Algorithmique et programmation Python

David TOCAVEN

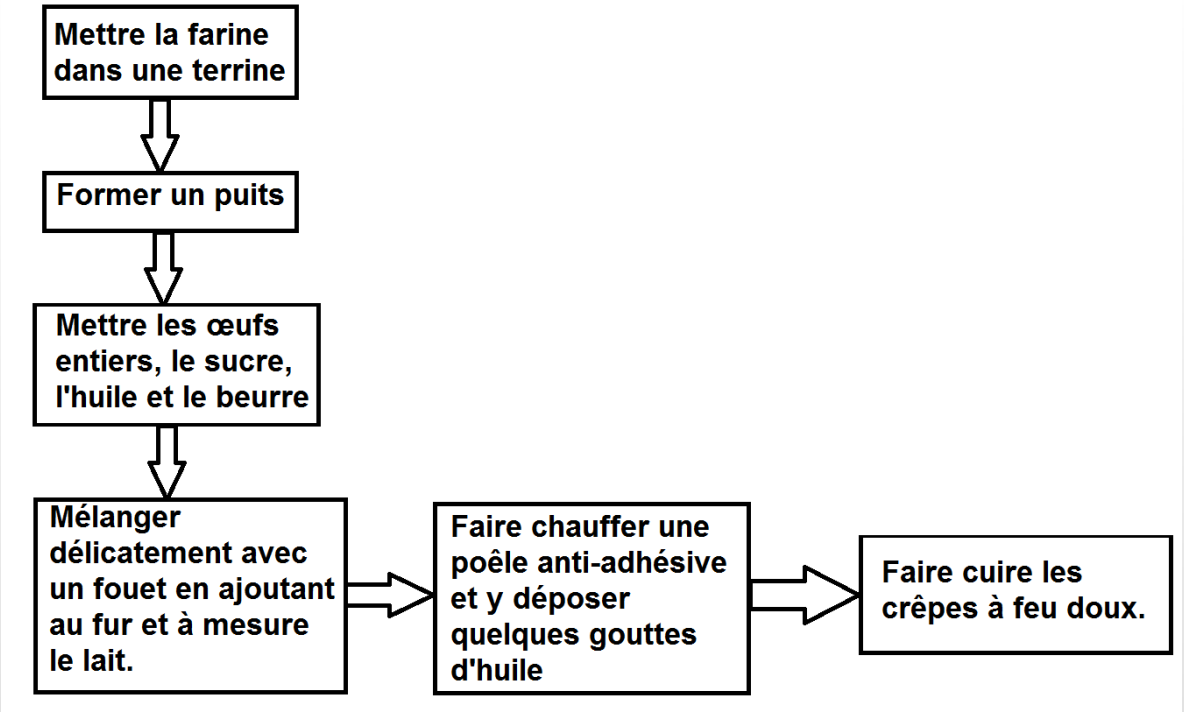
2022

Table des matières

1 Introduction

Afin de (re)découvrir le concept d'algorithme, voici deux activités qui présentent des algorithmes auquels on ne pense pas.

Activité 1 : Les crêpes



Sources : https://sweetrandomscience.blogspot.com/2014/01/quest-ce-quun-algorithme-explication.html

Questions

1. À quoi servent ces instructions ?
2. De quoi a-t-on besoin pour les réaliser ? (souligne les)
3. Quelles sont les différentes actions nécessaires ? (entoure les)

