Document d'accompagnement des stages de formation à l'algorithmique

Document rédigé par

le groupe "Algorithmique" de l'IREM de Grenoble

Mise à jour Mai 2012

Ont participé au groupe "Algorithmique"de l'IREM de Grenoble : Michèle Benois, Anne Bilgot, Nadia Brauner, Martine Brilleaud, Christian Davin, Damien Jacquemoud, Anne Krotoff, Bernard Lacolle, Pascal Lafourcade, Michel Lamarre, Simon Modeste, Gilles Mounier, Jean-Pierre Peyrin, Marie-Jo Schmitt

Contact : Bernard.Lacolle@imag.fr

Table des matières

Introduction

De très nombreuses pistes ont été proposées pour élaborer le contenu d'un stage de formation des enseignants de mathématiques à des éléments d'algorithmique.

Pour élaborer le schéma de ce stage nous nous sommes inspirés de constats faits dans le document officiel d'accompagnement proposant les pistes suivantes.

Les compétences suivantes pourront être identifiées et travaillées :

- *comprendre et analyser un algorithme préexistant ;*
- *modifier un algorithme pour obtenir un résultat particulier ;*
- *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.*

Nous avons choisi d'orienter l'étude de l'algorithmique en liaison avec les mathématiques et l'apprentissage du raisonnement. Ceci nous amène à privilégier l'élaboration de solutions dont on pourra aisément vérifier l'exactitude.

Nous avons également élaboré sur la plupart des thèmes des propositions de situations pour la classe que les professeurs pourront mettre en œuvre avec leurs élèves. Ces situations pourront servir de base pour la construction d'autres exercices pour la classe.

Webographie

- Les cours de Gérard Berry au collège de France
  - `http://www.college-de-france.fr/`

- Mode d'emploi Python
  - `http://inforef.be/swi/python.htm`

Première partie

Quelques éléments de base

Cette introduction est un préambule à la compréhension des algorithmes proposés dans la progression pédagogique qui lui fait suite.

Définition : Un algorithme est l'expression d'une solution à un problème par composition d'actions plus élémentaires.

Il y a 4 manières de composer des actions :

La composition séquentielle
La composition séquentielle de deux actions $A1$ et $A2$, notée "$A1; A2$" , indique que l'action $A1$ est effectuée, puis, une fois $A1$ terminée, l'action $A2$ est effectuée à son tour.

Exemple (moyenne M de deux nombres $N1$ et $N2$) :
$S \leftarrow N1 + N2$; $M \leftarrow S/2$

On peut aussi exprimer la composition séquentielle de deux actions par un simple changement de ligne.

La composition conditionnelle
La composition conditionnelle de deux actions $A1$ et $A2$, selon une condition C, notée "si C alors $A1$ sinon $A2$", indique que l'action $A1$ est effectuée si C est vraie, ou bien l'action $A2$ est effectuée si C est fausse. Une variante, notée "si C alors A", indique que l'action A est effectuée seulement si la condition C est vraie.

Exemple (maximum M de deux nombres $N1$ et $N2$) :
`si` $N1 \geq N2$ `alors` $M \leftarrow N1$ `sinon` $M \leftarrow N2$

La composition itérative
La composition itérative d'une action A (avec elle-même) selon une condition de terminaison C, notée "tantque non C faire A", indique que l'action A est répétée jusqu'à ce que la condition C devienne vraie. Si C est vraie au départ, l'action A n'est pas effectuée.

Exemple (somme S des N premiers nombres entiers ; $N \geq 1$) :
$i \leftarrow 1; S \leftarrow 1$
`tantque` $i \neq N$ `faire` $i \leftarrow i + 1; S \leftarrow S + i$
Remarque. La condition d'arrêt C est $(i = N)$.

La composition récursive
Une action A est récursive si elle est exprimée par une composition d'actions dont elle fait elle-même partie. Cette composition rend nécessaire la dénomination de l'action et l'introduction de paramètres. Nous en verrons l'usage dans le cadre de la progression pédagogique.

1 Manipulation des mémoires et affectation

Objectif algo : s'approprier la notion de variable ; effectuer des opérations simples sur les variables (affecter une valeur, échanger des valeurs, utiliser des valeurs stockées dans des variables pour construire une opération dont le résultat est stocké dans une autre variable).

1.1 Le coin des profs

Exercice 1 :

On considère deux variables A et B, dans lesquelles on stocke deux valeurs arbitraires a et b.

Question 1 – Écrire un algorithme qui échange les valeurs stockées dans les variables A et B (en utilisant une autre variable et/ou sans utiliser une autre variable).

Question 2 – Écrire un algorithme qui permute les valeurs stockées dans A, B et C (en effectuant une permutation circulaire).

Question 3 – Programmer chacun des deux algorithmes en Python.

1.2 Situation pour la classe : manipulation des mémoires et affectation

On expliquera d'abord ce que signifie $M \leftarrow N$ en comparant l'affectation au « copier - coller », pour que les élèves comprennent que ce que l'on a mis dans M reste dans N.

Exercice 1 :

Question 1 – Porter sur le schéma suivant les valeurs des variables A, B et C lors du déroulement de l'algorithme.

	A	B	C
	1	1	1
$A \leftarrow 2$			
$B \leftarrow 10$			
$C \leftarrow A + B$			
$C \leftarrow B - C$			

Question 2 – Qu'obtient-on en dernière ligne si à la première étape, on met 5 dans la variable A au lieu de mettre 2 ? Et si l'on met -3 ?

Question 3 – Même question pour des valeurs initiales quelconques mises dans les variables A, B et C.

	A	B	C
	a	b	c
$C \leftarrow A + B$			
$C \leftarrow B - C$			

Exercice 2 : Porter sur le schéma suivant les valeurs prises de la variable X lors du déroulement de l'algorithme.

	X
	0
$A \leftarrow 1$	
$B \leftarrow 9$	
$C \leftarrow 4$	
$D \leftarrow 8$	
$X \leftarrow A$	
$X \leftarrow X * 10 + B$	
$X \leftarrow X * 10 + C$	
$X \leftarrow X * 10 + D$	

Exercice 3 :

Question 1 – Porter sur le schéma suivant les valeurs des variables A et B lors du déroulement de l'algorithme.

	A	B
	a	b
$A \leftarrow B$		
$B \leftarrow A$		

Question 2 – En utilisant une troisième variable C, écrire un algorithme qui permute les valeurs des variables A et B.

Question 3 – Porter sur le schéma suivant les valeurs des variables A et B lors du déroulement de l'algorithme :

	A	B
	a	b
$A \leftarrow A + B$		
$B \leftarrow A - B$		
$A \leftarrow A - B$		

Exercice 4 : Trouver un algorithme qui transforme le contenu des variables A, B et C

A	B	C
a	b	c

en le contenu suivant (permutation circulaire) :

A	B	C
c	a	b

On utilisera une autre variable D.

Exercice 5 : On considère la recette suivante :

```
Choisir un nombre entier N.
Lui ajouter 4.
Multiplier la somme obtenue par le nombre N choisi.
Ajouter 4 à ce produit.
Écrire le résultat.
```

Question 1 – Faire fonctionner cette recette pour les entiers N compris entre 0 et 6.

Question 2 – Formuler une conjecture sur la valeur du résultat obtenu pour un entier N quelconque.

Question 3 – Écrire un algorithme qui prend en entrée un nombre entier N et qui renvoie le résultat de la recette dans une variable R (l'algorithme devra contenir exactement trois instructions d'affectation).

Question 4 – Programmer cet algorithme en Python et tester la conjecture pour d'autres nombres entiers.

Question 5 – Prouver la conjecture.

2 Schémas itératifs et manipulations des listes

Prérequis algo : manipulation des mémoires.
Objectif algo : manipuler des collections d'objets - prise en main d'un schéma itératif.

L'objectif de cette section est la prise en main d'un schéma itératif élémentaire ainsi que la manipulation de listes. Pour le schéma itératif, nous nous limiterons pour le moment au « tant que ».

2.1 Situation pour la classe : prise en main d'un schéma itératif

Exercice 1 :
Question 1 – Comparer les deux algorithmes suivants :

```
I ← 0
A ← 0
A ← A + I
I ← I + 1
A ← A + I
I ← I + 1
A ← A + I
I ← I + 1
```

```
I ← 0
A ← 0
tantque I ≤ 2
   A ← A + I
   I ← I + 1
```

en portant dans chacun des deux cas les valeurs prises par A et I après chaque instruction dans un tableau (cf. partie sur la manipulation des mémoires). Quel est l'intérêt de l'instruction tantque ?

Question 2 – Pour suivre le déroulement d'un algorithme comprenant plusieurs itérations, on peut se contenter d'écrire les valeurs prises par chaque variable une fois par itération (c'est ce que l'on fera dans les exercices suivants) : il faut alors choisir si on le fait au début ou à la fin de l'itération. En pratique, cela revient à choisir entre les deux tableaux suivants :

	I	A
Début itération 1		
Début itération 2		
Début itération ...		
Début itération ...		

	I	A
Fin itération 1		
Fin itération 2		
Fin itération ...		
Fin itération ...		

Remplir chacun des deux tableaux pour le deuxième algorithme.

