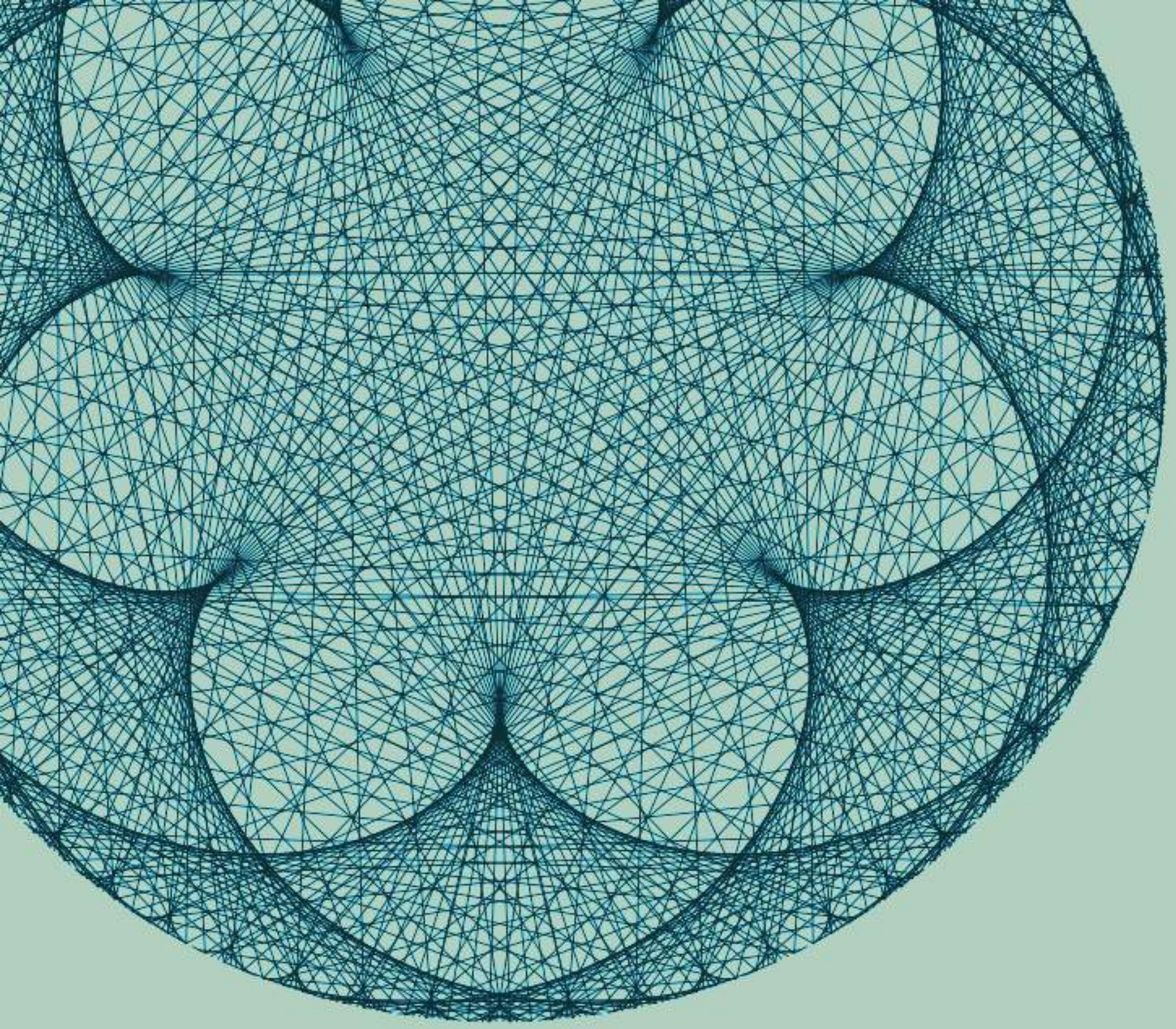


Algorithmique

OUTOFPLUTO Anthony Baillard Novembre 2017

Algorithmique

1. Introduction
Qu'est-ce que l'algorithmique ?
Brève histoire de l'algorithmique
Algobox et Scratch

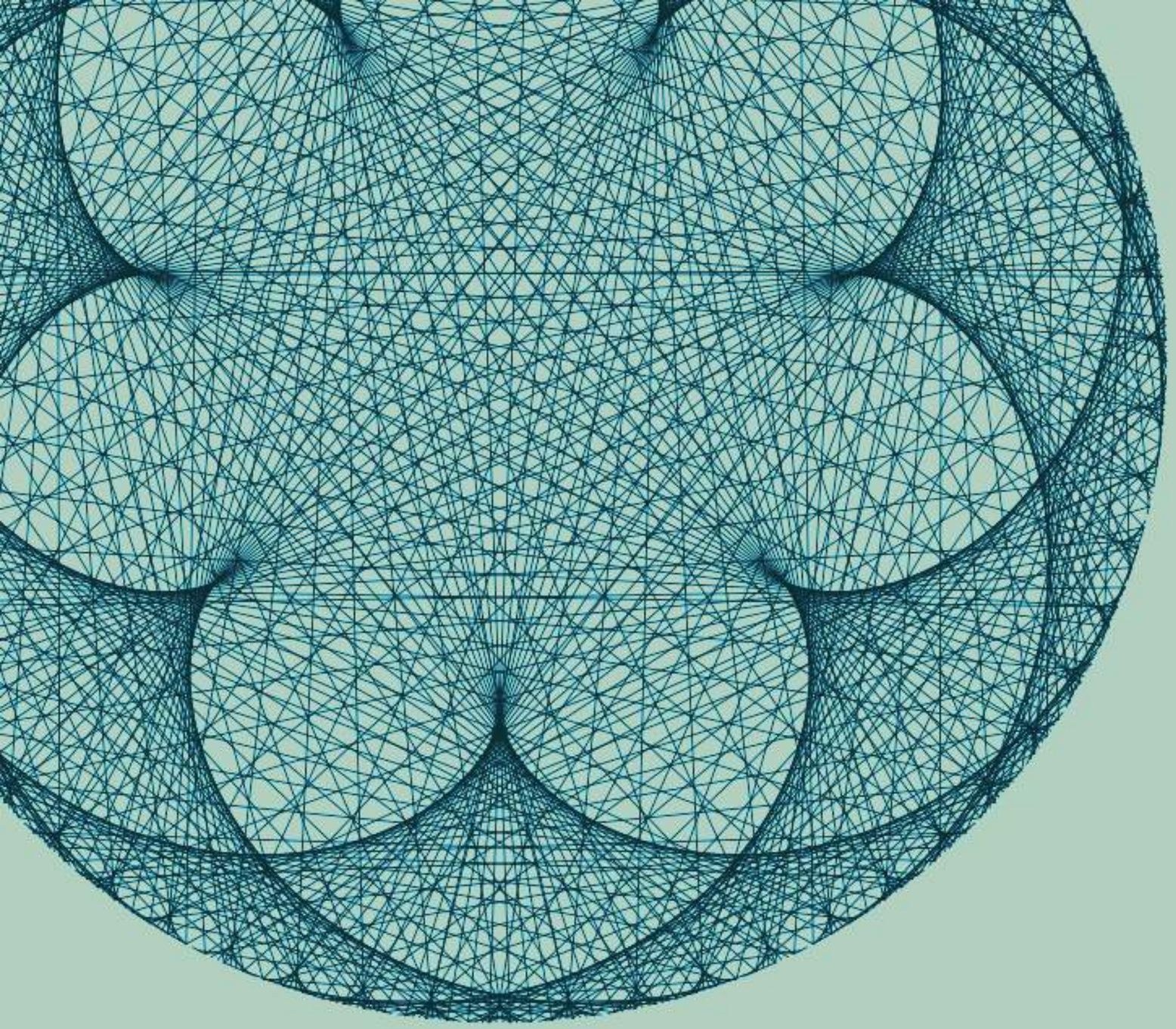
2. La représentation des algorithmes
Les organigrammes
Les pseudo-langages
Atelier n°1

3. Les structures de données simples
Les constantes
Les variables
Les tableaux
Atelier n°2
Opérateurs simples

4. Les structures de contrôle
Les structures séquentielles
Les structures conditionnelles
Les boucles
Atelier n°3

5. Le représentations complexes
Tableaux multidimensionnels
Listes chaînées
Piles et files
Arbres
Ateliers n°4 et 5
Graphes

6. Conclusion
Atelier n°6
L'implémentation
Heuristiques et complexité
Lectures et références



Introduction

L’algorithmique, qu’est-ce que c’est ?

L’algorithmique, qu’est-ce que c’est ?

Définitions

- **Encyclopedia Universalis** Un *algorithme* est la spécification d’un schéma de calcul, sous forme d’une suite [finie] d’opérations élémentaires obéissant à un enchaînement déterminé.
- Une *opération élémentaire* est une étape de calcul définie (non ambiguë) et indivisible en elle-même (ex: une addition)

Analyse et vocabulaire

Pour étudier les algorithmes, plusieurs outils avec leur vocabulaire spécifique ont été formalisés :

- les structures algorithmiques
- la terminaison, la correction et la complétude
- la complexité

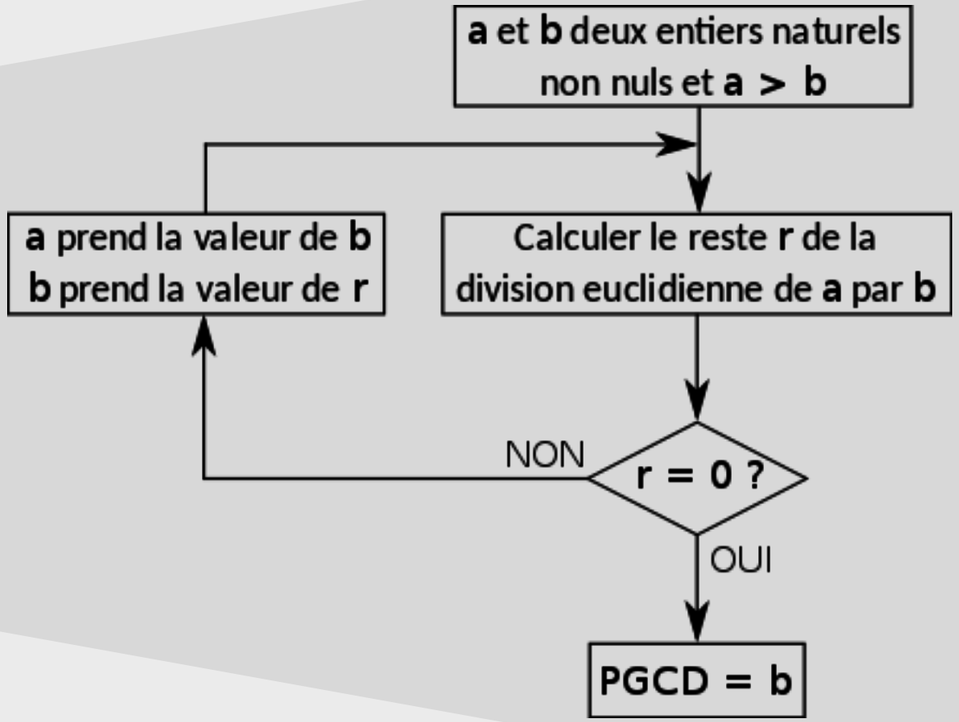
Implémentation informatique

Afin d’être exécuté par un ordinateur, un algorithme doit être traduit dans un langage informatique. C’est *l’implémentation* ou le *codage*.
La méthode est différente pour chaque langage.

