1. Introduction

- **Objectif de l'algorithmique :** obtenir d'une machine (ordinateur, calculatrice, robot,...) qu'elle fasse certaines tâches à notre place.
- **Problème :** expliquer à la machine de façon claire et rigoureuse comment elle doit faire. Pour cela, on doit lui décrire l'ensemble des opérations à effectuer pour obtenir le résultat souhaité : c'est ce qu'on appelle concevoir un **algorithme**.
  Le logiciel **AlgoBox** permet à travers un "mini-langage" et des commandes prêtes à l'emploi de créer des algorithmes qui détailleront de façon précise à la machine (l'ordinateur dans le cas présent) la liste précise des instructions à suivre. Une fois l'algorithme mis au point, il pourra être testé afin de vérifier que l'on obtient bien le résultat désiré.

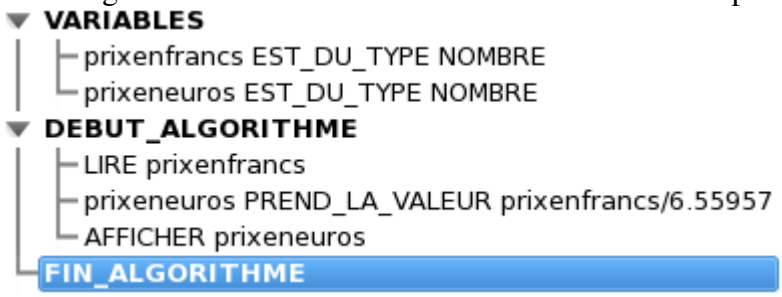
Exemple 1 : on voudrait que l'ordinateur convertisse en € un prix en frs donné par l'utilisateur.
On peut expliquer à l'ordinateur, avec un algorithme, ce qu'il doit faire pour y parvenir :

1. Tout d'abord, il va falloir stocker quelque part le prix en francs (entré par l'utilisateur) et le résultat (le prix en euros). Pour cela on utilise ce qu'on appelle en informatique des **variables**. Ici, nos deux variables stockeront des nombres que nous appelleront :
   `prixenfrancs` et `prixeneuros`.
   La première chose à faire dans notre algorithme va donc être d'expliquer à l'ordinateur qu'il va falloir utiliser deux variables (contenant des nombres) appelées :
   `prixenfrancs` et `prixeneuros`.
   Voilà comment faire avec **AlgoBox** :
   - Cliquer sur le bouton `Déclarer nouvelle variable`.
   - Dans le champ `Nom de la variable`, entrer `prixenfrancs`, vérifier que le `Type de la variable` est bien sur `NOMBRE` et cliquer sur `OK`.
   - Cliquer de nouveau sur le bouton `Déclarer nouvelle variable`.
   - Dans le champ `Nom de la variable`, entrer `prixeneuros`, vérifier que le `Type de la variable` est bien sur `NOMBRE` et cliquer sur `OK`.
2. Il faut maintenant permettre à l'utilisateur d'indiquer le prix en francs qu'il veut convertir et stocker ce nombre dans la variable `prixenfrancs`.
   Voilà comment faire avec **AlgoBox** :
   - Avec la souris, se placer (si ce n'est pas déjà le cas) sur la ligne `DEBUT_ALGORITHME`, puis cliquer sur le bouton `Nouvelle Ligne`.
   - Cliquer alors sur le bouton `Ajouter LIRE variable`.
   - Dans le champ `LIRE la variable`, vérifier que `prixenfrancs` est bien sélectionné et cliquer sur `OK`.
3. Procédons maintenant au calcul du prix en euros (1 euro représente 6.55957 francs).
   Voilà comment faire avec **AlgoBox** :
   - Avec la souris, se placer (si ce n'est pas déjà le cas) sur la ligne `LIRE prixenfrancs`, puis cliquer sur le bouton `Nouvelle Ligne`.
   - Cliquer alors sur le bouton `AFFECTER valeur à variable`.
   - Sélectionner `prixeneuros` juste après `La variable`, taper `prixenfrancs/6.55957` dans le champ `prend la valeur` et cliquer sur `OK` (Attention à bien taper 6.55957 et pas 6,55957 : en informatique, le séparateur décimal est le point et pas la virgule).
4. Il ne reste plus qu'à afficher le résultat.

Voilà comment faire avec **AlgoBox** :

- Avec la souris, se placer (si ce n'est pas déjà le cas) sur la ligne `prixeneuros PREND_LA_VALEUR prixenfrancs/6.55957`, puis cliquer sur le bouton `Nouvelle Ligne`.
- Cliquer alors sur le bouton `Ajouter AFFICHER variable`.
- Sélectionner `prixeneuros` juste après `AFFICHER la variable` et cliquer sur `OK`.