Exercice 2 : Détailler le fonctionnement de l'algorithme suivant, et le comparer au deuxième algorithme de l'exercice 1.

```
I ← 0
A ← 0
tantque I ≤ 2
   I ← I + 1
   A ← A + I
```

Exercice 3 :

Que pensez-vous de l'algorithme suivant ?

```
I ← 0
A ← 0
tantque I ≤ 2
   A ← A + I
```

Exercice 4 : On considère l'algorithme suivant.

```
I ← 0
S ← 0
Entrer N
tantque I ≤ N
   S ← S + I
   I ← I + 1
S ← S / N
Écrire S
```

Question 1 – Sachant que l'on a pour un entier N positif :

$$1 + 2 + \cdots + N = \frac{N(N+1)}{2},$$

que calcule cet algorithme ?

Question 2 – Le résultat est-il exact pour $N = 0$?

Question 3 – Que pensez-vous du cas $N < 0$?

Exercice 5 : Écrire un algorithme qui, étant donné un entier N, renvoie la somme des carrés des N premiers entiers naturels.

2.2 Situations pour la classe : manipulation des listes

Jusqu'à présent, quand nous avions plusieurs variables (plusieurs emplacements mémoire), nous avons désigné chacun(e) d'elles par un nom différent : par exemple A, B, C, D...

Pour pouvoir désigner facilement un grand nombre d'objets du même type sans attribuer un nom différent à chacun, il devient indispensable d'utiliser un nom unique pour désigner une *collection d'objets*, en repérant chaque objet de la collection par un *numéro*, appelé aussi *indice* : par exemple $A[0], A[1], A[2], A[3], A[4]$ désigneront 5 objets d'une collection A, ou plus généralement,

$$A[0], A[1], A[2], A[3], \ldots, A[N],$$

avec N entier naturel quelconque, désigneront $N+1$ objets d'une collection A.

Dans la suite, A s'appelera une *liste* et en général les éléments de A seront des nombres ou des caractères.

Dans cette partie algorithmique nous ne ferons pas référence à un langage de programmation. Signalons cependant que, selon le langage de programmation utilisé, on pourra rencontrer les termes *liste, tableau, vecteur,...* ; de même on trouvera les notations $A[I]$ ou $A(I)$ pour désigner l'élément d'indice I. Suivant les langages, les tableaux commenceront à l'indice 0 ou l'indice 1 (chaque option ayant ses avantages et inconvénients). Nous ferons abstraction dans la partie algorithmique de ces différences dues aux langages de programmation.

A titre d'exemple, pour créer la liste A dont les éléments sont $A[0], A[1], A[2], A[3], A[4]$, nous utiliserons l'algorithme :

```
I ← 0
tantque I ≤ 4
   Entrer A[I]
   I ← I + 1
```

ou, plus généralement, pour créer une liste A depuis l'élément A[DEBUT] à l'élément A[FIN] :

```
I ← DEBUT
tantque I ≤ FIN
   Entrer A[I]
   I ← I + 1
```

Exercice 1 :

Question 1 – On considère l'ensemble des valeurs numérotées :

$$A[0], A[1], A[2], A[3], A[4].$$

Que fait l'algorithme suivant ?

```
I ← 0
tantque I ≤ 4
   A[I] ← I
   I ← I + 1
```

Question 2 – Avec les valeurs ainsi calculées, que fait l'algorithme suivant ?

```
I ← 0
S ← 0
tantque I ≤ 4
   S ← S + A[I]
   I ← I + 1
```

Exercice 2 : Écrire un algorithme qui, élément par élément, échange les valeurs de la liste A :

$$A[0], A[1], A[2], A[3], A[4],$$

avec les valeurs de la liste B :

$$B[0], B[1], B[2], B[3], B[4].$$

Exercice 3 :

Question 1 – Que fait l'algorithme suivant ?

```
I ← 0
tantque I ≤ 2
   B[I] ← A[4-I]
   I ← I + 1
```

Question 2 – Que fait l'algorithme suivant ?

```
I ← 0
tantque I ≤ 2
   A[I] ← A[4-I]
   I ← I + 1
```

Exercice 4 :

Écrire un algorithme qui, à partir de la liste A : $A[0], A[1], A[2], A[3], A[4]$, renvoie la liste $B[0], B[1], B[2], B[3], B[4]$ comportant les valeurs de la liste A dans l'ordre inverse.

Exercice 5 : On considère l'ensemble de valeurs numérotées :

$$A[0], A[1], A[2], A[3], \cdots, A[N],$$

$$B[0], B[1], B[2], B[3], \cdots, B[N].$$

Écrire un algorithme qui échange (élément par élément) les valeurs de $A[I]$ et $B[I]$ pour toutes les valeurs de $I = 0, 1, 2, 3, \cdots, N$.

Exercice 6 : On considère l'ensemble des valeurs numérotées :

$$A[0], A[1], A[2], A[3], ..., A[N],$$

N étant un entier strictement positif.

Question 1 – Combien il y a-t-il d'éléments dans la liste A ?

Question 2 – Que fait l'algorithme suivant ?

```
I ← 0
S ← 0
tantque I ≤ N
   S ← S + A[I]
   I ← I + 1
S ← S / (N+1)
Écrire S
```

3 Schémas conditionnels

Prérequis algo : manipulation des mémoires, des collections d'objets.
Objectif algo : prise en main de l'instruction conditionnelle, mise en place de la répétition d'une action complexe.

3.1 Le coin des profs

Exercice 1 :

Question 1 – Écrire un algorithme qui calcule le maximum de deux variables A et B.

Question 2 – Écrire un algorithme qui calcule le maximum de trois variables A, B et C.

Question 3 – Votre algorithme se généralise-t-il à N variables ?

Question 4 – Écrire un algorithme qui calcule le maximum de N variables. Indiquer le nombre de comparaisons de l'algorithme.

Question 5 – Écrire un algorithme qui trie un ensemble de N variables. Indiquer le nombre de comparaisons de votre algorithme.

Question 6 – Programmer en Python l'un des algorithmes précédents.

Question 7 – Écrire un algorithme qui calcule les deux plus grandes valeurs parmi N variables.

Question 8 – Tester l'algorithme écrit dans la question précédente avec la liste 2, 4, 3, 1 (le programme devra renvoyer 4 et 3), puis avec la liste 4, 1, 4, 2 (le programme devra renvoyer 4 et 2). Que se passe-t-il avec la liste 4, 4, 4, 4 ?

3.2 Situation pour la classe : prise en main de l'instruction conditionnelle

Exercice 1 : Que fait l'algorithme suivant ?

```
Entrer X
Si X ≥ 0 alors
   Y ← X
Si X < 0 alors
   Y ← -X
```

Exercice 2 : Que fait l'algorithme suivant ?

```
Entrer X
Si X ≥ 0 alors
   Y ← X
sinon
   Y ← -X
```

3.3 Situation pour la classe : maximum, minimum, tri de 2 nombres

Exercice 1 : Écrire un algorithme lisant les valeurs de A et B et affectant le maximum de A et B à la variable `MAXIMUM`.

Une solution :

```
Entrer A, B
Si A ≥ B alors
   MAXIMUM ← A
Sinon
   MAXIMUM ← B
```

Note : on fera réfléchir les élèves au cas où $A = B$.

Exercice 2 : Compléter l'algorithme précédent pour affecter le maximum de A et B à la variable `MAXIMUM`, et le minimum de A et B à la variable `MINIMUM`.

Une solution :

```
Entrer A, B
Si A ≥ B alors
   MAXIMUM ← A
   MINIMUM ← B
Sinon
   MAXIMUM ← B
   MINIMUM ← A
```

Exercice 3 : Notre but est maintenant d'écrire un algorithme lisant les valeurs de A et B et affectant le maximum de A et B à la variable A, et le minimum de A et B à la variable B.

Question 1 – Que pensez-vous de l'algorithme suivant ?

```
Entrer A, B
Si A < B alors
   A ← B
   B ← A
```

Question 2 – Écrire l'algorithme correct en utilisant une variable supplémentaire C.

Exercice 4 : Refaire les exercices précédents dans le cas du minimum.

3.4 Situation pour la classe : maximum, minimum, tri de 3 nombres

Exercice 1 : On souhaite écrire un algorithme lisant les valeurs de A, B et C, et affectant le maximum de A, B et C à la variable `MAXIMUM`.

On essaiera de privilégier une solution utilisant le travail déjà fait. On pourra proposer une solution de ce type, en amenant petit à petit à la suppression des instructions inutiles.

```
Entrer A, B, C
# Calcul du maximum de A et B
Si A ≥ B alors
   MAXIMUM ← A
Sinon
   MAXIMUM ← B
# Calcul du maximum de MAXIMUM et C
Si C ≥ MAXIMUM alors
   MAXIMUM ← C
Sinon
# Ce calcul est inutile
   MAXIMUM ← MAXIMUM
```

Il y aura certainement d'autres propositions avec des « si alors sinon », pas toutes faciles à analyser. Il serait bon de privilégier pour les élèves les solutions qui se « lisent facilement, sans ambiguïté ».

Exercice 2 : Tri des 3 nombres A, B, C

> On essaiera de privilégier ce qui a déjà été écrit, la lisibilité des algorithmes écrits.

Question 1 – On utilise l'algorithme de l'exercice à la suite duquel les valeurs de A et B ont été ordonnées par valeurs décroissantes (le maximum dans A). On résumera le résultat de cet algorithme par « Ordonner A et B ».