Définition restrictive pour le reste du cours : Méthode de résolution d’un problème pouvant être réalisé par un ordinateur

L’algorithmique avant l’ordinateur

IIIème millénaire av J.-C.	Méthodes de calcul	Méthodes de calcul
-300	Grèce Antique (Euclide)	Plus Grand Commun Diviseur
-250	Grèce Antique (Archimède)	Calcul de Pi
800	Perse Antique	Equations du second degré
XIIème siècle	Europe	Principes de convergence et apparition du mot *algorismus*
XVIIème siècle	Europe	Concept de programme
XIXème siècle	Europe (Ada Lovelace)	Premier programme algorithmique (nombre de Bernouilli)



L’algorithmique au XXème et XXIème siècle

Années 30

Plusieurs mathématiciens formalisent le concept d’algorithme (Gödel, Church, Turing, Post).
Ce sont des modèles abstraits à ressources infinies.

Deuxième moitié du XXème siècle

Avec l’avènement de l’informatique, les algorithmes “s’adaptent” aux ordinateurs

- Traitement sur un certain nombre **fini** de données
- Opérations **effectives** (réalisable par une machine comme par un humain en un temps fini)
- Les algorithmes doivent toujours **se terminer**
- Les algorithmes sont **déterministes** (les mêmes données donnent toujours la même suite d’opérations)

Quelques réflexions

Quel avenir pour l'algorithmique ?

Tous les problèmes mathématiques ont-ils une solution algorithmique ?

Que faire lorsqu'un problème a une complexité trop élevée pour arriver à la solution en un temps raisonnable ?

Qu'est-ce que la computation quantique ? A quoi sert-elle/peut-elle servir ?

Algobox

Pour pouvoir réaliser des ateliers durant le cours, pensez à installer dès maintenant Algobox.
Compatible Windows, Mac OS et Linux.

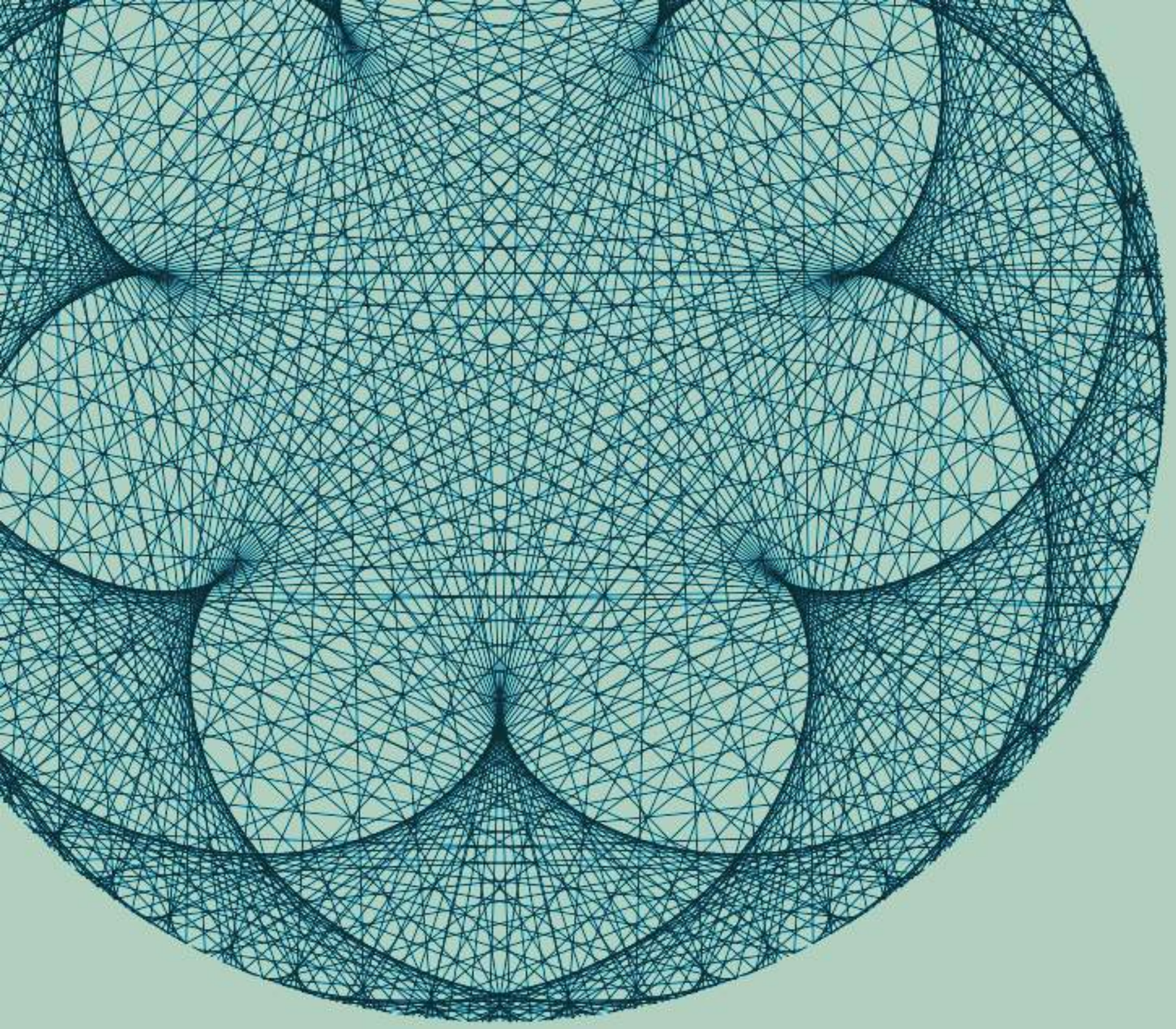
Téléchargement :
http://www.xm1math.net/algobox/download.html

Documentation à laquelle se référer :
http://www.xm1math.net/algobox/doc.html

Scratch

Pour commencer à jouer un peu avec les concepts d'algorithmique, nous allons découvrir un outil éducatif en ligne développé par le MIT.

Démo simple :
https://scratch.mit.edu/



La représentation des algorithmes

OUTOFPLUTO Anthony Baillard Novembre 2017

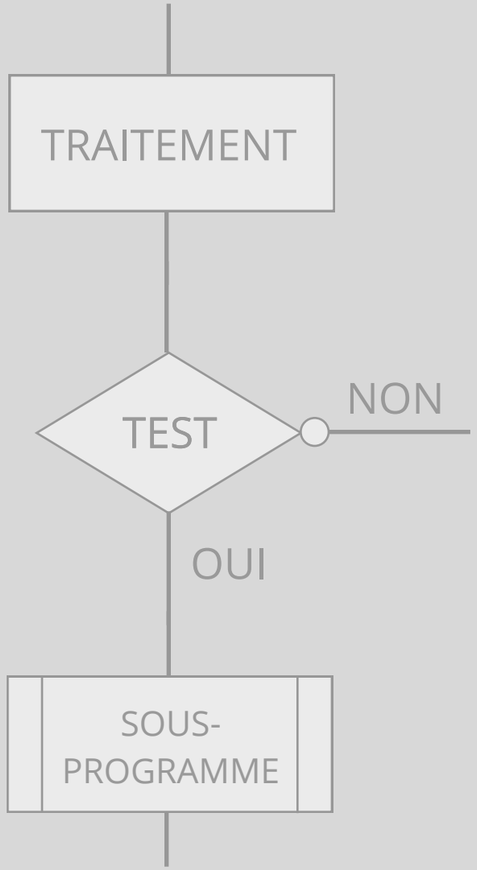
Les organigrammes

Les algorithmes sont souvent représentés sous forme d'*organigramme de programmation*.
Il existe un certain nombre de conventions concernant leur construction.

