDown)
update();

async function update(){
  while(true)
  {
    nextStep.clear();
    await new Promise(r => setTimeout(r, 100));

    if(paused) continue;
    else{

      for(let i = 0; i<gridArray.length;i++)
      {
        var sant = getIfStillAliveNextTurn(i);
        if(gridArray[i] != sant){
          nextStep.set(i,sant);
        }
      }

      for(const x of nextStep){
        setCell(x[0],x[1]);
      }

    }
  }
}

/**
 *
 * @param id : number
 * @returns {number}
 */
function getIfStillAliveNextTurn(id){
  let neighs = getNeighbours(id);
  let amount = 0;
  for(const n of neighs){
    if(n==1)continue;
    if(gridArray[n]){
      amount++;
    }
  }
  if(amount == 3)return true;
  else if (amount == 2)return gridArray[id];
  else return false;
}

/**
 *
 * @param id : number
 * @returns {(number)[[]]}
 */
function getNeighbours(id){
  let s = sizeOfGridSides;
  let val = [id-(s+1),id-s,id-(s-1),
    id-1, -1, id+1,
    id+(s-1),id+s,id+s+1];
  if(id%sizeOfGridSides==0){
    val[0] = -1;
    val[3] = -1;
    val[6] = -1;
  }
  if(id%sizeOfGridSides == sizeOfGridSides-1){
    val[2] = -1;
    val[5] = -1;
  }
}
```

```

        val[8] = -1;
    }
    if(parseInt(id/sizeOfGridSides) == 0){
        val[0] = -1;
        val[1]=-1;
        val[2] = -1;
    }
    if(parseInt(id/sizeOfGridSides) == sizeOfGridSides-1){
        val[6] = -1;
        val[7]=-1;
        val[8] = -1;
    }
    return val;
}

function onKeyDown(e) {
    if (e.code == "Enter") {
        paused = !paused;
        document.getElementById("debug").innerHTML = paused;
    }
}

/**
 *
 * @param event : PointerEvent
 */
function gridClick(event){

    let intX = parseInt(event.offsetX / sizeOfCells);
    let intY = parseInt(event.offsetY / sizeOfCells);
    let cellID = intX + intY * sizeOfGridSides;

    let cellVal = gridArray[cellID];
    if(cellVal){
        setCell(cellID,false);
    }
    else{
        setCell(cellID,true);
    }
    document.getElementById("debug").innerHTML = `id:${cellID}`
}

/**
 *
 * @param id : number
 * @param bool : bool
 */
function setCell(id,bool){

    id = parseInt(id);
    gridArray[id] = bool;
    let intY = parseInt(id/sizeOfGridSides);
    let intX = parseInt(id%sizeOfGridSides);

    let posX = intX * sizeOfCells +1;
    let posY = intY * sizeOfCells +1;
    if(bool){
        ctx.clearRect(posX,posY,sizeOfCells-2,sizeOfCells-2)
    }
    else{
        ctx.fillRect(posX,posY,sizeOfCells-2,sizeOfCells-2)
    }
}

function drawGridLines(){
    ctx.fillRect(0,0,length,length)

    for(let i = 0;i<sizeOfGridSides -1;i++){
        let pos= (sizeOfCells * i) +(sizeOfCells-1);
        ctx.clearRect(pos,0,2,length);
        ctx.clearRect(0,pos,length,2);
    }
    for(let i=0;i<sizeOfGridSides*sizeOfGridSides;i++){
        gridArray.push(false);
    }
}
}
</script>

```

Taquin

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Taquin

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:56

Table des matières

1. Apparence du jeu

1.1. Solution

2. Echange de case

2.1. Solution Intermediaire

2.2. Solution

3. Case valide

3.1. Solution

4. Random

4.1. Solution

5. Optimiser

5.1. Solution

6. Afficher la solution

6.1. Solution

7. Utiliser une image

7.1. Solution

8. Ajouter un niveau

8.1. Solution

1. Apparence du jeu

Créer l'apparence du jeu : quatre cases par quatre cases, chaque case étant un bouton HTML.

Aidez vous des valeurs suivantes que vous stockerez dans un tableau générer le taquin :

1, 11 , 8, 5, 7, 10, 4, 6, 13, 9, 2, 14, 15, 12, 3.

Générer l'ensemble des cellules avec un id sous la forme case_ligne_colonne.

Pour afficher un button avec pas de valeurs , utilisé l'espace insécable HTML

Obtenir le résultat ci-dessous avec du CSS :

Générateur de Takin - DWWM01

1	11	8	5
7	10	4	6
13	9	2	14
15	12	3	

1.1. Solution

```
<style>
  .cellule {
    background-color: aliceblue;
    border: 1px solid lightgray;
    display: inline-block;
    width: 100px;
    height: 100px;
    line-height: 100px;
    text-align: center;
    font-size: 1em;
  }

  .white{

    background-color: white;

  }

  #container {
    width: 400px;
    text-align: center;
    margin: 0 auto;
  }

  #grid {
    border: 4px solid black;
    float: left;
    margin: 0 10px;
    text-align: left;
    width: 400px;
    height: 400px;
  }

</style>
<div id="container">
  <h2>Générateur de Takin - DWWM01</h2>
  <div id="grid">

    </div>
</div>
<script>

let grid = 4;
let takin = [1, 11, 8, 5, 7, 10, 4, 6, 13, 9, 2, 14, 15, 12, 3];
grille();

function grille() {
  let id = 0;
  //générer 9 carrés div, lignes et colonnes
  for (let x = 1; x <= grid; x++) {

    for (let y = 1; y <= grid; y++) {
      //créer cellule
      let cel = document.createElement('button');
      cel.setAttribute("id", "case_" + x + "_" + y);
      cel.innerHTML = takin[id] ? takin[id] : "&nbsp;";
      cel.classList.add("cellule");
      if(!takin[id])
        cel.classList.add("white");

      //ajoute l'ensemble à grid
      document.getElementById("grid").appendChild(cel);
      id++;
    }
  }
}

</script>
```

2. Echange de case

Écrire le code JavaScript qui, lorsque l'on clique sur un bouton, l'échange avec la case vide. Pour réaliser cela, vous devez travailler par étape :

- Créer une fonction move (lig, col)
- Appeler cette fonction lors d'un click cellule () avec l'attribut HTML

```
onclick
```

- Stocker les coordonnées de la case vide pour l'intervertir avec les coordonnées de la cellule clickée
- Faire en sorte que la case blanche ne soit pas clickable

Générateur de Takin - DWWM01

1	11	8	5
10		4	6
7	13	3	14
15	12	2	9

Diagram illustrating a 4x4 grid state for the Takin puzzle. The grid contains numbers 1 through 15, with one empty cell (white) at row 2, column 2. Red arrows indicate a click on the cell containing 13 (row 3, column 2), and the word "click" is written below it.

L'idée est d'intervertir la valeur des deux cellules Uniquement (...et la class pour que bleu devient blanc...).

Vous pouvez utiliser la syntaxe suivante pour alterner les valeurs :

```
let bvalue = bnode.removeChild(bnode.childNodes[0]);
let evalue = enode.removeChild(enode.childNodes[0]);
/* on échange ces fils */
bnode.appendChild(evalue);
enode.appendChild(bvalue);
```

2.1. Solution Intermediaire

```
let bvalue = bnode.removeChild(bnode.childNodes[0]);
let evalue = enode.removeChild(enode.childNodes[0]);
/* on échange ces fils */
bnode.appendChild(evalue);
enode.appendChild(bvalue);
/* on échange les classes des deux boutons */
bnode.setAttribute('class','cellule white');
enode.setAttribute('class','cellule');
```

2.2. Solution

```

<style>
    .cellule {
        background-color: aliceblue;
        border: 1px solid lightgray;
        display: inline-block;
        width: 100px;
        height: 100px;
        line-height: 100px;
        text-align: center;
        font-size: 1em;
    }

    .white{

        background-color: white;

    }

    .actif{
        background-color: lightblue;
    }

    #container {
        width: 400px;
        text-align: center;
        margin: 0 auto;
    }

    #grid {
        border: 4px solid black;
        float: left;
        margin: 0 10px;
        text-align: left;
        width: 400px;
        height: 400px;
    }

</style>
<div id="container">
    <h2>Générateur de Takin - DWWM01</h2>
    <div id="grid">

        </div>
</div>
<script>

let grid = 4;
let takin = [1, 11, 8, 5, 7, 10, 4, 6, 13, 9, 2, 14, 15, 12, 3];
/* on mémorise l'emplacement de la case vide */
let emptyLig = 4;
let emptyCol = 4;

grille();

function grille() {
    let id = 0;
    //générer 9 carrés div, lignes et colonnes
    for (let x = 1; x <= grid; x++) {

        for (let y = 1; y <= grid; y++) {
            //créer cellule
            let cel = document.createElement('button');
            cel.setAttribute("id", "case_" + x + "_" + y);
            cel.innerHTML = takin[id] ? takin[id] : "&nbsp;";
            cel.classList.add("cellule");
            //ajouter la fonction
            cel.setAttribute("onclick", `move(${x},${y})`);
            if(!takin[id]) {
                cel.classList.add("white");
            }

            //ajoute l'ensemble à grid
            document.getElementById("grid").appendChild(cel);
            id++;
        }

    }
}

function move (lig,col) {
    /* on récupère les identifiants des deux boutons à intervertir */
    let bname = `case_${lig}_${col}`;
    let ename = `case_${emptyLig}_${emptyCol}`;
    /* on récupère les noeuds correspondant à ces boutons */
    let bnode = document.getElementById(bname);
    let enode = document.getElementById(ename);
    //ne fait rien si cellule blanche
    if(!bnode.classList.contains('white')){
        /* on récupère les fils textuels des deux boutons */
        let bvalue = bnode.removeChild(bnode.childNodes[0]);
        let evalue = enode.removeChild(enode.childNodes[0]);
        /* on échange ces fils */

```

```
        bnode.appendChild(evalue);
        enode.appendChild(bvalue);
        /* on échange les classes des deux boutons */
        bnode.setAttribute('class','cellule white');
        enode.setAttribute('class','cellule');
        /* on mémorise l'emplacement de la case vide */
        emptyLig    = lig;
        emptyCol    = col;
    }
}
```

</script>

3. Case valide

Adapter votre programme pour n'autoriser que les déplacements valides :

Générateur de Takin - DWWM01

1	11	8	2
10	6	4	5
3		13	14
15	12	7	9

Compter et afficher le nombre de déplacements effectués.

3.1. Solution


```

<style>
    .cellule {
        background-color: aliceblue;
        border: 1px solid lightgray;
        display: inline-block;
        width: 100px;
        height: 100px;
        line-height: 100px;
        text-align: center;
        font-size: 1em;
    }

    .white{

        background-color: white;

    }

    .actif{
        background-color: lightblue;
    }

    #container {
        width: 400px;
        text-align: center;
        margin: 0 auto;
    }

    #grid {
        border: 4px solid black;
        float: left;
        margin: 0 10px;
        text-align: left;
        width: 400px;
        height: 400px;
    }

    #message{
        padding: 10px;
        float: left;
        text-align: center;
        width: 100%;
    }

</style>
<div id="container">
    <h2>Générateur de Takin - DWWM01</h2>
    <div id="grid">

    </div>
    <div id="message"></div>
</div>
<script>

let grid = 4;
let takin = [1, 11, 8, 5, 7, 10, 4, 6, 13, 9, 2, 14, 15, 12, 3];
/* on mémorise l'emplacement de la case vide */
let emptyLig = 4;
let emptyCol = 4;
//Nombre de coups
let nb = 0;

grille();

function grille() {
    let id = 0;

    //générer 9 carrés div, lignes et colonnes
    for (let x = 1; x <= grid; x++) {

        for (let y = 1; y <= grid; y++) {
            //créer cellule
            let cel = document.createElement('button');
            cel.setAttribute("id", "case_" + x + "_" + y);
            cel.innerHTML = takin[id] ? takin[id] : "&nbsp;";
            cel.classList.add("cellule");
            //ajouter la fonction
            cel.setAttribute("onclick", `move(${x},${y})`);
            if(!takin[id]) {
                cel.classList.add("white");
            }

            //ajoute l'ensemble à grid
            document.getElementById("grid").appendChild(cel);
            id++;
        }
    }
}

function move (lig,col) {
    /* on récupère les identifiants des deux boutons à intervertir */
    let bname = `case_${lig}_${col}`;
    let ename = `case_${emptyLig}_${emptyCol}`;

```

```
/* on récupère les noeuds correspondant à ces boutons */
let bnode = document.getElementById(bname);
let enode = document.getElementById(ename);
//ne fait rien si cellule blanche
//verifier si case valide
if (!bnode.classList.contains('white') &&
    ((emptyLig==lig) && ((col==emptyCol-1)|| (col==emptyCol+1)))
    || ((emptyCol==col) && ((lig==emptyLig-1)|| (lig==emptyLig+1)))
) {
    /* on récupère les fils textuels des deux boutons */
    let bvalue = bnode.removeChild(bnode.childNodes[0]);
    let evalue = enode.removeChild(enode.childNodes[0]);
    /* on échange ces fils */
    bnode.appendChild(evalue);
    enode.appendChild(bvalue);
    /* on échange les classes des deux boutons */
    bnode.setAttribute('class','cellule white');
    enode.setAttribute('class','cellule');
    /* on mémorise l'emplacement de la case vide */
    emptyLig = lig;
    emptyCol = col;
    nb++;
    document.getElementById("message").innerHTML = `Vous avez réalisé ${nb} coups`;
}
}
```

</script>

4. Random

- Mélanger aléatoirement les cases au début du jeu.
 - créer un bouton pour générer la grille
- Détecter la fin de partie et afficher "Vous avez gagné"

Pour mélanger un tableau vous devrez partir d'une solution resolvable par exemple du tableau suivant :

```
let takin = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, "V"];
```

Créer un algorithme qui permet de permuter la case V (vide), sachant qu'elle ne peut se déplacer :

- soit a gauche si c'est possible (bordure)
- soit à droite si c'est possible (bordure)
- soit en haut si c'est possible (bordure) et permuter avec la valeur -4 (ligne du haut))
- soit en bas si c'est possible (bordure) et permuter avec la valeur +4 (ligne du bas))

Après avoir générer les permutations selon une itération random ou de votre choix, indiquer en combien de permutation ce taquin est resolvable.

Exemple :

Générateur de Takin - DWWM01

10	1	2	12
4	14	6	5
3	8	7	13
9	11		15

Takin resolvable maximum 83 coups.

Générer grille

4.1. Solution

```

<style>
  .cellule {
    background-color: aliceblue;
    border: 1px solid lightgray;
    display: inline-block;
    width: 100px;
    height: 100px;
    line-height: 100px;
    text-align: center;
    font-size: 1em;
  }

  .white{

    background-color: white;

  }

  #container {
    width: 400px;
    text-align: center;
    margin: 0 auto;
  }

  #grid {
    border: 4px solid black;
    float: left;
    margin: 0 10px;
    text-align: left;
    width: 400px;
    height: 400px;
  }

  #message, #nbPermutte{
    padding: 10px;
    float: left;
    text-align: center;
    width: 100%;
  }

</style>
<div id="container">
  <h2>Générateur de Takin - DWWMO1</h2>
  <div id="grid">

    </div>
    <pre id="nbPermutte"></pre>
    <pre id="message"></pre>
    <button id="generer">Générer grille</button>
  </div>
<script>

let grid = 4;
//tableau intial
let takin;
/* on mémorise l'emplacement de la case vide */
let emptyLig;
let emptyCol;
//Nombre de coups
let nb;

document.getElementById("generer").addEventListener("click", generer);

generer();

//mélange le tableau et réinitialise les variables
function generer(){

  nb = 0;
  emptyLig = grid;
  emptyCol = grid;
  takin = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, "V"];
  message("nbPermutte","",false);
  message("message","",false);
  random();
  grille();

}

function verifier(){

  let verifier = false;
  //on ne souhaite pas vérifier la dernière valeur du tableau qui doit être à "V" d'où - 1
  for (let i = 0; i < takin.length - 1;i++ ){
    let cellules = document.getElementById("grid").childNodes[i];

    if(Number(cellules.textContent) == i + 1){
      verifier = true;
    }else{
      verifier = false;
      break;
    }
  }
}

```

```

    }

    return verifier;
}

function random(){

    let iteration = Math.floor((Math.random() * 80) + 21); //iteration entr 20 et 100
    let permutation = 0;

    for(let i = 0; i < iteration ;i++){
        let possible = new Array(); //stock les permutation possible
        let pos = takin.indexOf("V"); //position de "V"
        //vérifier mouvement valide
        if(takin[pos - 1]){ //cellule gauche
            possible.push(pos - 1);
        }
        if(takin[pos + 1]){ //cellule droite
            possible.push(pos + 1);
        }
        if(takin[pos - grid]){ //cellule haut
            possible.push(pos - grid);
        }
        if(takin[pos + grid]){ //cellule bas
            possible.push(pos + grid);
        }
        //permutation alleatoire
        let permute = Math.floor(Math.random() * possible.length);
        let temp = takin[possible[permute]];
        takin[possible[permute]] = takin[pos];
        takin[pos] = temp;
        permutation++;
        trouve_empty();
    }
    console.log(takin);
    message("nbPermutte", `Takin resolvable maximum ${permutation} coups.`, false);
}

function trouve_empty() {
    let id = 0;
    bloc_externe:
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            if(takin[id] == "V"){
                emptyLig = x;
                emptyCol = y;
                break bloc_externe; //interrompt l'ensemble du bloc
            }
            id++;
        }
    }
}

function grille() {
    let id = 0;
    //supprime la grille
    document.getElementById("grid").innerHTML = "";
    //générer 9 carrés div, lignes et colonnes
    for (let x = 1; x <= grid; x++) {

        for (let y = 1; y <= grid; y++) {
            //créer cellule
            let cel = document.createElement('button');
            cel.setAttribute("id", "case_" + x + "_" + y);
            cel.innerHTML = takin[id] != "V" ? takin[id] : "&nbsp;";
            cel.classList.add("cellule");
            //ajouter la fonction
            cel.setAttribute("onclick", `move(${x},${y})`);
            if(takin[id] == "V") {
                cel.classList.add("white");
            }

            //ajoute l'ensemble à grid
            document.getElementById("grid").appendChild(cel);
            id++;
        }
    }
}

function move (lig,col) {
    /* on récupère les identifiants des deux boutons à intervertir */
    let bname = `case_${lig}_${col}`;
    let ename = `case_${emptyLig}_${emptyCol}`;
    /* on récupère les noeuds correspondant à ces boutons */
    let bnode = document.getElementById(bname);
    let enode = document.getElementById(ename);
    //ne fait rien si cellule blanche
    //verifier si case valide
    if (!bnode.classList.contains('white') &&
        ((emptyLig==lig) && ((col==emptyCol-1)|| (col==emptyCol+1)))
        || ((emptyCol==col) && ((lig==emptyLig-1)|| (lig==emptyLig+1)))
    ) {

```

```
/* on récupère les fils textuels des deux boutons */
let bvalue = bnode.removeChild(bnode.childNodes[0]);
let evalue = enode.removeChild(enode.childNodes[0]);
/* on échange ces fils */
bnode.appendChild(evalue);
enode.appendChild(bvalue);
/* on échange les classes des deux boutons */
bnode.setAttribute('class','cellule white');
enode.setAttribute('class','cellule');
/* on mémorise l'emplacement de la case vide */
emptyLig = lig;
emptyCol = col;
nb++;
message("message", `Vous avez réalisé ${nb} coups`, false);
if(verifier()){
    message("message", "Bravo vous avez gagné!", true);
}
}
}

function message(id, message, concat){
    if(concat){
        document.getElementById(id).innerHTML += `${message}\n`;
    }else{
        document.getElementById(id).innerHTML = `${message}\n`;
    }
}

</script>
```

5. Optimiser

Suite à la fonction Random précédente, nous souhaitons l'optimiser afin que la cellule vide qui à bougé ne revienne pas à la position précédente lors de l'itération. Comment améliorer votre code ?

5.1. Solution

```

<style>
  .cellule {
    background-color: aliceblue;
    border: 1px solid lightgray;
    display: inline-block;
    width: 100px;
    height: 100px;
    line-height: 100px;
    text-align: center;
    font-size: 1em;
  }

  .white{

    background-color: white;

  }

  #container {
    width: 400px;
    text-align: center;
    margin: 0 auto;
  }

  #grid {
    border: 4px solid black;
    float: left;
    margin: 0 10px;
    text-align: left;
    width: 400px;
    height: 400px;
  }

  #message, #nbPermutte{
    padding: 10px;
    float: left;
    text-align: center;
    width: 100%;
  }

</style>
<div id="container">
  <h2>Générateur de Takin - DWWM01</h2>
  <div id="grid">

    </div>
    <pre id="nbPermutte"></pre>
    <pre id="message"></pre>
    <button id="generer">Générer grille</button>
</div>
<script>

let grid = 4;
//tableau intial
let takin;
/* on mémorise l'emplacement de la case vide */
let emptyLig;
let emptyCol;
//Nombre de coups
let nb;

document.getElementById("generer").addEventListener("click", generer);

generer();

//mélange le tableau et réinitialise les variables
function generer(){

  nb = 0;
  emptyLig = grid;
  emptyCol = grid;
  takin = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, "V"];
  message("nbPermutte","",false);
  message("message","",false);
  random();
  grille();
}

function verifier(){

  let verifier = false;
  //on ne souhaite pas vérifier la dernière valeur du tableau qui doit être à "V" d'où - 1
  for (let i = 0; i < takin.length - 1;i++ ){
    let cellules = document.getElementById("grid").childNodes[i];

    if(Number(cellules.textContent) == i + 1){
      verifier = true;
    }else{
      verifier = false;
      break;
    }
  }
}

```

