Reformulation de programmes avec PluriAlgo

Ce document complète la partie « sous-programmes » de l'article, en approfondissant les deux mécanismes de reformulation.

Introduction

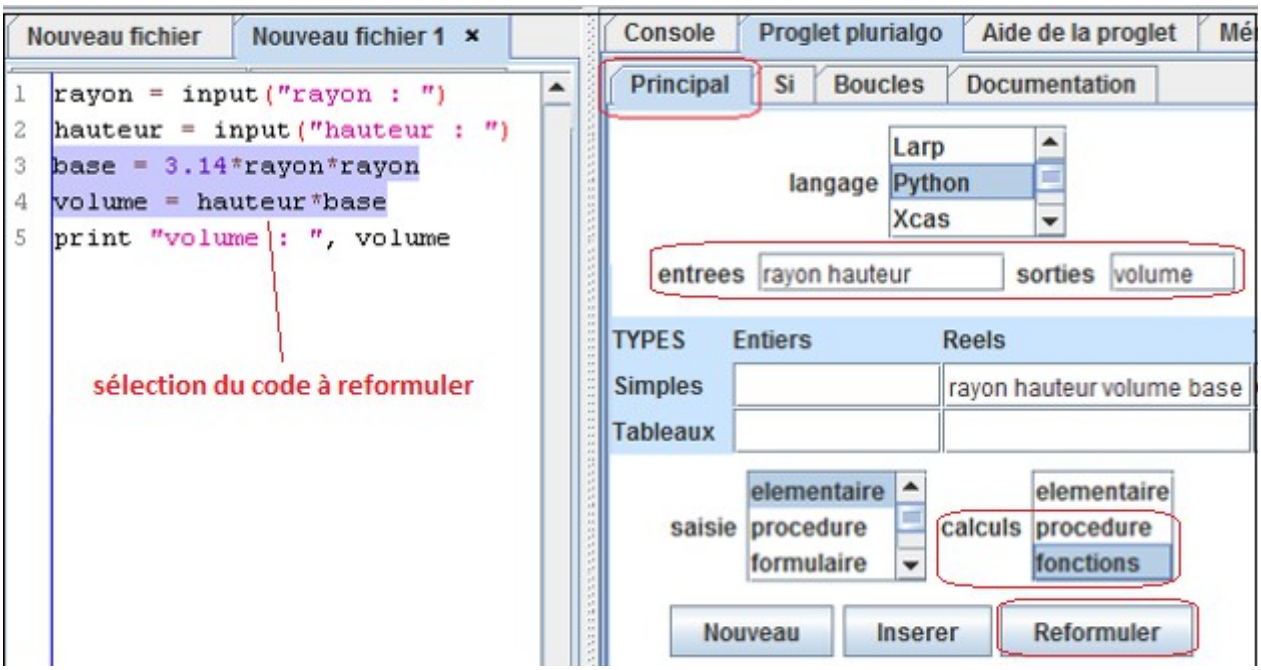
Comme indiqué dans l'article, PluriAlgo propose deux mécanismes de reformulation :

- la reformulation avec sélection de code permet de réécrire un programme initial en introduisant des sous-programmes.
- le reformulation sans sélection de code permet non seulement d'introduire des sous-programmes, mais aussi de créer de nouveaux types (enregistrements ou classes), des formulaires, de changer de langage...

