Algorithmique et programmation Python

Nicolas Poulain

avec la participation de Julien Baldacci

Table des matières

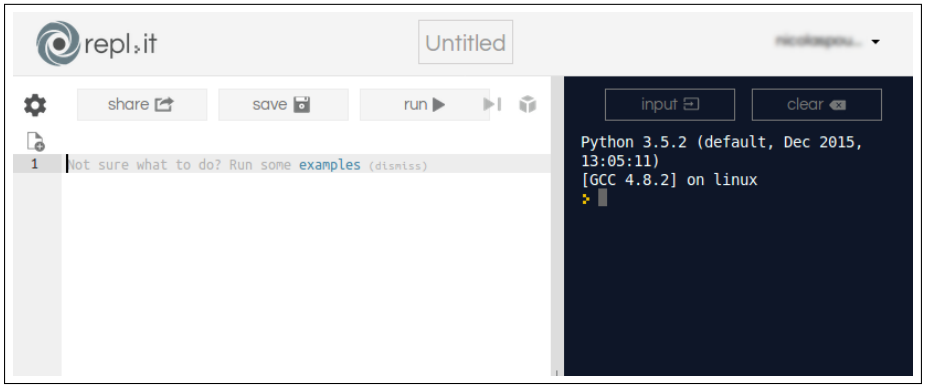

FIG. 1 : Repl.it en action

1 TL ;DR - L'essentiel de ce document en 15 minutes

Se lancer dans la programmation avec Python peut faire peur car les documentations qu'on trouve un peu partout proposent des choses intéressantes mais le niveau de départ n'est pas toujours adapté.

Un bon moyen de commencer avec Python est de s'affranchir des problèmes d'installation en utilisant un environnement en ligne – repl.it – https://repl.it/languages/python3 (FIG. 1) :

L'interface de programmation de repl.it est divisée en deux parties

- à gauche, sur fond clair, la fenêtre de script où nous allons pouvoir écrire des programmes complets qui pourront être sauvegardés plus tard (à condition de créer un compte)
- à droite, sur fond sombre, la console dans laquelle le code est exécuté comme sur une calculatrice,

Essayez vous-même de saisir les commandes suivantes dans la console (fenêtre de droite, sur fond sombre)

- $2 - 2 * 5$
- $1 / (0.5 - 1/2)$
- $5**2$
- $10 / 3$

C'est simple mais c'est limité, en effet les commandes suivantes renvoient des erreurs

- $\cos(0)$
- $\exp(1)$
- $\log(1)$

Pour aller plus loin et disposer de fonctions mathématiques sérieuses, il va falloir charger une bibliothèque nommée math en saisissant la commande from math import $*$. Les commandes ci-dessus fonctionnent alors comme on s'y attend.

Maintenant que nous avons vu les premières instructions dans la console, allons dans la fenêtre de script, à gauche pour écrire un programme permettant de calculer le périmètre ou l'aire d'un disque en fonction de son rayon.

Nous définissons deux fonctions aire et perim à l'aide du mot clé def terminé par le caractère : (deux points) obligatoire. La définition de chaque fonction se trouve dans le bloc d'instructions délimité par l'indentation, elle aussi obligatoire.

La constante pi est fournie par la librairie math, elle nous sera utile. Le symbole = est utilisé pour affecter une valeur à une variable.

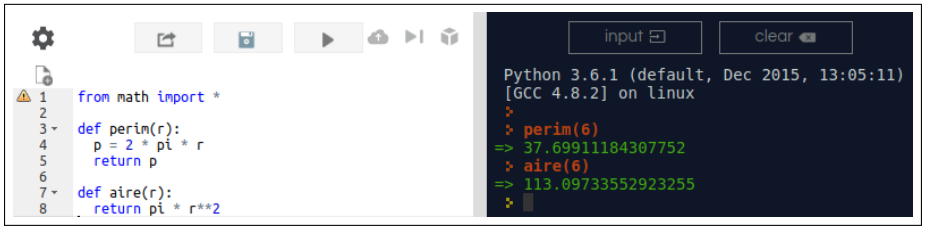

FIG. 2 : Aire et périmètre

```
from math import *

def perim(r):
  p = 2 * pi * r
  return p
```

On définit de même la fonction aire, ici sans utiliser de variable intermédiaire puisque ce n'est en réalité pas nécessaire. Remarquez la syntaxe permettant de calculer le carré de r : on écrit r**2.

```
def aire(r):
  return pi * r**2
```

Pour utiliser ce programme, il faut l'exécuter puis lancer un appel à chacune des fonctions, comme on le voit sur la copie d'écran suivante (FIG. 2) :

Puisque repl.it permet le partage, voici un lien permettant d'accéder à ce programme prêt à exécuter en cliquant sur le bouton *Run* en forme de triangle : https://repl.it/KuuY

Remarque : le partage est une fonctionnalité particulièrement intéressante pour une utilisation pédagogique de repl.it en classe. Nous en reparlerons dans le paragraphe 5.2 sur l'utilisation de Python en ligne.

Pour terminer cette initiation, voici un programme qui détermine les racines d'un polynôme de degré 2 à coefficients réels. Il y a ici une structure conditionnelle pour distinguer les cas selon le signe du discriminant.

Pour calculer les racines carrées, nous utiliserons la fonction sqrt (square root) fournie par la bibliothèque math. Les caractères suivant le symbole # sont des commentaires qui ne sont pas exécutés par le programme.

```
from math import *

def secondDeg(a,b,c):
  # Calcul du discriminant
  D = b**2 - 4*a*c

  # Deux cas se présentent :

  # soit Delta est strictement négatif
  if D < 0 :
    return D

  # soit il ne l'est pas
  else :
    return D, (-b - sqrt(D))/(2*a), (-b + sqrt(D))/(2*a)
```

Vous pouvez retrouver ce programme à l'adresse suivante : https://repl.it/KvvG (FIG. 3) :

Analysons les deux lignes suivantes

```
  if D < 0 :
    return D
```

FIG. 3 : Racines d'un polynôme du second degré

Le mot if est suivi de la condition D == 0 puis du signe " :" qui est indispensable et signifie en quelque sorte alors. Nous avons donc

si D est strictement négatif, alors.

La ligne suivante est décalée de deux espaces par rapport à la condition précédente. C'est obligatoire. Ce décalage constitue le bloc d'instructions qui sera exécuté lorsque la condition qui le précède est vraie.

si D est strictement négatif, alors renvoyer la valeur du discriminant.

Les lignes qui suivent permettent d'afficher les deux racines (éventuellement égales) de ce polynôme du second degré à coefficients réels.

Exercice : dans le cas où a est nul, le programme va afficher des erreurs ou un résultat faux. Modifier le programme pour éviter cela.

2 Deux exemples pour commencer

Les deux exemples ci-dessous ont en commun de mettre en oeuvre des algorithmes. Le premier représente un point de départ. En tant qu'astuce hors programme, il a été distribué "sous le manteau" à des élèves de 5e qui avaient besoin de savoir quand on peut cesser de chercher à simplifier des fractions. Le second souhaite montrer une autre extrémité. C'était pour de grands élèves une expérience scientifique et un défi programmatoire qui s'est étalé sur 6 heures.

2.1 Problème classique

Voici le début de la suite croissante des nombres premiers : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

Quel est le nombre premier suivant ? Quels sont les 5 suivants ?

Des collégiens, avec du papier, un crayon, un peu de temps et de méthode, sont capables de répondre en écrivant les entiers de 1 à 300 puis en biffant les multiples de 2, de 3, de 5, etc. C'est le *crible d'Eratosthène.*

Pourrions-nous automatiser cette recherche pour trouver les 10 000 premiers termes de cette suite ?

2.2 Expérience de simulation numérique

Une épidémie de grippe ravage une population. Une personne infectée a chaque jour p % de risques d'en mourir ; elle est contagieuse, et pendant 7 jours, chaque personne qui l'approche a q % de risques d'être contaminée. Passé ce délai, elle guérit.

Quelle est la probabilité qu'une personne prise au hasard survive à l'épidémie ?

Ce problème a été modélisé par des élèves (avec l'aide du prof !), un programme a tourné sur une machine et la probabilité cherchée a été approchée par des fréquences.

3 Algorithmique ou programmation ?

Les élèves, sans le savoir connaissent déjà beaucoup d'algorithmes, et dès le collège. Toutes les techniques opératoires en arithmétique, tous les programmes de construction géométrique (médiatrice, pentagone régulier) sont des algorithmes. Euclide est le seul algorithme qui est déclaré comme tel au collège, et ce n'est pas le plus simple.

Dans tous les cas, on a affaire à une suite finie d'opérations, exécutées dans un certain ordre. *Cette succession d'opérations forme un algorithme.*[1]

Le développement de l'informatique fait que les algorithmes, déjà omniprésents en mathématiques, ont aussi envahi la vie quotidienne : automates divers (pilotes d'avions, distributeurs…), cryptage, compression de données (vidéo), simulation numérique (météo), statistique, etc. Certains sont même secrets (*page ranking* de google ou monde de la finance) !

Profitant de la puissance de calcul des machines (exemple de l'épidémie de grippe ci-dessus), on peut automatiser certaines tâches, ou pousser des expériences mathématiques [2]. Aussi un algorithme est-il en général "implémenté" sur un ordinateur, comme on dit. C'est-à-dire qu'on a écrit l'algorithme en un langage interprétable par la machine. C'est *la programmation*. Cela suppose de respecter une forme précise et rigide, généralement en anglais et donc moins lisible.

L'algorithmique, elle, se distingue de la programmation en ce qu'elle vise à se débarrasser des questions de syntaxe.

Algorithme	**Programme**
• Peut fonctionner à la main	• Sur machine
• Expression libre	• Expression contrainte

En pratique, on ne se débarrasse pas de la syntaxe car on travaille beaucoup avec des machines. Et de toute façon, un algorithme reste un objet mathématique. On manipule donc des nombres, des données… [3]

4 Avec les élèves

4.1 Les attendus des programmes

Extrait de [@Eduscol2009]

> La sensibilisation de l'élève à la question de la "démarche algorithmique" pourra se faire en évitant toute technicité ou exposé systématique. On pourra sur ce thème consulter des publications réalisées dans le cadre des IREM. Les compétences suivantes pourront être identifiées et travaillées
>
> - comprendre et analyser un algorithme préexistant ;
> - modifier un algorithme pour obtenir un résultat particulier ;

1. Un algorithme permet de résoudre un problème, il donne un résultat. Ce résultat peut être la solution du problème qu'on s'est posé (Euclide détermine le pgcd), mais l'algorithme en lui-même a aussi un intérêt mathématique. C'est lui aussi la solution d'un problème. Par exemple, l'algorithme de Gauss prouve, *en la construisant*, l'existence d'une base orthogonale relativement à une forme quadratique donnée. Ici, le problème consiste à démontrer un résultat : l'existence de cette base. Construire la base orthogonale sur tous les exemples rencontrés ne donne pas pour autant une preuve générale. Gauss l'apporte en imaginant l'algorithme. Pour la conjecture de Goldbach *a contrario*, on attend toujours le contre-exemple ou l'algorithme de décomposition d'un pair en somme de nombres premiers.

2. Le théorème des quatre couleurs a été établi par une machine : il y avait des milliers de cas à vérifier.

3. Le problème, qu'on va répéter au long de ces pages, est que notre esprit ne suit pas du tout le même cours que les rouages d'un cerveau électronique. Pour suivre le chemin qu'on lui a tracé, sans chercher à savoir où elle va, la machine n'a besoin que d'une chose : qu'on lui dise quoi faire, sans exceptions. Au contraire, pour que nous comprenions et retenions, il nous faut de la hauteur, quitte à oublier les détails. C'est vrai des élèves. On examine plus bas les conséquences pédagogiques.

- analyser la situation : identifier les données d'entrée, de sortie, le traitement… ;
- mettre au point une solution algorithmique : comment écrire un algorithme en "langage courant" en respectant un code, identifier les boucles, les tests, des opérations d'écriture, d'affichage… ;
- valider la solution algorithmique par des traces d'exécution et des jeux d'essais simples ;
- adapter l'algorithme aux contraintes du langage de programmation : identifier si nécessaire la nature des variables… ;
- valider un programme simple.

Les programmes actuels ont fait des choix : - programmation impérative - pas de programmation orientée objet - uniquement des constructions itératives : pas de récursion

Lire https://fr.wikipedia.org/wiki/Programmation_informatique pour en savoir plus sur les paradigmes de programmation .

Plus récemment, le document d'accompagnement pour l'algorithmique et la programmation en seconde va dans le sens d'une clarification du cadre de travail : *Un langage de programmation simple d'usage est nécessaire pour l'écriture des programmes. Le choix du langage se fera parmi les langages interprétés, concis, largement répandus, et pouvant fonctionner dans une diversité d'environnements.* Python correspond très bien à cette description…

http://bit.ly/eduscol_algo_prog

4.2 Aborder l'algorithmique

Le programme des cycles 3 et 4 insistent désormais beaucoup sur l'algorithmique et la programmation notamment avec l'utilisation de Scratch. Le nombre d'élèves abordant le lycée sans aucune notion devrait donc aller en diminuant.

• **Partir des connaissances des élèves** : Les élèves connaissent déjà de nombreux algorithmes, dont certains sont simples. Ils peuvent servir d'introduction. Par exemple, le calcul de la puissance d'un nombre, développé ci-dessous.

• **Lire, tester et modifier des algorithmes** : Écrire un algorithme est un exercice difficile, introduisons-les donc déjà écrits.

- On peut les analyser en détail et les corriger (*que fait cet algorithme ? comment lui permettre d'aboutir dans tous les cas ?*).
- On peut chercher à les simplifier, les compléter, les accélérer ou en changer le comportement (*que se passe-t-il si on échange ces lignes ? comment faire la même chose avec un test de moins ?*).

Ce travail d'explication de texte permet d'aborder les subtilités propres à l'informatique en limitant la difficulté.

• **Algorithmes sur papier** : Faire tourner des algorithmes "à la main", sur papier, ne nécessite pas de connaissances particulières relatives aux machines, aux langages utilisés. Sur l'intérêt du papier pour s'imprégner d'un algorithme et le comprendre, voir la section suivante.

• **Initiations en ligne** : Pour une initiation aux concepts d'algorithmique et de programmation, de nombreux tutoriels et jeux en ligne sont apparus ces dernières années. En voici quelques exemples (FIGS. 4 et 5) :

4.3 Objectifs à viser

• **Enseigner la démarche algorithmique.** Il est artificiel et peu rentable de ne montrer que des algorithmes parfaits, déjà pleinement fonctionnels. Cela cache les étapes du processus qui ont mené à leur forme achevée, et qu'il est fructueux de montrer : d'abord une idée, puis une première formulation, enfin une rédaction rigoureuse.

