Retours donnés à vos prédécesseurs sur le projet d'informatique 2019-2020

Prépa CPP - informatique S4

Préliminaire

Les retours ci-dessous ont pour objectif de vous aider à progresser, notamment à améliorer vos pratiques de programmation et vos rapports dans le futur. Il ne s'agit en aucun cas d'une liste de reproches, et nous avons dans la notation été très tolérants sur les erreurs classiques décrites plus bas, étant conscients que ce type de travail est pour vous une nouveauté et que les conditions de ce semestre étaient particulièrement difficiles.

Principes du travail par projet et conseils généraux

- **Lisez attentivement !** C'est dommage de rater des points faciles en ne répondant pas entièrement à une question, d'en perdre en ne respectant pas une consigne, ou de ne pas bénéficier des informations et conseils diffusés sur la page web du cours.
- Le travail par projet est relativement nouveau pour vous. C'est un type d'enseignement qui deviendra plus classique dans la suite de vos études. Le projet est une façon d'aborder des notions nouvelles, et non pas une façon d'évaluer votre compréhension d'un cours en amphi comme pourrait l'être un devoir maison. Il est donc *parfaitement normal* qu'un certain nombre de choses soient nouvelles pour vous. L'idée est d'aborder ces notions par le projet, en lisant la documentation, en s'appuyant sur les exemples de la FAQ, et en posant des questions si besoin.
- Un tel projet n'a donc de l'intérêt que si travaillé tout au long du semestre. Il est illusoire d'espérer le réussir et en tirer quelque chose en s'y mettant une semaine avant la date de rendu. La réflexion dans la durée sur l'organisation des programmes et les questions-réponses avec vos enseignants font partie du processus.
- Un des objectifs du cours est de vous aider à devenir plus autonomes en programmation. Ne pas réussir un programme du premier coup est parfaitement normal, qu'on soit débutant ou expert. Lire et comprendre les messages d'erreurs, tester chaque fonction, utiliser des print ou le debugger pour comprendre ce qui se passe, etc, font partie du processus normal d'écriture d'un programme complexe et donc des choses qu'on cherche à vous faire découvrir.
- C'est pour ça que nos réponses à vos questions sont souvent d'autres questions : on cherche à vous aider à mieux formuler le problème, à décrire précisément le comportement attendu du programme et le comportement observé. C'est frustrant pour vous parce que vous espérez une réponse simple à une question simple, mais c'est nécessaire : vous commencez à vous attaquer à des problèmes complexes pour lesquels les défauts de logique ou l'absence de formulation précise de ce que doit faire un programme sont souvent des sources d'erreurs importantes.
- Vous devez comprendre et utiliser au mieux les informations que vous donne l'interpréteur Python. L'ordinateur doit être vu comme une façon d'obtenir un retour sur chaque programme (plus efficace que l'enseignant). Quand vous nous demandez de l'aide avec un programme qui plante, on vous fait souvent remarquer un message de l'interpréteur que vous n'avez pas pris en compte au lieu de répondre directement à votre question. Là aussi je sais que c'est frustrant pour vous, mais c'est nécessaire pour progresser en programmation.

Conseils pour le rapport

- Un élément central du rapport est le regard critique qu'on attend que vous portiez sur vos résultats. On attend notamment de vous que vous fassiez le lien entre vos prédictions et vos observations. Si un de vos résultats vous semble incohérent, mentionnez-le.

- L'honnêteté est essentielle : si vous dites dans le rapport qu'un programme "fonctionne et donne les résultats attendus" alors que les programmes que vous rendez ne tournent pas ou donnent des résultats aberrants, ça donne une mauvaise impression. Dans le meilleur des cas le correcteur verra cela comme de la négligence (absence de tests), mais il peut aussi penser que vous n'êtes pas honnêtes. En plus de perdre les points attribués à cette question, vous perdrez alors le bénéfice du doute pour toutes les autres questions. À l'inverse, quand une idée correctement implémentée ne donne pas les résultats espérés et que vous le mentionnez honnêtement dans le rapport, on donne généralement une partie des points.

- Plus n'est pas toujours mieux : une réponse concise est toujours appréciée. Ne cherchez pas à noyer le poisson (tenter de dissimuler l'absence d'une partie de la réponse au milieu d'une multitude de détails), le correcteur s'en rendra très probablement compte !

- Un graphique doit toujours faire ressortir clairement l'information que vous souhaitez mettre en avant. Prenez le temps de produire des graphiques proprement annotés (nom des axes...), lisibles (taille du texte et épaisseur des traits appropriés), avec les extrémités des axes bien choisies. Faites le lien entre le texte et les graphiques autant que possible.

- Soyez clairs, précis et consis pour vos résultats analytiques (et raisonnements associés), comme vous le seriez en mathématiques.

Conseils coté programmation

- **Testez systématiquement vos fonctions et programmes !** C'est absolument indispensable et ça vous fera gagner beaucoup de temps sur le long terme. Si une fonction de base n'est pas parfaitement correcte, tout le reste de votre travail risque d'être impacté. Pensez donc à tester scrupuleusement les fonctions clés sur des cas complexes. Les tests que vous réalisez peuvent être mentionnés dans le rapport et présents dans le rendu.

- Lorsqu'il y a des consignes précises, **respectez les strictement**. Si on demande d'écrire une fonction `do_this` dans `utils.py`, on ne veut pas une fonction `Do_This` dans `util.py` ni une fonction `dothis` dans `Utils.py`. Si on demande une fonction qui prend un argument qui est un couple de deux entiers, on ne veut pas une fonction qui prend deux arguments entiers. Cette rigueur est indispensable en programmation, elle vous permettra notamment d'écrire des fonctions interopérables avec celles écrites par d'autres, facilitant le travail en groupe. Elle permet également au correcteur de tester facilement vos fonctions, ce qui est à votre avantage : on peut alors vous attribuer une partie des points pour une fonction correcte même si le programme final ne fonctionne pas.

- Lorsque ce n'est pas imposé par l'énoncé, il faut toujours expliquer précisément ce que fait chacun de vos programmes, comment il peut être lancé, avec quels arguments... La convention est d'inclure un fichier README qui contient ces informations, mais vous pouvez autrement les inclure dans le rapport.

- Assurez-vous que ce que vous écrivez dans le rapport corresponde bien à la version finale des programmes que vous rendez. Si vous mettez des `print` un peu partout pour débugger, pensez à les enlevez dans la version du rendu.