5. Notre algorithme est maintenant terminé et devrait correspondre à ça :



Il n'y a plus qu'à le tester :

- Cliquer sur le bouton `Tester Algorithme`.
- Dans la fenêtre qui s'affiche, cliquer alors sur le bouton `Lancer algorithme`.
- Entrer alors un prix à convertir (100 par exemple) et cliquer sur `OK`. Le prix en euros apparaît alors dans le cadre `Résultat` .
- Pour convertir un autre montant, il suffit de cliquer à nouveau sur le bouton `Lancer algorithme`.
- Une fois les tests terminés, on revient à la fenêtre principale du programme en cliquant sur le bouton `fermer` en bas de la fenêtre. On peut alors sauvegarder son algorithme en cliquant sur le bouton `Sauver`.

Premières règles concernant la conception d'un algorithme avec AlgoBox :

- Il faut toujours commencer par déterminer les variables nécessaires à la bonne marche de l'algorithme et indiquer leurs noms à **AlgoBox** en utilisant le bouton `Déclarer nouvelle variable`. Une variable ne pourra être utilisée par **AlgoBox** que si elle est déjà déclarée.
- Pour que l'utilisateur puisse entrer des données, il faut utiliser le bouton `Ajouter LIRE variable` qui permettra à l'utilisateur de donner une valeur à la variable sélectionnée.
- Pour pouvoir donner une valeur à une variable à l'intérieur de l'algorithme, il faut utiliser le bouton `AFFECTER valeur à variable`. La boîte de dialogue qui apparaît permet de sélectionner la variable à laquelle on veut affecter une valeur et la valeur à lui affecter (éventuellement un calcul)
- Pour pouvoir afficher un résultat correspondant à la valeur d'une variable, il faut utiliser le bouton `Ajouter AFFICHER variable` et sélectionner la variable en question .
- Pour ajouter un nouvel élément à l'algorithme (autre que la déclaration d'une variable), il faut d'abord insérer une nouvelle ligne en se positionnant à l'endroit adéquat et en cliquant sur le bouton `Nouvelle Ligne`.

Remarque : pour repartir d'un algorithme vide dans **AlgoBox**, il suffit de cliquer sur le bouton `Nouveau` en haut du programme (il est alors demandé à l'utilisateur s'il veut d'abord enregistrer l'algorithme courant)

Activité 1 :
Concevoir un algorithme avec **AlgoBox** qui calcule l'aire d'un rectangle après que l'utilisateur ait entré la largeur et la longueur du rectangle. Pour cela :

- Cliquer d'abord sur le bouton `Nouveau` dans **AlgoBox**.
- Utiliser trois variables qui porteront comme nom : `largeur`, `longueur` et `aire`.

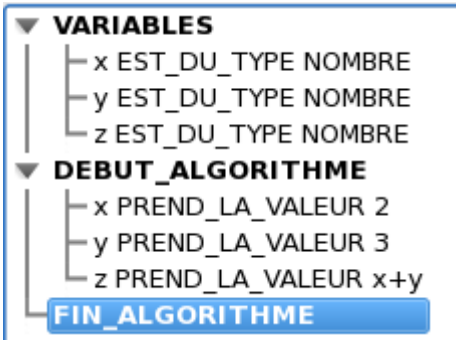
2. Variables et affectations

Remarques générales à propos des variables qui, comme nous l'avons déjà vu, servent à stocker des données :

- Un nom de variable ne contient que des chiffres et des lettres et débute par une lettre (pas d'accent)
- La valeur d'une variable peut varier au fil des instructions de l'algorithme.
- Les variables informatiques peuvent servir à stocker des données de différents types, mais au début nous nous contenterons d'utiliser des variables du type `NOMBRE` (type par défaut dans **AlgoBox**)

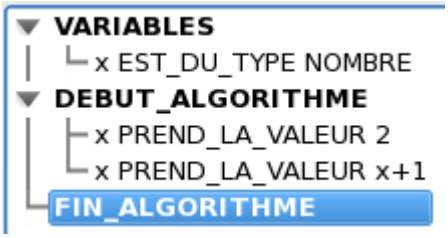
Il est aussi très important en informatique de comprendre ce qui se passe quand on affecte une valeur à une ou plusieurs variables. Pour cela, regardons quelques exemples :

Exemple 2.1 : On considère le code suivant



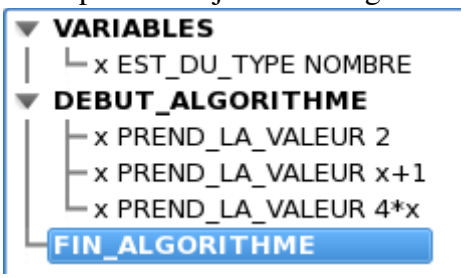
Après exécution de l'algorithme, la variable `z` contient la valeur 5 (l'ordinateur ajoute la valeur de `x` avec celle de `y`, puis stocke le résultat dans `z`). Quant aux variables `x` et `y`, elles contiennent toujours à la fin de l'algorithme les valeurs 2 et 3.