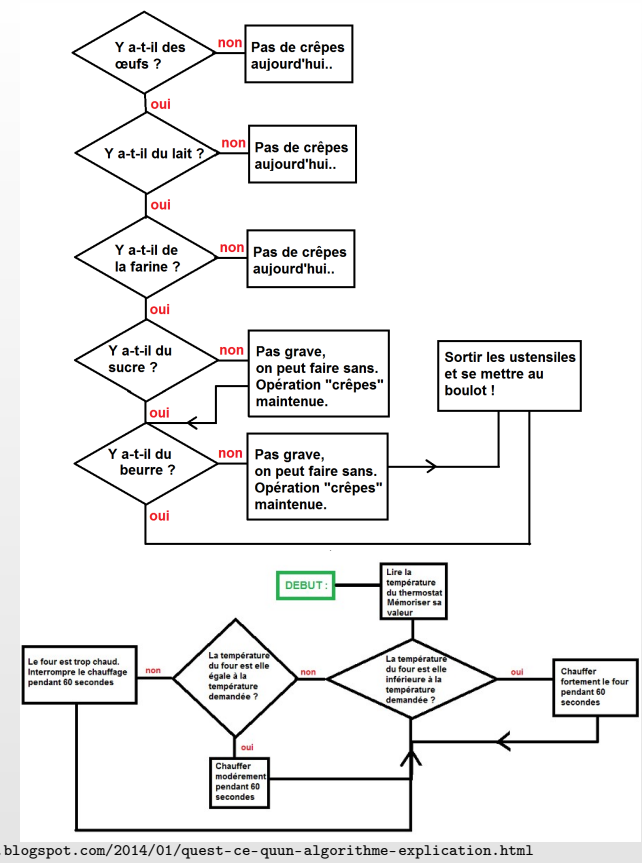
Réponses aux questions :

1. *Elles servent à cuisiner des crêpes.*
2. *Nous avons besoin de farine, d'œufs, de sucre, d'huile, de beurre et de lait.*
3. *Pour réaliser des crêpes les actions sont mettre, former un puit, mettre, mélanger délicatement, ajouter au fur et à mesure, faire chauffer, déposer quelques gouttes et faire cuire à feu doux.*

Définition d'un algorithme

Une suite finie et non-ambiguë d'opérations et d'instructions servant à résoudre une classe de problèmes.

Activité 2 : D'autres algorithmes



Sources : https://sweetrandomscience.blogspot.com/2014/01/quest-ce-quun-algorithme-explication.html

Questions

1. À quoi l'algorithme B sert-il ?
2. Quelle est la différence entre l'algorithme A et celui de l'activité 1 (voir fig. 1, page 2) ?
3. Quelle est la différence entre l'action réalisée par les losanges et les rectangles ? (considérez A et B)
4. Quels sont les blocs de départ et de fin ? (Pour A et B, séparément)
5. (Extra) Si vous aviez à relier l'algorithme de l'activité 1 et l'algorithme A par une seule flèche, entre quels blocs la placeriez-vous ?

Réponses aux questions :

1. *L'algorithme B sert à réguler la température d'un four.*
2. *L'algorithme A inclut une dimension de choix (questions fermées : oui/non) concernant la disponibilité des ingrédients que l'agorithme de l'activité 1 n'a pas.*
3. *Les losanges servent à exprimer des questions fermées (conditions), à effectuer des tests. Tandis que les rectangles servent à effectuer des actions (instructions).*
4. *Pour l'algorithme A, le bloc de départ est le losange "Y a-t-il des œufs ?" et ceux de fin sont les trois rectangles "Pas de crêpes aujourd'hui.." et celui "Sortir les ustensiles et se mettre au boulot!".*
5. *Elle serait entre les rectangles 'Sortir les ustensiles et se mettre au boulot!' de l'activité 1 et 'Mettre dans la farine dans une terrine' de l'algorithme A.*

2 Les variables

2.1 Définition

Dans un algorithme, il est souvent (presque systématiquement) nécessaire de manipuler des informations. Pour cela, on doit stocker les données, on les enregistre dans des variables.

Définition d'une variable

Une variable est l'association d'une étiquette et d'une boîte pouvant contenir une donnée.

Les principaux types de données que l'on manipule sont :

— Des entiers $\mathbb{Z}$ (ex : $-1;1;0;125$)

— Des Réels $\mathbb{R}$ (ex : $-1,55;1,3333\ldots;0,000000023;\pi;47$). Ils sont approximés dans des flottants.

— Des caractères (ex : 'c';'a';'#';'(';'1')

— Des booléens (binaire) (Prend la valeur Oui ou Non, 1 ou 0, $TRUE$ ou $FALSE$)

Une variable est décrite par différents éléments :

— Son nom, il doit être sans espace ou accentuation. (ex : pour le nombre de roues d'un véhicule, 'nombre_roue' ou 'nombreRoue' .)

— Son type (ex (suite) : un entier)

— Sa plage de valeur. (ex (suite) : entre 0 et 100.)

