7 Méthodes secrètes des informaticiens pour mieux programmer

Leroy Régis (@regilero Twitter/GitHub)

Résumé (max 300 mots)

Un programme R qui ne marche que sur un vieux poste Windows XP et que personne n'arrive à réinstaller ailleurs ? Un informaticien qui grimace en étudiant votre code ? Vous avez des doutes sur le fonctionnement de ce programme que vous aviez écrit il y a 5 ans ? Comment éviter ces situations ?

Le but de cette présentation est de diffuser quelques bonnes pratiques de programmation au sein d'une population composée de profils scientifiques qui pratiquent la programmation sans que cela ne soit leur principale expertise.
Cette présentation se base sur des retours d'expérience en audit ou en conseil auprès de différentes communautés scientifiques, avec identification de problèmes récurrents. Constats qui ont donnés lieu à la création d'une formation professionnelle sur les bonnes pratiques de développement destinée spécifiquement au Data Scientists.

La présentation est effectuée par Leroy Régis, DevOp, formateur, spécialiste en sécurité web, bases de données et en développement web Python et PHP. Il pratique parfois le R, le plus souvent pour adapter, corriger, et améliorer des programmes existants.

Mots-clefs (3 à 5) : Développement – Ingénierie – Bonnes pratiques

Développement

Note : Cette présentation peut s'adapter au format court (5 minutes) ou long (30 minutes) en fonction de vos besoins. Une présentation plus longue permet de montrer des démonstrations (effets du lintr, gestion de dépendances, exemples de README, etc.).

Ces « méthodes secrètes », ne sont pas secrètes mais simplement mal connues en dehors des équipes pouvant se prévaloir de l'assistance d'un informaticien. Il s'agit donc de rappeler quelques principes de bonne programmation et de donner des clefs pour obtenir des programmes R plus robustes, plus portables et plus facilement maintenables dans le temps.

1. De la documentation : que mettre dans un README ? Documenter l'usage mais aussi la pensée, les objectifs, les expérimentations, les volumes et les contraintes
2. Une configuration centralisée : éviter les répétitions et le code trop spécifique, centraliser et factoriser les éléments variables, gérer un paramétrage
3. Un environnement maîtrisé et reproductible (par exemple anaconda), comment gérer ses dépendances en R ?
4. Un programme ré-entrant : automatiser les corrections manuelles, permettre de relancer le programme pour ré-obtenir un résultat stable, intégrant les correctifs manuels
5. Une règle de 3 : copier-coller ou bien factoriser ?. S'autoriser un maximum de 3 copier/coller avant d'enclencher une réflexion de factorisation. Comment savoir si une fonction est trop longue ou trop petite ?

* Makina Corpus, regis.leroy@makina-corpus.com

6. Pourquoi utiliser un gestionnaire de version (git) ? Versionner sans dépôt distant, historiser, faciliter les modifications.
7. Les tests. Les Tests unitaires, mais aussi les outils de qualité comme les ‘linters’.

Pour aller plus loin : packager, contribuer, rapporter des bugs, publier, cerner les usages, la sécurité, etc.

Références

La formation sur laquelle cette présentation repose est en cours de développement, elle devrait cependant être disponible avant Juin 2023.