Exemple 2.2 : Considérons maintenant le code suivant



Après exécution de l'algorithme, la variable `x` contient la valeur 3. En effet quand la ligne `x PREND_LA_VALEUR x+1` est exécutée, l'ordinateur prend la valeur précédente de `x` (c'est à dire 2), ajoute 1 et stocke le tout dans `x` (à la place de la valeur précédente).

Exemple 2.3 : Ajoutons la ligne `x PREND_LA_VALEUR 4*x` à la fin du code précédent. Ce qui donne :



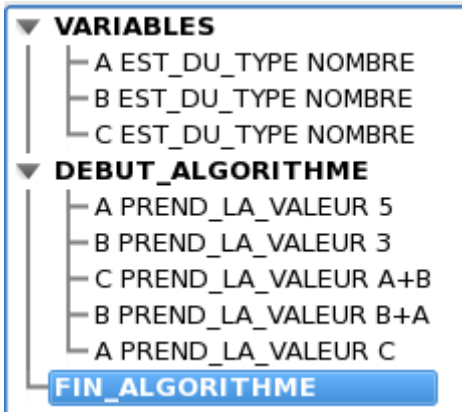
Après exécution de l'algorithme, la variable `x` contient cette fois-ci la valeur 12. En effet :

- Après exécution de la ligne `x PREND_LA_VALEUR 2`, la variable `x` contient évidemment la valeur 2.
- Après exécution de la ligne `x PREND_LA_VALEUR x+1`, la variable `x` contient, comme nous venons de le voir, la valeur 3.
- Après exécution de la ligne `x PREND_LA_VALEUR 4*x`, la variable `x` contient finalement la valeur 12 (4*3).

Conclusion :

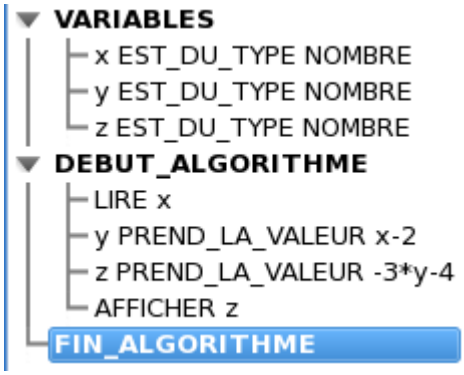
- Les opérations sur les variables s'effectuent ligne après ligne et les unes après les autres.
- Quand l'ordinateur exécute une ligne du type " *mavariable* `PREND_LA_VALEUR` *un calcul* ", il effectue d'abord *le calcul* et stocke **ensuite** le résultat dans *mavariable*.

Activité 2.1 :
On considère l'algorithme ci-dessous :

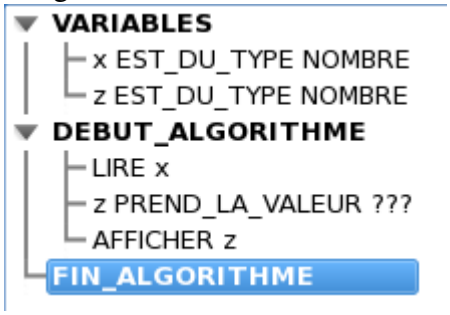


Déterminer quelles sont les valeurs des variables `A`,`B` et `C` à la fin de l'exécution de l'algorithme.

Activité 2.2 :
Concevoir avec **AlgoBox** un algorithme qui donne exactement les mêmes résultats (quelque soit la valeur de x) mais sans utiliser la variable `y` que l'algorithme ci-dessous.



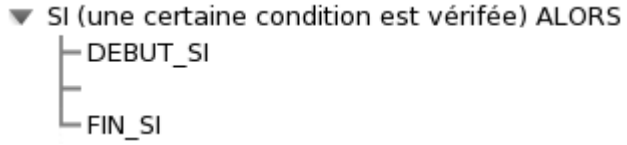
L'algorithme à concevoir devra être de la forme suivante (en remplaçant ??? par la bonne expression) :



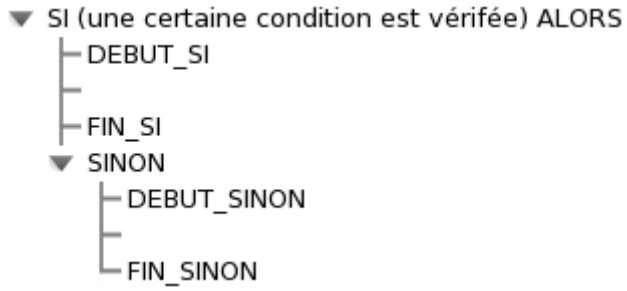
3. Instructions conditionnelles

Nous avons vu qu'un algorithme permet d'exécuter une liste d'instructions les unes à la suite des autres.
Mais **on peut aussi n'exécuter des instructions que si une certaine condition est remplie**.