— Sa valeur. (ex (suite) : 4)

Remarque : lorsque l'on parlera de languages de programmation, la plage de valeur sera précisée dans le type et la variable aura également un emplacement dans la mémoire du système informatique.

2.2 Création et affectation de variable

Maintenant que nous savons ce que sont les variables, nous allons voir comment faire pour en créer et stocker des informations dedans. Avant d'utiliser une variable, il faut la créer. C'est ce que l'on appelle *la déclaration* d'une variable. La façon de l'écrire (syntaxe) dépend du langage de programmation.

L'opération qui consiste à stocker une information dans une variable s'appelle *l'affectation* de variable.

Voici un exemple :

Exemple 1 : Déclaration et affectation d'une variable

Algorithme :
Entier (0;100) : nombre_roue ;
début
| nombre_roue ⟵ 4 ;
fin

⚠ IMPORTANT !

En algorithmique, toutes les lignes où l'on affecte une variable ou appelle une fonction doivent se terminer par un point virgule (' ;'). Cela permet de savoir où se terminent les lignes.

3 Les expressions

Elles sont des successions de calculs utilisées dans tous l'algorithme : dans les affectations, les paramètres des fonctions et les structures de contrôles.

Elles peuvent-être :

— Une valeur (ex : 12)

— Une variable (ex : $nombre\_roue$)

— Une constante

— Un appel de fonction (Voir la fonction *Lire()*, page 6)

— <expression> **Opérateur** <expression> (ex : $4 + x$)

— **Opérateur_Unaire** <expression> (ex : $nonA$)

4 Les commentaires

Lorsque l'on écrit des algorithmes, il est important d'expliquer ce qu'il s'y passe. Pour cela, nous ajoutons des explications écrites qui ne modifie pas le comportement de l'algorithme, ce sont des *commentaires*. Un commentaire explique synthétiquement (de façon courte et précise) et sans abréviations incompréhensibles l'élément le plus proche pour les gens qui lisent le code. Le commentaire peut être placé seul sur une ligne ou à la fin du code d'une ligne (après le ' ;'). Tout commentaire commence par '// '. C'est un mot-clé qui a pour effet d'empêcher tout ce qui se trouve à sa droite d'être pris en compte dans l'exécution de l'algorithme.

Exemple 2 : Les commentaires

Algorithme :

```
Entier (0;100) : nombre_roue; // Le nombre de roue d'un véhicule.
début
    nombre_roue ⟵ 4; // Il s'agit d'une voiture donc 4 roues.
fin
```

5 Les fonctions de lecture et d'écriture

Une fonction est comme un outil, c'est un bout de code qui à un nom et que l'on utilise pour effectuer une action. Parfois, elle a besoin d'informations en entrée pour effectuer cette action et parfois elle renvoie des informations.

Pour faire un algorithme, il est parfois intéressant de demander à l'utilisateur une information (lecture) et d'afficher des informations (écriture). Pour cela, il existe deux fonctions : *Lire()* pour demander des informations à l'utilisateur et *Ecrire()* qui affiche des informations.

5.1 Fonction de lecture

Lorsque l'on utilise la fonction *Lire()*, on met dans les parenthèses la question à poser à l'utilisateur entre guillements (ex : "Quel âge as-tu ?") et elle renvoie l'information. Pour récupérer cette valeur, on l'affectera dans une variable.

Exemple 3 : La fonction de lecture

Algorithme :

Entier (0;100) : nombre_roue ; `// Le nombre de roue d'un véhicule.`
début
    nombre_roue ⟵ `Lire(`*"Combien votre véhicule a-t-il de roues ?"*`) ; // Lecture du nombre de roues.`
fin

5.2 Fonction d'écriture

La fonction d'écriture n'a que des entrées et pas de sortie. Il faut lui donner les différents éléments à afficher séparé par des virgules.

Exemple 4 : La fonction d'écriture

Algorithme :

Entier (0;100) : nombre_roue ; `// Le nombre de roue d'un véhicule.`
début
    nombre_roue ⟵ `4;// Il s'agit d'une voiture donc 4 roues.`
    `Écrire(`*"Votre véhicule à "* , *nombre_roue* , *" roue(s)."*`) ;`
fin

Le résultat affiché sera :

```
Votre voiture à 4 roue(s).
```

Remarque : si l'on ne mettait pas d'espaces après 'à' et avant 'roue(s)', le message serait "Votre voiture à4roue(s).". Il faut y penser !

