

**ADRAR**  
FORMATION

Algorithmie

Alexandre Ahmad

# Plan du cours

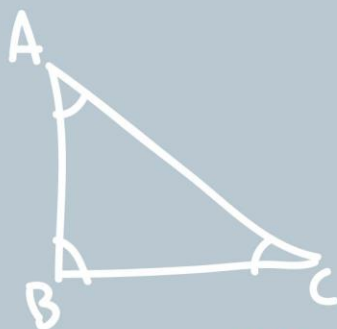
- introduction()
- Si `exercicesAlgo()` > moyenne
  - Introduction JS
  - Tant que PasFini
    - `exercicesJS()`
- Suivant résultat faire
  - Cas 25/20: `devenirMillionnaire()`
  - Autre: `continuerLaFormation()`

$$E = mc = \pi n^2$$

$$mc^2 = c\pi$$

$$\sqrt{18 \cdot yz}$$

$$\begin{aligned} yx - x + z &= 2 \\ x + yx - z &= \lambda \\ y + x + \lambda &= \lambda \end{aligned}$$



$$\pi 2x = \frac{3tx}{1x - y}$$



**1+5**

$$x+5$$

$$y = 2x + 3$$

$$f(x) = 5x+10$$

$$\text{ma\_fonction}(x,y) = 5x+10y$$



$$\text{ma\_fonction}(x) = \sum_{i=0}^{10} i \times x$$

# Ensemble des nombres

## *Entiers naturels*

Entiers non signés/unsigned int

$$\mathbb{N} = \{0; 1; 2; ; \dots\}$$

## *Entiers relatifs*

Entier/int(eger)

$$\mathbb{Z} = \{\dots - 3; -2; -1; 0; 1; 2; \dots\}$$

## *Décimaux*

Virgule flottante/float

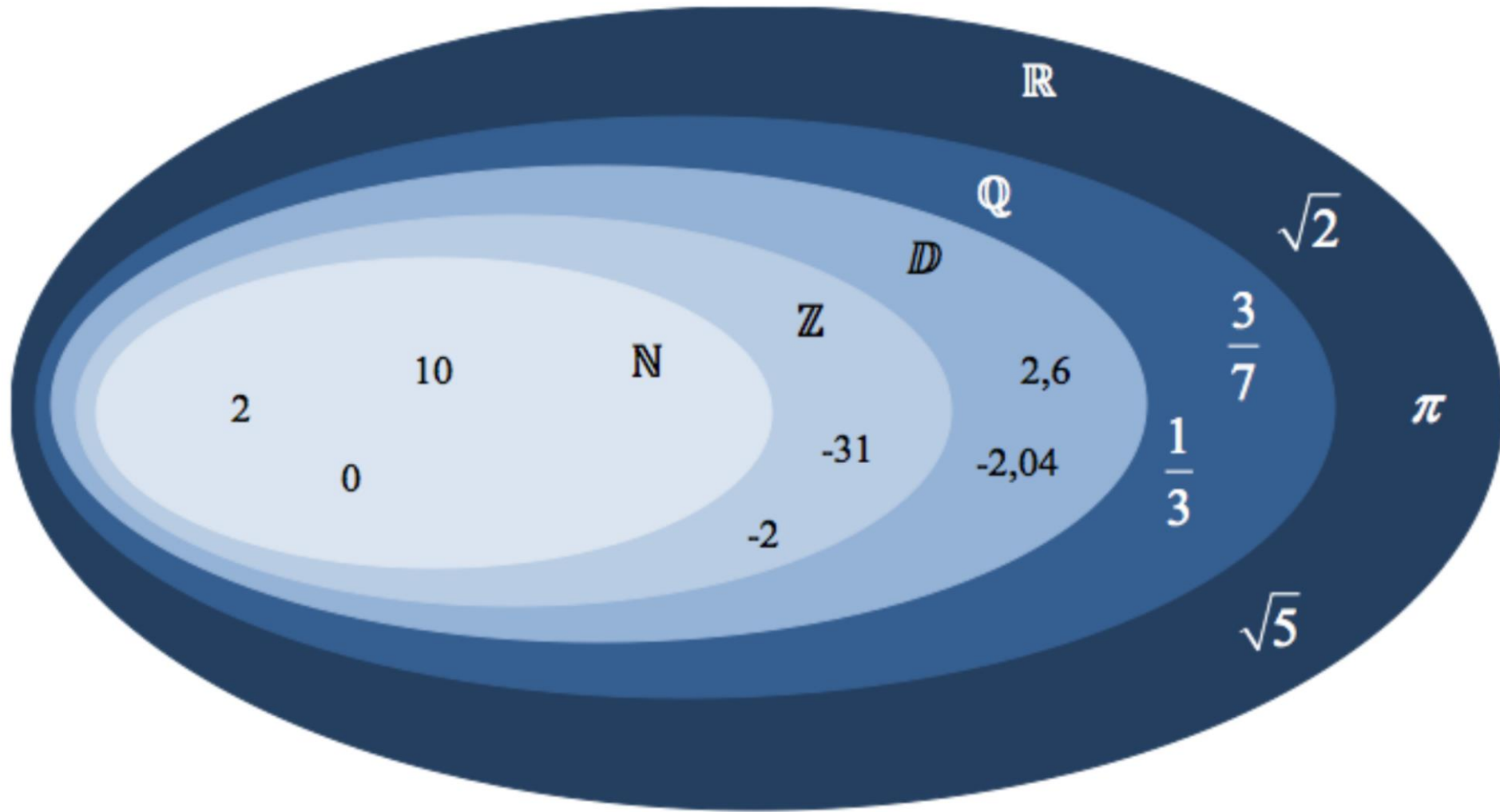
$$\mathbb{D} = \{0, 56; \frac{3}{4}; \dots\}$$

## *Réels*

double

$$\mathbb{R} = \{-57, 245452; \frac{1}{3}; \pi; 1000; \dots\}$$

# Ensemble des nombres



# Qu'est-ce qu'un algorithme ?

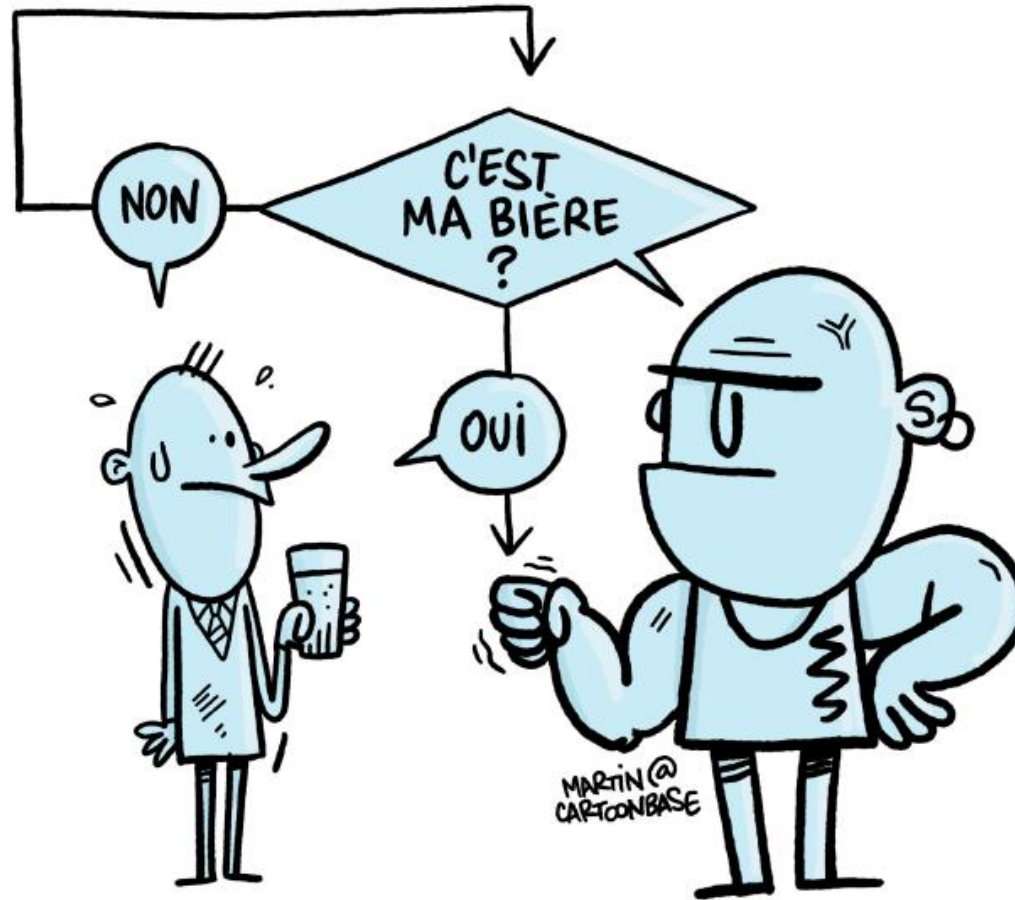
# Qu'est-ce qu'un algorithme ?

Suite finie d'opérations élémentaires permettant de résoudre un problème donné

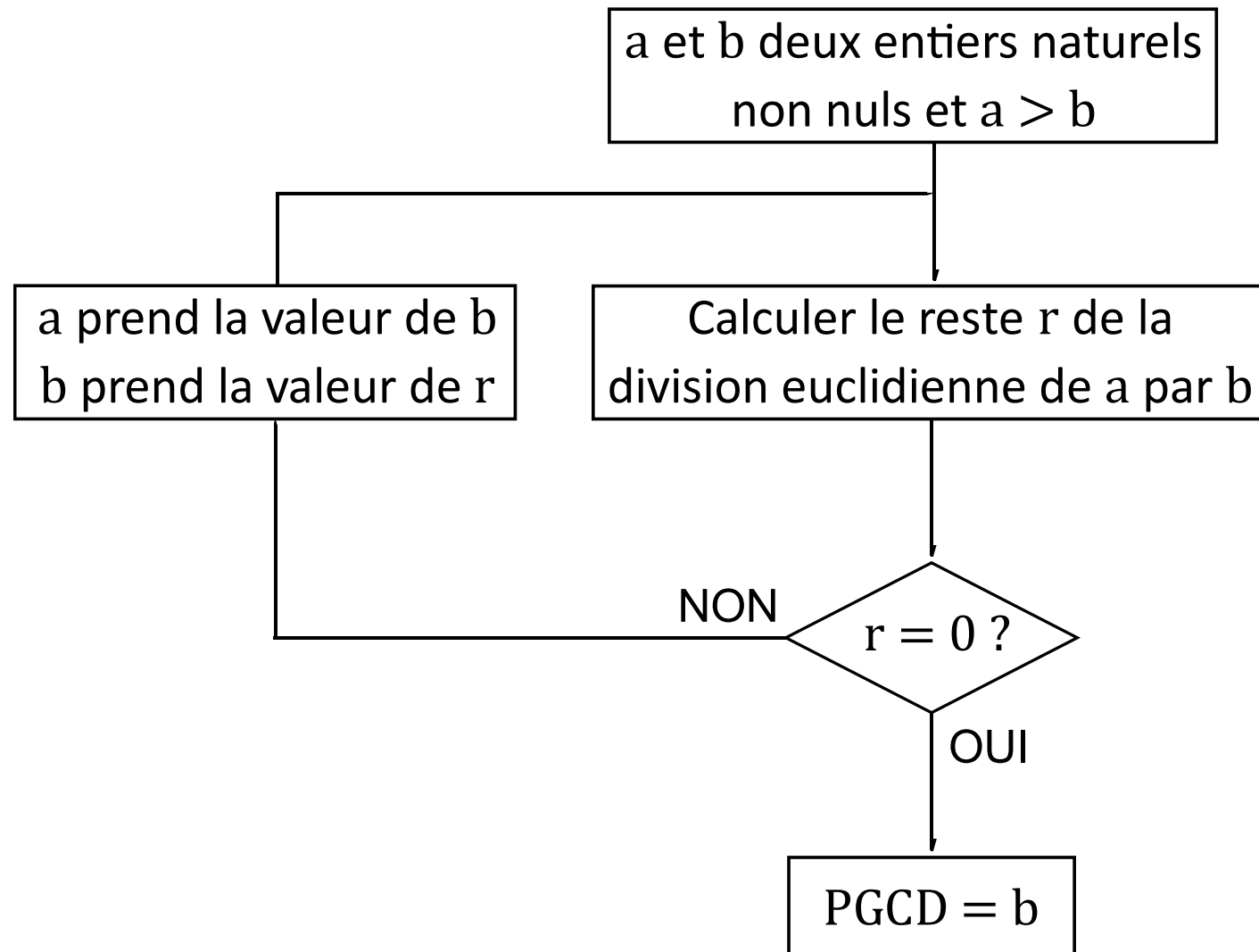
# Qu'est-ce qu'un algorithme ?



# Qu'est-ce qu'un algorithme ?



# Qu'est-ce qu'un algorithme ?





# Étapes de conception d'un programme informatique

1. Identifier le problème : quelle(s) donnée(s), quel(s) résultat(s) ?
2. Organiser les actions : écrire **l'algorithme** (pseudo-code, organigramme)
  - Réfléchir aux informations à manipuler
  - Analyser le problème et le décomposer éventuellement en sous-problèmes
  - Rendre l'algorithme **compréhensible** et **efficace**
  - Penser à l'utilisateur
3. Traduire cet algorithme en langage de programmation
- 4. Compiler** le programme pour qu'il puisse être **exécutable**

Un **langage de programmation** permet à un humain d'écrire un code pouvant être analysé par une machine puis transformé en un programme informatique.

Un **programme informatique** est une suite d'opérations prédéterminées pouvant être exécutées par une machine.

