Notes et références

Tu trouveras ici des commentaires et des indications de lectures sur chacune des activités.

Ressources générales

- *Apprendre à programmer avec Python 3* de Gérard Swinnen, aux éditions Eyrolles. C'est un des livres de référence en français pour débuter. La seconde moitié du livre contient des notions beaucoup plus avancées. Le livre est disponible gratuitement en téléchargement, selon la licence *Creative Commons BY-NC-SA* :
  inforef.be/swi/python.htm
- La documentation officielle de `Python` contient des tutoriels et les explications de chacune des fonctions. Malheureusement certaines pages sont en anglais
  docs.python.org/fr/3/
- *Wikipédia* est une source fiable pour en savoir plus sur certaines notions (principalement les projets) mais le niveau n'est pas toujours adapté à un lycéen.
- *Internet* et en particulier les forums ont bien souvent la réponse aux questions que tu te poses !
- Les plus fondus d'entre vous peuvent participer au *projet Euler* (en anglais) qui propose une liste d'énigmes mathématico-informatique. Accrochez-vous !
  projecteuler.net

1. Premiers pas

L'apprentissage d'un langage de programmation peut être très difficile. C'est assez dur d'apprendre tout seul dans son coin. Il n'est pas rare de reste bloquer plusieurs heures pour une bête erreur de syntaxe. Il faut commencer modestement, ne pas hésiter à recopier du code déjà écrit par d'autres, être persévérant et demander de l'aide rapidement !

2. Tortue (Scratch avec Python)

L'idéal c'est de bien maîtriser *Scratch* avant de s'attaquer à `Python` ! Pour ceux qui ont zappé *Scratch*, il y a les activités *Scratch au collège*. Le livre et les vidéos sont disponibles ici :
exo7.emath.fr et youtube.com/ScratchAuCollege
`Python` propose un module *Tortue* qui fonctionne sur le même principe que le déplacement avec *Scratch*. Bien sûr au lieu de déplacer des blocs, il faut écrire le code ! C'est un bon exercice de transcrire en `Python` toutes les activités que tu sais faire avec *Scratch*.

3. Si ... alors ...

On attaque vraiment la programmation avec le test « si/sinon ». L'ordinateur agit donc d'une façon ou d'une autre selon la situation. Ce n'est plus un automate qui fait toujours la même chose. En plus avec l'entrée au clavier, on peut commencer à avoir des programmes interactifs.
Les erreurs de syntaxe classiques sont :

- oublier les deux points après `if condition:` ou bien `else:`,
- mal indenter les blocs.

Ces erreurs seront signalées par `Python` avec le numéro de la ligne où il y a un problème (normalement votre éditeur place le curseur sur la ligne fautive). Par contre si le programme se lance mais qu'il ne fait pas ce qu'il faut, c'est sûrement la condition qui est mal formulée. C'est plus compliqué de bien comprendre la condition qui convient : un peu de logique et de réflexion avec papier/crayon sont les bienvenues. Certains éditeurs `Python` permettent aussi une exécution « pas à pas » du programme.
On reviendra par la suite sur le « vrai » et le « faux ». Il existe aussi le test

```
if ... elif ... elif ... else ...
```

qui permet d'enchaîner les tests et qui n'est pas abordé ici. Bien comprendre `if ... else ...` est suffisant !

