ALGORITHMIE & Les bases du langage PYTHON

Cours / TD

1. PARTIE 1 : Algorithmie : pseudo code et algorigramme

1.1. Algorithmie

1.1.1. Définition

Avant toute programmation, il est recommandé d'avoir une visualisation du programme qu'on va faire. Pour cela, il faut écrire ce que doit faire l'algorithme. Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème.
En programmation, le pseudo-code, également appelé LDA (pour Langage de Description d'Algorithmes) est une façon d'exprimer clairement et formellement un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier.
L'écriture en pseudo-code permet souvent de bien prendre toute la mesure de la difficulté de la mise en œuvre de l'algorithme, et de développer une démarche structurée dans la construction de celui-ci. En effet, son aspect descriptif permet de décrire avec plus ou moins de détail l'algorithme, permettant de ce fait de commencer par une vision très large et de passer outre temporairement certains aspects complexes, ce que n'offre pas la programmation directe. Ce langage est près d'un langage de programmation comme Pascal, C# ou C++, sans être identique à l'un ou à l'autre.
Bien qu'il n'existe pas de réelle norme pour le pseudo-code (mais plutôt des conventions, des habitudes), Il exprime des idées formelles dans une langue près du langage naturel de ses usagers (pour nous, le français) en lui imposant une forme rigoureuse. Quelques règles :

- Un algorithme doit débuter par l'instruction DÉBUT. Cette instruction indique le point de départ de l'exécution de l'algorithme. Réciproquement, l'instruction FIN indique le point où l'algorithme se termine.
- Le nom d'une variable ou d'une constante doit être significatif. On devrait savoir immédiatement, à partir de son nom, à quoi sert la variable ou la constante, et quel sens donner à sa valeur.
- Les majuscules et les minuscules sont des symboles distincts dans la plupart des langages de programmation, mais pas tous. Ainsi, pour éviter les ennuis, ne donnez pas à deux entités (une variable et une constante, par exemple) des noms qui ne différeraient que sur cet aspect.
- Les instructions se font une ligne à la fois (pas de ';' en pseudocode).
- On ne se préoccupe pas des types des variables et des constantes (ce principe n'est pas universel).
- Les opérations de base sont LIRE, ECRIRE et ← (affectation d'une valeur à une variable). Les opérations de base et/ou mots clés doivent être écrits en gras ou en majuscules.

Pour cela nous allons utiliser le logiciel LARP (acronyme «Logiciel d'Algorithmes et de Résolution de Problèmes»).

L'avantage de LARP est que le programme est un langage pseudo-code à syntaxe flexible et non un code source à compiler, ce qui permet de formuler des algorithmes en un langage semi-naturel plutôt que de devoir adhérer à une syntaxe rigide et cryptique telle que celle des langages de programmation traditionnels

✍ Exercice 1 : Corriger le pseudo code suivant :

```
DÉBUT
a <- 4
LIRE b
c <- a + b ;
ECRIRE C
```

.\scripts\part1\definition.larp

Nous verrons que le précédent pseudo code se traduira en langage de programmation Python par :

```
a = 4
b = 0
b=int(input(b))
c = a + b
print(c)
```

1.2. Les variables

1.2.1. A quoi servent les variables ?

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs. Il peut s'agir de données issues du disque dur, fournies par l'utilisateur (frappées au clavier), ou que sais-je encore.
Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. Ces données peuvent être de plusieurs types : elles peuvent être des nombres, du texte, etc. Toujours est-il que dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une variable.

Pour employer une image, une variable est une boîte, que le programme (l'ordinateur) va repérer par une étiquette. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.
En réalité, dans la mémoire vive de l'ordinateur, il n'y a bien sûr pas une vraie boîte, et pas davantage de vraie étiquette collée dessus. Dans l'ordinateur, physiquement, il y a un emplacement de mémoire, repéré par une adresse binaire. Si on programmait dans un langage directement compréhensible par la machine, on devrait désigner nos données par des 10011001 et autres 01001001.
Les langages informatiques plus évolués se chargent, entre autres rôles, d'épargner au programmeur la gestion fastidieuse des emplacements mémoire et de leurs adresses.

