Algorithmique - Programmation (en Python)

1 Algorithme

Définition 1
Un algorithme est une suite finie d'opérations/d'instructions élémentaires, à appliquer dans un ordre déterminé, à des données. Sa réalisation permet de résoudre un problème donné.

Remarque 1

1. L'intérêt d'un algorithme est de pouvoir être traduit dans un langage de programmation (comme `Python`) pour être mis en oeuvre sur un ordinateur.
2. Les trois phases d'un algorithme sont, dans l'ordre :
   — l'entrée des données
   — le traitement des données
   — la sortie des résultats

2 État de la mémoire

Il faut imaginer que la mémoire d'un ordinateur est formée d'une multitude de petites boîtes qui ont chacune un nom et qui contiennent chacune un nombre.

Exemple 1

v	w	x	y	z
10	14	8	27	9

Au cours de l'exécution du programme, le contenu de ces boîtes change mais leur nom reste identique.

Une instruction s'exécute toujours dans un état de mémoire m_0 et cette exécution produit un nouvel état de mémoire m_1

Quand on exécute la séquence : $x \longleftarrow 2 * w - y$, dans l'état de la mémoire m_0 , on obtient l'état de la mémoire m_1 :

v	w	x	y	z
10	14			

Quand on exécute la séquence : $v \longleftarrow w - x$; $y \longleftarrow v + z$, dans l'état de la mémoire m_1 , on obtient l'état de la mémoire m_2 :

v	w	x	y	z

L'état de la mémoire m_2 aurait été différent si on avait exécuté les instructions dans un autre ordre.

3 Variable et affectation

Lors de l'exécution d'un algorithme, on va avoir besoin de stocker des données, voire des résultats. Pour cela, on utilise des variables qui correspondent à des emplacemnts en mémoire.

On attribue un nom (si possible significatif par rapport à ce qu'elle représente dans le problème) à chaque variable.

Les variables utilisées peuvent être de plusieurs **types**, en particulier :

- les entiers : `int`
- les flottants (nombres à virgule, permet une approximation des réels) : `float`
- les chaînes de caractères (type par défaut) `str`
- les listes
- ....

Habituellement on précise les type de chaque variable qu'on nomme, cette étape est la **déclaration des variables**.

Définition 2
Les instructions de base sur des variables sont les suivantes :

- *la **saisie** : on demande à l'utilisateur de l'algorithme de donner une valeur à la variable ;*
- *l'**affectation** : le concepteur de l'algorithme donne une valeur à la variable ; cette valeur peut être le résultat d'un calcul ;*
- *l'**affichage** : on affiche la valeur de la variable.*

Remarque 2 Dans les algorithmes, on adoptera les conventions suivantes :

saisir une variable x : Saisir x

afficher la valeur d'une variable x : Afficher x

affecter la valeur 2019 à la variable x : $x \longleftarrow 2019$

Exemple 2

L'algorithme ci-dessous est un exemple d'algorithme calculant l'image d'un réel x par la fonction définie par $f(x) = 3x - 2$.

Algorithme	Programme Python

1 Saisir x
2 $y \longleftarrow 3x - 2$
3 Afficher y
	`x=float(input("x=?"))`
	`y=3*x-2`
	`print(y)`

Remarque 3

Dans l'algorithme ci-dessus, l'utilisateur saisit la variable x, alors que la variable y est affectée au cours du traitement.

4 Les tests

Définition 3
*La résolution des certains problèmes nécessite la mise en place d'un **test** (ou **condition**) pour savoir si l'on doit effectuer une tâche. Si la condition est remplie alors on effectue la tâche, sinon on effectue (éventuellement) une autre tâche.*

Dans un algorithme, on code la structure du « *Si ... Alors ... Sinon ...* » sous la forme suivante :

```
si Condition alors
    Tâche 1-1
    Tâche 1-2
    ...
sinon
    Tâche 2-1
    Tâche 2-2
    ...
fin si
```

Définition 4
Une condition est une expression qui peut prendre l'une des deux valeurs suivantes vrai ou faux.
*On dit également que c'est une valeur de type **"logique"** ou **"booléen"**.*

Remarque 4

1. Il est important de respecter les espaces laissés au début de chaque ligne, car ils permettent une meilleure lisibilité de l'algorithme.
2. Le « Sinon » n'est pas obligatoire ; s'il n'est pas présent, aucune tâche ne sera effectuée si la condition n'est pas remplie.