4. Fonctions

Assez rapidement il faut comprendre la structure d'un programme informatique : on décompose le programme en blocs de définitions et d'actions simples. On regroupe des actions simples dans des actions intermédiaires. Et à la fin le programme principal consiste juste à exécuter quelques bonnes fonctions.
Les arguments/paramètres d'une fonction sont d'un apprentissage délicat. On peut définir une fonction par `def fonc(x):` et l'appeler par `fonc(y)`. Le `x` correspond à une variable mathématique muette. En informatique, on préfère parler de la ***portée*** de la variable qui peut être locale ou globale.
Ce que l'on peut retenir, c'est que tout ce qui passe à l'intérieur d'une fonction n'est pas accessible en dehors de la fonction. Il faut juste utiliser ce que renvoie la fonction. Il faut s'interdire l'utilisation de `global` dans un premier temps.
Dernier point, il est vraiment important de commenter abondamment ton code et de bien expliquer ce que font tes fonctions. Si tu définis une fonction `fonc(x)` : prévois trois lignes de commentaires pour (a) dire ce que fait cette fonction, (b) dire quelle entrée est attendue (`x` doit être un entier ? un nombre flottant ? positif ?...), (c) dire ce que renvoie la fonction.
Il faut aussi commenter les points principaux du code, donner des noms bien choisis aux variables. Tu seras bien content d'avoir un code lisible lorsque tu reliras ton code plus tard !
Un bon informaticien devrait vérifier que le paramètre passé en argument vérifie bien l'hypothèse attendue. Par exemple si `fonc(x)` est définie pour un entier `x` et que l'utilisateur transmet une chaîne de caractères, alors un gentil message d'avertissement devrait être envoyé à l'utilisateur sans faire échouer tout le programme ! Les commandes `assert`, `try/except` permettent de gérer ce genre de problème. Pour notre part, nous nous abstiendrons de ces vérifications en supposant que l'utilisateur/programmeur utilise les fonctions et les variables en bonne intelligence !

5. Arithmétique – Boucle tant que – I

Nous utiliserons uniquement `Python3`. Si vous ne savez pas quelle version vous avez, tapez `7/2` : `Python3` renvoie `3.5` alors que `Python2` renvoie `3` (dans la version 2, `Python` considérait que la division de deux entiers devait renvoyer un entier).
Avec `Python3` c'est plus clair, `a / b` est la division habituelle (des nombres à virgules) alors que `a // b` est la division euclidienne entre deux entiers et renvoie le quotient.

6. Chaînes de caractères – Analyse d'un texte

Manipuler les chaînes permet de faire des activités sympas et quitter un peu le monde mathématique. Tu peux programmer des quiz, des programmes qui discutent avec l'utilisateur… Les chaînes de caractères sont surtout une bonne introduction à la notion de liste, qui est un outil essentiel par la suite.

7. Listes I

`Python` est particulièrement souple et agile pour l'utilisation des listes. Les « vieux » langages n'autorisaient souvent que les listes contenant un seul type d'élément, et pour parcourir une liste il fallait toujours procéder ainsi :

```
for i in range(len(liste)):
    print(liste[i])
```

Alors que :

```
for element in liste:
    print(element)
```

est beaucoup plus naturel !
Le tri d'une liste est une opération fondamentale en informatique. Imaginerait-on un dictionnaire contenant 60 000 mots, mais non classés par ordre alphabétique ? Ordonner une liste est une opération difficile, il existe beaucoup d'algorithmes de tri. L'algorithme du tri à bulles, présenté ici est l'un des plus simples. Programmer des algorithmes plus rapides au lycée est un beau challenge : il faut comprendre la récursivité et la notion de complexité.

8. Statistique – Visualisation de données

Si tous les lecteurs arrivent à programmer les calculs de somme, moyenne, écart-type, médiane… et leur visualisation, alors l'objectif de ce livre est atteint ! Cela prouve une bonne compréhension des outils mathématiques et informatiques de base.
Le module `tkinter` permet un affichage graphique. Pour commencer, il faut recopier les lignes d'un code qui fonctionne sans trop se poser de questions, puis l'adapter à ses besoins.

9. Fichiers

Les fichiers permettent de faire communiquer `Python` avec le monde extérieur : on peut par exemple récupérer un fichier de notes pour en calculer les moyennes et produire un bulletin pour chaque élève. Pour les plus avancés d'entre vous les fichiers sont une bonne occasion d'utiliser la gestion des erreurs avec `try/except`.
Les activités sur les images sont sympathiques et ces formats d'images seront utilisés par la suite, même si ce format a le gros désavantage de produire des fichiers de grande taille. Néanmoins ils sont standards et reconnus par les logiciels d'images (*Gimp* par exemple). Notez que certains logiciels écrivent des fichiers avec une seule donnée par ligne (ou bien toutes les données sur une seule ligne). C'est un bon exercice d'implémenter la lecture de tous les formats possibles !