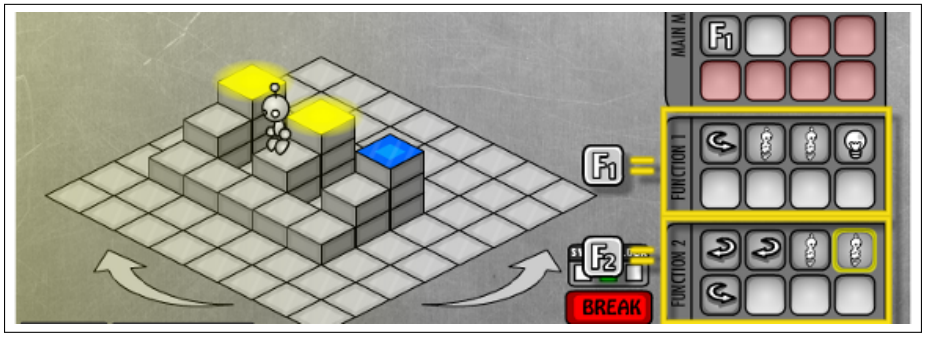

FIG. 4 : Light Bot 2 : http://armorgames.com/play/6061/light-bot-2

FIG. 5 : Hour Of Code : http://code.org/learn

Le professeur *fera prendre conscience aux élèves qu'eux aussi doivent procéder par étapes*, en les poussant à écrire dans un premier temps quelque chose de plus synthétique, quitte à être incomplet ou informel. C'est ce premier jet qui a le potentiel de devenir un algorithme au sens plein.

• **Enseigner le "débugguage"** Pour la bonne gestion des élèves en salle informatique, le professeur a intérêt à ce que ses élèves soient les plus autonomes possible. Ils doivent donc apprendre *à se tester*. Cela passe par le fastidieux travail consistant à contrôler sur des exemples que toutes les variables ont le comportement adéquat (afficher tous les calculs intermédiaires). Papier et crayon sont donc nécessaires à portée de la main, il faut insister.

• **Enseigner la rédaction** Un programme bien écrit est un programme qu'on comprend et qu'on peut relire des mois plus tard. Ainsi, les noms choisis doivent être explicites (*préférer* nbre_voyageurs *à* n3 *ou à* X) et la structure logique, l'enchaînement des instructions doivent être rendus visibles par une **indentation** correcte (voir le langage python). Par ailleurs, l'écriture de commentaires synthétiques mais explicites dans le programme devraient être exigés avec la même insistance que pour la rédaction en français dans les copies de mathématiques.

4.4 Évaluer sur l'algorithmique

Quoi : les compétences liées aux trois modalités fondamentales de l'activité en algorithmique

- analyser le fonctionnement ou le but d'un algorithme existant ;
- modifier un algorithme existant pour obtenir un résultat précis ;
- créer un algorithme en réponse à un problème donné.

Comment

- Oralement, par exemple dans des activités du type "épreuve pratique"

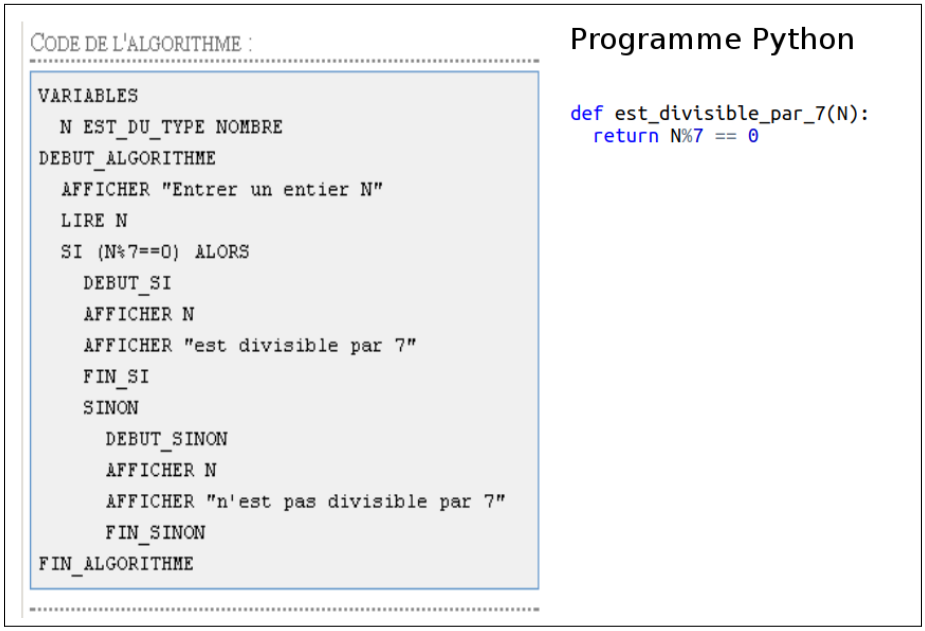

FIG. 6 : Algobox Vs Python

- Demander des algorithmes ou programmes en devoir maison
- Faire réaliser un projet personnel aux élèves par petits groupes

5 Python

Python est un langage qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées à chaque traitement. Il est particulièrement répandu dans le monde scientifique, et possède de nombreuses extensions destinées aux applications numériques. *Un programme python ne fonctionne que si l'utilisateur en a indenté le code, ce qui lui donne un intérêt pédagogique dans l'apprentissage de la qualité de la* `rédaction` *d'un programme.*

La syntaxe du langage Python est simple et très efficace. À titre d'exemple voici (FIG. 6) : deux programmes permettant de savoir si un nombre est ou non divisible par 7. Avec Algobox à gauche et avec une fonction Python à droite. On constate que

- les déclarations de variables ont disparu
- les entrées sorties (LIRE, AFFICHER) n'apparaissent plus
- les délimiteurs de blocs (DEBUT_..., FIN_...) sont inutiles

Cela permet de se concentrer sur ce qui est important en classe de Lycée : l'algorithme, c'est à dire la méthode de résolution du problème.

5.1 Installation sur un ordinateur

Le logiciel WinPython qui propose l'interpréteur Python ainsi qu'une interface graphique est téléchargeable pour Windows à l'adresse suivante `http://winpython.github.io/`

Télécharger la dernière version (WinPython 3.x.x.x) et suivre les instructions d'installation.

5.2 Utilisation en ligne

Il existe désormais de nombreux environnements en ligne pour le développement en Python. Ils permettent de disposer dans un simple navigateur web d'un éditeur et d'un interpréteur Python prêts à l'emploi et donc sans aucune installation.

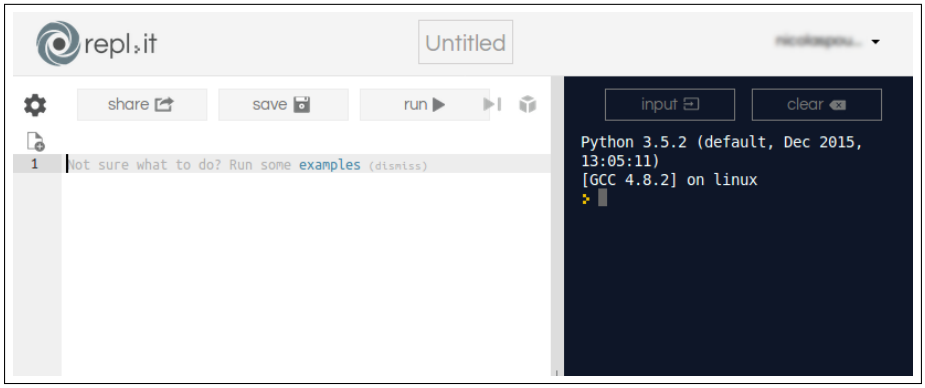

FIG. 7 : repl.it en action

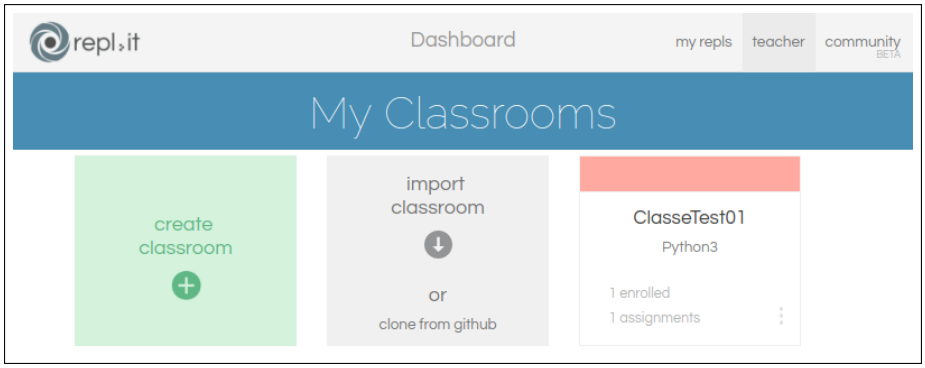

FIG. 8 : Gérer ses classes

Autre avantage, les scripts sont enregistrés en ligne (sur un cloud) et peuvent être partagés entre utilisateurs.

Parmi ces environnements, repl.it se distingue par la possibilité qu'il offre de partager des travaux avec ses élèves réunis dans des classes.

L'interface de programmation de repl.it est divisée en deux parties (FIG. 7)

- à gauche, sur fond sombre, la fenêtre de script où nous allons pouvoir écrire des programmes complets qui pourront être sauvegardés plus tard.
- à droite, sur fond clair, la console dans laquelle le code est exécuté comme sur une calculatrice.

Le mode enseignant permet de créer ses classes, d'y accueillir ses élèves afin de leur proposer des exercices à compléter pour ensuite pouvoir les corriger. Lorsqu'on dispose d'un compte sur repl.it et qu'on est connecté, on peut cliquer sur l'onglet *teacher* puis créer une classe (FIG. 8).

À l'intérieur de la classe, on peut inviter des étudiants en cliquant sur le bouton *invite more*. Le plus simple est de leur communiquer l'adresse du lien d'invitation directe *Direct invitation link*. Ce lien va les diriger vers la classe (en les faisant éventuellement passer par le formulaire de création de compte la première fois) (FIG. 9).

Bien sûr, avant d'inviter les élèves, on aura créé devoir à réaliser (*assignement*) (FIG. 10).

Les élèves rejoignant la classe y verront ce devoir et pourront enregistrer leurs travaux. L'ensemble des contributions des élèves sera alors automatiquement visible pour le professeur qui pourra vérifier et annoter les productions individuellement.

Voir aussi

- https://pythonanywhere.com
- https://trinket.io
- http://www.skulpt.org/
- https://www.python.org/
- https://www.tutorialspoint.com/codingground.htm

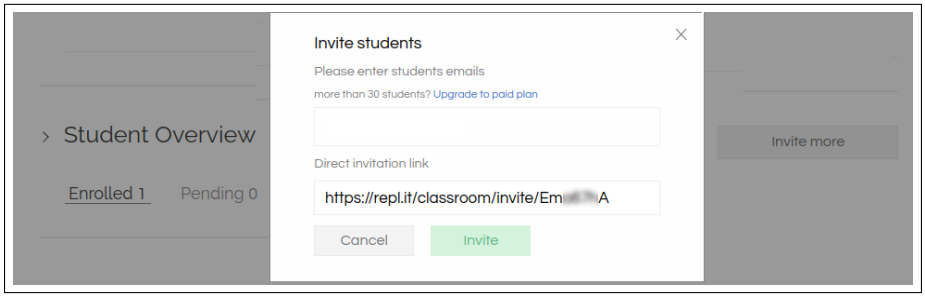

FIG. 9 : Inviter ses élèves dans sa classe

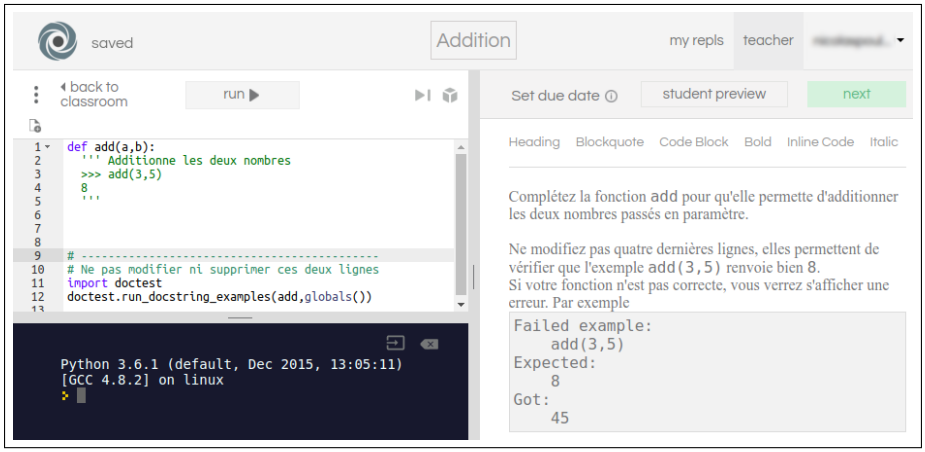

FIG. 10 : Créer un devoir

5.3 Sur un appareil mobile

Le développement des outils mobiles en classe (tablettes et smartphones) permet de détacher de plus en plus facilement l'usage de Python de la salle informatique. Il existe des outils pour faire de Python un outil simple et quotidien de modélisation, de vérification ou de calcul en classe.

5.3.1 Vidéo-projeter l'écran d'un téléphone travaillant sur Python

Il existe des compilateurs Python autonomes permettant de travailler sans connexion pour les deux grandes plateformes QPython3 (gratuit pour Android) et Pythonista (environ 10€ pour iOS).

Une fois l'application installée, on peut programmer (FIG. 11) :

En appuyant sur le bouton d'exécution (icône triangulaire sur la fenêtre de gauche), on bascule de l'éditeur à la console où le résultat s'affiche.

Pour montrer l'écran d'un smartphone au tableau grâce au vidéo-projecteur, on peut utiliser un logiciel de partage d'écran. Sous Android, on dispose du très efficace LiveScreen (ou Screen Stream Over Http, gratuits tous les deux). Sous iOS le logiciel LonelyScreen (environ 15€ par an) est un outil similaire.

Sur l'ordinateur vidéo-projeté, il suffit d'ouvrir un navigateur web à l'adresse indiquée par LiveScreen sur le smartphone connecté au Wifi. L'écran du smartphone s'affiche alors en temps réel dans le navigateur.