Exemple 3

1. Exemple permettant de déterminer si le triangle ABC est rectangle en A

Algorithme	Programme Python

Saisir AB
Saisir AC
Saisir BC
si $BC^2 = AB^2 + AC^2$ **alors**
  Afficher "ABC est rectangle en A"
sinon
  Afficher "ABC n'est pas rectangle en A"
fin si
	`ab=float(input("AB=?"))`
	`ac=float(input("AC=?"))`
	`bc=float(input("BC=?"))`
	`if bc**2==ab**2+ac**2:`
	`  print("ABC est rectangle en A")`
	`else:`
	`  print("ABC n'est pas rectangle en A")`

2. L'algorithme ci-dessous est un exemple d'algorithme calculant l'image d'un réel x par la fonction définie par $f(x) = \dfrac{1}{(x-1)(x+2)}$ en respectant son ensemble de définition.

Algorithme	Programme Python

1 Saisir x
2 **si** $x = 1$ **alors**
3 ⎸ Afficher "1 n'a pas d'image par f"
4 **sinon**
5 ⎸ **si** $x = -2$ **alors**
6 ⎸ ⎸ Afficher "-2 n'a pas d'image par f"
7 ⎸ **sinon**
8 ⎸ ⎸ $y \longleftarrow 1/((x-1)*(x+2))$
9 ⎸ ⎸ Afficher y
10 ⎸ **fin si**
11 **fin si**
	`x=float(input("x="))`
	`if x==1:`
	`  print("1 n'a pas d'image par f")`
	`elif x==-2:`
	`  print("-2 n'a pas d'image par f")`
	`else:`
	`  y=1/((x-1)*(x+2))`
	`  print('f(',x,')=',y)`

3. L'algorithme ci-dessous simule un jeu de pile ou face avec une pièce non truquée. « Pile » est représenté par le nombre 0 et « Face » par le nombre 1.

Algorithme	Programme Python

1 Saisir choix (pile : 0 ou face : 1)
2 Tirage est aléatoirement 0 ou 1
3 **si** *choix* = *tirage* **alors**
4 ⎸ Afficher "Gagné"
5 **sinon**
6 ⎸ Afficher "Perdu"
7 **fin si**
	`import random`
	`choix=int(input("Pile:0 ou Face:1 ?"))`
	`tirage=random.randint(0,1)`
	`if choix==tirage:`
	`  print("Gagné")`
	`else:`
	`  print("Perdu")`

Remarque 5

On peut écrire des conditions plus complexes en reliant des comparaisons à l'aide des opérateurs logiques **ET**, **OU** et **NON**.

Exemple 4

- Condition 1 : "avoir un sac à dos Eastpack"
- Condition 2 : "avoir un sac à dos noir"

— "Condition 1 **ET** condition 2" sera vraie si les deux conditions sont toutes les deux vraies.
Avec les conditions précédentes : "avoir un sac à dos Eastpack noir"

— "Condition 1 **OU** condition 2" sera vraie si l'une au moins des deux conditions est vraie.
Avec les conditions précédentes :
  — "avoir un sac à dos Eastpack noir" (conditions 1 et 2 vraies)
  — "avoir un sac à dos Eastpack vert" (condition 1 vraie)
  — "avoir un sac à dos sans marque noir" (condition 2 vraie)

— "**NON** (condition 1)" sera vraie si le sac est d'une couleur autre que noire.

5 La boucle "for" ("pour")

Définition 5
Lorsque l'on doit répéter un nombre de fois connu à l'avance la même tâche, on utilise une boucle itérative de la forme "Pour ... allant de ... à ... faire ...".

Dans un algorithme, cette structure est codée de la façon suivante :

```
pour variable allant de valeur de départ à valeur de fin faire
    Tâche 1
    Tâche 2
    ...
fin pour
```

La variable utilisée dans la boucle est appelée **compteur**. À chaque passage dans la boucle, sa valeur est automatiquement augmentée de 1.