10. Arithmétique – Boucle tant que – II

L'arithmétique en général et les nombres premiers en particulier ont une très grande importance en informatique. Ce sont eux qui sont à la base de la cryptographie moderne et assurent la sécurité des transactions sur internet.
Les algorithmes présentés ici sont bien sûr élémentaires. Il existe des techniques sophistiquées pour savoir si un nombre de plusieurs centaines de chiffres est premier ou pas en quelques secondes. Cependant, factoriser un entier de grande taille reste un problème difficile.
Un ordinateur montre sa puissance quand il manipule une grande quantité de nombres ou bien de très grands nombres. Avec l'arithmétique, on a les deux en même temps !

11. Binaire I

La première difficulté avec l'écriture binaire c'est de bien faire la différence entre un nombre et l'écriture du nombre. Nous sommes tellement habitués à l'écriture décimale que l'on a oublié son origine, 1234 c'est juste $1 \times 1000 + 2 \times 100 + 3 \times 10 + 4 \times 1$.
Comme les ordinateurs travaillent avec des 0 et 1 il faut être à l'aise avec l'écriture binaire. Le passage à l'écriture binaire n'est pas très difficile, il est tout de même préférable de faire quelques exemples à la main avant de s'attaquer à la programmation.
On utilisera l'écriture binaire pour d'autres problèmes sur le principe suivant : vous avez 4 interrupteurs alors 1.0.0.1 signifie que vous actionnez le premier et le dernier interrupteur et pas les autres.

12. Listes II

Les listes sont tellement utiles que `Python` possède toute une syntaxe efficace pour les gérer : le tranchage des listes et les listes par compréhension. On pourrait bien sûr s'en passer (c'est d'ailleurs le cas pour la plupart des autres langages) mais ce serait dommage. Nous aurons aussi besoin de listes de listes et en particulier de tableaux à deux dimensions pour afficher de belles images.

13. Binaire II

Il y a 10 sortes de personnes, celles qui comprennent le binaire et les autres !

14. Probabilités – Paradoxe de Parrondo

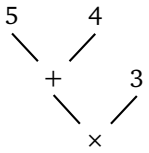
On attaque les projets avec un peu de probabilité et un joli paradoxe, surprenant (comme tous les paradoxes), mais en plus découvert récemment.
Cette activité est calquée sur l'article « Paradoxe de Parrondo » par Hélène Davaux (*La gazette des mathématiciens*, 2017).

15. Chercher et remplacer

Chercher et remplacer sont deux actions tellement courantes que l'on ne se rend pas toujours compte de leur force. Tout d'abord c'est un bon exercice de programmer soi-même les opérations de recherche et de remplacement. Il existe une version sophistiquée de ces opérations qui s'appellent les expressions rationnelles (*regex*). C'est un langage à part entière, très puissant, mais un peu ésotérique.
Il y a aussi les mathématiques du « chercher/remplacer » ! Les activités proposées sont des exemples qui illustrent le théorème principal de l'article *A complete characterization of termination of* $0^p1^q \to 1^r0^s$, par H. Zantema et A. Geser (*AAECC*, 2000) qui est introduit par Pierre Lescanne sur le site « Images de mathématiques » : Est-ce que ça s'arrête ?

16. Calculatrice polonaise – Piles

L'écriture polonaise (le nom exact est « écriture polonaise inverse ») est une autre façon d'écrire les opérations algébriques. Encore une fois, le plus dur c'est de s'adapter à ce changement d'écriture. Cela permet de voir les opérations (et leurs priorités) sous un nouveau jour. Une autre vision intéressante est de voir une opération sous la forme d'un d'arbre binaire :



qui représente $(5+4)\times 3$ ou encore en notation polonaise $5\ 4\ +\ 3\ \times$.
On a essayé de repousser au maximum le changement d'une variable globale dans une fonction, mais ici il est naturel. L'utilisation de `global` est à éviter en général.
La notion de pile est une façon très simple de structurer et d'accéder aux données. C'est pourtant la bonne façon de gérer une expression avec parenthèses ! L'analogie avec la gare de triage devrait être éclairante. Nous retrouverons les piles dans l'activité sur les L-systèmes.
Une pile fonctionne sur le principe « dernier entré, premier sorti » (*filo* pour *first in, last out*). Une autre gestion des données possible est sur le principe « premier entré, premier sorti« (*fifo* pour *first in, first out*) comme dans une file d'attente.