6 Les structures de choix

Comme nous l'avons vu dans l'activité 2, il peut être intéressant de faire des choix en fonction d'une condition (question). Pour cela, nous utiliserons un structure de choix, La condition *Si - Alors - Sinon.*

6.1 L'alternative Si - Alors - Sinon

Dans une alternative *Si - Alors - Sinon*, on teste une condition et différentes actions sont réalisées suivant la valeur de cette condition (par exemple "est-ce que le véhicule à 4 roues" peut être testé par *nombre_roue == 4*). Celle-ci est booléene (elle vaut vrai ou faux). Si elle est vrai, les actions du *Si* sont réalisées. Si elle est fausse, ce seront les actions du *Sinon* qui seront effectuées. Dans aucun cas les deux ou aucunes des deux seront effectuées durant la même exécution. Le bloc *Sinon* est optionnel. Dans le cas où il ne serait pas présent, si la condition est fausse, les actions présentent dans le bloc *Si*, ne sont pas réalisées.

Exemple 5 : L'alternative *Si - Alors - Sinon*

Algorithme :

```
Entier (0;100) : nombre_roue; // Le nombre de roue d'un véhicule
début
    nombre_roue ⟵ 4; // Il s'agit d'une voiture donc 4 roues
    si (nombre_roue == 4) alors
        // Cas 'peut-être une voiture'.
        Écrire("C'est peut-être une voiture car le véhicule a quatre roues.");// Les
            camions ont parfois quatre roues.
    sinon
        // Cas 'pas une voiture'.
        Écrire("Ce n'est pas une voiture car le véhicule n'a que deux roues. ");
    fin
fin
```

Le résultat affiché sera :

```
C'est peut-être une voiture car le véhicule a quatre roues.
```

Remarques : Il y a plusieurs avantages à utiliser la condition Sinon *lorsque cela est possible : C'est plus rapide à écrire et c'est plus clair à lire que deux conditions* Si.

7 Les répétitions

Dans cette partie, nous allons découvrir qu'il est possible d'effectuer plusieurs fois la même action sans recopier l'action à faire plusieurs fois. On verra aussi qu'il existe plusieurs façons de choisir quand on arrête de répéter l'action. Nous allons commencer par la structure de répétition *Tant - que*.

7.1 La répétitive Tant Que

Cette structure de répétition permet d'effectuer différentes actions, écritent entre le début *tant que <condition booléenne> faire* et la fin de la structure *fin*. Le moment où l'on arrête de répéter l'action est lorsque la condition <*condition booléenne*> est fausse. Si elle est fausse dès la première fois où l'on y rentre, on effetue jamais les actions dans la structure.

Exemple 6 : La répétitive *Tant que*

Algorithme :

```
Entier (0;100) : nombre_roue_secret; // Le nombre de roue d'un véhicule
    secret que l'on souhaite trouver
Entier (0;100) : nombre_roue_trouve; // Le nombre de roue du véhicule secret
    que l'on a trouvé
début
    nombre_roue_secret ⟵ 4; // Il s'agit d'une voiture
    nombre_roue_trouve ⟵ 0; // On commence à zéro car c'est la valeur
        minimale possible.
    tant que nombre_roue_trouve < nombre_roue_secret faire
        Écrire("Le véhicule n'a pas ", nombre_roue_trouve, " roue(s).");
        // affiche un message : pas encore le bon nombre de roues
        nombre_roue_trouve ⟵ nombre_roue_trouve + 1;
        // on augmente la valeur de nombre_roue_trouve de 1
    fin
    Écrire("Le véhicule a ", nombre_roue_trouve, " roue(s).")
fin
```

Le résultat affiché sera :

```
Le véhicule n'a pas  0  roue(s).
Le véhicule n'a pas  1  roue(s).
Le véhicule n'a pas  2  roue(s).
Le véhicule n'a pas  3  roue(s).
Le véhicule a  4  roue(s).
```