5.3.2 Afficher des scripts créés sur un téléphone

Si les élèves disposent d'une connexion à l'internet sur leur mobile, on peut utiliser une application multi-plateforme nommée SoloLearn.(FIG. 12) :

Fonctionnalité intéressante de SoloLearn : le partage qui permet l'échange et la construction collaborative dans la classe. Les élèves disposant d'un compte peuvent créer leurs programmes, les enregistrer et les

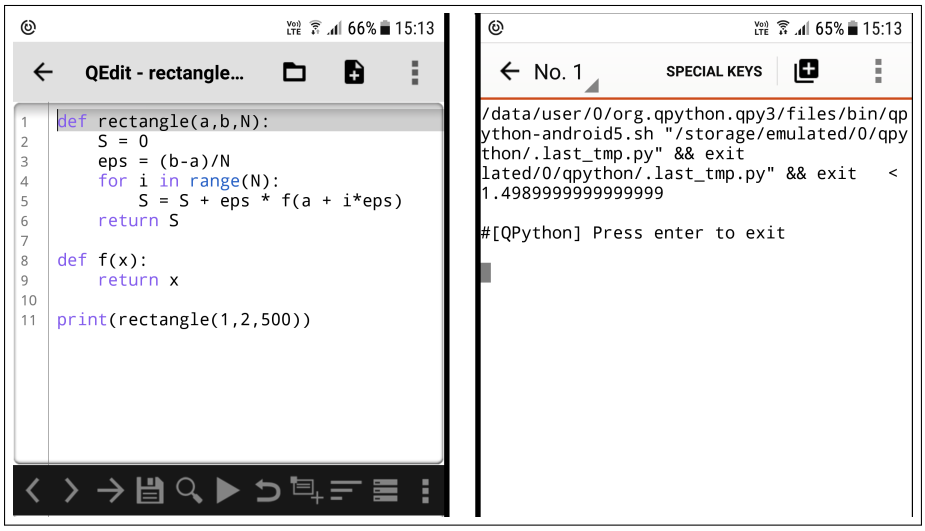

FIG. 11 : QPython3 en action sur Android

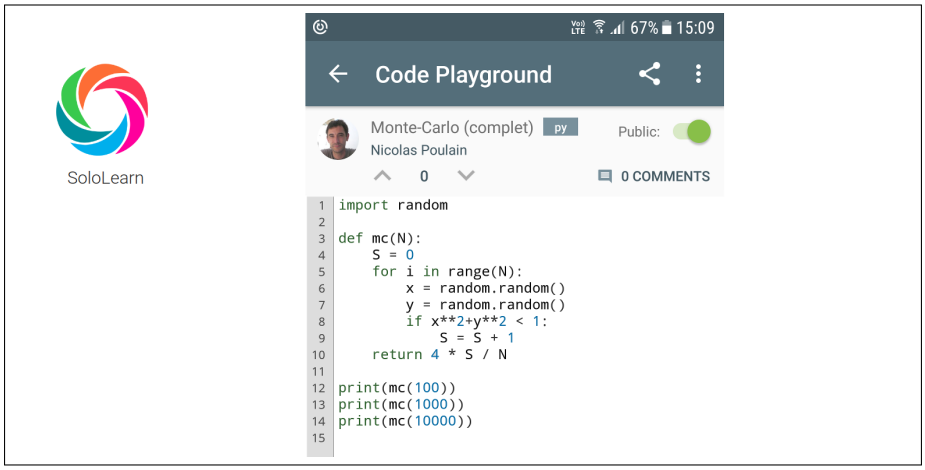

FIG. 12 : Sololearn pour Android et iOS

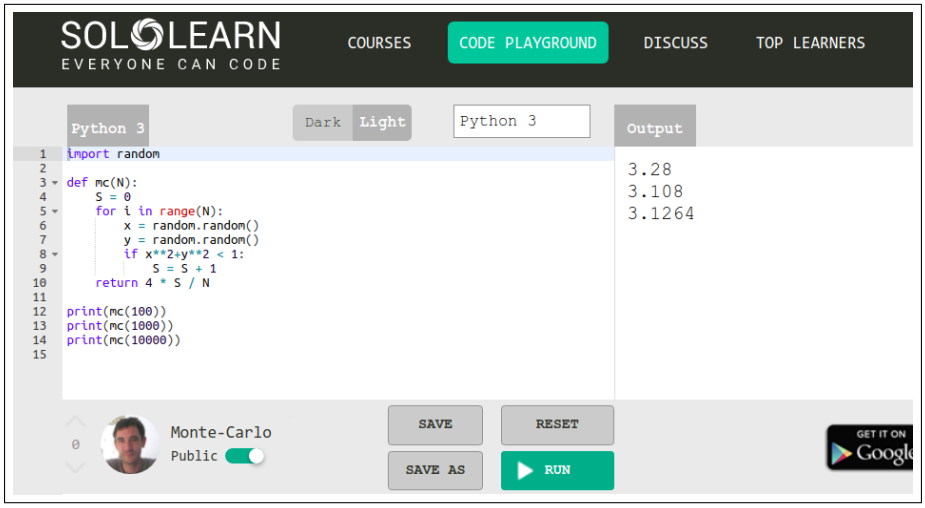

FIG. 13 : Le site sololearn.com sur ordinateur vidéo-projeté

partager au moyen du petit interrupteur “Public”. Afficher le programme d’un élève au vidéo-projecteur est alors très simple puisqu’il suffit de se rendre sur le site www.sololearn.com/Codes/ pour retrouver la production de l’élève, l’afficher et l’exécuter devant la classe.

6 Ressources pour se former sur Python

- Le document d’accompagnement pour l’algorithmique et la programmation en seconde http://bit.ly/eduscol_algo_prog propose une gamme complète de problèmes, du plus élémentaire au plus sophistiqué dans la gamme pouvant être proposée au Lycée.
- Cette documentation, elle-même, fournit un certain nombre de clés au travers des divers exemples commentés.
- L’académie de Martinque propose au téléchargement un document à la fois très complet et très progressif. Son titre est *Algorithmes au Lycée : Python ?* par Vincent Tolleron.
- Enfin, une ressource en ligne classique : le site python.lycee.free.fr propose une formation complète.

Remarque : ces deux dernières ressources proposent des exemples qui ne sont pas toujours dans l’esprit du programme. On pourra lire à ce propos l’article de Yann Salmon : https://www.yannsalmon.fr/les-fonctions-informatiques-dans-le-nouveau-programme-de-seconde/

7 Découverte de Python

Les variables sont une des bases de la programmation informatique. Il s’agit tout simplement de valeurs numériques stockées en mémoires dont on se sert au fur et à mesure de l’execution du programme.

La programmation impérative est la plus répandue (Fortran, Pascal, C,...) et la plus proche de la machine réelle. Son principe est de modifier à chaque instant l’état de la mémoire via les affectations. Historiquement, son principe a été initié par John VON NEUMANN et Alan TURING.

Par exemple, lorsqu’on entre avec Python

```
>>> a = 2
```

cela signifie que la case mémoire portant le petit fanion a contient à cet instant la valeur 2. On peut changer le contenu de cette case mémoire un peu plus tard

```
>>> a = 4
```

et cette fois la case a contient la valeur 4.

La ligne suivante additionne 5 à la variable a pour créer la variable b qui contiendra donc la valeur 9.

```
>>> b = a + 5
```

Les variables sont probablement la première structure d'un algorithme à étudier avec les élèves.

Les enjeux :

La modélisation Le choix des variables, des nombres qui traduisent mathématiquement le problème, sont la première étape de son traitement algorithmique. Les comptes facebook de nos élèves ne sont qu'un ensemble de paramètres et de bits.

La distinction entrées/sorties Faire l'analogie avec antécédent/image et paramètre/valeur. Quand on pose V=f(a,b), il n'est pas question de définir ni a ni b. On s'appuie au contraire dessus pour construire V.

Les difficultés de vocabulaire Une variable peut ne jamais varier et être constante, ce n'est qu'un symbole affecté d'une valeur. D'ailleurs, *dans de nombreux langages, le signe d'égalité ne représente pas l'égalité mathématique mais bien l'affectation.*

L'aspect séquentiel d'un algorithme et ses conséquences sur les valeurs prises par les variables.

Observons la suite d'instructions python suivante

```
>>> a = 1
>>> b = 2
>>> b = a - b
>>> a = a - b
```

- Si l'on regarde les deux dernières lignes, elles affectent apparemment la même valeur (a-b) à la variable a et à la variable b et pourtant c'est faux car les variables a et b ne contiennent pas les mêmes valeurs selon le moment où elles sont appelées.
- Si l'on change l'ordre d'une des deux dernières lignes, les valeurs présentes dans les cases mémoires nommées a et b vont être différentes.

7.1 Lire, comprendre et modifier un algorithme

1. Quel est le résultat de ce programme ?

   ```
   A = 1
   A = 5 + A
   ```

   La variable A contient la valeur 5 ? la valeur 1 ? la valeur 6 ?

2. *Variables liées.* Les élèves en ont déjà rencontré s'ils ont fait un peu de tableur :

   ```
   A = 10
   B = 2 * A
   C = B + 1
   ```

   - Que valent B,C ?
   - Que se passe-t-il pour les valeurs de B et C si on modifie la valeur de A dans la première ligne en écrivant A = 1 ? (comparer avec l'absence d'effet sur B,C si on ajoute une ligne A = 3 à la fin)

3. Comparer les valeurs de la variable s dans les deux programmes suivants

   ```
   a = 0; b = 1;
   a = b;
   s = a + b
   ```

et

```
a = 0; b = 1;
b = a;
s = a + b
```

4. Modifier l'algorithme Python suivant afin qu'il donne exactement les mêmes résultats (quelque soit la valeur de x) mais sans utiliser la variable y

```
def f(x):
  y = x-2
  z = -3*y-4
  return z
```

5. *De l'usage des variables.* Qu'on puisse affecter et réaffecter à volonté une variable offre des possibilités nouvelles.

   Comparer les programmes équivalents qui codent l'algorithme ci-dessous, extrait du DNB 2008 :
   - Choisir un nombre
   - Calculer le carré de ce nombre
   - Multiplier par 10
   - Ajouter 25.
   - Écrire le résultat

N,C,M,S : nombres		N : nombre
lire N		lire N
C=N^2		N=N^2
P=C+10	et	N=N+10
S=P+25		N=N+25
Afficher S		Afficher N

7.2 Un algorithme de tri

Tiré de [@Cordes]. L'algorithme ci-dessous est censé ranger par ordre croissant trois valeurs a, b, c. L'auteur n'a pas eu peur de donner à étudier à ses lycéens cet algorithme qui est un tri bulle[4] (défectueux). Pour ne pas les noyer, il a réduit à 3 le nombre de valeurs à ordonner et il a rédigé l'algorithme en langage naturel. On travaille donc sur papier, ce qui permet d'*échanger les valeurs* (pas si simple sur une machine).

Algorithme

```
Si a>b échanger a et b
Si b>c échanger b et c
Si a>c échanger a et c
```

L'analyse et la modification sont possibles en testant sur des exemples les 6 configurations de départ $a \leq b \leq c$, $a \leq c \leq b$, $b \leq a \leq c$, etc.

Tri bulle défectueux	**Tri bulle correct**	**Commentaires**
Si $a > b$ échanger a et b	Si $a > b$ échanger a et b	donc $b \geq a$
Si $b > c$ échanger b et c	Si $b > c$ échanger b et c	donc $c \geq b, a$
Si $a > c$ échanger a et c	Si $a > b$ échanger a et b	donc $c \geq b \geq a$

4. Ce tri est ainsi appelé car les grandes valeurs remontent la liste comme des bulles de champagne, dit-on.

Une fois l'algorithme corrigé, on pourrait demander d'en écrire un étendu qui ordonne quatre valeurs. Cela suppose d'avoir bien compris cet algorithme[5].

7.3 Une fonction pour convertir des devises

Un convertisseur Euro -> Dollar (taux de change à 1.29) constitue un premier algorithme simple ne mettant oeuvre que des déclarations et affectations de variables.

Prérequis Aucun

Principe Une somme (en euros) est donnée, la somme correspondante en dollars est calculée.

Intérêt Algorithme simple qui permet aux élèves de découvrir l'environnement de programmation Python qui sera utilisé pendant l'année.

On pourra demander aux élèves de s'inspirer de cet algorithme pour en concevoir un nouveau capable à partir d'un rayon donné, d'afficher une valeur approchée du périmètre et de l'aire d'un disque.

```
1 def eurodollar(eur):
2   """ Retourne la somme dol (en dollars) correspondant a la somme eur (en euros) """
3   dol = 1.29 * eur
4   return dol
```

Syntaxe commentée Le mot clé def permet de déclarer une fonction : ici eurodollar. La fonction eurodollar prend un unique paramètre : eur, qu'on va convertir en dollars dans la ligne suivante. Cette nouvelle valeur est stockée dans la variable dol. Une fonction se doit de renvoyer une valeur : le mot clé return qui met fin à l'exécution de la fonction va s'en charger.

Les lignes commençant par le caractère # ou qui sont entourées de """ sont des commentaires et ne sont pas exécutées. Commenter ses programmes est une excellente habitude que les élèves devraient prendre dés le début.

Après avoir lancé l'exécution du programme, rien ne se passe mais la fonction eurodollar est désormais prête à être utilisée. Pour cela, dans la console, on exécute les commandes suivantes :

Résultat :

```
>>> eurodollar(100)
> 129
>>> eurodollar(12.5)
> 16.125
```

7.4 Quatre paramètres pour l'équation d'une droite (épisode 1)

Étant donnés deux points $A(x_A, y_A)$ et $B(x_B, y_B)$, donner une équation de la droite passant par ces deux points.

Prérequis Affectation de variable

Principe Commençons par une première version simple qui va progresser dans un deuxième temps. L'énoncé est volontairement vague.

On peut chercher à déterminer les coefficients m et p de l'équation $y = mx + p$ de la droite passant par deux points donnés.

5. Autant il est évident d'échanger des termes qui ne se présenteraient pas dans l'ordre, autant il reste un mystère : combien de temps faut-il poursuivre les échanges avant d'avoir rangé tous les termes ?

Intérêt Ici c'est la modélisation du programme qui représente la première difficulté. Notamment du point de vue mathématique. Les élèves doivent commencer par travailler sur papier pour déterminer les variables nécessaires puis concevoir les étapes de calcul permettant l'affichage du résultat.