Exemple 5

1. L'algorithme ci-dessous affiche le carré des entiers allant de 1 à un entier n donné.

Algorithme	Programme Python

Saisir N
pour I *allant de* 1 *à* N **faire**
  Afficher I^2
fin pour
	`n=int(input("n="))`
	`for i in range(1,n+1):`
	`  print(i**2)`

Remarque 6

ATTENTION : l'instruction i in **range**(a, b) en Python fait varier i de a à $b-1$.

2. L'algorithme ci-dessous affiche la somme des entiers consécutifs allant de 1 à un entier n donné.

Algorithme	Programme Python

Saisir N
$S \longleftarrow 0$
pour I *allant de* 1 *à* N **faire**
  $S \longleftarrow S + I$
fin pour
Afficher S
	`n=int(input("n="))`
	`s=0`
	`for i in range(1,n+1):`
	`  s=s+i`
	`print(s)`

6 La boucle "while" ("tant que")

Définition 6
Lorsque l'on doit répéter un nombre de fois inconnu à l'avance la même tâche sous une certaine condition, on utilise une boucle tant que.

Dans un algorithme, cette structure est codée de la façon suivante :

```
tant que condition faire
    Tâche 1
    Tâche 2
    ...
fin tq
```

Exemple 6

Un individu a emprunté à un ami une somme de 2500 euros (prêt sans intérêts). Pour rembourser son ami, il prévoit de lui remettre 110 euros par mois. Mais comme cela ne correspond pas à un nombre pile de mois, il se demande quel sera le montant à rembourser le dernier mois.

L'algorithme ci-dessous permet de donner une réponse

Algorithme	Programme Python
1 montant $\longleftarrow$ 2500 2 **tant que** *montant* $\geqslant$ 110 **faire** 3 $\quad$ montant $\longleftarrow$ montant -110 4 **fin tq** 5 afficher montant	1 montant=2500 2 while montant>=110: 3 $\quad$ montant=montant-110 4 print(montant)

7 Les fonctions

On peut simplifier l'écriture des programmes en utilisant des fonctions, en suivant le modèle des fonctions numériques déjà vues en mathématiques.

Définition 7
Une fonction est un bloc d'instructions qui a reçu un nom, dont le fonctionnement dépend d'un certain nombre de ***paramètres*** *(appelés* ***arguments*** *de la fonction) et qui renvoie un unique résultat (le plus souvent à l'aide de l'instruction Retourner).*

Dans un algorithme, une fonction est codée de la façon suivante :

1 **Fonction** nom_ fonction(*arguments*)
2 **début**
3 $\quad$ Tâches
4 $\quad$ **Retourner** *résultat*
5 **fin**

Remarque 7

1. Une fonction peut n'avoir aucun argument.
2. Une fonction peut être appelée dans un programme ou dans une autre fonction.

Exemple 7

1. On peut définir la fonction *mul_trois* qui à partir des trois paramètres a,b et c renvoie comme résultat le produit de ces trois nombres

Algorithme	Programme Python
1 **Fonction** mul_trois(a,b,c) 2 **début** 3 $\quad$ **Retourner** a*b*c 4 **fin**	1 def mul_trois(a,b,c): 2 $\quad$ r=a*b*c 3 $\quad$ return r

Si on veut par exemple calculer le produit des trois nombres 2 , 5 et 6, on appelle la fonction :

Algorithme	Programme Python
1 mul_trois(2,5,6)	1 `print(mul_trois(2,5,6))`

2. La fonction ci-dessous prend en argument 2 nombres et retourne la valeur du plus grand.

Algorithme	Programme Python
1 **Fonction** max(a,b) 2 **début** 3 | **si** $a \geqslant b$ **alors** 4 | | Retourner a 5 | **sinon** 6 | | Retourner b 7 | **fin si** 8 **fin**	1 `def max(a,b):` 2 `  if a>=b:` 3 `  return a` 4 `  else:` 5 `  return b`

8 Les listes

Définition 8
On appelle liste une variable, formée en quelque sorte de cases, chaque case étant numérotée et pouvant contenir une donnée.

Si L est le nom d'une liste, l'expression $L[n]$ désigne le contenu de la n ième case de la liste L.