# Pseudo langage et organigramme

décrit un algorithme indépendamment de tout langage

## Pseudo-langage

Exemple: permutation de valeurs

début

entier  $a, b, temp$

lire  $a, b$

$temp \leftarrow a$

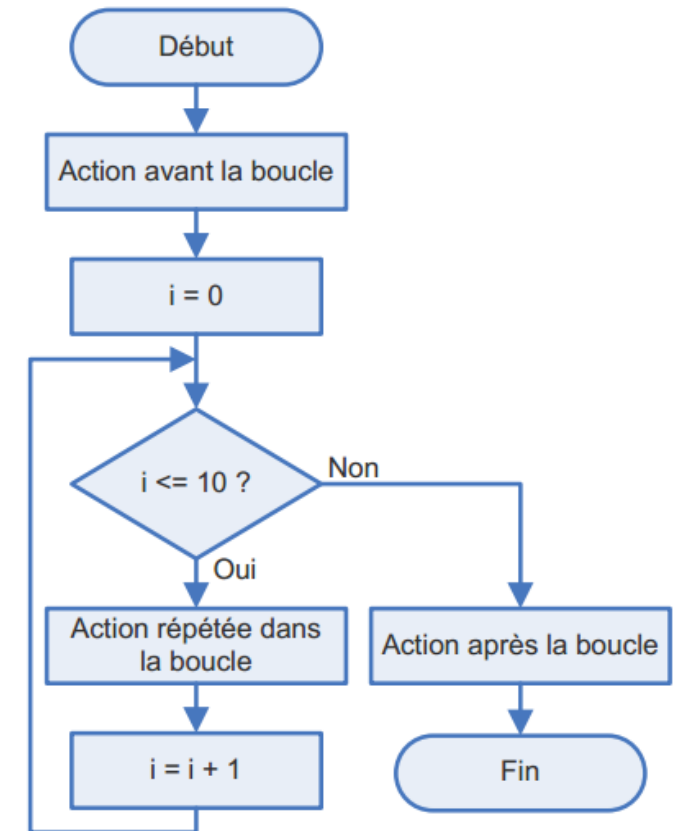
$a \leftarrow b$

$b \leftarrow temp$

afficher  $a, b$

fin

## Organigramme



# Instructions

Une **instruction informatique** désigne une étape dans un **programme informatique**. Une instruction dicte à l'ordinateur l'action nécessaire qu'il doit effectuer avant de passer à l'instruction suivante. Un **programme informatique est constitué d'une suite d'instructions**.

Exemple: permutation de valeurs

début

entier *a, b, temp*

lire *a, b*

*temp* ← *a*

*a* ← *b*

*b* ← *temp*

afficher *a, b*

fin

# Qu'est-ce qu'un algorithme ?



# Instructions

Une **instruction informatique** désigne une étape dans un **programme informatique**. Une instruction dicte à l'ordinateur l'action nécessaire qu'il doit effectuer avant de passer à l'instruction suivante. Un **programme informatique est constitué d'une suite d'instructions**.

Exemple: permutation de valeurs

début

entier *a, b, temp*

lire *a, b*

*temp*  $\leftarrow$  *a*

*a*  $\leftarrow$  *b*

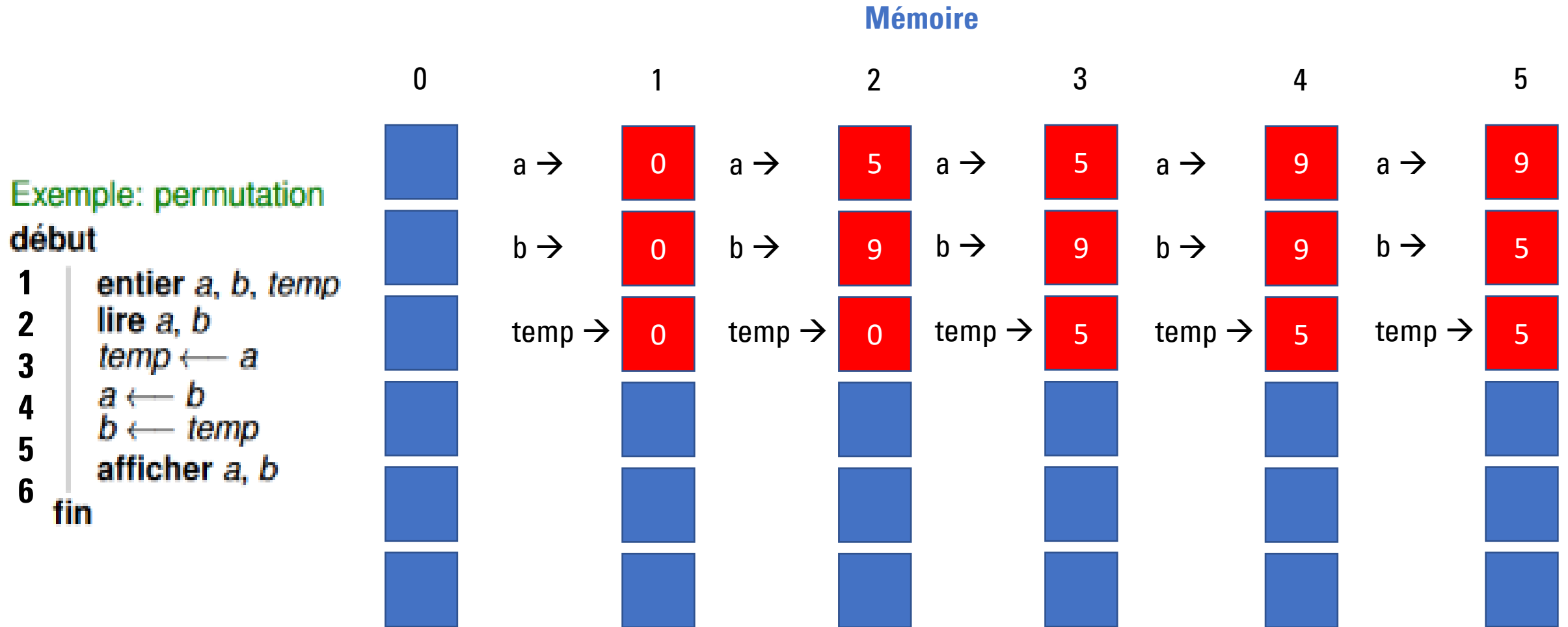
*b*  $\leftarrow$  *temp*

afficher *a, b*

fin

# Variables

Une **variable** est un identificateur associant un nom à une valeur ou un objet.



# Opérateurs sur les types simples

- **Arithmétiques (entiers)**: toutes les entrées sont des **entiers** et la sortie est un **entier**.

Nom	Symbole
addition	+
soustraction	-
multiplication	×
division entière	/
reste	mod
inversion de signe	-

- **Arithmétiques (réels)**: au moins une entrée est un **réel** et la sortie est un **réel**.

Nom	Symbole
addition	+
soustraction	-
multiplication	×
division	/
inversion de signe	-

- **Comparaisons**: les deux entrées sont des **entiers**, **caractères\*** ou **réels**. La sortie est un **booléen**.

Nom	Symbole
est égal à	=
est plus petit que	<
est plus grand que	>
est plus petit ou égal à	≤
est plus grand ou égal à	≥

- **Logiques**: toutes les entrées sont des **booléens** et la sortie est un **booléen**.

Nom	Symbole
conjonction	<b>et</b>
disjonction	<b>ou</b>
négation	<b>non</b>

# Procédures, fonctions, méthodes

Une **procédure** permet d'isoler un fragment de programme, et d'en faire une opération générale, paramétrable, susceptible d'être utilisée de façon répétée.

Exemple: permutation

début

1 entier *a*, *b*, *temp*

2 lire *a*, *b* ←

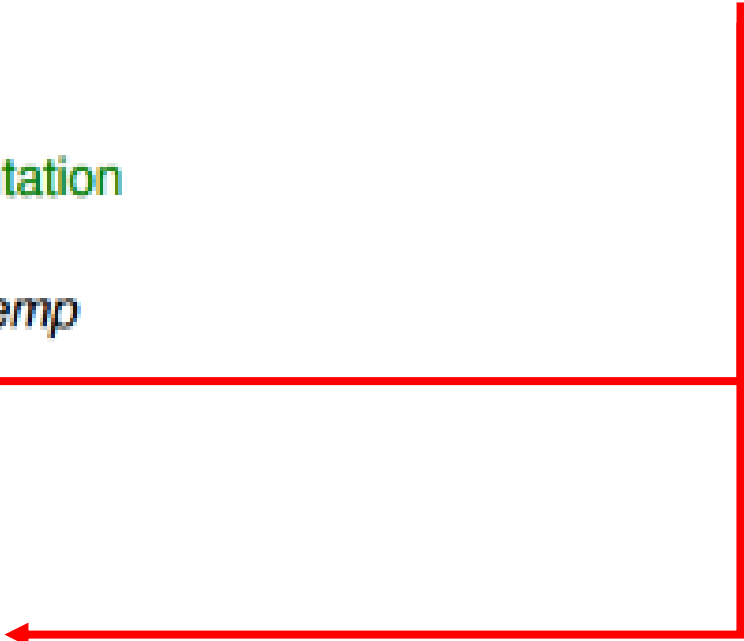
3 *temp* ← *a*

4 *a* ← *b*

5 *b* ← *temp*

6 afficher *a*, *b* ←

fin





---

## L'instruction de test "si alors sinon"

Dans **si** *condition* **alors** *bloc 1* **sinon** *bloc 2*, la condition est une expression booléenne. Le bloc 1 est exécuté si la condition est vraie ; le bloc 2 est exécuté si la condition est fausse.

Exemple: racine carrée

début

réel  $x, y$

lire  $x$

**si**  $x \geq 0$  **alors**

$y \leftarrow \text{sqrt}(x)$

**afficher**  $y$

**sinon**

**afficher** "Valeur indéfinie"

fin

---

# L'instruction de boucle "pour"

L'instruction *pour* est utilisée lorsque le nombre d'itérations est **connu à l'avance**: elle initialise un **compteur**, l'incrémente après chaque exécution du bloc d'instructions, et vérifie que le compteur ne dépasse pas la borne supérieure.

Exemple: Somme des entiers de 1 à  $n$

début

entier  $n, s, i$

lire  $n$

$s \leftarrow 0$

**pour  $i$  de 1 à  $n$  faire**

└  $s \leftarrow s + i$

**afficher  $s$**

fin

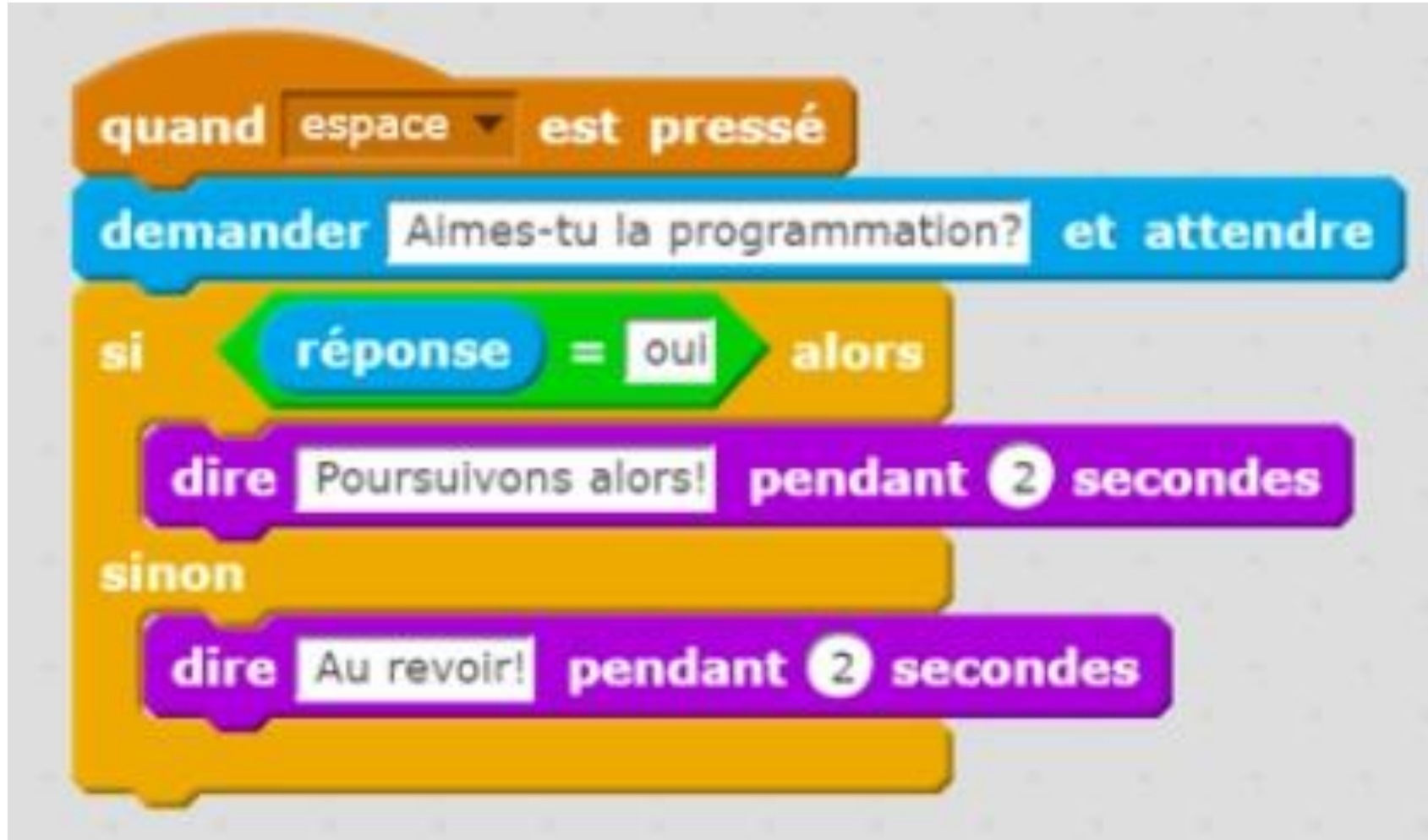






# Pseudo langage et organigramme

décrit un algorithme indépendamment de tout langage



# Codage + Géométrie = *Algoblocs* !

Apprenez à programmer !  
Découvrez les boucles, les variables, les fonctions...  
en dessinant des formes géométriques !

▶ Commencer !