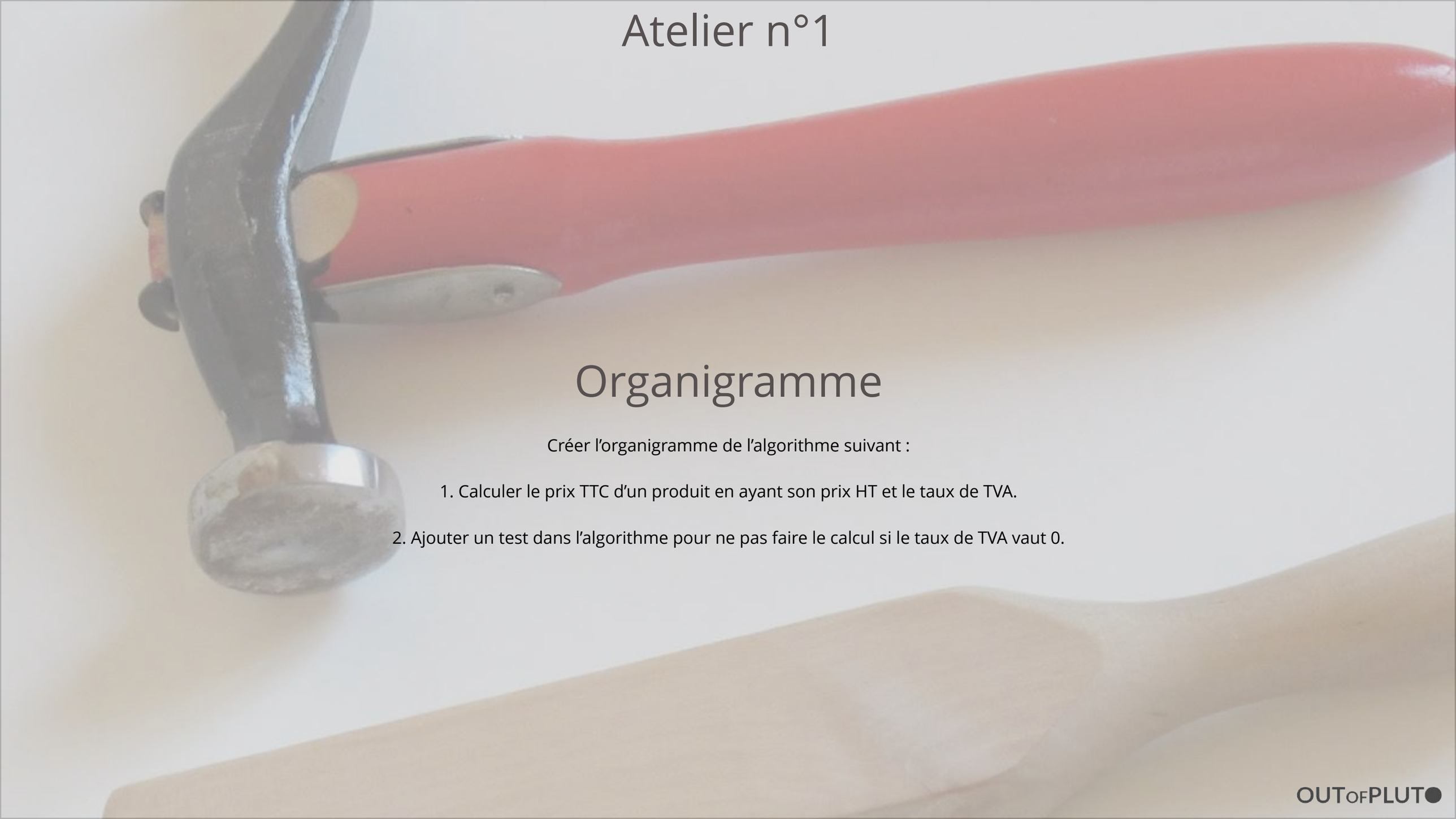
La plupart des opérations sont des *traitements*.
La boîte contient la description du traitement.
La boîte a une entrée et une sortie.

Certaines opérations sont des tests.
La boîte contient la description du test.
La boîte à une entrée et deux sorties.

Lorsqu'un algorithme devient trop complexe, on
utilise des boîtes de sous-programmes qui représentent
un algorithme entier.
La boîte à une entrée et une sortie.



Atelier n°1



Organigramme

Créer l’organigramme de l’algorithme suivant :

1. Calculer le prix TTC d’un produit en ayant son prix HT et le taux de TVA.

2. Ajouter un test dans l’algorithme pour ne pas faire le calcul si le taux de TVA vaut 0.

Les pseudo-langages

Pour que la compréhension d'un algorithme ne soit pas dépendante du langage de programmation,
il existe des *pseudo-langages.*

Certains sont en français.
C'est le cas d'Algobox, dont voici quelques exemples :

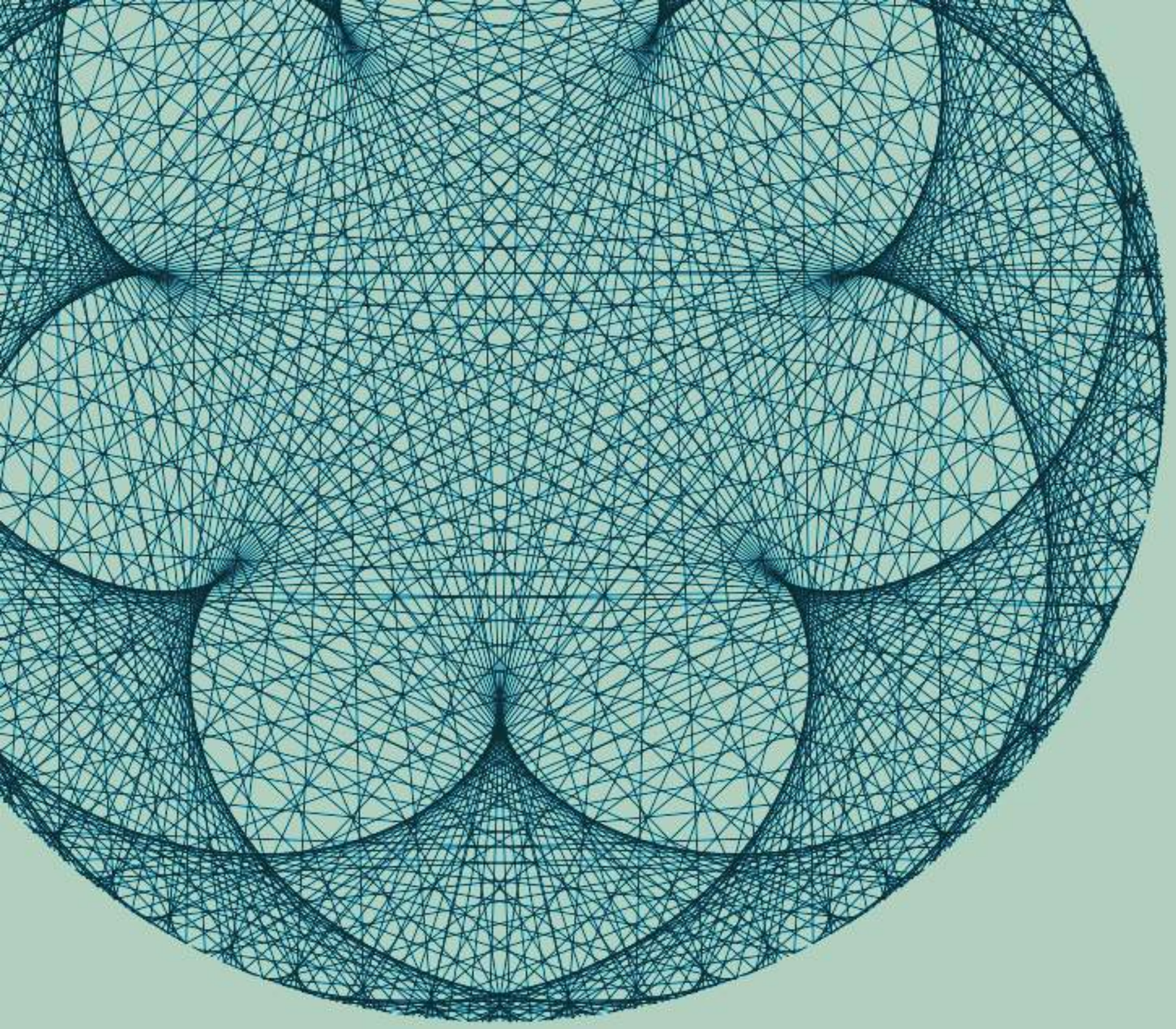
```
COMPTEUR EST_DU_TYPE NOMBRE

DEBUT_ALGORITHME

NOUVEAU PREND_LA_VALEUR 14

AFFICHER "texte"

TANT_QUE (TEST) FAIRE
    DEBUT_TANT_QUE
    FIN_TANT_QUE
```



Les structures de données

Les déclarations

Les données qui sont utilisées dans un algorithme doivent être déclarées avant leur usage.
Cela revient à définir les éléments du problème.

Une déclaration comporte :

- un nom, un mot ou une lettre, généralement explicite (*y* ou *temps* ou *diviseur)*
- un type, qui détermine comment la donnée est utilisable, i.e., quelles opérateurs peuvent lui être appliqués

Les principaux types sont :

- les nombres, sur lesquels il est possible d'appliquer tous les opérateurs arithmétiques
- les chaînes de caractères (*string*), sur lesquels il est possible d'appliquer des opérateurs spécifiques
- les tableaux (*array*) qui sont des regroupements de nombres

Exemple :

```
nombre          Pi  = 3,1415

chaine      prénom = ”Anthony”

tableau     base10 = 1 2 3 4 5 6 7 8 9 0
```

Les constantes et les variables

Les constantes et les variables sont des *données déclarées*.

Les constantes ont une valeur définie au départ de l'algorithme et ne changeront jamais.

Les variables ont une valeur définie (ou non) au départ de l'algorithme et qui peut changer au cours de l'exécution.

Remarques sur les variables :

Une variable qui n'a pas été affectée à une valeur inconnue.
Une variable peut rester inaffectée jusqu'à la fin d'un algorithme. Dans ce cas, sa valeur est inconnue.
Une variable qui n'est pas affectée durant un algorithme est superflue.
Il peut exister des variables intermédiaires.

Les tableaux ou les listes

Les tableaux sont des listes de nombres.

Les tableaux permettent de regrouper et d'organiser des nombres.

Pour accéder à un nombre dans un tableau, il faut connaître sa position.

Généralement, le premier élément est l'élément à la position 0.

S'il y a *n* éléments dans un tableau, le dernier élément se trouve à la position *n-1.*

Exemples :

base10 =

1	2	3	4	5	6	7	8	9	0
0	1	2	3	4	5	6	7	8	9

base10[4] vaut 5

Remarques :

Il n'y a pas de tableaux de chaînes de caractères en algorithmique théorique.
Fondamentalement, les chaînes de caractères SONT des tableaux (de caractères).