```

    }

    return verifier;
}

function random(){

    let iteration = Math.floor((Math.random() * 80) + 21); //iteration alléatoire entr 20 et 100
    let permutation = 0;
    let old_pos = takin.indexOf("V"); //stocker la position actuel de V

    for(let i = 0; i < iteration ;i++){
        let possible = new Array(); //stock les permutation possible
        let pos = takin.indexOf("V"); //position de "V"
        //vérifier mouvement valide
        if(takin[pos - 1]){ //cellule gauche
            possible.push(pos - 1);
        }
        if(takin[pos + 1]){ //cellule droite
            possible.push(pos + 1);
        }
        if(takin[pos - grid]){ //cellule haut
            possible.push(pos - grid);
        }
        if(takin[pos + grid]){ //cellule bas
            possible.push(pos + grid);
        }
        //permutation alleatoire
        let permute = Math.floor(Math.random() * possible.length);

        //si le changement de case n'est pas revenir à la position antécédente, sinon pas de permutation
        if(possible[permute] != old_pos){
            //on stocke l'ancienne position avant de changer
            old_pos = pos;
            let temp = takin[possible[permute]];
            //permutation
            takin[possible[permute]] = takin[pos];
            takin[pos] = temp;
            permutation++;
            //actualise la position ligne_colonne de "V"
            trouve_empty();
        }
    }

    console.log(takin);
    message("nbPermutte", `Takin resolvable maximum ${permutation} coups.`, false);
}

function trouve_empty() {
    let id = 0;
    bloc_externe:
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            if(takin[id] == "V"){
                emptyLig = x;
                emptyCol = y;
                break bloc_externe; //interrompt l'ensemble du bloc
            }
            id++;
        }
    }
}

function grille() {
    let id = 0;
    //supprime la grille
    document.getElementById("grid").innerHTML = "";
    //générer 9 carrés div, lignes et colonnes
    for (let x = 1; x <= grid; x++) {

        for (let y = 1; y <= grid; y++) {
            //créer cellule
            let cel = document.createElement('button');
            cel.setAttribute("id", "case_" + x + "_" + y);
            cel.innerHTML = takin[id] != "V" ? takin[id] : "&nbsp;";
            cel.classList.add("cellule");
            //ajouter la fonction
            cel.setAttribute("onclick", `move(${x},${y})`);
            if(takin[id] == "V") {
                cel.classList.add("white");
            }

            //ajoute l'ensemble à grid
            document.getElementById("grid").appendChild(cel);
            id++;
        }
    }
}

function move (lig,col) {
    /* on récupère les identifiants des deux boutons à intervertir */
    let bname = `case_${lig}_${col}`;

```

```

let ename = `case_${emptyLig}_${emptyCol}`;
/* on récupère les noeuds correspondant à ces boutons */
let bnode = document.getElementById(bname);
let enode = document.getElementById(ename);
//ne fait rien si cellule blanche
//verifier si case valide
if (!bnode.classList.contains('white') &&
    ((emptyLig==lig) && ((col==emptyCol-1)|| (col==emptyCol+1)))
    || ((emptyCol==col) && ((lig==emptyLig-1)|| (lig==emptyLig+1)))
) {
    /* on récupère les fils textuels des deux boutons */
    let bvalue = bnode.removeChild(bnode.childNodes[0]);
    let evalue = enode.removeChild(enode.childNodes[0]);
    /* on échange ces fils */
    bnode.appendChild(evalue);
    enode.appendChild(bvalue);
    /* on échange les classes des deux boutons */
    bnode.setAttribute('class','cellule white');
    enode.setAttribute('class','cellule');
    /* on mémorise l'emplacement de la case vide */
    emptyLig = lig;
    emptyCol = col;
    nb++;
    message("message", `Vous avez réalisé ${nb} coups`, false);
    if(verifier()){
        message("message", "Bravo vous avez gagné!", true);
    }
}
}

function message(id, message, concat){
    if(concat){
        document.getElementById(id).innerHTML += `${message}\n`;
    }else{
        document.getElementById(id).innerHTML = `${message}\n`;
    }
}
}

</script>

```

6. Afficher la solution

Lors de la génération Random nous souhaitons donner la possibilité d'afficher la solution.

Créer un bouton solution qui affiche les mouvements à effectué pour résoudre le takin sous la forme d'un tableau de ce type :

Haut, Bas, Gauche, Gauche, Bas...Etc... = Bravo vous avez gagné.

Générateur de Takin - DWWM01

3	8	7	10
1	15	2	4
5	6	12	14
9		13	11

Takin resolvable maximum 38 coups.

Droite (13),Haut (12),Droite (14),Haut (4),Haut (10),Gauche (7),Gauche (8),Gauche (3),Bas (1),Bas (5),Droite (6),Haut (15),Droite (2),Droite (10),Bas (4),Bas (11),Gauche (12),Haut (14),Gauche (15),Gauche (6),Bas (9),Droite (13),Droite (14),Haut (15),Haut (10),Droite (4),Haut (7),Gauche (8),Bas (4),Droite (7),Haut (8),Gauche (4),Gauche (3),Bas (2),Bas (6),Droite (10),Droite (11),Bas (12)

Générer grilleVoir solution

6.1. Solution

```

<style>
  .cellule {
    background-color: aliceblue;
    border: 1px solid lightgray;
    display: inline-block;
    width: 100px;
    height: 100px;
    line-height: 100px;
    text-align: center;
    font-size: 1em;
  }

  .white{

    background-color: white;

  }

  textarea{
    float: left;
    width: 100%;
    margin-bottom: 20px;
    display: none;
  }

  #container {
    width: 400px;
    text-align: center;
    margin: 0 auto;
  }

  #grid {
    border: 4px solid black;
    float: left;
    margin: 0 10px;
    text-align: left;
    width: 400px;
    height: 400px;
  }

  #message, #nbPermutte{
    padding: 10px;
    float: left;
    text-align: center;
    width: 100%;
  }

</style>
<div id="container">
  <h2>Générateur de Takin - DWWM01</h2>
  <div id="grid">

    </div>
    <pre id="nbPermutte"></pre>
    <pre id="message"></pre>
    <textarea id="solution"></textarea>
    <button id="generer">Générer grille</button>
    <button id="affSolution">Voir solution</button>
  </div>
<script>

let grid = 4;
//tableau intial
let takin;
//tableau solution
let solution;
/* on mémorise l'emplacement de la case vide */
let emptyLig;
let emptyCol;
//Nombre de coups
let nb;

document.getElementById("generer").addEventListener("click", generer);
document.getElementById("affSolution").addEventListener("click", function(){affSolution(true)});
generer();

//mélange le tableau et réinitialise les variables
function generer(){

  nb = 0;
  emptyLig = grid;
  emptyCol = grid;
  takin = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, "V"];
  solution = new Array();
  affSolution(false);
  message("nbPermutte","",false);
  message("message","",false);
  random();
  grille();

}

```

```

function verifier(){

    let verifier = false;
    //on ne souhaite pas vérifier la dernière valeur du tableau qui doit être à "V" d'où - 1
    for (let i = 0; i < takin.length - 1;i++) {
        let cellules = document.getElementById("grid").childNodes[i];

        if(Number(cellules.textContent) == i + 1){
            verifier = true;
        }else{
            verifier = false;
            break;
        }
    }

    return verifier;
}

function random(){

    let iteration = Math.floor((Math.random() * 80) + 21); //iteration aléatoire entr 20 et 100
    let permutation = 0;
    let old_pos = takin.indexOf("V"); //stocker la position actuel de V

    for(let i = 0; i < iteration ;i++){
        let possible = new Array(); //stock les permutation possible
        let tmpSolution = new Array(); //stock les solutions possible
        let pos = takin.indexOf("V"); //position de "V"
        //vérifier mouvement valide
        if(takin[pos - 1] && verifBorder(chercheCoord(takin[pos - 1]))){ //cellule gauche
            possible.push(pos - 1);
            tmpSolution.push(`Droite (${takin[pos - 1]})`);
        }
        if(takin[pos + 1] && verifBorder(chercheCoord(takin[pos + 1]))){ //cellule droite
            possible.push(pos + 1);
            tmpSolution.push(`Gauche (${takin[pos + 1]})`);
        }
        if(takin[pos - grid]){ //cellule haut
            possible.push(pos - grid);
            tmpSolution.push(`Bas (${takin[pos - grid]})`);
        }
        if(takin[pos + grid]){ //cellule bas
            possible.push(pos + grid);
            tmpSolution.push(`Haut (${takin[pos + grid]})`);
        }
        //permutation aléatoire
        let permute = Math.floor(Math.random() * possible.length);

        //si le changement de case n'est pas revenir à la position antécédente, sinon pas de permutation
        if(possible[permute] != old_pos){
            //on stocke l'ancienne position avant de changer
            old_pos = pos;
            let temp = takin[possible[permute]];
            //Stock dans solution
            solution.push(tmpSolution[permute]);
            //permutation
            takin[possible[permute]] = takin[pos];
            takin[pos] = temp;
            permutation++;
            //actualise la position ligne_colonne de "V"
            trouve_empty();
        }
    }

    console.log(takin);
    message("nbPermutte", `Takin resolvable maximum ${permutation} coups.`, false);
}

function chercheCoord(valeur){
    let id = 0;
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            if (valeur == takin[id]){
                let lig = x;
                let col = y;
                return [x,y];
            }
            id++;
        }
    }
}

function verifBorder(tab){

    let valide = false;
    let lig = tab[0];
    let col = tab[1];

    if(
        ((emptyLig==lig) && ((col==emptyCol-1)|| (col==emptyCol+1)))
        || ((emptyCol==col) && ((lig==emptyLig-1)|| (lig==emptyLig+1)))
    ){
        valide = true;
    }
}

```



```

    };

    return valide;
}

//reinitialise les coordonnées de la case vide
function trouve_empty() {
    let id = 0;
    bloc_externe:
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            if(takin[id] == "V"){
                emptyLig = x;
                emptyCol = y;
                break bloc_externe;//interrompt l'ensemble du bloc
            }
            id++;
        }
    }
}

function grille() {
    let id = 0;
    //supprime la grille
    document.getElementById("grid").innerHTML = "";
    //générer 9 carrés div, lignes et colonnes
    for (let x = 1; x <= grid; x++) {

        for (let y = 1; y <= grid; y++) {
            //créer cellule
            let cel = document.createElement('button');
            cel.setAttribute("id", "case_" + x + "_" + y);
            cel.innerHTML = takin[id] != "V" ? takin[id] : "&nbsp;";
            cel.classList.add("cellule");
            //ajouter la fonction
            cel.setAttribute("onclick", `move(${x},${y})`);
            if(takin[id] == "V") {
                cel.classList.add("white");
            }

            //ajoute l'ensemble à grid
            document.getElementById("grid").appendChild(cel);
            id++;
        }
    }
}

function move (lig,col) {
    /* on récupère les identifiants des deux boutons à intervertir */
    let bname = `case_${lig}_${col}`;
    let ename = `case_${emptyLig}_${emptyCol}`;
    /* on récupère les noeuds correspondant à ces boutons */
    let bnode = document.getElementById(bname);
    let enode = document.getElementById(ename);
    //ne fait rien si cellule blanche
    //verifier si case valide
    if (!bnode.classList.contains('white') && verifBorder([lig,col])) {
        /* on récupère les fils textuels des deux boutons */
        let bvalue = bnode.removeChild(bnode.childNodes[0]);
        let evalue = enode.removeChild(enode.childNodes[0]);
        /* on échange ces fils */
        bnode.appendChild(evalue);
        enode.appendChild(bvalue);
        /* on échange les classes des deux boutons */
        bnode.setAttribute('class', 'cellule white');
        enode.setAttribute('class', 'cellule');
        /* on mémorise l'emplacement de la case vide */
        emptyLig = lig;
        emptyCol = col;
        nb++;
        message("message", `Vous avez réalisé ${nb} coups`, false);
        if(verifier()){
            message("message", "Bravo vous avez gagné!", true);
        }
    }
}

function affSolution(afficher) {

    let affSolution = document.getElementById("solution");
    if(afficher){
        affSolution.style.display = 'block';
        affSolution.value = solution.reverse();
    }else{
        affSolution.style.display = 'none';
        affSolution.value = '';
    }

}

/**
 *
 * @param id

```

```
* @param message
* @param concat
*/
function message(id, message, concat){
  if(concat){
    document.getElementById(id).innerHTML += `${message}\n`;
  }else{
    document.getElementById(id).innerHTML = `${message}\n`;
  }
}

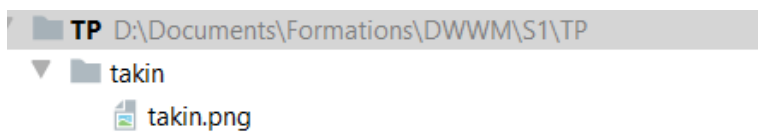
</script>
```

7. Utiliser une image

Nous souhaitons maintenant remplacer nos cellules par cette image :



- faites un click droit sur l'image et enregistrez là sur votre projet dans un dossier appelé takin :



- Pour afficher cette image dans une cellule, utilisé les propriété css ci-dessous :

```
background-image: url(./takin/takin.png);  
background-position: 0px 0px;
```

L'élément CSS background-position: 0px 0px; permet de positionner l'image en haut à gauche, vous pouvez faire :

```
/* image cellule 1 */  
background-position: 0px 0px;  
/* image cellule 2 */  
background-position: -100px 0px;  
/* image cellule 5 */  
background-position: 0px -100px;
```

Vous devez obtenir le résultat suivant :

Générateur de Takin - DWWM01



Takin resolvable maximum 56 coups.

Générer grille

Voir solution

7.1. Solution

```

<style>
  .cellule {
    background-color: aliceblue;
    border: 1px solid lightgray;
    display: inline-block;
    width: 100px;
    height: 100px;
    line-height: 100px;
    text-align: center;
    font-size: 1em;
    background-image: url("../takin/takin.png");
    background-position: 0px 100px ;
  }

  .white{

    background-color: white;
    background-image: none;
  }

  textarea{
    float: left;
    width: 100%;
    margin-bottom: 20px;
    display: none;
  }

  #container {
    width: 400px;
    text-align: center;
    margin: 0 auto;
  }

  #grid, #win {

    border: 4px solid black;
    float: left;
    margin: 0 10px;
    text-align: left;
    width: 400px;
    height: 400px;
  }

  #win{
    position: absolute;
    background-color: rgba(255,0,0,0.8);
    display: none;
  }
  #message, #nbPermutte{
    padding: 10px;
    float: left;
    text-align: center;
    width: 100%;
  }

</style>
<div id="container">
  <h2>Générateur de Takin - DWWMO1</h2>
  <div id="win"></div>
  <div id="grid"></div>
  <pre id="nbPermutte"></pre>
  <pre id="message"></pre>
  <textarea id="solution"></textarea>
  <button id="generer">Générer grille</button>
  <button id="affSolution">Voir solution</button>
</div>
<script>
//taille cellule
let size = 100;
//taille grille
let grid = 4;
//tableau intial
let takin;
let background = new Array();
//tableau solution
let solution;
/* on mémorise l'emplacement de la case vide */
let emptyLig;
let emptyCol;
//Nombre de coups
let nb;

document.getElementById("generer").addEventListener("click", generer);
document.getElementById("affSolution").addEventListener("click", function(){affSolution(true)});
initBackground();
generer();

//mélange le tableau et réinitialise les variables
function generer(){

  nb = 0;
  emptyLig = grid;

```

```

emptyCol = grid;
document.getElementById("win").style.display = "none";
taken = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, "V"];
solution = new Array();
affSolution(false);
message("nbPermutte", "", false);
message("message", "", false);
random();
grille();
}

function initBackground(){
    let id = 0;
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            background[id] = `${(y-1) * size}px ${(x-1) * size}px`;
            id++;
        }
    }
}

function verifier(){
    let verifier = false;
    //on ne souhaite pas vérifier la dernière valeur du tableau qui doit être à "V" d'où - 1
    for (let i = 0; i < taken.length - 1; i++) {
        let cellules = document.getElementById("grid").childNodes[i];
        if(cellules.style.backgroundColor == background[i]){
            verifier = true;
        }else{
            verifier = false;
            break;
        }
    }

    return verifier;
}

function random(){
    let iteration = Math.floor((Math.random() * 80) + 21); //iteration aléatoire entr 20 et 100
    let permutation = 0;
    let old_pos = taken.indexOf("V"); //stocker la position actuel de V

    for(let i = 0; i < iteration; i++){
        let possible = new Array(); //stock les permutation possible
        let tmpSolution = new Array(); //stock les solutions possible
        let pos = taken.indexOf("V"); //position de "V"
        //vérifier mouvement valide
        if(taken[pos - 1] && verifBorder(chercheCoord(taken[pos - 1]))){ //cellule gauche
            possible.push(pos - 1);
            tmpSolution.push(`Droite (${taken[pos - 1]})`);
        }
        if(taken[pos + 1] && verifBorder(chercheCoord(taken[pos + 1]))){ //cellule droite
            possible.push(pos + 1);
            tmpSolution.push(`Gauche (${taken[pos + 1]})`);
        }
        if(taken[pos - grid]){ //cellule haut
            possible.push(pos - grid);
            tmpSolution.push(`Bas (${taken[pos - grid]})`);
        }
        if(taken[pos + grid]){ //cellule bas
            possible.push(pos + grid);
            tmpSolution.push(`Haut (${taken[pos + grid]})`);
        }
        //permutation aléatoire
        let permute = Math.floor(Math.random() * possible.length);

        //si le changement de case n'est pas revenir à la position antécédente, sinon pas de permutation
        if(possible[permute] != old_pos){
            //on stocke l'ancienne position avant de changer
            old_pos = pos;
            let temp = taken[possible[permute]];
            //Stock dans solution
            solution.push(tmpSolution[permute]);
            //permutation
            taken[possible[permute]] = taken[pos];
            taken[pos] = temp;
            permutation++;
            //actualise la position ligne_colonne de "V"
            trouve_empty();
        }
    }

    console.log(taken);
    message("nbPermutte", `Takin resolvable maximum ${permutation} coups.`, false);
}

function chercheCoord(valeur){
    let id = 0;
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {

```



```

        if (valeur == takin[id]){
            let lig = x;
            let col = y;
            return [x,y];
        }
        id++;
    }
}

function verifBorder(tab){

    let valide = false;
    let lig = tab[0];
    let col = tab[1];

    if(
        ((emptyLig==lig) && ((col==emptyCol-1)|| (col==emptyCol+1)))
        || ((emptyCol==col) && ((lig==emptyLig-1)|| (lig==emptyLig+1)))
    ){
        valide = true;
    };

    return valide;
}

//reinitialise les coordonnées de la case vide
function trouve_empty() {
    let id = 0;
    bloc_externe:
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            if(takin[id] == "V"){
                emptyLig = x;
                emptyCol = y;
                break bloc_externe;//interrompt l'ensemble du bloc
            }
            id++;
        }
    }
}

function grille() {
    let id = 0;
    //supprime la grille
    document.getElementById("grid").innerHTML = "";
    //générer 9 carrés div, lignes et colonnes
    for (let x = 1; x <= grid; x++) {

        for (let y = 1; y <= grid; y++) {
            //créer cellule
            let cel = document.createElement('button');
            cel.setAttribute("id", "case_" + x + "_" + y);
            //cel.innerHTML = takin[id] != "V" ? takin[id] : "&nbsp;";
            cel.classList.add("cellule");
            cel.style.backgroundPosition = background[takin[id]-1];
            //ajouter la fonction
            cel.setAttribute("onclick", `move(${x},${y})`);
            if(takin[id] == "V") {
                cel.classList.add("white");
            }

            //ajoute l'ensemble à grid
            document.getElementById("grid").appendChild(cel);
            id++;
        }
    }
}

function move (lig,col) {
    /* on récupère les identifiants des deux boutons à intervertir */
    let bname = `case_${lig}_${col}`;
    let ename = `case_${emptyLig}_${emptyCol}`;
    /* on récupère les noeuds correspondant à ces boutons */
    let bnode = document.getElementById(bname);
    let enode = document.getElementById(ename);
    //ne fait rien si cellule blanche
    //verifier si case valide
    if (!bnode.classList.contains('white') && verifBorder([lig,col])) {
        /* on récupère les fils textuels des deux boutons */
        let bvalue = bnode.style.backgroundPosition;
        let evalue = enode.style.backgroundPosition;
        /* on échange ces fils */
        bnode.style.backgroundPosition = evalue;
        enode.style.backgroundPosition = bvalue;
        /* on échange les classes des deux boutons */
        bnode.setAttribute('class','cellule white');
        enode.setAttribute('class','cellule');
        /* on mémorise l'emplacement de la case vide */
        emptyLig = lig;
        emptyCol = col;
        nb++;
        message("message", `Vous avez réalisé ${nb} coups`, false);
    }
}

```



```
        if(verifier()){
            document.getElementById("win").style.display = "block";
            message("message", "Bravo vous avez gagné!", true);
        }
    }
}

function affSolution(afficher) {

    let affSolution = document.getElementById("solution");
    if(afficher){
        affSolution.style.display = 'block';
        affSolution.value = solution.reverse();
    }else{
        affSolution.style.display = 'none';
        affSolution.value = '';
    }

}

}

/**
 *
 * @param id
 * @param message
 * @param concat
 */
function message(id, message, concat){
    if(concat){
        document.getElementById(id).innerHTML += `${message}\n`;
    }else{
        document.getElementById(id).innerHTML = `${message}\n`;
    }
}

}

</script>
```

8. Ajouter un niveau

Ajouter une liste déroulante permettant de déterminer un niveau pour générer le Takin :

- Facile < 10 itérations
- Débutant, entre 20 et 40 itérations
- Expert, > 40 itérations

8.1. Solution

