Informatique en CPGE (2018-2019)
Algorithmique et programmation de base (partie 1)

1 Introduction

Informatique : traitement automatisée de l'information.

Algorithme : ensemble de règles opératoires dont l'application permet de résoudre un problème en un nombre fini d'opérations.

Programme : séquences d'instructions et de données enregistrées sur un support et susceptibles d'être traitée par un ordinateur.

Données : objets manipulés par le programme.

Le programme est la traduction d'un algorithme dans un langage de programmation qui impose une syntaxe rigoureuse.

Objectif : écrire un programme dans un langage donné dont l'exécution permet de résoudre un problème.

Les étapes : A partir du problème à étudier, on commence par identifier et structurer les objets qui interviennent (les données), puis on détermine la méthode permettant de résoudre le problème (l'algorithme), et enfin on combine le tout en un programme.

Lorsque l'algorithme est déterminé, il faut répondre à plusieurs questions avant de passer à la suite.

- L'algorithme est-il correct, donne-t-il la bonne réponse dans tous les cas ? Ce n'est pas toujours facile de répondre à cette question.
- Est-ce que l'algorithme est suffisamment efficace : cela demande une analyse de complexité. On se limitera aux cas faciles.
- Est-ce que l'algorithme est facile à implémenter ? Un algorithme simple est toujours préférable.

On peut alors passer aux instructions dans le langage de programmation choisi.

Une fois le programme écrit, on effectue des tests afin de vérifier que son comportement est conforme à ce que l'on attend. Eventuellement on corrige et on recommence. En général, les erreurs de syntaxe, qui sont les plus courantes, seront détectées et le programme ne s'exécutera pas. Le problème de la sémantique est plus compliqué à régler ; le programme peut avoir un sens mais ne pas s'exécuter, ou ne jamais s'arrêter, ou encore fonctionner mais ne pas donner le résultat attendu et ce dernier cas est le pire.

2 Les Langages

2.1 Constituants

Une langue :

- **Syntaxe** : des mots (noms, verbes, adjectifs, ...) et des règles de grammaire permettent de former des phrases.
- **Sémantique** : une phrase bien formée n'a pas toujours un sens.

Catégories de langages :

- langue naturelle : syntaxe et sémantique ne sont pas rigoureuses. On peut se comprendre même avec des phrases pas toujours correctes. Le récepteur est un être humain intelligent.

- langage informatique : syntaxe rigoureuse et rigide. La sémantique est complexe et difficile à bien appréhender. Le récepteur est une machine sans intelligence.

- langage de description d'algorithme : une syntaxe moins rigide mais assez stricte pour ne pas donner d'ambiguité sur le sens de ce qui est écrit.

2.2 Langage de programmation

Il existe des centaines de langages de programmation. Certain comme le C sont dits de "bas niveau", c'est-à-dire assez proche du langage binaire de la machine et donc assez complexe, d'autres comme Python sont dits de "haut niveau", plus éloigné du binaire donc plus souple, utilisant des opérations abstraites déjà prévues dans le langage.

Le langage utilisé sera Python ; c'est un langage interprété, la séquence d'instructions ou code source est exécutée directement. D'autres langage sont compilés, le code est d'abord traduit en langage machine. Les programmes compilés sont en général plus rapide mais il est plus difficile d'y déceler d'éventuelles erreurs.

Python est un langage général utilisable dans de nombreux domaines et portable (le même code source peut être interprété sous Windows, linux, Mac OS, ...). C'est un langage qui a beaucoup évolué depuis son introduction par Guido von Rossum en 1990. La dernière version est Python 3.7.0.

Un langage de programmation comprend :

— des **commentaires** : ils servent à expliquer (en langue naturelle) le comportement du programme et ne jouent aucun rôle dans le comportement du langage. En Python, un commentaire est un texte précédé du symbole #. Ce texte n'est pas interprété.

— des noms appelés **identificateurs** : ils désignent des variables, des fonctions,...

— des **expressions** : ce sont des combinaisons de noms avec des valeurs et des opérateurs. (par exemple "x+2")

— des **instructions** : ce sont des commandes qui demandent de faire quelque chose. (par exemple "z=x+2" demande d'affecter à z la valeur de x+2).

— **l'indentation** : cela consiste à laisser une ou plusieurs espaces au début des lignes. L'indentation permet la lisibilité du programme ; en Python, elle fait partie de la syntaxe et remplace l'utilisation d'accolades en C++ ou en Java.

Un programme **source** est une phrase de ce langage. C'est une suite de caractères, (un programme écrit en Python est un script, soit une suite de définitions et d'instructions), pouvant être écrite dans un éditeur de texte et compréhensible par l'humain. Un **interpréteur** pour Python, ou un **compilateur** pour d'autres langages, permet ensuite de traduire et d'exécuter le programme sur une machine.

