Notions d'algorithme

Table des matières

1 Introduction

1.1 Algorithme

Définition 1 : Un algorithme est une suite d'instructions, qui une fois exécutées correctement, conduit à un résultat donné.
Pour fonctionner, un algorithme doit donc contenir uniquement des instructions **compréhensibles** par celui qui devra l'exécuter.

Exemple :

- Ci-contre, un algorithme en langage courant.

  Si on applique cet algorithme au nombre 3, on a :

  $$3 \xrightarrow{+1} 4 \xrightarrow{\times 2} 8 \xrightarrow{-3} 5$$

  Il peut être associé à une fonction affine :

  $$f(x) = 2(x+1) - 3 = 2x + 2 - 3 = 2x - 1$$

```
Choisir un nombre.
Ajouter 1.
Multiplier 2.
Soustraire 3.
Afficher le résultat.
```

- Si l'on cherche quel nombre a donné 0. Il faut remonter l'algorithme :

  $$0 \xrightarrow{+3} 3 \xrightarrow{\div 2} \frac{3}{2} \xrightarrow{-1} \frac{1}{2}$$

  On peut écrire l'algorithme réciproque ci-contre :
  La nouvelle fonction affine définie est alors :

  $$g(x) = \frac{x+3}{2} - 1 = \frac{1}{2}x + \frac{1}{2}$$

```
Choisir un nombre.
Ajouter 3.
Diviser par 2.
Soustraire 1.
Afficher le résultat.
```

Remarque : La maîtrise de l'algorithmique requiert deux qualités :

- il faut avoir une certaine **intuition** pour se mettre à la place de la machine.
- il faut être **méthodique et rigoureux** pour bien comprendre ce que va effectuer le programme et ne pas oublier tous les cas de figure qui peuvent se présenter.

1.2 Conventions pour écrire un algorithme

Historiquement, plusieurs types de notations ont représenté des algorithmes.

- Il y a eu notamment **une représentation graphique**, avec des carrés, des losanges, etc. qu'on appelait des organigrammes. Cependant dès que l'algorithme se complique un peu, ce n'est plus adapté.
- On utilise généralement une série de conventions appelée « **pseudo-code** », qui ressemble à un langage de programmation authentique sans syntaxe.
  Certains langages de programmation demande de déclarer les variables auparavant (comme en C++) mais dans la plupart des langages, comme Python, cela n'est pas nécessaire.

Exemple : Soit les deux algorithmes précédents en pseudo-code :

```
x réel
Lire x
y=x+1
y=2y
y=y−3
Afficher y
```

```
x réel
Lire x
y=x+3
y=y/2
y=y−1
Afficher y
```

1.3 Types d'instructions et langages de programmation

Les ordinateurs, ne sont capables de comprendre que quatre catégories d'instructions. Ces quatre familles d'instructions sont :

- la lecture (entrées) / l'affichage (sorties)
- l'affectation de variables
- les tests
- les boucles

Un algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage de programmation.

Pour écrire un algorithme sur un ordinateur, il est nécessaire d'avoir un **langage** de programmation. Chaque langage a une **syntaxe** : règles qui régissent les différentes manières dont les éléments du langage peuvent être combinés pour obtenir des programmes. La ponctuation (par exemple l'apposition d'un symbole « : » ou « ; » en fin de ligne d'instruction d'un programme) relève de la syntaxe.

Il existe des milliers de langages de programmation, ayant chacun ses spécificités.

Certains langage demande d'être compilé, comme le latex pour écrire ce livre, ou interprété, comme le langage Python.

- Un **compilateur** est un programme informatique qui transforme un code source écrit dans un langage de programmation (le langage source) en langage machine (le langage cible).
- Un **interpréteur** se distingue d'un compilateur par le fait que, pour exécuter un programme, les opérations d'analyse et de traductions sont réalisées à chaque exécution du programme (par un interpréteur) plutôt qu'une fois pour toutes (par un compilateur).

2 Les Instructions

2.1 Lecture et affichage d'une variable

Définition 2 : Entrées - Sorties

- **Lire**, « imput en anglais », une variable signifie que l'utilisateur doit rentrer une valeur pour que le programme la lise
- **Afficher**, « print en anglais », une variable signifie que le programme renvoie la valeur de la variable que le programme a stocké.