Les élèves seront amenés à faire fonctionner l'algorithme. L'occasion sera alors donnée de parler des *tests* c'est à dire des vérifications par exécution. Quelle est la robustesse d'un tel programme ? Que se passe-t-il si la droite est verticale, ou si les deux points sont confondus ?

L'autre intérêt est dans la différentiation qu'il permet. Si on peut se contenter de cette version, on pourra chercher à implémenter les cas particuliers évoqués ci-dessus. Ces approfondissement sont l'occasion de revenir au problème, de le réinvestir.

Première question : combien de variables utiliser ?

Faisons la liste.

- x_A, y_A pour l'abscisse et l'ordonnée du point A.
- x_B, y_B pour l'abscisse et l'ordonnée du point B.
- m pour le coefficient directeur.
- p pour l'ordonnée à l'origine.

```
def eq_reduite(xa,ya,xb,yb):
  """ Calcule les coefficients m et p de l'équation réduite
  de la droite passant par (xa,ya) et (xb,yb)
  """
  m = (yb-ya) / (xb-xa)
  p = ya - m*xa
  return m,p
```

Résultat :

```
>>> eq_reduite(1,2,2,4)
> (2,1)
```

Syntaxe commentée Une remarque de taille : nous venons de définir une fonction qui prend quatre arguments. De plus, elle renvoie un couple de valeurs. L'air de rien, on a construit une fonction dont la nature n'est pas du tout dans les habitudes d'un élève de lycée puisqu'il s'agit d'une fonction définie sur un sous-ensemble de $\mathbb{R}^4$ à valeurs dans $\mathbb{R}^2$.

7.5 Coût de transport

Pour un déplacement, un club sportif s'adresse à une entreprise de transport qui dispose de 4 bus de 50 places loué chacun 800 euros. Faire afficher le prix moyen par voyageur suivant le nombre de supporters.

Prérequis Affectation de variable

Principe Le problème est assez simple à résoudre pour peu qu'on sache utiliser la fonction partie entière floor. Pour un nombre N de supporters, on trouve le nombre de bus à réserver $B = \text{floor}\left(\frac{N-1}{50}\right)+1$ puis le prix $P = \frac{800B}{N}$.

Intérêt Ici encore, la modélisation mathématique donne à réfléchir. On pourra aussi s'interroger sur les variables à utiliser et leur nom afin de faciliter la lecture du code. On complexifiera plus tard avec l'exercice suivant : faire afficher toutes les possibilités qui donnent un prix moyen inférieur à 20 euros.

```
import math

def prix_bus(N):
  """ Calcule le prix pour N supporters """
  nb_bus = math.floor((N-1)/50)+1
  prix = (nb_bus*800) / N
  return prix
```

Syntaxe commentée On fait ici appel à une nouvelle commande : import math, permet de charger la bibliothèque math de Python . Elle nous donnera accès aux fonctions usuelles comme floor pour la partie entière, sqrt pour la racine carrée. On trouve parfois la syntaxe from math import * qui permet d'écrire directement floor plutôt que math.floor.

Voir http://docs.python.org/2/library/math.html pour l'ensemble des possibilités de la bibliothèque math.

8 Des blocs indentés : la particularité de Python

Souvent les problèmes nécessitent l'étude de plusieurs situations qui ne peuvent pas être traitées par les séquences d'actions simples. Puisqu'on a plusieurs situations, et qu'avant l'exécution, on ne sait pas à quel cas de figure on aura à exécuter, dans l'algorithme on doit prévoir tous les cas possibles.

Ce sont les *structures conditionnelles* qui le permettent, en se basant sur l'évaluation d'une expression logique dont le résultat est VRAI ou FAUX

Remarque : Un bloc d'instructions est une suite d'instructions dont l'exécution est solidaire. L'utilisation de structures conditionnelles conduit à la création de blocs d'instruction.

L'indentation (décalage vers la droite) permet de bien faire apparaître ces blocs. Dans la plupart des langages, l'indentation n'est pas obligatoire. Sur les calculatrices, elle n'existe même pas : on ne voit alors pas toujours bien les blocs d'instructions et c'est la source de nombreuses erreurs.

Lorsqu'on écrit un programme, il faut toujours avoir à l'esprit qu'il sera relu et qu'il *DOIT* être facilement lisible. L'indentation du code doit être une habitude rapidement prise, cela tombe bien, c'est une obligation avec Python puisque c'est cette indentation qui définit les blocs.

8.1 Tests et blocs : Lire, comprendre et modifier un algorithme

1. Déterminer quelles sont les valeurs des variables a et b à la fin de l'exécution du programme suivant

```
a = 1
b = 3
if a>0:
  a = a+1
if b>4:
  b = b-1
else:
  b = b+1
```

Réponse : a=2 et b=4 car seule l'affectation b = b−1 n'est pas exécutée.

2. Déterminer quelles sont les valeurs des variables a et b à la fin de l'exécution du programme suivant

```
a = 1
b = 1
if a==b:
  a = a-1
else:
  b = b+1
b = b-1
```

Réponse : a=0 et b=0 car seule l'affectation b = b+1 n'est pas exécutée.

8.2 `if...else` : Valeur absolue d'un nombre

Prérequis Affectation
Principe La valeur absolue d'un nombre réel x est x si $x \geq 0$ et $-x$ si $x < 0$.
Intérêt Algorithme simple qui permet aux élèves de construire un algorithme en utilisant soit deux instructions SI soit une instruction SI−SINON.

```
def val_abs(x):
  """ Renvoie la valeur absolue du nombre x. """
  if x<0:
    return -x
  else:
    return x
```

Résultat :

```
>>> val_abs(-12)
> 12
>>> val_abs(0.34)
> 0.34
```

Syntaxe commentée Le mot-clé if évalue une expression logique et exécute un bloc d'instructions si l'expression est vraie. Le mot-clé optionnel else permet d'exécuter en alternative des blocs d'instructions.

Dans le langage python l'annonce du début d'un bloc s'effectue par ":" et c'est l'indentation qui définit les blocs[6], une réduction de l'indentation marquant alors la fin du bloc courant.

8.3 `assert` pour l'équation d'une droite (épisode 2)

Prérequis Affectation de variable et structures conditionnelles

Principe Cette fois, on améliore le programme pour qu'il prenne en compte les cas particuliers.

Intérêt On a vu plus haut (voir épisode 1) ce programme qu'on va ici améliorer. C'est une bonne idée de donner une suite à un travail déjà commencé. On gagne du temps avec un environnement familier et cela montre que le travail de programmation est évolutif et modulaire.

On peut se contenter d'afficher "Erreur" lorsque les abscisses des deux points sont égales mais on peut aller plus loin et proposer

- si les deux abscisses sont différentes, l'équation du type $y = mx + p$
- si les deux abscisses sont égales mais les deux ordonnées sont différentes, la droite est verticale
- Le cas où les deux points sont confondus, sera à traiter comme encore un autre cas particulier où aucune équation ne peut être proposée.

```
def eq_reduite(xa,ya,xb,yb):
  """ Equation réduite de la droite passant par (xa,ya) et (xb,yb)
  Renvoie une erreur si les points sont confondus
  sinon Renvoie xa si la droite est verticale
  sinon Renvoie m,p coefficients de l'équation réduite """
  assert(xa!=xb and ya!=yb)
  if (xa==xb and ya!=yb):
    return(xa)
  if (xa!=xb and ya!=yb):
    m = (yb-ya)/(xb-xa)
    p = ya-m*xa
    return m,p
```

6. les indentations sont donc **obligatoires** et éventuellement imbriquées.

Syntaxe commentée

- La commande assert vérifie que les conditions sont bien vérifiées. Ici, le programme s'arrêtera si x est nul, pour éviter une division par 0.
- On peut écrire ce programme de différentes façons. La manière ci-dessus est sans doute la plus lisible. Cependant, on peut lui reprocher plusieurs redondances dans les tests. Ci dessous voici une autre écriture, plus rationnelle mais moins lisible, particulièrement sur la question de l'absence de test après le return(xa)

```
def eq_reduite(xa,ya,xb,yb):
  """ Equation réduite de la droite passant par (xa,ya) et (xb,yb)
  Renvoie une erreur si les points sont confondus
  sinon Renvoie xa si la droite est verticale
  sinon Renvoie m,p coefficients de l'équation réduite """
  assert(xa!=xb and ya!=yb)
  if (xa==xb):
    return(xa)
  m = (yb-ya)/(xb-xa)
  p = ya-m*xa
  return m,p
```

8.4 Tests de divisibilité avec partie entière ou division euclidienne

Construire un algorithme permettant de savoir si un nombre entier x est ou non divisible par un entier naturel

Prérequis Structures conditionnelles

Principe Deux méthodes (au moins) sont envisageables

- en utilisant la division euclidienne, on teste si le reste est nul
- sans utiliser la division euclidienne, on teste si le x/y est entier (comment ?)

Intérêt L'algorithme est assez simple à construire mais étant donné que plusieurs méthodes sont possibles, les élèves pourront confronter leurs résultats. Par ailleurs on pourra ajouter des structure conditionnelles afin de vérifier que les valeurs saisies sont bien entières, non nulle etc.

```
import math

def x_divise_y(x,y):
  """ Teste si x divise ou pas y. """
  assert (x!=0)
  if math.floor(y/x) == y/x
    return True
  else:
    return False
```

Autre méthode, plus courte et donc plus difficile à comprendre...

```
def x_divise_y(x,y):
  """ Teste si x divise ou pas y. """
  assert (x!=0)
  return y%x == 0 # y%x signifie y modulo x
```

Résultat :

```
>>> x_divise_y(2,5)
> False
>>> x_divise_y(2,6)
> True
```

Syntaxe commentée On fait ici appel à de nouvelles commandes

- Dans la première version, la fonction floor de la bibliothèque math de python chargée en première ligne renvoie la partie entière.
- Dans la deuxième version, on utilise la fonction modulo % qui renvoie le reste de la division euclidienne.
- pour tester si deux nombres sont égaux, on utilise l'opérateur == (à ne pas confondre avec l'opérateur = d'affectation). Sa négation s'écrit !=.

8.5 `if...elif...else` : Solutions d'un polynôme de degré 2 à coefficients réels

Résoudre dans $\mathbb{R}$ l'équation du second degré $ax^2+bx+c=0$.

Prérequis Affectation de variable, structures conditionnelles

Principe On calcule le discriminant $\Delta = b^2-4ac$ puis on résoud le problème selon le signe de Δ.

Intérêt Intérêt mathématique évident et travail sur les structure si−sinonsi−sinon

Dans un second temps, on pourra attirer l'attention des élèves sur les erreurs de calcul liées à la représentation de nombres réels, ensemble fini (!) pour un ordinateur.

Exemple : le discriminant de l'équation $x^2+0.1x+0.0025=0$ est nul, mais vérifions cela avec Python.

```
>>> 0.1**2 - 4*1*0.0025
1.734723475976807e-18
```

La structure conditionnelle qui va conduire aux trois cas suivant la valeur de Δ devrait donc tenir compte de cette imprécision. Il faudra par exemple estimer que Δ est nul si $|\Delta| < 1.0e-15$.

```
from math import *

def secondDeg(a,b,c):
  """ Calcule le discriminant puis les
  éventuelles racines du polynome ax^2+bx+c
  """
  delta = b**2 - 4*a*c
  if delta<0 :
    return delta
  elif delta==0 :
    return delta, -b/(2*a)
  else :
    return delta, (-b - math.sqrt(delta))/(2*a), (-b + math.sqrt(delta))/(2*a)
```

9 Tracés et graphismes avec matplotlib

Pour le tracé de graphes de fonctions, le module matplotlib est classique.

9.1 Courbes représentatives de fonctions

Dans l'exemple suivant, on construit une courbe avec plt.plot(X, Y, options...) qui va placer les points construits à partir des abscisses de la liste X et des ordonnées de la liste Y (de même taille) et les relier selon les options choisies.

Le résultat du programme suivant est présenté sur la FIG. 14

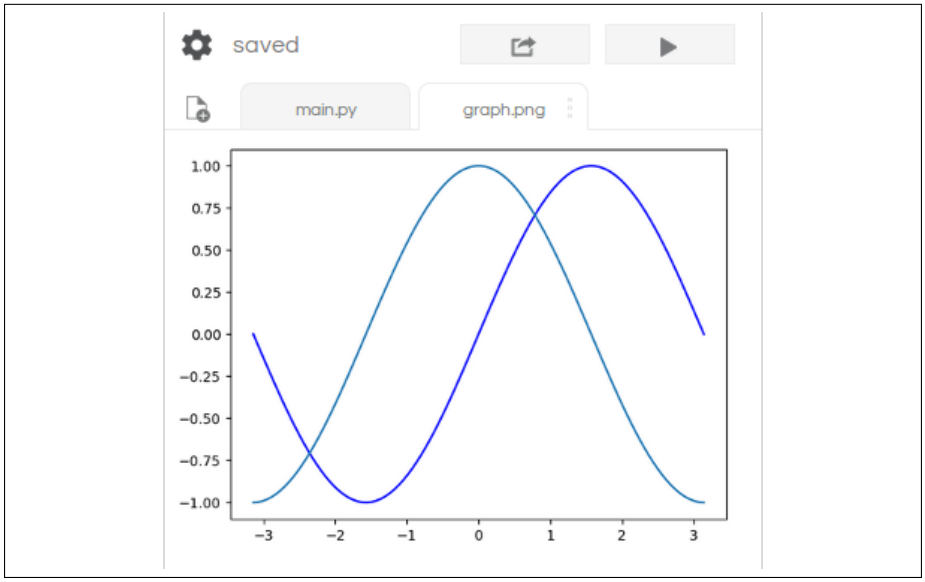

FIG. 14 : Graphes de fonctions avec matplotlib

```
from matplotlib.pyplot import *
from numpy import *

X = linspace(-pi, pi, 100) # une liste de 100 valeurs entre -pi et pi
Y = sin(X)
plot(X, Y, color="blue", linewidth=1.5, linestyle="-")
plot(X, cos(X))
savefig("graph")
```

9.2 Nuages de points

Pour dessiner un nuage de points, on procède de la même manière : Une liste d'abscisses et une liste de même taille contenant les ordonnées correspondantes.

Le résultat du programme suivant est présenté sur la FIG. 15.

```
from matplotlib.pyplot import *
from numpy import *
from random import *

X, Y = [], []
for i in range(2000):
  X = X + [random()]
  Y = Y + [random()**2]

plot(X, Y, 'ro')
savefig("distribution")
```

9.3 Diagrammes en bâtons

Dans le programme suivant, on lance deux dés à 6 faces et on enregsistre dans la liste L, l'effectif pour chaque valeur obtenue de 0 à 12 (évidemment 0 et 1 restent à 0 mais le programme est plus simple à écrire ainsi).