```

<style>
    .cellule {
        background-color: aliceblue;
        border: 1px solid lightgray;
        display: inline-block;
        width: 100px;
        height: 100px;
        line-height: 100px;
        text-align: center;
        font-size: 1em;
        background-image: url("../takin/takin.png");
        background-position: 0px 100px ;
    }

    .white{

        background-color: white;
        background-image: none;
    }

    textarea{
        float: left;
        width: 100%;
        margin-bottom: 20px;
        display: none;
    }

    #container {
        width: 400px;
        text-align: center;
        margin: 0 auto;
    }

    #grid, #win {

        border: 4px solid black;
        float: left;
        margin: 0 10px;
        text-align: left;
        width: 400px;
        height: 400px;
    }

    #win{
        position: absolute;
        background-color: rgba(255,0,0,0.8);
        display: none;
    }
    #message, #nbPermutte{
        padding: 10px;
        float: left;
        text-align: center;
        width: 100%;
    }

</style>
<div id="container">
    <h2>Générateur de Takin - DWWMO1</h2>
    <div id="win"></div>
    <div id="grid"></div>
    <pre id="nbPermutte"></pre>
    <pre id="message"></pre>
    <textarea id="solution"></textarea>
    <label for="select">Niveau</label>
    <select id="level">
        <option value="10">Facile</option>
        <option value="40">Débutant</option>
        <option value="80">Expert</option>
    </select>
    <button id="generer">Générer grille</button>
    <button id="affSolution">Voir solution</button>
</div>
<script>
//taille cellule
let size = 100;
//taille grille
let grid = 4;
//tableau intial
let takin;
let background = new Array();
//tableau solution
let solution;
/* on mémorise l'emplacement de la case vide */
let emptyLig;
let emptyCol;
//Nombre de coups
let nb;

document.getElementById("generer").addEventListener("click", generer);
document.getElementById("affSolution").addEventListener("click", function(){affSolution(true)});
initBackground();
generer();

```

```

//mélange le tableau et réinitialise les variables
function generer(){
    nb = 0;
    emptyLig = grid;
    emptyCol = grid;
    document.getElementById("win").style.display = "none";
    takin = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, "V"];
    solution = new Array();
    affSolution(false);
    message("nbPermutte", "", false);
    message("message", "", false);
    random(Number(document.getElementById("level").value));
    grille();
}

function initBackground(){
    let id = 0;
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            background[id] = `${- ((y-1) * size)}px ${- ((x-1) * size)}px`;
            id++;
        }
    }
}

function verifier(){
    let verifier = false;
    //on ne souhaite pas vérifier la dernière valeur du tableau qui doit être à "V" d'où - 1
    for (let i = 0; i < takin.length - 1; i++) {
        let cellules = document.getElementById("grid").childNodes[i];
        if(cellules.style.backgroundColor == background[i]){
            verifier = true;
        }else{
            verifier = false;
            break;
        }
    }

    return verifier;
}

function random(level){
    let iteration = Math.floor((Math.random() * level) + level); //iteration aléatoire
    let permutation = 0;
    let old_pos = takin.indexOf("V"); //stocker la position actuel de V

    for(let i = 0; i < iteration; i++){
        let possible = new Array(); //stock les permutation possible
        let tmpSolution = new Array(); //stock les solutions possible
        let pos = takin.indexOf("V"); //position de "V"
        //vérifier mouvement valide
        if(takin[pos - 1] && verifBorder(chercheCoord(takin[pos - 1]))){ //cellule gauche
            possible.push(pos - 1);
            tmpSolution.push(`Droite (${takin[pos - 1]})`);
        }
        if(takin[pos + 1] && verifBorder(chercheCoord(takin[pos + 1]))){ //cellule droite
            possible.push(pos + 1);
            tmpSolution.push(`Gauche (${takin[pos + 1]})`);
        }
        if(takin[pos - grid]){ //cellule haut
            possible.push(pos - grid);
            tmpSolution.push(`Bas (${takin[pos - grid]})`);
        }
        if(takin[pos + grid]){ //cellule bas
            possible.push(pos + grid);
            tmpSolution.push(`Haut (${takin[pos + grid]})`);
        }
        //permutation aléatoire
        let permute = Math.floor(Math.random() * possible.length);

        //si le changement de case n'est pas revenir à la position antécédente, sinon pas de permutation
        if(possible[permute] != old_pos){
            //on stocke l'ancienne position avant de changer
            old_pos = pos;
            let temp = takin[possible[permute]];
            //Stock dans solution
            solution.push(tmpSolution[permute]);
            //permutation
            takin[possible[permute]] = takin[pos];
            takin[pos] = temp;
            permutation++;
            //actualise la position ligne_colonne de "V"
            trouve_empty();
        }
    }
    console.log(takin);
    message("nbPermutte", `Takin resolvable maximum ${permutation} coups.`, false);
}

```

```

function chercheCoord(valeur){
    let id = 0;
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            if (valeur == takin[id]){
                let lig = x;
                let col = y;
                return [x,y];
            }
            id++;
        }
    }
}

function verifBorder(tab){

    let valide = false;
    let lig = tab[0];
    let col = tab[1];

    if(
        ((emptyLig==lig) && ((col==emptyCol-1)|| (col==emptyCol+1)))
        || ((emptyCol==col) && ((lig==emptyLig-1)|| (lig==emptyLig+1)))
    ){
        valide = true;
    };

    return valide;
}

//reinitialise les coordonnées de la case vide
function trouve_empty() {
    let id = 0;
    bloc_extern:
    for (let x = 1; x <= grid; x++) {
        for (let y = 1; y <= grid; y++) {
            if(takin[id] == "V"){
                emptyLig = x;
                emptyCol = y;
                break bloc_extern;//interrompt l'ensemble du bloc
            }
            id++;
        }
    }
}

function grille() {
    let id = 0;
    //supprime la grille
    document.getElementById("grid").innerHTML = "";
    //générer 9 carrés div, lignes et colonnes
    for (let x = 1; x <= grid; x++) {

        for (let y = 1; y <= grid; y++) {
            //créer cellule
            let cel = document.createElement('button');
            cel.setAttribute("id", "case_" + x + "_" + y);
            //cel.innerHTML = takin[id] != "V" ? takin[id] : "&nbsp;";
            cel.classList.add("cellule");
            cel.style.backgroundPosition = background[takin[id]-1];
            //ajouter la fonction
            cel.setAttribute("onclick", `move(${x},${y})`);
            if(takin[id] == "V") {
                cel.classList.add("white");
            }

            //ajoute l'ensemble à grid
            document.getElementById("grid").appendChild(cel);
            id++;
        }
    }
}

function move (lig,col) {
    /* on récupère les identifiants des deux boutons à intervertir */
    let bname = `case_${lig}_${col}`;
    let ename = `case_${emptyLig}_${emptyCol}`;
    /* on récupère les noeuds correspondant à ces boutons */
    let bnode = document.getElementById(bname);
    let enode = document.getElementById(ename);
    //ne fait rien si cellule blanche
    //verifier si case valide
    if (!bnode.classList.contains('white') && verifBorder([lig,col])) {
        /* on récupère les fils textuels des deux boutons */
        let bvalue = bnode.style.backgroundPosition;
        let evalue = enode.style.backgroundPosition;
        /* on échange ces fils */
        bnode.style.backgroundPosition = evalue;
        enode.style.backgroundPosition = bvalue;
        /* on échange les classes des deux boutons */
        bnode.setAttribute('class','cellule white');
        enode.setAttribute('class','cellule');
    }
}

```

```

    /* on mémorise l'emplacement de la case vide */
    emptyLig = lig;
    emptyCol = col;
    nb++;
    message("message", `Vous avez réalisé ${nb} coups`, false);
    if(verifier()){
        document.getElementById("win").style.display = "block";
        message("message", "Bravo vous avez gagné!", true);
    }
}
function affSolution(afficher) {

    let affSolution = document.getElementById("solution");
    if(afficher){
        affSolution.style.display = 'block';
        affSolution.value = solution.reverse();
    }else{
        affSolution.style.display = 'none';
        affSolution.value = '';
    }

}

/**
 *
 * @param id
 * @param message
 * @param concat
 */
function message(id, message, concat){
    if(concat){
        document.getElementById(id).innerHTML += `${message}\n`;
    }else{
        document.getElementById(id).innerHTML = `${message}\n`;
    }
}
}

</script>

```

Evaluation de fin de module

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Evaluation de fin de module

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:57

Table des matières

1. Avant-propos

2. Exercice 1

2.1. Correction

3. Exercice 2

3.1. Correction

4. Exercice 3

4.1. Correction

5. Exercice 4

5.1. Correction

1. Avant-propos

Les exercices sont indépendants. L'exercice 4 est facultatif.

L'épreuve dure 4 heures, vous avez la possibilité de prendre une pause de 15 minutes chacun votre tour.

Déposer chaque projet compressé dans le dépôt prévu sur amio-fit.

Vous pouvez vous aider des TP réalisés en cours, mais pas de recherche sur Internet indiquant la solution algorithmique. **N'allez pas copier/coller une solution toute faite !**

Si vous avez terminé l'épreuve, merci de rester silencieux à votre poste.

2. Exercice 1

Principe

Une suite de Syracuse est **une suite de nombre entiers** définie de la manière suivante :

On part d'un **nombre entier plus grand que zéro** :

- **Si le nombre est pair on le divise par 2**
- **Si le nombre est impair on le multiplie par 3 et on ajoute 1**

Au bout de quelques itérations, on finit toujours par atteindre la valeur 1 quel que soit le nombre de départ.

Travail demandé

Ecrire un programme qui demande à l'utilisateur de saisir un nombre entier compris entre 1 et 100.
Tant que le nombre saisi n'est pas valide, le programme redemande à l'utilisateur d'entrer un autre nombre.

Le programme affiche la suite de Syracuse partant de ce nombre jusqu'à atteindre la valeur 1.

Exemple de résultat attendu pour le nombre 22 :

22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

2.1. Correction

```
<!-- code HTML -->
<label>Ajouter un nombre :</label>
<input type="text" id="nombre" value="">
<button id="syracuse">Suite Syracuse</button>
<pre id="resultat"></pre>

<script>

  //Boutton Evenement
  document.getElementById("syracuse").addEventListener("click", function(){

    let nombre = document.getElementById("nombre").value;
    document.getElementById("resultat").innerHTML += "Suite de Syracuse pour le nombre " + nombre + "\n";
    document.getElementById("resultat").innerHTML += Syracuse(nombre);

  });

  function Syracuse(nombre){

    //verifier Nombre
    while(!(nombre >= 0 && nombre <= 100)){

      nombre = prompt("Votre nombre doit être compris entre 0 et 100");

    }

    //suite une chaîne
    let suite = "";
    //boucle tant que nombre n'est pas égale à 1
    while(nombre != 1){

      suite += nombre + ", ";
      //si pair
      if (nombre%2 == 0){
        nombre /= 2;
      }else{
        nombre = (nombre * 3) + 1;
      }

    }

    //fin de la suite on ajoute 1
    suite += "1";

    return suite;

  }

</script>
```

3. Exercice 2

Soit un tableau initialisé avec des entiers, le tableau contient des doublons.

```
let tableau = [1, 2, 4, 2, 4, 2, 8, 12];
```

Ecrire un programme qui demande un nombre à l'utilisateur et **recherche le nombre d'occurrences dans le tableau** (nombre de fois où il est présent). Le tableau n'est pas trié.

Tester différentes situations (utiliser un `switch` de préférence) :

- Un nombre pas présent dans le tableau
- Un nombre présent une seule fois dans le tableau
- Un nombre présent plusieurs fois dans le tableau

3.1. Correction

```
<!-- code HTML -->
<label>Ajouter un nombre :</label>
<input type="text" id="nombre" value="">
<button id="verifier">Verifier nombre</button>
<pre id="resultat"></pre>

<script>

  //Boutton Evenement
  document.getElementById("verifier").addEventListener("click", function(){

    let nombre = document.getElementById("nombre").value;
    document.getElementById("resultat").innerHTML += "Nombre " + nombre + " - " + Verifier(nombre) + "\n";

  });

  function Verifier(nombre){

    let tableau = [1, 2, 4, 2, 4, 2, 8, 12];
    let compteur = 0;
    let resultat = "";

    //boucle tableau
    for(let i = 0; i < tableau.length; i++){

      if(tableau[i] == nombre){
        compteur++;
      }

    }

    //analyse de compteur
    switch (compteur) {

      case 0:
        resultat = "Non présent dans le tableau";
        break;
      case 1:
        resultat = "Présent une seule fois dans le tableau";
        break;
      default:
        resultat = "Présent plusieurs fois dans le tableau";
        break;
    }

    return resultat;

  }

</script>
```

4. Exercice 3

Principe

On cherche à déterminer le jour de la semaine (lundi, mardi, etc.) pour une date donnée.

Pour cela on utilise les variables suivantes :

- c = 1 pour janvier et février, c = 0 pour les autres mois.
- a = année - c
- m = mois + 12*c - 2

Formule :

- $j = (jour + a + a/4 - a/100 + a/400 + (31*m)/12) \% 7$

Remarque : Dans toutes les divisions dans « / » de la formule j, on ne garde que la partie entière du résultat. Par exemple, 35/4= 8. Pour cela utilisez **Math.trunc()**. Math.trunc(35/4) = 8;

La réponse obtenue pour j correspond alors à un jour de la semaine suivant :

0 = dimanche, 1 = lundi, 2 = mardi, etc.

Exemple, quel jour de la semaine pour le **13 mars 2004** ?

- c = 0
- a = 2004 - 0
- m = 3 + (12 * 0) - 2 = 1
- $(13 + 2004 + 2004/4 - 2004/100 + 2004/400 + (31*1)/12) \% 7 = 2505 \% 7 = 6$

6 = **Samedi**.

Travail à réaliser

Ecrire une fonction qui permet se calcul, cela fonctionne pour une date comprise entre les années 1583 à 9999 (condition de vérification optionnelle).

Demandez à l'utilisateur de saisir une date sous la forme "25/12/2015" et afficher le résultat sous la forme : "Le 25/12/2015, tombe un vendredi !". Testez si cela fonctionne bien vous devez obtenir les résultats suivant :

```
Le 20/02/2003, tombe un jeudi !
Le 12/01/2001, tombe un vendredi !
Le 13/03/2004, tombe un samedi !
Le 25/12/2015, tombe un vendredi !
Le 22/03/2022, tombe un mardi !
```

4.1. Correction

```
<!-- code HTML -->
<label>Ajouter une date jj/mm/aaaa :</label>
<input type="text" id="date" value="22/03/2022">
<button id="jour">Obtenir le jour de la semaine</button>
<pre id="resultat"></pre>

<script>

  //Boutton Evenement
  document.getElementById("jour").addEventListener("click", function(){

    let date = document.getElementById("date").value;
    document.getElementById("resultat").innerHTML += "Le " + date + ", tombe un " + Jour(date) + " !\n";

  });

  function Jour(date){

    let semaine = ["dimanche", "lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi"];
    let jour = parseInt(date[0] + date[1]);
    let mois = parseInt(date[3] + date[4]);
    let annee = parseInt(date[6] + date[7] + date[8] + date[9]);

    //ternaire pour c
    let c = mois < 3 ? 1 : 0;
    let a = annee - c;
    let m = mois + (12 * c) - 2;

    let j = (jour + a + Math.trunc(a/4) - Math.trunc(a/100) + Math.trunc(a/400) + Math.trunc((31*m)/12))%7;

    return semaine[j];

  }

</script>
```


5. Exercice 4

Principe

Un nombre est dit **parfait** lorsqu'il est **égal à la somme de ses diviseurs** (1 est considéré comme un diviseur mais pas le nombre lui-même).

Exemples :

6 est parfait car 1, 2 et 3 sont ses diviseurs et que : $1 + 2 + 3 = 6$

28 est parfait car 1, 2, 4, 7, 14 sont ses diviseurs : $1 + 2 + 4 + 7 + 14 = 28$.

Travail à réaliser

Ecrire un programme qui demande à l'utilisateur de saisir un nombre puis détermine s'il est parfait.

Optionnellement : afficher la liste de tous les nombres parfaits inférieurs à 10 000.

5.1. Correction

```
<!-- code HTML -->
<label>Ajouter un nombre :</label>
<input type="text" id="nombre" value="28">
<button id="parfait">Est il parfait ?</button>
<button id="max">Afficher nombres parfait < 10 000</button>
<pre id="resultat"></pre>

<script>

  //Bouton Evenement
  document.getElementById("parfait").addEventListener("click", function(){

    let nombre = document.getElementById("nombre").value;
    document.getElementById("resultat").innerHTML += nombre + (Parfait(nombre) ? " est " : " n'est pas ") + "parfait.\n";

  });

  document.getElementById("max").addEventListener("click", function(){

    for (let i = 2; i < 10000; i++){

      if(Parfait(i)){

        document.getElementById("resultat").innerHTML += i + ", est un nombre parfait.\n"

      }

    }

  });

  function Parfait(nb){

    let sommeDiviseur = 0;
    //chercher diviseur
    for (i = 1; i < nb; i++){
      //Vérifier si diviseur est entier
      if(nb%i == 0){
        sommeDiviseur += i;
      }
    }

    return sommeDiviseur == nb ? true : false;