- Cela se fait grâce à la commande `SI...ALORS` que l'on peut insérer dans `AlgoBox` à l'aide du bouton `Ajouter SI...ALORS`.
  On obtient alors la structure suivante :



- Il est aussi possible d'indiquer en plus à l'algorithme de traiter le cas où la condition n'est pas vérifiée.
  (cela se fait en cochant l'option `Ajouter SINON` *dans la boîte de dialogue correspondante à cette commande)*
  On obtient alors la structure suivante :



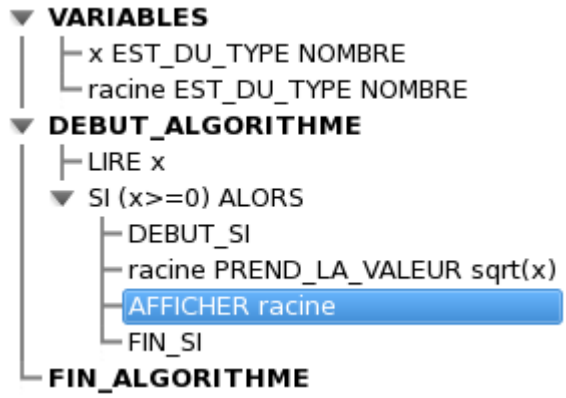
Regardons sur deux exemples, comment cela se passe en pratique.

Exemple 3.1 : Comment créer un algorithme qui demande un nombre à l'utilisateur et qui affiche la racine carrée de ce nombre s'il est positif.

1. Il va nous falloir deux variables contenant des nombres : une pour le nombre entré par l'utilisateur que nous allons appeler `x` et une autre pour le résultat que nous allons appeler `racine`.
   Après avoir lancé **AlgoBox** (ou cliqué sur le bouton `Nouveau` si le programme est déjà lancé) :
   - On clique sur le bouton `Déclarer nouvelle variable` .
   - Dans le champ `Nom de la variable`, on entre `x`, on vérifie que le `Type de la variable` est bien sur `NOMBRE` et on clique sur `OK` .
   - On clique de nouveau sur le bouton `Déclarer nouvelle variable` .
   - Dans le champ `Nom de la variable`, on entre `racine`, on vérifie que le `Type de la variable` est bien sur `NOMBRE` et on clique sur `OK` .
2. Il faut maintenant permettre à l'utilisateur d'entrer le nombre dont on veut la racine :
   - Avec la souris, on se place (si ce n'est pas déjà le cas) sur la ligne `DEBUT_ALGORITHME`, puis on clique sur le bouton `Nouvelle Ligne`.
   - On clique alors sur le bouton `Ajouter LIRE variable`.
   - Dans le champ `LIRE la variable`, on vérifie que `x` est bien sélectionné et on clique sur `OK`.
3. Il faut maintenant ajouter la commande `SI...ALORS` :
   - Avec la souris, on se place (si ce n'est pas déjà le cas) sur la ligne `LIRE x`, puis on clique sur le bouton `Nouvelle Ligne`.
   - On clique alors sur le bouton `Ajouter SI...ALORS`.
   - Dans le champ après `Si la condition :`, on entre `x>=0` (c'est notre condition pour pouvoir calculer la racine) et on clique sur `OK` (la case `Ajouter SINON` n'étant pas cochée).
4. Détaillant maintenant les instructions que l'algorithme doit suivre quand notre condition est vérifiée :
   - En étant bien positionné sur la ligne vide (entre la ligne `DEBUT_SI` et la ligne `FIN_SI`), on clique alors sur le bouton `AFFECTER valeur à variable`.
   - Pour `La variable`, on sélectionne `racine`, puis dans le champ après `prend la valeur`, on entre `sqrt(x)` et on clique sur le bouton `OK`.

- En étant bien positionné sur la ligne `racine PREND_LA_VALEUR sqrt(x)`, on clique sur le bouton `Nouvelle Ligne`.
- Il n'y a plus qu'à cliquer sur le bouton `Ajouter AFFICHER variable`, puis sélectionner `racine` après `AFFICHER la variable` et cliquer sur `OK`.