Mode Creation

Défis

## Nouveau : *Algoblocs+* !

Vous appréciez Algoblocs ? Alors, découvrez *Algoblocs+*, une version spécialement destinée à l'Enseignement qui comporte :

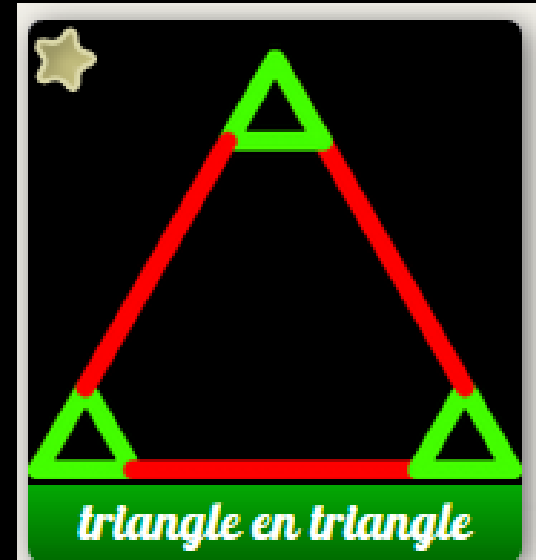
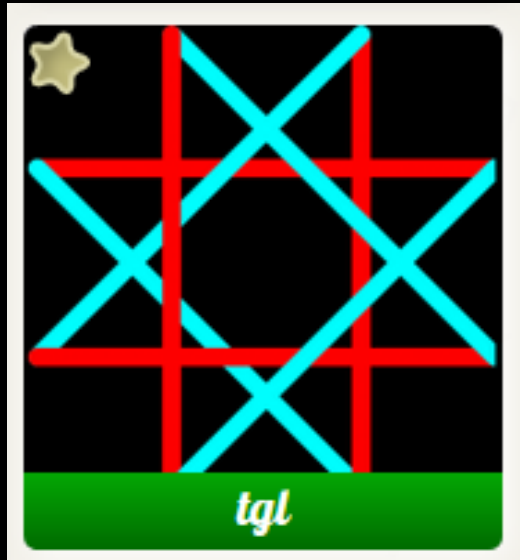
- encore plus d'exercices
- une progressivité particulièrement étudiée
- un accompagnement pédagogique

Algoblocs+ est édité par Génération 5 en version "locale" pour l'ensemble des postes de l'établissement (durée illimitée). Il ne nécessite donc pas de connexion Internet. Cliquez ici pour découvrir *Algoblocs+*.

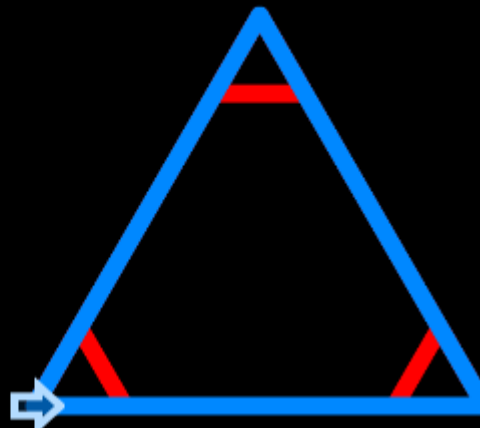
<https://www.algoblocs.fr/>

Identifiants: VotrePrénom2002 mdp:prenom

Exemple: Alexandre2002 mdp:alexandre



Variables  
 angle = ???  
 x = ???  
 distance = ???



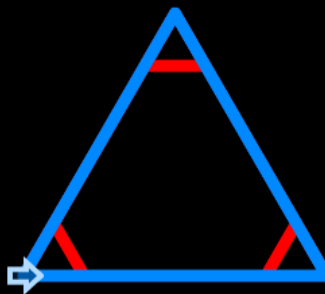
Défi triangle des Bermudes  
 Une fonction qui dessine 1 triangle, prend distance et angle en paramètre  
 1 variable  
 <28 blocks

<https://www.algoblocs.fr/>

Identifiants: VotrePrénom2002 mdp:prenom  
 Exemple: Alexandre2002 mdp:alexandre

## Résultat

Variables  
angle = ???  
x = ???  
distance = ???



Retour

Vitesse : ☐ Auto ☐ Grille

Blocs 27/28

Tout supprimer Afficher code

- Boucles
- Variables
- Nombres
- Conditions
- Fonctions
- Déplacements
- Apparence

```
répéter 3 fois
  TracerTriangle avec :
    distance 100
    angle 60
  fixer x à x + 60
  avancer de 500 pixels
  former angle sur la gauche de 60
changer couleur
TracerTriangle avec : distance 500 angle 60
```

```
fonction TracerTriangle avec : distance, angle
  répéter 3 fois
    avancer de distance pixels
    former angle sur la gauche de angle
```

Parcourir les  
exercices ▾

# Algorithme et programmation - 3e

## Interprétation

### Exercice 1 : Initiation - Trois variables, une lecture, deux calculs enchainées

On considère l'algorithme ci-dessous :

$$\begin{cases} a \leftarrow 5 + N \\ b \leftarrow 5 + a \end{cases}$$

Si  $N = 9$ , quelle est la valeur finale de  $b$  ?



Valider ✓



# Editeurs Javascript

Sublime text

<https://www.sublimetext.com/>

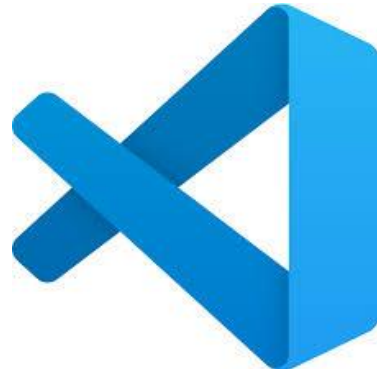
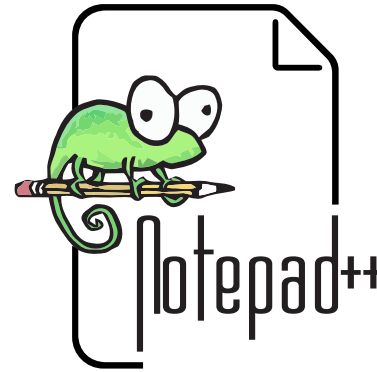
Notepad++

<https://notepad-plus-plus.org/>

Visual Studio code

<https://code.visualstudio.com/>

Plugin liveserver



# Cheat Sheet javascript

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>
      JS Cheat Sheet
    </title>
  </head>

  <body style="text-align: center;">
    <h1 style="color:green;">
      JS  Cheat Sheet
    </h1>
    <p style="color:orange;" id="principal">
    </p>
    <script>

      // on va programmer en javascript ici

    </script>
  </body>
</html>
```

# Cheat Sheet javascript

```
// on va programmer en javascript ici
```

```
// On écrit directement dans l'espace HTML  
document.write("Hello HTML World!");
```

```
//on écrit dans la console (F12)  
console.log("Hello console World!");
```

```
//On écrit dans l'élément d'id "principal"  
document.getElementById("principal").innerHTML = "Hello paragraph World !";
```

```
//Tout ce qui est après '//' sur la ligne n'est que du commentaire
```

```
/* Et après '/*' c'est  
tout  
le
```

```
code  
jusqu'à
```

```
*/
```

# Cheat Sheet javascript

```
// Les Variables et leurs types
```

```
document.write("<BR>allez on saute ! <BR>"); // saut de ligne en HTML !
```

```
/* les variables s'écrivent avec un nom qui décrit  
ce qu'elles contiennent écrit en camelCase */
```

```
let someNumber = 5;
```

```
document.write(someNumber);
```

```
someNumber = 10.3335;
```

```
document.write("<br>");
```

```
document.write(someNumber);
```

```
let 5emeLettre = 'e'; // ne peut commencer par un chiffre (_, lettre)
```

```
let _test = 10; // ok !
```

```
// Addition
```

```
someNumber = 30 + 5;
```

```
document.write("<br>Addition: ");
```

```
document.write(someNumber);
```

```
// division, multiplication
```

```
someNumber = 10 / 2;
```

```
document.write("<br>Division: ");
```

```
document.write(someNumber);
```

# Cheat Sheet javascript

```
let anotherNumber = 55.364;  
document.write("<br>chiffre à virgule: ");  
document.write(anotherNumber);
```

```
// arrondi 0.4-> dessous 0.6 --> dessus  
anotherNumber = Math.round(anotherNumber);  
document.write("<br>Arrondi: ");  
document.write(anotherNumber);
```

```
anotherNumber = 55.6;  
anotherNumber = Math.round(anotherNumber);  
document.write("<br>Arrondi: ");  
document.write(anotherNumber);
```

# Cheat Sheet Javascript

```
// floor -> arrondi vers le sol.
```

```
anotherNumber = 55.6;  
anotherNumber = Math.floor(anotherNumber);  
document.write("<br>Arrondi en dessous: ");  
document.write(anotherNumber);
```

```
// Ceil (ing): arrondi vers le plafond
```

```
anotherNumber = 55.6;  
anotherNumber = Math.ceil(anotherNumber);  
document.write("<br>Arrondi au dessus: ");  
document.write(anotherNumber);
```

```
// une valeur aléatoire entre 0 inclus et 1 exclu [0, 1[
```

```
anotherNumber = Math.random(); //  
document.write("<br>Chiffre aléatoire entre 0 et 1: ");  
document.write(anotherNumber);
```

```
// une valeur aléatoire entre 0 inclus et 5 inclus [0, 5]
```

```
anotherNumber = 1 + Math.floor(Math.random()*5); //  
document.write("<br>Chiffre aléatoire entre 1 et 5:");  
document.write(anotherNumber);
```

# Cheat Sheet Javascript

```
// chaîne de caractère | string
```

```
let someText = "du texte";
```

```
document.write("<br>");
```

```
document.write(someText);
```

```
// Concaténation de texte
```

```
someText = "Bonjour, je " + "est un autre." + "<BR>";
```

```
// même avec des variables
```

```
document.write("<p style=color:red;> " + someText + "</p>");
```

```
// Il faut neutraliser les "
```

```
// essayer sans les \
```

```
document.write("<p style=\"color:green;\"> " + someText + "</p>");
```

```
someText = 12 + "3"; // 12 en chaîne concaténée avec 3 en chaîne
```

```
document.write(someText);
```

```
document.write("<br>");
```

```
document.write(1 + 2 + "3"); // ?
```

```
document.write("<br>");
```

# Cheat Sheet javascript

```
// Booléan
```

```
let someBoolean = true;  
document.write(someBoolean);  
document.write("<br>");
```

```
someBoolean = false;  
document.write(someBoolean);  
document.write("<br>");
```



# Cheat Sheet javascript

```
// les types
document.write("<br>");
document.write(typeof someNumber);
document.write("<br>");
document.write(typeof someText);
document.write("<br>");
document.write(typeof someBoolean);

someText = "12";
someNumber = parseInt(someText); // conversion d'une chaine en chiffre
document.write("<br>");
document.write(someNumber);

someText = "99.33 days";
someNumber = parseFloat(someText); // conversion d'une chaine en chiffre
document.write(someNumber);
document.write("<br>");
```

# Javascript Exercices (Lot 1)

1. Initialiser une variable x qui prend la valeur de 10
2. Afficher le résultat de x
3. Initialiser une variable y qui prend la valeur de x additionnée de 50
4. Afficher le résultat de y
5. Initialiser une variable z qui est le résultat de la multiplication de x et y
6. Afficher le résultat sous la forme: « Le résultat de la multiplication de 10 et 60 est 600 »
  
7. Initialiser une variable qui contient « Antoine », puis une autre qui contient « Griezmann »
8. Afficher sous la forme « AntoineGriezmann »
9. Afficher sous la forme « Antoine Griezmann »
10. Initialiser une variable qui contient « Ninja », puis une autre qui contient « Go »
11. Afficher sous la forme « Go Ninja Go Ninja Go »
  
12. Initialiser une variable qui contient votre prénom, et une autre votre nom, puis une autre votre âge.
13. Afficher sous la forme « Je m'appelle votre\_nom votre\_prénom et j'ai votre\_age, et j'en suis bien content.e. »
14. Afficher maintenant sous la forme:  
    « Je m'appelle:  
    **votre\_nom votre\_prénom et j'ai votre\_age,**  
    et j'en suis bien content.e. »

**correction**

**ma\_fonction(x,y) = 5x+10y**

**ma\_fonction(1,2) = ?**

**ma\_fonction(2,5) = ?**

**a = 3**

**b = 4**

**ma\_fonction(a,b) = ?**

**a = 6**

**b = 7**

**ma\_fonction(a,b) = ?**

# Fonctions Javascript

Format complet de description des fonctions: **la signature**

```
function nomFonction(liste optionnelle de paramètres)
{
    // corps de la méthode code dans le bloc
}
```

↑  
param1, param2, ...

```
1 // fonction sans paramètre
2 function bonjour()
3 {
4     document.write(« Bonjour »);
5 }
6
7 bonjour(); //Appel de la fonction. Ordre: 5-1-2-3-4
8 bonjour(); //encore ! Décidément ! Ordre: 6-1-2-3-4
```

// Ceci est une déclaration de fonction  
// aucun code n'est exécuté  
// Comme un livre dans une bibliothèque.  
// Tant qu'il n'est pas ouvert ...

# Fonctions Javascript

Format complet de description des fonctions: **la signature**

$f(x) = 2x+5 \rightarrow$  en JS

```
// fonction avec paramètre et retour d'information
1  function f(x)
2  {
3      return 2*x+5;
4  }

// Ceci est un déclaration de fonction
// aucun code n'est exécuté
// Comme un livre dans une bibliothèque.
// Tant qu'il n'est pas ouvert ...

5  let resultatCalcul = f(3); //Appel de la fonction. Ordre: 5d-1-2-3-4-5g
6  let resultatCalcul2 = f(24); //Ordre: 6d-1-2-3-4-6g
```

# Cheat Sheet javascript

// déclaration d'une fonction: regrouper et ISOLER des instructions

```
function afficher(paramText)
{
    document.write(paramText);
    document.write("<br>");
}
```

// réutilisation !