  }

</script>
```

Présentation du socle Algorithme S1

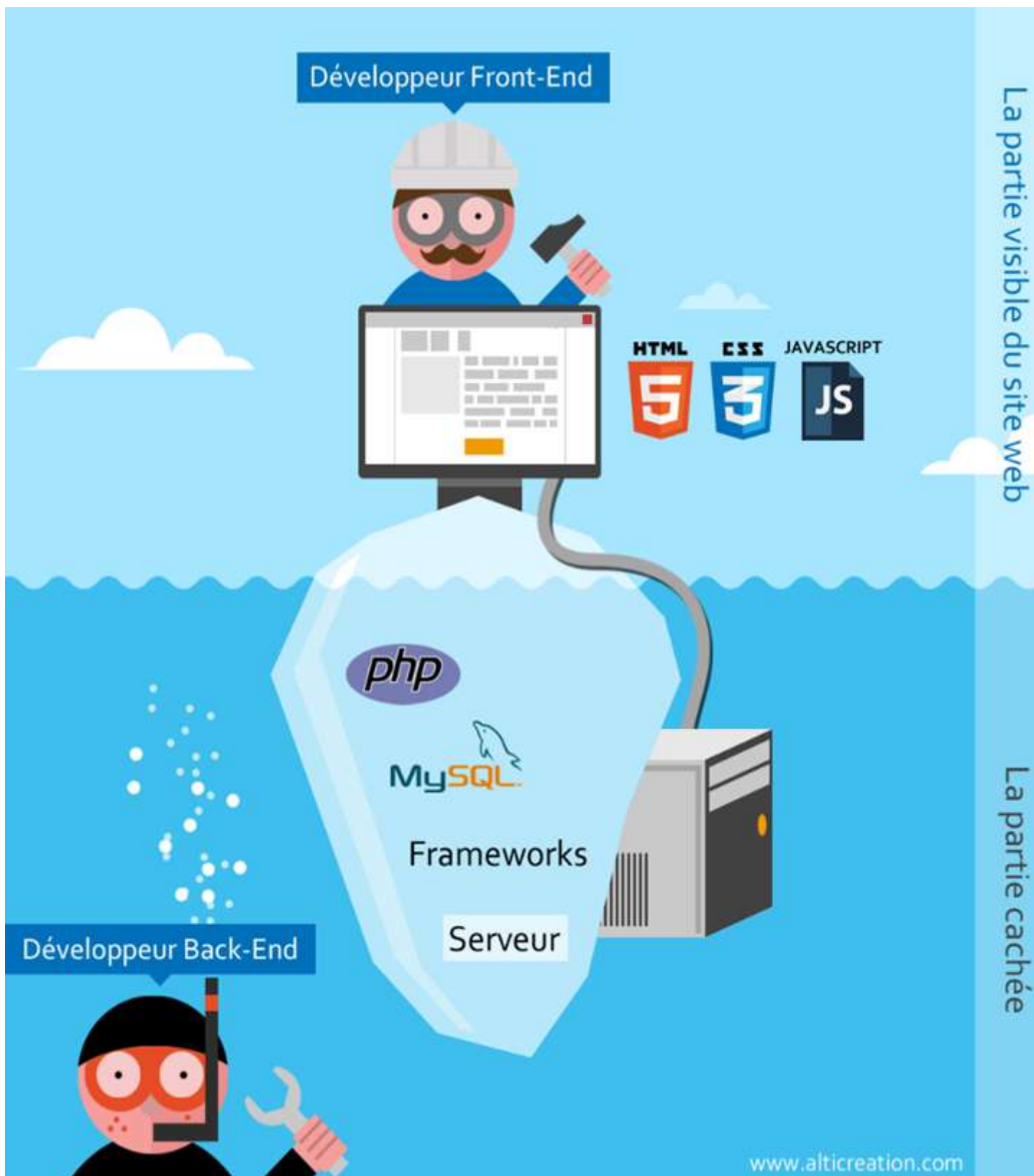
Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Présentation du socle Algorithme S1

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:51

Table des matières

- 1. Algorithme JAVASCRIPT
- 2. Nous étudierons quoi ?
- 3. On se situe où ?
- 4. Pourquoi, comment ?
- 5. L'algorithme, c'est quoi ?
- 6. Et concrètement ?

1. Algorithme JAVASCRIPT



2. Nous étudierons quoi ?

Javascript qui est un langage de développement web frontal (aussi appelé front-end en anglais) souvent associé aux production HTML, CSS d'une page internet :

- Le **HTML** : **langage de balisage** conçu pour représenter les pages web
- LE **CSS** : **langage informatique** qui décrit la présentation des documents HTML
- Le **Javascript** : **langage de programmation** de scripts principalement employé dans les pages web interactives

On peut aussi utiliser **JAVASCRIPT** en langage serveur avec node.js.



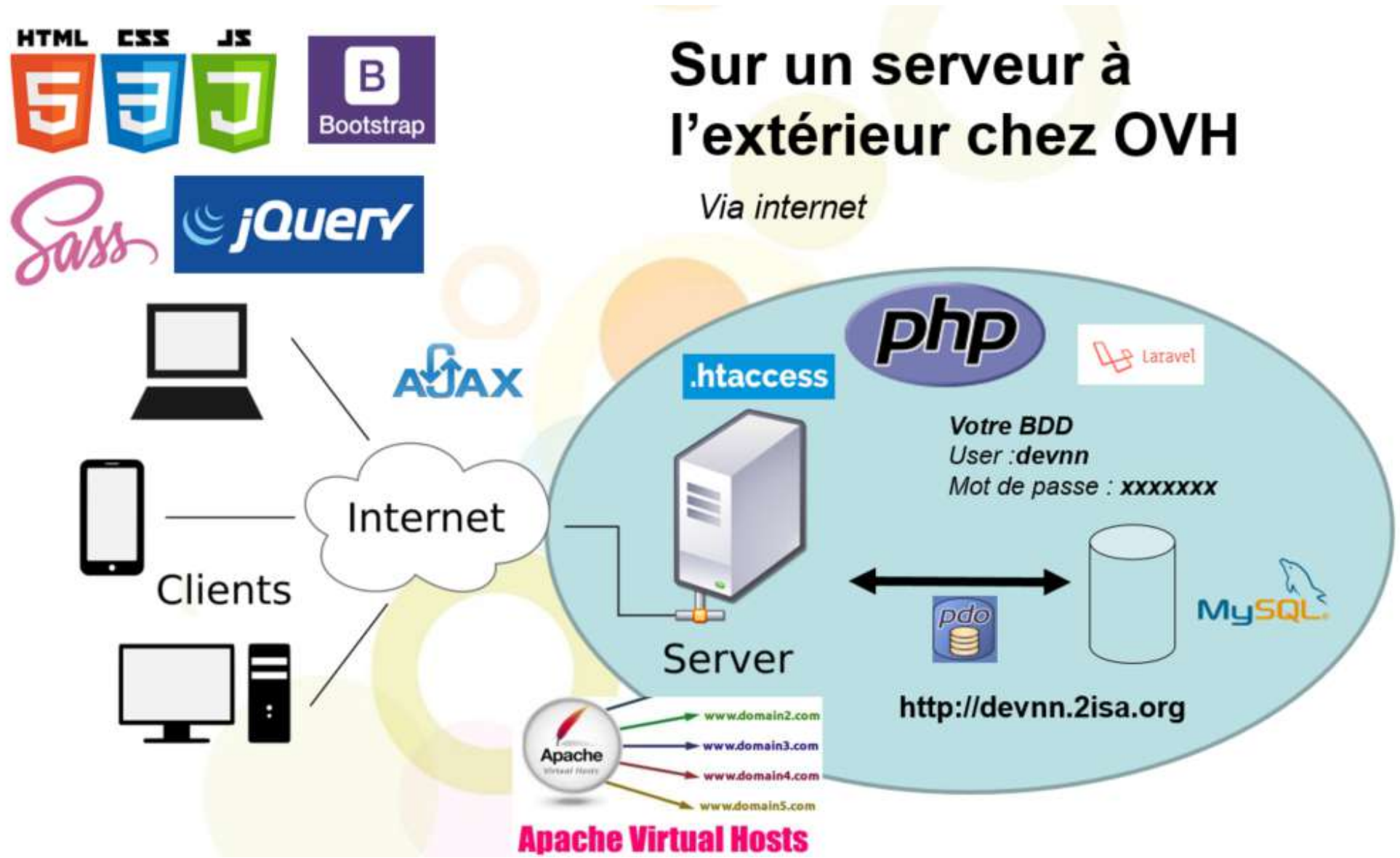
Nous utiliserons des outils :

- PHPSTORM - Utiliser un environnement de développement (IDE)
- Un interpréteur PHP pour l'environnement serveur
- La console Developer Tools de Chrome

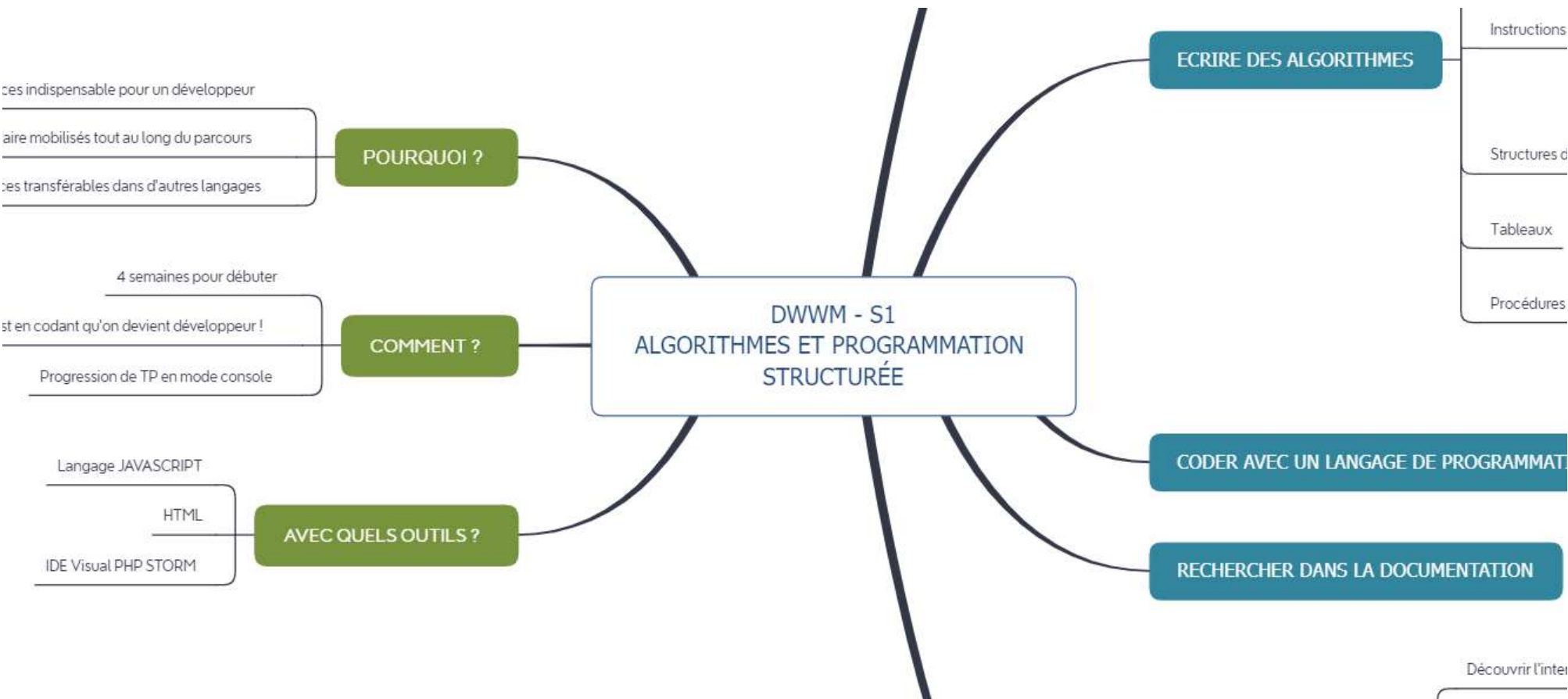
Dans ce Module nous verrons :

- Les variables
- Les opérateurs
- Les conditions
- Les boucles
- Les fonctions
- Les tableaux

3. On se situe où ?

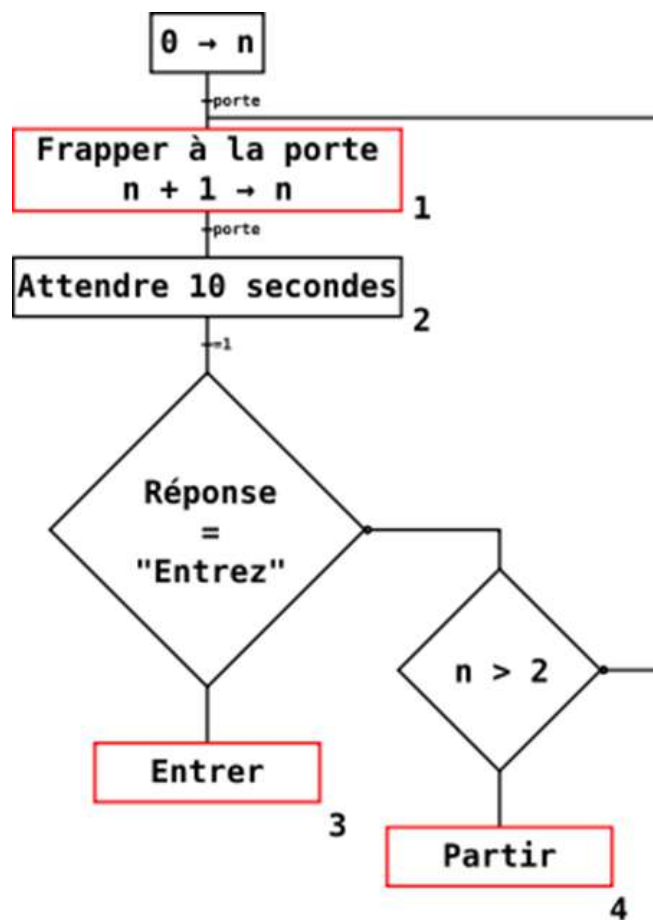


4. Pourquoi, comment ?



5. L'algorithme, c'est quoi ?

- Un algorithme est une procédure de calcul bien définie qui prend en entrée un ensemble de valeurs et qui délivre en sortie un ensemble de valeurs.
- La liste des instructions pour aller d'un point à un autre est une sorte d'algorithme. Ci-contre un exemple avec Google-Map.
- L'entrée est le départ et la destination, la sortie l'itinéraire
- Un algorithme est une méthode générale pour résoudre un type de problèmes. Il est dit correct lorsque, pour chaque instance du problème, il se termine en produisant la bonne sortie, c'est-à-dire qu'il résout le problème posé.
- Un algorithme sera donc dit performant s'il utilise avec parcimonie les ressources dont il dispose, c'est-à-dire le temps CPU, la mémoire RAM et la consommation électrique.



6. Et concrètement ?

- LES VARIABLES – ce qui change ?
- LES OPERATEURS – l'arithmétiques (+, -, * ...) ?
- LES CONDITIONS ?
- LES BOUCLES ?
- LES FONCTIONS ?



Premier script - PHP Storm HelloWorld

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Premier script - PHP Storm HelloWorld

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:52

Table des matières

1. Licence étudiant JetBrains

2. Créer un projet

3. Hello World

4. HTML, CSS, JS, PHP

1. Licence étudiant JetBrains

L'éditeur JetBrains propose une palette d'outils pour les développeurs (IntelliJ IDEA, PhpStorm, WebStorm, ...).

En tant que **stagiaire à 2ISA**, vous pouvez obtenir **une licence d'utilisation** pour tous les produits du « **JetBrains Product Pack for Students** » : IntelliJ IDEA Ultimate, DataGrip, WebStorm, PhpStorm, ReSharper Ultimate, Rider, CLion, AppCode, PyCharm, RubyMine.

Cet avantage est accordé aux personnes qui possèdent **une adresse mail en 2isa.org**.

Pour en bénéficier, rendez-vous sur la page <https://www.jetbrains.com/student/> puis cliquez sur « APPLY NOW » pour vous enregistrer à partir de votre adresse mail (suffixe **2isa.org**).

Free for students: Professional developer tools from JetBrains


Are you learning Java, PHP, Ruby, Python, JavaScript, Objective-C or .NET technologies?


Or maybe you just plan to? Do it right from the start, with award-winning professional developer tools from JetBrains. And the best part: it's free of charge.


All Products Pack


Get access to all desktop products including IntelliJ IDEA Ultimate, ReSharper Ultimate and other IDEs. All you need to apply is to be a student and have access to your student email address or a valid ISiC card.


Find out more in [FAQ](#) below.


**ReSharper Ultimate**
ReSharper, ReSharper C++, dotCover, dotTrace and dotMemory bundled in one license


**AppCode**
Smart iOS/macOS IDE


**PyCharm**
Powerful Python & Django IDE


**IntelliJ IDEA Ultimate**
A complete toolset for JVM-based web, mobile and enterprise development


**CLion**
Cross-platform C/C++ IDE

**Rider**
Cross-platform .NET IDE

**DataGrip**
Database and SQL IDE

**RubyMine**
IDE for Ruby and Rails

**PhpStorm**
IDE for Web & PHP

**WebStorm**
Smart JavaScript IDE

[APPLY NOW](#)
for all products at once!



JetBrains Products for Learning

Apply with:

UNIVERSITY EMAIL ADDRESS

ISiC/ITIC MEMBERSHIP

OFFICIAL DOCUMENT

Status:

☒ I'm a student

☐ I'm a teacher

Name:

nom

prenom

Our software will be registered to your real name.

Email address:

prenom.nom@2isa.org

Your valid university email address, e.g. john.smith@mit.edu.
I confirm that the email address provided above belongs to me.

Country:

France

☐ I have read and I accept the [JetBrains Privacy Policy](#)

APPLY FOR FREE PRODUCTS

Vous obtiendrez **une licence pour tous les produits du Pack pour une durée d'un an renouvelable**.

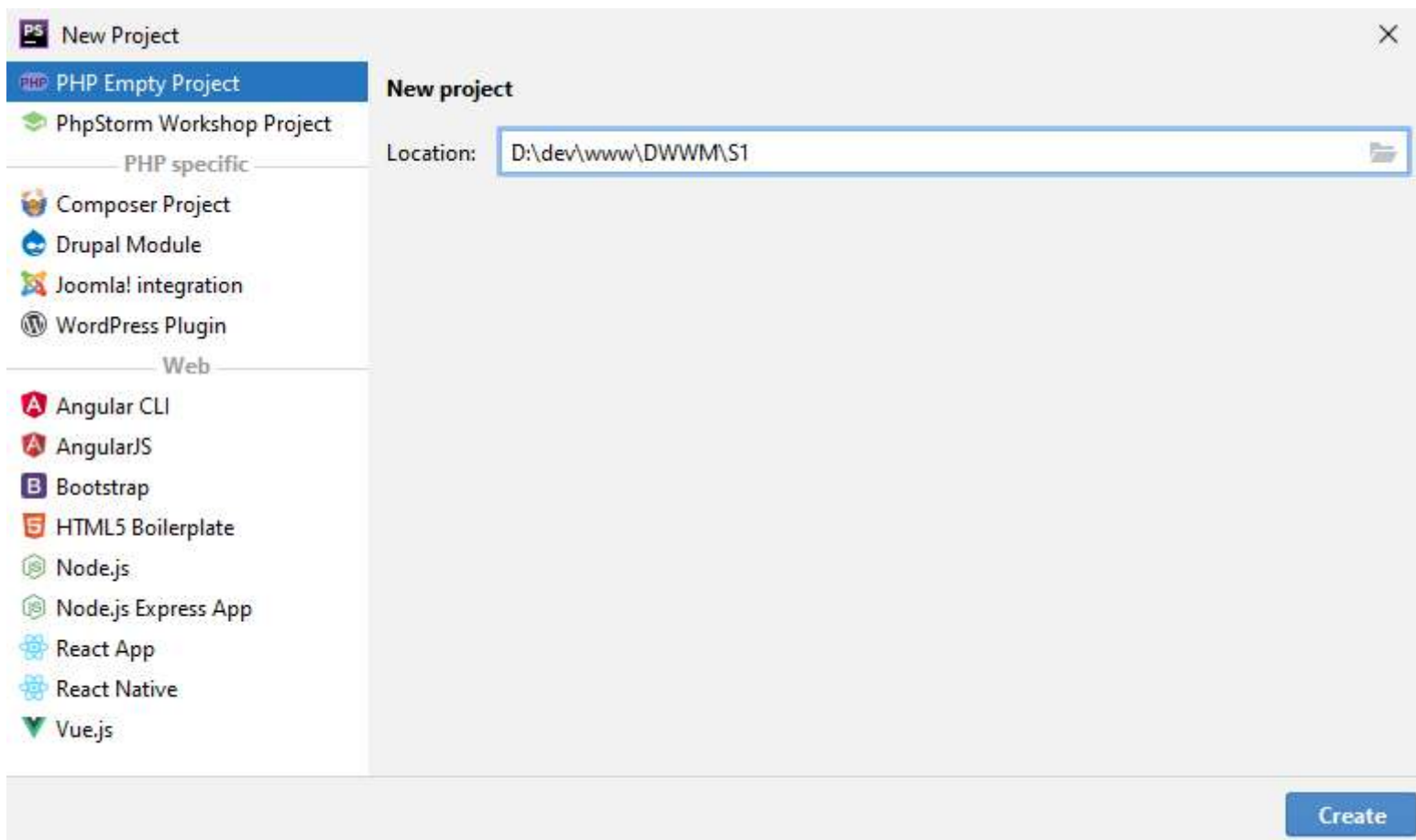
Ainsi, vous ne serez plus limité par les 30 jours de la version d'évaluation.

Vous noterez que l'usage de ces outils est **limité à des fins pédagogiques**.

A vous de jouer !

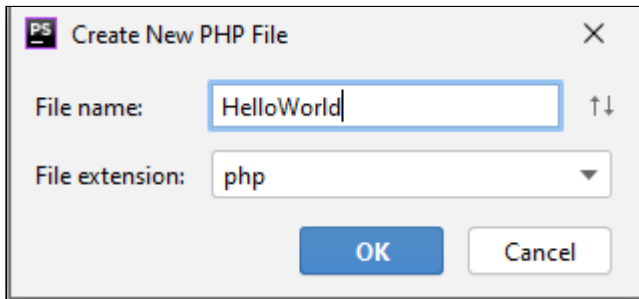
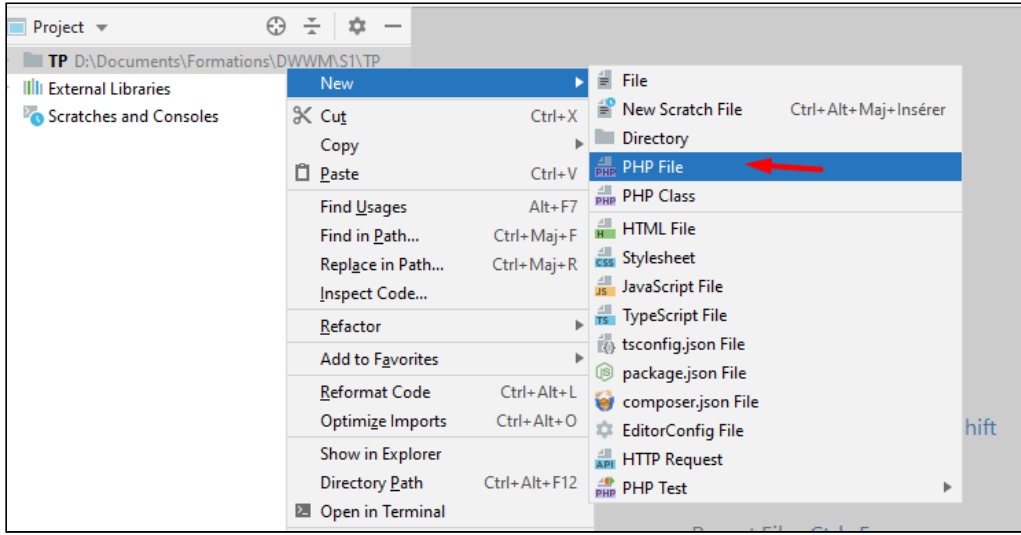
2. Créer un projet

Pour configurer notre environnement de travail nous allons créer un nouveau projet PHP. Choisissez un dossier facile à retrouver, par exemple D:\formations\DWWW\S1 :



3. Hello World

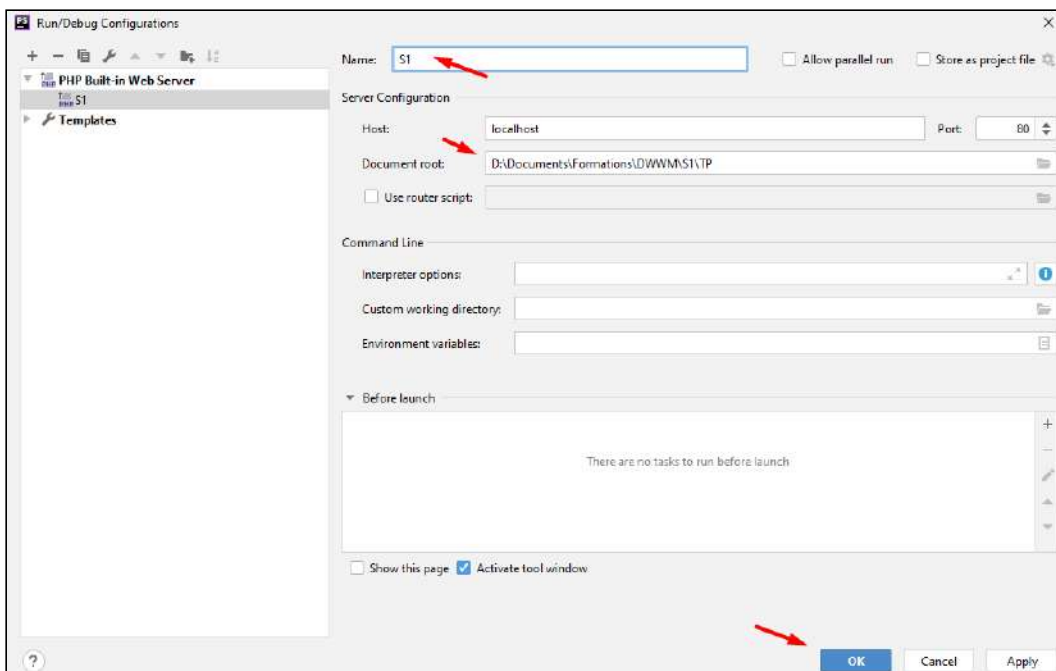
Créer un fichier PHP, appelez le HelloWorld :



Ensuite configurer l'interpréteur, ajouter une configuration :



Ajouter : PHP Built-in Web Server, ajouter un nom et faire OK :



Ajoutons maintenant notre premier code Javascript dans le fichier HelloWorld :

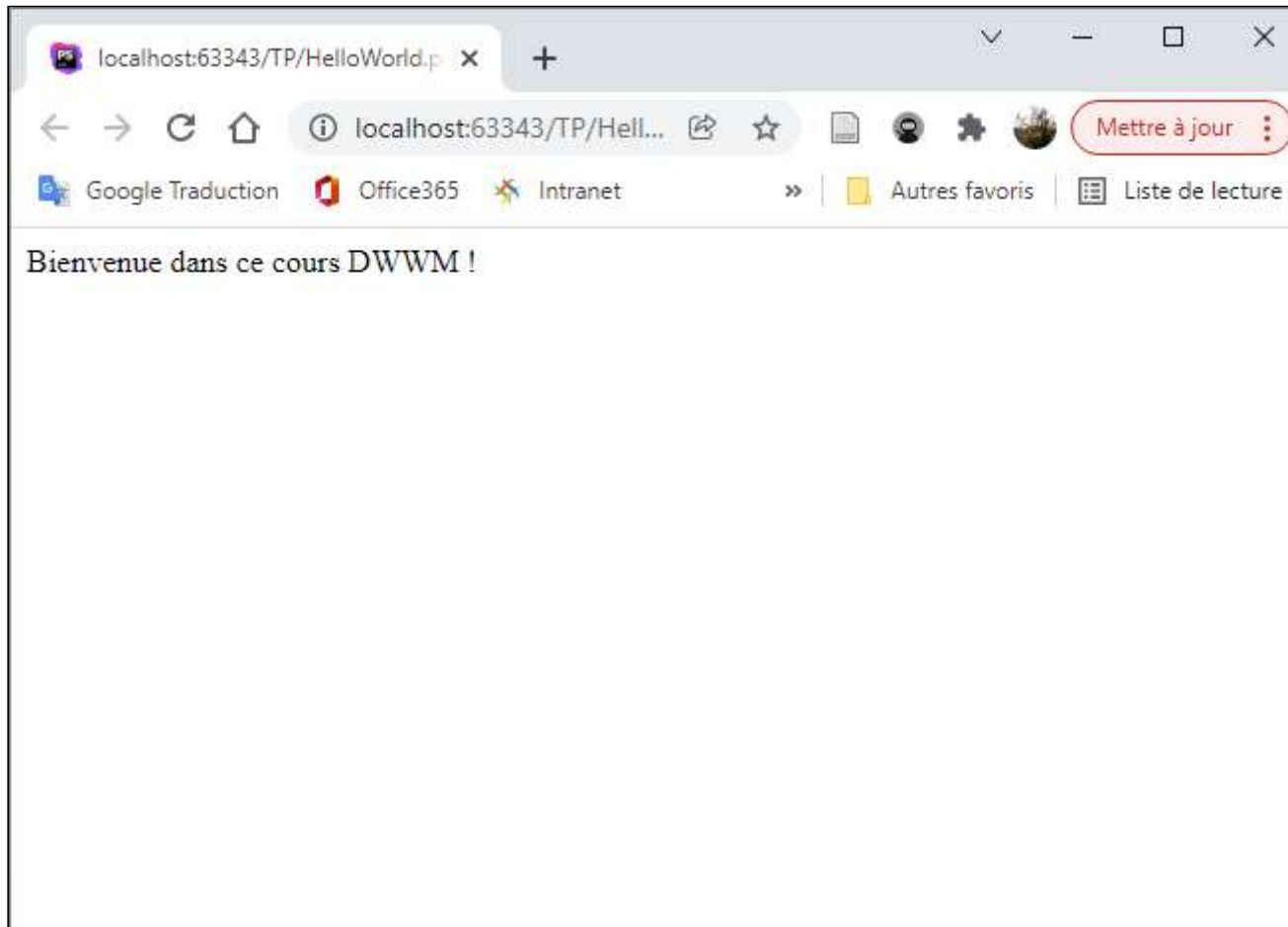
```
<!-- Le code Javascript s'imbrique dans le HTML il s'exécute entre les balises HTML <SCRIPT> -->
<!-- Une simple programme HelloWorld en javascript -->
<script>
    //Code Javascript
    document.writeln("Bienvenue dans ce cours DWWM !");
    //Fin de code Javascript
</script>
```



Pour tester votre code appuyez sur l'icone navigateur :



Votre navigateur devrait s'ouvrir et afficher :



Bravo, vous avez exécuté votre premier code Javascript !

4. HTML, CSS, JS, PHP

Remarquez que nous avons un environnement de travail permettant d'exécuter les 4 codes suivant dans un le même fichier `HelloWorld.php` :

- Le HTML : code de balisage. Langage client.
- Le CSS : code de mise en forme d'une page HTML. Langage client.
- Le JS : code d'algorithmme coté client.
- Le PHP : code d'algorithmme coté serveur.

Voici un exemple de code complet qui interagi avec l'ensemble de ces langages :