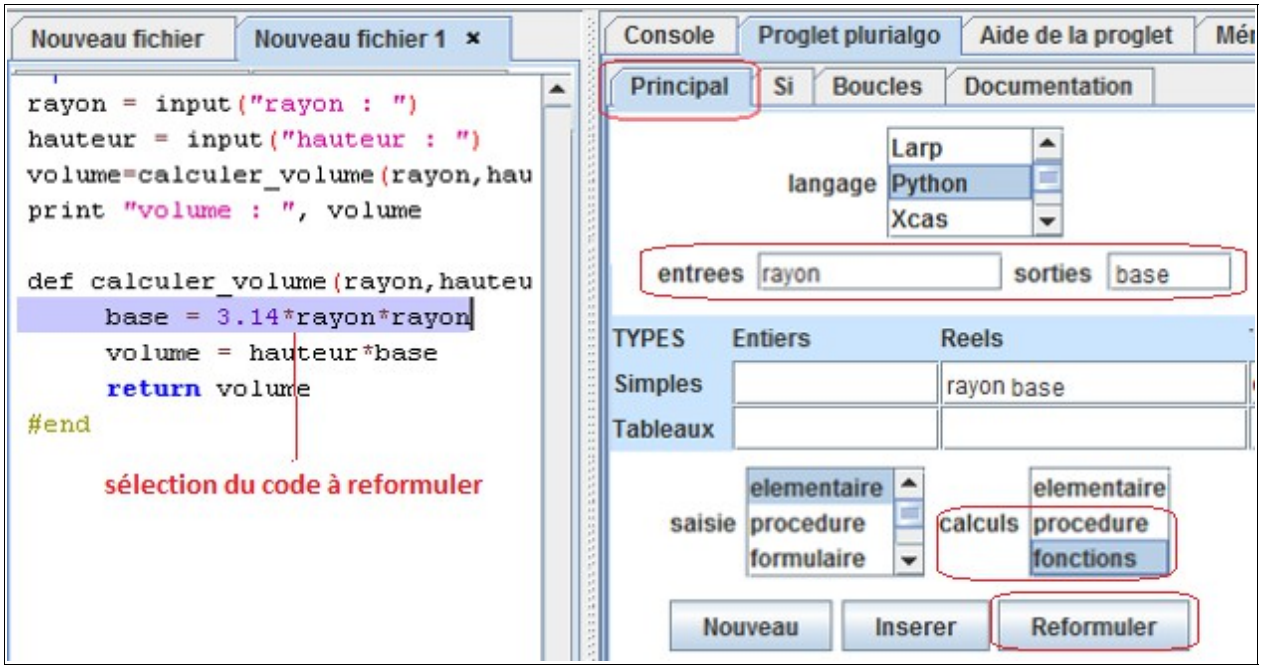
Si vous souhaitez uniquement introduire des sous programmes, il vaut mieux privilégier la reformulation avec sélection de code car elle offre une plus grande souplesse d'utilisation.

Reformulation avec sélection de code

Pour illustrer cette souplesse d'utilisation, nous poursuivons l'exemple (Python) introduit dans l'article :



La fonction créée (nommée par défaut calculer_volume) peut à son tour être reformulée :



Cela permet d'introduire une fonction calculant la base du cylindre :

```
rayon = input("rayon : ")
hauteur = input("hauteur : ")
volume=calculer_volume(rayon,hauteur)
print "volume : ", volume

def calculer_volume(rayon,hauteur) :
    base = calculer_base(rayon)
    volume = hauteur*base
    return volume
#end

def calculer_base(rayon) :
    base = 3.14*rayon*rayon
    return base
#end
```

Reformulation sans sélection de code

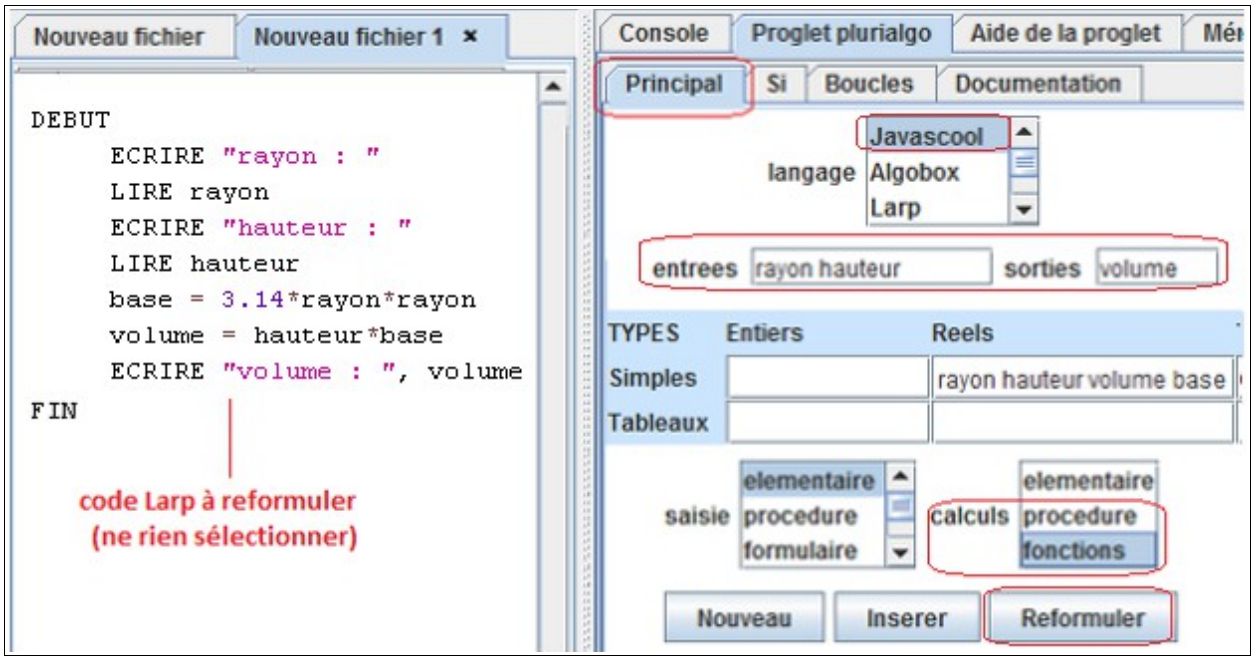
Le mécanisme de reformulation sans sélection de code est beaucoup plus contraignant : il nécessite en effet que PluriAlgo soit capable d'analyser le programme initial, ce qui exclut plusieurs langages (Java, C++, Javascript, R...).