```
afficher("C'est plus pratique, non ?");
afficher("Vous en pensez quoi ? " + "<BR> oui monsieur, je trouve que vous avez souvent
        raison !");
```

# JS Exercice

Changer tous les  
`document.write (xxx); document.write(« <br> »)`  
par la méthode `afficher`,  
Sauf le **premier**



# Cheat Sheet javascript

// La même chose, autant de paramètre que nécessaire

```
function diviser(param1, param2)
{
    let resultat = param1 / param2;
    return resultat;
}
```

```
let resultatDivision = diviser(10, 20);
afficher("Le résultat de la division est: " + resultatDivision);
afficher("Le résultat de la division est: " + diviser(50, 2));
```

```
let param1=20, param2=5; // quoi ?
let resultat = diviser(param1, param2); // Est-ce la fin du monde ?
// let déclare la variable dans une portée de block, ie. les accolades.
{
    let a = 10;
    afficher(a); // existe, dans le block
}
```

```
afficher(a); // a n'existe plus. Erreur
// donc param1 et param2, resultat, n'existent quand le contexte local
// de la fonction diviser, pas de redéfinition (ce qui est interdit)
let param1=65; // erreur, déjà défini dans le contexte
```

# Cheat Sheet javascript

```
/* en JS il est possible d'utiliser la déclaration avec var
   var y =5;
   et sans rien
   z = 10;
   mais vous verrez ça plus tard, retenez juste que c'est MAL, TRÈS
   MAL.
   C'est une hérésie. Vade retro satana.

   let itBe = "declaring variables with wisdom";
*/
```

# JS Exercices (Lot 2)

1. Faire un fonction **helloWorld()** qui retourne « Hello ADRAR World! » (et l'appeler + l'afficher)
2. Créer la fonction **quiEstLeMeilleurProf()**. Elle doit retourner Le prof de programmation JS !
3. Créer la fonction **jeRetourneMonArgument()** qui retourne l'argument de type chaîne de caractère.

**Exemple** d'utilisation :

```
resultat = jeRetourneMonArgument("123"); // afficher resultat, il doit valoir "123"
```

4. Créer la fonction **concatenation()**. Elle prendra deux arguments. Elle devra retourner la concaténation des deux.

**Exemple** : argument 1 = Antoine Argument 2 = Griezmann; Résultat : AntoineGriezmann

5. Créer la fonction **concatenationAvecEspace()**. Elle prendra deux arguments de type string. Elle devra retourner la concaténation des deux.

**Exemple** : argument 1 = Ngolo Argument 2 = Kante; Résultat : Ngolo Kante

6. Créer la fonction **monAge()**. Elle devra retourner votre âge actuel.

7. Créer la fonction **somme()**. Elle prendra deux arguments. Elle devra retourner la somme des deux. **Exemple** : argument 1 = 5 Argument 2 = 5 ; Résultat : 10
8. Créer la fonction **soustraction()**. Elle prendra deux arguments. Elle devra retourner la soustraction des deux. **Exemple** : argument 1 = 5 Argument 2 = 5 ; Résultat : 0
9. Créer la fonction **multiplication()**. Elle prendra deux arguments. Elle devra retourner la multiplication des deux. **Exemple** : argument 1 = 5 Argument 2 = 5 ; Résultat : 25
10. Créer la fonction **calculTTC()**. Elle prendra un argument, le prix HT, et retournera la valeur TTC( taux 5,5%). **Exemple** : argument 1 = 10; Résultat : 10,55
11. Créer la fonction **afficherPlatsDuJour()**. Elle prendra 6 arguments, le nom de l'entrée, du plat et du dessert, ainsi que leur prix en HT. Elle devra afficher la carte avec les prix en TTC en respectant l'indentation suivante. **Exemple** : argument 1 = « œufs mimosa », 3, « entrecôte frites », 10, « Banoffee », 5;  
Résultat : Le chef vous propose aujourd'hui
  - Entrée: des œufs mimosa (3,16€)
  - Plat: entrecôte frites (10,55€)
  - Dessert: Banoffee (5,27€)Nous remercions notre aimable clientèle par avance de vouloir régler en bitcoin.

**correction**

# Passage des paramètres

Qu'est-ce qui se passe à l'appelle suivant:

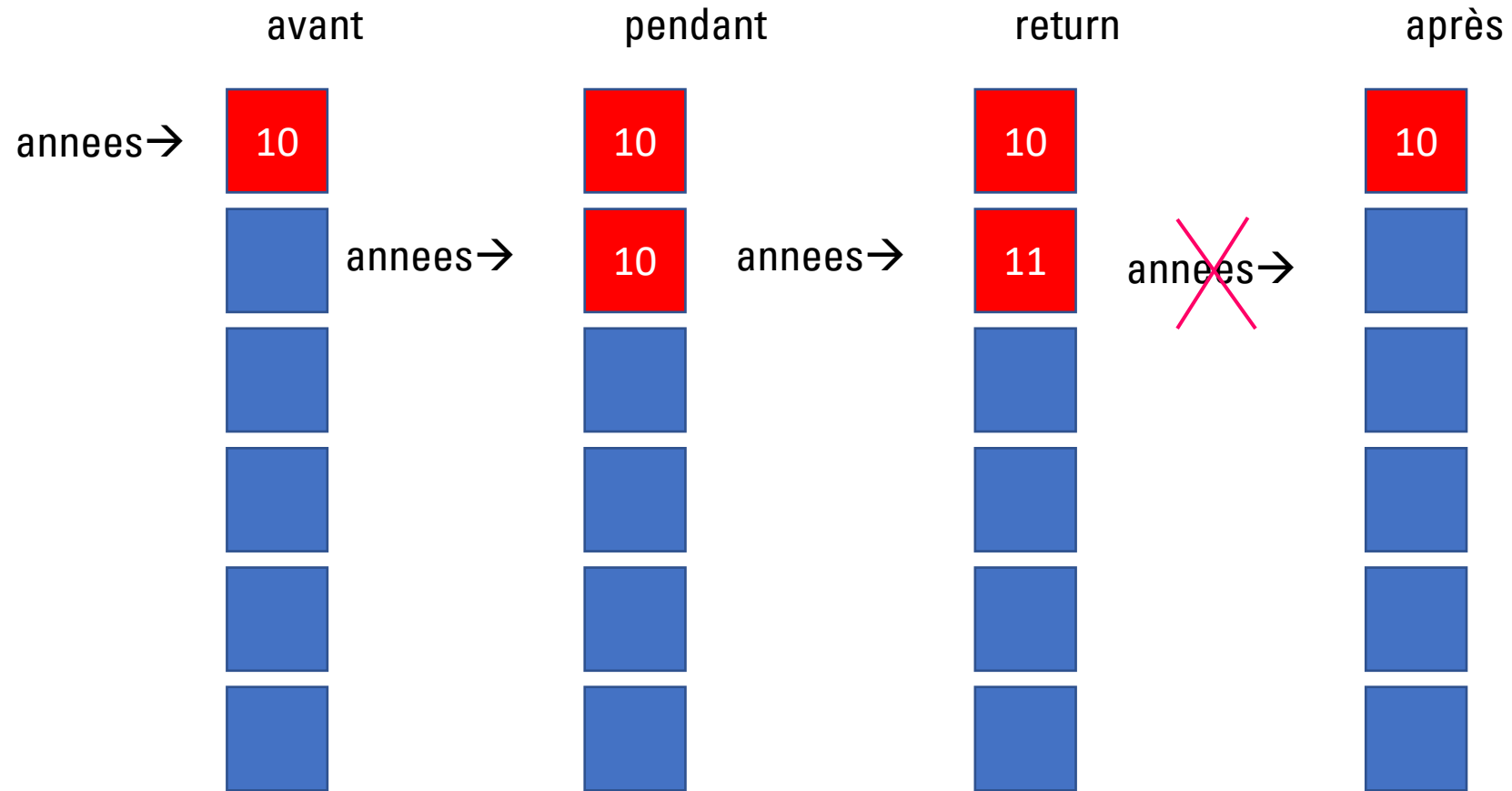
```
{  
    let annees = 10;  
    let toto = 50;  
    let Age = monAgeDansXAnnees(annees);  
}
```

```
function monAgeDansXAnnees(annees)  
{  
    annees = annees +1;  
    return annees;  
}
```

Au niveau du paramètre d'entrée et de sortie de la fonction ?

Est-ce que c'est la même ?

# Passage des paramètres



# Passage des paramètres

```
let nbAnnees = 10;  
afficher("Avant: " + nbAnnees);  
let age = monAgeDansXAnnees(nbAnnees);  
afficher("Après: " + nbAnnees);
```

```
function monAgeDansXAnnees(nbAnnees)  
{  
  nbAnnees*=100;  
  afficher("Pendant: " + nbAnnees);  
  return monAge()+nbAnnees;  
}
```



# Passage des paramètres

```
let nbAnnees = 10;  
afficher("Avant: " + nbAnnees);  
let age = monAgeDansXAnnees(nbAnnees);  
afficher("Après: " + nbAnnees);
```

```
function monAgeDansXAnnees(nbAnnees)  
{  
  nbAnnees*=100;  
  afficher("Pendant: " + nbAnnees);  
  return nbAnnees;  
}
```

**Avant:10**  
**Pendant: 1000**  
**Après: 10**

Le passage d'une variable de type primitif est fait par copie

# Portée des variables

Une variable **let** n'existe que dans le **bloc { }** dans lequel elle est créée.

// méthode, if, switch, for ....

```
{  
  let spock1 = 0;  
  {  
    afficher("block2")  
    let spock2 = 10;  
    afficher(spock1);  
    afficher(spock2);  
  }  
  afficher("block1")  
  afficher(spock1);  
  afficher(spock2);  
}
```

```
afficher("block0")  
afficher(spock1);  
afficher(spock2);
```

# Portée des variables

Une variable **let** n'existe que dans le **bloc { }** dans lequel elle est créée.

// méthode, if, switch, for ....

```
{
  let spock1 = 0;
  {
    afficher("block2")
    let spock2 = 10;
    afficher(spock1);
    afficher(spock2);
  }
  afficher("block1")
  afficher(spock1);
  afficher(spock2);
}
```

```
afficher("block0")
afficher(spock1);
afficher(spock2);
```

```
{
  var spock1 = 0;
  {
    afficher("block2")
    spock2 = 10;
    afficher(spock1);
    afficher(spock2);
  }
  afficher("block1")
  afficher(spock1);
  afficher(spock2);
}
```

```
afficher("block0")
afficher(spock1);
afficher(spock2);
```

# Portée des variables

```
{
  function test()
  {
    afficher("block2")
    var spock2 = 10;
    spock3 = 3;
    afficher(spock1);
    afficher(spock2);
  }

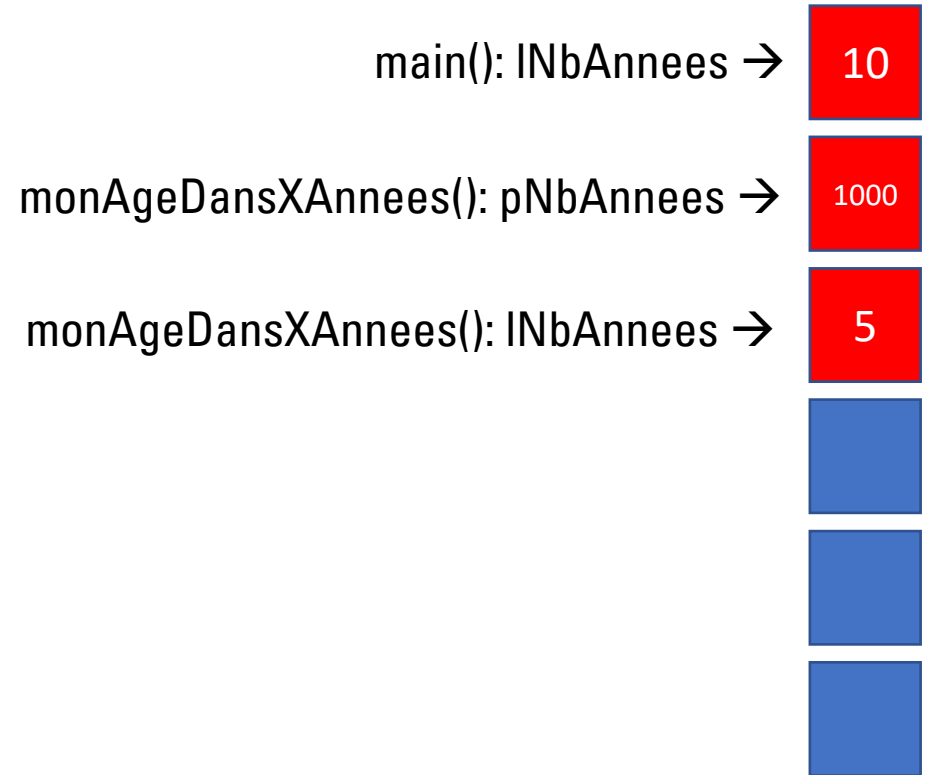
  let spock1 = 0;
  test();

  afficher("block1")
  afficher(spock1);
  afficher(spock2);
  afficher(spock3);
}

afficher("block0")
afficher(spock1);
afficher(spock2);
afficher(spock3);
```

# Portée des variables: lever l'ambiguïté sur la pile

```
let lNbAnnees = 10;  
monAgeDansXAnnees(lNbAnnees);  
}  
  
function monAgeDansXAnnees(pNbAnnees)  
{  
  pNbAnnees*=100;  
  let lNbAnnees = 5;  
  return pNbAnnees;  
}
```



*Mon tip: préfixer par l (local) et p (paramètre)  
// utiliser le refactor ;)*

# Fonctions avec des langages typés

Format complet de description des fonctions: **la signature**

<visibilité> <(qualifiants)> <type\_retour> nomFonction(liste paramètres)

The diagram illustrates the components of a function signature template. Four vertical arrows point upwards from specific examples to the corresponding parts of the template:   
1. An arrow points from 'Public + Private - Protected # Package ~' to '<visibilité>'.   
2. An arrow points from 'static, final' to '<(qualifiants)>'.   
3. An arrow points from 'Primitif, objet, void' to '<type\_retour>'.   
4. An arrow points from 'int param1, String param2' to 'liste paramètres'.

**Public** +  
Private –  
Protected #  
Package ~

**static**, final

**Primitif**, objet, **void**

int param1, String param2

# Branchements