Notre algorithme est fini et devrait correspondre à ça :

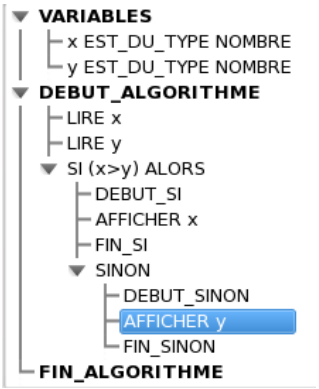


Exemple 3.2 : Conception d'un algorithme qui demande à l'utilisateur d'entrer deux nombres (stockés dans les variables `x` et `y`) et qui affiche le plus grand des deux.
Analyse du problème :

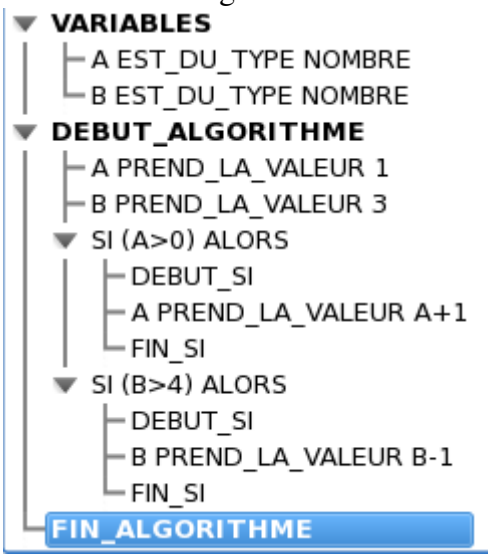
- Il va falloir d''abord déclarer les deux variables `x` et `y` du type NOMBRE (bouton `Déclarer nouvelle variable`), puis LIRE le contenu de ces deux variables (bouton `Ajouter LIRE variable`) .
- On utilise ensuite le raisonnement suivant :
  - SI x>y alors le plus grand des deux nombres correspond à la valeur de x qu'on affiche.
  - SINON (dans la cas contraire), le plus grand des deux nombres correspond à la valeur de y qu'on affiche.

  Concrètement avec **AlgoBox** :
  - on crée une nouvelle ligne (bouton `Nouvelle Ligne`) après les lignes `LIRE x` et `LIRE y`, puis on clique sur le bouton `Ajouter SI...ALORS`.
  - Dans le champ après `SI la condition` de la boîte de dialogue, on entre `x>y` et on coche la case `Ajouter SINON` avant de cliquer sur `OK`.
  - On se place sur la ligne vide entre `DEBUT_SI` et `FIN_SI`, puis on clique sur `Ajouter AFFICHER variable` et on sélectionne `x` comme variable à afficher.
  - On se place sur la ligne vide entre `DEBUT_SINON` et `FIN_SINON`, puis on clique sur `Ajouter AFFICHER variable` et on sélectionne `y` comme variable à afficher.

L'algorithme devrait correspondre à ça :



Activité 3.1 :
On considère l'algorithme ci-dessous :



Déterminer quelles sont les valeurs des variables `A` et `B` à la fin de l'exécution de l'algorithme.

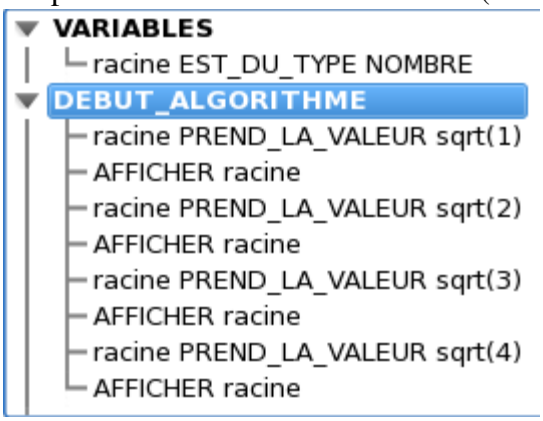
Activité 3.2 :
Concevoir avec **AlgoBox** un algorithme correspondant au problème suivant :

- on demande à l'utilisateur d'entrer un nombre (qui sera représenté par la variable `x`)
- si le nombre entré est différent de 1, l'algorithme doit stocker dans une variable `y` la valeur de 1/(x-1) et afficher la valeur de `y` *(note : la condition x différent de 1 s'exprime avec le code `x!=1`)*. On ne demande pas de traiter le cas contraire.

4. Boucle POUR...DE...A

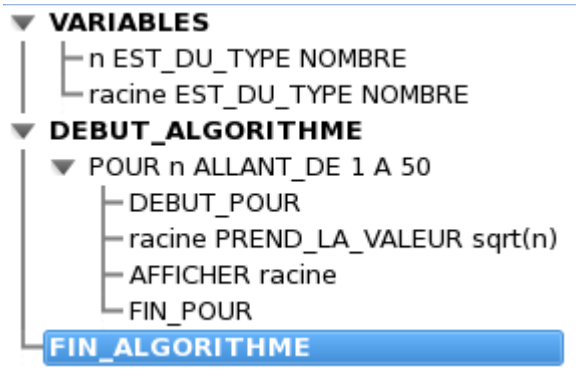
Il est aussi possible de demander à l'ordinateur de répéter une même tache autant de fois que l'on veut. Cela se fait grâce à ce qu'on appelle en informatique une **boucle**.
Regardons de suite un exemple pour voir l'intérêt de la chose :

Exemple 4 : Supposons que l'on veuille créer un algorithme qui affiche la racine carrée de tous les entiers de 1 jusqu'à 50.
On pourrait commencer comme cela (en continuant jusqu'à sqrt(50)):



On voit tout de suite l'inconvénient de cette méthode : on répéterait 50 fois le même genre de lignes de code.