```
<!-- code HTML -->
<!-- Balise h1 pour faire un titre avec l'attribut id pour l'identifier -->
<h1 id="titre">Titre de ma page</h1>
<h2>Sous-Titre de ma page</h2>

<style>
  /* code CSS pour mettre en forme, ici je change la couleur */
  #titre{
    color:red;
  }
</style>

<?php
//un simple code PHP pour écrire une ligne
echo "<h3>Voici du code PHP</h3>";
?>

<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  document.writeln("Voici du code JS");
  //Fin de code Javascript
</script>
```

Résultat :

Titre de ma page

Sous-Titre de ma page

Voici du code PHP

Voici du code JS

Utiliser le Debugger

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Utiliser le Debugger

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:52

Description

L'instruction `debugger` permet de faire appel à un outil de débogage (qui peut par exemple permettre de placer un point d'arrêt). Si cette fonctionnalité de débogage n'est pas disponible, l'instruction n'aura aucun effet.

Table des matières

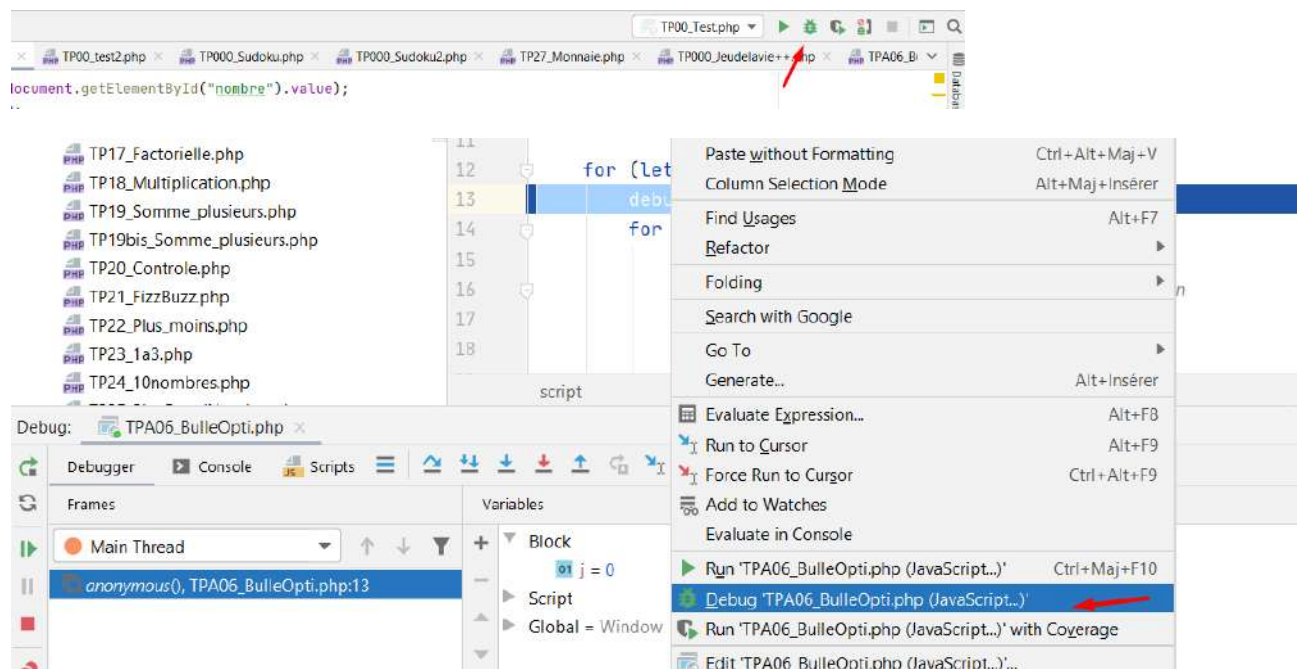
1. Utiliser le Debugger

1. Utiliser le Debugger

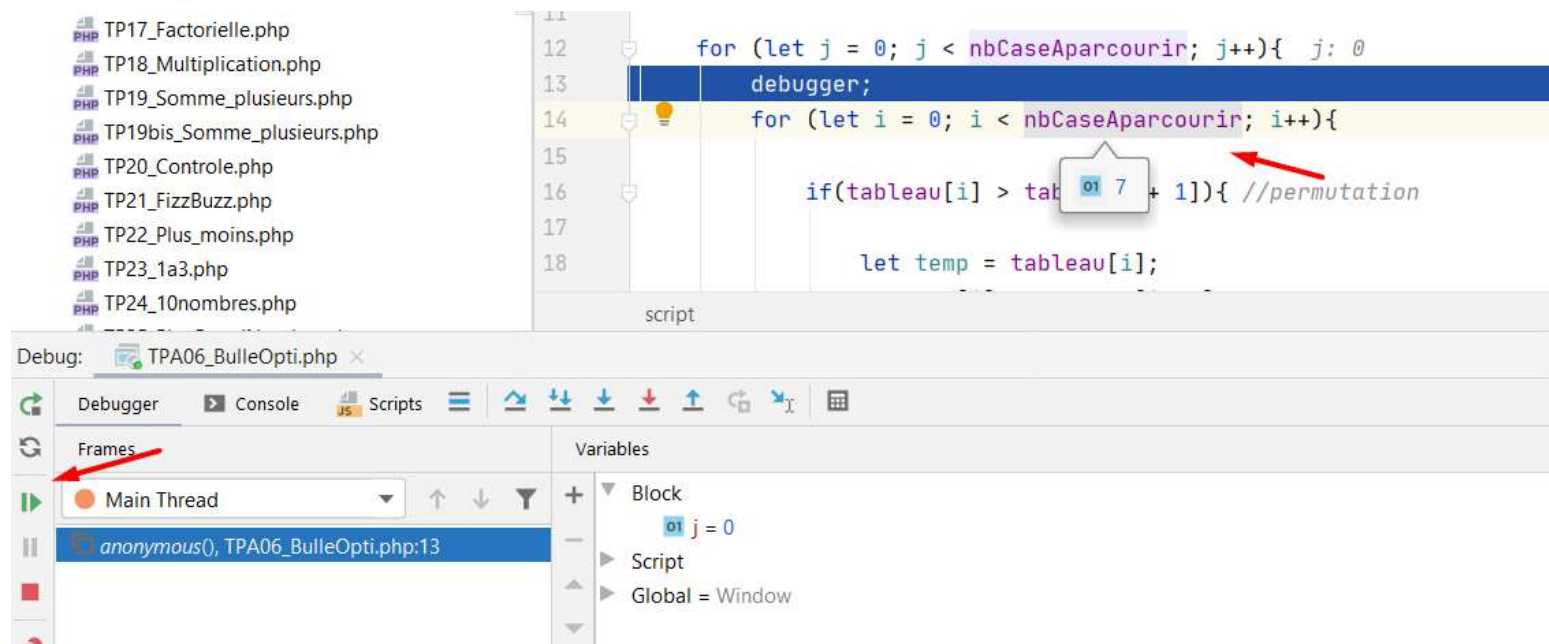
Il est possible de mettre en pause votre programme avec PHPStorm pour suivre l'évolution de votre programme en utilisant l'instruction suivante :

```
debugger;
```

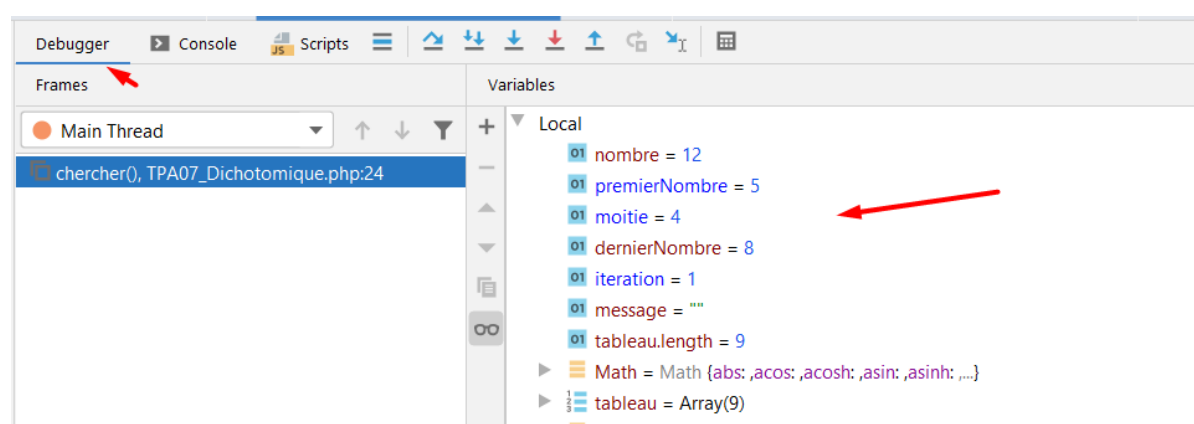
Lancer ensuite votre programme avec l'instruction Debug :



Vous pouvez ensuite suivre l'évolution de votre programme par étape avec la touche F9 (resume program) :



Visualiser l'évolution de vos variables :



Support de cours

Site: [AMIO-FIT](#)
Cours: S1 - Algorithmes et programmation structurée
Livre: Support de cours

Imprimé par: Davy Blavette
Date: mercredi 30 mars 2022, 10:53

Description

Ce support présente les éléments de base à connaître pour **construire des algorithmes** et **débuter avec un premier langage de programmation JAVASCRIPT**.

Il vous guidera pendant tout ce module. Revenez-y souvent au cours de votre apprentissage.

Table des matières

1. Introduction

2. Structure d'un algorithme

2.1. De l'algorithme au programme

2.2. Un exemple

2.3. Les identificateurs

2.4. Les commentaires

3. Les variables

3.1. Qu'est ce qu'une variable

3.2. Définition d'une variable

3.3. Déclarer une variable

3.4. Var, Let et const

3.5. Les constantes

4. Les types fondamentaux

4.1. Les entiers

4.2. Les réels

4.3. Les booléens

4.4. Exercice Morgan

4.5. Solution Morgan

4.6. Les caractères

4.7. Les chaînes de caractères

5. Les opérateurs

5.1. Les expressions

6. Les instructions d'entrée/sortie

6.1. Exercices

7. Les structures de contrôle

7.1. Enchaînement séquentiel

7.2. Instructions conditionnelles

7.3. Instructions de répétitions

7.4. Break et label

8. Les tableaux

8.1. Déclaration d'un tableau

8.2. La boucle For

8.3. Foreach

8.4. La boucle For...IN

8.5. Exercices

9. Les fonctions et procédures

9.1. Définition

9.2. Exemple fonction

9.3. Fonction et objet

9.4. Fonction anonyme

9.5. Exercices

9.6. Passage de paramètres

9.7. Exercices

9.8. Exercices

1. Introduction

Ce document décrit les éléments de base à connaître pour **construire des algorithmes** et **débuter avec un premier langage de programmation JAVASCRIPT**.

Nous aborderons :

- la structure d'un algorithme
- les variables et les constantes
- les types
- les expressions et les opérateurs
- les instructions (entrées-sorties, affectations, structure de contrôle)

Pour **mettre en pratique** ces notions fondamentales, vous réaliserez **une série d'exercices** progressifs que vous coderez en JAVASCRIPT.

2. Structure d'un algorithme

Un **algorithme** est **une suite finie d'opérations** ou instructions permettant de **résoudre un type de problème**.

Un algorithme (comme un programme) est composé de 3 parties principales :

1. La partie **définitions** permet de définir les «entités» qui pourront être manipulées dans l'algorithme : constantes, types, sous-programmes.

2. La partie **déclarations** permet de déclarer les **données** qui sont utilisées par le programme.
Les données sont représentées par des **variables**.

3. La partie **instructions** constitue **le programme principal**.

Les **instructions** sont **interprété par un interpréteur (le navigateur en JS)** pour transformer les **données**.

Un algorithme est **une solution indépendante des langages de programmation** qui définit **la logique d'enchaînement des opérations** pour aboutir au résultat.

Pour être compris par l'interpréteur, **un algorithme** devra être **codé dans un langage de programmation**.

Définition vs déclaration :

Quelque chose qui est **défini** reste **inchangé pendant toute l'exécution** du programme.

Quelque chose de **déclaré** est **amené à changer au cours de l'exécution du programme**

.

2.1. De l'algorithme au programme

Programme

Un ordinateur exécute **des programmes** pour effectuer des tâches.
Vous utilisez des programmes tous les jours :

- Votre navigateur web
- Un traitement de texte (Word)
- Un jeu vidéo

Un ordinateur ne comprend que du **code binaire**.
Mais il serait trop compliqué d'écrire des programmes en code binaire.

Pour **écrire des programmes plus simplement**, on utilise **des langages de plus haut niveau**. Il existe de nombreux langages de programmation.



JAVASCRIPT

C'est **le langage** dans lequel vous allez "parler".
Un langage définit des règles de syntaxe qui lui sont propres.

Compilation vs Interprétation

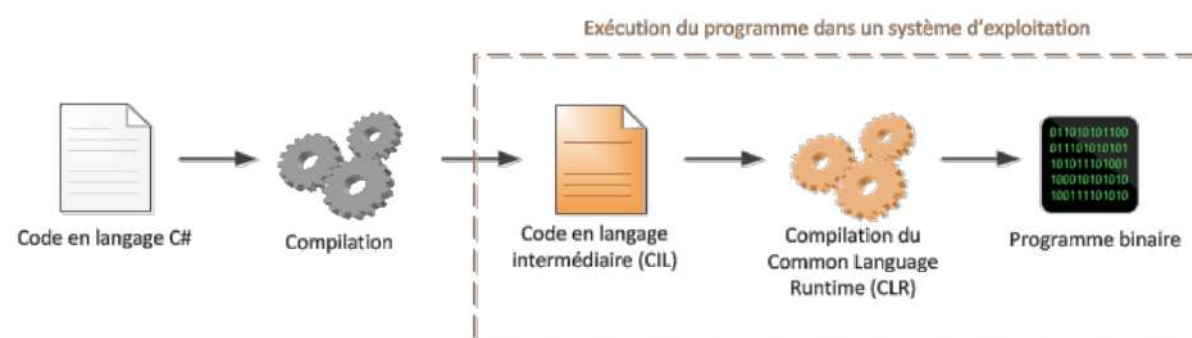
Compilation

Le **code source** doit être **compilé** pour être compris par l'ordinateur.

Dans les langages traditionnels comme C ou C++, la compilation produit directement le code binaire.

En C#, le **code source** est transformé **code intermédiaire** (CIL).
Le code intermédiaire n'est pas exécutable directement. En Windows, il prend la forme d'un .exe.
Pour être exécuté, le code CIL est traduit "à la volée" en code binaire par le CLR (Common Language Runtime) installé sur l'ordinateur.

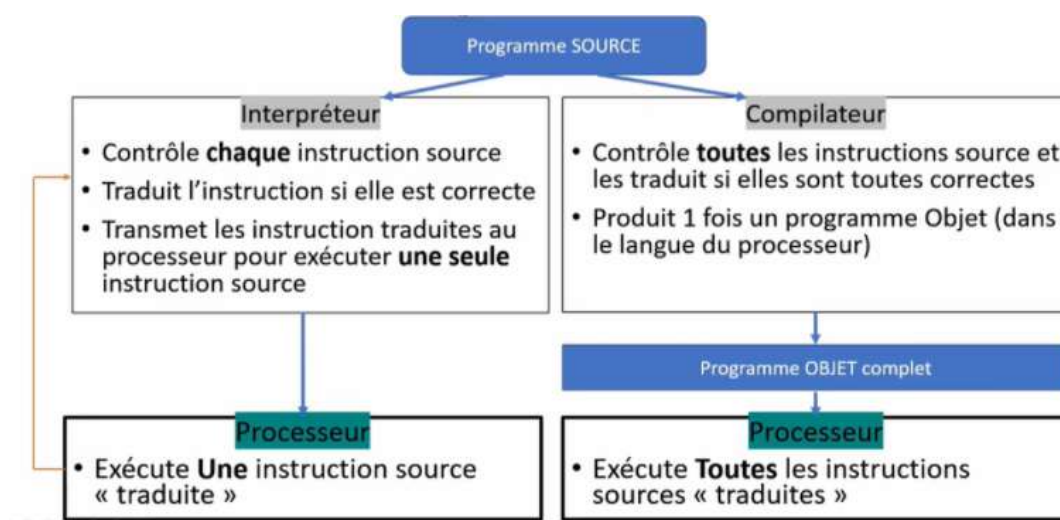
Ainsi le programme est toujours adapté à l'ordinateur sur lequel il tourne.
On dit que **le code est géré**.



Interprétation

La tâche principale d'un compilateur est de produire un code objet correct qui s'exécutera sur un ordinateur. La plupart des compilateurs permettent d'optimiser le code, c'est-à-dire qu'ils vont chercher à améliorer la vitesse d'exécution, ou réduire l'occupation mémoire du programme.

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives et à ce titre est une partie essentielle des applications web. Avec les technologies HTML et CSS, JavaScript est parfois considéré comme l'une des technologies cœur du World Wide Web. Une grande majorité des sites web l'utilisent, et la majorité des navigateurs web disposent d'un moteur JavaScript dédié pour l'**interpréter**, indépendamment des considérations de sécurité qui peuvent se poser le cas échéant.



2.2. Un exemple

Programme en JS

```
<!-- Déterminer Le périmètre d'un cercle à partir de son rayon -->
<script>
  //Code Javascript
  //Définition des constantes

  const PI = Math.PI; //valeur de PI

  //Déclaration des variables
  var rayon = 5;
  var perimetre;

  //Instructions
  perimetre = 2 * PI * rayon;
  document.write("Le périmètre est égale à " + perimetre);

  //Fin de code Javascript
</script>
```

Le périmètre est égale à 31.41592653589793

Dans le programme ci-dessus on retrouve :

- Un commentaire général indique l'objectif du programme (les lignes commençant par *//* sont des commentaires en javascript <!-- en HTML)
- Une partie **Définitions** où l'on retrouve la déclaration d'une constante PI
- Une partie **Déclarations** où sont déclarées deux variables représentant respectivement le rayon et le périmètre
- Une partie **Instructions** regroupant les traitements réalisés par le programme

2.3. Les identificateurs

Toutes les entités qui apparaissent (le programme, les variables, les constantes, les types, les sous-programmes, etc.) **doivent avoir un nom**.

Ce nom permet à l'ordinateur de les distinguer et aux hommes de les comprendre et de les désigner.

Les noms ainsi donnés sont appelés **identificateurs**.

Conventions de nommage

Chaque langage possède ses propres conventions de nommage des identificateurs.

En programmation, on peut retrouver plusieurs conventions de nommage. Voici la listes des plus répandues :

- Le Camel case : les mots sont liés, sans espace. Chaque mot commence par une majuscule, sauf le premier

```
firstName
```

- Le Pascal case : les mots sont liés, sans espace. Chaque mot commence par une majuscule

```
FirstName
```

- Le Kebab case (ou Spinal case) : les mots sont en minuscule et liés entre eux par un tiret

```
first-name
```

- Le Snake case (ou Underscore) : les mots sont en minuscule et liés entre eux par un underscore

```
first_name
```

En JavaScript, nous n'utiliserons que le Camel case `firstName` et le Pascal case `FirstName`.

Elles sont partout. Logique : on s'en sert constamment. Bien les définir est crucial pour comprendre son code !

