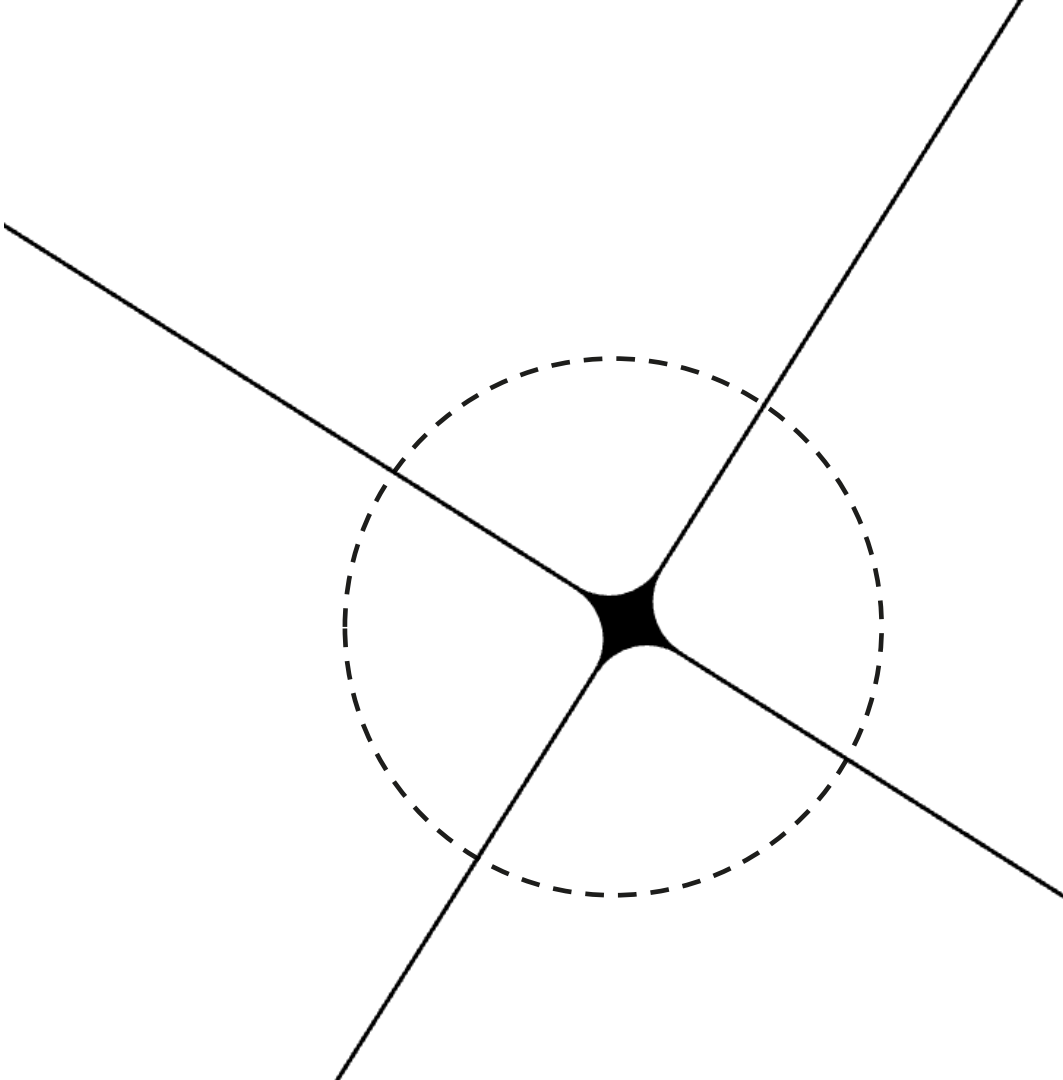
INSTITUT POLYTECHNIQUE DE PARIS

TELECOM SudParis

Introduction TP 5

Julien Romero



Interpréteur et arbre de syntaxe abstrait

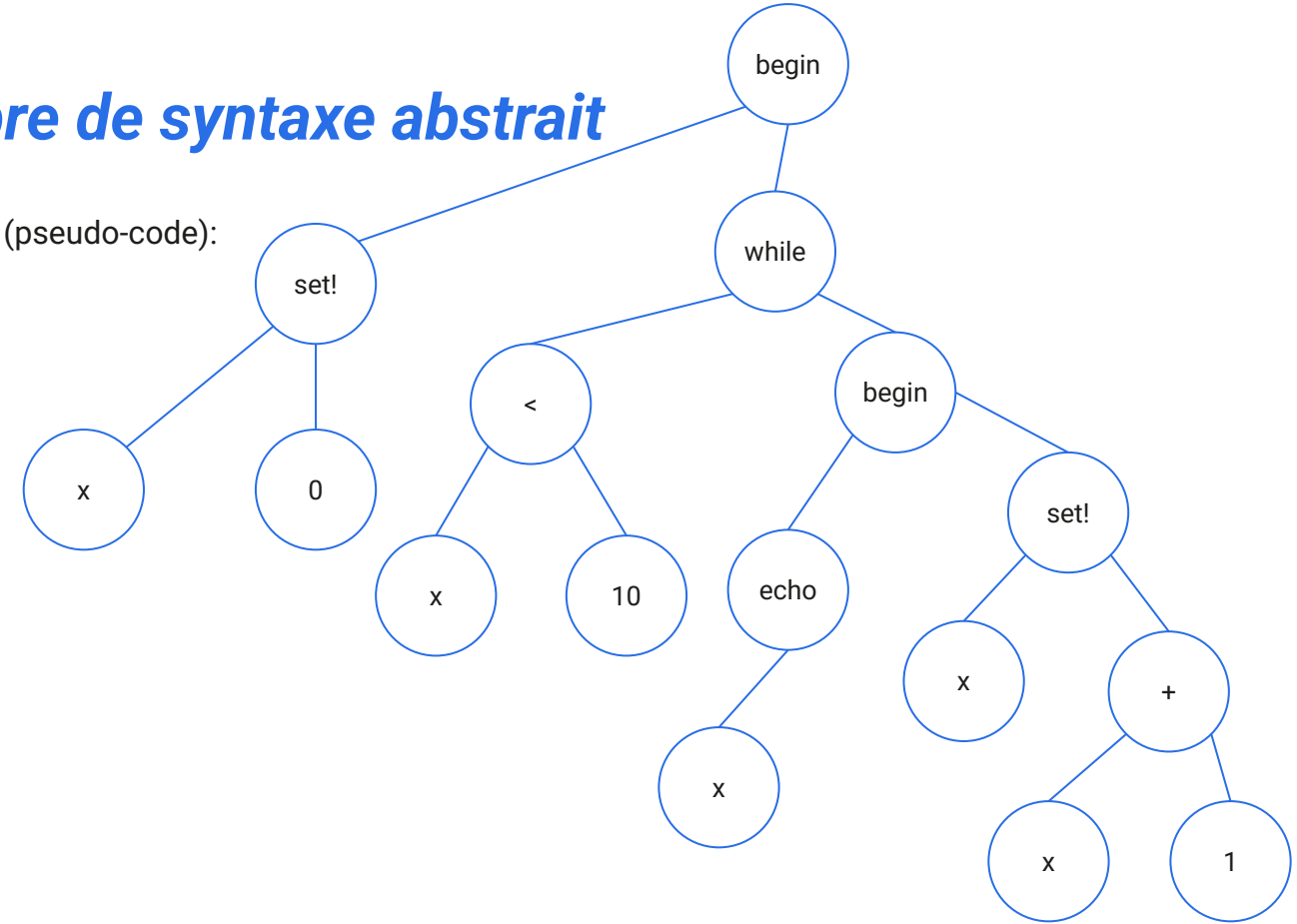
- Dans ce cours, nous allons implémenter un **interpréteur** à partir d'un **arbre de syntaxe abstrait**.
- Un **interpréteur** est un programme capable d'exécuter un autre programme. Il commence généralement par une phase transformant le code source en un arbre de syntaxe abstrait, mais cette étape n'est pas vue dans le TP.
- Un **arbre de syntaxe abstrait** est un arbre dont les nœuds représentent différentes fonctionnalités du langage (variable, constante, addition, condition, boucle, …) et les fils d'un nœuds les "paramètres" d'une fonctionnalités.
    - Par exemple, une condition **if** a besoin de trois paramètres: la condition, ce qu'il faut faire si la condition est remplie, et ce qu'il faut faire dans le cas contraire (else).
    - En lisant cet arbre de la bonne manière, on peut exécuter le programme qu'il représente.