Grâce à une boucle `POUR...DE...A`, on peut éviter tout ça et faire beaucoup plus court.
Voilà ce que ça donnerait comme algorithme :



Avant de voir comment faire avec **AlgoBox**, donnons d'abord quelques explications :

- D'abord il nous a fallu une nouvelle variable `n` qui va servir à représenter tous les entiers de 1 à 50. *(`n` est appelé **compteur** de la boucle)*
- Regardons comment l'ordinateur va interpréter le code de la boucle elle-même :

```
POUR n ALLANT_DE 1 A 50
    DEBUT_POUR
    racine PREND_LA_VALEUR sqrt(n)
    AFFICHER racine
    FIN_POUR
```

  - Première étape : l'ordinateur affecte à la variable `n` la valeur 1. Puis il effectue les opérations comprises entre `DEBUT_POUR` et `FIN_POUR` en remplaçant `n` par 1. Autrement dit, `racine` va prendre la valeur `sqrt(1)` et l'ordinateur affiche le résultat.
  - Deuxième étape : l'ordinateur augmente automatiquement de 1 la valeur de `n` qui vaut donc maintenant 2. Puis il effectue à nouveau les opérations comprises entre `DEBUT_POUR` et `FIN_POUR` en remplaçant cette fois-ci `n` par 2. Donc, `racine` va prendre la valeur `sqrt(2)` et l'ordinateur affiche le résultat.
  - Troisième étape : l'ordinateur augmente de nouveau automatiquement de 1 la valeur de `n` qui vaut donc maintenant 3. Puis il effectue les opérations comprises entre `DEBUT_POUR` et `FIN_POUR` en remplaçant `n` par 3. Autrement dit, `racine` va prendre la valeur `sqrt(3)` et l'ordinateur affiche le résultat.
  - L'ordinateur continue ainsi le processus jusqu'à ce qu'il ait traité le cas où `n` vaut 50. On obtient bien ainsi la liste des racines carrées des entiers de 1 jusqu'à 50.

Voyons maintenant comme créer en pratique cet algorithme avec **AlgoBox** :

- On déclare d'abord les deux variables `n` et `racine` (du type NOMBRE) avec le bouton `Déclarer nouvelle variable`.
- On se place sur `DEBUT_ALGORITHME` et on ajoute une nouvelle ligne (bouton `Nouvelle Ligne`).
- Pour créer la structure de la boucle, on clique sur `Ajouter POUR...DE...A`. Dans la boîte de dialogue qui apparaît: on sélectionne `n` comme variable après `Pour la variable`, on entre 1 et 50 pour les champs `ALLANT DE` et `A`, et on clique sur `OK`.
- En étant bien positionné sur la ligne vide entre `DEBUT_POUR` et `FIN_POUR`, on clique sur `AFFECTER valeur à variable`. Dans la boite de dialogue, on sélectionne `racine` pour la variable, puis on entre `sqrt(n)` dans le champ après `prend la valeur` et on clique sur `OK`.
- On crée une nouvelle ligne (bouton `Nouvelle Ligne`), puis on clique sur `Ajouter AFFICHER variable`.

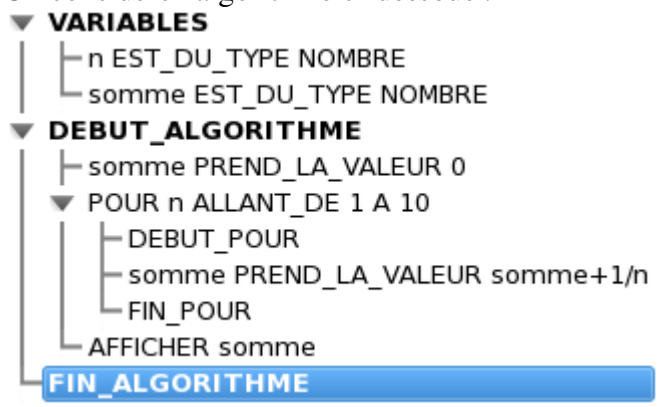
Dans la boite de dialogue, on sélectionne la variable `racine` et on coche la case `Ajouter un retour à la ligne` avant de cliquer sur `OK`.