```
// Bad
let value = "John"
let val = "John"

// Good
let firstName = "John"
```

À la lecture de votre code, si vous avez du mal à comprendre ce que fait une variable, n'hésitez pas à la renommer.

Pour déclarer une variable on préférera le faire en Camel case :

```
// Bad : Pascal case
let FirstName = "John"

// Bad : Kebab case
let first-name = "John"

// Bad : Snake case
let first_name = "John"

// Good : Camel case
let firstName = "John"
```

Les booléens

Pour déclarer des booléens, on ajoute une petite subtilité : un verbe juste devant.

Exemple :

```
// Bad
let visible = true
let admin = true
let online = true
let run = true

// Good
let isVisible = true
let areAdmin = true
let hasOnline = true
let canRun = true
```

Les constantes

Vous vous souvenez, je vous ai dit qu'en JavaScript on utilisait seulement le Camel case et le Pascal Case ? Et bien les constantes sont un peu l'exception à cette règle. Mais rien de bien méchant : les mots sont en majuscules séparés par un underscore.

```
// Bad
const time = 60
const daysInYear = 365

// Good
const TIME = 60
const DAYS_IN_YEAR
```

Les fonctions

Les fonctions utilisent le Camel case et le même patern que les booléens : elles utilisent un verbe.

```
// Bad
let user() => {
  ...
}

// Good
let getUser() => {
  ...
}
```

Les classes

Les classes sont déclarées en Pascal case.

Pourquoi... ? Parce qu’elles ont la classe ! Ou parce que le concept de classe est plus fort qu’une déclaration de variable !

```
// Bad
classe userRole {
  ...
  getUser() {
    ...
  }
}

// Good
classe UserRole {
  ...
  getUser() {
    ...
  }
}
```

2.4. Les commentaires

Les **commentaires** sont utilisés par les développeurs pour ajouter des informations à leur code.

L'ordinateur ne lit pas les commentaires, ils sont utilisés pour donner les informations nécessaires aux humains pour comprendre le code.

Le langage C# permet d'utiliser deux types de commentaires :

Les commentaires ligne introduits par **//**

- Utilisés **pour faire apparaître les raffinages** dans la structure de l'algorithme.
Ils précèdent les instructions qu'ils décrivent et sont alignés.
- Utilisés **pour expliquer ce qui a été fait**.
Ils sont alors placés à la fin de la ligne et ont un rôle de justification.
- Utilisés **pour expliquer le rôle d'une variable ou d'une constante**

Les commentaires bloc entourés par **/* et */**

- Utilisés **pour faire apparaitre les différentes parties d'un programme**.



Commentez pour vous-même pour comprendre votre code quand vous le reprendrez plus tard
Pour permettre à **d'autres développeurs** de comprendre le code et de le maintenir.

En **HTML** les commentaires seront définis par **<!-- et -->**

3. Les variables

Dans cette partie nous allons nous intéresser à la notion de **variable**.

3.1. Qu'est ce qu'une variable

Un algorithme est fait pour **résoudre un ensemble de problèmes semblables** (cf définition du terme « algorithmique » dans le petit Robert).

Par exemple, un logiciel qui gère la facturation d'une entreprise doit connaître les noms et adresses des clients, les produits commandés, etc.

Ces informations ne peuvent pas être devinées par le programme et doivent donc être saisies par les utilisateurs.
Ces informations sont appelées **données en entrée**. Elles proviennent de l'extérieur du programme et sont utilisées dans le programme.

Inversement, le programme effectue des opérations, des calculs dont les résultats devront être transmis aux utilisateurs.

Par exemple :

- le total d'une facture est calculé comme la somme de tous les produits commandés en multipliant le nombre d'articles par leur prix unitaire ;
- le total des ventes pour une période donnée est obtenu en faisant la somme des montants des factures, etc.

Ces informations seront affichées à l'écran, imprimées, stockées dans des bases de données ...
Ces informations qui sortent du programme sont appelées **données en sortie**.

Le programmeur utilise **des variables** pour représenter **les données manipulées par un programme** :

- **les données en entrée**
- **les données en sortie**
- et aussi les données permettant de faire des **calculs intermédiaires**.

Ainsi, pour calculer le montant d'une ligne d'une facture, le programmeur expliquera que le montant est obtenu en multipliant le prix unitaire par la quantité d'articles commandés, il utilisera 3 variables. :

```
prixUnitaire; // Le prix unitaire de l'article
quantite; // La quantité d'articles commandés
montant; // Le montant de la ligne du bon de commande
```

3.2. Définition d'une variable

Une **variable** peut être vue comme **une zone dans la mémoire** (vive) de l'ordinateur qui est utilisée **pour conserver les données qui seront manipulées par le programme**.

Une **variable** est caractérisée par quatre informations :

1. Son **rôle** : il indique **à quoi va servir la variable** dans le programme.

Par exemple, on peut utiliser une variable pour conserver le prix unitaire d'un article, exprimé en euros.

Cette information doit apparaître dans l'algorithme sous la forme d'un commentaire informel (texte libre) associé à la variable.

2. Son **nom** : il permet d'identifier la variable. On parle d'**identificateur**.

Ce nom doit être significatif (refléter le rôle de la variable).

Un compromis doit bien sûr être trouvé entre expressivité et longueur.

On peut par exemple appeler `prixUnitaire` une variable qui représente le prix unitaire d'un article, `prixUnitaireDunArticle` est beaucoup trop long.

3. Son **type** : caractérise **l'ensemble des valeurs** qu'une variable peut prendre.

Une variable est utilisée pour représenter des données qui sont manipulées par le programme.

Par exemple : le prix unitaire représente le prix d'un article exprimé en euros.

On peut considérer qu'il est représenté par un réel. Il ne peut alors prendre que ce type de valeurs.

De la même manière, la quantité ne peut prendre que des valeurs entières et le `nom` est une chaîne de caractères. On dira respectivement que le type de `prixUnitaire` est **réel (float, double ou decimal en C#)**, le type de `quantite` est **entier (int en C#)** et le type de `nom` est **string** en C# (chaîne de caractères). En **Javascript** il n'y a pas besoin de définir le type de variables, c'est le compilateur qui déterminera automatiquement son type. Les trois types primitif en javascript sont **String, Number et Boolean**.

4. Sa **valeur** : une variable contient **une information qui peut varier au cours de l'exécution** d'un programme.

C'est cette information que l'on appelle **valeur de la variable**.

La valeur d'une variable doit correspondre au type de la variable.

Ainsi, une variable quantité de type entier pourra prendre successivement les valeurs de 10, 25 ou 3.

La valeur d'une variable n'est connue que lorsque le programme est exécuté.

Règle : Une variable doit toujours être initialisée avant d'être utilisée.

Les autres informations (**nom**, **rôle** et **type**) sont définies lors de la conception du programme, pendant la construction de l'algorithme.

Le **rôle**, le **nom** et le **type** sont des informations *statiques* qui doivent être précisées lors de la déclaration de la variable.

En revanche, la **valeur** d'une variable est une information *dynamique* qui changera au cours de l'exécution du programme.

Attention : Le nom d'une variable doit être significatif : il doit suggérer, si possible sans ambiguïté, la donnée représentée par cette variable. En général, dès qu'on déclare une variable, il faut mettre un commentaire pour expliquer ce qu'elle représente et le rôle qu'elle joue dans l'algorithme.

L'opérateur **typeof** renvoie une chaîne qui indique le type de son opérande :

```
console.log(typeof 42);
// expected output: "number"

console.log(typeof 'blubber');
// expected output: "string"

console.log(typeof true);
// expected output: "boolean"

console.log(typeof undeclaredVariable);
// expected output: "undefined"
```

3.3. Déclarer une variable

Une variable pour faire simple est un conteneur (associé à un nom) qui garde des valeurs. Ce conteneur sera disponible durant toute l'exécution de notre programme. En mathématiques, on connaît une variable comme étant une lettre qui représente un nombre, en programmation aussi c'est le même fonctionnement.

Pour déclarer une variable, on utilise le mot clé `let` suivi du nom de la variable puis l'opérateur `=` et la valeur de la variable

```
let age = 18
```

Je déclare une variable *age* qui à la valeur 18.

En Javascript, on peut utiliser 3 mots clé pour déclarer une variable: `let`, `const` et `var`.

`let` et `var` permettent de déclarer des variables qui peuvent changer durant l'exécution du programme.

`const` quand à lui permet de déclarer des variables qui ne changeront pas durant l'exécution du programme, ce sont des données immuables (constantes).

Javascript a été créé à l'origine avec un seul mot-clé pour définir une variable, ce mot réservé est : `var`.

Mais depuis la version 6 (ECMAScript 2015) de la spécification du langage, deux nouveaux mot-clés sont apparus : `let` et `const`.

Si l'on devait résumer la définition de ces deux nouvelles manières de déclarer des variables, on pourrait dire que :

- `let` sert à définir une variable locale
- `const` sert à déclarer une référence constante

3.4. Var, Let et const

Il existe trois manières de déclarer des variables en Javascript, mais quelles sont les vraies différences ?

```
var age = 18;  
let age = 18;  
const age = 18;
```

	var	let	const
Stockée en global	✓	✗	✗
Portée de la fonction	✓	✓	✓
Portée du block	✗	✓	✓
Ré-assignation	✓	✓	✗
Re-déclaration	✓	✗	✗
Hoisting	✓	✗	✗

Stocké en global ?

- var : oui ✓
- let : non ✗
- const : non ✗

Lorsque var est utilisé pour déclarer une variable en dehors d'une fonction, alors cette variable sera forcément référencée dans l'objet global du script.

Se limite à la portée d'une fonction ?

- var : oui ✓
- let : oui ✓
- const : oui ✓

La portée (ou scope) d'une fonction est la seule à mettre toutes les variables sur un même pied d'égalité et déclarer une variable à l'intérieur d'une fonction n'aura logiquement pas d'incidence sur le reste des données du code.

Se limite à la portée d'un block ?

- var : non ✗
- let : oui ✓
- const : oui ✓

Pour ceux qui se demanderaient ce qu'est un bloc, c'est simplement l'ensemble des instructions comprises entre deux accolades "{...}" sauf pour lorsque ces accolades délimitent une fonction ou un objet JSON.

Un bloc d'instruction se trouve souvent après un if, else, for, while, etc. Donc souvenez-vous qu'une variable déclarée avec le mot clé "var" dans un for sera automatiquement globale.

Peut être réassigné ?

- var : oui ✓
- let : oui ✓
- const : non ✗

Réassigner signifie que l'on va soit modifier la valeur de la variable lorsque l'on joue avec les types primitifs, soit que l'on va modifier la référence de la variable lorsque l'on utilise un type complexe.

Peut être redéclaré ?

- var : oui ✓
- let : non ✗
- const : non ✗

Redéclarer une variable (sauf lorsque sa portée le permet) est impossible sauf si c'est avec var, mais ce n'est pas une bonne pratique et il est conseillé d'éviter de le faire.

Est affecté par le hoisting ?

- var : oui ✓
- let : non ✗

- const : non ❌

Le hoisting est une notion plus complexe qui intervient au moment de l'interprétation du code JS. Cette mécanique consiste à faire "virtuellement" remonter la déclaration d'une variable (ou d'une fonction) tout en haut de son scope lors de l'analyse du code par le moteur d'interprétation Javascript.

3.5. Les constantes

Certaines informations manipulées par un programme **ne changent jamais**, ce sont des **constantes**.

Par exemple : la valeur de π , le nombre maximum d'étudiants dans une promotion, etc.

Plutôt que de mettre explicitement leur valeur dans le code du programme (constantes littérales), il est préférable de leur donner un nom symbolique (et significatif). On parle alors de constantes (symboliques).

Par convention, les constantes sont nommées en MAJUSCULES.

Exemples (en JS) :

```
const PI = 3.1415; // valeur de PI
const MAJORITE = 18; // age correspondant à la majorité
const TVA = 19.6; // taux de TVA en vigueur
const CAPACITE = 120; // Nombre maximum d'étudiants dans une promotion
const INTITULE = "Algorithmique et Programmation"; // intitulé de la formation
```

π peut être considérée comme une constante absolue.

Les constantes MAJORITE, TVA, CAPACITE sont utilisées pour paramétrer le programme.
Ces valeurs ne peuvent pas changer au cours de l'exécution d'un programme.

L'utilisation des **constantes** rend le **code plus lisible** et **facilite la maintenance**.

En effet, si un changement doit être réalisé (par exemple, la précision de PI utilisée n'est pas suffisante), il suffirait de changer la valeur de la constante symbolique au moment de sa définition et de recompiler le programme pour que la modification soit prise en compte dans le reste de l'algorithme.

4. Les types fondamentaux

Un **type** caractérise les valeurs que peut prendre une **variable**.

Il définit également les opérations, généralement appelées **opérateurs**, qui pourront être appliquées sur les données de ce type.

Les types fondamentaux sont les types que les différents langages de programmation proposent par défaut :

- Les **Entiers** : nombres entiers (sans virgule)
- Les **Réels** : nombres à virgule
- Les **Booléens** : 2 valeurs possibles *VRAI* ou *FAUX*
- Les **Caractères** : lettre, chiffre, caractère spécial ...
- Les **Chaînes de caractères** : une suite de caractères.

Les types ont deux intérêts principaux :

- Permettre de **vérifier automatiquement** (par le compilateur) **la cohérence de certaines opérations**.

Par exemple, une valeur définie comme entière ne pourra pas recevoir une valeur chaîne de caractères.

- **Connaître la place nécessaire pour stocker la valeur de la variable**.

Ceci est géré par le compilateur du langage de programmation et est en général transparent pour le programmeur.

Opérateur : à chaque type est associé un ensemble d'opérations (somme, différence, multiplication, etc...)

Opérateurs de comparaison : tous les types présentés plus haut sont munis de certaines des opérations de comparaison suivantes : <, >, <=, >=, == et !=

Remarque différence entre == et ===

```
let a = 12;
let b = "12";

console.log(a == b); // True
console.log(a === b); // False vérification avec typage
```


4.1. Les entiers

Le type **entier** caractérise **les entiers relatifs** (entiers signés, positifs ou négatifs).

Exemples : 10, 0, -9

Les **opérateurs** disponibles **sur les entiers** sont :

+ (somme)

- (différence)

***** (multiplication)

/ (renvoie le quotient de la division entière)

% (renvoie le modulo, c'est à dire le reste de la division entière)

Les opérateurs + et - peuvent être *unaires* ou *binaires*, c'est à dire qu'ils peuvent avoir une ou deux opérandes.

Exemples :

10 % 3 renvoie 1

10 / 3 renvoie 3

1 / 2 renvoie 0

4.2. Les réels

Le type **réel** caractérise **les nombres à virgule**.

Exemple : 10.1 ; 5.7; -4.4 ; 0.125

Les opérateurs disponibles sont : **+** **-** ***** **/**

Les opérateurs + et - peuvent être unaires ou binaires.

4.3. Les booléens

Le type **booléen** caractérise les valeurs booléennes qui ne peuvent prendre que deux valeurs, *VRAI* ou *FAUX*.

Les opérations sont **Et**, **Ou** et **Non** définies par la table de vérité suivante :

A	B	A Et B	A Ou B	Non A
VRAI	VRAI	VRAI	VRAI	FAUX
VRAI	FAUX	FAUX	VRAI	FAUX
FAUX	VRAI	FAUX	VRAI	VRAI
FAUX	FAUX	FAUX	VRAI	VRAI

Lois de De Morgan

Non (A **Et** B) = (**Non** A) **Ou** (**Non** B)

Non (A **Ou** B) = (**Non** A) **Et** (**Non** B)

Non (**Non** A) = A

Exercice

En utilisant des tables de vérité, vérifiez que les lois de Morgan sont correctes.

4.4. Exercice Morgan

A partir du code ci-dessous, réaliser une fonction morgan(a,b).

```
//Code Javascript
let a = true;
let b = true;

console.log(a && b); //True
console.log(a || b); //True
console.log(!a); //False
console.log("--MORGAN--");
console.log(!(a && b) == !a || !b); //true
console.log(!(a || b) == (!a && !b)); //true
console.log(!(!a) == a); //true
console.log("--FIN MORGAN--");

a = true;
b = false;

console.log(a && b); //False
console.log(a || b); //True
console.log(!a); //False
console.log("----");
a = false;
b = true;

console.log(a && b); //False
console.log(a || b); //True
console.log(!a); //True
console.log("----");
a = false;
b = false;

console.log(a && b); //False
console.log(a || b); //False
console.log(!a); //True
console.log("----");
```

4.5. Solution Morgan

```
//Code Javascript
document.writeln(morgan(true, true));
document.writeln(morgan(true, false));
document.writeln(morgan(false, true));
document.writeln(morgan(false, false));

function morgan(a, b) {
    let loi1 = !(a && b) == !a || !b;//true
    let loi2 = !(a || b) == (!a && !b);//true
    let loi3 = !(!a) == a;//true

    return "<h4>Loi1 = " + loi1 + " - Loi2 = " + loi2 + " - Loi3 = " + loi3 + "</h4>";
}
```

4.6. Les caractères

Le type **caractère** représente les caractères. On les entoure toujours par '' (simples quotes)

Exemples : 'a' , 'B' , '4' , '!' , '_' , ''

Il est possible d'utiliser le caractère d'échappement \ pour afficher des caractères spéciaux, par exemple :

- \" : permet d'afficher le caractère '

```
document.write('je m\'appelle Jean');
```

- \" : permet d'afficher le caractère \

```
document.write('La barre oblique inversée \\ et connue également sous les appellations backslash ou antislash');
```

- 'n' : retour à la ligne

- 't' : permet d'afficher une tabulation

```
document.write('<pre>Est il possible de faire un retour \n à la ligne et une \t tabulation ?</pre> \n Remarque vous devez utiliser dans ce cas la balise HTML "pre".');
```

Remarque : Il est possible d'utiliser l'accent grave `` pour entourer un string. Ceci nous permet d'y inclure directement la variable.

```
//Code Javascript
let maVariable = 'une chaine exemple';
document.getElementById('pre').innerHTML = `test ${maVariable}`;
```

4.7. Les chaînes de caractères

Le type **string** caractérise les chaînes de caractères.
Une valeur de type string est entourée de guillemets (" ").

Exemples :

"une chaîne de caractères"

"ABC"

"12"

"Une chaîne avec des des guillemets (\")"

Attention : il ne faut pas confondre le nom d'une variable et une chaîne de caractères !

5. Les opérateurs

Les opérateurs sont des symboles qui permettent d'exécuter des opérations dans un algorithme ou programme informatique. Par exemple si on veut effectuer une addition de deux variables entiers ou réels on utilise l'opérateur d'addition connu par le célèbre symbole +.

En algorithmique, on dénombre généralement 4 familles d'opérateurs:

- **Opérateurs d'assignation (ou affectation):** Ce sont des opérateurs qui permettent d'affecter (assigner) une valeur à une variables.

