Notions de base en algorithmie

1 Introduction à l'algorithmique

1.1 Notion d'algorithme

Définition

Un algorithme est une procédure se terminant en nombre *fini* d'étapes qui permet de résoudre une classe de problèmes, écrite de façon suffisamment *détaillée* pour être suivie par un humain ne possédant pas de compétence particulière et qui n'est même pas obligé de comprendre le problème qu'il est en train de résoudre.

Ainsi un algorithme fonctionne avec des *données* précisant le cas particulier du problème qu'il traite. En retour, il construit un *résultat* répondant à ce cas particulier du problème.

1.2 Notion de programme

Les algorithmes existent depuis l'antiquité dans le but de résoudre certaines tâches très calculatoires. Si l'exécution d'un algorithme à la main est en théorie toujours possible, certains algorithmes nécessitent un nombre de calculs très important, ce qui rend cette exécution d'une part longue et augmente le risque d'erreurs humaines d'autre part.

Avec l'avènement de l'informatique, il est devenu possible d'automatiser les algorithmes de sorte à ce qu'il soient exécutés de manière plus rapide et plus sûre. Le prix de cette automatisation est la traduction de l'algorithme sous une forme lisible par la machine.

Définition

Un *programme* est la traduction d'un algorithme dans un langage particulier, qui est à la fois compréhensible pour l'homme et interprétable pour la machine. Il est exprimé dans un langage de programmation constitué d'un assemblage d'instructions regroupées dans un fichier texte appelé *code source* du programme. Ce code source est ensuite traduit en langage machine par un compilateur.

L'exécution d'un programme commence à la première instruction et continue en exécutant d'autres instructions en suivant des règles précises. Le parcours des instructions au cours de l'exécution d'un programme s'appelle le *flot d'exécution*.

(nous utiliserons Python 3 dans notre cours)

Algorithme

Algorithme 1 : Test de parité

Données : $a \in \mathbb{N}^+$
Résultat : afficher si a est pair ou impair

```
1 a←valeur saisie par l'utilisateur
2 si reste(a/2)=0 alors
3 |  afficher a est pair
4 sinon
5 |  afficher a est impair
```

python

Test_parite.py

```
"""
Le programme prendra une valeur saisie par l'utilisateur
et affichera s'il est pair ou impair.
"""
a=int(input(" Saisir un nombre entier strictement positif "))
if a%2==0 :
    print("a est pair")
else :
    print("a est impair")
```

2 Variables

Lors de son exécution, un algorithme manipule des données dont les variables. Une variable est définie par :

Définition

—
—
—

Définition

Identificateur
L'identificateur correspond au nom de la variable. Il doit être explicite. Pour nommer une variable, il est possible d'utiliser :
— les lettres de l'alphabet en minuscules ($\mathbf{a} \rightarrow \mathbf{z}$) ou en majuscules ($\mathbf{A} \rightarrow \mathbf{Z}$) ;
— des chiffres ($\mathbf{0} \rightarrow \mathbf{10}$) ;
— l'underscore _.
Le nom d'une variable commence par une lettre.

Définition

Typage
Le typage correspond à la nature de la variable (booléen, nombre entier, nombre réel *etc*). A chaque type est associé une convention de codage informatique.

On parle de typage statique lorsqu'il est nécessaire de définir le type d'une variable lors de sa création.
On parle de typage dynamique lorsque, par exemple, le type le mieux adapté est choisi automatiquement par le système informatique. Python appartient à cette catégorie.

2.1 Types simples

Ce sont les types les plus fondamentaux. Ils ne sont pas identiques dans tous les langages, mais présentent une base commune. En python, on retrouve notamment :

Les booléens :	type bool, deux états possible donc 1 bit
Les entiers relatifs :	type int sur au moins 32 bits
Les nombres décimaux :	type float (selon la norme IEEE754)
Les nombres complexes :	type complex codés sous la forme $a + bj$ où a et b sont deux nombres décimaux et j vérife $j^2 = -1$.

2.2 Application 1

: Choisissez le type qui vous semble adapté pour représenter chacune des valeurs suivantes :
— la pointure d'un individu ;
— le nom d'un individu ;
— la valeur de pi ;
— le nombre d'humain.