Montrer que la succession des instructions :
« `Ordonner A et B` »
« `Ordonner A et C` »
« `Ordonner B et C` »
donne le résultat attendu et détailler cet algorithme.

> Un résultat possible :
>
> ```
> Entrer A,B,C
> # Ordonner A et B      # Ordonner A et C      # Ordonner B et C
> Si A < B alors         Si A < C alors         Si B < C alors
>    X ← A                  X ← A                  X ← B
>    A ← B                  A ← C                  B ← C
>    B← X                   C ← X                  C ← X
> ```

Question 2 – Essayer de proposer d'autres solutions.

3.5 Situation pour la classe : maximum, minimum dans un ensemble de valeurs

Exercice 1 : Nous considérons un ensemble de valeurs « numérotées » que nous appellerons :

$$A[0], A[1], A[2], A[3], A[4].$$

On reprendra la question 1 de l'exercice 1 où l'on a écrit un algorithme affectant le maximum de A et B à la variable `MAXIMUM`. On résumera cette action élémentaire par :

« Calculer le maximum de A et B (dans `MAXIMUM`) ».

Question 1 – Que fait la suite d'instructions suivantes ?

```
« Calculer le maximum de A[0] et A[1] (dans MAXIMUM) »
« Calculer le maximum de A[2] et MAXIMUM (dans MAXIMUM) »
« Calculer le maximum de A[3] et MAXIMUM (dans MAXIMUM) »
« Calculer le maximum de A[4] et MAXIMUM (dans MAXIMUM) »
```

Question 2 – Que fait la suite d'instructions suivantes ?

```
MAXIMUM prend la valeur A[0]
≪ Calculer le maximum de A[1] et MAXIMUM (dans MAXIMUM) ≫
≪ Calculer le maximum de A[2] et MAXIMUM (dans MAXIMUM) ≫
≪ Calculer le maximum de A[3] et MAXIMUM (dans MAXIMUM) ≫
≪ Calculer le maximum de A[4] et MAXIMUM (dans MAXIMUM) ≫
```

Question 3 – Lire, interpréter et compléter les algorithmes naturels suivants pour réaliser les deux séquences d'instructions élaborées ci-dessus.

```
Entrer A[0], A[1], A[2], A[3], A[4], A[5].
# ≪ Calculer le maximum de A[0] et A[1] (dans MAXIMUM) ≫


pour i = 2 à 4 faire
#≪ Calculer le maximum de A[i] et MAXIMUM (dans MAXIMUM) ≫
```

```
Entrer A[0], A[2], A[3], A[4], A[5].
MAXIMUM = A[0]
pour i = 1 à 4 faire
#≪ Calculer le maximum de A[i] et MAXIMUM (dans MAXIMUM) ≫
```

Pour aller plus loin

Exercice 1 : Traiter le cas : $A[0], A[1], A[2], A[3], \ldots, A[N]$.

Exercice 2 : Appliquer la méthode à la détermination du maximum et minimum d'une fonction que l'on a évaluée en des points

Exercice 3 : Écrire les algorithmes permettant de trouver le maximum des éléments A[DEBUT] ... A[FIN] et élaborer un algorithme de tri.

Deuxième partie

Activités

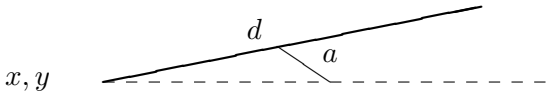
4 Dessine-moi un carré ...

Les questions sont progressives. Chaque question doit conduire à un programme Python.

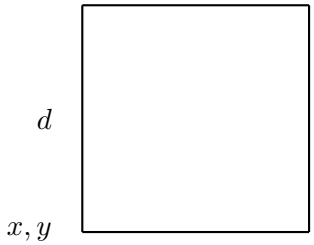
Pour importer la librairie de dessin : `from turtle import*`

Exercice 1 : Étant donné deux points $< x_1, y_1 >$ et $< x_2, y_2 >$, tracer le segment qui relie les deux points.

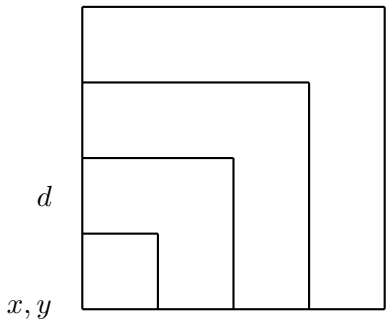
Exercice 2 : Étant donné un point $< x, y >$, un angle a et une distance d, tracer le segment depuis $< x, y >$ de direction a et de longueur d.



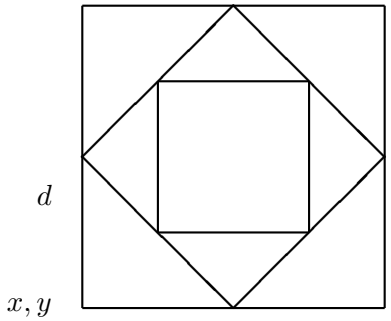
Exercice 3 : Étant donné un point $< x, y >$, et une distance d, tracer le carré depuis $< x, y >$ de côté d.



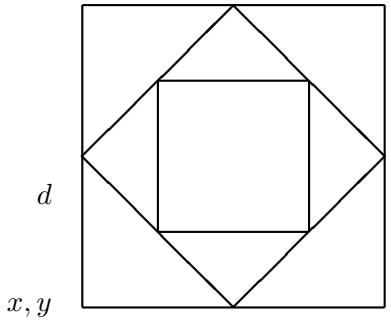
Exercice 4 : Étant donné un point $< x, y >$, et une distance d, tracer le carré depuis $< x, y >$ de côté d, et n carrés intérieurs.



Exercice 5 : Étant donné un point $< x, y >$, et une distance d, tracer le carré depuis $< x, y >$ de côté d, et n carrés intérieurs.



Exercice 6 : Étant donné un point $< x, y >$, et une distance d, tracer le carré depuis $< x, y >$ de côté d, et n carrés intérieurs sans lever la plume et sans repasser deux fois sur le même trait.



5 PGCD et preuve

Prérequis math : savoir ce qu'est le pgcd de deux nombres et les algorithmes appris en troisième.
Prérequis algo : les instructions "si" et "tant que" doivent être connues.
Objectif math : réinvestir l'algorithme d'Euclide (enseigné en troisième).
Objectif algo : programmer un algorithme connu pour automatiser une tâche répétitive.

5.1 Le coin des profs

Exercice 1 : Deux algorithmes de calcul du pgcd de deux entiers naturels figurent dans le programme de Troisième : l' « algorithme des soustractions » et l'algorithme d'Euclide.

Exemple : calcul du PGCD de 90 et 28

En appliquant l'algo. des soustractions :
$90 - 28 = 62$;
$62 - 28 = 34$;
$34 - 28 = 6$;
$28 - 6 = 22$;
$22 - 6 = 16$;
$16 - 6 = 10$;
$10 - 6 = 4$;
$6 - 4 = 2$;
$4 - \underline{2} = \underline{2}$;
Le pgcd de 90 et de 28 est donc égal à 2.

En appliquant l'algo. d'Euclide :
$90 = 28 \times 3 + 6$;
$28 = 6 \times 4 + 4$
$6 = 4 \times 1 + \underline{2}$
$4 = 2 \times 2 + \underline{0}$
Le pgcd de 90 et de 28 est donc égal à 2.

Question 1 – Écrire l'algorithme des soustractions (entrée : deux entiers entrés par l'utilisateur ; sortie : PGCD de ces deux entiers).

Question 2 – Le coder en langage Python.

Question 3 – Écrire l'algorithme d'Euclide (entrée : deux entiers entrés par l'utilisateur ; sortie : PGCD de ces deux entiers).

Question 4 – Le coder en langage Python.

On peut également calculer le PGCD de deux entiers à l'aide des décompositions en facteurs premiers de ces deux entiers.
Exemple : calcul du PGCD de 90 et 28
$90 = 2 \times 3^2 \times 5; 28 = 2^2 \times 7$ donc $PGCD(90, 28) = 2^1 \times 3^0 \times 5^0 \times 7^0 = 2$.

Question 5 – Écrire un algorithme permettant de calculer le PGCD de deux entiers naturels en utilisant cette méthode, et le programmer en Python.

Suggestions : on pourra par exemple
– écrire une fonction auxiliaire qui prend en entrée un entier naturel et renvoie la liste des

nombres premiers inférieurs ou égaux à cet entier ;
– écrire une fonction auxiliaire qui prend en entrée un entier N et renvoie la décomposition en facteurs premiers de cet entier, sous la forme de deux listes : une liste de facteurs premiers, une liste des exposants associés ;
– écrire enfin une fonction qui prend en entrée deux entiers naturels, et renvoie leur PGCD.

5.2 Idées pour des situations pour la classe : PGCD

Les documents d'accompagnement relatifs à l'algorithmique en classe de Seconde indiquent que : « l'introduction de chaque nouvel élément (variable, boucle, itération, etc.) devrait apparaître lors de la résolution de problèmes pour lesquels les démarches habituelles sont malcommodes ou peu performantes : par exemple dans le cas de répétition d'une tâche, ou dans le cas d'un traitement trop long pour être envisagé à la main. Ces situations peuvent être rencontrées lors de l'extension à des cas plus généraux de situations déjà rencontrées : recherche du pgcd de nombres très grands, tri d'un très grand nombre de valeurs numériques, simulations sur des échantillons de grande taille... »

Exercice 1 : **Tester si un nombre est premier**

Question 1 – Le nombre 119 est-il un nombre premier ?