Remarque : Lorsque nombre_roue_trouve *rentre dans la boucle pour la première fois, il a la valeur* 0. *Comme* 0 *est inférieur à* nombre_roue_secret *qui vaut* 4, *la condition est vrai. Donc l'exécution se poursuit dans la structure* tant que. *Le message s'affiche puis* nombre_roue_trouve *augmente de un. Elle vaut maintenant 1. La fin de la boucle étant atteinte, on reste la condition. Elle est toujours vrai (*1<4*). On affiche à nouvelle le message et* nombre_roue_trouve *vaut* 2 *en fin de boucle. On teste la condition à nouveau : elle est toujours vrai* 3<4. *On affiche le message et ajoute* 1 *à* nombre_roue_trouve. *Maintenant* nombre_roue_trouve *vaut* 4. *On teste à nouveau la condition :* 4<4 ? *Non, donc on passe aux instructions en dessous de* fin *de la boucle tant que. On affiche le message et l'exécution de l'algorithme est finie.*

⚠ IMPORTANT !

Si la condition est toujours vrai, les opérations dans la structure s'effectuerons indéfiniment... Cela peut-être souhaité (comme dans l'algorithme B de activité 2) ou une source de problème.

7.2 L'itérative Pour

Comme précédemment, il s'agit d'une structure permettant de répéter une suite d'instructions mais cette fois ci, les opérations s'effectuent un nombre donné de fois au lieu d'être lié à une condition.

Exemple 7 : L'itérative *Pour*

Algorithme : Afficher n nombres

```
Entier (0;100) : nombre_actuel; // La valeur actuelle que l'on veut afficher.
début
    nombre_actuel ⟵ 0; // On commence à zéro par sécurité.
    pour nombre_actuel allant de 0 à 5 faire
        Écrire(nombre_actuel);
        // affiche la valeur du nombre actuel
        nombre_actuel⟵ nombre_actuel + 1;
    fin
fin
```

Le résultat affiché sera :

```
0
1
2
3
4
```

Remarque : on peut voir que le résultat s'arrête à 4. Cela est dût au fait que la condition que teste la boucle est nombre_actuel < 5.

8 Les procédures et fonctions

Parfois, il est intéressant de réutiliser plusieurs fois le même bout de code à différents endroits. Pour cela, deux concepts ont été créer en algorithmique : Les *procédures* et les *fonctions*.

8.1 Les procédures

Par exemple si l'on veut afficher la moyenne d'une classe de seconde, puis de toutes les classes de secondes mais aussi toutes celle de premières et également celles de terminales. Pour éviter de devoir recopier le code qui calcule affiche la phrases et la moyenne partout dans le code, on peut en faire une *procédure*. C'est un bout de code qui effectue une ou des opérations précises. Elle a un nom et *paramètres*. Les paramètres sont les informations (variables) que l'on va lui donner en entrées. Dans l'exemple qui suit, on a créer une fonction qui affiche le nom et la moyenne de ce que l'on souhaite. Ici, on a choisi d'afficher les moyennes de trois classes, les secondes 201, 202 et 203.

Exemple 8 : Les procédures

Algorithme : fonction afficher_moyenne

Réel (0;20) : valeurMoyenne; `// La valeur de la moyenne que l'on veut afficher.`

chaîne de carartères[100] : nomMoyenne; `// Le nom de la moyenne que l'on veut afficher.`

Entrées : nomMoyenne, valeurMoyenne

début

Écrire(*"La moyenne de "* + *nomMoyenne* + *" est : ",valeurMoyenne,"/20"*)

fin

Algorithme : Algorithme principal

Réel(0;20) :eleves_0_201, eleves_1_201, eleves_2_201, classe_201, classe_202, classe_203; `// Les variables qui contiennent les moyennes, respectivement, des élèves 0, 1 et 2 de la 2nd01, ainsi que des classes 2nd01, 2nd02 et 2nd03.`