```
if(condition)
{
    // qq chose
}
else
{
    // sinon, pas obligatoire
}
```

```
// autre écriture
if(condition) {
    // qq chose
} else {
    // sinon, pas obligatoire
}
```

# Si alors Sinon

**début**

**réel**  $x, y$

**lire**  $x$

**si**  $x \geq 0$  **alors**

$y \leftarrow \text{sqrt}(x)$

**afficher**  $y$

**sinon**

**afficher** "Valeur indéfinie"

**fin**

let  $x=25, y;$

if( $x \geq 0$ )

{

$y = \text{Math.sqrt}(x);$

**afficher**( $y$ );

}

else

{

**afficher**("Valeur indéfinie");

}



# Switch

---

## L'instruction de test "suivant cas"

Dans l'instruction **suivant condition cas** où  $v_1$  l'expression pouvant prendre plusieurs valeurs  $v_i$  du cas correspondant est exécuté.

Exemple: choix de menu

début

entier menu

lire menu

**suivant menu faire**

cas où 1

| afficher "Menu enfants"

cas où 2

| afficher "Menu végétarien"

autres cas

| afficher "Menu standard"

fin

---

```
let prenom = "toto";
switch(prenom)
{
    case "toto": // prenom==« toto »
        afficher("Rigolo");
        break;
    case "JCVD": // prenom==« JCVD »
        afficher("Aware");
        break;
    case "Dark Vador":
        afficher("Je suis ton père");
        break;
    default:
        afficher("Le silence est d'or");
        break;
}
```

# Opérateurs sur les types simples

- **Arithmétiques (entiers)**: toutes les entrées sont des **entiers** et la sortie est un **entier**.

Nom	Symbole
addition	+
soustraction	-
multiplication	×
division entière	/
reste	mod
inversion de signe	-

- **Arithmétiques (réels)**: au moins une entrée est un **réel** et la sortie est un **réel**.

Nom	Symbole
addition	+
soustraction	-
multiplication	×
division	/
inversion de signe	-

- **Comparaisons**: les deux entrées sont des **entiers**, **caractères\*** ou **réels**. La sortie est un **booléen**.

Nom	Symbole
est égal à	=
est plus petit que	<
est plus grand que	>
est plus petit ou égal à	≤
est plus grand ou égal à	≥

- **Logiques**: toutes les entrées sont des **booléens** et la sortie est un **booléen**.

Nom	Symbole
conjonction	<b>et</b>
disjonction	<b>ou</b>
négation	<b>non</b>

# Opérateurs conditionnels

Opérateur	Description
< , <= , > , >=	Comparaison numérique
== , != , ===	Relationnel (égalité, inégalité)
!	« Non » logique
&& ,    , ^	Opérateurs et/ou/xor



if( x == 0 )    ET PAS if (x=0)

# Opérateurs logique

Table de vérité de ET		
a	b	a ET b
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité de OU		
a	b	a OU b
0	0	0
0	1	1
1	0	1
1	1	1

Table de vérité de XOR (OU exclusif)		
a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

**Début**

**booléen a, b**

**si a ou b alors**

afficher « un est bon »

**Fin si**

**Fin**

```
let a=true, b = true;
```

```
if (a || b)
```

```
{
```

```
    afficher("un est bon");
```

```
}
```

# Opérateurs logique - Négation

**Début**

**booléen a <- vrai**

**Si non a alors**

afficher« négation vraie»

**Sinon**

afficher « vraie négation»

**Fin si**

**Fin**

```
if(!a)
```

```
{
```

```
    afficher("négation vraie");
```

```
}
```

```
else
```

```
{
```

```
    afficher(« vraie négation »);
```

```
}
```

# Égalité stricte === !==

```
let a = 10, b = "10";
```

```
if(a == b)
{
    afficher("ok");
}
else
{
    afficher("ko");
}
```

```
let a = 10, b = "10";
```

```
if(a === b)
{
    afficher("ok");
}
else
{
    afficher("ko");
}
```

# Opérateurs conditionnels

Opérateur	Description
<, <=, >, >=	Comparaison numérique
==, !=, ===	Relationnel (égalité, inégalité)
!	« Non » logique
&&,   , ^	Opérateurs et/ou/xor

**if**( x>=0 && (x!=2 || x!=4) && !vrai) x++;

Quelles valeurs pour x=-1,0,1,2,3,4,5 et vrai=false

Quelles valeurs pour x=-10,200 et vrai = true

# JAVASCRIPT Exercices (lot 2)

1. Créer la fonction **estMajeur()**. Elle prendra un argument de type int. Elle devra retourner un boolean. Si âge  $\geq 18$  elle doit retourner true si âge  $< 18$  elle doit retourner false.  
**Exemple** : âge = 5 ==> false âge = 34 ==> true.
2. Créer la fonction **quelPermis()**. Elle prendra un argument l'âge d'une personne, et devra afficher « Passager d'un véhicule » si la personne a moins de 16 ans, « Eligible conduite accompagnée » si la personne a entre 16 et 18 ans, et « Eligible Permis B » si l'âge est supérieur ou égal à 18 ans. Tester avec 5, 10, 16, 17, 18, 19, 80;
3. Créer la fonction **signe()**. Elle prendra un réel et affichera s'il est positif, négatif ou nul.
4. Créer une fonction **plusGrand2()**. Elle prendra en arguments deux nombres entiers. Elle devra retourner le plus grand des deux.
5. Créer une fonction **plusPetit2()**. Elle prendra en arguments deux nombres entiers. Elle devra retourner le plus petit des deux.
6. Créer une fonction **plusPetit3()**. Elle prendra en arguments trois nombres entiers. Elle devra retourner le plus petit des trois.



# JAVASCRIPT Exercices (lot 2)

7. Créer une fonction **plusGrand3()**. Elle prendra en arguments trois nombre entier. Elle devra retourner le plus grand des trois.
8. Créer une fonction **positifsOuPas()**, qui prend en paramètre 3 réels, et qui renvoie vrai si les 3 sont positifs, sinon elle renvoie faux.
9. Modifier **toutes les fonctions** afin qu'elles ne tiennent que sur une seule ligne.
10. Créer une fonction qui s'appelle **capitale()**. Elle prendra un argument de type string. Elle devra retourner le nom de la capitale des pays suivants :
  - France ==> *Paris*
  - Allemagne ==> *Berlin*
  - Italie ==> *Rome*
  - Maroc ==> *Rabat*
  - Espagne ==> *Madrid*
  - Portugal ==> *Lisbonne*
  - Angleterre ==> *Londres*
  - Tout autre pays ==> *Inconnu*

Il faudra utiliser la structure **SWITCH** pour faire cet exercice.

11. Créer la fonction **mentionBachelier()**. Elle prendra un argument de type pourcentage, qui représente sa moyenne au long de l'année (entre 0,0 et 1,0). Elle devra retourner la mention.

- < 50 %:"pas réussi"
- < 60%:"réussi"
- <70%:"satisfaction"
- < 80%:"distinction"
- < 90%:"une grande distinction"
- <= 100%: "la plus grande distinction"

**Exemple** : argument 1 = 0,85; Résultat : « une grande distinction »

**correction**

# Opérateur ternaire

return Condition ? valeurA:valeurB;

```
if(Condition)
{
    return valeurA;
}
else
{
    return valeurB;
}
```

# Opérateur ternaire

Condition ? exprA:valeurB;

Condition ? Condition2? valeurA:valeurC:valeurB;

Condition ? (Condition2? valeurA:valeurC):valeurB;

return pCondition ? pValeur < 20 ? true: false: true;

# Organisation de fichiers, mesMaths

1. Créez un nouveau fichier, **mesMaths.js**
2. Déplacez tout votre code « mathématique » dans ce nouveau fichier
3. Include dans le head l'inclusion de cette bibliothèque:

```
<head>  
  <script type="text/javascript" src="mesMath.js"></script>  
</head>
```
4. Appelez les méthodes dans le fichiers JSCheatSheet (plusGrand(...))

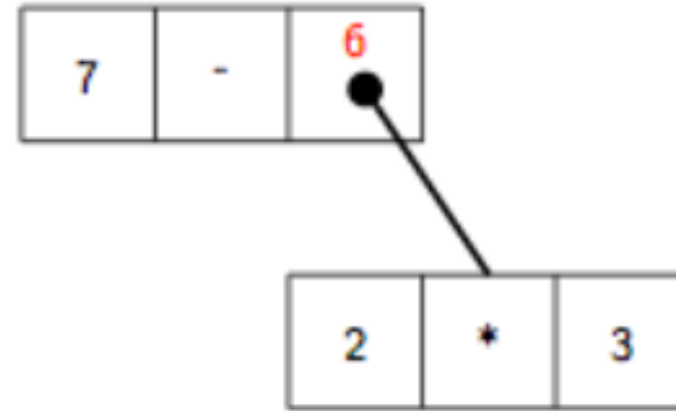
# Précédence des opérateurs

$$7+2*3 = ?$$

# Précédence des opérateurs

$$7+2*3 = ?$$

$$\begin{aligned} & 7 + \underline{2 * 3} \\ = & \underline{7 + 6} \\ = & 13 \end{aligned}$$



$$(7+2)*3 = ?$$

$$\begin{aligned} & (7 + 2) * 3 \\ = & \underline{(7 + 2)} * 3 \\ = & \underline{9} * 3 \\ = & 27 \end{aligned}$$

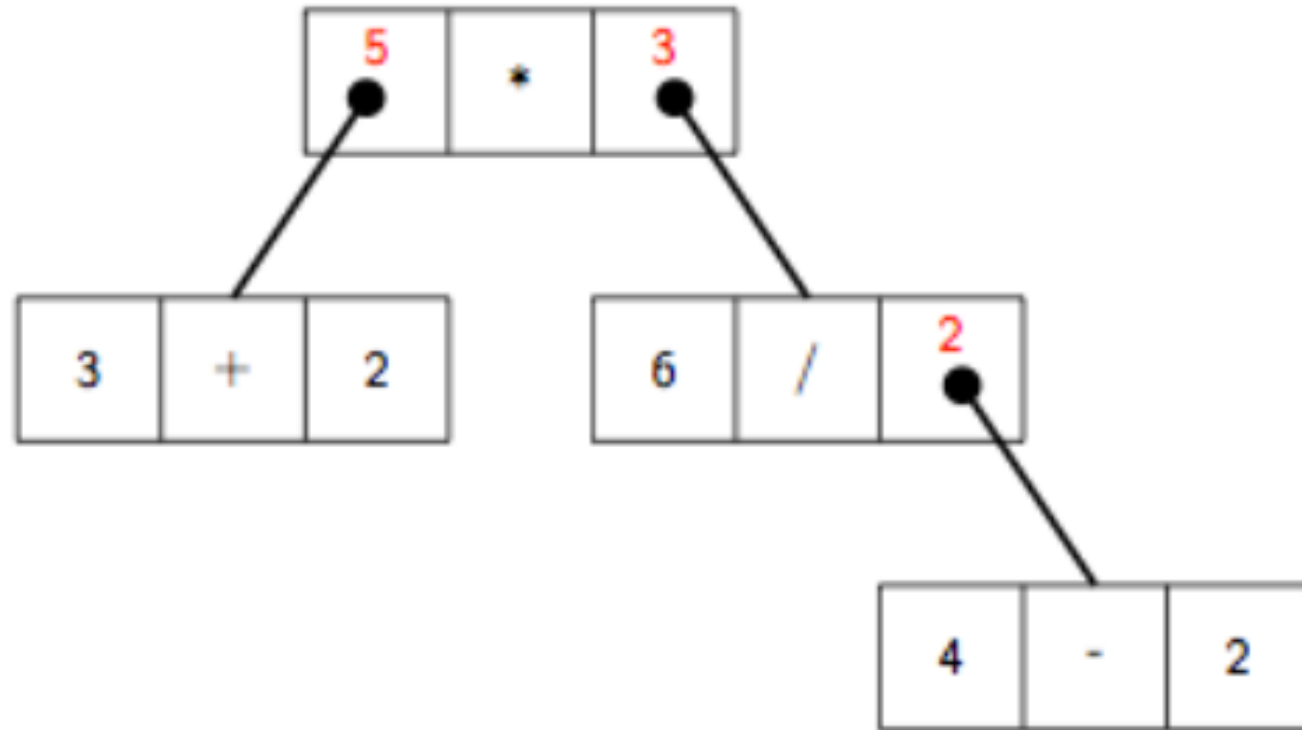


# Précédence des opérateurs

$$(3 + 2) * (6 / (4 - 2)) = ?$$

# Précédence des opérateurs

$$(3 + 2) * (6 / (4 - 2)) = ?$$



# Précédence des opérateurs

$$1-2-3= ?$$

$$x=y=2+3= ?$$

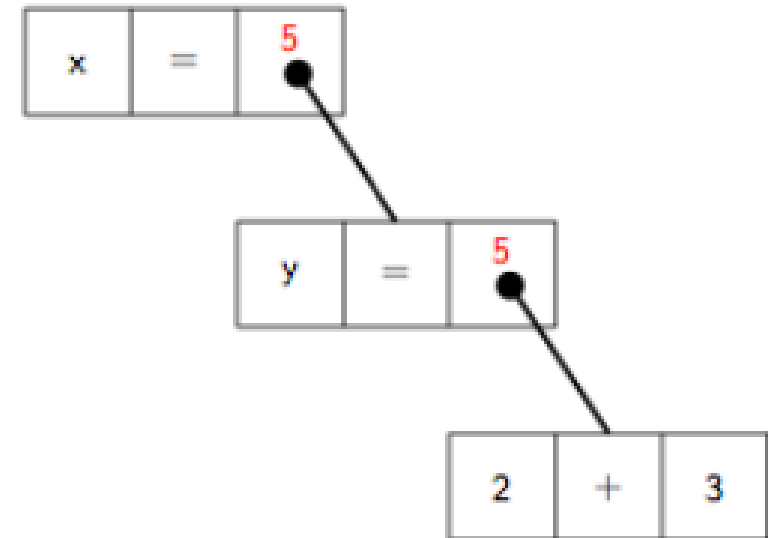
# Précédence des opérateurs

1-2-3= ?

$$\begin{aligned} & \underline{1 - 2} - 3 \\ &= \underline{-1 - 3} \\ &= -4 \end{aligned}$$

x=y=2+3= ?

$$\begin{aligned} & x = y = \underline{2 + 3} \\ &= x = \underline{y = 5} \\ &= \underline{x = 5} \\ &= 5 \end{aligned}$$



# Précédence des opérateurs

Précédence	Type d'opérateur	Associativité	Opérateurs individuels
0	Groupement	Non applicable	( ... )
1	Accès à un membre	Gauche à droite	... . ...
	Accès à un membre calculé	Gauche à droite	... [ ... ]
	<code>new</code> (avec une liste d'arguments)	Non applicable	<code>new</code> ... ( ... )
	Appel de fonction	Gauche à droite	... ( ... )
	Chaînage optionnel	Gauche à droite	?.
2	<code>new</code> (sans liste d'arguments)	Droite à gauche	<code>new</code> ...
3	Incrémementation suffixe	Non applicable	... ++
	Décrémementation suffixe	Non applicable	... --
4	NON logique	Droite à gauche	! ...
	NON binaire	Droite à gauche	~ ...
	Plus unaire	Droite à gauche	+ ...
	Négation unaire	Droite à gauche	- ...
	Incrémementation préfixe	Droite à gauche	++ ...
	Décrémementation préfixe	Droite à gauche	-- ...
	<code>typeof</code>	Droite à gauche	<code>typeof</code> ...
	<code>void</code>	Droite à gauche	<code>void</code> ...
	<code>delete</code>	Droite à gauche	<code>delete</code> ...
	<code>await</code>	Droite à gauche	<code>await</code> ...
5	Exponentiation	Droite à gauche	... ** ...

# Précédence des opérateurs

10 \* 5 + 100 / 10 - 5 + 7 % 2

# Précédence des opérateurs

$10 * 5 + 100 / 10 - 5 + 7 \% 2$

$50 + 100 / 10 - 5 + 7 \% 2$

# Précédence des opérateurs

$10 * 5 + 100 / 10 - 5 + 7 \% 2$

$50 + 100 / 10 - 5 + 7 \% 2$

$50 + 10 - 5 + 7 \% 2$



# Précédence des opérateurs

$10 * 5 + 100 / 10 - 5 + 7 \% 2$

$50 + 100 / 10 - 5 + 7 \% 2$

$50 + 10 - 5 + 7 \% 2$

$50 + 10 - 5 + 1$

# Précédence des opérateurs

$$10 * 5 + 100 / 10 - 5 + 7 \% 2$$

$$50 + 100 / 10 - 5 + 7 \% 2$$

$$50 + 10 - 5 + 7 \% 2$$

$$50 + 10 - 5 + 1$$

$$60 - 5 + 1$$

# Précédence des opérateurs

$$10 * 5 + 100 / 10 - 5 + 7 \% 2$$

$$50 + 100 / 10 - 5 + 7 \% 2$$

$$50 + 10 - 5 + 7 \% 2$$

$$50 + 10 - 5 + 1$$

$$60 - 5 + 1$$

$$55 + 1$$

$$56$$

# Associativité des opérateurs

$x=1$

$x++=?$

$++x=?$

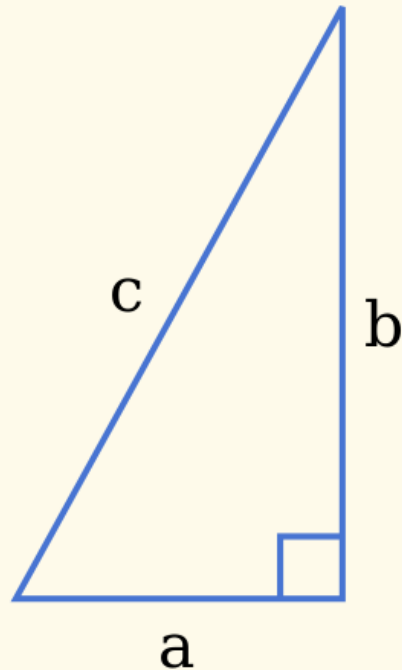
$y=--x+--x-++x*2$

$x=?$

$Y=?$

# JAVASCRIPT Exercices (lot 3)

1. Écrivez **une fonction** qui calcule la longueur de l'hypoténuse suivant le théorème de Pythagore. Pour utiliser cette fonction il faut la paramétrer avec les deux côtés *cote1* et *cote2* suivant le schéma suivant. Utilisez Math.sqrt() pour calculer la racine carré. Affichez le résultat.




$$a^2 + b^2 = c^2$$

# JAVASCRIPT Exercices (lot 3)

2. Écrire **une fonction** qui calcule le résultat d'une équation du second degré, de type  $ax^2+bx+c=0$ . Elle prendra trois paramètres, **a**, **b** et **c**, et doit afficher le résultat. Ne pas traiter le cas où le discriminant est négatif, indiquez « solution complexe ».

**Discriminant** [ [modifier](#) | [modifier le code](#) ]

 Article détaillé : [Discriminant](#).

On considère l'équation suivante, où  $a$ ,  $b$  et  $c$  désignent des nombres réels et  $a$  est différent de 0 :

$$ax^2 + bx + c = 0.$$

On dispose de la définition suivante<sup>5</sup>:

**Définition du discriminant** — Le discriminant de l'équation est la valeur  $\Delta$  définie par :

$$\Delta = b^2 - 4ac.$$

Celui-ci est parfois aussi appelé réalisant, et noté  $\rho$ <sup>6</sup>.

Cette définition est la source du théorème associé à la résolution de l'équation du second degré, dans le cas où l'on recherche des solutions réelles<sup>7</sup>:

**Résolution de l'équation** — Si le **discriminant** est strictement positif, l'équation admet deux solutions  $x_1$  et  $x_2$  données par les formules suivantes :

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \quad \text{et} \quad x_2 = \frac{-b - \sqrt{\Delta}}{2a}.$$

Si le **discriminant** est nul, l'équation admet une **racine** double :

$$ax^2 + bx + c = a \left( x + \frac{b}{2a} \right)^2 \quad \text{et} \quad x_1 = x_2 = -\frac{b}{2a}.$$

Si le **discriminant** est strictement négatif, l'équation n'admet pas de solution réelle, mais admet deux solutions complexes (voir ci-après [Résolution dans l'ensemble des nombres complexes](#)).

[https://fr.wikipedia.org/wiki/%C3%89quation\\_du\\_second\\_degr%C3%A9#Discriminant](https://fr.wikipedia.org/wiki/%C3%89quation_du_second_degr%C3%A9#Discriminant)

```

// Calcul de l'équation du second degré, avec a, b, c donné.
// Ne traite pas le cas en solution avec des nombres complexes.
// voir https://fr.wikipedia.org/wiki/%C3%89quation\_du\_second\_degr%C3%A9#Discriminant

function secondDegre(pA, pB, pC)
{
    // Cas spécial a==0 => équation du premier degré
    if (pA == 0)
    {
        afficher (« Cas non géré car a == 0 »);
        return;
    }

    // Calcul du discriminant
    let lDiscriminant = pB * pB - 4 * pA * pC;
    let lX1, lX2;
    if (lDiscriminant < 0)
    {
        afficher("Solution complexe.");
    }

    else // 2 solutions existent
    {
        let l2A = 2*pA;
        if (lDiscriminant > 0)
        {
            // (-b +/- D) / 2a
            let lSqrtDisc = Math.sqrt(lDiscriminant);
            lX1 = (-pB + lSqrtDisc) / l2A;
            lX2 = (-pB - lSqrtDisc) / l2A;
        }
        else // Une seule solution
        {
            if (lDiscriminant == 0)
            {
                // -b / 2a
                lX1 = lX2 = (-pB / (l2A));
            }

            // Affichage des résultats
            afficher("x1:" + lX1);
            afficher("x2:" + lX2);
        }
    }
}

```

# JAVASCRIPT Exercices (lot 3)

3. Utilisez un formulaire pour tester les valeurs pour Pythagore et l'équation du second degré

```
<body style="text-align: center;">
  <h1 style="color: ■ green;">
    JS Sheet Cheat
  </h1>
  <p style="color: ■ orange;" id="principal"></p>

  <FORM NAME="monFormulaire">
    <INPUT TYPE="number" NAME="a" VALUE="">Valeur de a:<BR>
    <INPUT TYPE="number" NAME="b" VALUE="">Valeur de b:<BR>
    <INPUT TYPE="number" NAME="c" VALUE="">Valeur de c:<BR>
    <INPUT TYPE="button" NAME="monBouton" VALUE="Calculer équation" onClick="traitementFormulaire(monFormulaire)">
  </FORM>

  <script>

    function traitementFormulaire(unFormulaire)
    {
      afficher(unFormulaire.a.value);
      afficher(unFormulaire.b.value);
      afficher(unFormulaire.c.value);
    }
  </script>
```



# Paradigmes de programmation

## Fonctionnelle

Fondée sur l'évaluation d'expressions

Ex: Lisp, Caml, ML, Haskell

```
# let multiply n list =  
    let f x =  
        n * x in  
    List.map f list;;
```

## Impératif

Fondé sur l'exécution d'instructions qui modifient l'état de la mémoire

```
Pixel p1 = new Pixel(1,3);  
p1.printOnScreen();
```

## Orientée-objet

C'est un style de programmation où l'on considère que des composants autonomes (les objets) disposent de ressources et de moyens d'interactions entre eux

# Types de base (primitives)

Type	Taille	Valeurs
boolean	1 bit	« true » ou « false »
char	16 bits	Unicode \u0000 à \uFFFF
byte	8 bits	-128 à 127
short	16 bits	-32768 à 32767
int	32 bits	-2147483648 à 2147483647
long	64 bits	-9223372036854775808 à 9223372036854775807
float	32 bits	+/- 3.402E+38 à +/-1.402E-45
double	64 bits	+/- 1.798E+308 à +/- 4.941E-324

<- JAVA

JS

$2^{53} = 9007199254740991$ : safe int

La propriété statique `MAX_VALUE` vaut environ  $1.79E+308$  (soit  $2^{1024}$ ).

Les valeurs supérieures à `MAX_VALUE` sont représentées par [Infinity](#) (pour l'infini).

<- JS

# Exercice multiplications

Créer une fonction **TableMultiplication()**. Elle prendra en arguments un nombre. Elle devra afficher la table de multiplication de 0 à 10 du nombre passé en paramètre de la manière suivante:

Table de multiplication du nombre A:

A x 0 = 0

A x 1 = ..

...

A x 10 = ..

Quelle jolie table !

## L'instruction de boucle "pour"

L'instruction *pour* est utilisée lorsque le nombre d'itérations est **connu à l'avance**: elle initialise un **compteur**, l'incrémente après chaque exécution du bloc d'instructions, et vérifie que le compteur ne dépasse pas la borne supérieure.

Exemple: Somme des entiers de 1 à  $n$

```
début
  entier  $n, s, i$ 
  lire  $n$ 
   $s \leftarrow 0$ 
  pour  $i$  de 1 à  $n$  faire
     $s \leftarrow s + i$ 
  afficher  $s$ 
fin
```

## L'instruction de boucle "tant que"

La boucle *tant que* est utilisée lorsque le nombre d'itérations **n'est pas connu à l'avance**: elle exécute le bloc d'instructions tant que la condition reste vraie.

Exemple: Somme des entrées saisies par l'utilisateur (version "tant que")

```
début
  entier  $n \leftarrow 1, s \leftarrow 0$ 
  tant que  $n \neq 0$  faire
    afficher "Entrer un entier (0 pour arrêter) : "
    lire  $n$ 
     $s \leftarrow s + n$ 
  afficher  $s$ 
fin
```

## L'instruction de boucle "répéter jusqu'à"

La boucle *répéter jusqu'à* est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance, et qu'il faut lancer **au moins une** exécution du bloc d'instructions. Elle exécute le bloc jusqu'à ce que la condition d'arrêt devienne vraie.

Exemple: Somme des entrées saisies par l'utilisateur (version "répéter jusqu'à")

```
début
  entier  $n, s \leftarrow 0$ 
  répéter
    lire  $n$ 
     $s \leftarrow s + n$ 
  jusqu'à  $n = 0$ 
  afficher  $s$ 
fin
```

# Boucles

```
for(let i=0; i<10; i++)
{
  // faire quelque chose
}
```

```
while(condition)
{
  // faire quelque chose
}
```

```
do
{
  // faire quelque chose
} while (condition);
```

## Exercices multiplications 2

Pour chaque exercice **créer une fonction** qui affiche la table de multiplication, en prenant **2 paramètres**, le multiplicateur et le nombre maximum de la table:

1. Avec une boucle for
2. Avec un Do while
3. Avec un while

# Exercices multiplications 3

**Écrire une fonction** qui affiche toutes les possibilités d'un code PIN d'une carte bancaire (4 chiffres de 0-9)

1. Avec une boucle for | Do | While
2. Mesurer le temps
  1. Avec affichage
  2. Sans affichage

```
let lDate = new Date();
let lTimeStart = lDate.getTime(); // temps en millisecondes depuis 1970
document.write(lTimeStart);

// ... faire quelque choses
let lDateStop = new Date();
let lTimeStop = lDateStop.getTime(); // temps en millisecondes depuis 1970
```

# Exercices multiplications 4

**Écrire une fonction** qui affiche toutes les possibilités d'un code PIN d'une carte bancaire nouvelle génération hexadécimal (4 chiffres de 0-9-A-B-C-D-E-F)

1. Avec une boucle For | Do | While
2. Mesurer le temps
  1. Avec affichage
  2. Sans affichage

```
let lDateStart = new Date();  
let lTimeStart = lDateStart.getTime(); // temps en millisecondes depuis 1970  
  
// ... faire quelque choses  
let lDateStop = new Date();  
let lTimeStop = lDateStop.getTime(); // temps en millisecondes depuis 1970  
// calculer la différence
```

# Chaîne de caractères

```
let maString = "abcd";  
// Obtenir la taille d'une chaine de caractère  
let taille = maString.length;  
// Obtenir le caractère numéro 3 d'une chaine  
let unCharactere = maString.charAt(3); // retourne 'd', commence à 0  
  