Et pour les autres langages (Javascool, Python, Larp...), l'analyseur peut ne pas fonctionner : si vous souhaitez uniquement introduire des sous programmes, il vaut donc mieux privilégier la reformulation avec sélection de code, d'autant plus qu'elle permet un découpage plus fin en sous-programmes.

Mais la reformulation sans sélection de code a d'autres atouts...

Exemple 1 : changement de langage

Le langage du programme final (à fixer dans l'onglet Principal) peut différer de celui du programme initial (« deviné » par PluriAlgo). Dans mes enseignements, j'utilise cette possibilité pour gagner du temps lorsque je passe d'une initiation à l'algorithmique effectuée en Larp à l'étude d'un langage professionnel (Visual Basic, Java...) :



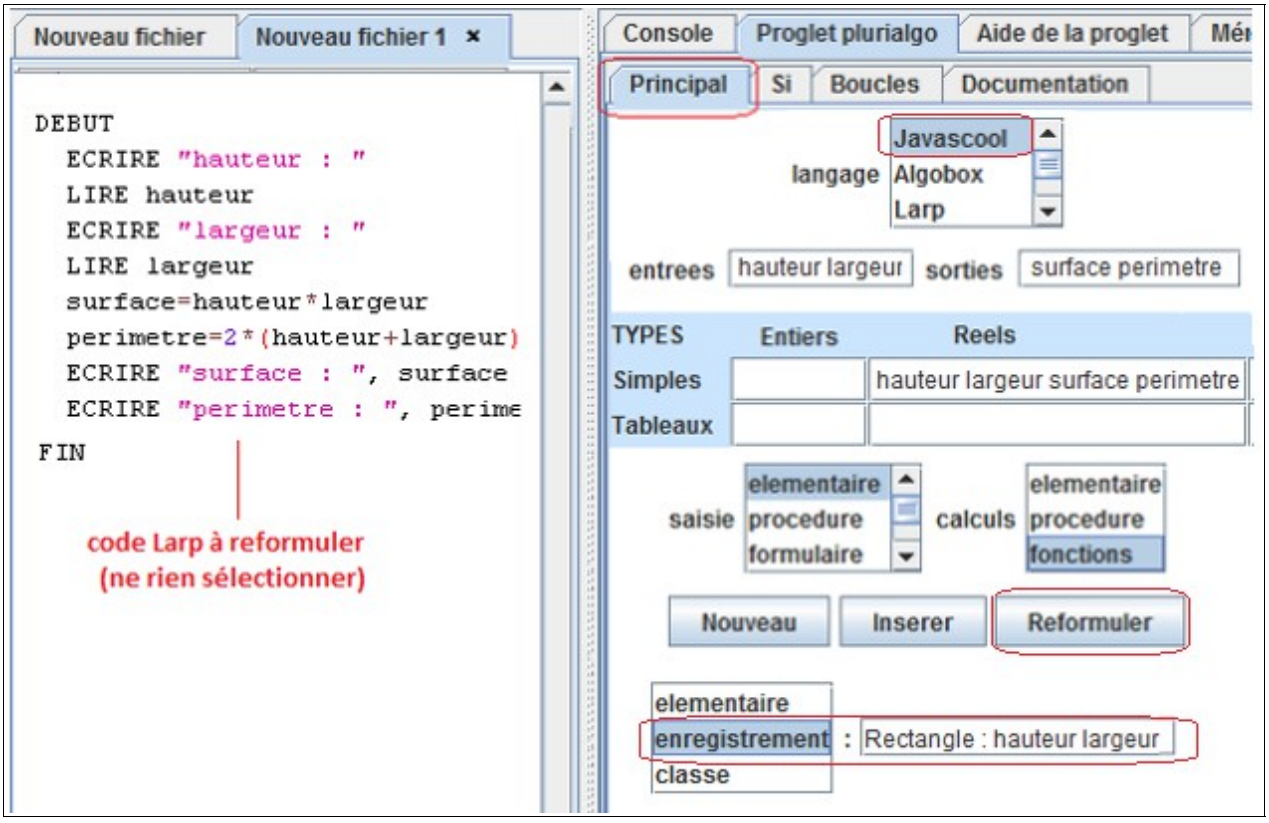
Le résultat obtenu, ici en Javascool, est le suivant :

```
double calculer_volume(double rayon, double hauteur) {
   double base;
   double volume;
   base = 3.14*rayon*rayon;
   volume = hauteur*base;
   return volume;
}
```

```
void main() {
   double rayon;
   double hauteur;
   double volume;
   rayon = readDouble( "rayon : " );
   hauteur = readDouble( "hauteur : " );
   volume = calculer_volume(rayon, hauteur);
   println( "volume : " + volume );
}
```