début

```
// Affectations des moyennes des élèves de la 2nd01.
eleve_0_201 ⟵ 10;
eleve_1_201 ⟵ 8;
eleve_2_201 ⟵ 17;
// Affectations des moyennes des classes 2nd02 et 2nd03.
classe_202 ⟵ 12;
classe_203 ⟵ 10;
// Calcul de la moyenne de la 2nd01
classe_201 ⟵ (eleves_0_201 + eleves_1_201 + eleves_2_201) / 3 ;
// Affichage de la moyenne de la 201
afficher_moyenne("la classe 201",classe_201);
// Affichage de la moyenne de la 202
afficher_moyenne("la classe 202",classe_202);
// Affichage de la moyenne de la 203
afficher_moyenne("la classe 203",classe_203);
```

fin

Le résultat affiché sera :

```
La moyenne de la classe 201 est :  11.666666666666666 /20
La moyenne de la classe 202 est :  12 /20
La moyenne de la classe 203 est :  10 /20
```

Remarque : La fonction pour afficher des messages Ecrire(..) *est une procédure.*

8.2 Les fonctions

Les fonctions sont très proches des procédures, elles ont juste en plus des sorties. C'est-à-dire qu'elles peuvent renvoyer une valeur une fois qu'elles ont fini de s'exécuter. Par rapport à l'exemple

précédent, il pourrait être pratique d'avoir une fonction qui calcule la moyenne entre trois valeurs. Grâce à cela, on pourrait calculer simplement la moyenne entre les trois élèves de 2nd01 mais aussi la moyenne entre les trois classes de 2nd. Dans l'exemple suivant, nous ne redonnerons pas la définition de la procédure *afficher_moyenne()*, présentée à l'exemple précédent (8, page 10).

Exemple 9 : Les fonctions

Algorithme : fonction moyenne3valeurs

Réel (0;20) : val1, val2, val3, valMoy; `// Respectivement, les trois valeurs dont ont veut calculer la moyenne et la valeur de la moyenne.`

Entrées : val1, val2, val3

Sorties : valMoy

début

    Écrire("La moyenne de " + nomMoyenne + " est : ",valeurMoyenne,"/20")

fin

Algorithme : Algorithme principal

Réel(0;20) :eleves_0_201, eleves_1_201, eleves_2_201, classe_201, classe_202, classe_203, classes_secondes; `// Les variables qui contiennent les moyennes, respectivement, des élèves 0, 1 et 2 de la 2nd01, des classes 2nd01, 2nd02 et 2nd03 et de toutes les classes de secondes.`

début

```
    // Affectations des moyennes des élèves de la 2nd01.
    eleve_0_201 ⟵ 10;
    eleve_1_201 ⟵ 8;
    eleve_2_201 ⟵ 17;
    // Affectations des moyennes des classes 2nd02 et 2nd03.
    classe_202 ⟵ 12;
    classe_203 ⟵ 10;
    // Calcul de la moyenne de la 2nd01
    classe_201 ⟵ moyenne3valeurs(eleve_0_201 , eleve_1_201 , eleve_2_201) ;
    // Calcul de la moyenne de toutes les classes de secondes
    classes_seconde ⟵ moyenne3valeurs(classe_201,classe_202,classe_203);
    // Affichage de la moyenne de la 201
    afficher_moyenne("la classe 201",classe_201);
    // Affichage de la moyenne de toutes les classes des secondes
    afficher_moyenne("toutes les classes de secondes",classes_secondes);
```

fin

Le résultat affiché sera :

```
La moyenne de la classe 201 est 11.67/20
La moyenne de toutes les classes de secondes est 11.22/20
```

Remarques :

— *Dans le programme principal, les variables que l'on utilisent comme paramètres d'une fonction, comme par exemple* classe_201 *dans le dernier appel de* moyenne3valeurs()*, sont copiées dans*

une variable d'entrée de la fonction. Dans notre exemple, dans val1. *Cela veut dire que si l'on modifie la valeur de val1 dans la fonction, cela n'aura pas d'incidence sur celle de* classe_201 *dans le programme principal.*

— *Toutes les variables qui sont déclarés dans une fonction n'existent que dans celle-ci. Ex : val1 n'existe pas dans l'algorithme principal. Lorsque l'on appelle une fonction, elle crée les variables dont elle a besoin, les utilisent et lorsque l'on arrive à la fin et qu'elle renvoie le résultat, elle détruit toutes les variables. Il en va de même pour les procédures.*