Bonnes pratiques de programmation

Cette page est basé sur un document qu'Aurélien Nicosia (ULaval) a créé en 2023 appelé "Bonnes pratiques de programmation en R". Celui-ci a été mis à jour et écrit dans un contexte plus général.

Peu importe le langage informatique, employer de bonnes pratiques de programmation signifie respecter certaines **normes** afin de créer du "bon" code. On peut donc se demander ce qu'est un "bon" code. Pour moi, un "bon" code est un code qui remplit trois objectifs :

1. Il produit les résultats escomptés.
2. Il est facile à maintenir.
3. Son temps de calcul est raisonnable.

Pourquoi est-ce souhaitable d'adopter de bonnes pratiques ? Cela permet que le code soit compris et utilisé par n'importe qui (et en particulier, soi-même dans le futur). À long terme, les bonnes pratiques apportent une augmentation de notre productivité en évitant les répétitions inutiles.

Objectif 1 : Produire les résultats escomptés

La priorité lors du développement de tout code informatique est certainement l'écriture d'un code qui réalise bien ce qu'il doit réaliser. Donc, un "bon" " code doit produire les bons résultats. Pour y arriver, le code doit d'abord être fonctionnel, c'est-à-dire ne pas contenir de bogues. Pour s'assurer d'écrire du code qui fonctionne correctement, il faut simplement le tester. Il vaut mieux tester fréquemment, à chaque petit ajout, plutôt que de produire beaucoup de code avant de le tester. Ainsi, il y a beaucoup moins de débogage à faire. Un courant de pensée en informatique prône même l'écriture des tests avant l'écriture du code (*test driven development*). Cependant, une meilleure pratique est de formaliser les tests afin de pouvoir facilement les lancer à nouveau lors de modifications futures apportées au code.

Objectif 2 : Code facile à maintenir

Maintenir un code informatique signifie de s'assurer qu'il continue de fonctionner correctement dans le futur, malgré les modifications qui lui sont apportées. Un code utilisé fréquemment est un code appelé à être mis à jour, soit pour y ajouter des fonctionnalités, soit pour corriger des bogues non détectés par les tests, mais découverts par des utilisateurs. Reprendre un code écrit par quelqu'un d'autre, ou écrit par nous-mêmes quelques mois auparavant, n'est pas toujours une tâche facile. Cependant, s'il s'agit d'un code correctement écrit, il ne devrait pas être trop difficile à comprendre et à modifier.

La maintenance d'un code est basé sur trois principes : son versionnage, sa compréhensibilité et sa réutilisabilité.

Le principe du versionnage est d'utiliser un logiciel qui enregistre les différentes modifications faites sur le code. Le plus connu est **Git**. Il permet de naviguer entre les différentes versions de votre code, de créer plusieurs versions (appelée branches), de collaborer à plusieurs sur un même code. C'est vraiment un indispensable à avoir dans sa panoplie. Le versionnage peut ensuite être enregistré sur Github qui gère la machinerie sous-jacente. À titre d'exemple, ce site utilise Git et Github pour son versionnage et vous pouvez voir les différentes versions ici.

Un code compréhensible est clair et se lit bien (presque comme du texte). Il comporte souvent des instructions qui parlent d'elles-mêmes. Ces instructions sont typiquement succinctes, car une instruction trop longue effectue souvent plusieurs tâches difficilement discernables. Si la lecture d'une instruction ne permet pas à un programmeur initié dans le langage informatique employé de comprendre ce qu'elle réalise, il est alors recommandé d'insérer un commentaire dans le code pour expliquer à quoi sert l'instruction. En plus de commentaires pour expliquer certains instructions, toutes fonctions devraient être documentées. La documentation d'une fonction devrait contenir : un texte explicatif de ce que fait la fonction, une description des arguments acceptés en entrée, une description des résultats produits et un exemple d'utilisation. Lorsque que l'on programme, il est aussi de bon ton de suivre un **guide de style**. Un guide de style est un ensemble de règles sur lesquelles les développeurs se sont mis d'accord et qui permet d'avoir une syntaxe similaire à travers les différents projets. En R, vous pouvez utiliser le guide de style du tidyverse. En Python, vous pouvez utiliser le guide de style PEP8 écrit par le créateur de Python. Et en Julia, vous pouvez utiliser le guide de style fournit avec le manuel du langage. Vous pouvez utiliser un **linter**, un outil d'analyse de code statique pour vous aidez à respecter ces guides de styles. À noter que ces guides de style sont des recommendations et il n'y a rien d'obligatoire à les suivre. Certaines règles se contredisant, je vous recommanderai même de ne pas en suivre certaines.

La façon la plus commune d'avoir un code facile à réutiliser est d'en faire des fonctions que l'on peut ensuite partager à travers un package.

Objectif 3 : Code suffisamment rapide

Après nous être assurés que notre code fonctionne correctement et qu'il est facilement maintenable, nous pouvons nous préoccuper de son temps d'exécution. Bien qu'il ne s'agisse pas du critère le plus important pour définir ce qu'est du "bon" code, c'est tout de même un critère à ne pas négliger, car un code trop lent risque de ne pas être utilisé. Pour produire du code computationnellement efficace, il faut :

— mettre en pratique quelques trucs simples, i.e. utiliser les syntaxes optimisées des différents langages ;
— comparer le temps d'exécution de différentes façons de programmer une tâche ;
— parfois faire du calcul en parallèle ;
— parfois programmer des bouts de code dans un autre langage plus bas niveau.

En résumé

En résumer, pour adopter de bonnes pratiques de programmation, il faut :

— **Tester** son code fréquemment son code.
— Utiliser un logiciel de **gestion de versions**.
— **Documenter** son code.
— Respecter un **guide de style**.
— **Factoriser** son code en créant des fonctions et des packages.
— **Optimiser** le temps d'exécution.