Exemple d'arbre de syntaxe abstrait

Pour le programme suivant (pseudo-code):

```
x = 0
while x < 10:
    print(x)
    x = x + 1
```

On peut avoir l'arbre:

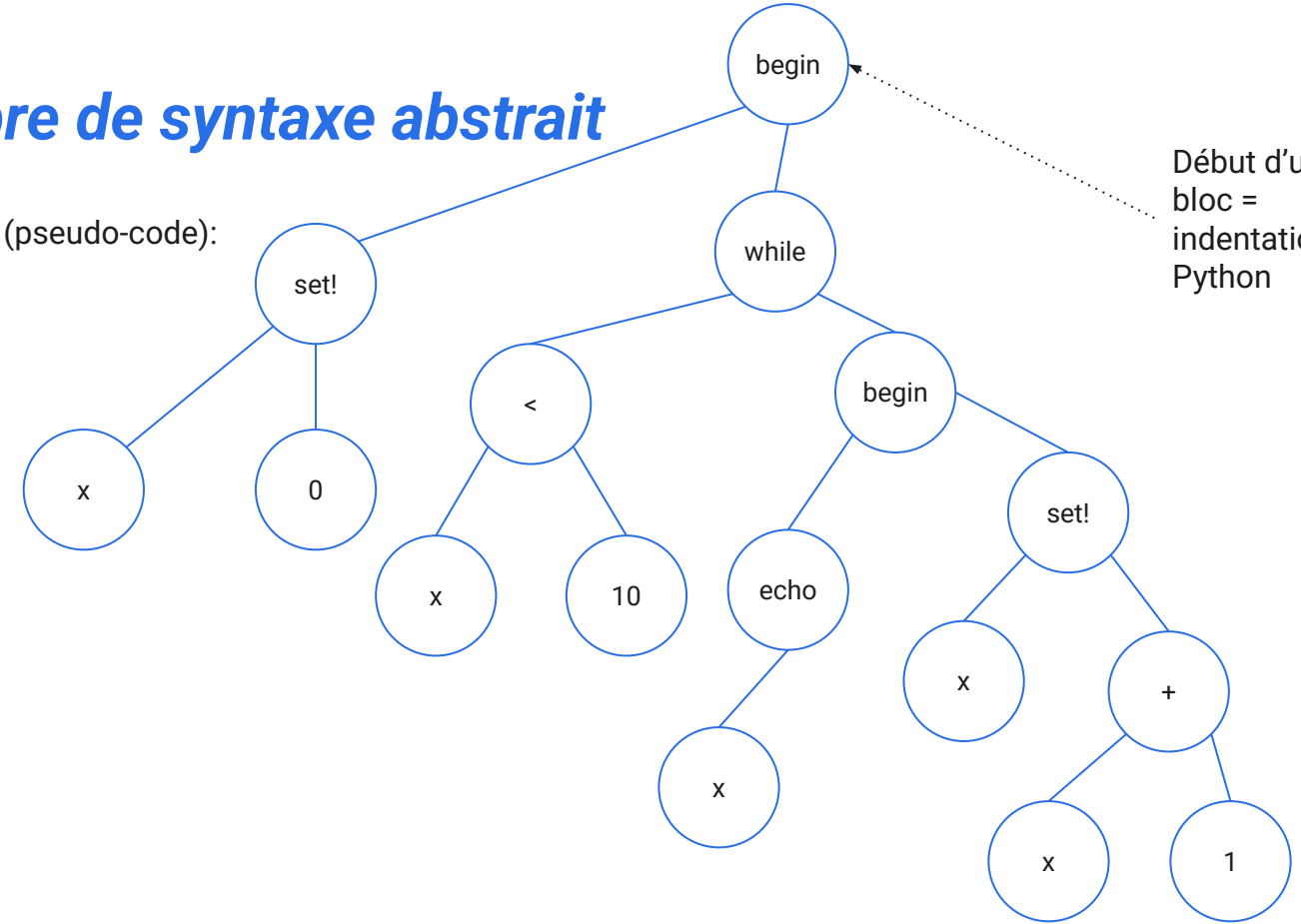


Exemple d'arbre de syntaxe abstrait

Pour le programme suivant (pseudo-code):

```
x = 0
while x < 10:
    print(x)
    x = x + 1
```

On peut avoir l'arbre:



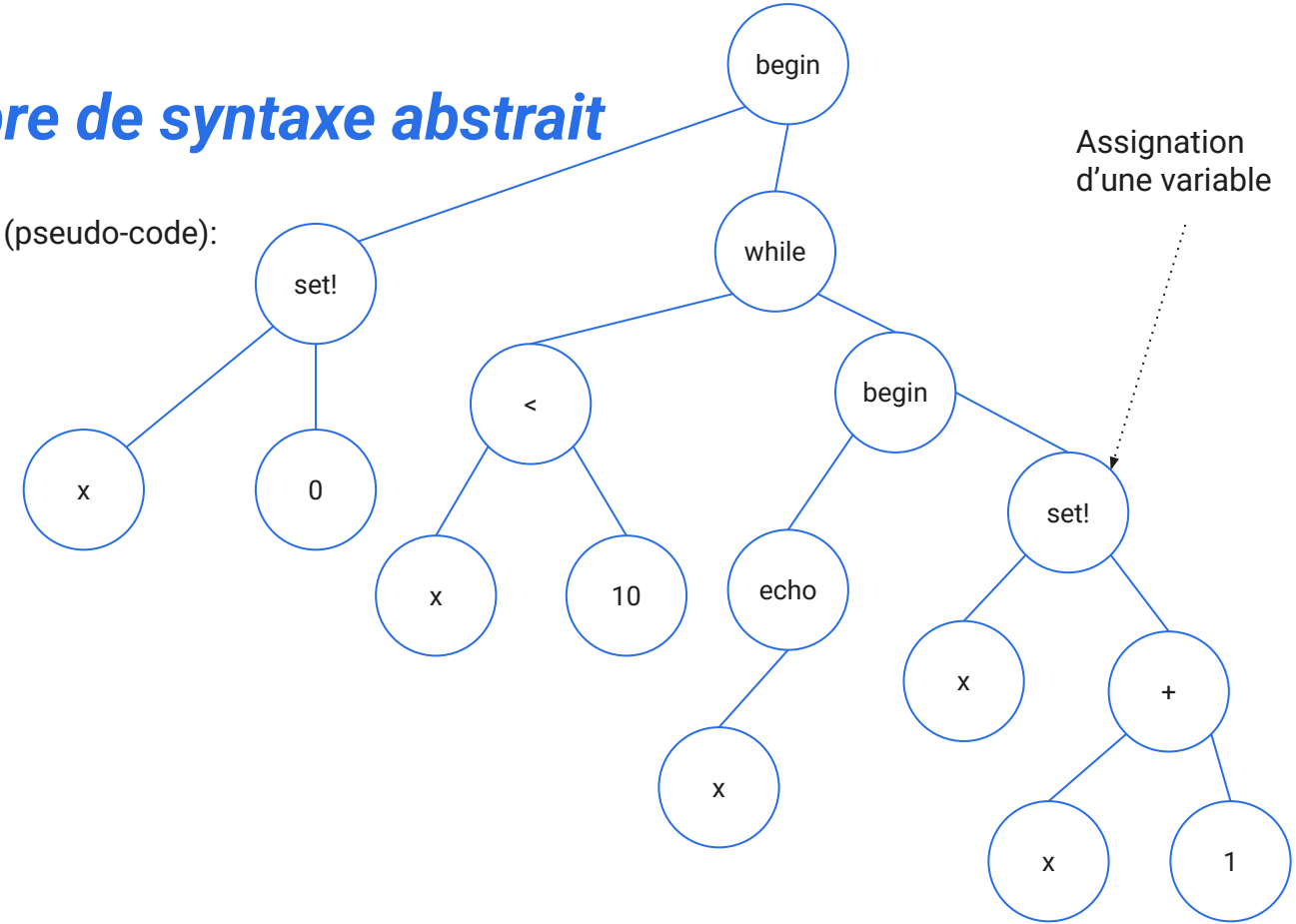
Début d'un bloc = indentation en Python

Exemple d'arbre de syntaxe abstrait

Pour le programme suivant (pseudo-code):

```
x = 0
while x < 10:
    print(x)
    x = x + 1
```

On peut avoir l'arbre:

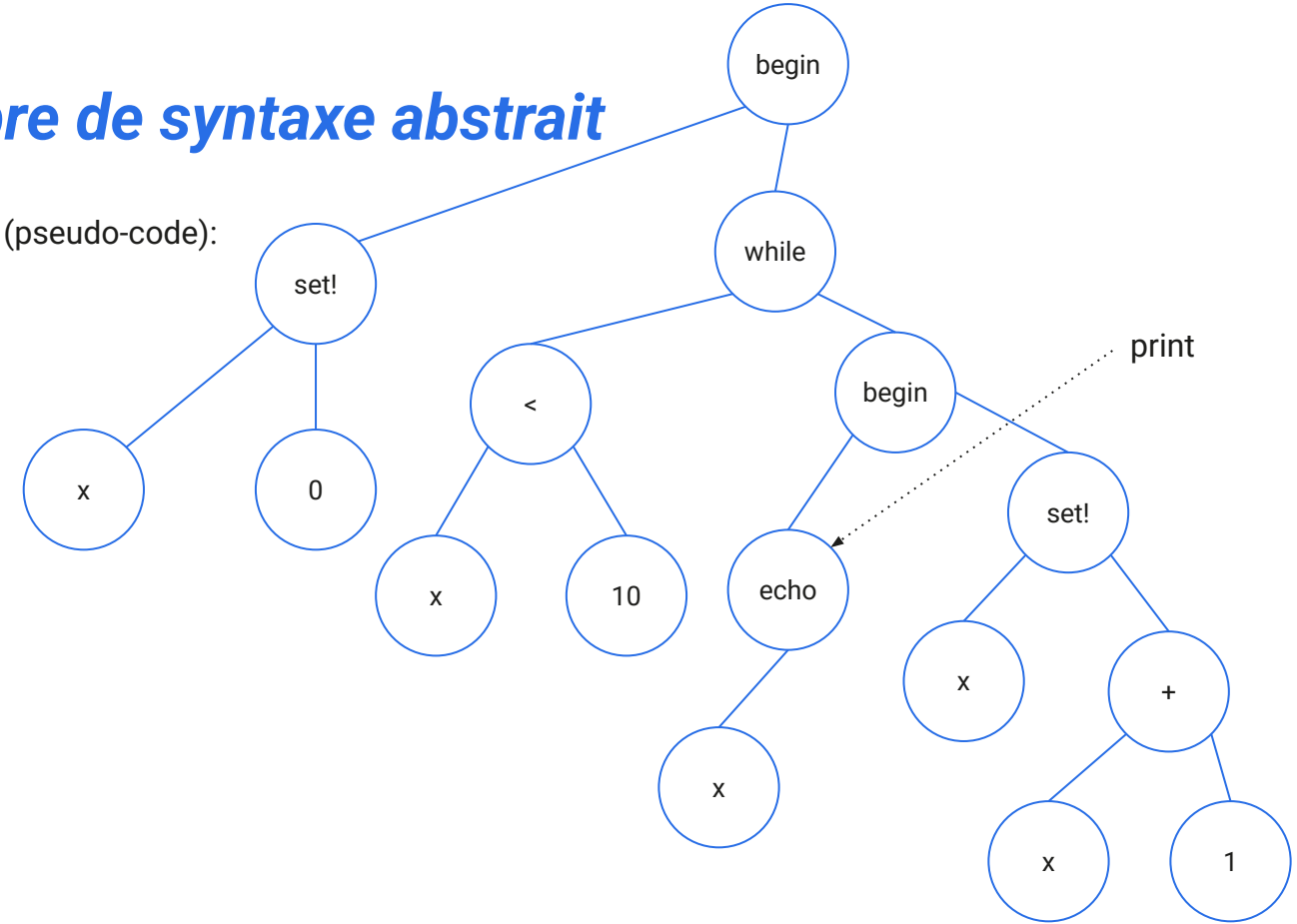


Exemple d'arbre de syntaxe abstrait

Pour le programme suivant (pseudo-code):

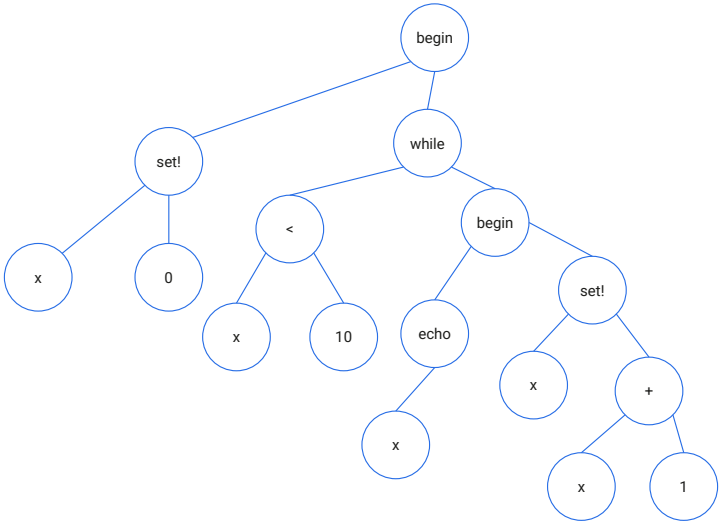
```
x = 0
while x < 10:
    print(x)
    x = x + 1
```

On peut avoir l'arbre:



Forme préfixée et parenthésée

- Nous allons représenter l'arbre de syntaxe abstrait sous la forme préfixée et parenthésée. Chaque nœud/fonctionnalité `op` se notera de la manière suivante: $(op\ f_1\ f_2\ f_3\ \dots\ f_n)$ où $f_1, f_2, f_3, \dots, f_n$ sont les représentations sous forme préfixée et parenthésée des fils.
  - S'il n'y a pas de fils, on écrira simplement `op` (par exemple, pour une constante ou la valeur d'une variable)
  - Exemple: l'arbre précédent s'écrit (begin (set! x 0) (while (< x 10) (begin (echo x) (set! x (+ x 1)))))