Atelier n°2

Variables et tableaux avec Algobox

Créer différents types de variables avec Algobox

OUTOFPLUTO

Les opérateurs sur les types simples

Nombres

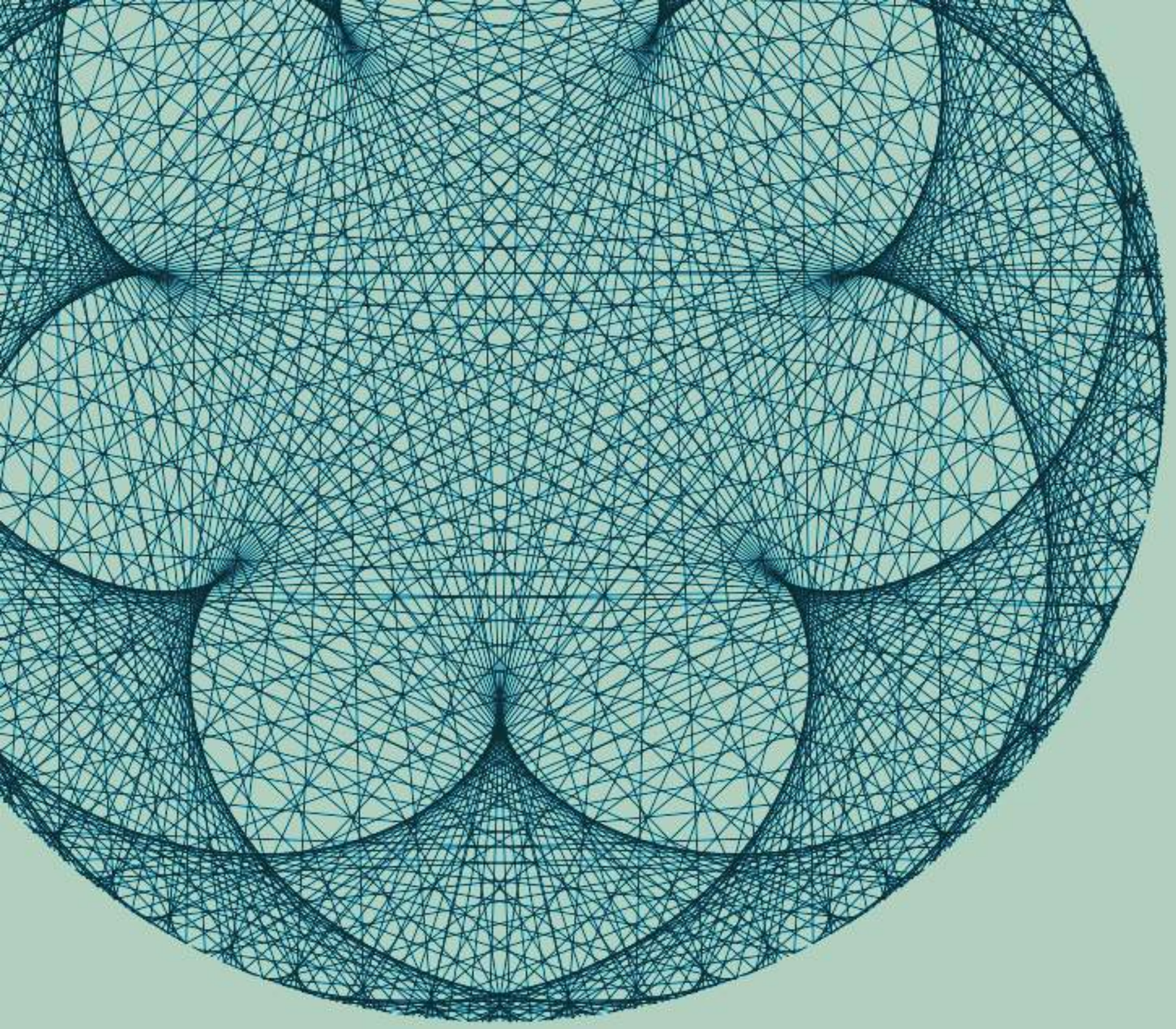
Lecture
Affectation
Opérateurs arithmétiques unaires : Signe, Valeur absolue, Inverse
Opérateurs de test unaires : Est positif, est nul, est négatif
Opérateurs arithmétiques binaires : Addition, Soustraction, Multiplication, Division, Modulo, Racine carrée...
Opérateurs de comparaison binaires : Plus grand que, plus petit que, égal

Chaînes de caractères

Lecture
Affectation
Opérateurs d'insertion et suppression
Opérateur de mesure : Longueur
(Opérateurs de comparaisons)

Tableaux

Lecture
Affectation
Opérateurs d'insertion et suppression
Opérateur de mesure : Longueur
(Opérateurs de comparaisons)



Les structures de contrôle

Les structures de contrôle

Les structures de contrôle permettent d'organiser l'algorithme, de définir la succession logique des opérations à effectuer.

On distingue trois catégories principales de structures de contrôle :

- Les séquences
- Les expressions conditionnelles
- Les boucles

La notion de *fonction* est également importante (même si elle n'est pas définie comme une structure de contrôle). Elle équivaut au sous-programme des organigrammes.

Les séquences

Les séquences constituent la structure de contrôle la plus simple : c’est une succession d’opérations consécutives.

C’est donc la structure la plus commune, par exemple :

Les expressions conditionnelles

Les expressions conditionnelles permettent d'exécuter des séquences uniquement lorsque certaines conditions sont remplis.

Typiquement :

```
Si   x > 0   alors   afficher “positif”   sinon   afficher “négatif”
```

Ou :

```
Si   x < 0   alors   afficher “-”
afficher valeur absolue de x
```

Les boucles

Les boucles permettent de répeter la même séquence plusieurs fois d'affilée.
Elles sont particulièrement utilisées avec les listes ou les tableaux pour réaliser une opération sur tous les éléments.

Il y a deux syntaxes principales les boucle *pour* (*for*) et les boucles *tant que* (*while* et *do while*)
Par exemple :

```
Pour   c   allant de   0   à   10   faire
    afficher “le carré vaut “
    afficher c*c
Fin Pour
```

```
Tant que   a est plus grand que b   faire
    a prend une valeur aleatoire
    b prend une valeur aleatoire
Fin Tant que
```

Les boucles pour

Les boucles se basent sur l'incrémentation (ou la décrémentation) d'une variable numérique : le compteur.

Les opérations à l'intérieur de la boucle peuvent lire la valeur du compteur.
Elles ne doivent pas modifier la valeur du compteur, au risque de modifier le comportement attendu de la boucle.

La définition des conditions de la boucle est importante pour ne pas créer une boucle sans fin !

Exemples de boucles non fonctionnelles :

```
Pour   c   allant de   20   à   10   par pas de   1   faire
    afficher c
Fin Pour
```

```
Pour   c   allant de   0   à   10   par pas de   2   faire
    c prend la valeur 0
Fin Pour
```

Les boucles tant que

Les boucles *tant que* se basent sur une expression conditionnelle pour leur arrêt.

Elles nécessitent généralement une initialisation préalable des variables servant dans l'expression conditionnelle.
Elles nécessitent généralement dans la séquence conditionnelle que la condition puisse devenir FAUSSE.

Ces deux points sont importants pour ne pas créer une boucle sans fin !

Exercice : Recréer la boucle

```
Pour   c   allant de   0   à   10   par pas de   2   faire
    afficher c
Fin Pour
```

Les boucles *faire tant que*

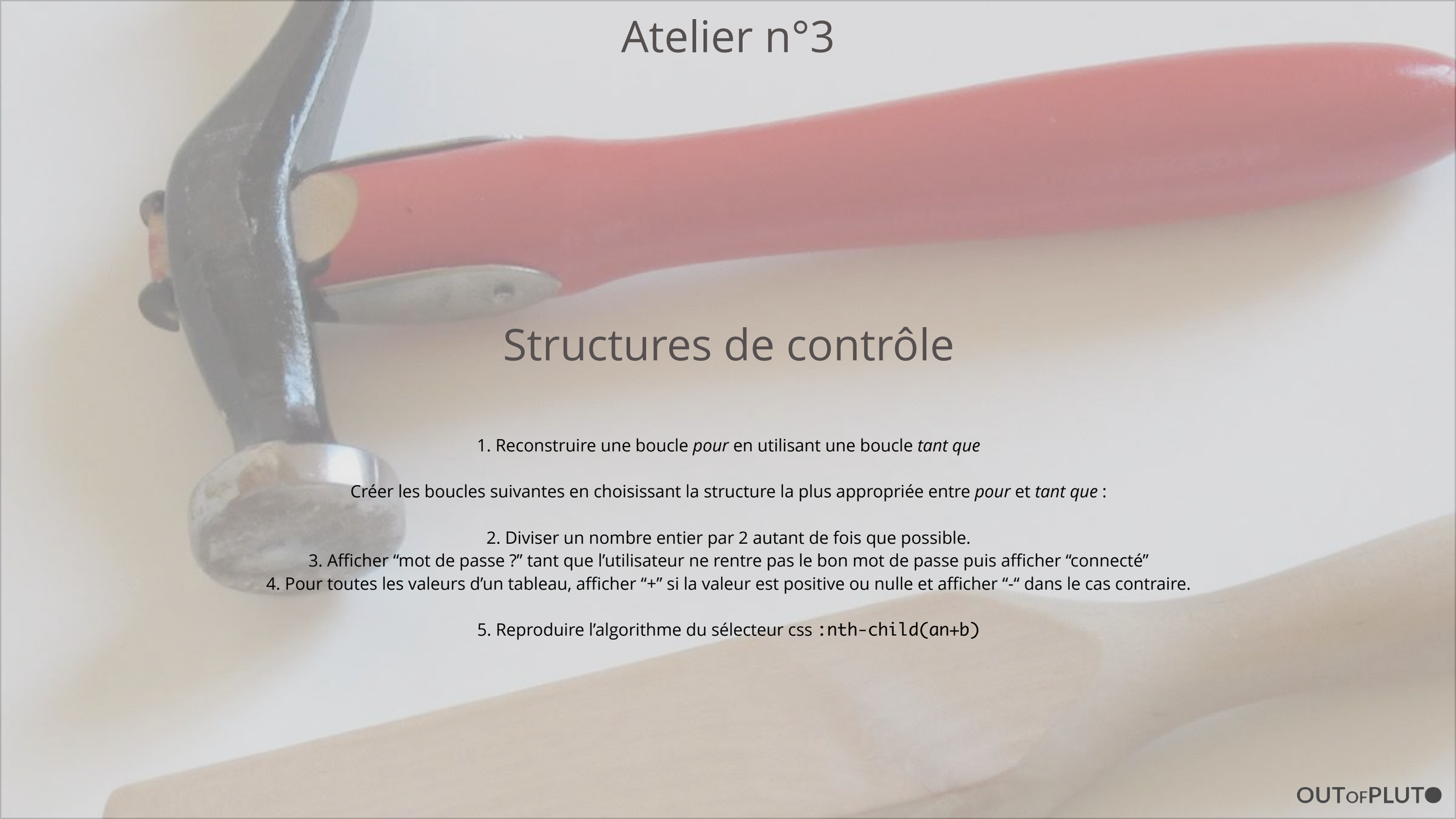
Les boucles *faire tant que* et *tant que* se basent sur la même logique.