Remarques importantes sur les boucles `POUR...DE...A` dans **AlgoBox** :

- La variable servant de compteur pour la boucle doit être du type NOMBRE et doit être déclarée préalablement (comme toutes les variables).
- Dans **AlgoBox**, cette variable est automatiquement augmentée de 1 à chaque fois.
- On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais **les instructions comprises entre** `DEBUT_POUR` et `FIN_POUR` **ne doivent en aucun cas modifier la valeur de la variable qui sert de compteur**.

Activité 4.1 :
Concevoir avec **AlgoBox** un algorithme qui affiche, grâce à une boucle `POUR...DE...A`, les résultats des calculs suivants : 8*1 ; 8*2 ; 8*3 ; 8*4 ; ... jusqu'à 8*10 .
L'algorithme devra utiliser deux variables : une appelée `n` pour le compteur de la boucle et une autre appelée `produit` pour stocker et afficher le résultat des calculs.

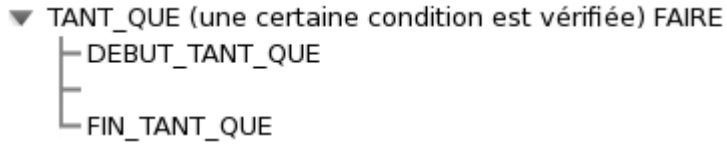
Activité 4.2 :
On considère l'algorithme ci-dessous :



A quel calcul mathématique correspond la valeur de la variable `somme` qui est affichée à la fin de l'exécution de l'algorithme?

5. Structure TANT QUE...

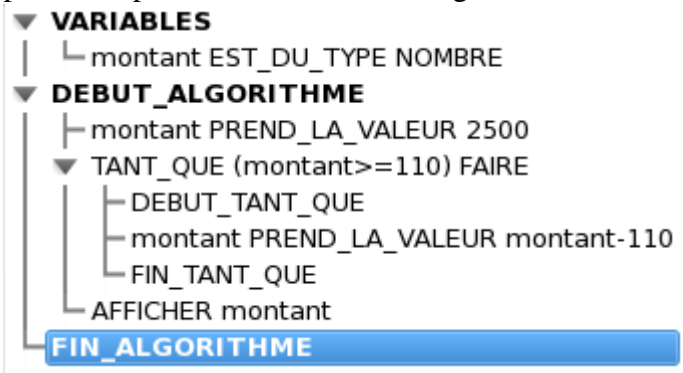
Les boucles POUR...DE...A vues à la page précédente sont très pratiques pour répéter des instructions à condition de savoir le nombre de répétitions nécessaires. Or ce n'est pas toujours le cas : il est alors possible d'avoir recours à la structure TANT QUE...qui permet de répéter une série d'instructions (comprises entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE`) tant qu'*une certaine condition est vérifiée* :



Regardons sur deux exemples comment cette structure peut-être utilisée pour résoudre un petit problème algorithmique :

Exemple 5.1 : Un individu a emprunté à un ami une somme de 2500 euros (prêt sans intérêts). Pour rembourser son ami, il prévoit de lui remettre 110 euros par mois. Comme cela ne donne pas un nombre exact de mois, il se demande quel sera le montant à rembourser le dernier mois.

En utilisant la variable `montant` pour représenter le montant qu'il reste à rembourser, le problème peut se résoudre avec l'algorithme suivant :

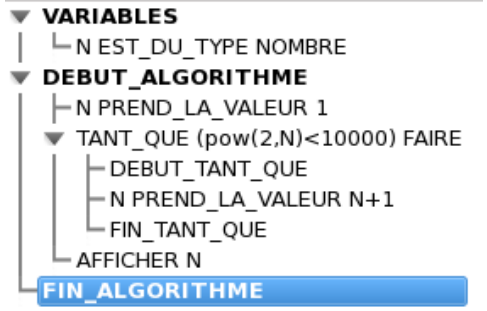


Explication : tant que le montant qui reste à rembourser est supérieur ou égal à 110, l'individu rembourse 110 euros à son ami et le montant à rembourser diminue d'autant. Mais, pour le dernier mois de remboursement, il reste moins de 110 euros à rembourser : l'instruction dans le `TANT_QUE` n'est pas traitée et on affiche alors ce qu'il reste à rembourser.