Question 2 – Le nombre 139 est-il un nombre premier ?

Question 3 – Écrire un algorithme permettant de tester si un nombre entier est un nombre premier.

Question 4 – Le coder en langage Python.

Exercice 2 : **PGCD et simplification de fractions**

Partie 1 (Brevet des Collèges 2005, Aix-Marseille, Corse, Montpellier, Nice, Toulouse)

Question 1 – Trouver le PGDC de 6209 et 4435 en détaillant la méthode.

Question 2 – En utilisant le résultat de la question précédente, expliquer pourquoi la fraction $\frac{4435}{6209}$ n'est pas irréductible.

Question 3 – Donner la fraction irréductible égale à $\frac{4435}{6209}$.

Partie 2 (Généralisation)

Question 4 – Écrire un algorithme de calcul du PGCD de deux entiers naturels.

Question 5 – Écrire un algorithme permettant de déterminer si une fraction est irréductible, et, si elle ne l'est pas, permettant de donner sa forme simplifiée.
Entrée : deux nombres entiers (le numérateur et le dénominateur de la fraction à tester).
Sortie : deux nombres entiers (le numérateur et le dénominateur de la fraction simplifiée).

Exercice 3 :

Partie 1 (CRPE Bordeaux, Caen 2000)

Le service des espaces verts veut border un espace rectangulaire de 924m de long sur 728m de large à l'aide d'arbustes régulièrement espacés. Un arbuste sera placé à chaque angle du terrain. La distance entre deux arbustes doit être un nombre entier de mètres.

Question 1 – Déterminer toutes les valeurs possibles de la distance entre deux arbustes.

Question 2 – Déterminer, dans chaque cas, le nombre d'arbustes nécessaires à la plantation.

Partie 2 (Généralisation)

Question 3 – Écrire un algorithme permettant de dresser la liste de tous les diviseurs d'un nombre entier.

Question 4 – Le coder en langage Python.

Question 5 – Écrire un algorithme permettant de dresser la liste de tous les diviseurs communs entre deux nombres entiers.

Question 6 – Le coder en langage Python.

Question 7 – On revient au contexte donné en partie 1 ; écrire un algorithme qui, étant donné les dimensions d'un espace rectangulaire, renvoie les distances possibles entre arbustes et le nombre d'arbustes nécessaires à la plantation.

Question 8 – Le coder en langage Python.

6 À propos des nombres entiers

6.1 Système décimal

Il s'agit d'une suite de chiffres mais à regarder de plus près cela pourrait prendre plusieurs formes :

1. l'écriture décimale usuelle : 1679027
2. une liste de chiffres : $[1, 6, 7, 9, 0, 2, 7]$
3. une liste de caractères : "1679027"
4. une liste de caractères : ["1", "6", "7", "9", "0", "2", "7"]
5. ...

La représentation (1), la plus naturelle (!), est en fait complexe et a un double rôle :

1. cette représentation n'est en fait que la chaîne de caractères (3) (sans guillemets) représentant le nombre en question ;
2. l'ordinateur comprend une instruction du type $A \longleftarrow$ 1679027 ce qui veut dire qu'il traduit la chaîne de caractères précédente en un nombre "machine" sur lequel on va pouvoir faire les opérations arithmétiques. Plus précisément, l'ordinateur est capable d'appliquer des opérateurs arithmétiques sur cette représentation (ce n'est pas le cas sur les autres représentations) et par exemple, l'algorithme suivant a un sens :

```
A ⟵ 1679027
B ⟵ A + 1
C ⟵ A × B
```

6.2 Système en base quelconque

La liste $[1, 6, 7, 9, 0, 2, 7]$ fait explicitement référence à la notion de représentation, et donc pose la question de la base. Par contre, considérons les représentations en binaire du nombre $\overline{101}^{(2)}$ comme on l'écrit en mathématique. Si on écrit ce nombre sous la forme (1), l'instruction $A \longleftarrow 101$ affectera évidemment à A le nombre décimal 101 et non $1 \times 2^1 + 0 \times 2^1 + 1 \times 2^0 = 5$.

Si on travaille en base quelconque éventuellement supérieure à 10, seules les représentations 3 et 4 sont valables en y ajoutant des symboles adéquats, par exemple pour le système hexadécimal : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Pour simplifier, nous ne traiterons ici que de bases $b \leq 10$, et seules les représentations 1 et 2 pourraient être utilisées.

Il nous paraît donc plus clair d'utiliser uniquement la représentation (2) pour une base $b < 10$ même si elle est plus lourde, et se permettre les deux représentations (1) et (2) pour les nombres décimaux.

Tous nos algorithmes seront élaborés en supposant que nous pouvons faire les opérations usuelles sur les nombres entiers en "machine" : addition, soustraction, multiplication, quotient et reste de la division entière.

6.3 Travailler avec une représentation en liste

6.3.1 Des choix à faire !

Tout d'abord il nous faudra toujours faire référence à la base b considérée. Pour des raisons de simplicité, nous ne considérons que des bases $b \leq 10$.

Considérons maintenant la liste $A = [1, 6, 7, 9]$. En supposant que les indices commencent à zéro, nous avons donc :

$$A[0] = 1, A[1] = 6, A[2] = 7, A[3] = 9,$$

et le nombre correspondant est alors :

$$A[0] \times b^3 + A[1] \times b^2 + A[2] \times b^1 + A[3] \times b^0,$$

et nous n'avons pas une correspondance directe entre les indices de la liste et les puissances de b.

A contrario, le nombre :

$$A[0] \times b^0 + A[1] \times b^1 + A[2] \times b^2 + A[3] \times b^3,$$

ne correspond plus au "visuel" $A = [1, 6, 7, 9]$ mais au "visuel" $A = [9, 7, 6, 1]$.

Le tout est de faire un choix et de s'y tenir : nous préférons le premier choix "visuel" et d'ailleurs nous serons également amené à utiliser des représentations comme $[0, 0, 6, 7, 9]$.

6.3.2 Entrer un nombre sous forme d'une liste

On notera d'abord que l'algorithme élémentaire suivant et sa traduction Python :

```
Demander A
```

```
A = input( )
```

accepte aussi bien une entrée 1679027, qu'une entrée $[1, 6, 7, 9, 0, 2, 7]$.

Ceci suggère probablement d'autres façons d'entrer des nombres sous forme de liste. Rappelons que nous ne considérons que des bases $b \leq 10$. Le symbole, ou caractère, utilisé dans l'ensemble $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ sera appelé un "chiffre".

Exercice 1 : Écrire un algorithme puis un programme Python faisant l'acquisition d'un nombre A de N chiffres, sous la forme 2, en entrant un par un chacun de ses chiffres avec une écriture "naturelle" de la gauche vers la droite

Exercice 2 : Suivant le modèle de la seconde solution de l'exercice précèdent, écrire un algorithme puis un programme Python faisant l'acquisition d'un nombre A de N chiffres dans une liste de $M \geq N$ chiffres.

Exercice 3 : Écrire un algorithme puis un programme Python créant à partir d'un nombre représenté par une liste A à N chiffres une liste B de $M \geq N$ chiffres.

6.4 Changement de base

Exercice 1 :

Question 1 – Soit le nombre LXIII. Donner son écriture en base 10, en base 2, en base 7, en base b.

Question 2 – Un nombre N est écrit en base b, b entier naturel non nul inférieur à 10. Donner un algorithme permettant d'obtenir son écriture en base 10. On ne vérifiera pas que ce nombre a une écriture valide en base b.

Question 3 – Un nombre N est écrit en base 10. Donner un algorithme permettant d'obtenir son écriture en base b, b entier naturel non nul inférieur à 10.

Question 4 – Un nombre N est écrit en base b, b entier naturel non nul inférieur à 10. Donner un algorithme permettant d'obtenir son écriture en base 10 après avoir vérifié que ce nombre a une écriture valide en base b.

6.5 Somme, produit et complexité

On considère deux nombres entiers X et Y comportant respectivement n et m chiffres (on imagine n et m assez grands). On souhaite écrire les algorithmes permettant de calculer par les techniques posées classiques (en colonnes) la somme $X + Y$ et le produit $X \times Y$. Les algorithmes de ce paragraphe manipulent des listes qui contiennent les chiffres de X et de Y. Les nombres considérés sont donc représentés par des listes (chaque chiffre du nombre est un élément de la liste). Cela permet de manipuler des nombres très longs. (voir la section sur les manipulations de liste)

Exercice 1 :

Question 1 – Pour $X = [1, 2, 3, 4]$ et $Y = [3, 1, 4, 1]$, que contient la liste Z à la fin de l'algorithme suivant ?

Entrée : `Deux listes X et Y de même taille n`
Sortie: `Une liste Z`

```
pour i allant de 0 à n-1
   Z[i] ← max(X[i],Y[i])
```

Question 2 – Lorsque les deux nombres entiers X et Y comportent respectivement n et m chiffres, combien de chiffres peut contenir au maximum leur somme $X + Y$?

Question 3 – Écrire un algorithme permettant de calculer la somme $X + Y$ lorsque X et Y ont la même longueur n.

Question 4 – Même question lorsque X et Y n'ont plus la même longueur.

Question 5 – Programmer cet algorithme en Python.