Quelle est la différence entre ces deux boucles ?

```
Tant que   a est plus grand que b   faire
    a prend une valeur aleatoire
    b prend une valeur aleatoire
Fin Tant que
```

```
Faire
    a prend une valeur aleatoire
    b prend une valeur aleatoire
Tant que   a est plus grand que b
```

Atelier n°3



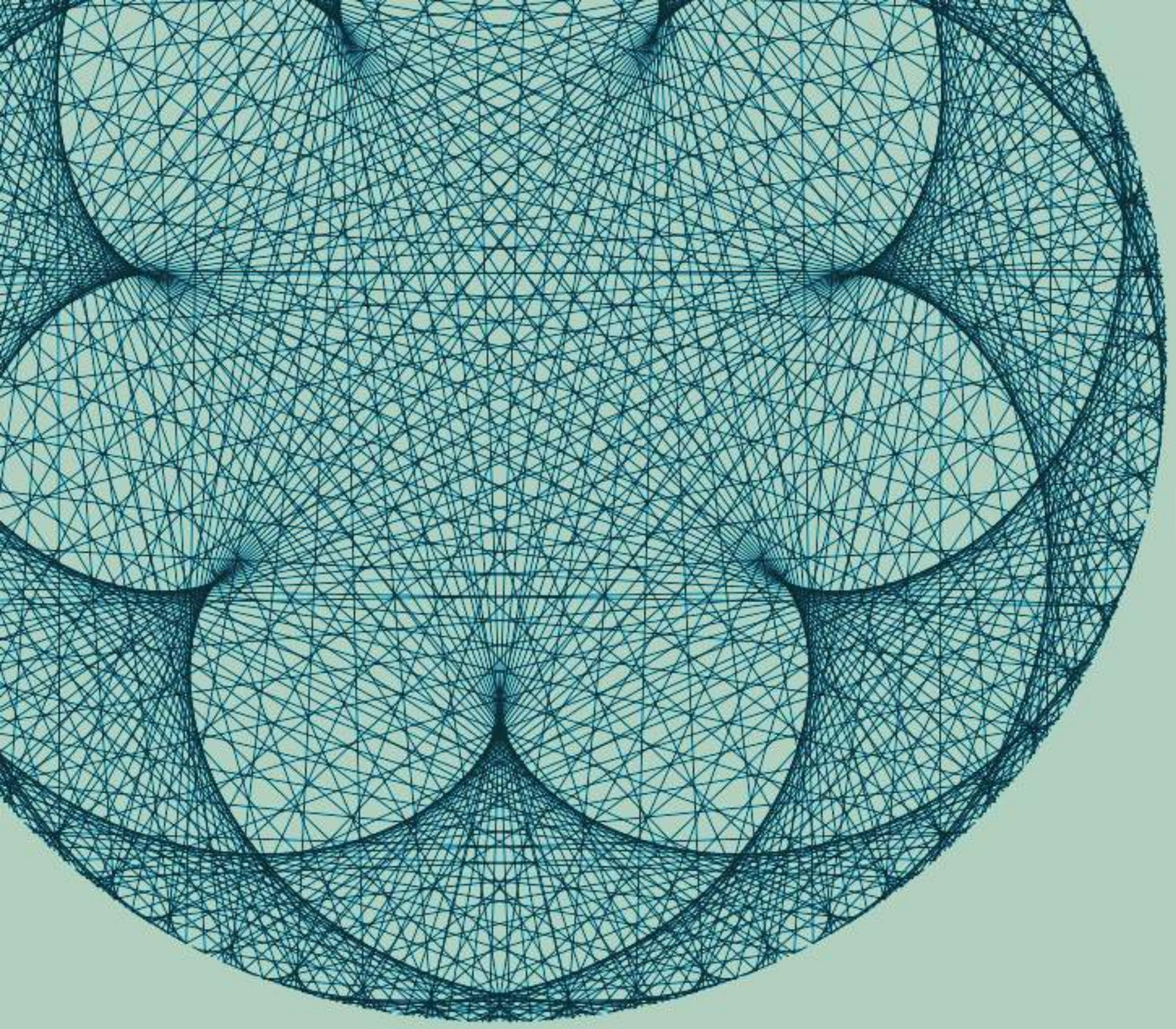
Structures de contrôle

1. Reconstruire une boucle *pour* en utilisant une boucle *tant que*

Créer les boucles suivantes en choisissant la structure la plus appropriée entre *pour* et *tant que* :

2. Diviser un nombre entier par 2 autant de fois que possible.
3. Afficher “mot de passe ?” tant que l’utilisateur ne rentre pas le bon mot de passe puis afficher “connecté”
4. Pour toutes les valeurs d’un tableau, afficher “+” si la valeur est positive ou nulle et afficher “-“ dans le cas contraire.

5. Reproduire l’algorithme du sélecteur css `:nth-child(an+b)`



Les représentations complexes

OUTOFPLUTO Anthony Baillard Novembre 2017

Les tableaux multidimensionnels

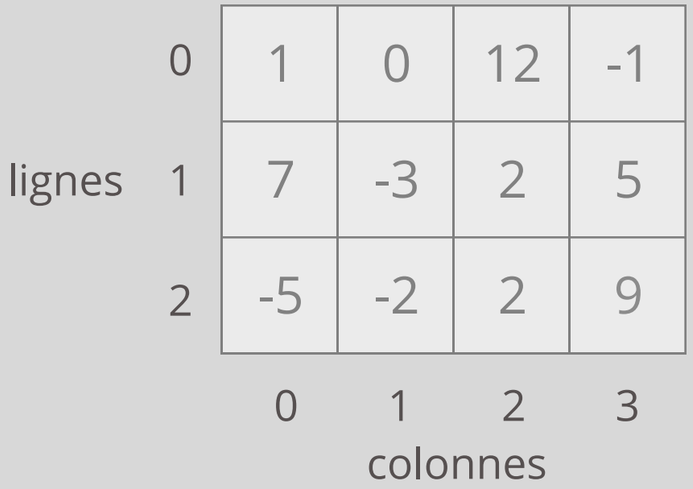
Introduction aux tableaux à plusieurs dimensions

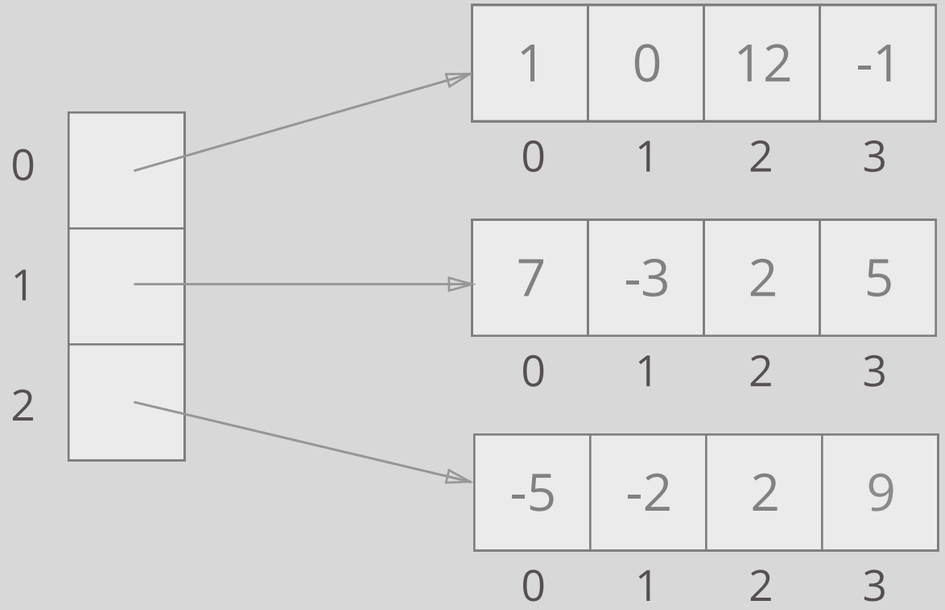
lignes	0	1	2	3
0	1	0	12	-1
1	7	-3	2	5
2	-5	-2	2	9

colonnes

Les tableaux multidimensionnels

Introduction aux tableaux à plusieurs dimensions



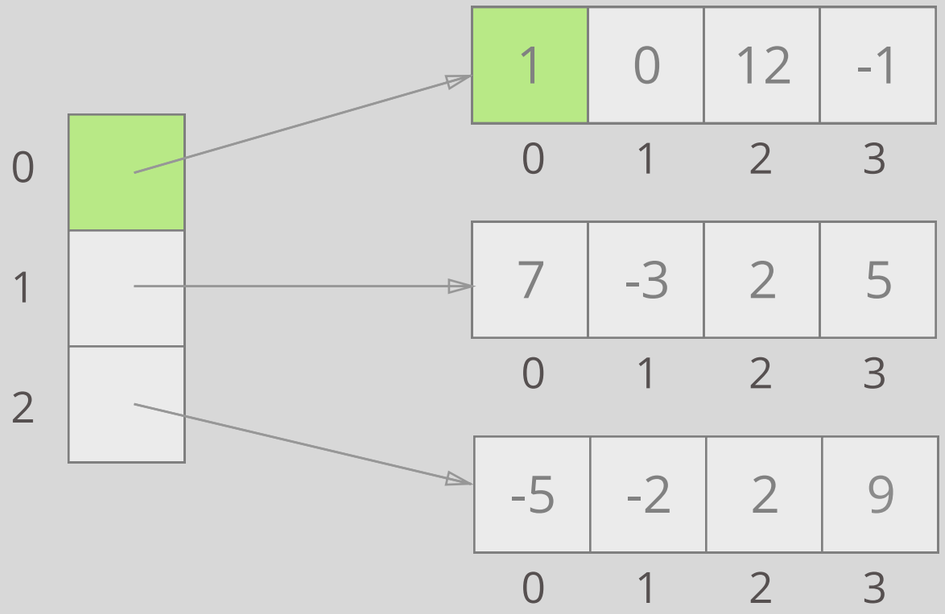


Les tableaux multidimensionnels

Introduction aux tableaux à plusieurs dimensions

lignes	0	1	2	3
0	1	0	12	-1
1	7	-3	2	5
2	-5	-2	2	9

colonnes



`tableau[0][0]` vaut 1

Combien vaut `tableau[1][3]` ?
Combien vaut `tableau[2][1]` ?
A quelle(s) position(s) peut-on trouver la valeur 2 ?