17. Visualiseur de texte – Markdown

Le but de cette fiche est double : découvrir le *Markdown* qui est un langage très pratique pour formater un texte, mais aussi comprendre la justification d'un paragraphe.
Évidemment le langage *Markdown* possède davantage de balises que celles présentées ici. On pourrait poursuivre le projet en réalisant la justification avec des polices de tailles et de formes différentes, voir même créer un petit traitement de texte complet.

18. L-système

On retrouve le thème « chercher/remplacer » mais cette fois avec une vision géométrique. Les figures obtenues sont belles, faciles à programmer à l'aide de la tortue, mais le plus joli c'est de voir le tracé en direct des L-systèmes. Pour les L-systèmes définis par des expressions contenant des crochets on retrouve la notion de pile.
Les formules sont tirées du livre *The algorithmic beauty of plants*, par P. Prusinkiewicz et A. Lindenmayer (Springer-Verlag, 2004) en accès libre ici : *The algorithmic beauty of plants* (pdf).
Les illustrations qui débutent chaque partie de ce livre sont des itérations du L-système appelé la courbe de Hilbert et défini par :

```
depart = "X"  regle1 = ("X","gYAdXAXdAYg")  regle2 = ("Y","dXAgYAYgAXd")
```

19. Images dynamiques

Cette activité est calquée sur l'article « Images brouillées, images retrouvées » par Jean-Paul Delahaye et Philippe Mathieu (*Pour la Science*, 1997). Cet article s'intéresse en plus au calcul du nombre d'itérations qu'il faut avant de retrouver l'image de départ. La notion mathématique sous-jacente est celle de *permutation* : une transformation bijective d'un ensemble fini (ici l'ensemble des pixels) dans lui-même.

20. Jeu de la vie

Un grand classique de l'informatique amusante ! On trouvera sur internet des dizaines de sites avec des constructions aux propriétés incroyables et plein d'autres idées. Mais le plus fascinant reste que des règles extrêmement simples conduisent à des comportements complexes qui ressemblent à la vie et la mort des cellules.

21. Graphes et combinatoire de Ramsey

Les graphes sont des objets très courants en mathématique et en informatique. Le problème présenté ici est simple et amusant. Mais ce qu'il faut peut être retenir de cette fiche, c'est l'accroissement de la difficulté avec le nombre de sommets. Ici on effectue les calculs jusqu'à 6 sommets et on ne peut pas aller beaucoup plus loin avec notre méthode de vérification exhaustive.
Un grand mathématicien Paul Erdös a déclaré que si des extra-terrestres débarquaient sur Terre en menaçant de détruire notre planète sauf si on savait résoudre le problème des 5 amis/5 étrangers, alors en mobilisant tous les ordinateurs et les mathématiciens du monde on arriverait à s'en sortir (on sait que la réponse est entre 43 et 48 personnes). Par contre si les extra-terrestre nous demandaient de résoudre le problème pour 6 amis/6 étrangers alors le plus simple serait de se préparer à la guerre !

22. Bitcoin

Avant de placer toutes ses économies dans des *bitcoins* mieux vaut en comprendre le fonctionnement ! Les activités présentées ici ont pour but de présenter une version (très) simplifiée de la *blockchain* qui est à la base de cette monnaie virtuelle. Le principe de la *blockchain* et de la preuve de travail ne sont pas si compliqués, tu trouveras des explications plus détaillées dans les articles de Jean-Paul Delahaye parus dans la revue *Pour la science* :

- Bitcoin, la cryptomonnaie (2013) (www.lifl.fr/∼jdelahay/pls/2013/241.pdf)
- Les preuves de travail (2014) (cristal.univ-lille.fr/j̃delahay/pls/2014/245.pdf)
- Du bitcoin à Ethereum : l'ordinateur-monde (2016) (cristal.univ-lille.fr/j̃delahay/pls/2016/276.pdf)

23. Constructions aléatoires

Ces constructions aléatoires sont tout d'abord des récréations informatiques qui produisent des jolies figures, toutes ressemblantes mais toutes différentes. Mais elles font aussi l'objet de travaux mathématiques modernes et difficiles. Martin Hairer a obtenu la médaille Fields en 2014 pour l'étude de la frontière supérieure de nos blocs qui tombent, dont la forme est régie par une équation, appelée « équation KPZ ». Une bonne activité bonus serait de faire tomber des blocs du jeu *Tetris* à la place des petits carrés.