Exercice 2 : Écrire un algorithme permettant de calculer le produit aX où a est un entier inférieur à 10.

Exercice 3 : Écrire un algorithme permettant de calculer le produit XY.

Exercice 4 : Modifier les algorithmes précédents pour les faire fonctionner dans une autre base.

Exercice 5 : Programmer l'un des algorithmes précédents en langage Python.

6.6 Idée de situation pour la classe : somme, produit

Exercice 1 : Écrire un algorithme qui affiche le résultat de l'addition de 2 nombres de 20 chiffres.

Exercice 2 : Écrire un algorithme qui double un nombre de 20 chiffres.

Exercice 3 : Écrire un algorithme qui affiche le résultat de la multiplication d'un nombre de 30 chiffres par 7.

Exercice 4 : Écrire un algorithme qui affiche le résultat de la multiplication de 2 nombres de 20 chiffres

7 Recherche dichotomique

7.1 Je cherche un nombre entre 1 et 1000

Prérequis algo : Instructions conditionnelles et boucles conditionnelles.
Prérequis math : Mathématiques du collège. Récurrence pour la preuve de la dernière question.

Ce jeu se joue à deux joueurs A et B. Le joueur A choisit secrètement un nombre cible compris strictement entre 1 et 1000. Le joueur B doit deviner ce nombre en faisant le minimum de propositions. À chaque proposition du joueur B, le joueur A répond par « le nombre cherché est plus grand », « le nombre cherché est plus petit » ou « bravo, vous avez gagné » selon la position de la proposition par rapport à la cible à atteindre.

Exercice 1 :

Le but de cet exercice est de jouer contre l'ordinateur.

Question 1 – Proposer un algorithme pour que l'ordinateur tienne le rôle du joueur A.

Question 2 – Programmer et tester cet algorithme en langage Python. On utilisera les instructions suivantes :

```
from random import *  // pour importer la librairie
randrange(1000)       // pour tirer un nombre entre 0 et 999
```

Question 3 – Imaginer une stratégie « optimum » qui permette au joueur B de toujours gagner avec un maximum de 10 propositions. Tester cette stratégie avec le programme de la Question 2.

Question 4 – Proposer un algorithme pour que l'ordinateur tienne maintenant le rôle du joueur B et applique la stratégie « optimum » précédente. A chaque proposition de l'ordinateur vous répondrez par « + », « - » ou « bravo » pour « le nombre cherché est plus grand », « le nombre cherché est plus petit » ou « c'est gagné ».

Question 5 – Écrire et tester cet algorithme en langage Python.

Question 6 – Quels sont les nombres à choisir pour que l'ordinateur trouve la solution en au moins 9 coups ?

Question 7 – Montrer que l'algorithme « optimum » permet de toujours trouver l'entier en au plus 10 essais.

Indication – On peut utiliser un raisonnement par récurrence sur K : si $B - A \leq 2^K$, alors on peut trouver le nombre compris strictement entre A et B en au plus K essais.

Remarque – Dans une situation pour la classe, avec les mêmes exercices sans les questions 6 et 7, on essayera de faire émerger chez les élèves la méthode de recherche par dichotomie.

7.2 Recherche de la solution d'une équation $f(x) = 0$ par dichotomie

Prérequis math : fonction, fonction croissante, décroissante.
Prérequis algo : tantque, instruction conditionnelle.
Objectif math : solution d'une équation $f(x) = 0$.
Objectif algo : mettre en place un algorithme itératif de calcul.

Cet exercice, souvent présenté comme l'un des plus naturel pour illustrer la liaison entre mathématique et algorithmique, se révèle en fait assez complexe.

La principale difficulté est que les calculs sont effectués de façon approchée. Un test $f(x) = 0$ pose problème car la quantité $f(x)$ n'est évaluée que de façon approchée.

Dans certains cas, calcul exact, cette question peut ne pas se poser, mais dans le cas général il nous faudra réfléchir au préalable à cette question.

Hypothèse faite dans tout le paragraphe : Nous ferons l'hypothèse que la fonction f est définie, croissante sur l'intervalle $[a, b]$ et que l'équation $f(x) = 0$ possède une solution unique sur $[a, b]$. La propriété de continuité n'est pas nécessaire !

Calcul exact

Partons tout d'abord avec l'hypothèse que le calcul de $f(x)$ se fait de façon exacte et examinons l'algorithme suivant :

```
Entrer a
Entrer b
Entrer precision
tantque (b-a) > precision
   c ← (a+b)/2
   si f(c) > 0
      b ← c
   sinon
      a ← c
Écrire c
```

Argumentation : considérons le point $c = \frac{a+b}{2}$ et regardons ce que nous pouvons déduire du test :
– si $f(c) > 0$, comme la fonction est croissante, la solution unique cherchée est dans l'intervalle $[a, c]$ (même $[a, c[$).
– dans le cas contraire ($f(c) \leq 0$) la solution cherchée est dans l'intervalle $[c, b]$.

Conclusion : en chaque début d'itération on peut affirmer que la solution appartient à l'intervalle $[a, b]$, l'amplitude de l'intervalle étant divisé par 2 à chaque étape, il est assez facile de prouver que l'algorithme s'arrête.

Remarque 1 : on peut également montrer que si la solution appartient à l'intervalle $[a, b[$ alors cette condition est maintenue tout au long de l'algorithme. En effet :
– si $f(c) > 0$, comme la fonction est croissante, la solution unique cherchée est dans l'intervalle $[a, c[$.
– dans le cas contraire ($f(c) \leq 0$) la solution unique est dans l'intervalle $[c, b[$.

Remarque 2 : Considérons l'exemple suivant. Soit n un entier $0 \leq n \leq 1000$ et considérons la fonction f définie sur $[0, 1001[$ par :

$$\begin{cases} f(x) = -1, \ x \in \ [0, n[\\ f(n) = 0 \\ f(x) = 1, \ x \in \ ]n, 1001[\end{cases}$$

On remarquera que l'algorithme de dichotomie résout le problème de la recherche du nombre n à condition que la longueur de l'intervalle résultant soit inférieur à 1, ce qui est réalisé après 10 itérations.

Calcul exact et arrêt éventuel de l'algorithme sur la valeur exacte

Nous supposons toujours que le calcul se fait de façon exacte et nous voulons intégrer dans l'algorithme un arrêt éventuel quand la valeur exacte est trouvée.

Nous partons de l'hypothèse que la solution recherchée est dans l'intervalle $]a, b[$, ce qui signifie que l'on a testé auparavant les valeurs extrêmes a et b.

L'adaptation de l'algorithme proposée est la suivante :

```
Entrer a
Entrer b
Entrer precision
trouve = 0
tantque ((b-a) > precision) et (non trouve)
   c ← (a+b)/2
   si f(c) = 0
      trouve = 1
   sinon
      si f(c) > 0
         b ← c
      sinon
         a ← c
si trouve = 1
   Écrire "solution exacte =",c
sinon
   Écrire "solution approchée dans", [a,b[
```

Calcul approché

Nous abordons le cas de calcul approché, comme le pratiquent les ordinateurs en arithmétique flottante. Nous ferons les hypothèses raisonnables suivantes pour l'évaluation approchée d'une quantité ξ :
– si la valeur exacte $\xi < 0$, alors sa valeur approchée est strictement négative.
– si la valeur exacte $\xi = 0$, alors sa valeur approchée est soit nulle, soit strictement positive, soit strictement négative, en fait on ne peut rien conclure,
– si la valeur exacte $\xi > 0$, alors sa valeur approchée est strictement positive.

Avec ces hypothèses, quelle signification peut-on alors donner au test $f(c) > 0$ sachant que le calcul de $f(c)$ a été fait de façon approchée :
– soit la valeur exacte de $f(c)$ est effectivement strictement positive,
– soit, du fait de l'approximation, on a pour la valeur exacte $f(c) = 0$.

Avec ces mêmes hypothèses, dans le cas contraire où la valeur approchée de $f(c)$ vérifie le test $f(c) \leq 0$ alors :
– soit la valeur exacte de $f(c)$ est effectivement strictement négative,
– soit, du fait de l'approximation, on a pour la valeur exacte $f(c) = 0$.

Avec une fonction f définie, continue, croissante sur l'intervalle $[a, b]$ et une solution unique de l'équation $f(x) = 0$ sur $[a, b]$ considérons le point $c = \frac{a+b}{2}$ et regardons ce que nous pouvons déduire du test $f(c) > 0$.

– Si la quantité exacte $f(c)$ est effectivement strictement positive, alors la solution unique cherchée est dans l'intervalle $[a, c]$ (même $[a, c[$).
– si du fait de l'approximation la valeur exacte de $f(c)$ est effectivement nulle, alors la solution unique cherchée est également dans l'intervalle $[a, c]$.

Dans le cas contraire où le test vérifie $f(c) \leq 0$.

– Si la quantité exacte $f(c)$ est effectivement strictement négative, alors la solution unique cherchée est dans l'intervalle $[c, b]$ (même $]c, b]$),
– si du fait de l'approximation la valeur exacte de $f(c)$ est effectivement nulle, alors la solution unique cherchée est également dans l'intervalle $[c, b]$.