Les tableaux multidimensionnels

Représenter les tableaux multidimensionnels avec les tableaux à une dimension

lignes	0	1	2	3
0	1	0	12	-1
1	7	-3	2	5
2	-5	-2	2	9

colonnes

Définissons `nombre_colonnes = 4`

Alors

`tableau[1][0] = tableau[1*nombre_colonnes+0] = tableau[4]`

Quel est l'index de `tableau[1][3]` ?

Quelle est la position de `tableau[9]` ?

Quelle est la règle générique pour accéder à un élément ?

Les tableaux multidimensionnels

Représenter les tableaux multidimensionnels avec les tableaux à une dimension

Règles générique :

Définissons `nombre_colonnes = 4`

Pour trouver l'index d'un élément en connaissant sa position :
`index = row*nombre_colonnes+col`

Pour trouver la valeur d'un élément selon sa position dans le tableau :
`tableau[row][col] = tableau[row*nombre_colonnes+col]`

Pour trouver la position d'un élément en connaissant son index :
`col = index % nombre_colonnes`
`row = (index - col) / nombre_colonnes`

lignes				
0	1	0	12	-1
1	7	-3	2	5
2	-5	-2	2	9
	0	1	2	3

colonnes

Atelier n°3

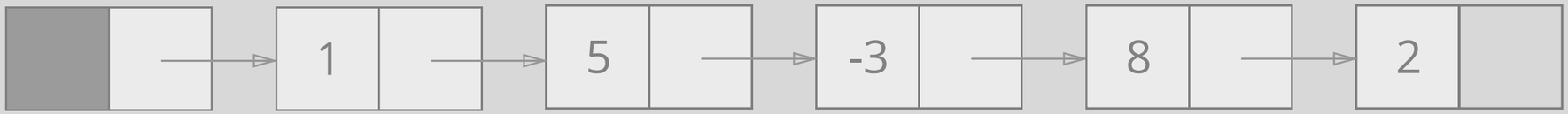
Tableau à deux dimensions avec Algobox

Créer un tableau à deux dimensions avec Algobox
Affecter et afficher les valeurs du tableau
(en utilisant les méthodes vues précédemment)

OUTOFPLUTO

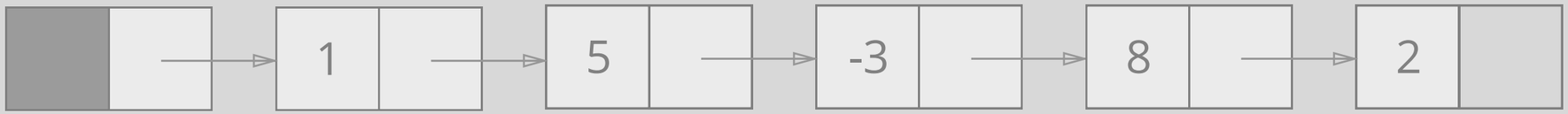
Les listes chaînées

Chaque élément contient une valeur et un *pointeur* vers l'élément suivant.
Le premier élément n'est qu'un pointeur pour trouver le début de la liste.
Le dernier élément n'est qu'une valeur, son pointeur est *nul.*

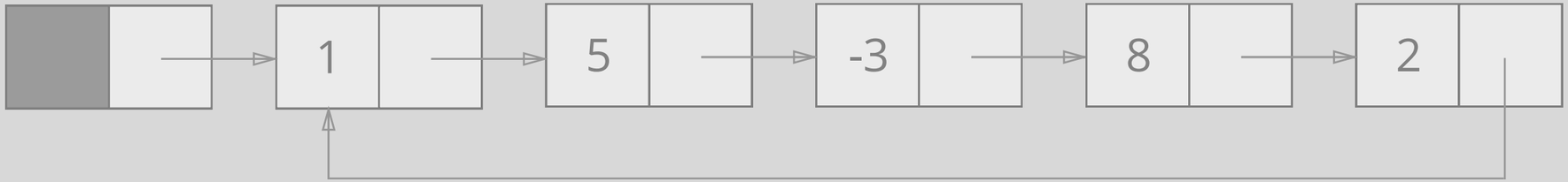


Les listes chaînées

Chaque élément contient une valeur et un *pointeur* vers l'élément suivant.
Le premier élément n'est qu'un pointeur pour trouver le début de la liste.
Le dernier élément n'est qu'une valeur, son pointeur est *nul.*



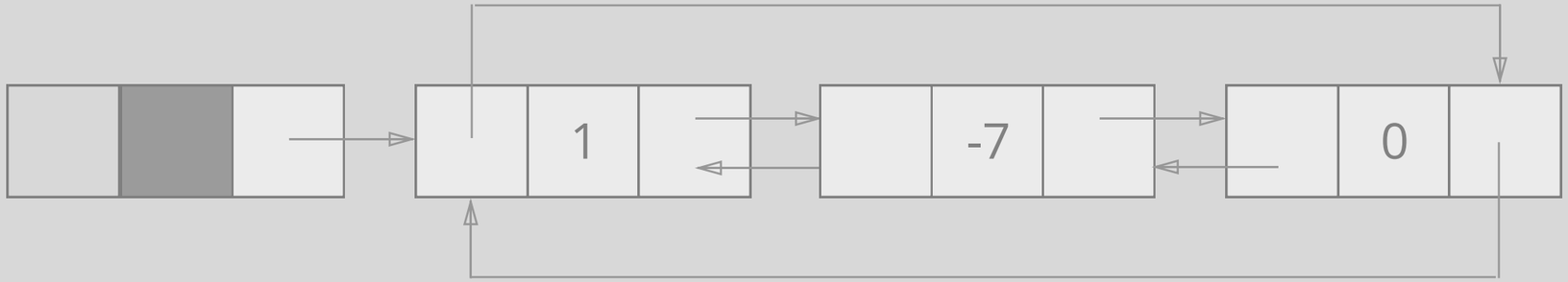
Les listes chaînes peuvent être circulaires.
Il n'y a dans ce cas pas de dernier élément et il faut un *point d'entrée*.



Les listes chaînées

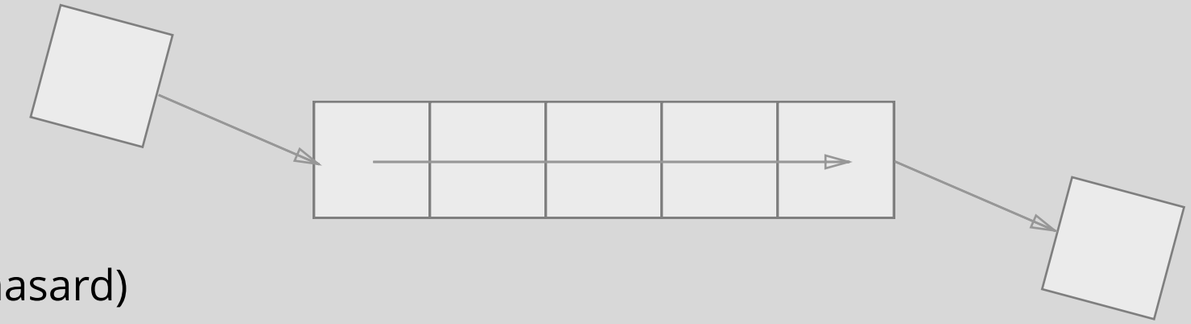
Les listes peuvent être doublement chaînées.
Dans ce cas, il y a deux premiers éléments et deux derniers éléments, un pour chaque sens.

Les listes chaînes peuvent être doublement chaînées circulaires.
Dans ce cas, il n’y a jamais de dernier élément et il faut un *point d’entrée*.



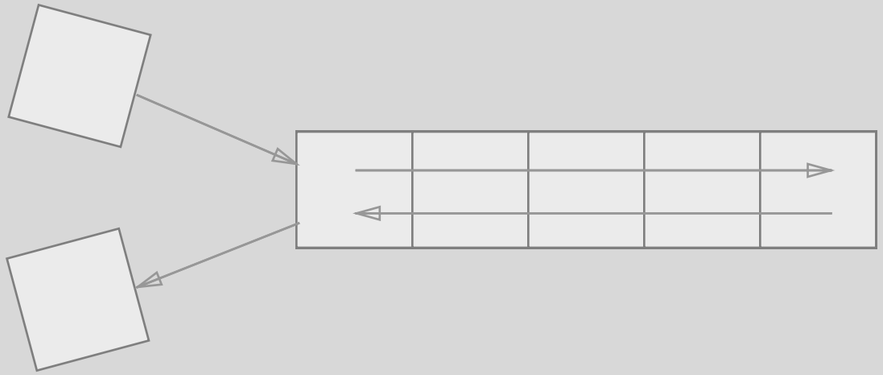
Les piles et les files

Files = FIFO (First In First Out)



Exemple, file d’attente à La Poste (au hasard)

Piles = FILO (First In Last Out)