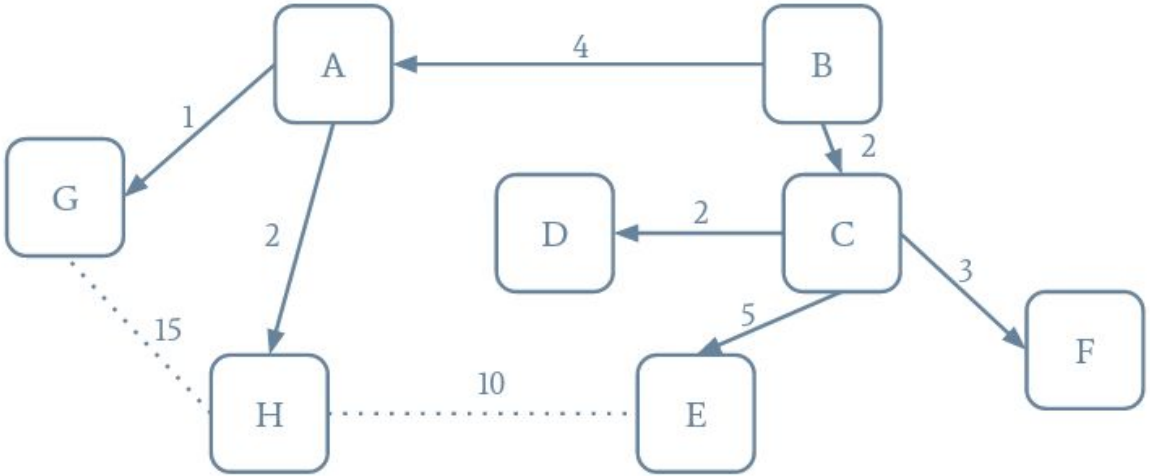
Implémentation avec l'héritage

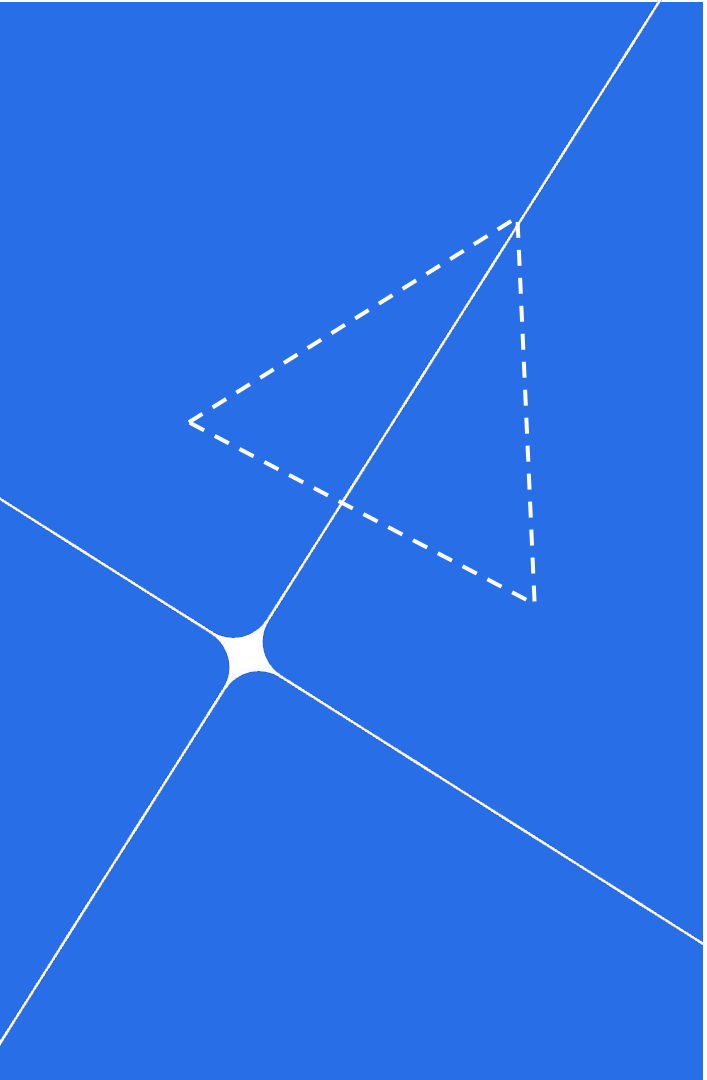
- Chaque nœud a une fonction `int execute()` qui changera en fonction du nœud
  - Un scalaire retournera simplement sa valeur
  - Une addition de deux nœuds retournera la somme des résultats des executes: left.execute() + right.execute().
  - Une condition `if` executera la condition et, en fonction du résultat, exécutera le `if` ou le `else`.
- La méthode `public String toString()` sera aussi à réimplémenter pour automatiquement générer la forme préfixée parenthésée (similaire à `int execute()`)

Et ensuite ?

Pour les plus rapides, implémentation de l'algorithme de Kruskal permettant de trouver l'arbre couvrant minimal.

Un **arbre couvrant** d'un graphe est un arbre dont les nœuds sont les mêmes que le graphe et dont les arêtes sont incluses dans celles du graphe. Comme c'est un arbre, il n'utilisera pas toutes les arêtes. De plus, il est dit **minimal** si c'est l'arbre couvrant tel que la **somme des poids des arêtes soit minimale**.





En route pour le TP