Le résultat du programme suivant est présenté sur la FIG. 16.

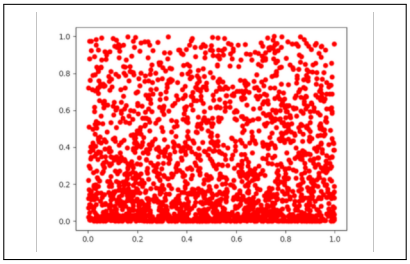

FIG. 15 : Nuage de points avec avec matplotlib

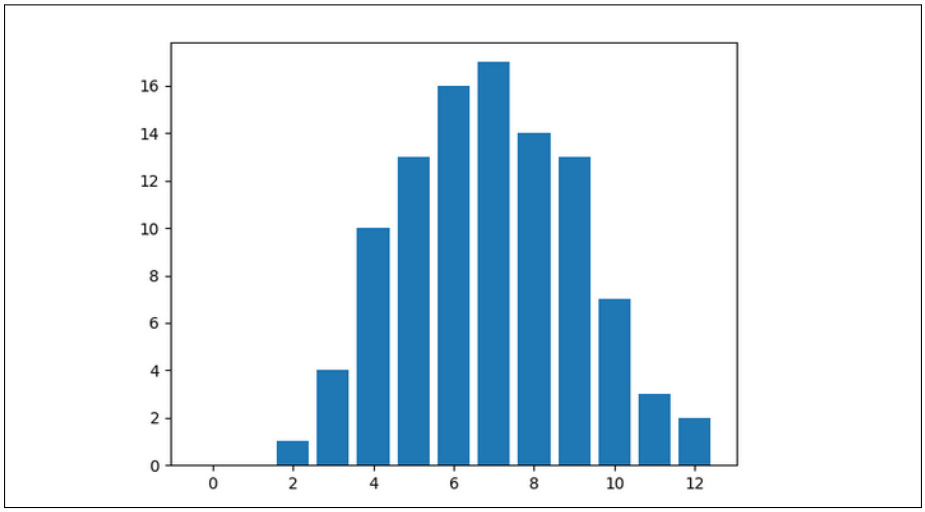

FIG. 16 : Diagramme en bâtons avec avec matplotlib

```
from matplotlib.pyplot import *
from numpy import *
from random import *

X = [0,1,2,3,4,5,6,7,8,9,10,11,12]
L = [0,0,0,0,0,0,0,0,0,0,0 ,0 ,0 ]
for i in range(100):
  double_de=randint(1,6)+randint(1,6)
  L[double_de]=L[double_de]+1

bar(X,L)
savefig("diagramme")
```

10 Vers l'infini et au delà avec les structures répétitives

Dans les problèmes quotidiens, on ne traite pas uniquement des séquences d'actions, sous ou sans conditions, on est fréquemment obligé d'exécuter un traitement (séquence d'actions), plusieurs fois. En effet, pour saisir les N notes d'un étudiant et calculer sa moyenne, on est amené à saisir N variables, puis faire la somme et ensuite diviser la somme par N. Cette solution nécessite la réservation de l'espace par la déclaration des variables, et une série de séquences d'écriture/lecture. Ce problème est résolu à l'aide des *structures répétitives*. Celles ci permettent de donner un ordre de répétition d'une action ou d'une séquence d'actions une ou plusieurs fois.

10.1 `range`, le compagnon de `for` : Méthode de Monte-Carlo

Une méthode statistique d'approximation du nombre π.

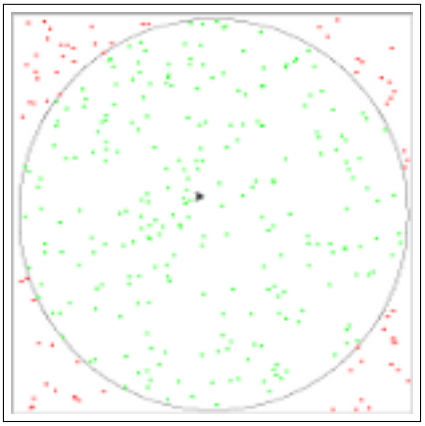

FIG. 17 : Monte-Carlo dans Python

Prérequis Affectation de variable, conditions et boucle for

Principe On considère un carré C de côté 1 et le quart de disque D inscrit dans ce carré. Pour une goutte tombant dans le carré, la probabilité P de tomber dans le disque est $P = \frac{\text{aire}(D)}{\text{aire}(C)} = \frac{\pi}{4}$.

En tirant au hasard N points de coordonnées (x, y) avec x et y compris entre 0 et 1, on peut compter le nombre S de points tels que $x^2 + y^2 < 1$. Le nombre $\frac{S}{N}$ sera une approximation statistique de la probabilité P. On pourra en déduire une approximation de π en remarquant que $\frac{S}{N} \simeq \frac{\pi}{4}$ et donc $\pi \simeq 4\frac{S}{N}$.

Intérêt Plusieurs prolongements sont possibles : proposer un affichage graphique, améliorer la vitesse d'exécution du programme en diminuant le nombre de calculs et de variables mis en jeu. Chronométrer et commenter.

```
from random import *

def monte_calrlo(N)
  """ Calcule une approximation de pi a partir de N tirs aleatoires. """
  S = 0
  for i in range(N):
    x = random.random()
    y = random.random()
    if x**2+y**2<1:
      S = S+1
  return 4*S/N
```

Syntaxe commentée

- Dans la boucle for de l'exemple ci-dessus, la variable i va prendre ses valeurs dans la liste des entiers de 0 à N-1 construite par range(N). . C'est le moyen le plus rapide de construire une boucle à N itérations.
- Le module random chargé dans la première ligne est indispensable pour les problèmes de modélisation en statistiques. Il permet comme ici de disposer d'une fonction random() qui génère un nombre pseudo-aléatoire compris entre 0 et 1.

10.2 Tortue logo : Lire et comprendre un programme générant un tracé

Quelle est l'image correspondant à l'algorithme ?

Réponse : il s'agit de l'image 3, comme le montre le programme python suivant dont on voit le résulat sur la FIG. 19 et qui fait avancer (forward) et tourner de 90ř vers la droite (right) une tortue.

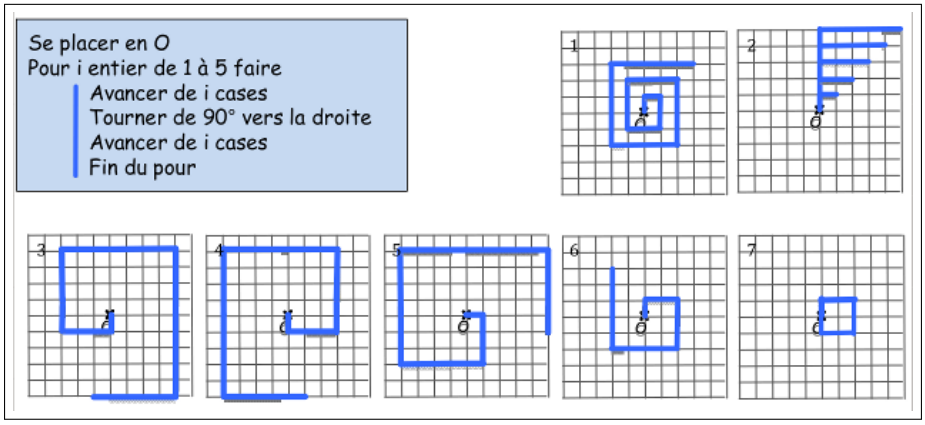

FIG. 18 : Tortue

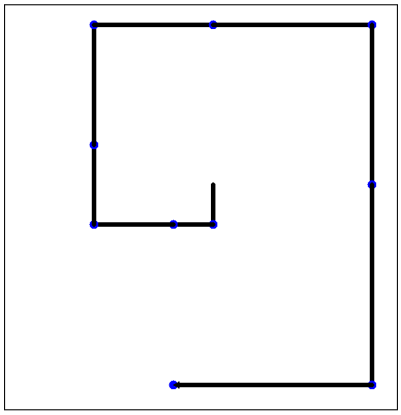

FIG. 19 : Tortue Python

```
from turtle import *

pu()      # pen up : on ne laisse pas de trace
goto(0,0) # aller à l'origine
pd()      # pen down : on laisse desormais une trace

for i in range(1,6):
  forward(i)
  dot("blue")
  right(90)
  forward(i)
  dot("blue")
```

10.3 Problème d'indentation : comprendre et modifier un script avec boucle et tests

Le programme suivant a été écrit pour calculer la somme des entiers pairs strictement inférieurs à 11 (variable s) et la somme des entiers impairs strictement inférieurs à 11 (variable t).

Pourquoi renvoie-t-il t = 0 ?

```
s = 0
t = 0
for i in range (11):
  if i%2==0:
    s = s+i
if i%2==1:
  t=t+i
```

```
return s,t
```

Corriger le programme pour qu'il affiche les résultats attendus : (30,25).

10.4 Test de primalité

Prérequis Affectation, structures conditionnelles, boucle while

Principe Un nombre premier est un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même).

Comment vérifier si un entier N est premier ou non ? Voici un algorithme possible : à partir de 2 tester tous les nombres entiers k jusqu'à en trouver un qui divise N.

Cette tâche aura une fin, c'est certain puisque

- soit N n'est pas premier et on va trouver un entier k<N divisant N
- soit N est premier et on va trouver un entier k=N qui ... divise N

ainsi, N est premier si et seulement si k=N à l'issue de la boucle

Intérêt Même si l'algorithme est simpliste, il fonctionne bien grâce à la puissance et la vitesse de calcul des machines modernes. Cependant on peut l'améliorer puisqu'aucun diviseur de N ne peut être plus grand que $\frac{N}{2}$. D'autre part, il faudra veiller à ce que les tests sur de nombres comme N=0, N=1 ou N=2 ne posent pas de problème. Quitte à placer des exceptions sous forme de SI ... ALORS.

Ce thème sera prolongé avec l'algorithme du crible d'Ératosthène.

```
import math

def est_premier(N):
  """ Teste si un nombre est premier ou non. """
  premier=True;               # par defaut, on suppose N premier.
                              # on revisera eventuellement cette supposition
  for k in range(2,N):
    if math.floor(N/k)==N/k:  # autre facon de dire que k divise N
      premier = False         # on a trouve un diviseur<N donc N non premier
  return premier
```

Remarque : Évitez à tout prix Python2, l'opérateur de division / y effectue une division entière ainsi, 1/2=0 plutôt que 1/2=0.5. C'est une source de confusion importante. Cette ambigüité a disparu dans Python3.

Variante :

On peut tout à fait écrire un algorithme du même type en utilisant une boucle while. Il sera aussi efficace pour les nombres premiers en revanche il n'effectuera pas de tours inutiles pour les nombres non premiers, voyez plutôt

```
def est_premier(N):
  """ Teste si un nombre est premier ou non. """
  k=2
  while N%k != 0:
    k=k+1
  return k==N
```

10.5 La suite de Syracuse

La suite de Syracuse est définie par récurrence, de la manière suivante :

On part d'un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. En répétant l'opération, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur.

Prérequis Affectation, structures conditionnelles, boucle while

Principe Mathématiquement on définit la suite $u_0 = N$ et pour tout entier $n \geq 0$

- Si u_n est pair, alors $u_{n+1} = \frac{u_n}{2}$
- Si u_n est impair, alors $u_{n+1} = 3u_n + 1$

Par exemple, à partir de 14, on construit la suite des nombres appelée suite de Syracuse du nombre 14

14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2

Après que le nombre 1 a été atteint, la suite des valeurs (1,4,2,1,4,2 ...) se répète indéfiniment en un cycle de longueur 3, appelé cycle trivial.

Intérêt La conjecture (donc non démontrée) affirme que la suite de Syracuse de n'importe quel entier strictement positif atteint 1. Il est étonnant qu'un énoncé aussi simple puisse résister la communauté des mathématiciens qui s'est pourtant largement mobilisée sur ce problème depuis plus de 60 ans.

Algorithmiquement, le problème est simple mais pose généralement problème aux élèves. Comme pour toutes les suites $(u_n)_n$ construites par une récurrence $u_{n1} = f(u_n)$ l'idée (non triviale) consiste à ne pas s'encombrer de l'indice n. En effet à chaque itération, on remplace : $u = f(u)$.

L'observation graphique de la suite pour $N = 15$ et pour $N = 127$ montre que la suite peut s'élever assez haut avant de retomber. Les graphiques font penser à la chute chaotique d'une feuille emportée par le vent. De cette observation est né tout un vocabulaire imagé : on parlera notamment de l'altitude et de la durée de vol de la suite. On définit alors :

- **la durée de vol** : c'est le plus petit indice n tel que $u_n = 1$
- **l'altitude maximale** : c'est la valeur maximale de la suite

Quelques résultats pour vérifier vos programmes :

- 6 - 3 - 10 - 5 - 16 - 8 - 4 - 2 - 1 ; la durée du vol pour **6** est de **8** et son altitude maximale est de **16**
- 10 - 5 - 16 - 8 - 4 - 2 - 1 ; la durée du vol pour **10** est de **6** et son altitude maximale est de **16**
- La durée du vol pour **15** est de **17** et son altitude maximale est de **160**
- La durée du vol pour **127** est de **46** et son altitude maximale est de **4372**

```
1  import math
2
3  def duree_vol(u):
4    """ Calcule la duree de vol et l'altitude max de la suite de syracuse du nombre u. """
5    dr_vol = 0
6    alt_max = 0
7
8    while u!=1:
9      if u%2==0:
10       u = u/2
11     else:
12       u = 3*u+1
13     dr_vol = dr_vol+1  # duree augment à chaque tour
14     if alt_max < u:    # est-on jamais alle aussi haut ?
15       alt_max=u        # enregistement de l'altitude
16
17   return dr_vol, alt_max
```

10.6 Recherche du PGCD de deux nombres

10.6.1 Premier algorithme simpliste : on teste tous les nombres

Prérequis Affectation, structures conditionnelles, boucle while

Principe Étant donnés deux nombres A et $B \leq A$, on cherche le PGCD en testant tous les nombres inférieurs à B

Liste des variables à utiliser

- A et B : les nombres entiers dont on cherche le PGCD
- D : le diviseur potentiel à tester
- P : le pgcd

Algorithme en langage naturel

```
Demander les valeurs de A et B à l'utilisateur (B<=A)
D reçoit 1 et P reçoit D
TantQue D <= B
  Si D divise à la fois A et B
    Alors P reçoit D
  Fin_Si
  D est augmenté de 1
Fin_TantQue
Écrire à l'écran que le PGCD est P
```

Intérêt On peut se poser de nombreuses questions

- On pourra réfléchir au comportement du programme en l'absence de la ligne D est augmenté de 1
- Que se passe-t-il si dans la boucle, cette ligne est placée non pas après le bloc conditionnel mais avant ?
- Que peut-il se passer si l'utilisateur entre des nombres quelconques et que faire pour que le programme réagisse correctement ?

