Le programme « calc3 »

Alexandre CASAMAYOU-BOUCAU,
Pascal CHAUVIN,
Guillaume CONNAN

1er janvier 2015

Table des matières

1 Introduction

Entre la première édition de l'ouvrage et la présente version, le langage **Python** et son environnement ont fortement évolué.

L'équipe en charge du développement du langage **Python 3** a opéré des choix pour les futures versions du langage, avec pour effet d'enrichir, de modifier le comportement de certaines routines, parfois même de façon profonde. Si les choses évoluent généralement dans le sens d'un meilleur confort ou de fonctionnalités accrues, cette tradition dans le monde des langages informatiques a comme conséquence pour le programmeur de devoir parfois effectuer des efforts significatifs d'adaptation de son code.

Comme il n'est pas rare de le constater dans le monde de **Python** —nombreuses sont par exemple les distributions Linux dans lesquelles les deux environnements **Python** coexistent au sein d'un même système d'exploitation—, l'occasion du changement de version pour **Python 3** a entrainé la réécriture du programme « calc3 ». De ce fait et en introduisant de surcroît quelques nouveautés, les sources du programme ont vu leur volume multiplié par trois (soit une bonne centaine de pages de listings).

Il reste néanmoins bien des endroits du code du programme où les manques et perfectionnements possibles de « calc3 » sont évidents : ces évolutions sont une nouvelle source d'exercices pour l'étudiant. La flexibilité de la conception orientée objet permet notamment d'enrichir le projet dans son état actuel : il s'agit avant tout d'un programme d'apprentissage, en quelque sorte une « preuve de concept ».

2 De Python 2 à Python 3

La migration ne pouvait se résumer à adapter le code source pour prendre en compte les nouveaux comportements de certaines fonctions ou structures définies dans les bibliothèques standard de **Python**.

Même si **Python 3** fournit des outils de migration (par exemple, l'utilitaire « `2to3` » fait partie de toute branche de **Python 3**), il demeure certaines situations qui résistent, et pour lesquelles le programmeur doit reprendre le code à la main.

3 Organisation du projet : des choix de développement

3.1 Organisation du programme

Pour cette nouvelle version de « calc3 », nous avons repensé l'organisation du code source du programme :

— nous avons introduit un nouveau développement modulaire et une hiérarchie de modules. Tout en ayant conservé l'ossature logique de la version précédente du programme, nous avons apporté une hiérarchisation aux différents éléments du programme ;

— pour chaque unité, nous avons pris systématiquement le soin d'écrire immédiatement les tests unitaires qui accompagnent chaque module. Pour cela, nous avons construit, afin d'illustrer le principe, les tests unitaires « à la main », dans un but d'apprentissage de la programmation : le concept de test unitaire est bien sûr présent dans une bibliothèque standard du langage. Encore une fois, notre propos pédagogique est celui de de démonter la roue.

3.2 Les tests unitaires

Nous avons pris le parti de systématiser les tests unitaires pour le développement du projet « calc3 ». Nous rappelons que **Python** offre une gestion des tests unitaires, mais nous avons préféré en adapter le principe, dans un but d'étude, avec les choix suivants :

— chaque module du programme, en tant qu'unité logique, est rédigé dans un fichier source, auquel on joint immédiatement un fichier de tests. Par exemple, le module dédié à la gestion des nombres rationnels se nomme « `rationnel.py` » et s'accompagne du fichier « `rationnel_tests.py` ». Ces fichiers sont placés dans un répertoire évoquant leur nom, c'est-à-dire « `rationnel_mod` » ici, au sein de la hiérarchie globale du projet ;

— chaque fichier de tests est composé au fur et à mesure de l'écriture du module, en adjoignant différentes procédures de test indépendantes, dans lesquelles on teste à chaque fois la conformité entre le comportement désiré et celui obtenu à l'exécution du test, de façon automatisée ;

— chaque procédure de test pourra être paramétrée indépendamment des autres, avec les possibilités d'être silencieuse ou non (pas de sortie textuelle, par exemple), et bloquante ou non. En fin des tests, un rapport sur la console, mentionne la moindre erreur détectée.

Il convient sur ce point d'insister sur deux pratiques répandues et orthogonales entre la démonstration mathématique et la programmation. Au contraire de la démonstration mathématique, qui ne se satisfait pas d'une vérité établie à partir de quelques exemples, on valide une propriété sur quelques exemples considérés comme suffisamment génériques, comme ce fut d'ailleurs le cas jadis dans la tradition mathématique. Cette approche est la base des tests unitaires en programmation. Aujourd'hui néanmoins, la tendance est à la mise en œuvre de systèmes de preuve formelle des programmes. Cette notion en lien avec l'algorithmique est récente dans l'histoire de la programmation, et l'utilisation des systèmes de preuve est encore réservée au développement des programmes critiques.