//Parcourir une chaine de caractère  
for (let i = 0; i < maString.length; i++)  
{  
    //c prendra à chaque itération la valeur d'un caractère de la chaine : a puis b puis c ...  
    let c = maString.charAt(i);  
    // Faire quelque chose  
}
```

<https://htmlcheatsheet.com/js/>



# Exercices chaine de caractères

Créer un fichier **mesStrings.js**. Pour chaque exercice **créer une fonction** qui:

1. Retourne la phrase en entrée sans les 'e'
2. Retourne le nombre de 'a'
3. Retourne la chaine à l'envers «toto» -> «otot»
4. Compte le nombre de majuscule (utiliser `toUpperCase`)
5. Supprime les voyelles (chaine d'entrée en minuscule uniquement)
6. Retourne la chaine sans les majuscules
7. Plus grand caractère de la chaine (chaine d'entrée en minuscule)
8. Retirer les espaces mais uniquement au début de la chaine

" toto en vacances" -> " toto en vacances"

*Pour les plus rapides :*

1. Retirer les espaces au début et à la fin (`trim`)  
" toto en vacances " -> "toto en vacances"
2. Indique si une chaine est un palindrome
3. Vérifie si une chaine est un mot de passe valide: 8 chars min, 1 maj | min | chiffre

**correction**

# Passage des paramètres

Qu'est-ce qui se passe à l'appelle suivant:

```
let lMonNom = "Pince Me";  
afficher("Avant: " + lMonNom);  
changementDeNom(lMonNom);  
afficher("Après: " + lMonNom);
```

```
function changementDeNom(pMonNom)  
{  
    pMonNom = "Pince moi";  
}
```

Au niveau des paramètres d'entrée et de sortie de la fonction ?  
Expliquez et vérifiez.

# Passage des paramètres

Qu'est-ce qui se passe à l'appelle suivant:

```
let lMonNom = "Pince Me";  
afficher("Avant: " + lMonNom);  
changementDeNom(lMonNom);  
afficher("Après: " + lMonNom);
```

```
function changementDeNom(pMonNom)  
{  
  pMonNom = "Pince moi";  
}
```



**Avant:Pince Me**  
**Après: Pince Me**

# Passage des paramètres

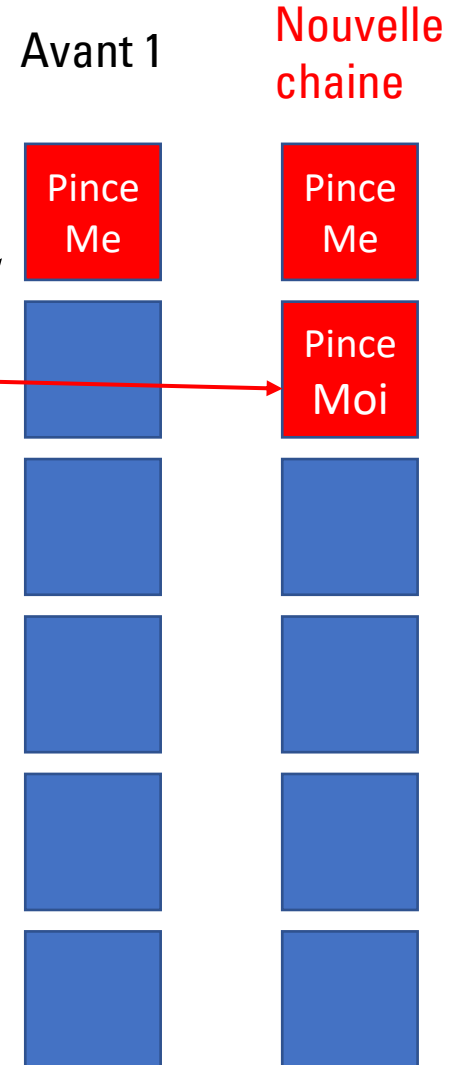
Qu'est-ce qui se passe à l'appelle suivant:

```
let lMonNom = "Pince Me";  
changementDeNom(lMonNom);
```

```
function changementDeNom(pMonNom)  
{  
  pMonNom = "Pince moi";  
}
```

**Avant: Pince Me**  
**Après: Pince Me**

lMonNom →  
ChangementDeNom(): pMonNom



Les objets sont passés par références, c'est-à-dire une copie de l'adresse

# Passage des paramètres

Autre ex.

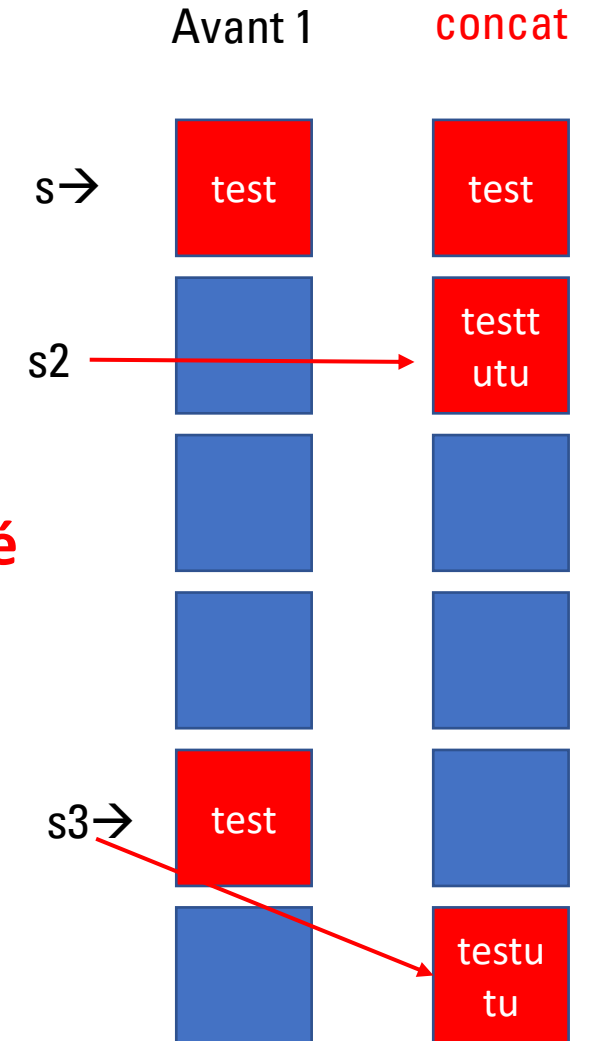
```
let s = « test »;  
let s2 = s + « tutu »;
```

Afficher(s) → test

```
let s3 = « test »;  
s3 += « tutu »;
```

Afficher(s3) → testtutu

**MAIS la classe String est immuable**  
**Une fois l'objet créé il ne peut être modifié**



# Tableaux

```
afficher("Les tableaux");
//Déclaration d'un tableau. C'est une liste d'éléments
let monTab = [];
//initialise et remplit le tableau de taille 6
let monTab2 = [3, 5, 6, 4, 2, 8];
afficher("monTab2: " + monTab2);

//Lire dans un tableau.
// Attention les indices commencent à 0
let maVar = monTab2[2]; // lit la 3ème valeur, 6
monTab2[0] = 4; //change la première valeur de 3 à 4
//Taille du tableau

let size = monTab2.length;
afficher(size);
// ajouter une valeur en fin de tableau
monTab2.push(50);
// ou
monTab2[monTab2.length] = 60;
afficher(monTab2);

// supprimer en fin de tableau
monTab2.pop();
afficher(monTab2);
```

```
// supprimer en début de tableau
monTab2.shift();
afficher(monTab2);

// ajouter en début de tableau
monTab2.unshift("du texte");
afficher(monTab2);

// ajoute des éléments où on le souhaite
// en supprimant d'autres si on le souhaite
//param1: début d'ajout param2: combien à supprimer
//puis la liste des éléments à ajouter
monTab2.splice(1,1, "toto", "titi");
afficher(monTab2);

// ne garde de 4 (inclus) à 7 (exclu)
monTab = monTab2.slice(4,7);
afficher(monTab);

// Conversion en chaîne de caractère
let maChaine = monTab.toString();
afficher(maChaine);

let toto = "toto";
afficher(toto[1]);
```

# Les tableaux

- Ce sont des **ensembles de données**
  - De type identique ou différents

```
tab = [0,0,0,0,0,0]; // construction d'une maison sur le terrain
// tableau de 6 cellules [0,0,0,0,0,0]
// tab: une adresse vers la maison (50 route de ..)
tab[0] = 12; // remplissage d'une des pièces de la maison
// [12, 0, 0, 0, 0, 0]
tab[2] = 2; // remplissage d'une autre pièce de la maison
// [12, 0, 2, 0, 0, 0]
...
let y = tab[1]; // récupération d'une valeur (ex. droite) : y=56
```

```
for(indice=0; indice<tab.length; indice++)
{
    afficher( tab[indice] );
    // ou un autre traitement
}
```

Numbers [] tab

caracters [] tab

String[] tab

indice

Accès à  
la case

Parcourir:  
Faire varier  
l'indice

Nombre d'éléments: tab.length

Nombre d'éléments: tab.length					
12	56	2	0	120	10
z	e	R	t	A	b
toto	titi	tutu	lala	lulu	lele
0	1	2	3	4	5
de 0 à tab.length-1					
tab[0]	tab[1]	tab[2]	tab[3]	tab[4]	tab[5]
Indice = 0 puis 1 puis 2, puis .. tab.length-1					



# Les variables aléatoires (random)

```
// génère une chiffre décimal aléatoire entre [0 and 1[  
let lRand= Math.random();  
afficher(lRand);
```

```
// Donne un chiffre entier entre [0 et 5]  
lRand = Math.floor(Math.random() * 6);  
afficher(lRand);
```

Créer une fonction **monRand**, en fonction d'un min et d'un max, retourne un nombre entier de façon aléatoire compris dans cet intervalle.

[min, max]

[5,10]

[-50, 20]

[-100, -20]



# Tableaux – Exercices 1

Créez un fichier JS ArrayUtils qui contiendra les exercices sur les tableaux.

**Implémentez** les méthodes suivantes

- *Méthodes à appeler depuis votre fichier HTML « programme principal »*
- *Initialisez les tableaux et les passer en paramètres depuis ce fichier HTML*

*/\*\* Remplit le tableau de n cases de valeurs aléatoires comprises entre 0 et 99 \*/*

*fillTab(tab, size)*

*/\*\* Affiche le tableau dans sur 1 seule ligne de type [valeur1: xx; valeur2:yy; ...] \*/*

*printTab(tab)*

*/\*\* Retourne la valeur maximum du tableau \*/*

*getMax(tab)*

*/\*\* Permute l'emplacement i et j dans le tableau \*/*

*permute(tab, i, j)*

*/\*\* Effectue une copie du tableau. Pas de slice, concat, ou autre.\*/*

*copy(tab)*

# Tableaux – Exercices 2

Dans le fichier JS ArrayUtils ajoutez les exercices suivants.

**Implémentez** les méthodes suivantes qui

- *Méthodes à appeler depuis votre fichier HTML « programme principal »*
- *Initialisez les tableaux et les passer en paramètres depuis ce fichier HTML*

1. Retourne la somme
2. Retourne la moyenne
3. Affiche les valeurs supérieures à la moyenne (ne retourne rien)
4. Retourne le nombre d'occurrence de la valeur maximale (Version  $O(2n)$  et  $O(n)$ )
5. */\*\* Crée et retourne un nouveau tableau qui est la concaténation des 2 tableaux  
NE pas utiliser JOIN, SPLICE, SLICE ! \*\*/  
fusion(tab1, tab2)*

# Palindrome

Créer les 3 fonctions suivantes:

1. `palindromeFor(String)`
2. `palindromeWhile(String)`
3. `palindromeDoWhile(String)`

Qui vérifient que le paramètre est bien un palindrome, en utilisant respectivement des boucles `for`, `while` et `do while`.

## Exemple:

- Elu par cette crapule
- Je ne suis pas un palindrome
- Noël a trop par rapport à Léon
- lol

# La boucle par la récursivité

$$\text{Somme}(10) = 1 + 2 + 3 + \dots + 10$$

# La boucle par la récursivité

$$\text{Somme}(10) = 1 + 2 + 3 + \dots + 10$$

→ Pour  $i$  allant de 1 ... 10  
somme +=  $i$  OK

# La boucle par la récursivité

$$\text{Somme}(10) = 1 + 2 + 3 + \dots + 10$$

→ Pour  $i$  allant de 1 ... 10  
somme +=  $i$  OK

→  $\text{Somme}(10) = 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$

# La boucle par la récursivité

$$\text{Somme}(10) = 1 + 2 + 3 + \dots + 10$$

→ Pour  $i$  allant de 1 ... 10  
somme +=  $i$  OK

$$\begin{aligned} \rightarrow \text{Somme}(10) &= 10 + \underbrace{9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1}_{\text{Somme}(9)} \\ &\quad \underbrace{\hspace{1.5cm}}_{\text{Somme}(8)} \quad \underbrace{\hspace{1.5cm}}_{\text{Somme}(1)} \end{aligned}$$



# La boucle par la récursivité

$$\text{Somme}(10) = 1 + 2 + 3 + \dots + 10$$

→ Pour  $i$  allant de 1 ... 10  
somme +=  $i$  OK

$$\rightarrow \text{Somme}(10) = 10 + \underbrace{9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1}_{\text{Somme}(9)}$$

$\underbrace{\hspace{10em}}_{\text{Somme}(8)} \quad \underbrace{\hspace{2em}}_{\text{Somme}(1)}$

$$\text{Somme}(10) = 10 + \text{Somme}(9)$$

$$\text{Somme}(9) = 9 + \text{Somme}(8)$$

...

$$\text{Somme}(1) = 1 + \text{Somme}(0)$$

$$\text{Somme}(0) = 0 \text{ (condition d'arrêt de la récursivité)}$$

$$\text{Somme}(i) = i + \text{Somme}(i-1)$$

# La boucle par la récursivité

```
afficher("Somme: " + sommeRecursive(10));
```

```
sommeRecursive(pNumber)
{
    // on ajoute le précédent si pas « à la fin » de la somme
    if(pNumber > 0) // condition d'arrêt
        return pNumber + sommeRecursive(pNumber-1);
    else
        return 0; // arrêt
}
```

Permet d'avancer  
« d'indice »

Permet de  
« boucler », appel en  
cascade, jusqu'à la  
condition d'arrêt

# La boucle par la récursivité en une ligne ...

# La boucle par la récursivité

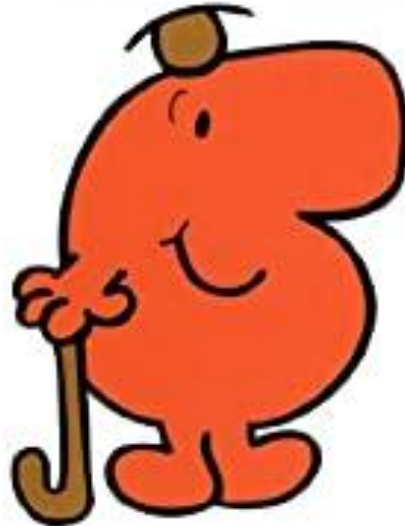
// en vrai ça ne sert à rien de commenter ...