2.2 Variable

Définition 3 : 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.

Remarque : Le nom de la variable (l'étiquette de la boîte) obéit à des impératifs changeant selon les langages.
Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces et les accents. Un nom de variable correct commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilisé.

On peut distinguer 3 types principaux de variable :

- Type numérique Python : un entier ou un flottant (décimal)
- Type alphanumérique : du texte qui doit être mis entre guillemets "texte"
- Type booléen : false ou true

2.3 Affectation d'une variable en Python

Affecter une valeur à une variable, c'est lui attribuer une valeur.

$\boxed{a = 24}$ la variable a prend la valeur 24.

$\boxed{b = a}$ la variable b prend la valeur de a.

$\boxed{c = \text{"Paul"}}$ la variable c prend la valeur "Paul"

Remarque : On peut trouver d'autres notations pour l'affectation :

$\boxed{a := 24}$, $\boxed{a \leftarrow 24}$ ou $\boxed{24 \rightarrow a}$ la variable a prend la valeur 24

On peut aussi affecter une variable à l'aide d'une opération :

$\boxed{c = a + 4}$ la variable c prend la valeur $a + 4$.

On peut changer la valeur d'une variable avec elle-même :

$\boxed{b = b + 1 \text{ ou } b+ = 1}$ incrémente de 1 la variable b.

Opérateur alpha-numérique : concaniser (symbole &)

$\boxed{a = \text{"Paul"} ; b = \text{"Milan"} ; c = a\&b}$ la variable c prend la valeur "PaulMilan".

Les opérateurs numériques en Python sont :
$+, -, *$ (multiplication), / (division), $**$ (puissance)

Les opérateurs logiques sont : and, or et not

3 Les tests en Python

L'instruction des tests se fait avec if suivi d'un condition et se terminant par « : » avec une **indentation** de 4 espaces sur un ordinateur ou 2 points sur la Ti 83.

Trois formes sont possibles

```
if condition:
    instruction 1
suite ...
```

Forme simple

Si la condition n'est pas vérifiée, le programme ignore le test et passe à la suite

```
if condition:
    instruction 1
else:
    instruction 2
```

Forme complète

Si la condition n'est pas vérifiée, le programme exécute l'instruction 2

```
if condition 1:
    instruction 1
elif condition 2:
    instruction 2
else:
    instruction 3
```

Forme multicondtions

Si la condition 1 n'est pas vérifiée et si la condition 2 est vérifiée, le programme exécute l'instruction 2.

Si les conditions 1 et 2 ne sont pas vérifiées, le programme exécute l'instruction 3.

La condition portant sur une variable peut être :

- Une comparaison avec une autre valeur (égalité, inégalité, non égalité)
- Une comparaison avec une autre variable (égalité, inégalité, non égalité)

On peut la décomposer en 2 conditions reliées par un opérateur logique :

- condition 1 **and** condition 2 : les deux conditions doivent être vérifiées.
- condition 1 **or** condition 2 : l'une au moins des conditions doit être vérifiée.

On peut aussi imbriquer un ou plusieurs autres tests à l'intérieur d'un test.

Exemple :

On peut programmer la fonction valeur absolue, « abs », par le programme suivant :

abs(−2) renvoie 2, abs(3) renvoie 3, abs(0) renvoie 0

```
def abs(x):
    if x>=0:
        y=x
    else:
        y=-x
    return y
```

4 Les boucles

Définition 4 : Une **boucle** est une structure répétitive qui effectue un certain nombre de fois un calcul à l'aide d'un **compteur** ou sous le contrôle d'une **condition**.

4.1 Boucle itérative

Un boucle itérative est une boucle gérée par un compteur.

En **Python** on utilise le compteur i prenant les valeur dans un ensemble d'entiers naturels défini par « **range**([début], fin,[pas] ».