2.3 Types itérables

Il existe des types composés de valeurs de type simple. Ces types sont appelés itérables. Y figurent, par exemple, :

- les listes (type list), définies entre crochet : $[12.5, 15.7]$;
- les n-uplets ou *tuple* en anglais, qui sont des listes non modifiables : $(12.5, 15.7)$;
- les chaines de caractères (str), définies entre guillemets ou apostrophes : $"PCSI"$ ou $'PCSI'$;
- les dictionnaires ;
- etc.

3 Expressions et instructions

3.1 Expressions

Définition

Expression

Une expression est une suite de caractère conduisant à l'évaluation d'un calcul. Un résultat est retourné.

Une expression peut utiliser des constantes ou des variables liées par des opérateurs ou des fonctions. A chaque type est associé des opérateurs et des fonctions spécifiques. Utiliser un opérateur inadéquat au type du reste de l'expression amène à une erreur.

```
>>> 1+1
2
>>> 'a'+'a'
'aa'
>>> 1 == 1
True
```

```
>>> abs(-3)
3
>>> abs('PCSI')
TypeError: bad operand type for abs(): 'str'
```

Python dispose de 68 fonctions natives, toujours disponibles. Ces fonctions sont utilisables avec certains types uniquement. A ces fonctions, s'ajoute les opérations entre variables décrites ci-dessous.

3.2 Opérateurs sur les entiers et les décimaux

Les opérateurs sur les entiers et les décimaux sont identiques et résumés dans le tableau suivant :

Opérateur	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division flottante
//	Division entière
%	Reste de la division entière
-x	Opposé en préfixe
abs(x)	Valeur absolue
**	Exponentiation

La question de l'ordre dans lequel sont évaluées les expressions qui comprennent plusieurs opérateurs sans parenthèses (par ex. 2+3*4) fait l'objet de règles généralisant celles sur les priorités opératoires. On parle de précédence. Concrètement, en l'absence de parenthèses qui sont toujours évaluées en premier on évalue dans l'ordre :

1. les exponentiations
2. les multiplications, divisions entières et modulos
3. les additions et les soustractions

3.3 Précédence

L'évaluation des fonctions est traitée comme un traitement de parenthèses. C'est à dire que sont d'abord évaluées les expressions qui sont des paramètres des fonctions appellées, puis les résutats des fonctions elles-mêmes et enfin les règles de précédence sont appliquées.

Malheureusement ces règles ne suffisent pas toujours à lever les ambiguïtés. Comment comprendre les expressions $2-3-1$ et 2**3**2 ? Dans la plupart des cas les expressions sont évaluées de gauche à droite. Il y a néanmoins une exception pour l'exponentiation, qui est évaluée de droite à gauche. Ainsi $2-3-1$ est évalué comme -2 là où 2**3**2 est évalué comme 2**9 soit 512.

3.4 Instructions

Un programme utilise un certain nombre de variable. Les valeurs de ces variables constituent l'**état du programme** (ou l'état de la mémoire).

Définition

Une instruction amène une modification de l'état d'un programme. Une instruction peut inclure des expressions.

```python
a = 1
a = 101**0.5
del a
```

3.5 Instructions simples : déclaration et affectation

Un programme utilise des variables. Lors de son exécution, il doit définir l'identifiant, le type et la valeur de chacune des variables afin de pouvoir les gérer dans l'espace mémoire. Les deux premières opérations s'appellent déclaration et la troisième affectation.

En règle générale, la déclaration s'effectue en début d'algorithme. L'affectation peut avoir lieu n'importe quand.

Algorithme

Données : $a \in \mathbb{N}$; $b \in \mathbb{N}$
Résultat : Effectuer un calcul
1 a←22
2 $b \leftarrow a + a^2$

3.5.1 *Application 2*

Un programme est réalisé par l'unique ligne suivante :

```
x=x+1
```

1. Traduire ce programme en pseudo-code.
2. x apparait deux fois. Expliquer la signification de chacune.
3. Pourquoi ce programme ne fonctionne pas ? Que faut-il ajouter ?

3.5.2 Quelques raccourcis

Spécificité

```
a+=1          a=a+1
a+=b          a=a+b
a-=b          a=a-b
a*=b          a=a*b
a/=b          a=a/b
a**=b         a=a**b
a%=b          a=a%b
```

3.5.3 Déclaration automatique

Spécificité

En python, les variables n'ont jamais un type définitif puisque le langage peut le modifier en cours d'exécution. Aussi, il n'est pas possible de déclarer les variables.

3.5.4 *Application 3*

1. Proposer un algorithme qui inverse les valeurs de deux variables x et y.

3.5.5 Affectation simultannée

L'affectation simultannée ou multiple permet l'affectation simultanément plusieurs variables.