Exemple 8

Pour créer une liste L, par exemple de 4 notes qui sont 12, 15, 14 et 18, on saisit : $L = [12, 15, 14, 18]$.

Pour calculer la moyenne de ces notes on peut définir la fonction ci-dessous :

Algorithme	Programme Python
1 **Fonction** moyenne(L) 2 **début** 3 | $S \longleftarrow 0$ 4 | **pour** *note dans* L **faire** 5 | | $S \longleftarrow S$+note 6 | **fin pour** 7 | retourner S/longueur de L 8 **fin**	1 `def moyenne(L):` 2 `  S=0` 3 `  for note in L:` 4 `  S=S+note` 5 `  moyenne=S/len(L)` 6 `  return(moyenne)`

9 Boîte à outils Python

<table>
<tr><td>Saisie</td><td colspan="3">A=input(”A= ?”) si A est une chaîne de caractères (type str)
A=int(input(”A= ?”)) si A est un nombre entier (type int)
A=float(input(”A= ?”)) si A est un flottant (type float)</td></tr>
<tr><td>Affichage</td><td colspan="3">print(A) affiche le contenu de la variable A
print(”bonjour”) affiche le texte saisi entre guillemets
print(”la valeur de A est”,A) mélange texte et variable</td></tr>
<tr><td>Affectation</td><td colspan="3">A=2021 A prend la valeur 2021</td></tr>
<tr><td>Commentaire</td><td colspan="3">Les commentaires dans un programme s’écrivent après #</td></tr>
<tr><td rowspan="3">Utiliser un module</td><td colspan="3">from module import fonction
un module est un fichier contenant le plus souvent un ensemble de fonctions</td></tr>
<tr><td colspan="3">from math import * le module math contient les définitions de nombreuses fonctions mathématiques : pi, sin(), sqrt(),...</td></tr>
<tr><td colspan="3">from random import * le module random contient les définitions liées aux nombres aléatoires :
random() simule un nombre décimal dans $[0;1[$
randint(a,b) simule un nombre entier dans $[a;b]$
uniform(a,b) simule un nombre décimal dans $[a;b]$</td></tr>
<tr><td>Opérations</td><td>addition
soustraction
multiplication
puissance
division
reste de division entière
quotient de division entière</td><td>+
-
*
**
\
%
//</td><td>2 + 5 donne 7
8 - 2 donne 6
6 * 7 donne 42
2 ** 3 donne 8
7 \ 2 donne 3.5
7 % 3 donne 1
7 // 3 donne 2</td></tr>
</table>

<table>
<tr><td rowspan="2">Instruction conditionnelle ”Si” :
Si condition C_1
Alors instruction A_1
Sinon Si condition C_2
Alors instruction A_2
Sinon instruction A_3</td><td colspan="2">Tester : A==B ; A !=B (A $\neq$ B) ;A>=B ; A<=B ;A<B ;A>B</td></tr>
<tr><td colspan="2">ET ; OU ; NON : A AND B ; A OR B ; NOT A</td></tr>
<tr><td rowspan="2">Boucle ”Pour” :
Pour i allant de 1 à n
instruction A_1
Fin pour</td><td>if condition C_1 :
instruction A_1
elif condition C_2 :
instruction A_2
else :
instruction A_3</td><td>penser au ” :”
penser à l’indentation (décalage à droite)
penser au ” :”

il n’y a pas d’instruction de fin,
de même pour while, for et def</td></tr>
<tr><td>for i in range$(1, n+1)$:
instruction A_1</td><td>penser au ” :”</td></tr>
<tr><td></td><td>Boucle ”Tant que” :
Tant que condition C_1
instruction A_1
Fin tant que</td><td>while condition C_1 :
instruction A_1</td></tr>
<tr><td>for i in range(n)
for i in range(n_0, n)
for i in range(n_0, n, p)</td><td>i prend les valeurs de 0 à $n-1$
i prend les valeurs de n_0 à $n-1$
i prend les valeurs de n_0 à $n-1$
jusqu’au dernier inférieur ou égal à $n-1$
avec un pas de p</td></tr>
<tr><td>Fonction :</td><td colspan="2">def nom_fontion$(a, b, ...)$:
instruction A_1
$y = ..$
return y</td></tr>
</table>