```
function s (x)
{
    return (x > 0 ) ? x + s (x-1):0;
}
```

# La boucle par la récursivité

Et à l'envers, de 0 à 10 ?

**M. À L'ENVERS**



# La boucle par la récursivité

```
afficher("Somme: " + sommeRecursive(10, 0));
```

```
sommeRecursive(pMax, pCurrent)
{
    // on ajoute le suivant si pas à la fin de la somme
    if(pCurrent < pMax) // condition d'arrêt
        return pCurrent + sommeRecursive(pMax, pCurrent+1);
    else
        return pCurrent; // arrêt
}
```

Permet d'avancer  
« d'indice »

Permet de  
« boucler », appel en  
cascade, jusqu'à la  
condition d'arrêt

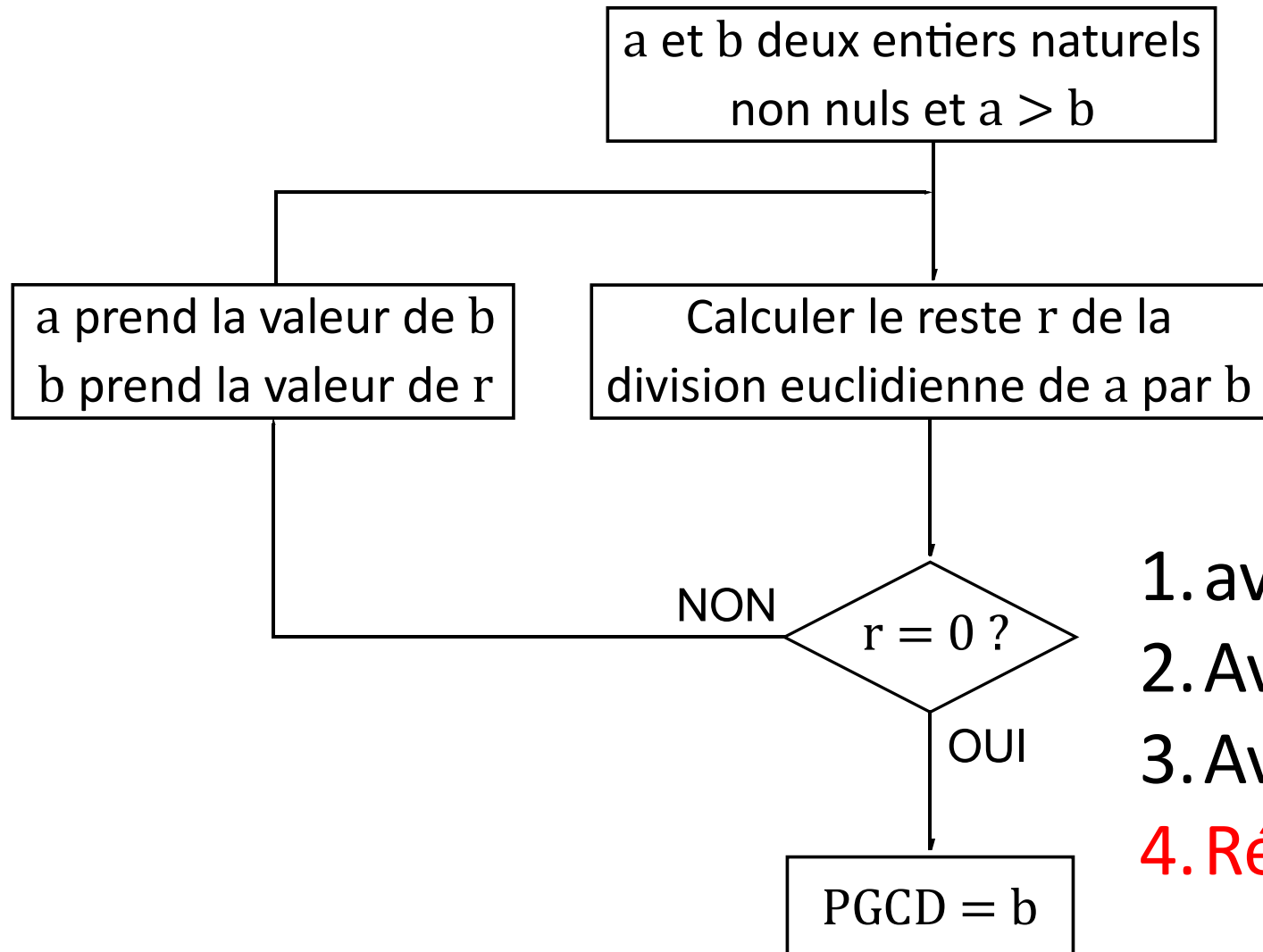
# La boucle par la récursivité

```
afficher("Somme: " + somme (10));
```

```
sommeRecursive(pMax, pCurrent)
{
    // on ajoute le suivant si pas à la fin de la somme
    if(pCurrent < pMax) // condition d'arrêt
        return pCurrent + sommeRecursive(pMax, pCurrent+1);
    else
        return pCurrent; // arrêt
}
```

```
somme (pMax) // une deuxième fonction plus parlante
{
    return sommeRecursive(pMax, 0);
}
```

# C'est vrai ça ? Vérifiez le !



1. avec un **for**
2. Avec un **while**
3. Avec un **do while**
4. **Récuratif**



# Palindrome

Ajouter une fonction `palindromeRécursif` qui vérifient que le paramètre est bien un palindrome, en utilisant la récursivité.

## Exemple:

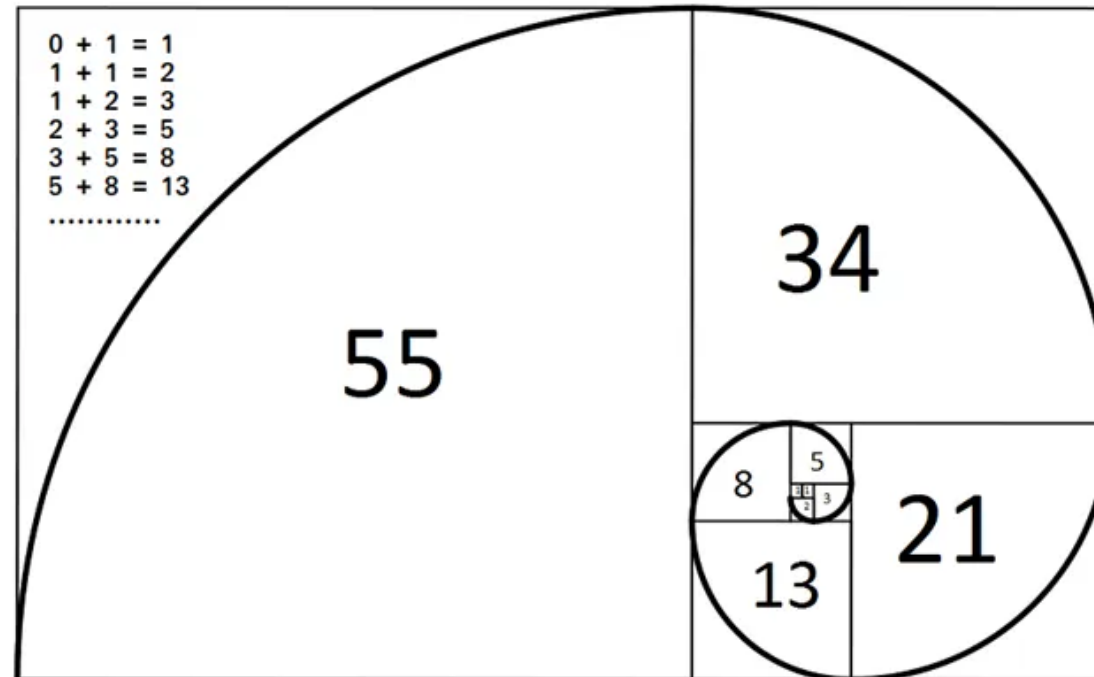
- Elu par cette crapule
- Je ne suis pas un palindrome
- Noël a trop par rapport à Léon
- lol

# Récurtivité

Ecrire une **fonction réursive** qui calcule la suite de **Fibonacci**:

La suite est définie par  $\mathcal{F}_0 = 0$ ,  $\mathcal{F}_1 = 1$ , et  $\mathcal{F}_n = \mathcal{F}_{n-1} + \mathcal{F}_{n-2}$ , pour  $n > 1$ .

$\mathcal{F}_0$	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$	$\mathcal{F}_6$	$\mathcal{F}_7$	$\mathcal{F}_8$	$\mathcal{F}_9$	$\mathcal{F}_{10}$	$\mathcal{F}_{11}$	$\mathcal{F}_{12}$	$\mathcal{F}_{13}$	$\mathcal{F}_{14}$	$\mathcal{F}_{15}$	$\mathcal{F}_{16}$	...	$\mathcal{F}_n$
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	...	$\mathcal{F}_{n-1} + \mathcal{F}_{n-2}$



# Récurtivité

Ecrire une **fonction** qui calcule la suite de **factorielle de n**:

$$n! = 1 \times 2 \times 3 \times \dots \times n$$

**Exemple (à vérifier):**

$$1! = 1$$

$$2! = 1 \times 2 = 2$$

$$3! = 1 \times 2 \times 3 = 6$$

$$4! = 1 \times 2 \times 3 \times 4 = 24$$

$$10! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10 = 3\,628\,800$$

Avec un:

1. for
2. while
3. do while
4. récursive

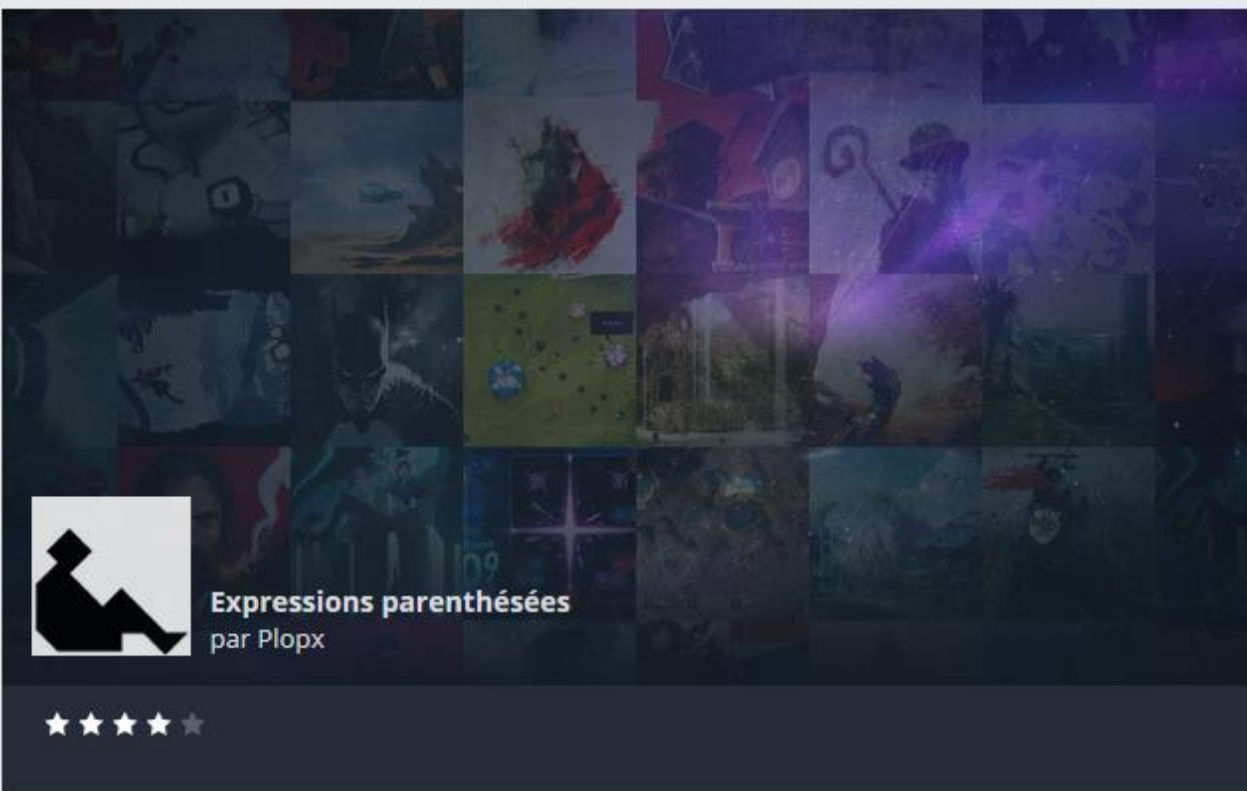
# Défis codingame

1. <https://www.codingame.com/training/easy/power-of-thor-episode-1>
2. <https://www.codingame.com/training/easy/the-descent>
3. <https://www.codingame.com/training/easy/temperatures>
4. <https://www.codingame.com/training/easy/sudoku-validator>



# Défis codingame 2

1. <https://www.codingame.com/training/easy/mime-type>
2. <https://www.codingame.com/training/easy/horse-racing-duals>
3. <https://www.codingame.com/training/easy/brackets-extreme-edition>







*That's all Folks!*