1.2.2. Déclaration des variables et affectations

TYPES	REMARQUES
Booléens	Vrai ou faux
Caractère	Symboles typographiques
Chaine de caractères	Ensemble de caractères entre " "
Entier	Entiers relatifs
Flottant	"Utilisé pour les réels"

Il existe d'autres types de variables. Le fait de déclarer en pseudo code le type de la variable n'est pas une obligation, mais vous le verrez dans de nombreux sites.
Le langage PYTHON est auto typé. Le typage se fait lors de l'affectation.
En pseudo-code, l'instruction d'affectation se note avec le signe ←

```
DÉBUT
toto <- 40        \\  On attribue la valeur 40 a la variable toto
FIN
```

.\scripts\part1\variables.larp

1.3. Les instructions

1.3.1. TEST SI …ALORS …SINON

```
DÉBUT
SI {condition} ALORS
   {suite_instructions}
SINON
   {suite_instructions}
FINSI
FIN
```

Teste une expression booléenne (condition) et exécute un bloc d'instruction(s) si la condition est vraie
La partie SINON n'est pas obligatoire. On peut utiliser l'instruction SI ... ALORS
Pour indiquer les blocs d'instructions :

- L'indentation qui se caractérise par le bloc décalé. ,→ Ce principe sera utilisé en PYTHON.

.\scripts\part1\si alors sinon.larp

1.3.2. La boucle POUR

```
DÉBUT
POUR {variable} = {valeur_initiale} JUSQU'À {valeur_finale}
   {suite_instructions}
FINPOUR
FIN
```

L'instruction POUR est utilisée lorsque le nombre d'itérations est connu à l'avance : elle initialise un compteur, l'incrémente après chaque exécution du bloc d'instructions, et vérifie que le compteur ne dépasse pas la borne supérieure.

.\scripts\part1\pour.larp

On peut préciser le pas d'incrémentation du compteur. Par défaut le pas est de 1. Le mot clé INCREMENT pour le pas.

```
POUR {variable} = {valeur_initiale} JUSQU'À {valeur_finale} INCRÉMENT {valeur_incrément}
```

1.3.3. La boucle TANTQUE

```
DÉBUT
TANTQUE {condition} FAIRE
   {suite_instructions}
FINTANTQUE
FIN
```

L'instruction TANTQUE est utilisée lorsque le nombre d'itérations n'est connu à l'avance : elle exécute le bloc d'instructions tant que la condition est vraie.

.\scripts\part1\tantque.larp

1.4. L'importance des commentaires

Il faut prendre l'habitude de commenter ses algorithmes et ses programmes.
Quelques remarques :

- Cela permet une relecture facile du code
- Cela permet une lecture plus facile pour une personne tierce (un correcteur par exemple)
- Commenter n'est pas paraphraser. Le commentaire doit apporter une information Voici un algorithme avec

des commentaires :

```
\\ Cet algorithme cherche la valeur de n pour que p passe de 50 a 100 avec une augmentation de 10%
n <- 0              \\ Initialisation de n
p <- 50             \\ Initialisation de p
TANTQUE p<100 FAIRE
   p <- p * 1.1     \\ p suivie de 10% d'augmentation
   n <- n + 1       \\ Incrementation de n
FINTANTQUE
ECRIRE n
FIN
```

.\scripts\part1\commentaires.larp

1.5. Table d'exécution d'un algorithme

Il existe différentes manières de réaliser une trace de programme et/ou d'algorithmes. Une trace :

- Permet de suivre pas à pas l'algorithme ;
- Permet de détecter des erreurs ;
- Permet de contrôler que l'algorithme fait bien ce que l'on avant prévu ;
- Permet de déterminer ce que fait un algorithme.