Spécificité

```
>>> a,b=1,2
>>> a
    1
>>> b
    2
```

3.5.6 *Application 4*

Décrire l'évolution de l'état du programme suivant pour chaque ligne de code, en précisant s'il s'agit d'une expression ou d'une instruction. L'état initial du programme est le suivant : b est défini comme un entier et sa valeur est 5.

```
1 a=2
2 a=b-7
3 b+1
4 b+=1
5 c=a-b
6 a=1./a + 1 -1./a
```

3.6 Cas particulier des entrées-sorties

On appelle entrées-sorties les opérations qui permetttent de communiquer avec un utilisateur.

— La commmande `print()` qui permet d'afficher à l'écran un texte ou la valeur d'une variable. Après un `print()`, le programme continue son exécution ;
— La commande `input()` interrompt un programme jusqu'à ce que l'utilisateur tape au clavier une séquence de caractères.

4 Structuration d'un programme

4.1 Flux d'instructions

Un programme se compose donc de lignes d'instructions qui s'exécutent les unes après les autres.

python
```python
# Fichier: calcul_sphere.py
# Outils importés
from math import sqrt,pi
# Dimensions en x/y/z de l'ellipsoïde.
dx = float(input("dx = "))
dy = float(input("dy = "))
dz = float(input("dz = "))
# Calcul du volume sphérique englobant:
r = sqrt(dx**2+dy**2+dz**2)
s = 4/3*pi*r**3
# Calcul du volume de l'ellipsoïde.
e = 4/3*pi*dx*dy*dz
# Rapport entre les deux:
es = e / s
# Affichage du résultat
print("ellipsoïde:",e,"sphère:",s,"rapport:",es)
```

4.2 Instructions composées

On appelle instructions composées les instructions qui modifie l'ordre d'exécution d'un programme (sous certaines conditions). Il s'agit de regrouper les instructions à exécuter sous forme de blocs.

```
ligne d'en-tête :
    instruction 1
    instruction 2
    instruction 3
```

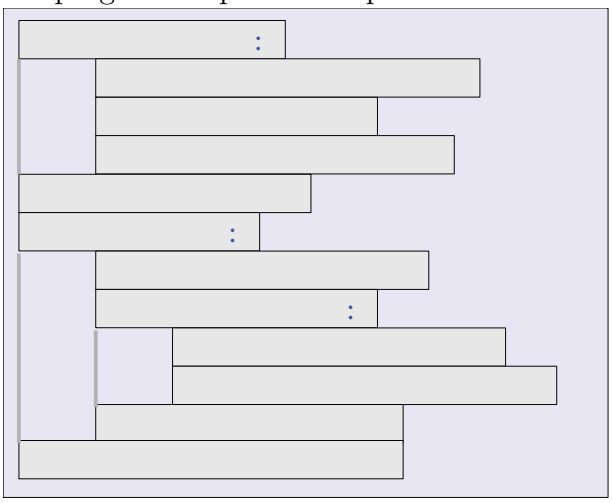
Algorithme

Données : $a \in \mathbb{N}$
Résultat : afficher la moitié d'un nombre pair

```
1 a←25
2 si reste(a/2)=0 alors
3 |  b←a/2
4 |  afficher a est pair
5 |  afficher sa moitié vaut b
6 sinon
7 |  afficher a est impair
```

python
```python
a=25
if a%2==0 :
    b=a/2
    print("a est pair")
    print("sa moitiée vaut ",b)
else :
    print("a est impair")
```

Un programme peut alors prendre un structure à plusieurs niveaux :



5 Python et la programmation orientée objet

La programmation orientée objet est une approche de programmation qui sera vue en deuxième année. Elle présente néanmoins quelques avantages que nous utiliserons dès cette année.

Spécificité

En python, *tout est objet*. Et tous objet est caractérisés par des attribus et des méthodes.

- un attribut est une propriété particulière de l'objet ;
- Une méthode est assimilable à une fonction spécifique de l'objet.

- l'attribut peut être récupéré avec la syntaxe : objet.attribut
- la méthode peut être appelé avec la syntaxe : objet.méthode(argument)

```
>>> a=1.5+3.j
>>> a. real
1.5
>>> a. imag
3.0
>>> a. conjugate ()
(1.5-3j)
>>> a.conjugate
<built-in method conjugate of complex object at 0x02D68938>
```