Dans les deux cas on peut donc remplacer l'intervalle $[a, b]$ soit par l'intervalle $[a, c]$ soit par l'intervalle $[c, b]$ et l'algorithme déjà présenté :

```
Entrer a
Entrer b
Entrer precision
tantque (b-a) > precision
   c ← (a+b)/2
   si f(c) > 0
      b ← c
   sinon
      a ← c
Écrire c
```

se retrouve "robuste aux calculs approchés".

Attention : les problèmes de précision peuvent apparaître si l'on demande un encadrement trop précis.

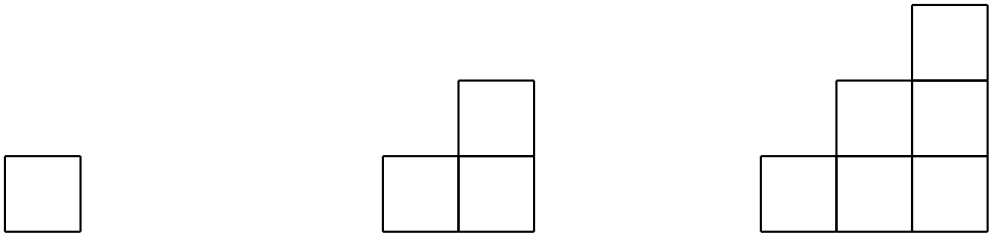
Troisième partie

Situations pour la classe proposées par des participants

8 L'escalier

(par Raphael Brakha)

On peut considérer qu'un escalier est composé de cubes juxtaposés et / ou empilés comme sur le dessin ci- dessous.



escalier à une marche | escalier à deux marches | escalier à trois marches

Exercice 1 :

Question 1 – Dessiner un escalier de 5 marches.

Question 2 – Calculer le nombre de cubes nécessaires pour fabriquer un escalier de 6 marches puis un escalier de 7 marches.

On cherche le nombre de cubes nécessaires pour fabriquer un escalier comportant 365 marches (comme dans le phare des baleines, à l'île de Ré)

Question 3 – Écrire un algorithme qui permet de calculer le nombre de cubes nécessaires pour N marches.

Question 4 – Programmer cet algorithme.

Exercice 2 :

On donne maintenant N cubes. On cherche à déterminer le nombre de marches du plus grand escalier que l'on peut ainsi construire, quitte à ce que certains cubes ne servent pas.

Question 1 – Avec 20 cubes, quel est le plus grand escalier que l'on peut construire ?

Question 2 – Écrire un algorithme qui permet de répondre à la question posée.

Question 3 – Programmer cet algorithme.

9 Introduction sur les boucles

Pré-requis : Affectations de mémoires et "si alors sinon" avec programmation sur Python.

Objectif : Premières boucles avec "pour".

Attention en Python l'instruction `for i in range(10)` donne les entiers de 0 à 9.

Exercice 1 :

Question 1 – Dérouler l'algorithme suivant et écrire les valeurs successives de A, B et S :

	A	B	S
$A \leftarrow 0$			
$B \leftarrow 1$			
$S \leftarrow A + B$			
Afficher S	...	...	...
$A \leftarrow$ B			
$B \leftarrow$ S			
$S \leftarrow A + B$			
Afficher S	...	...	...
$A \leftarrow$ B			
$B \leftarrow$ S			
$S \leftarrow A + B$			
Afficher S	...	...	...
$A \leftarrow$ B			
$B \leftarrow$ S			
$S \leftarrow A + B$			
Afficher S	...	...	...
$A \leftarrow$ B			
$B \leftarrow$ S			
$S \leftarrow A + B$			
Afficher S	...	...	...

Question 2 – L'algorithme affiche la suite de nombres : 1 ; 2 ; 3 ; 5 ; 8
– Quel serait le nombre suivant de la suite ?
– Comment est calculé chaque nombre de la suite ?
Cette suite de nombres s'appelle la "suite de Fibonacci".

Question 3 – En s'inspirant de l'algorithme ci-dessus, pour afficher les 10 premiers nombres de cette suite, il faudrait répéter 10 fois les instructions :

$A \leftarrow B$
$B \leftarrow S$
$S \leftarrow A + B$

Comment améliorer alors l'écriture de l'algorithme ?

Question 4 – Écrire l'algorithme en utilisant l'instruction

```
pour I = 1 à 10
```

Question 5 – Adapter l'algorithme précédent pour afficher les N premiers nombres de la suite de Fibonacci.

Question 6 – Saisir et tester le programme suivant :

```
N=input ("N = ")
B=1
S=1
print S
for I in range (N-1):
   A=B
   B=S
   S=A+B
   print S
```

Pour aller plus loin : Afficher le quotient $\frac{A}{B}$. Que se passe t-il quand N augmente ?

Exercice 2 : Question 1 – Que fait l'algorithme suivant :

```
S ← 0
Saisir N
Pour I = 1 à N
   S ← S + I
Afficher S
```

Question 2 – Programmer et tester un programme affichant la somme des N premiers entiers.

Séance ultérieure pour introduire le "Tant Que" : Algorithme affichant tous les termes de la suite de Fibonacci inférieurs à 500.

10 Incroyable mais vrai !...

(par Raphael Brakha)

On prend une feuille de papier ordinaire, d'épaisseur 0,2 mm. On plie cette feuille en deux, puis en deux, etc...

Exercice 1 : **Le nombre de pliages est connu.**

On se propose de déterminer l'épaisseur de la feuille à l'issue de N pliages.

Question 1 – À la main, calculer l'épaisseur de la feuille au bout de 5 pliages.

Question 2 – À votre avis, quelle est l'épaisseur de la feuille au bout de 15 pliages ?

Question 3 – Écrire un algorithme permettant de calculer l'épaisseur de la feuille au bout de N pliages.

Question 4 – Programmer cet algorithme sur machine.

Question 5 – Faire fonctionner votre algorithme pour 15 pliages et comparer le résultat obtenu avec votre estimation.

Exercice 2 : **Le nombre de pliages dépend d'une condition.**

On se propose maintenant de déterminer combien de pliages seront nécessaires pour dépasser une certaine épaisseur.

Question 1 – À votre avis, combien de pliages sont nécessaires pour que l'épaisseur dépasse la hauteur de la Tour Eiffel (330m) ?

Question 2 – Écrire un nouvel algorithme permettant de déterminer le nombre de pliages de la feuille pour que l'épaisseur dépasse une hauteur donnée (choisie par l'utilisateur)..

Question 3 – Programmer cet algorithme sur machine.

Question 4 – Faire fonctionner votre algorithme pour 330m et comparer le résultat obtenu avec votre estimation.

11 Le jeu des allumettes

(par Romain Janvier et Franck Chambon)

Prérequis : Boucles, tests, affichages mêlant texte et variables.

Deux joueurs doivent, à tour de rôle, prendre une, deux ou trois allumettes parmi celles présentes devant eux. Celui qui prend la dernière a gagné. Pour l'instant nous supposerons qu'il y a 10 allumettes au début du jeu.

Exercice 1 : Dans un premier temps, nous allons essayer de faire un jeu simple permettant à deux joueurs de s'affronter.

Question 1 – Lorsqu'un joueur décide du nombre d'allumettes qu'il veut prendre, quelles sont les conditions que doit satisfaire sa réponse ?

Question 2 – Écrire un algorithme qui fait jouer deux adversaires à tour de rôle. On pourra utiliser une variable `J` pour indiquer le joueur courant, qui vaudra 1 ou 2, et une variable `N` pour le nombre d'allumettes restantes.

Question 3 – Faire un programme en python correspondant à cet algorithme. Voici un exemple d'utilisation :

```
joueur 1 a toi de jouer
il reste 10 allumettes.
tu en prends?3
joueur 2 a toi de jouer
il reste 7 allumettes.
tu en prends?3
joueur 1 a toi de jouer
il reste 4 allumettes.
tu en prends?3
joueur 2 a toi de jouer
il reste 1 allumettes.
tu en prends?1
Bravo joueur 2 tu as gagne.
```

Pour afficher la valeur d'une variable entre deux textes sur la même ligne, on peut utiliser :

```
print "texte1",variable,"texte2"
```

Question 4 – Donner des exemples d'utilisation de votre programme pour montrer que vous tenez bien compte de toutes les conditions données dans la première question.

Exercice 2 : Nous allons maintenant essayer d'apporter quelques améliorations au programme. Ces optimisations sont optionnelles et vous pouvez, si vous le voulez, passer à l'exercice suivant sans les avoir toutes trouvées.

Question 1 – Si ce n'est pas encore le cas, modifier la partie du programme pour changer de joueur sans utiliser de test, mais uniquement une soustraction.

Question 2 – Modifier le programme pour demander le nombre d'allumettes au départ.

Question 3 – Changer l'affichage du nombre d'allumettes en affichant des barres à la place d'un nombre. Par exemple :

```
Il reste: |||||
```

Exercice 3 : Nous allons maintenant modifier le programme pour qu'un joueur puisse affronter l'ordinateur.

Question 1 – Il existe une stratégie permettant de toujours gagner à ce jeu. Afin de la trouver, il faut identifier les situations gagnantes, c'est à dire celles qui assurent la victoire du joueur dont c'est le tour. Par exemple, s'il reste une seule allumette, la victoire est garantie. Trouver deux autres situations gagnantes.

Question 2 – Donnez une situation qui n'est pas gagnante. Assurez-vous que quel que soit le nombre d'allumettes prises dans cette situation, le joueur suivant sera dans une position gagnante. On parle alors de situation perdante.

