TP 8 – Implémentation de MatL en Python

Dans ce TP, nous allons revenir sur MatL et proposer une implémentation en Python pure.

Exercice 1. COO – Diagramme de classe

Encore une fois, nous allons utiliser un diagramme de classe pour représenter les différents éléments proposés par le langage. Nous allons directement considérer les opérations binaires, opération unaire ainsi que l'utilisation et la définition de variables (donc Expressions, Statements, ...)

1. Proposer un diagramme de classe représentant le système. Que peut-on remarquer par rapport aux types définit en OCaml ?

Exercice 2. Just do it

Écrire les tests et implémenter MatL en Python. Pour faire ceci, il est nécessaire de faire deux choses fondamentales.

1. Écrire les classes correspondantes à tout les types d'expression, statements, ... de MatL.

2. Donner des méthodes d'évaluation sur chaquun de ces types pour calculer le résultat du tout.

3. Évidemment, écrire les tests unitaires pour l'évaluation du langage.

4. Quel est l'intérêt d'avoir une grosse hiérarchie de classe ?

Exercice 3. S'il vous reste du temps

Normalement, si vous avez bien fait le diagramme de classe et que vous avez bien compris le principe de l'objet ici, le TP devrait être assez court. On va donc prendre du temps pour implémenter un simple analyseur de code qui permettra statiquement de vérifier que les variables utilisées ont bien été définies.

1. Pour implémenter facilement ce comportement, nous avons besoin de traverser l'AST MatL. Nous allons utiliser pour ceci le principe d'un visiteur. Vous avez, au choix, la possibilité de coder le pattern visiteur ou, pour éviter de réimplémenter un tel comportement, d'utiliser le `singledispatch` proposé par Python. Si vous décidez d'utiliser le `singledispatch`, cherchez une documentation sur le net (c'est pas ce qui manque).

2. Comment représenter les erreurs dans l'analyse de l'AST ? Proposer un type conservant les informations nécessaires.

3. Implémenter l'analyse de l'AST en utilisant le single dispatch.