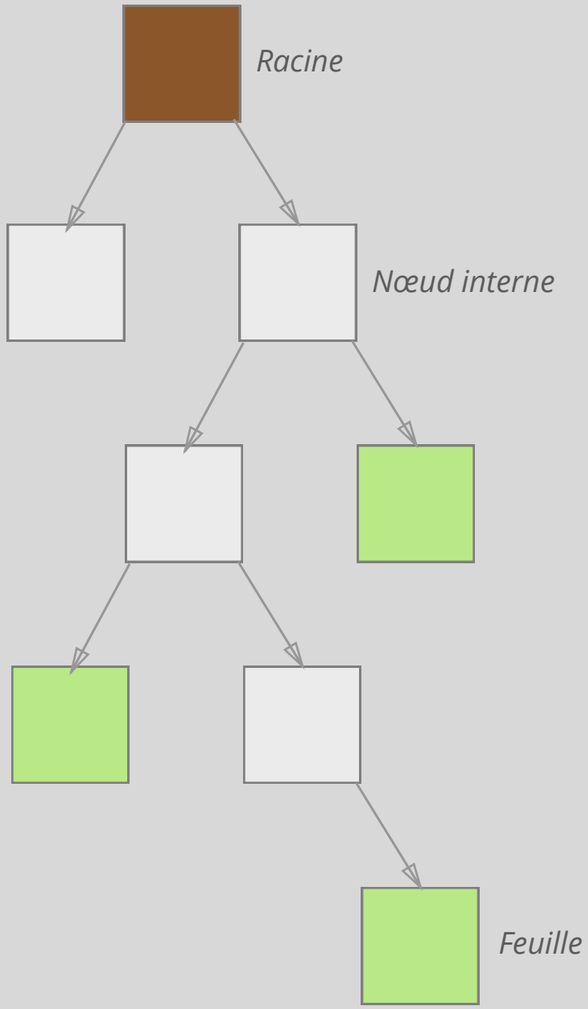


Exemple, une pile d’assiettes

Les arbres

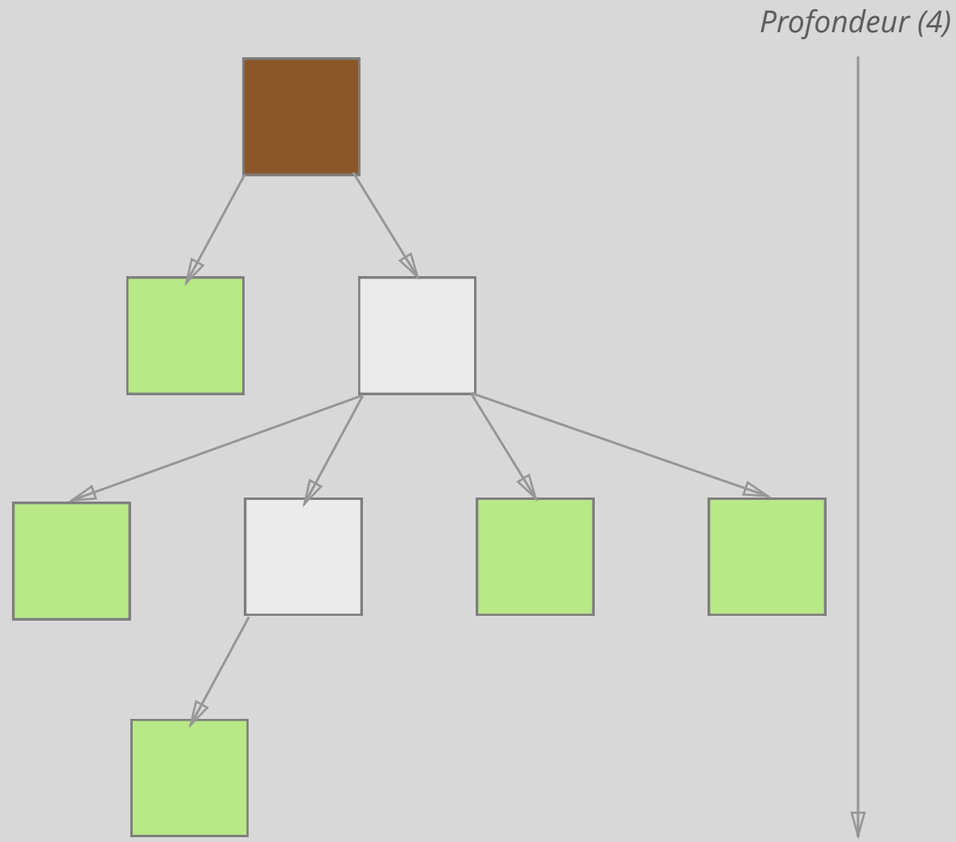
Arbres binaires

Exemples : tournois



Arbres n-naires

Exemple : généalogie



Atelier n°4

Arbres

Créer l’arbre du tableau final de Roland Garros femme 1927

Quart de finale	Demi-finale	Finale
Cornelia Bouman Billie Tapscott	Bobbie Heine **Cornelia Bouman**	**Cornelia Bouman** Irene Bowder
Eileen Bennett Marguerite Broquedis	**Irene Bowder** Eileen Bennett	
Cilly Aussem **Irene Bowder**		
Lili Alvarez **Bobbie Heine**		

Les graphes non orientés

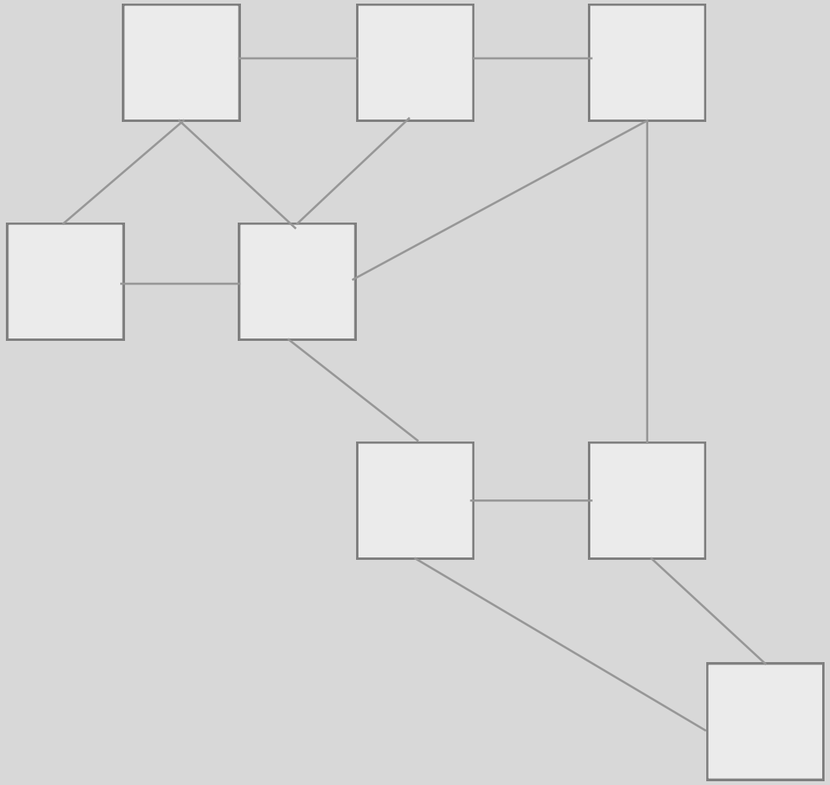
Les graphes sont des structures permettant de relier des éléments dans un réseau.

Comme pour les arbres, chaque élément est appelé un nœud.

Une liaison entre deux nœuds est appelé un arc.

Les arcs peuvent être pondérés ou non.

Exemples : carte des liaisons aériennes.
La pondération peut être le temps de vol ou le prix du billet.



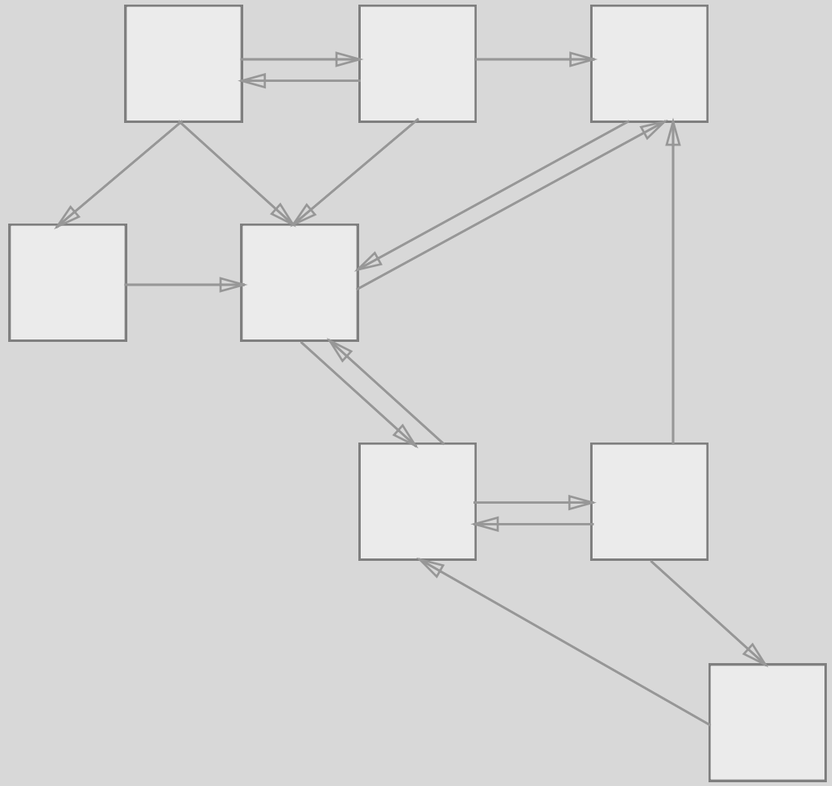
Les graphes orientés

Contrairement aux graphes non orientés, les arcs ne peuvent être parcourus que dans un sens.

Evidemment, il peut y avoir deux arcs réciproques.

Les arcs peuvent être pondérés ou non.

Exemple de graphe orienté ?



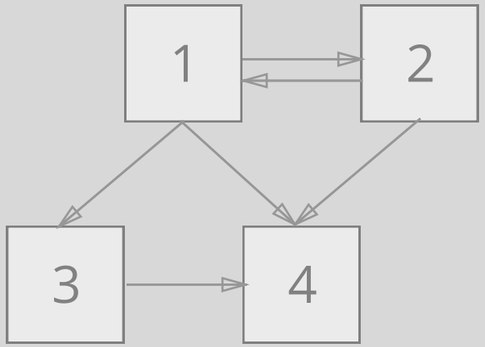
La représentation des graphes

En algèbre linéaire (et donc avec des listes),
on représente les graphes sous forme de *matrices d'adjacence*.