```
let pomme = 10; // = est un opérateur d'assignation mais aussi...
pomme += 10;    // += autoincrémente rayon de 10
pomme -= 10;    // -= décrémente rayon de 10
pomme *= 10;    // *= multiplie rayon de 10
pomme /= 10;    // /= divise rayon de 10

document.write('Pomme est de : ' + pomme); //+ est un opérateur de concaténation
//Pomme est de : 10
```

- **Opérateurs de calcul (ou arithmétiques):** Ce sont des opérateurs qui permettent d'effectuer des opérations de calcul comme l'addition ou le produit.

```
//opérateurs arithmétiques (plus, moins, multiplié, divisé, modulo)
let pomme = 10;
let cerise = 5;
let panier = pomme + cerise * cerise / pomme - cerise % pomme;

document.write('Panier est de : ' + panier);
//Panier est de : 7.5
```

Multiplication (*) et de la division (/) ont une priorité plus élevée que l'addition (+) et la soustraction (-) .

- **Opérateurs de comparaison:** Ce sont des opérateurs qui permettent de comparer deux éléments (variables, constantes ou valeurs statiques).

```
//Opérateurs de comparaison
let pomme = 10;
let cerise = 5;
let banane = 5;

document.writeln(pomme < cerise); //false
document.writeln(pomme > cerise); //true
document.writeln(pomme == cerise); //false
document.writeln(cerise == banane); //true
document.writeln(cerise != banane); //false
document.writeln(pomme >= banane); //true
```

- **Opérateurs logiques:** Ce sont des opérateurs qui permettent d'effectuer des opérations logiques comme le ET logique ou le OU logique...

```
document.writeln(true && true == true); //true
document.writeln(true && false == true); //false
document.writeln(false && false == true); //false
document.writeln(true || false == false); //true
document.writeln(false || true == false); //false
```

[En savoir+ sur les opérateurs](#)

5.1. Les expressions

Définition

Une **expression** est une combinaison de variables ou de littéraux, d'opérateurs (arithmétiques, comparaison, logiques) qui **est évaluée** selon les règles de priorité **pour produire une valeur**.

```
10.0           // une constante littérale
PI             // une constante symbolique
rayon          // une variable de type réel
2 * rayon      // l'opérateur * appliqué sur 2 et rayon
2 * PI * rayon // une expression mêlant opérateurs, constantes et variables
rayon >= 0     // expression avec un opérateur de comparaison
```

Validité d'une expression

Pour qu'une **expression** soit **valide**, **les types de ses opérandes** doivent être **compatibles**.
Par exemple, faire l'addition d'un entier et d'un booléen n'a pas de sens.

- Des **opérandes** ayant le **même type** sont **compatibles**
- Le type A est compatible avec le type B si et seulement si le passage d'un type A à un type B se fait **sans perte d'information**.

En d'autres termes, tout élément du type A a un correspondant dans le type B.

Par exemple, entier est compatible avec réel mais l'inverse est faux.

Exemple : Soit l'expression $n + x$ avec n entier et x réel
Il y a « coercion » : l'entier n est converti en un réel puis l'addition est réalisée sur les réels, le résultat étant bien sûr réel.

Evaluation d'une expression

Une expression est évaluée **en fonction de la priorité des opérateurs** qui la composent, **en commençant par ceux de priorité plus forte**.
A **priorité égale**, les opérateurs sont évalués **de gauche à droite**.

Priorité des opérateurs

Du plus fort au plus faible

1. . (accès à un champ)
2. +, -, Non (unaires)
3. *, /, Div, Mod, Et
4. +, -, Ou
5. <, >, <=, >=, ==, !=

Toujours possible de mettre des **parenthèses pour modifier les priorités**.

Exemple :

Calculer le prix TTC d'un article

PrixHT * (1 + TVA)

En cas de doute ou pour la clarté, mettre des parenthèses.

6. Les instructions d'entrée/sortie

Opération de sortie

Affiche les résultats d'un programme sur une page HTML ou la console.

L'instruction se fait en 2 temps :

- Evaluer l'expression à afficher
- Afficher sur le périphérique de sortie la valeur de l'expression

Pour afficher un résultat javascript vous pouvez l'afficher dans une page HTML ou sur la console de l'outil de développement (DevTools de Chrome).

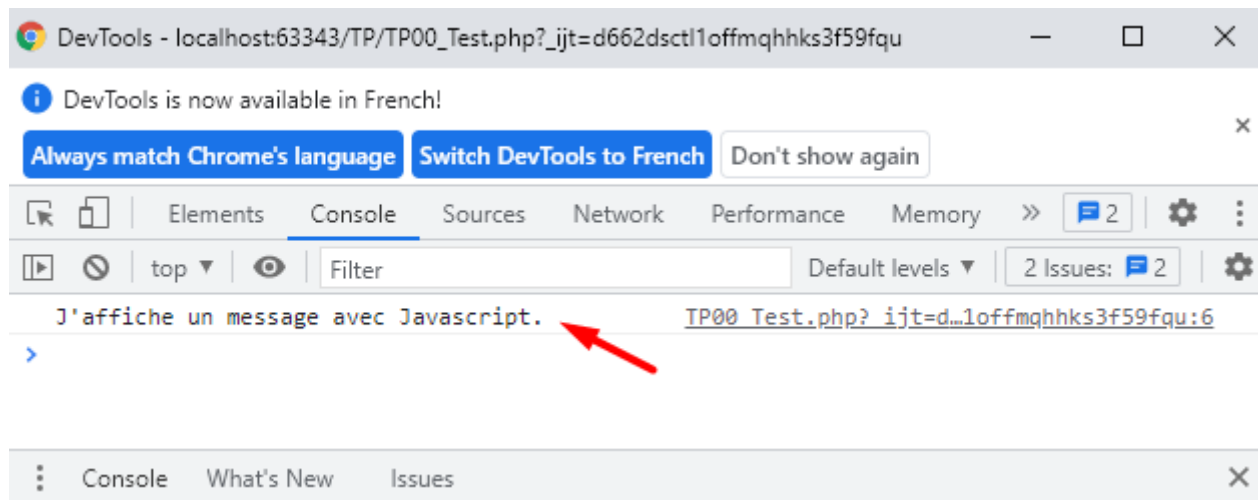
Pour un affichage HTML on manipule le DOM, Le **Document Object Model (DOM)** est une interface de programmation normalisée par le W3C, qui permet à des scripts d'examiner et de modifier le contenu du navigateur web. Le DOM en javascript ce fait par l'objet document, on peut ensuite manipuler le document comme dans cette exemple :

```
<!-- Un élément de paragraphe HTML destiné à restituer Les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //L'objet document (DOM) puis récupérer L'ement d'id 'message' puis appliquer La methode innerHTML qui consiste à insérer du contenu.
  document.getElementById("message").innerHTML = "J'affiche un message avec Javascript.";
</script>
```

On peut aussi afficher un élément dans la console du navigateur avec `console.log()` :

```
<script>
  //Afficher dans la console devtools
  console.log("J'affiche un message avec Javascript.");
</script>
```

La console devtools dans chrome s'affiche dans votre navigateur avec la touche **F12**.



Opération d'entrée

Certaines données en entrée du programme sont **saisies par l'utilisateur** et **lues sur le périphérique d'entrée** (console) **au moment de l'exécution du programme**.

L'instruction se fait en 2 temps :

- Récupérer la saisie sur le périphérique d'entrée
- Ranger cette information dans une variable afin de l'utiliser dans le programme

```
<label for="name">Saisir un nombre:</label>
<input type="text" id="prenom" name="prenom" required>
<button id="button">Valider</button>
<!-- Un élément de paragraphe HTML destiné à restituer Les résultats -->
<p id="message"></p>
<!-- Un simple code en javascript pour écrire une ligne -->
<script>
  //Code Javascript
  //Ajout d'un evenement de type click sur Le bouton
  document.getElementById("button").addEventListener("click",message);

  //Fonction
  function message() {
    //variable prenom
    let prenom = document.getElementById("prenom").value;
    //Affichage du message dans la zone paragraphe id message
    document.getElementById("message").innerHTML = "Bonjour " + prenom;
  }

  //Fin de code Javascript
</script>
```



Remarque : Nous utilisons ici une fonction car nous créons un événement de type click sur le bouton.

6.1. Exercices

Exercices

- [Hello World](#)
- [Cube d'un nombre](#)
- [Périmètre d'un cercle](#)

7. Les structures de contrôle

Dans un langage impératif, on définit **l'état d'un programme en cours d'exécution** par deux choses :

- l'ensemble des valeurs des variables du programme
- l'instruction qui doit être exécutée.

L'exécution d'un programme est alors **une séquence d'affectations** qui font **passer d'un état initial** (les valeurs des variables sont indéterminées) à **un état final** considéré comme **le résultat**.

Les structures de contrôle décrivent **comment les affectations s'enchaînent** séquentiellement. Elles définissent donc **le transfert du contrôle après la fin de l'exécution d'une instruction**.

Remarque : Les « *goto* » ont été bannis en 1970 avec la naissance de la programmation structurée.

En **programmation structurée**, le transfert de contrôle s'exprime par :

1. **enchaînement séquentiel** (une instruction puis la suivante) ;
2. **traitements conditionnels** ;
3. **traitements répétitifs** (itératifs) ;
4. **appel d'un sous-programme** (un autre programme qui réalise un traitement particulier).

Pour chacune des structures de contrôle présentées ci-après, sont données la syntaxe de la structure dans notre langage algorithmique, les règles à respecter (en particulier pour le typage), la sémantique (c'est-à-dire la manière dont elle doit être comprise et donc interprétée), des exemples ou exercices à but illustratifs.

7.1. Enchaînement séquentiel

Les instructions sont exécutées dans l'ordre où elles apparaissent.

Dans la plupart des langages de programmation, on utilise **le point virgule pour indiquer la fin d'une instruction**.

Exemple (JS) :

```
<script>

  let var1 = 4; // 1ère instruction
  let var2 = -7;
  let tmp = var1;
  var1 = var2;
  var2 = tmp; // dernière instruction

</script>
```

Question : Que fait cette séquence d'instructions ?

7.2. Instructions conditionnelles

Les instructions conditionnelles permettent d'effectuer des séquences d'instructions en fonction de l'évaluation d'une condition booléenne.

Si ... Alors ...

```
if (condition)
{
    // séquence d'instructions
}
```

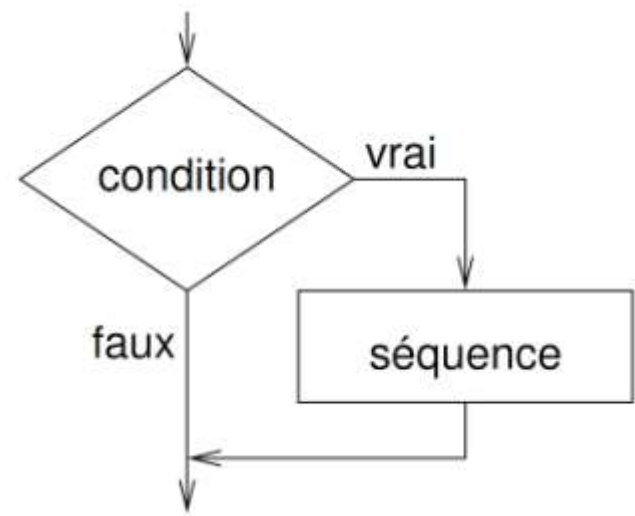
Règle : La condition est nécessairement une expression booléenne.

Évaluation :

- La condition est évaluée;
- Si la condition est vraie, la séquence d'instructions est exécutée puis le contrôle passe à l'instruction qui suit l'accolade fermante.
- Si la condition est fausse, le contrôle passe directement à l'instruction après l'accolade fermante sans que la séquence d'instruction ne soit exécutée.

En d'autres termes, la séquence est exécutée si et seulement si la condition est vraie.

Organigramme :



Exercices

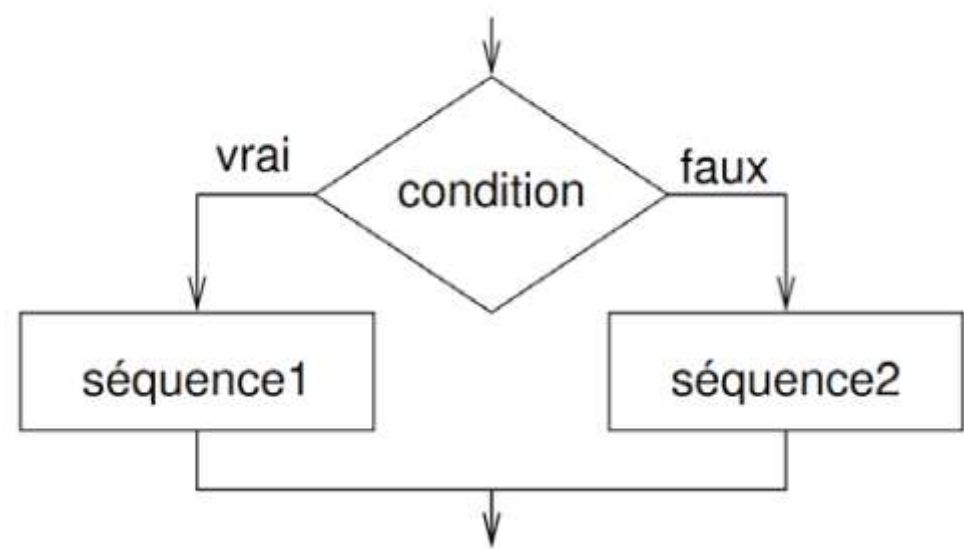
[Pair ?](#)

Si ... Alors ... Sinon ...

```
if (condition)
{
    // séquence d'instructions si la condition est vraie
}
else
{
    // séquence d'instructions si la condition est fausse
}
```

Si la condition est vraie, la première séquence est exécutée, sinon c'est deuxième séquence qui est exécutée. Dans les deux cas, après l'exécution de la séquence, l'instruction suivante exécutée est celle qui suit la dernière accolade fermante.

Organigramme :



Equivalent Ternaire :

```
<!-- Déterminer le périmètre d'un cercle à partir de son rayon -->
<!-- mon commentaire -->
<script>
    //Code Javascript
    //Définition des constantes

    let a = 3;
    let b = 4;
    let c = 0;

    //Opérateur de condition SI
    if(a == b){

        c = a + b;

    }else if(a == 2){

        c = 1;

    }else{

        c = 2;

    }

    document.writeln("Valeur de c : " + c);

    //ternaire idem
    c = a == b ? a + b : a == 2 ? 1 : 2;

    document.writeln("Valeur de c : " + c);

    //Fin de code Javascript
</script>
```

Exercice

[Maximum entre deux valeurs réelles](#)

Exercice : Sinon et Si

Réécrire une instruction Si...Alors...Sinon sans utiliser le Sinon.

Quel est l'intérêt du Sinon ?

La clause Sinon Si

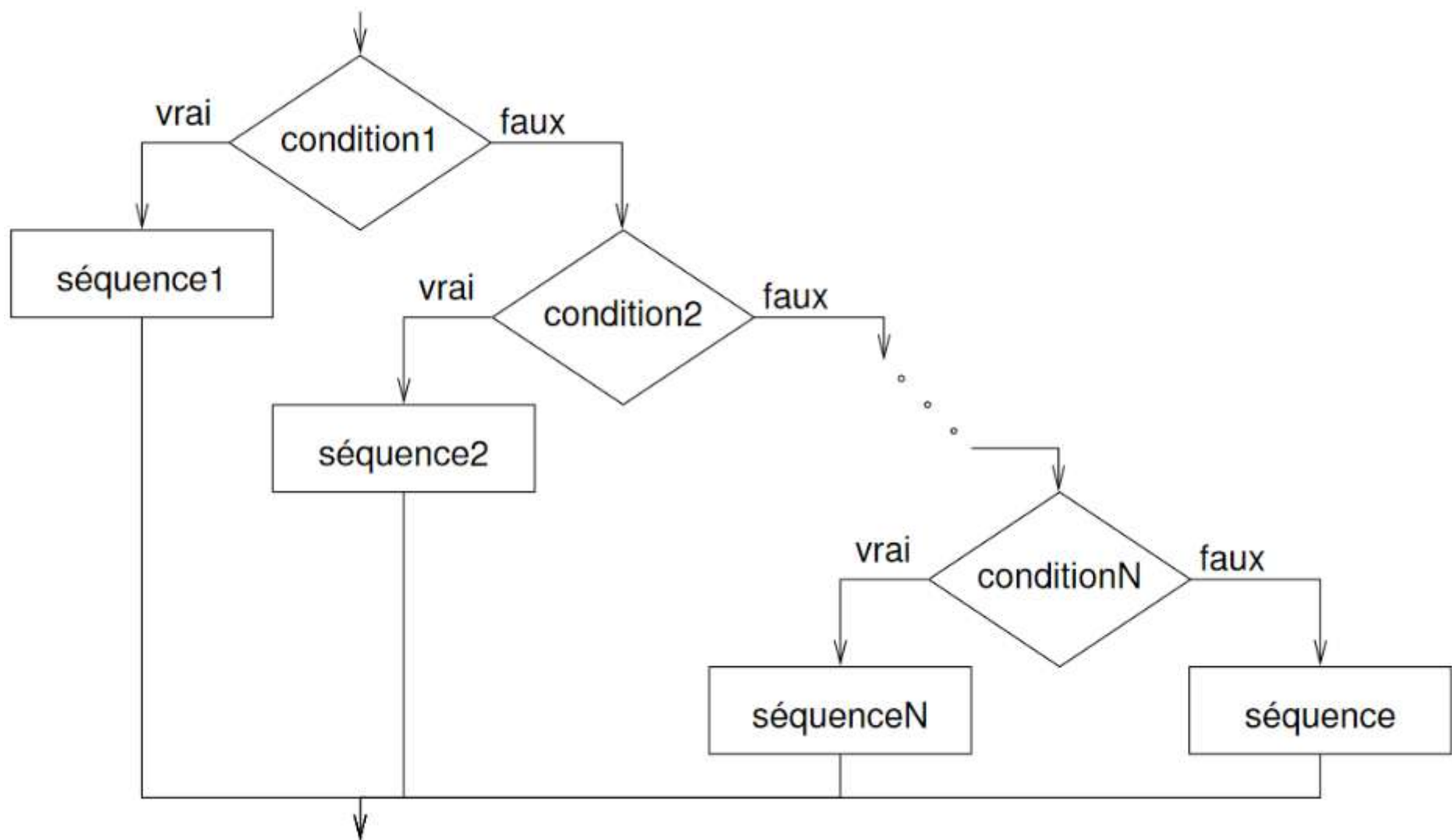
La conditionnelle Si...Alors...Sinon peut être complétée avec des clauses Sinon Si suivant le schéma suivant :

```
if (condition1)
{
    // séquence d'instructions si La condition1 est vraie
}
else if (condition2)
{
    // séquence d'instructions si La condition2 est vraie}
...
else if (conditionN)
{
    // séquence d'instructions si La conditionN est vraie
}
else
{
    // séquence d'instructions si toutes Les conditions précédentes sont fausses
}
```

Les conditions sont évaluées dans l'ordre d'apparition.

Dès qu'une condition est vraie, la séquence associée est exécutée. L'instruction suivante à exécuter sera alors celle qui suit l'accolade fermante. Si aucune condition n'est vérifiée, alors la séquence associée au Sinon, si elle existe, est exécutée.

Organigramme :



Exercices

- [Signe d'un entier](#)
- [Signe d'un produit](#)
- [Catégories](#)
- [Facture photocopies](#)

La clause Selon

Exemple : en JS en utilisant une expression retournant un entier

```

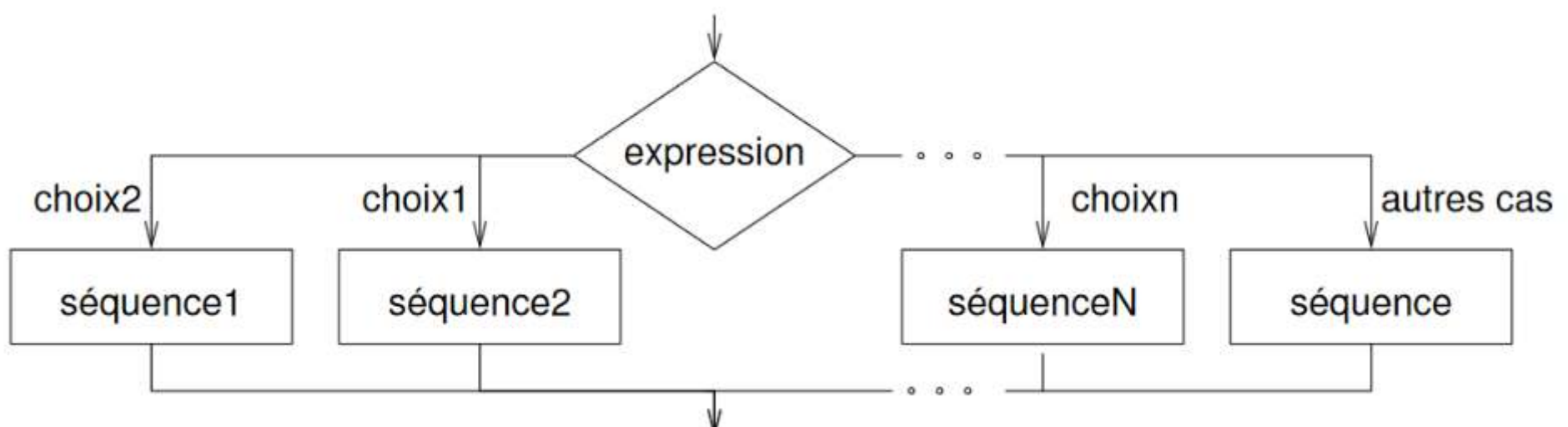
<script>
  //Constante
  const EXPR = 'Papaye';

  switch (EXPR) {
    case 'Orange':
      console.log('Orange à 2 €/kg.');
```

Cette clause est principalement utilisé pour remplacer une clause Si...Sinon Si...Sinon, elle est plus "rapide" à écrire. Mais attention elle n'est pas toujours utilisable.

L'expression est évaluée, puis sa valeur est successivement comparée à chacun des choix possibles. Dès qu'il y a correspondance, les comparaisons sont arrêtées avec l'instruction optionnelle `break;` et la séquence associée est exécutée. Les différents choix sont donc **exclusifs**. Si aucun choix ne correspond, alors la séquence `default;` associée au Sinon, si elle existe, est exécutée.

Organigramme :



Exercice

7.3. Instructions de répétitions

Répétition TantQue