Exemple 2 : création de nouveaux types (enregistrements ou classes)

Ce nouvel exemple (calcul de la surface et du périmètre d'un rectangle) montre comment créer un nouveau type (Rectangle). Il permet également d'expliquer ce qui se passe lorsqu'on reformule un programme ayant plusieurs sorties.



Le nom du type à introduire (ici Rectangle) est suivi de « : », puis de ses composants (ici hauteur et largeur). Deux options de regroupement sont proposées : « enregistrement » et « classe ». Leur validité dépend du langage proposé, les deux options étant possibles pour le langage ici choisi (Javascool).

> Remarque : nous n'étudierons que l'option « enregistrement » dans ce pdf, mais les programmes obtenus avec l'option « classe » sont disponibles dans le fichier zippé contenant tous les programmes de l'article.

Un type Rectangle regroupant la hauteur et la largeur est créé (ici en Javascool) quand on clique sur le bouton **Reformuler** :

```
class Rectangle {
      double hauteur;
      double largeur;
}
```

Deux fonctions (une par sortie) sont également créées puisque l'option de calcul « fonctions » a été choisie :

```
double calculer_surface(Rectangle objet) {
      double perimetre;
      double surface;
      surface = objet.hauteur*objet.largeur;
      perimetre = 2*(objet.hauteur+objet.largeur);
      return surface;
}

double calculer_perimetre(Rectangle objet) {
      double surface;
      double perimetre;
      surface = objet.hauteur*objet.largeur;
      perimetre = 2*(objet.hauteur+objet.largeur);
      return perimetre;
}
```

A supprimer

Ces deux fonctions sont paramétrées par une variable de type Rectangle, alors qu'elles auraient été paramétrées par les deux entrées (hauteur et largeur) s'il n'y avait pas eu regroupement.

On retrouve dans chaque fonction les deux instructions (hors entrées-sorties) du programme initial, sous une forme modifiée faisant intervenir la variable de type Rectangle. Pour chaque fonction, il faut supprimer une des deux instructions : le calcul du périmètre dans la fonction calculer_surface et le calcul de la surface dans la fonction calculer_perimetre.

En ce qui concerne le programme principal, il n'y a aucune adaptation à faire :

```
void main() {
      Rectangle objet = new Rectangle();
      double surface;
      double perimetre;
      objet.hauteur = readDouble( "objet.hauteur : " );
      objet.largeur = readDouble( "objet.largeur : " );
      surface = calculer_surface(objet);
      perimetre = calculer_perimetre(objet);
      println( "surface : " + surface );
      println( "perimetre : " + perimetre );
}
```