10.6.2 Second algorithme : méthode des soustractions successives

Prérequis Affectation, structures conditionnelles, boucle while

Principe Rappelons que si un nombre est un diviseur de 2 nombres n et p, alors il est aussi un diviseur de leur différence $n-p$. Étant donnés deux nombre A et $B \leq A$, on cherche le PGCD en soustrayant et en réaffectant.

Liste des variables à utiliser

- A et B : les nombres entiers dont on cherche le PGCD
- C : la différence

Algorithme en langage naturel

```
Demander les valeurs de A et B à l'utilisateur (B=<A)
C reçoit A–B
TantQue C != 0
  Si B < C
    Alors intervertir B et C
  Fin_Si
  A reçoit B
  B reçoit C
```

```
    C reçoit A–B
  Fin_TantQue
  Écrire à l'écran que le PGCD est B
```

Intérêt Cet algorithme ne peut pas être construit correctement avant d'avoir effectué le travail sur papier pour quelques couples de nombres. Une fois ce travail répétitif effectué, les élèves perçoivent comment construire l'algorithme et comment s'opèrent les réaffectations de variables.

- On pourra réutiliser ici un précédent travail sur l'interversion de deux nombres.
- Il faudra sans doute insister sur l'importance de l'ordre des trois affectations de la boucle.

10.6.3 Troisième algorithme : méthode des divisions successives

Prérequis Affectation, structures conditionnelles, boucle while

Principe Si les deux nombres A et $B \leq A$ sont éloignés, plutôt que de soustraire à de nombreuses reprises, on peut utiliser le fait que le PGCD de A et B est égal au PGCD de B et du reste de la division euclidienne de A par B.

Liste des variables à utiliser

- A et B : les nombres entiers dont on cherche le PGCD
- R : le reste dans la division euclidienne

Algorithme en langage naturel

```
Demander les valeurs de A et B à l'utilisateur (B<=A)
Si A < B
  Alors intervertir A et B
Fin_Si
R reçoit le reste de la divion euclidienne de A par B
TantQue R != 0
  A reçoit B
  B reçoit R
  R reçoit le reste de la divion euclidienne de A par B
Fin_TantQue
Écrire à l'écran que le PGCD est B
```

Intérêt Encore une fois, il faut avoir fait fonctionner la méthode sur papier pour pour pouvoir construire un algorithme fonctionnel.

- C'est un algorithme difficile. Il devra sans doute être accompagné d'exemples où les divisions successives et les affectations des variables seront effectuées en suivant scrupuleusement la séquence décrite par l'algorithme.
- Une erreur classique consiste à dire que dans la dernière ligne, le PGCD est R. On pourrait demander en exercice de corriger cette erreur volontairement glissée.

10.7 `randint` pour les lancers de dés

Simuler le lancer de 1000 dés puis

1. Écrire un algorithme capable d'afficher la proportion de 6.
2. Écrire un algorithme capable d'afficher le rang du premier 6.
3. Écrire un algorithme capable d'afficher le rang du troisième 6. (Remarque : réfléchir au fait que si on avait demandé le rang du cinq-centième 6, cela aurait peut-être posé problème...)
4. Écrire un algorithme capable d'afficher le rang du dernier 6.

Prérequis Affectation de variable, conditions et boucles for et while

Principe La fonction randint(a,b) de la bibliothèque random renvoie un nombre réel aléatoire dans l'intervalle borné par les deux entiers a et b. La simulation d'un lancer de dé s'en déduit simplement DE=randint(1,6).

Intérêt Il faudra utiliser habilement et selon les cas les boucles de type for ou while. Une bonne compréhension des spécificités de ces deux types de boucles sera utile. Remarquer qu'il est faisable mais difficile de faire un programme unique qui puisse répondre aux quatre questions.

1. Proportion de 6

```
S = 0
Pour i allant de 1 à 1000
  d = entier aléatoire entre 1 et 6
  Si d est égal à 6
    Alors S = S+1
  FinSi
FinPour
Affiche "Proportion : " S/1000
```

2. Rang du premier 6

```
d = 0; i = 0
TantQue d<6
  d = entier aléatoire entre 1 et 6
  i = i+1
FinTantQue
Affiche "Rang du permier 6 : " i
```

3. Rang du troisième 6

```
d = 0; i = 0; nbSix=0
TantQue d<6 ET nbSix<3
  d = entier aléatoire entre 1 et 6
  i = i+1
  Si d=6
    Alors n = n+1
  FinSi
FinTantQue
Affiche "Rang du troisième 6 : " i
```

4. Rang du dernier 6

```
Pour i allant de 1 à 1000
  d = entier aléatoire entre 1 et 6
  Si d=6
    Alors rgDernier = i
  FinSi
FinTantQue
Affiche "Rang du dernier 6 : " rgDernier
```

10.8 Marche aléatoire

Un ivrogne sort du bar pour rentrer chez lui à une distance de 100 m. Il est tellement ivre qu'à chaque pas le faisant avancer d'un mètre, il se décale aléatoirement d'un mètre vers la gauche ou vers la droite. Sachant qu'il part au milieu d'un trottoir de 10 m de large, estimer la probabilité qu'il arrive chez-lui sans tomber dans le caniveau. (FIG. 20)

FIG. 20 : Marche aléatoire

Prérequis Affectation de variable, conditions et boucles for et while

Principe Il est nécessaire de générer un nombre aléatoire valant -1 ou 1. Pour cela on pourra utiliser les fonctions avancées des langages de programmation mais il peut être intéressant de n'utiliser que la fontion random() dont il était question dans l'exercice précédent sur les lancers de dé.

Intérêt
- Cet exercice trouve un prolongement naturel avec le problème de la planche de Galton.
- Une version graphique montrant les mouvements du personnage est tout à fait envisageable.
- La modélisation peut conduire à des solutions diverses et variées. Voici une proposition

```
from random import *

def marche_alea(dist_maison):
  """renvoie la suite des pas du marcheur
    +1 (resp. -1) pour un pas en diagonale vers le haut (resp. le bas)
    ex :
    > print(marche_alea(10))
    => [1, 1, -1, 1, -1, 1, -1, 1, 1, 1]
  """
  L = []
  for i in range(dist_maison):
    L = L + [choice([-1,1])]
  return (L)

def rentre_bien(L,dist_caniveaux):
  """ verifie si le marcheur a est de retour sans accident
  ex :
  > rentre_bien([1, -1, -1, -1, -1, -1, 1, -1, -1, -1])
  => False
  > rentre_bien([1, 1, -1, 1, -1, -1, -1, 1, 1, -1])
  => True
  """
  y = 0
  for i in L:
    y = y + i
    if y >= dist_caniveaux or y <= -dist_caniveaux:
      return False
  return True

def stats(N,dist_maison,dist_caniveaux):
  """ lance N parties et renvoie la proportion de retours sans accident"""
  S = 0
  for i in range(N):
    L = marche_alea(dist_maison)
    if rentre_bien(L,dist_caniveaux):
      S = S+1
```

```
37    return S/N
```

Remarque Notez l'usage de la fonction choice(L) qui pioche au hasard un élément de la liste L. Nous aurons l'occasion de reparler des listes dans le prochain chapitre.

Voici une vesion graphique du problème avec le module pylab. Dans PythonAnywhere, ce script va générer une image nommée ici marcheur.png qu'on peut afficher une fois le programme exécuté.

```
1  import pylab
2  import random
3
4  xcadre = [10, 0, 0, 10, 10, 10.2,10.2]
5  ycadre = [-5,-5, 5, 5,  -5, -5,  5]
6  pylab.plot(xcadre, ycadre)
7
8  x = 0
9  xlist = [x]
10 y = 0
11 ylist = [y]
12
13 while (-5<y<5 and x<10):
14     x = x+1
15     xlist.append(x)
16     y = y+random.choice([-1,1])
17     ylist.append(y)
18
19 pylab.plot(xlist,ylist)
20 pylab.savefig("marcheur.png")
```

10.9 Jeu du lièvre et de la tortue

À chaque tour, on lance un dé. Si le 6 sort, alors le lièvre gagne la partie, sinon la tortue avance d'une case. La tortue gagne quand elle a avancé 6 fois. Question : le jeu est-il à l'avantage du lièvre ou de la tortue ?

```
1  import random
2
3  def lievre_tortue(N):
4    nb_victoire_tortue = 0
5    nb_victoire_lievre = 0
6    for i in range(N):
7      tour = 0
8      de = 0
9      while tour<6 and de!=6:
10       tour = tour+1
11       de = random.randint(1,6)
12     if  de==6:
13       nb_victoire_lievre = nb_victoire_lievre+1
14     else:
15       nb_victoire_tortue = nb_victoire_tortue+1
16   return nb_victoire_lievre, nb_victoire_tortue
```

Le résultat est sans appel : victoire du lièvre !

```
   lievre_tortue(10000)
=> (6678, 3322)
   lievre_tortue(10000)
=> (6639, 3361)
```

Remarque : L'usage des listes simplifie l'implémentation de ce type de problème. Ici, on crée lancers, la liste des résultats des six lancers de dé puis on attribue la victoire en fonction de la présence du nombre 6 dans cette liste.

```
import random

def lievre_tortue(N):
  nb_victoire_lievre, nb_victoire_tortue = 0,0
  for i in range(N):
    lancers = [random.randint(1,6) for i in range(6)]
    if 6 in lancers:
      nb_victoire_lievre = nb_victoire_lievre+1
    else:
      nb_victoire_tortue = nb_victoire_tortue+1
  return nb_victoire_lievre, nb_victoire_tortue
```

10.10 Algorithme de Héron

Il s'agit d'un algorithme classique d'approximation de la racine carrée d'un nombre réel positif.

Prérequis Affectation de variable, boucle for ou while.

Principe Soit a un réel strictement positif et n un entier naturel. Si x_n est une approximation strictement positive par excès de $\sqrt{a}$, alors a/x_n est une approximation par défaut de $i\sqrt{a}$. La moyenne arithmétique de ces deux approximations est $x_{n+1} = \frac{1}{2}(x_n + a/x_n)$ et constitue une nouvelle approximation par excès de $\sqrt{a}$.

En posant $x_0 = a$, on construit ainsi une suite dont on peut démontrer la convergence vers $\sqrt{a}$.

Intérêt En dehors du fait que cet algorithme pourra faire l'objet de calculs mathématiques intéressants (démontrer les assertions ci-dessus, prouver que les suites extraites $(x_{2n})_{n\in\mathbb{N}}$ et $(x_{2n+1})_{n\in\mathbb{N}}$ sont respectivement décroissante et croissante puis qu'elles convergent vers une limite commune qui est bien $\sqrt{a}$), les élèves auront sans aucun doute des difficultés à concevoir l'algorithme en utilisant uniquement des variables réelles. Il faudra les aider à comprendre que la variable n n'a pas d'intérêt pour l'approximation et qu'à chaque étape, seuls les deux derniers rangs sont utiles aux calculs.

```
from math import *

def heron(x,epsilon):
  """ Calcule une valeur approchee de la racine carree de x jusqu'a ce que
  la difference entre 2 termes # consecutifs soit inferieure a epsilon
  """
  w = 0    # le rang precedent est inititalise a 0

  while abs(w-x)>epsilon :
    w = x
    x = (x+a/x)/2
  return x
```

10.11 Intégration numérique par la méthode des rectangles

Approcher $\int_a^b f(t)\,\mathrm{d}t$ pour une fonction f et un intevalle $[a,b]$ raisonnables...

Prérequis Affectation de variable, boucle for

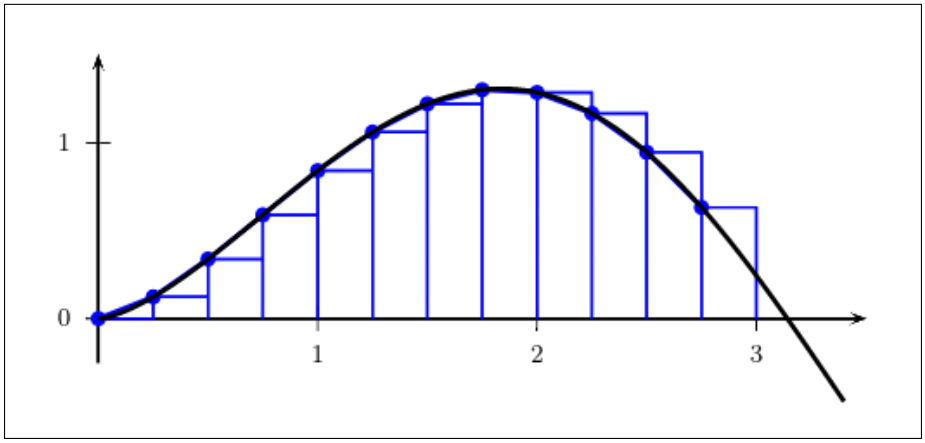

FIG. 21 : Méthode des rectangles : approximation de $\int_0^3 f(t)\,\mathrm{d}t$

Intérêt Le document d'accompagnement pour l'algorithmique et la programmation en seconde propose une méthode d'approximation de la longueur d'un arc de courbe au moyen d'une subdivision de l'intervalle. La méthode présentée ici peut être aussi bien constituer un préalable qu'une suite. On pourra aussi prolonger le travail sur la méthode des rectangles par la méthode des trapèzes dont la vitesse de convergence est meilleure.

```
def rectangle(f,a,b,N):
  """ Calcule l'intégrale de f sur l'intervalle [a,b]
  approchee par N rectangles a gauche. """
  S = 0
  eps = (b-a)/N
  for i in range(N):
    S = S + eps * f(a + i*eps)
  return S

def f(x):
  return 2*x+3
```