- début : optionnel, par défaut le début est 0.
- fin : la valeur finale sera l'entier **immédiatement inférieur** à fin.
- pas : optionnel, par défaut le pas est de 1 (le pas peut être négatif)

$\text{range}(5) = \{0,1,2,3,4\}$, $\text{range}(3,8) = \{3,4,5,6,7\}$, $\text{range}(2,12,3) = \{2,5,8,11\}$

Exemple : Programmation de factorielle n (n !)

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

On obtient :

n	1	2	3	4	5
s	1	2	6	24	120

```
def fact(n):
    s=1
    for i in range(1,n+1):
        s=s*i
    return s
```

4.2 Boucle conditionnelle

Une boucle conditionnelle est gérée par une condition. Tant que (**while** en Python) cette condition est vérifiée la boucle continue.

Exemple : Programmation de la partie entière $\text{E}(x)$.

$$n \leqslant x < n+1 \ \Rightarrow \ \text{E}(x) = n$$

- si $x \geqslant 0$, la condition pour rentrer dans la boucle est l'**inégalité contraire** à $x < n+1$ soit $x \geqslant n+1$.
- si $x < 0$, la condition pour rentrer dans la boucle est l'**inégalité contraire** à $x \geqslant n$ soit $x < n$.

$\text{E}(4.3)$ renvoie 4 et $\text{E}(-2.3)$ renvoie -3.

```
def E(x):
    n=0
    if x>=0:
        while x>=n+1:
            n=n+1
    else:
        while x<n:
            n=n-1
    return n
```

⚠ Dans le cas où la condition est toujours vérifiée, le programme ne sort pas de la boucle c'est ce qu'on appelle une « **boucle infinie** ».

Si lors de l'exécution d'un programme, la calculatrice « mouline » sans s'arrêter, c'est que peut-être vous avez programmé une boucle infinie.

5 Résumé instructions Python

5.1 Principes de base

- Type de variables : **int** : entier, **float** : décimal, **bool** : variable booléene

int	783	0	-192
float	9.23	0.0	$-1.7.\dot{-}6$
bool	True	False	

- Pour afficher un commentaire, on précède le texte du symbole : #
- Opérateurs usuels : $+, -, *, /$ Division euclidienne : % reste et // le quotient

  $**$ exponentiation (mise à la puissance) $2**10 = 2^{10} = 1\,024$.
- Affichage : **print**("x=",a) : affiche $x =$ valeur de a.

5.2 Tests

- **Test** : bloc d'instructions exécutés uniquement si la condition est vraie.

```
if condition :
    instructions
elif condition :  #si nécessaire
else:             #si nécessaire
    instructions
```

- Conditions :

  if $a == b$: si a est égal à b , **if** $a! = b$: si a est différent de b

  if $2 < a <= 5$: si a est dans $]2\,;\,5]$

5.3 Boucles

- **Boucle itérative**

```
for variable in range(a,b) :
    bloc instructions
```

- **Boucle conditionnelle**

```
while condition :
    bloc instructions
```

5.4 Fonction

Une fonction est la traduction d'un algorithme qui :

1) prend en entrée un ou plusieurs arguments.
   S'il n'y a pas d'argument laisser les parenthèses vides.
2) effectue une suite d'instructions
3) retourne un résultat

```
def nomfct(x,y,...) :
    bloc instructions
    return résultat
```

5.5 Les modules

- Un module est un fichier python qui regroupe des fonctions et des définitions de constantes : comme math ou random.
- On appelle un module mod par : **from** mod **import** *.
- Fonctions mathématiques du module math
  1) **floor**(-7.6) : partie entière de -7.6, donne ici -8.0.
  2) **sqrt**(2) : racine carrée de 2.
  3) **factorial**(4) : factorielle 4, donc 24 *(uniquement pour les entiers positifs)*.
- Fonctions trigonométriques du module math
  1) fonctions trigonométriques : **sin**(x), **cos**(x), **tan**(x) *(x en radian)*
  2) fonctions trigonométriques inverses : **asin**(x), **acos**(x), **atan**(x)
  3) **degrees**(x) : convertit x en degrés
  4) **radians**(x) : convertit x en radian
  5) **pi** : constantes π
- Fonctions du module random
  1) **random**() : valeur flottante dans $[0\,;\,1]$
  2) **randint**(a,b) : valeur entière entre a inclus et b inclus