Question 3 – Pour trouver les autres situations gagnantes, on cherche celles mettant l'autre joueur dans une situation perdante. Dire pour toutes les situations de moins de 15 allumettes si elles sont perdantes ou gagnantes.

Question 4 – Comment faire, lorsqu'un joueur est dans une situation gagnante pour mettre l'autre joueur dans une situation perdante ?

Question 5 – Faire l'algorithme qui correspond à un tour de jeu de l'ordinateur, en appliquant la stratégie au dessus. S'il se trouve dans une situation perdante, il pourra par exemple prendre une seule allumette.

Question 6 – Modifier votre programme pour que le rôle du joueur 2 soit tenu par l'ordinateur.

Question 7 – Changer le programme pour demander au début de la partie le nombre maximum d'allumettes pouvant être prises en un tour. Que faut-il modifier dans la stratégie de l'ordinateur pour tenir compte de ce changement ?

Remarque – Peut-être qu'il faut donner la stratégie gagnante aux élèves pour leur permettre de ne pas être bloqués. Ou alors on peut laisser plusieurs jours (semaines ?) aux élèves pour trouver la stratégie gagnante.

Remarque – On peut aussi imaginer organiser un tournoi entre les élèves, avec de vraies allumettes. Il est fort probable que le gagnant trouve ou connaisse la stratégie gagnante. Il peut alors l'expliquer aux autres élèves. Ou alors les élèves peuvent affronter le professeur afin de tester leurs stratégies. On peut poser la limite d'une tentative par élève pour les obliger à réfléchir avant de se lancer à répondre.

12 Fabrication de portes

(par Bernadette Oury-Rabourdin)

Ce qui suit est plutôt une suite d'activités pour introduire petit à petit les différentes notions d'algorithmique du programme en enrichissant progressivement une situation initiale. (Chaque notion étant ensuite retravaillée avec des exercices s'intégrant dans les chapitres étudiés en cours).

Idée de départ : si on programme une fonction f dans l'éditeur de fonction de la calculatrice et qu'on lui donne le réel x, elle nous renvoie $f(x)$. Comment pourrait-on faire pour avoir la même chose avec une fonction de deux variables x et y ?

Une situation : un artisan fabrique des portes en hêtre et des portes en chêne. Il met 3,5h pour fabriquer une porte en hêtre et 2,5h pour fabriquer une porte en chêne. Il fabrique x portes en hêtre et y portes en chêne par semaine.

Exercice 4 : On s'intéresse au temps de travail hebdomadaire de l'artisan.

Question 1 – L'artisan fabrique 5 portes en hêtre et 3 portes en chêne une certaine semaine. Quel est son temps de travail pour cette semaine ?

Question 2 – L'artisan fabrique x portes en hêtre et y portes en chêne une certaine semaine. Exprimer, en fonction de x et de y, son temps de travail pour cette semaine.

Question 3 – Écrire un algorithme qui permette de calculer le temps de travail hebdomadaire de l'artisan, connaissant le nombre de portes en hêtre et celui de portes en chêne à fabriquer durant une semaine.

Question 4 – L'artisan a une contrainte : son temps de travail ne doit pas dépasser 40h par semaine. Écrire un algorithme qui lui permette de savoir s'il peut fabriquer dans la semaine le nombre de portes en hêtre et le nombre de portes en chêne qu'il a en commande.

Exercice 5 : L'artisan a une deuxième contrainte : les portes sont toutes livrées en fin de semaine aux clients et il ne peut pas stocker plus de 13 portes dans son atelier.

Question 1 – Écrire une inéquation traduisant cette contrainte.

Question 2 – Modifier l'algorithme précédent pour qu'il permette à l'artisan de savoir s'il peut fabriquer dans la semaine le nombre de portes en hêtre et le nombre de portes en chêne qu'il a en commande.

Exercice 6 : La vente d'une porte en hêtre rapporte 90 euros et que celle d'une porte en chêne rapporte 80 euros. L'artisan souhaite savoir combien de portes de chaque sorte il doit fabriquer chaque semaine pour avoir un bénéfice maximum (en respectant, bien sûr, ses contraintes de

fabrication).

Question 1 – Dans cette question, on ne tient pas compte de la contrainte sur le temps de travail. Écrire un algorithme qui, pour un nombre de portes en chêne fabriquées donné, permette à l'artisan de calculer le bénéfice fait en fonction du nombre de portes en hêtre fabriquées. Attention : ne pas oublier la contrainte de stockage !

Question 2 – Comment modifier l'algorithme pour que l'artisan puisse obtenir les bénéfices dans tous les cas de fabrication possible ?

Question 3 – Comment modifier l'algorithme précédent pour tenir compte aussi de la contrainte sur le temps de travail ?

Question 4 – Quel est le bénéfice maximum ? Pour combien de portes de chaque type est-il obtenu ?

13 Le lièvre et la tortue

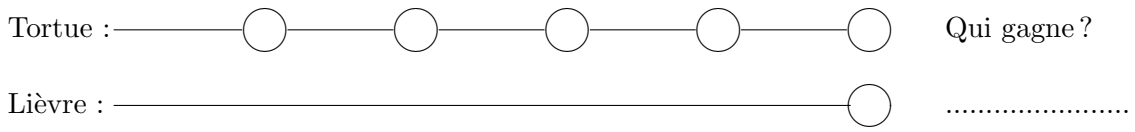
(par Hélène Langlais, Julien Duval, Martine Piétu, Dominique Callies)

Règle du jeu : 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 de 5 cases.

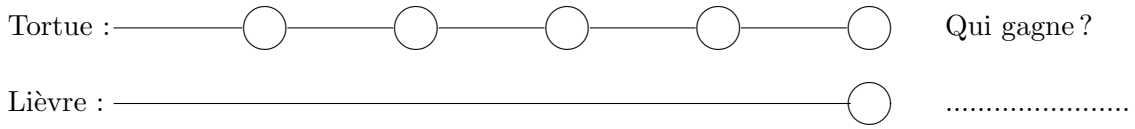
L'objectif est de déterminer si le jeu est à l'avantage du lièvre ou de la tortue ?

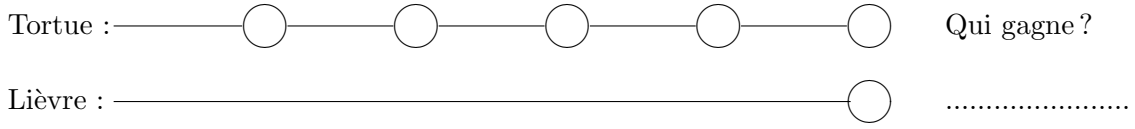
Exercice 1 : **On joue**

Question 1 – Lancer le dé, et remplir le schéma ci-dessous :



Question 2 – Faire deux autres parties.





Question 3 – A votre avis, le jeu est-il à l'avantage du lièvre ou de la tortue ?

Exercice 2 : **Algorithme**

Question 1 – Compléter l'algorithme ci-dessous, écrit en langage naturel :

```
Les variables:
   PositionTortue
   PositionLièvre
   Dé
Initialisation:
   PositionTortue = 0
   PositionLievre = 0
 Course:
   Tant que PositionTortue < ..... et ............................ , alors:
      Dé = nombre entier aléatoire entre 1 et 6
      Si Dé = ..... alors:
         PositionLievre = .....
      Sinon:
         PositionTortue = ............................
Résultat:
   Si PositionLievre = 5 alors:
      Afficher ............................
   Sinon:
      Afficher ............................
```

Question 2 – Traduire en "Python" l'algorithme précédent et le tester.

Exercice 3 : **Fluctuation d'échantillonnage**

Question 1 – Avec une boucle "while", modifier l'algorithme précédent pour simuler 1000 parties.

Question 2 – Compter le nombre de parties gagnées par le lièvre et par la tortue et afficher ces résultats.

Question 3 – Faire "tourner" l'algorithme précédent plusieurs fois et remplir le tableau ci-dessous :

Échantillon numéro	Nombre parties gagnées par le lièvre	Nombre parties gagnées par la tortue
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Question 4 – A votre avis, le jeu est-il à l'avantage du lièvre ou de la tortue ?

Exercice 4 : **Probabilité**

Question 1 – A l'aide d'un arbre, déterminer le probabilité que la tortue gagne.

Question 2 – En déduire la probabilité que le lièvre gagne.

Question 3 – Comparer les résultats avec ceux obtenus dans l'exercice 3.

Exercice 5 : **Modifications possibles**

Question 1 – On peut modifier les règles du jeu et décider que la tortue avance de 2 (ou 3, ou...) cases à la fois.

Question 2 – On peut également modifier le nombre de cases à parcourir par la tortue.

Quatrième partie

Python : le strict nécessaire

Traduire un algorithme dans un langage de programmation est nécessaire : l'objectif d'un algorithme conçu par « un être humain » est de faire exécuter de façon automatique ces opérations à une « machine ».

Le passage sur machine permet de constater que l'algorithme a été bien conçu, il donne un retour positif, concret du travail théorique effectué. C'est important d'un point de vue pédagogique en particulier pour les élèves.

Le choix d'un langage est en même temps important et pas important. Nous avons trouvé en PYTHON un langage « simple » :

– une interface utilisateur réduite à 2 ou 3 fenêtres,
– une syntaxe impliquant l'écriture d'un code lisible proche d'un langage algorithmique naturel,
– l'utilisation de très peu d'instructions, de mot-clés...