Voyons maintenant comme créer en pratique cet algorithme avec **AlgoBox** :

- On déclare d'abord la variable `montant` (du type NOMBRE) avec le bouton `Déclarer nouvelle variable`.
- On se place sur `DEBUT_ALGORITHME`, on ajoute une nouvelle ligne (bouton `Nouvelle Ligne`) et on clique sur le bouton `AFFECTER valeur à variable`. Dans la boite de dialogue, on sélectionne `montant` pour la variable, puis on entre `2500` dans le champ après `prend la valeur` et on clique sur `OK`.
- En étant bien positionné sur `montant PREND_LA_VALEUR 2500`, on crée une nouvelle ligne (bouton `Nouvelle Ligne`), puis on clique alors sur `Ajouter TANT QUE...`.
  Dans la boîte de dialogue qui apparaît: on entre comme condition `montant>=110` et on clique sur `OK`.
- En étant bien positionné sur la ligne vide entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE`, on clique sur `AFFECTER valeur à variable`.
  Dans la boite de dialogue, on sélectionne `montant` pour la variable, puis on entre `montant-110` dans le champ après `prend la valeur` et on clique sur `OK`.
- Après s'être positionné sur la ligne `FIN_TANT_QUE`, on crée une nouvelle ligne (bouton `Nouvelle Ligne`), puis on clique sur `Ajouter AFFICHER variable`. Dans la boite de dialogue, on sélectionne la variable `montant` avant de cliquer sur `OK`.

Exemple 5.2 : On cherche à connaître le plus petit entier N tel que $2^N \geqslant 10000$
Pour résoudre ce problème de façon algorithmique, l'idée est de calculer les puissances consécutives de 2 jusqu'à ce qu'on atteigne 10000. Une structure TANT QUE est particulièrement adaptée à ce genre de problème car on ne sait pas a priori combien de calculs seront nécessaires. En utilisant la variable `N`, le problème peut se résoudre avec l'algorithme suivant (*`pow(2,N)` est le code AlgoBox pour calculer* 2^n) :



Explication : tant que $2^N<10000$, on augmente de 1 la valeur de `N` (qui vaut 1 au début de l'algorithme). Par contre, dés que 2^N dépasse 10000, l'instruction entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` n'est pas traitée : on passe alors directement à l'affichage du résultat.
Note : il est **indispensable** d'affecter à `N` la valeur 1 **avant** d'ajouter la structure TANT QUE, sans quoi l'algorithme ne peut plus fonctionner.

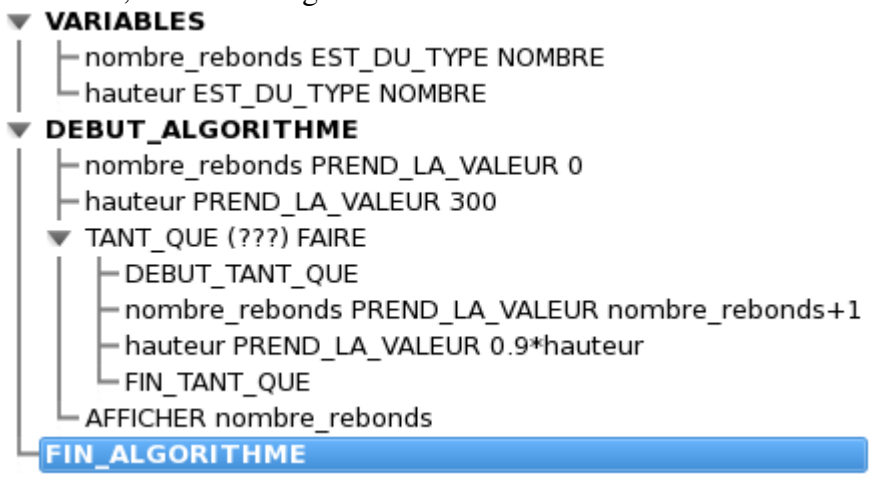
Activité 5.1 : Créer en pratique cet algorithme avec **AlgoBox.**

Remarques importantes sur les structures `TANT QUE...` dans **AlgoBox** :

- On utilise cette structure quand on veut répéter une série d'instructions sans que l'on sache à l'avance combien de fois *(quand on connaît exactement le nombre de répétitions à effectuer, on utilise plutôt une boucle `POUR...DE...A`)*
- Si la condition du `TANT QUE...` est fausse dès le début, les instructions entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` ne sont jamais exécutées *(la structure TANT QUE ne sert alors strictement à rien)*.
- Il est indispensable de s'assurer que la condition du `TANT QUE...` finisse par être vérifiée (le code entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` doit rendre vraie la condition tôt ou tard), sans quoi l'algorithme ne pourra pas fonctionner. Cette structure est donc à manier avec prudence...

Activité 5.2 :
On considère le problème suivant :

- On lance une balle d'une hauteur initiale de 300 cm.
- On suppose qu'à chaque rebond, la balle perd 10% de sa hauteur (la hauteur est donc multipliée par 0.9 à chaque rebond).
- On cherche à savoir le nombre de rebonds nécessaire pour que la hauteur de la balle soit inférieure ou égale à 10 cm.
  Pour cela, on utilise l'algorithme ci-dessous :



Par quelle condition faut-il remplacer ??? dans la ligne `TANT_QUE (???) FAIRE` pour que l'algorithme réponde au problème.