10.12 Une calculatrice graphique avec Python

Voici un programme Python contenant une fonction graph qui trace une courbe paramétrée sur [−4,4[:

```
def line(x1, y1, x2, y2):
  """ Trace le segment [(x1,y1) (x2,y2)] """
  pu(); goto(x1, y1); pd(); goto(x2, y2)

def graph(f):
  """ Dessine une representation graphique de la fonction f. """
  x = -4
  pu(); goto(x, f(x)); pd()
  while x < 4:
    x += 0.1
    goto(x, f(x))

from turtle import *
setworldcoordinates(-4,-3,4,5)
line(-4,0,4,0)
line(0,-3,0,5)
pensize(3)

from math import cos
```

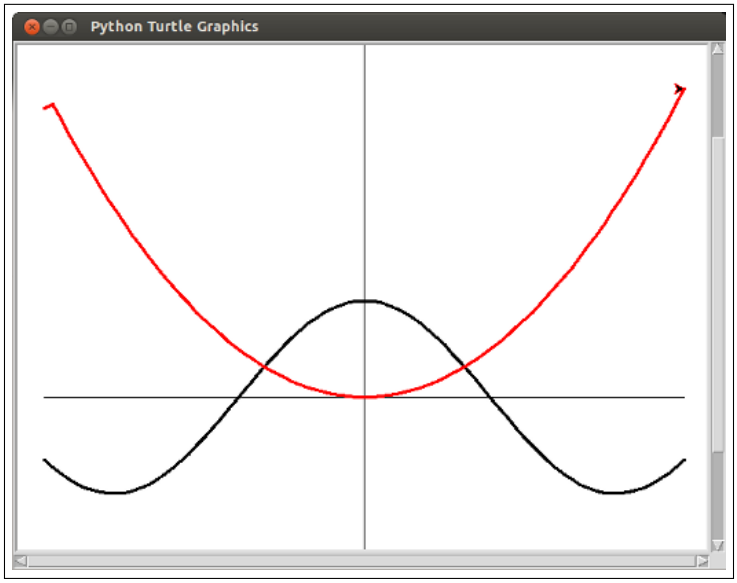

FIG. 22 : Courbes avec turtle

```
20 graph(cos)
21
22 def mafonction(x):
23     return x*x/5
24
25 pencolor("red")
26 graph(mafonction)
27
28 input() # evite que le graphe disparaisse
```

Sachant qu'il produit le résultat présenté sur la FIG. 22, le modifier pour

1. Ajouter en bleu la courbe représentative de la fonction $x \mapsto \frac{x^3}{8}$
2. Ajouter en vert la courbe représentative de la fonction $x \mapsto \frac{1}{x}$ (attention...)
3. Placer une marque pour chaque point de coordonnées entières sur l'axe des abscisses. Voir http://docs.python.org/2/library/turtle.html
4. Dessiner un grillage dont les intersection sont les points de coordonnées entières

10.13 Méthode de la dichotomie

Soit $f : [a,b] \to \mathbb{R}$ une fonction continue telle que $f(a)f(b) < 0$. D'après le théorème des valeurs intermédiaires, l'équation $f(x) = 0$ admet une solution sur l'intervalle $[a,b]$, *qu'on suppose unique*, et qu'on cherche à approcher numériquement à $\varepsilon > 0$ près.

Prérequis Affectation de variable, boucle for ou while.

Principe La méthode est la suivante : Calculons $f(m)$ où $m = (a+b)/2$. Le résultat est

- soit nul, auquel cas la solution cherchée est trouvée exactement,
- soit du signe contraire de l'un des nombres $f(a)$ et $f(b)$. Si c'est $f(a)$, alors pour la même raison que ci-dessus, le zéro de f se trouve dans l'intervalle $[f(a), f(m)]$. Si c'est $f(b)$, le zéro de f se trouve dans l'intervalle $[f(m), f(b)]$.

L'intervalle de recherche est donc de longueur moitié moindre. Répétant l'opération, on se retrouve à chercher le zéro de f dans un intervalle de longueur arbitrairement petite.

Intérêt Pas facile à concevoir et la boucle est difficile à programmer par les élèves. Commencer par le "jeu du plus ou moins" ou programmer sans imposer de précision au départ, peut permettre d'aborder le problème en douceur...

Algorithmiquement, on peut en profiter pour introduire la programmation structurée (programmation d'une sous-fonction qui se contente de diviser l'intervalle en 2, puis appels répétés à cette sous-fonction). On peut la programmer avec un WHILE, c'est le plus naturel ou utiliser un FOR en résolvant une inéquation avec logarithmes pour déterminer le nombre d'étapes nécessaires. On peut aussi, lorsque les élèves ont compris le principe de l'algorithme, leur soumettre un programme incomplet qui trouve le zéro et ne s'y arrête pas. Autre question : Que donne l'algorithme si on lui soumet un intervalle où la fonction s'annule plusieurs fois ? L'hypothèse *le zéro est unique dans* $[a,b]$ est-elle nécessaire ?

```
def dichotomie(a,b,f,epsilon):
  """ Renvoie les extremites d'un intervalle de largeur epsilon
  contenant une probable racine de l'equation f(x)=0 sur l'intervalle [a,b].
  """
  m=(a+b)/2.0
  while b-a >= epsilon and f(m)!=0 :
    m=(a+b)/2.0
    if f(m)*f(a) > 0:
      a=m
    else:
      b=m
  return a,b

def g(x):
  return x**2-2.0
```

Remarque python est un langage assez évolué pour prendre la fonction f en paramètre. C'est le *callback*.

11 Les listes

11.1 Calcul de moyenne

Étant donnée une liste de nombres, calculer sa moyenne.

Prérequis Affectation de variable et boucles for

Principe On va découvrir le type de données Python nommé "listes", ce qui permettra de calculer la moyenne d'une quantité non définie au départ de données.

Intérêt L'intérêt pratique est presque nul mais algorithmiquement, le fonctionnement est un cas d'école. De plus la programmation de cet exemple en Python est très proche du langage naturel.

```
def moyenne(L):
  """ Renvoie la moyenne des nombres contenus dans la liste L. """
  S = 0
  for i in L:
    S = S + i
  moyenne = S / len(L)
  return moyenne
```

Syntaxe commentée Quelle simplicité ! Remarquez la syntaxe de la boucle for ; en français, on traduirait par "*pour chaque nombre i de la liste L*" ? Python est ici très proche du langage naturel.

On pourra prolonger ce travail avec la recherche des minimum et maximum puis pourquoi pas, le calcul de l'écart type. La question de la recherche de la médiane est plus complexe : c'est un bon exercice pour des élèves rapides.

La fonction len, abbréviation de *length* renvoie bien sûr la longueur de la liste, c'est à dire le nombre d'éléménts.

11.2 Arithmétique indienne

Voici trois algorithmes

1. *Algorithme de Prabhakar* : chaque terme est formé à partir du précédent par la somme des carrés de ses chiffres. Par exemple : pour $u_0 = 162$, on aura $u_1 = 1^2 + 6^2 + 2^2 = 41$ puis $u_2 = 4^2 + 1^2 = 17$ etc. On constate que quel que soit l'entier naturel duquel on part on aboutit
   - soit au cycle 4,16,37,58,89,145,42,20,4 etc.
   - soit à 1
   - soit à 0 (uniquement pour 0).

2. *Variante de l'algorithme précédent* : chaque terme est formé à partir du précédent par la somme des cubes de ses chiffres. Par exemple : pour $u_0 = 162$, on aura $u_1 = 1^3+6^3+2^3 = 225$ puis $u_2 = 2^3+2^3+5^3 = 141$ etc.
   - Se limiter aux 500 premiers termes de la suite.
   - Vérifier que pour les nombres de la forme $3n,\ n \in \mathbb{N}$, la suite converge vers 153.
   - Vérifier que pour les nombres de la forme $3n+2,\ n \in \mathbb{N}$, la suite converge vers 371.

3. *Nombres consistants* : chaque terme est formé à partir du produit des chiffres du précédent. On appelle *persistance d'un nombre* le nombre d'étapes nécessaires pour obtenir un nombre à un seul chiffre. On appelle *nombre consistant* tout nombre dont la persistance est supérieure à 4. Dresser pour un entier $N < 80$ donné la liste des N premiers nombres consistants.

Prérequis Affectation de variable, conditions et boucles for et while

Principe On va se limiter aux nombres à quatre chiffres. Écrire tout d'abord les quatre fonctions capables d'exraire les quatre chiffres d'un nombre entier inférieur ou égal à 9999. Ensuite on pourra se lancer dans la construction des trois algorithmes pour les trois exercices.

Intérêt L'utilisation de fonctions pour la décomposition va permettre de simplifier la conception et l'écriture du problème.

Une fois les quatre fonctions construites, seul l'exercice sur les nombres consistants demande une petite astuce puisqu'il faudra veiller à ne pas systématiquement extraire les quatre chiffres pour les multiplier entre eux ; par exemple les quatre chiffres du nombre 367 sont 0,4,6 et 7 on pensera à effectuer le produit $4 \times 6 \times 7$ plutôt que $0 \times 4 \times 6 \times 7$.

Avec des listes, on pourrait envisager un programme ne se limitant pas aux nombres à quatre chiffres.

Commençons par les quatre fonctions d'extraction de chiffres

```
1 from math import *
2
3 def unt(x):  # Renvoie le chiffre des unites d'un nombre x
4   return int(x-(floor(x/10))*10)
5
6 def dzn(x):  # Renvoie le chiffre des dizaines d'un nombre x
7   return unt(floor(x/10))
8
9 def cnt(x):  # Renvoie le chiffre des centaines d'un nombre x
```

```
  return unt(floor(x/100))

def mll(x):  # Renvoie le chiffre des milliers d'un nombre x
  return unt(floor(x/1000))
```

La première et la seconde suite de Prabakhar (somme des carré puis des cubes des chiffres) s'écrit alors

```
def praba_carre(n):
  """ Renvoie la suite de Prabakhar somme des
  carres des chiffres pour un nombre n < 10 000
  """
  L = []
  while n != 0 and n != 1 and n != 4 :
    n = unt(n)**2 + dzn(n)**2 + cnt(n)**2 + mll(n)**2
    L.append(n)
  return L

def praba_cube(n):
  """ Renvoie la suite de Prabakhar somme des
  cubes des chiffres pour un nombre n < 10 000
  """
  L = []
  while n != 0 and n != 1 and n != 4 :
    n = unt(n)**3 + dzn(n)**3 + cnt(n)**3 + mll(n)**3
    L.append(n)
  return L
```

Syntaxe commentée TODO parler de append

Enfin pour afficher la liste des premiers nombres consistants, on va définir deux fonctions annexes

```
def prod_chiffres(n):
  """ Renvoie le produit des chiffres d un nombre entier n """
  if n >= 1000:
    n = unt(n)*dzn(n)*cnt(n)*mll(n)
  if 100 <= n < 1000:
    n = unt(n)*dzn(n)*cnt(n)
  if 10 <= n < 100:
    n = unt(n)*dzn(n)
  return n

def persistance(n):
  """ Calcule la persistance d'un nombre N """

  persistance = 0
  while n>=10:
    n = prod_chiffres(n)
    persistance = persistance + 1
  return persistance

def consistants(N):
  """ Renvoie la liste des N premiers nombres consistants """
  L = []
  nbre_de_nbres_csistts = 0
  n = 10 # le premier nombre a tester
  while nbre_de_nbres_csistts < N and n < 10000:
    n = n + 1
    if persistance(n) >= 4:
      L.append(n)
      nbre_de_nbres_csistts = nbre_de_nbres_csistts + 1
  return L
```

11.3 Algorithme de Kaprekar

Chaque terme est formé à partir du précédent par la différence de deux entiers dont l'un est formé des chiffres en ordre décroissant et l'autre des mêmes chiffres en ordre croissant. Par exemple : pour $u_0 = 5387$, à partir de ses chiffres, [5,3,8,7], on forme les nombres 8753 et 3578. On calcule alors $u_1 = 8753 - 3578 = 5175$ puis on recommence avec les chiffres de 5175 : [5,1,7,5] ce qui donne $u_2 = 7551 - 1557 = 5994$, etc. par récurrence.

L'algorithme de Kaprekar, a l'étonnante propriété de converger soit vers 0, soit vers 6174 pour tous les nombres entiers inférieurs à 9999.

Dans l'exemple précédent, nous avons déjà construit les quatre fonctions capables de décomposer un nombre pour obtenir ses chiffres constitutifs. Il est alors simple de construire la liste L de ces quatre chiffres.

Concernant le tri de la liste L, afin de ne pas ajouter de difficulté à ce problème arithmétique, utiliserons les fonctions de tri que Python possède déjà : L.sort() pour le tri croissant et L.reverse() pour le tri décroissant.

```
# Les fonctions mll, cnt, dzn et unt sont
# définies comme dans l'exemple précédent.

def kaprekar(N):
  """ Renvoie la suite de kaprekar d un nombre N """
  kap = []
  while N!=0 and N!=6174:
    L = [mll(N),cnt(N),dzn(N),unt(N)]
    m = L.sort()
    min = 1000*m[0]+100*m[1]+10*m[2]+m[3]
    M = L.reverse()
    Max = 1000*M[0]+100*M[1]+10*M[2]+M[3]
    N=Max-min
    kap.append(M)
  return kap
```

12 Marché aux algorithmes

(beaucoup des exercices sont extraits de Gérard Cordes)

12.1 La machine à café

Une machine à café est remplie d'un peu de monnaie en pièces de 10 et 20 centimes. Ayant introduit de l'argent et commandé une boisson, un client attend qu'on lui rende sa monnaie, qui s'élève à la valeur d'une variable appelée montant.

Ecrire une fonction qui calcule en fonction de montant les nombres NDix et NVingt de pièces de chaque valeur que le client doit recevoir, de façon que la machine conserve le plus de monnaie possible pour les autres clients à venir.

Remarques On a volontairement limité l'énoncé pour *cette activité est susceptible de prendre de nombreuses formes* en fonction du comportement plus ou moins sophistiqué que l'on veut voir adopter par la machine. C'est ce qui fait sa richesse et sa souplesse :

- Le plus faible aura la satisfaction d'avoir un éclairage sur le fonctionnement des automates qu'il utilise tous les jours.
- Il est facile de diversifier, d'en demander plus au bon élève qui a déjà fini, qui sera content d'écrire un test vérifiant que le montant demandé est bien multiple de 10 centimes. On peut donc prolonger le travail pour aller vers un programme concret.

- Si on remplace la machine à café par le distributeur de billets, on pourra demander une boucle qui implémente la procédure de sécurité qui ne laisse que trois essais pour taper le code.