```
while (condition)
{
    // séquence d'instructions
}
```

On parle souvent de *boucle* pour les répétitions.

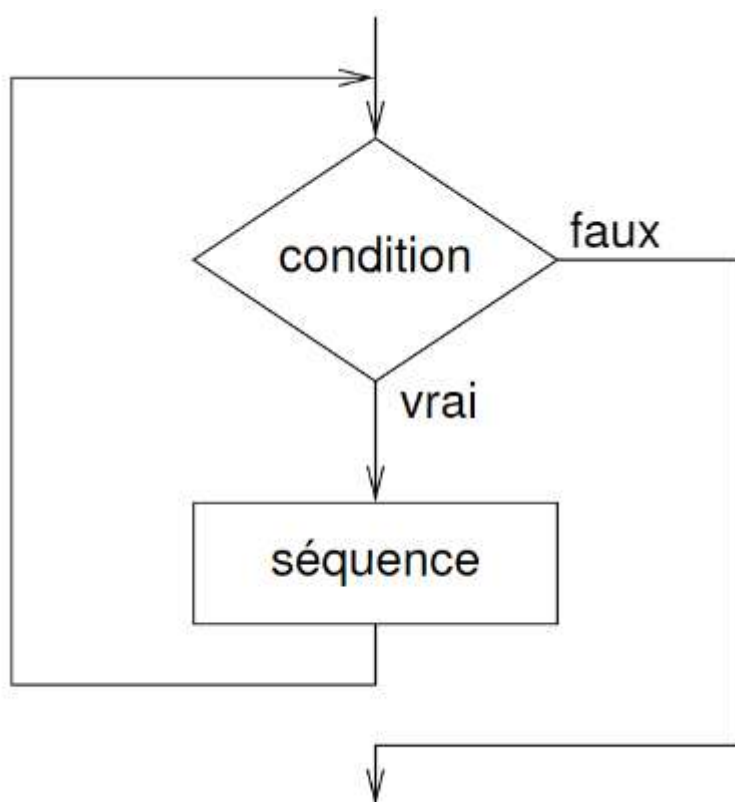
Règles :

- La **condition** doit être **une expression booléenne**
- Pour que **la répétition se termine**, il est nécessaire que **la séquence modifie la condition**.

La condition est évaluée :

- Si elle vaut FAUX alors la boucle se termine et l'exécution se poursuit avec l'instruction qui suit l'accolade fermante.
- Si elle vaut VRAI alors la séquence d'instructions est exécutée puis la condition est de nouveau évaluée.

Organigramme :



Question : Que sait-on sur l'état du programme (l'état de la condition) lorsque l'instruction suivante à exécuter est celle qui suit l'accolade fermante ?

Remarque : Comme le test de la condition est fait en premier, la séquence peut ne pas être exécutée. C'est le cas quand la condition est fausse dès le début.

Problème de la terminaison :

Une propriété importante d'un algorithme/programme est qu'il doit se terminer.

Jusqu'à présent la terminaison était assurée car avec les instructions séquentielles et conditionnelles, l'exécution se fait toujours vers l'avant. Il est **obligatoire** d'atteindre la fin du programme.

Avec les répétitions, on peut revenir en arrière dans le programme. Il faut s'assurer que l'on finira par sortir des répétitions d'un programme sinon le programme risque de s'exécuter indéfiniment. On parle alors de *boucle sans fin*.

Exercices

- [Somme des n premiers entiers](#)
- [Somme des n premiers entiers pairs](#)
- [Table de multiplication](#)

Théorème : Tout algorithme peut être exprimé à l'aide de l'affectation et des trois structures Si...Alors, TantQue et enchaînement séquentiel. Mais ce n'est pas forcément pratique, aussi des structures supplémentaires ont été introduites.

Répétition Répéter

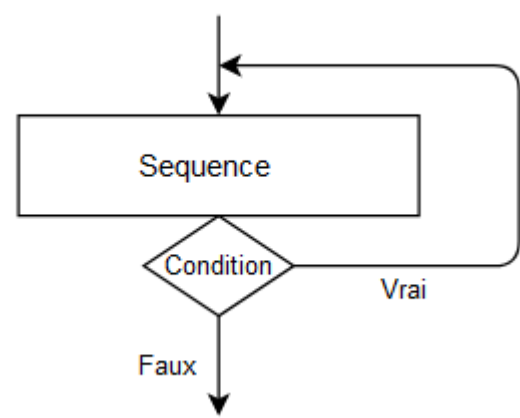
```
do
{
    // séquence d'instructions
} while (condition);
```

Remarques :

- la **condition est évaluée qu'après l'exécution de la séquence**

- la séquence est exécutée au moins une fois
- la condition doit être modifiée par la séquence d'instructions

Organigramme :



Exercices

- [Plusieurs sommes des n premiers entiers](#)
- [Saisie contrôlée d'un numéro de mois](#)

Répétition Pour

```
for (initialisation; condition; incrementation)
{
    // séquence d'instruction
}

// Exemple 1 : Répétition d'une même séquence 15 fois
for (let i = 0; i < 15; i++)
{
    // séquence d'instruction
}

// Exemple 2 : Répétition d'une même séquence 15 fois
for (let i = 15; i > 0; i--)
{
    // séquence d'instruction
}
```

Règle :

- Le type de la variable initialisée doit être compatible avec la condition
- La séquence d'instructions ne doit pas modifier la valeur de la variable utilisée pour la boucle Pour

Remarque : Cette structure est utilisée lorsqu'on connaît à l'avance le nombre d'itérations à faire.

Attention : Il n'y a pas d'équivalent du Pour dans les organigrammes. On peut utiliser la traduction du Pour sous forme de TantQue ou de Répéter.

Exercices

- [Calculer la somme des n premiers entiers avec une boucle for ou while](#)
- [Factorielle](#)

Quelle répétition choisir ?

La première question à se poser est : « Est-ce que je connais a priori le nombre d'itérations à effectuer ? ».

Dans l'affirmative, on choisit la boucle Pour.

Dans la négative, on peut généralement employer soit un TantQue, soit un Répéter.
En général, utiliser un Répéter implique de dupliquer une condition et utiliser un TantQue implique de dupliquer une instruction.

Dans le cas où on choisit d'utiliser un Répéter, il faut faire attention que les instructions qu'il contrôle seront exécutées au moins une fois. S'il existe des cas où la séquence d'instructions ne doit pas être exécutée, alors il faut utiliser un Tant Que (ou protéger le Répéter par un Si).

Une heuristique pour choisir entre TantQue et Répéter est de se demander combien de fois on fait l'itération. Si c'est au moins une fois alors on peut utiliser un Répéter sinon on préférera un Tant Que.

Exercices

- [FizzBuzz Challenge](#)
- [Jeu du + et du -](#)

7.4. Break et label

L'instruction `break` permet de terminer la boucle en cours ou l'instruction `switch` ou `label` en cours et de passer le contrôle du programme à l'instruction suivant l'instruction terminée.

```
let i = 0;

while (i < 6) {
  if (i === 3) {
    break;
  }
  i = i + 1;
}

console.log(i);
// expected output: 3
```

L'instruction `break` peut être utilisée avec une étiquette (*label*) optionnelle qui permet d'interrompre une instruction étiquetée. L'instruction `break` doit être imbriquée au sein de l'instruction référencée. L'instruction étiquetée peut correspondre à n'importe quel instruction de bloc ; il n'est pas nécessaire qu'elle soit précédée par une instruction de boucle.

Une instruction `break`, suivie ou non d'une étiquette, ne peut pas être utilisée dans le corps d'une fonction appartenant elle-même à une boucle, à une instruction `Instructions/switch` ou à une instruction `label`.

```
let id = 0;
bloc_externe:
for (let x = 1; x <= grid; x++) {
  for (let y = 1; y <= grid; y++) {
    if(takin[id] == "V"){
      emptyLig = x;
      emptyCol = y;
      break bloc_externe;//interrompt l'ensemble du bloc
    }
    id++;
  }
}
```

8. Les tableaux

Un problème

Imaginons que nous souhaitons créer un programme permettant de lire et de stocker 5 notes saisies par l'utilisateur puis de calculer la moyenne.

Une première solution pourrait ressembler à ceci :

```
let moyenne;

let note1 = 12;
let note2 = 14;
let note3 = 08;
let note4 = 13;
let note5 = 12;

//Calculer la moyenne
moyenne = (note1 + note2 + note3 + note4 + note5) / 5;

//Afficher le résultat
console.log("La moyenne est : " + moyenne);
```



C'est très long ! Et encore il n'y a que 5 notes ...

Heureusement, il existe un outil permettant de faire ce genre de traitement plus facilement : ce sont **les tableaux**.

8.1. Déclaration d'un tableau

Un **tableau** est une **variable** permettant de ranger un **nombre précis d'occurrences**.

A la déclaration d'un tableau, en JS il n'est pas nécessaire d'indiquer **la taille du tableau** et **le type de données** qui seront stockées.

```
let fruits = ['Apple', 'Banana'];  
console.log(fruits.length);
```

`length` permet d'obtenir la taille du tableau.

Pour **accéder à une "case" d'un tableau**, il faut indiquer **l'indice** de la case souhaitée.

```
let first = fruits[0];  
// Apple  
  
let last = fruits[fruits.length - 1];  
// Banana
```



En JS, l'indice de la 1ère case est 0.

L'indice est un entier qui pour un tableau de taille n , l'indice varie de 0 à $n-1$.

8.2. La boucle For

L'énorme avantage des tableaux c'est qu'on va pouvoir les traiter avec des boucles.

Reprenons notre programme en utilisant un tableau pour stocker les notes.

Parcourir un tableau avec la boucle FOR :

```
<pre id="message">

</pre>
<script>
  let moyenne = 0;
  let notes = [12, 14, 08, 13, 12];

  //Calculer la moyenne
  for(let i = 0; i < notes.length ;i++) {

    ///valeur de tableau notes à l'index i
    moyenne += notes[i];

  };

  //longeur du tableau notes.length
  moyenne = moyenne / notes.length;

  //Afficher le résultat
  document.getElementById("message").innerHTML = "La moyenne est : " + moyenne;

</script>
```

L'indice qui permet d'accéder à un élément d'un tableau est un nombre en clair, une variable ou une expression, c'est un entier :

- Supérieur ou égal à 0
- Inférieur au nombre d'éléments du tableau.

Il est possible d'accéder (via son **index**) à un élément du tableau :

```
let note = notes[1];
// 14
```

8.3. Foreach

Un autre exemple pour parcourir un tableau avec foreach :

```
let moyenne = 0;
let notes = [12, 14, 08, 13, 12];

//Calculer la moyenne
notes.forEach(function(item, index, array) {

    moyenne += item;

});

//longeur du tableau notes.length
moyenne = moyenne / notes.length;

//Afficher le résultat
document.getElementById("message").innerHTML = "La moyenne est : " + moyenne;
```



Ici, nous avons utilisé 1 boucle pour parcourir les éléments du tableau :

```
notes.forEach(function(item, index, array) { //instruction});
```

Ajouter un élément au tableau avec **push** :

```
notes.push('16');
```

...et trouver l'index de la valeur d'un tableau :

```
let pos = notes.indexOf(16);
// 5
```

[Voir la documentation officielle pour aller plus loin.](#)

8.4. La boucle For...IN

Il est possible d'initialiser un tableau en spécifiant les index autre qu'un nombre entier. Dans ce cas on utilisera la syntaxe suivante :

```
let notes = { a: 1, b: 2, c: 3 };

for (let index in notes) {
  console.log(`${index}: ${notes[index]}`);
}
```

a: 1
b: 2
c: 3

Remarquez ici que l'on utilise une boucle de type For...IN. En effet cette boucle à l'inverse de `foreach` nous retourne bien des index de type `String`, `foreach` ne fonctionne pas dans ce cas de figure.

8.5. Exercices

[Moyenne des notes](#)

[Voyelles](#)

Quel est le résultat de chacun de ces programmes ?

- [Programme 1](#)
- [Somme de 2 tableaux](#)
- [Schtroumpf](#)
- [Tri à bulle](#)
- [Recherche dichotomique](#)

9. Les fonctions et procédures

Jusqu'ici, nous avons utilisé une fonction appelé `message()` nous permettant d'exécuter un code en lien avec un événement de type `click`.

```
document.getElementById("button").addEventListener("click",message);

//Fonction
function message() { //instructions}
```

Nous allons maintenant créer des procédures et fonctions afin de :

- **Structurer** davantage le code
- **Réutiliser un bloc d'instructions** avec des paramètres différents et ainsi éviter de dupliquer des blocs de code.

Procédures et fonctions rendent le **code plus lisible** et **plus facile à maintenir**.

9.1. Définition

Une fonction est **un bloc nommé réutilisable** qui contient **une suite d'instructions** (qui est vu comme un tout). En programmation orienté objet (POO) cela s'appelle une **méthode**.

Une fonction prend **0 à N paramètres** et peut renvoyer **un résultat**.

Un programme provoque l'exécution des instructions **en appelant la fonction par son nom** et **en en spécifiant les éventuels arguments requis**.

Une méthode est définie par **son nom**, **la liste de ces paramètres** et **le type de son résultat**. Ces éléments définissent **la signature de la fonction**.

9.2. Exemple fonction

Reprenons le calcul du périmètre d'un cercle.

Ici, la méthode `perimetreCercle` calcule le périmètre d'un cercle à partir de son rayon passé en paramètre et renvoie comme résultat le périmètre du cercle.

Le calcul du périmètre est défini en seul endroit.

```
<pre id="message"></pre>
<script>
  let rayon = 5;
  let perimetre = perimetreCercle(rayon);

  //Afficher le résultat
  document.getElementById("message").innerHTML = "Périmetre : " + perimetre;

  //Calcule le périmètre d'un cercle
  function perimetreCercle (rayon)
  {
    let resultat = Math.PI * rayon * 2;
    return resultat;
  }
</script>
```

La méthode `perimetreCercle` est **définie une fois** et peut être **appelée par le programme principal à plusieurs reprises avec des paramètres différents**.

Les paramètres sont utilisés pour faire **passer des données entre l'appelant et la méthode appelée**.

C'est **une zone mémoire commune entre l'appelant et l'appelé**.
L'ordre des paramètres est important.

La variable `resultat` déclarée dans la méthode est **une variable locale**.
Sa portée est limitée au bloc dans lequel elle a été définie, elle ne peut être utilisée qu'a l'intérieur de la méthode (entre les accolades).



Une fonction peut renvoyer un résultat
Pour cela, on utilise l'instruction `return` (fait sortir de la fonction)

Conventions de nommage

Le nom d'une fonction ou des paramètres commence toujours en minuscule (camelCasing).

Le nom de la méthode et le nom des paramètres doivent être "parlants".
Les **noms** contribuent à éclairer sur le but de la méthode et sur le rôle des paramètres, ils participent à **documenter la méthode**.

Par exemple : on préférera `rayon` à `r`.

*Il est possible de générer une documentation pour une fonction avec les commentaire `/**[ENTER]` :*

```
/**
 * Retourne le perimetre d'un cercle
 * @param rayon:Number
 * @returns {number}
 */
function perimetreCercle (rayon)
{
  let resultat = Math.PI * rayon * 2;
  return resultat;
}
```

9.3. Fonction et objet

Les paramètres primitifs (comme les nombres) sont passés à la fonction **par valeur**. La valeur est passée à la fonction mais si cette dernière change la valeur du paramètre, cela n'aura pas d'impact au niveau global ou au niveau de ce qui a appelé la fonction.

Si l'argument passé à la fonction est un objet (une valeur non-primitive, comme un objet [Array](#) ou un objet défini par l'utilisateur), et que la fonction change les propriétés de cet objet, ces changements seront visibles en dehors de la fonction. Par exemple :

```
function maFonction(monObjet) {
    monObjet.fabricant = "Toyota";
}

var mavoiture = {fabricant: "Honda", modele: "Accord", annee: 1998};
var x, y;

x = mavoiture.fabricant;    // x aura la valeur "Honda"

maFonction(mavoiture);
y = mavoiture.fabricant; // y aura la valeur "Toyota"
                        // (la propriété fabricant a été modifiée par la fonction)
```

Note : Affecter un nouvel objet au paramètre n'aura **pas** d'effet en dehors de la fonction car cela revient à changer la valeur du paramètre plutôt que la valeur d'une des propriétés de l'objet. Par exemple

```
function maFonction(monObjet) {
    monObjet = {fabricant: "Ford", modele: "Focus", annee: 2006};
}

var mavoiture = {fabricant: "Honda", modele: "Accord", annee: 1998};
var x, y;

x = mavoiture.fabricant;    // x reçoit la valeur "Honda"

maFonction(mavoiture);
y = mavoiture.fabricant;    // y reçoit la valeur "Honda"
```

Dans le premier exemple, l'objet **mavoiture** était passé à la fonction **maFonction** qui le modifiait. Dans le second exemple, la fonction n'a pas modifié l'objet qui avait été passé en argument, elle a créé une nouvelle variable locale, possédant le même nom que l'objet global passé en argument : il n'y a donc pas de modifications sur cet objet global.

9.4. Fonction anonyme

Syntaxiquement, la déclaration de fonction utilisée ci-dessus est une instruction. On peut également créer une fonction grâce à une **expression de fonction**. De telles fonctions peuvent être **anonymes** (ne pas avoir de nom correspondant). La fonction `carré` aurait pu être définie de la façon suivante :

```
var carre = function (nombre) { return nombre * nombre };  
var x = carre(4); //x reçoit la valeur 16
```

[Aller plus loin sur les fonctions en JS](#)

9.5. Exercices

[Bonjour](#)

[EstPair](#)

[Mention](#)

[Distance](#)

9.6. Passage de paramètres

Nous allons maintenant mettre le focus sur **le passage de paramètres entre l'appelant et l'appelé**.

Passage par valeur

Par défaut, les paramètres sont passés **par valeur**.

Qu'est-ce que cela veut dire ?

Une copie de la valeur de chaque paramètre est faite au moment de l'appel.

En conséquence, si **la fonction appelée modifie la valeur d'un paramètre**, celle-ci sera **perdue à la fin de la fonction** et ne sera pas répercutée dans l'appelant.

Ce comportement n'est pas sorcier ! C'est exactement ce qui se passe quand vous initialisez une variable à partir d'une autre. La valeur est copiée au moment de l'initialisation puis les 2 variables vivent leur vie indépendamment.

Démonstration

```
Simple();

function Simple()
{
    let x = 3;
    console.log("valeur de x : " + x);
    Doubler(x);
    console.log("valeur de x : " + x);
}

function Doubler(n)
{
    console.log("valeur de n : " + n);
    n = n * 2;
    console.log("valeur de n : " + n);
}
```

Résultat du programme : la valeur de x reste inchangée au retour dans le programme principal.

```
valeur de x : 3
valeur de n : 3
valeur de n : 6
valeur de x : 3
```

Assigné la valeur avec Return

Une première possibilité est d'utiliser Return

```
Simple();

function Simple()
{
    let x = 3;
    console.log("valeur de x : " + x);
    x = Doubler(x);
    console.log("valeur de x : " + x);
}

function Doubler(n)
{
    console.log("valeur de n : " + n);
    n = n * 2;
    console.log("valeur de n : " + n);
    return n;
}
```

Résultat : la valeur de x a changé au retour dans le programme principal

```
valeur de x : 3
valeur de n : 3
valeur de n : 6
valeur de x : 6
```

La portée des variables VAR / LET

let permet de déclarer des variables dont la portée est limitée à celle du **bloc** dans lequel elles sont déclarées. Le mot-clé var, quant à lui, permet de définir une variable globale ou locale à une fonction (sans distinction des blocs utilisés dans la fonction).

```
let x = 3;
Simple();

function Simple()
{

    console.log("valeur de x1 : " + x);//3
    Doubler();
    console.log("valeur de x2 : " + x);//6
}

function Doubler()
{
    for (let x = 3; x < 5; x++){ }//Nouvelle variable

    console.log("valeur de x3 : " + x);//3
    x = x * 2;
    console.log("valeur de x4 : " + x);//6
}
```

Différence avec var :

```
let x = 3;
Simple();

function Simple()
{

    console.log("valeur de x1 : " + x);//3
    Doubler();
    console.log("valeur de x2 : " + x);//3
}

function Doubler()
{
    for (var x = 3; x < 5; x++){ }//Nouvelle variable

    console.log("valeur de x3 : " + x);//5
    x = x * 2;
    console.log("valeur de x4 : " + x);//10
}
```

9.7. Exercices

[Réponse Oui/Non](#)

[Conversion d'un nombre décimal en binaire](#)

9.8. Exercices

[Inversion de chaine](#)

[Bissextile](#)

[Compression](#)

[Code Cesar](#)