4 Structure du programme

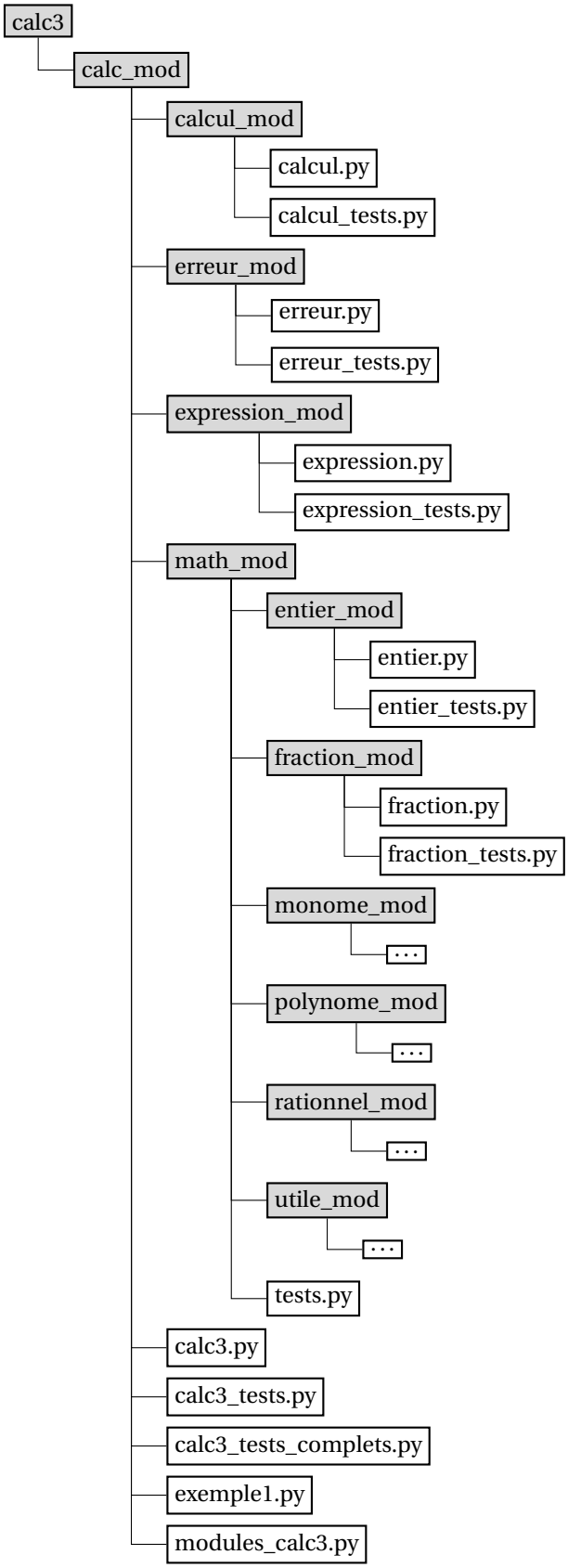
4.1 Les modules en Python

Du point de vue du code **Python**, un programme et un module **Python** ne présentent pas de différence.

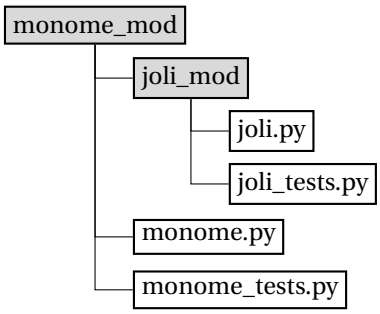
Pour distinguer un programme d'un module, la convention consiste simplement à enregistrer un fichier source dans un répertoire (du système de fichiers). On créé ensuite dans le même répertoire un fichier unique, souvent vide, toujours nommé « `__init__.py` ». La présence de ce fichier provoquera, à la première exécution du programme, la création automatique d'un sous-répertoire « `__pycache__` », comportant du code précompilé pour la machine virtuelle **Python**.

4.2 Hiérarchie des modules du programme « calc3 »

Les rectangles comportant des points de suspension indiquent deux fichiers sources *ad-hoc*, à la manière des sous-répertoires précédents :



On donne ici une vue du contenu du répertoire des modules modélisant le type correspondant à la notion de monôme :



Le point-clef, par la suite, réside dans le fait de déclarer à la machine virtuelle exécutant **Python**, le chemin d'accès aux différents endroits de la hiérarchie. On consultera le code source du programme pour s'en faire une idée, par exemple.

Pour conclure, on mentionnera que **Python** dispose aujourd'hui d'outils de déploiement d'un projet, qui dépassent largement le cadre de cet ouvrage. Citons notamment les systèmes « `Setuptools` » et « `pip` » pour l'installation de programmes **Python** dans les règles de l'art.