Pour savoir s'il existe un arc du nœud *source* vers le nœud *dest,* on vérifie la valeur de

```
matrice[source][dest]
```

Par exemple, il existe un arc de 1 vers 4, donc `matrice[1][4] = 1`
Autre exemple, il n'existe pas d'arc de 2 vers 3, donc `matrice[2][3] = 0`



	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	0	0	0	1
4	0	0	0	0

Les opérateurs sur les types complexes

Listes chaînées

Lecture
Affectation
Opérateurs d'insertion et suppression
Opérateur de mesure : Longueur
(Opérateurs de comparaisons)
Itérateur

Piles et files

Push
Pop

Arbres

Lecture
Affectation
Opérateurs d'insertion et suppression
Opérateur de mesure : Profondeur
(Opérateurs de comparaisons)
Itérateur

Graphes

Lecture
Affectation
Opérateurs d'insertion et suppression
Opérateur de mesure : Taille
(Opérateurs de comparaisons)
Itérateur

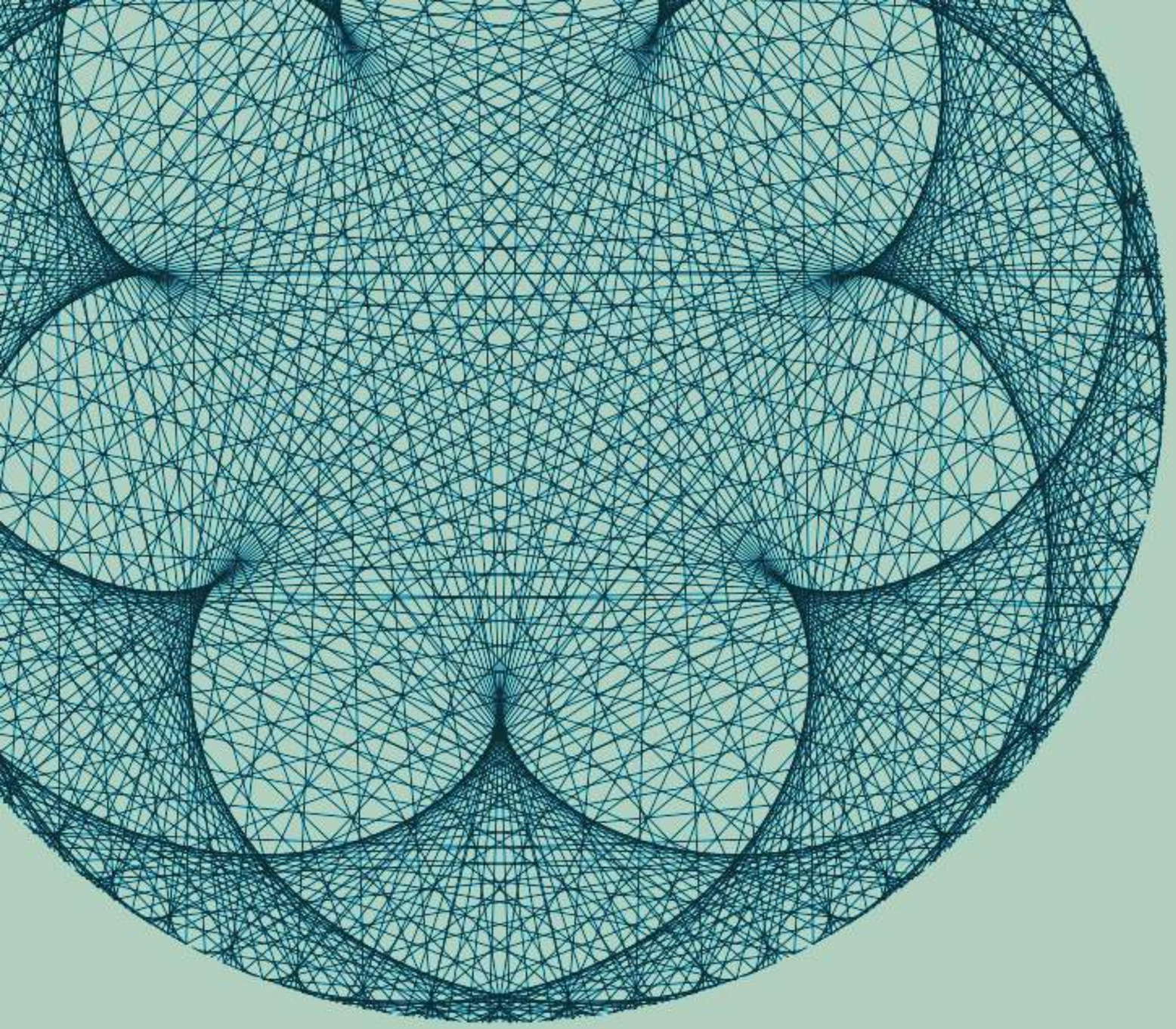
La récursivité

La récursivité est une méthode d'exploration/parcours d'une structure récursive.

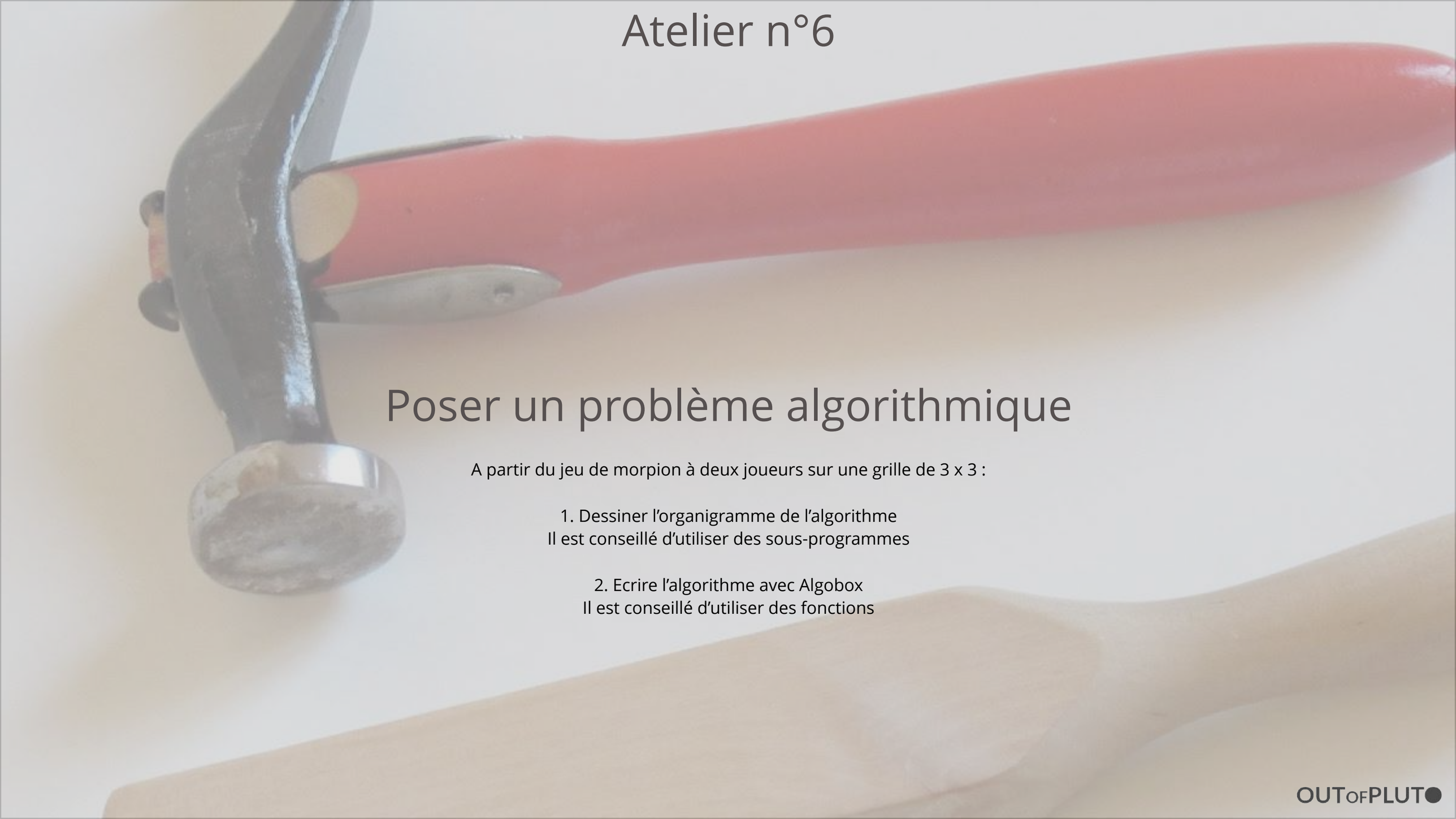
Les listes chaînées sont des structures récursives : un élément contient un *pointeur* vers un autre élément.
Les éléments de la liste doivent être du même type.

Un opérateur qui permet de parcourir récursivement une structure s'appelle un *itérateur* ou un *visiteur.*

La récursivité est équivalente à une boucle.



Pour aller plus loin



Atelier n°6

Poser un problème algorithmique

A partir du jeu de morpion à deux joueurs sur une grille de 3 x 3 :

1. Dessiner l'organigramme de l'algorithme
Il est conseillé d'utiliser des sous-programmes

2. Ecrire l'algorithme avec Algobox
Il est conseillé d'utiliser des fonctions

L’implémentation

- La finitude des moyens

Lors de l’implémentation d’un algorithmique sur un ordinateur, les ressources sont limités : mémoire finie et temps d’éxecution non nul.

- Les langages

L’implémentation dépend grandement du langage : C, python, javascript, PHP. Certains sont plus adaptés que d’autres pour la gestion des données ou pour l’execution de code mathématique. Un bon choix de départ peut être python.

- La syntaxe

Chaque langage a une syntaxe qui lui est propre pour les structures de données, de contrôle, pour les déclarations de variables. Cependant, cela n’a théoriquement que peu d’influence sur la “traduction” de l’algorithme.

- Les structures de données

Selon les langages, les structures de données disponibles peuvent être plus riches que les structures Nombre, Chaîne et Liste.
Typiquement, les nombres sont divisés entre entiers (*integer*) et réels (*float*). Il existe aussi des tableaux associatifs où les indices peuvent être des chaînes de caractères

- Les structures de contrôle

Selon les langages, les structures de contrôle peuvent également être plus riches. Les boucles pour peuvent prendre plusieurs conditions et il existe des syntaxes pour éviter les si/sinon imbriqués (syntaxe *switch case*)

- Les langages non typés

Certains langages (python par exemple), autorise la déclaration de variables sans type. Il faut alors être très rigoureux avec le code.

Complexité et heuristique

Lectures et références

Froidevaux, Gaudel, Soria, Algorithmique et types de données
Knuth, The Art of Computer Programming