Dans la mesure du possible, on peut organiser une trace d'exécution d'un algorithme en constituant un tableau avec toutes les variables de l'algorithme. Il faut numéroter toutes les lignes de votre algorithme. En colonne, il faut indiquer le nom des variables et en ligne les numéros de ligne.

```
1 DÉBUT
2 a <- 0
3 n <- 5
4 TANTQUE a * a <= n FAIRE
5   a <- a + 1
6 FINTANTQUE
7 a <- a - 1
8 FIN
```

.\scripts\part1\tableExecution.larp

1.6. Exercices

✍ Exercice 2 : compléter le tableau

# ligne	n	a	Test TANTQUE	Commentaires
1	5	0	xxx	Initialisation des variables

Remarque : Avec le logiciel LARP nous pouvons vérifier les résultats en faisant une exécution pas à pas

En mathématiques, vous auriez une version minimaliste de ce tableau, qui correspondrait à l'état des variables :

n	5	5	5	5	5	
a	0					
Test TANTQUE						

✍ Exercice 3 : Voici un algorithme de recherche de seuil. Compléter la trace de cet algorithme.

```
DÉBUT
u <- 1
n <- 0
TANTQUE u <= 1000 FAIRE
   u <- 2 * u
   n <- n + 1
FINTANTQUE
FIN
```

.\scripts\part1\exo3.larp

n	0					
U	1					
Test TANTQUE						

✍ Exercice 4 : Écrire un algorithme en pseudo code qui calcule la moyenne de trois nombres a, b et c. Le résultat sera stocké dans une variable m.

.\scripts\part1\exo4.larp

✍ Exercice 5 : Écrire un algorithme qui renvoie le max de deux nombres a et b. Le résultat sera stocké dans une variable max.

.\scripts\part1\exo5.larp

✍ Exercice 6 : Écrire un algorithme qui stocke dans une variable max le maximum de trois variables a, b et c données.

.\scripts\part1\exo6.larp

✍ Exercice 7 : Écrire un algorithme qui calcule la factorielle d'un nombre (factorielle se note avec un!). Par exemple 4! = 4×3×2×1 = 24

.\scripts\part1\exo7.larp

✍ Exercice 8 : Écrire un algorithme qui permet d'échanger le contenu de deux variables a et b.

.\scripts\part1\exo8.larp

✍ Exercice 9 : Écrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer : 1+2+3+4+5 = 15

.\scripts\part1\exo9.larp

✍ Exercice 10 : Après avoir réalisé la trace de cet algorithme avec a=17 et b=3, précisez que représente a et i après exécution de algorithme

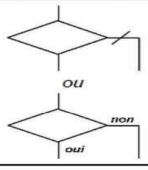
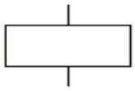
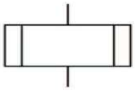
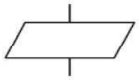
```
DÉBUT
LIRE a
LIRE b
i <- 0
u <- b
TANTQUE a <= b FAIRE
   a <- a - u
   i <- i + 1
FINTANTQUE
FIN
```

.\scripts\part1\exo10.larp

Il existe un mode de représentation des algorithmes sous forme d'organigrammes : Les algorigrammes

Vous pouvez vous rendre à l'adresse suivante :
https://troumad.developpez.com/C/algorigrammes/

Il existe des logiciels permettant de réaliser des organigrammes (LARP, lucidchart, word, paint, etc)

SYMBOLE	DÉSIGNATION	SYMBOLE	DÉSIGNATION
	**début** ou **fin** d'un algorithme		**Test ou Branchement conditionnel** décision d'un choix parmi d'autres en fonction des conditions
	symbole général de **« traitement »** opération sur des données, instructions, ... ou opération pour laquelle il n'existe aucun symbole normalisé		**sous-programme** appel d'un sous-programme
	**entrée / sortie**		**Liaison** Les différents symboles sont reliés entre eux par des lignes de liaison. Le cheminement va de haut en bas et de gauche à droite. Un cheminement différent est indiqué à l'aide d'une flèche

✍ Exercice 11 : Vous pouvez réaliser un algorigramme d'un algorithme que vous avez traité dans le TD

Nb : Pour vérifier votre travail, avec LARP vous pouvez passer de l'algorigramme au pseudo code à directement (l'inverse n'est pas vrai).

.\scripts\part1\exo11.larp