De plus PYTHON est un logiciel libre que chacun pourra se procurer et utiliser, et ceci sous n'importe quel environnement WINDOWS, LINUX, MACOS...

Modes opératoires de Python

Python possède deux modes opératoires :
– un mode interactif,
– travail à partir d'un programme écrit dans un fichier.
Travailler à partir d'un programme écrit dans un fichier nous semble plus adapté à un apprentissage cohérent de l'algorithmique qui consiste à considérer une séquence d'instructions dans son ensemble. Nous constaterons que les programmes écrits feront au plus une dizaine ou une vingtaine d'instructions. Il est préférable d'utiliser l'extension **.py** pour le fichier qui contient le programme. Ceci permet à l'éditeur de "reconnaître" qu'il s'agit d'un programme écrit en Python et d'utiliser si possible la coloration syntaxique.

Python propose un environnement permettant de disposer en même temps :
– de la fenêtre « interactive » (Python Shell) : elle permettra d'exécuter éventuellement des instructions isolées, mais surtout d'entrer des données, de sortir des résultats des calculs ;
– d'une fenêtre correspondant à un « éditeur de texte » intégré (éditeur IDLE) qui proposera une façon très ergonomique d'écrire le programme et d'exécuter directement ce programme à partir de l'éditeur.
Apprendre à travailler avec ces deux fenêtres à l'écran procurera un confort que chacun appréciera.

Commenter un programme

Dans l'écriture d'un programme Python tout texte précédé du signe # et ceci jusqu'à la fin de ligne ne sera pas pris en compte.

Instruction d'affectation

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

Instructions « entrée/sortie »

On utilisera les instructions `input()` et `print`.

```
# Entrée et sortie d'un nombre sans "texte"
A = input()
print A
# Entrée et sortie d'un nombre avec "texte"
A = input("A = ")
print "A = ", A
```

Instruction conditionnelle et tests logiques

Tout d'abord considérons l'instruction conditionnelle simple « si » correspondant aux différents schémas algorithmiques :

```
Si condition alors Instruction
```

```
Si condition Alors
Instruction1
Instruction2
...
Finsi
```

Pour isoler l'ensemble des instructions on peut les « décaler vers la droite » : indenter.

```
Si condition Alors
   Instruction1
   Instruction2
   ...
```

A ce moment il est inutile de faire figurer le « finsi ». C'est ce que fait l'éditeur Python après le symbole « : »

```
if condition:
   instruction1
   instruction2
   ...
# Suite du programme
```

Exemple de programme Python

```
# Calcul de la valeur absolue
X = input("Entrez un nombre: ")
print "Nombre entré: ",X

# On examine si X est positif ou nul
if X >= 0:
   Y = X
# On examine si X est strictement négatif
if X < 0:
   Y = -X
print "Valeur absolue du nombre: ", Y
```

On considère maintenant l'instruction conditionnelle « si, alors, sinon » correspondant au schéma algorithmique

```
Si condition alors
   Instruction1
   Instruction2
   ...
sinon
   Instruction3
   Instruction4
   ...
```

Exemple de programme PYTHON

```
# Calcul de la valeur absolue
X = input("Entrez un nombre: ")
print "Nombre entré: ",X
# On examine si X est positif ou nul et on fait Y = X
# et dans le cas contraire Y = - X
if X >= 0:
   Y = X
else:
   Y = -X

print "Valeur absolue du nombre: ", Y
```

En marge des instructions conditionnelles nous avons rencontré des « tests logiques » : ce sont des expressions prenant des valeurs « vrai » ou « faux ».

A égal B s'écrira en Python `A == B` pour distinguer de l'affectation `A = B` .

A inférieur ou égal, ou strictement inférieur à B s'écriront en Python

```
A <= B
A < B
```

A supérieur ou égal, ou strictement supérieur à B s'écriront en Python

```
A >= B
A > B
```

A différent de B `A != B`

Listes et opérations liées

Qu'est-ce qu'une liste : c'est un ensemble de N éléments numérotés de 0 à $N-1$. L'accès à l'élément i de la liste L se fera par $L[i]$.

Exemple :

```
# Liste de 5 entiers
L = [0,1,2,3,4]
# Affichage de la liste en une seule instruction:
# c'est la liste en tant que telle entre [ ]
print L
# Affichage de chacun des éléments de la liste
# c'est la succession de chacun des éléments de la liste
print L[0],L[1],L[2],L[3],L[4]
```

Une liste particulière : l'instruction `range (N)` permet de créer la liste des N premiers entiers de 0 à $N-1$.

```
# Entrer un entier
N = input()
L = range(N)
# Affichage de la liste
print L
```

Des opérations bien utiles : l'instruction $L + L$ construit la liste consistant à mettre L deux fois l'une après l'autre. L'instruction $L * 4$ met 4 fois bout à bout la liste L.

```
# Entrer un entier
N = input()
L = [0]
# On reproduit N fois la liste L
L = L*N
# Affichage de la liste
print L
```

Explorer successivement les éléments d'une liste :

L'instruction `for ...in ...` permet de parcourir tous les éléments d'une liste. On examinera par exemple le programme :

```
# Entrer un entier
N = input()
L = range(N)
print L
# Parcours de chacun des éléments de la liste et écriture de chaque élément
for i in L:
   print i
```

On peut pour chaque élément de la liste effectuer un ensemble d'instructions, par exemple :

```
# Entrer un entier
N = input()
L = range(N)
print L
# Parcours et traitement de chacun des éléments de la liste
for i in L:
   j = i*i
   print "Un nombre",i,"et son carré",j
```

Instruction itérative

Nous avons déjà décrit l'instruction `for ...in ...` à propos des listes.

Python ne possède qu'un seul autre type d'instruction itérative : elle permet de traiter tous les cas possibles.

Elle correspond au schéma algorithmique :

```
Tantque condition alors
   Instruction1
   Instruction2
...
Fintantque
```

```
Tantque condition alors
   Instruction1
   Instruction2
...
```

En voici un exemple en langage PYTHON

```
N = 5
I = 1
while I <= N:
   print I
   I = I + 1
```

Les fonctions

Il ne faut tout d'abord pas confondre avec la notion de fonction dont l'introduction fait partie du programme de mathématique.

Pour dire court, une fonction en Python sera une suite d'instructions qui seront exécutées à l'appel de la fonction.

Par exemple, dans le programme suivant, la fonction cube permet d'élever au cube le paramètre x de la fonction : le programme joint calcule les « cubes » des entiers de 0 à 10.

```
def cube (x):
   return x*x*x

for i in range(11):
   print i, cube(i)
```

Le schéma général d'une déclaration de fonction est :

```
def fct (arguments):
   corps de la fonction
   return resultat
```

Une fonction peut comporter plusieurs paramètres, et même aucun paramètre !

Les modules externes

Si nous devons utiliser des fonctions mathématiques comme sinus, cosinus,... il nous faut "importer" le module mathématique en faisant figurer au début du programme :

```
from math import*
```

Si nous voulons utiliser des générateurs de nombres aléatoires, il nous faut importer le module correspondant :

```
from random import*
```

Par exemple pour générer un entier aléatoire entre deux entiers a et b, on utilisera

```
randint(a,b)
```

Le programme suivant, après avoir construit un tableau contenant les entiers [0,1,2,..,N] lui affecte N éléments aléatoires entre 0 et 10 :

```
from random import*
N = input()
L = range(N)
i = 0
while i != N :
   L[i] = randint(0,10)
   i = i+1
print L
```

On peut disposer également (entre autre) de la loi uniforme : `uniform(a, b)` et de la loi gaussienne : `gauss(mu, sigma)`

Le module TORTUE

Le module TORTUE qu'on appellera par l'instruction

```
from turtle import*
```

est particulièrement intéressant pour l'apprentissage de l'algorithmique. Il s'agit d'un ensemble d'instructions de dessins élémentaires. Dans un premier temps les instructions suivantes suffiront :

`reset()`	effacer le dessin
`goto(x, y)`	aller à l'endroit de coordonnées x, y
`forward(distance)`	avancer d'une distance donnée
`backward(distance)`	reculer d'une distance donnée
`up()`	relever le crayon (pour ne plus dessiner)
`down()`	abaisser le crayon (pour dessiner)
`color(`*couleur*`)`	<*couleur*> est une chaîne : 'red', 'blue', 'green',...
`left(angle)`	tourner à gauche d'un angle exprimé en degrés
`right(angle)`	tourner à droite d'un angle exprimé en degrés
`xcor()`	renvoyer l'abscisse de la position de la tortue
`ycor()`	renvoyer l'ordonnée de la position de la tortue
`seth(angle)`	positionner la tête de la tortue de l'angle "angle" avec l'horizontal
`heading()`	retourner la position de la tête de la tortue

Attention : par défaut, pour la tortue les angles sont donnés en degré.

Attention : pour que l'utilisateur puisse fermer la fenêtre de la tortue suivant le processus usuel il faut achever le programme par l'instruction :

```
mainloop()
```

Le programme suivant combine l'utilisation de la notion de fonction et un dessin à l'aide du module turtle :

```
from turtle import*

def segment (x1,y1,x2,y2):
   up()
   goto (x1,y1)
   down ()
   goto (x2,y2)

color('red')
segment (0,0,100,100)
color('green')
segment (100,100,100,200)

mainloop()
```