12.2 le SIX en trois coups

Voici un jeu : on lance un dé cubique bien équilibré jusqu'à ce que le six sorte. Si le six est sorti avant le troisième coup ou au troisième coup, c'est gagné sinon c'est perdu.

Questions : Ecrire un algorithme pour simuler 1000 fois ce jeu. Estimer la probabilité de gagner à ce jeu .

12.3 Distance d'arrêt

La distance d'arrêt D_A en mètres d'un véhicule roulant à la vitesse V (en km/h) peut se calculer par la formule

$D_A = \frac{V}{3,6} + \frac{V \times V}{254 \times C_F}$

où C_F est le coefficient de frottement des pneus sur la route.

On donne : sur route sèche $C_F = 0,8$ et sur route mouillée $C_F = 0,4$. Ecrire un algorithme qui donne la distance d'arrêt (en mètres) d'un véhicule quand on connaît sa vitesse en km/h et la météo du jour.

12.4 Un premier algorithme de tri

1. Écrire un algorithme capable de vérifier qu'une liste est triée par ordre croissant.
2. Écrire un algorithme capable de fusionner deux listes croissantes en un nouvelle liste croissante elle aussi. Par exemple fusion_croiss([2,6,9], [1,4,5,9]) renverra [1,2,4,5,6,9,9] .

12.5 Simulations statistiques avec une pièce

1. Écrire un algorithme capable de renvoyer la fréquence de l'événement "Face" pour n tirages de "Pile" ou "Face".
2. Écrire un algorithme capable de renvoyer la fréquence de l'événement "2 Faces successifs" pour n tirages de Pile ou Face.
3. Écrire un algorithme capable de renvoyer la fréquence de l'événement "3 Faces sur 3" pour n simulation de 3 tirages de Pile ou Face.
4. Écrire un algorithme capable de renvoyer la fréquence de l'événement "Au moins 80 Faces sur 100" pour n simulation de 100 tirages de Pile ou Face.

Remarque : Ce type d'exercice s'étend sans problème aux jeux de dés ou de boules colorées.

12.6 Années bissextiles

Une année est bissextile si son millésime est divisible par 4 ou par 400 mais pas seulement par 100. Écrire un algorithme cacable de déterminer si une année est bissextile.

12.7 Temps écoulé entre deux horaires

Écrire un algorithme capable de calculer la durée entre deux horaires deonnés d'une même journée (hh_1, mm_1, ss_1) et (hh_2, mm_2, ss_2).

Plusieurs stratégies sont possibles : avec ou sans conversion préalable. On pourra prolonger cet exercice en ajouter une date (année, mois, jour) ce qui obligera à se pencher sur le problème précédent.

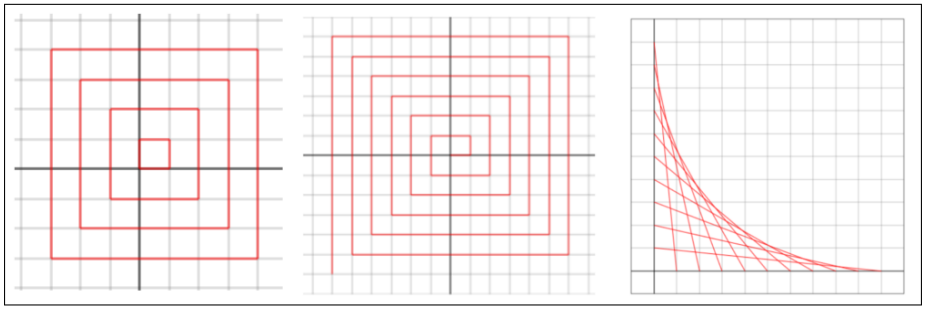

FIG. 23 : Carrés emboîtés, spirale carrée et enveloppe de l'astroïde

12.8 Très chouette fonction définie par un algorithme

On considère la fonction f définie dans R en posant : $f(x) = y$ avec y donné par l'algorithme suivant programmé sous Algobox :

```
LIRE x
u PREND LA VALEUR x+3
v PREND LA VALEUR x−1
y PREND LA VALEUR u∗u−v∗x
AFFICHER y
```

(a) Calculer $f(2)$.
(b) Calculer $f(10)$.
(c) La fonction f ainsi définie est-elle affine ?

12.9 Fred prend le taxi

Pour une course en taxi, Fred hésite entre deux tarifs : Tarif 1 : prise en charge de 2.40€ au départ puis 1.20€ par kilomètre. Tarif 2 : prise en charge de 4.5€ au départ puis 0.90€ par kilomètre. On appelle x le nombre de kilomètres parcourus en taxi.

1. Ecrire un programme Algobox qui choisit pour Fred le meilleur tarif en fonction de x
2. Simplifier l'algo pour qu'il oublie de répondre si les deux tarifs sont identiques et écrire des questions qui guident les élèves pour tester le programme et le corriger.

12.10 Feux tricolores

On considère un feu tricolore qui règle la circulation à un carrefour. Ecrire un algorithme qui détermine la couleur du feu lors du prochain changement connaissant la couleur (vert, orange ou rouge) actuelle du feu.

Algorithme à écrire en langage naturel et à faire tourner à la main, en boucle (quelle couleur obtient-on ?)

12.11 Tracés

Construire un carré de côté c sur Algobox. Puis des carrés emboîtés. Construire une spirale carrée. Construire l'enveloppe de l'astroïde (voir les trois figures suivantes qui valent mieux qu'un long discours).

12.12 Aires et périmètres

Écrire les algorithmes permettant de calculer les aires et périmètres de figures classiques du plan.

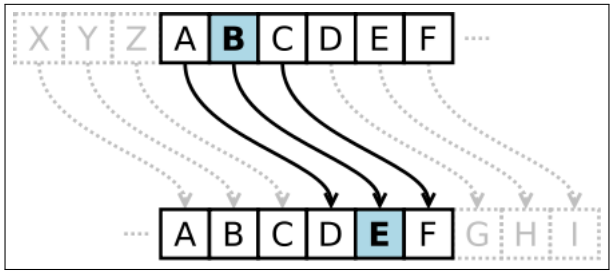

FIG. 24 : Cryptage César

12.13 Géométrie analytique

1. **Nature d'un triangle** – Écrire un algorithme capable de déterminer la nature d'un triangle connaissant les coordnnées cartésiennes de ses trois sommets.
2. **Nature d'un qaudrilatère** – Écrire un algorithme capable de déterminer la nature d'un quadrilatère connaissant les coordnnées cartésiennes de ses quatre sommets.
3. **Le point est-il sur la droite ?** – Écrire un algorithme capable de déterminer si un point $M(x_M, y_M)$ appartient à une droite d'équation $y = ax + b$.
4. **ALignement de trois points** – Écrire un algorithme capable de déterminer si trois points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ sont alignés ou non.
5. **Intersection de deux cercles** – Écrire un algorithme capable de déterminer si deux cercles $C_1(x_1, y_1, r_1)$ et $C_2(x_2, y_2, r_2)$ ont une intersection vide ou non.

Remarque : Dans ces exercices, il faudra tenir compte des erreurs d'arrondi et accorder un marge d'erreur aux test d'égalité.

12.14 Triplets pythagoriciens

Trouver tous les triplets d'entiers non nuls (x, y, z) compris entre 1 et 1000 et vérifiant la relation de Pythagore : $x^2 + y^2 = z^2$.

12.15 Nombre de Lychrel

http ://fr.wikipedia.org/wiki/Nombre_de_Lychrel

12.16 Suite de Fibonacci

Elle débute de la manière suivante :

`1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, . . .`

Elle est caractérisée par le fait que chaque nombre à partir du troisième est la somme des deux précédents. On pourra faire calculer le rapport de deux termes consécutifs, dont la valeur converge vers le nombre d'or.

12.17 Cryptage de Jules césar

Source Wikipedia En cryptographie, le chiffrement par décalage, aussi connu comme le chiffre de César (voir les différents noms), est une méthode de chiffrement très simple utilisée par Jules César dans ses correspondances secrètes (ce qui explique le nom "chiffre de César").

Le chiffre de César fonctionne par décalage des lettres de l'alphabet. Par exemple dans l'image, il y a une distance de 3 caractères, donc B devient E dans le texte codé.

Écrire un algorithme capable de crypter ou décripter une chaine de caractères saisie par un utilisateur.

On pourra par exemple utiliser les fonctions ord et chr de python.

```
>>> ord('a')
97
>>> chr(97)
'a'
>>> chr(ord('a') + 3)
'd'
```

Remarque : La fonction Python qui va être construite ici est d'un genre tout à fait nouveau pour les élèves : c'est une fonction de l'ensemble des lettres de l'alphabet à valeurs dans lui même. Ainsi, si on note cripto_cesar la fonction et A l'ensemble des lettres de l'alphabet, on a

$$\texttt{cripto\_cesar} : A \to A$$

Les fonctions ord et chr quant à elles vérifient

$$\texttt{ord} : A \to \mathbb{N} \qquad \text{et} \qquad \texttt{chr} : \mathbb{N} \to A$$

12.18 Calcul de l'indice de masse corporelle

On mesure l'obésité, c'est-à-dire, l'excès de graisse, à l'aide de l'indice de masse corporelle, noté I, évalué à partir du poids P (en kg) et de la taille T (en mètres) d'un individu par la formule :

$I = \frac{P}{T^2}$.

I est une fonction des deux variables P et T. Suivant une classification établie par l'Organisation Mondiale de la Santé, un individu est en surpoids lorsque $I > 25$.

(a) Calculer l'indice de masse corporelle d'une personne de poids 80 kg et de taille 1.75 m. Cette personne est-elle en surpoids ?

(b) Même question pour une personne de poids 70 kg et de taille 1.70 m.

(c) Ecrire un algorithme qui demande à l'utilisateur son poids en kg et sa taille en mètres, puis qui calcule l'indice I et enfin qui affiche si l'utilisateur en surpoids ou non.

(d) Programmer cet algorithme sur Algobox.

(e) Pour un poids de 60 kg, à quelle taille un individu est-il en surpoids ?

12.19 Avec un polynôme de degré 2

1. Écrire un algorithme capable de calculer le discriminant puis les éventuelle racines.
2. Écrire un algorithme capable de calculer pour quel entier, la fonction polynôme atteint son maximum sur un intervalle $[m, n]$ avec $m < n$ entiers donnés.
3. Tout polynôme du second degré $ax^2 + bx + c$ peut s'écrire sous la forme canonique $a(x-\alpha)^2 + \beta$ et lorsque son discriminant est positif, sous la forme factorisée $(x - x_1)(x - x_2)$. Écrire un algorithme capable de proposer la forme canonique et la forme factorisée lorsque c'est possible.

12.20 Une suite de nombres

Pour un entier naturel non nul N, écrire un algorithme d'afficher les N premiers termes de la suite suivante~:

1, 12, 123, 1234, 12345, 123456, etc.

Attention, c'est bien la suite d'entiers qu'on demande de définir, et pas un programme qui afficherait des chaînes de caractère. Cet exercice est donc plus difficile qu'il n'y paraît...

12.21 Arithmétique classique

1. **Les diviseurs d'un entier** – Écrire un algorithme capable d'afficher la liste des diviseurs d'un entier naturel.
2. **Premier ou pas premier ?** – Écrire un algorithme capable de dire si un entier naturel est premier ou non.
3. **Une liste de nombre premiers** – Écrire un algorithme capable d'afficher la liste des nombres premiers inférieurs à un entier naturel donné.
4. **Une liste de nombre premiers jumeaux** – Deux nombres premiers sont jumeaux si leur différence vaut 2 (ex. (5,7) ou (17,19)). Écrire un algorithme capable d'afficher la liste des nombres premiers inférieurs à un entier naturel donné.

12.22 Calcul de la moyenne d'une suite de notes

Deux facons de procéder pour deux types de boucles.

1. L'utilisateur saisit le nombre de notes en sa possession puis il saisit chacune des notes. L'algorithme renvoie alors la moyenne de ces notes.
2. L'utilisateur saisit des notes et termine sa saisie en entrant la note -1 qui signifie la fin de sa liste de notes. L'algorithme renvoie alors la moyenne des notes entrées.

Prolongement possible : Prévoir une gestion des coefficients.

12.23 Remboursement d'emprunt

Écrire un algorithme capable de donner le nombre d'années nécessaires pour rembourser un emprunt d'une somme S avec un taux à T% et des annuités fixes d'un montant A

12.24 Minimum de trois nombres

Écrire un algorithme cacapble d'extraire le plus petit d'une liste de trois nombres. mini(87,32,46) devrait renvoyer 32.

12.25 Planche de Galton

Source Wikipedia. Une planche de Galton est un dispositif inventé par Francis Galton qui illustre la convergence d'une loi binomiale vers une loi normale.

Des clous sont plantés sur la partie supérieure de la planche, de telle sorte qu'une bille lâchée sur la planche passe soit à droite soit à gauche pour chaque rangée de clous. Dans la partie inferieure les billes sont rassemblées en fonction du nombre de passages à gauche et de passage à droite qu'elles ont fait.

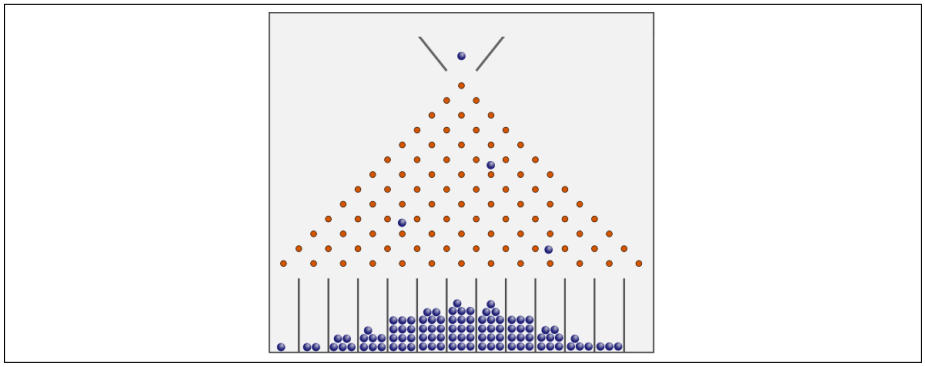

FIG. 25 : Planche de Galton

Ainsi chaque case correspond à un résultat possible d'une expérience binomiale (en tant qu'une expérience de Bernoulli répétée) et on peut remarquer que la répartition des billes dans les cases approche la forme d'une courbe de Gauss, autrement dit : la loi binomiale converge vers la loi normale. Il s'agit donc d'une illustration au théorème de De Moivre-Laplace.

Index