3 Algorithme

3.1 Définitions

Le mot "algorithme" vient du nom de l'auteur persan Al-Khuwarizmi Mohammed (né vers 780 - mort vers 850) qui a écrit en langue arabe le plus ancien traité d'algèbre (de l'arabe "al jabar" qui signifie "la transposition" et du grec "arithmos" qui signifie "nombre") dans lequel il décrivait des procédés de calcul à suivre étape par étape pour résoudre des problèmes ramenés à des équations.

Un algorithme peut se définir comme étant l'ensemble des règles et instructions à suivre, effectivement exécutables, permettant d'obtenir un résultat clairement défini en un nombre fini d'étapes.

Il ne faut pas confondre "algorithme et "programme". Un algorithme peut s'exprimer avec une notation indépendante de tout langage de programmation et il s'implémente, (c'est la "mise en œuvre"), dans un programme qui lui est écrit dans un langage particulier compréhensible par une machine. Un algorithme n'est pas compréhensible par un ordinateur qui ne peut qu'exécuter un programme. De plus, un algorithme

doit toujours donner une réponse après un nombre fini d'opérations, alors que l'exécution d'un programme peut conduire à une boucle infinie et ne jamais s'arrêter.

Un algorithme se compose de **données** et d'**instructions**.

Dans les problèmes que nous allons résoudre, les données peuvent être de différents types. On rencontre les types **nombre** ou **texte** (chaînes de caractères), ou le type **logique** (vrai ou faux), ou le type **graphique** (des points).

Les instructions à donner pour une exécution automatique doivent être suffisament simples pour être traduites dans un langage de programmation. Les instructions élémentaires sont :

— les instructions d'entrée et de sortie : l'entrée (ou la lecture) des données peut se faire en interrogeant l'utilisateur ou par extraction à partir d'un fichier ; la sortie (ou l'écriture) peut se faire par un affichage sur l'écran, une impression sur papier, une écriture dans un fichier ou bien les résultats obtenus peuvent aussi rester en mémoire, (la sortie permet à l'utilisateur de voir les valeurs des variables après le traitement ou en cours de traitement si on veut contrôler l'exécution du programme) ;

— les affectations et les calculs : l'affectation d'une donnée dans une variable consiste en la création d'une zone de mémoire (dans la machine) à laquelle on donne un nom, ensuite diverses formules permettent d'effectuer des calculs à l'aide d'opérations élémentaires, par exemple pour déterminer la valeur d'une fonction pour une valeur donnée de la variable.

3.2 Exemple

Ecrire un algorithme de résolution dans $\mathbb{R}$ d'une équation du second degré.

4 Eléments de base en Python

Lorsqu'on ouvre **Idle** pour Python, on peut écrire directement des instructions qui sont interprétées à chaque retour à la ligne. On peut aussi ouvrir une nouvelle fenêtre en allant dans le menu **File**, puis **New Window**, (ou au clavier avec "ctrl + N") et entrer toutes les instructions souhaitées. On enregistre ensuite le fichier par **Save As . . .** ; une fenêtre s'ouvre, où l'on choisit le dossier d'accueil et on entre un nom pour le fichier avec l'extension .py ("nom.py"). On peut alors lancer le programme en allant dans le menu **Run**, puis **Run module** ou en appuyant sur la touche F5 du clavier.

En sortie d'un programme, on peut écrire les résultats obtenus dans un fichier ; ceci sera étudié plus tard. On peut aussi les afficher sur l'écran avec la fonction **print**. Par exemple `print('bonjour')` ou `print(25+36)` ou encore `print('La somme est égal à',25+36)` qui affiche "La somme est égale à 61".

4.1 Objets, expressions et types numériques

Un programme en Python manipule des objets. Chaque objet a un **type** qui indique l'ensemble des valeurs dont ils font partie et l'ensemble des propriétés qui les caractérisent. C'est le type qui définit ce que le programme peut faire avec un objet.

Les types peuvent être scalaires ou non scalaires. Pour les scalaires, il y a :

- le type `int` (abréviation de l'anglais integer), pour représenter les nombres entiers illimités et on écrit par exemple 5 ou 23214 ou -7 ;
- le type `float` pour représenter les nombres réels et on écrit par exemple 5.2 ou -26.721. Les nombres du type `float` sont stockés dans la machine sous la forme de "nombres en virgule flottante" entre $-1,7 \times 10^{308}$ et $1,7 \times 10^{308}$, ceci sera étudié plus tard.
- le type `bool` pour représenter les valeurs booléennes `True` et `False` ;
- le type `None` qui n'a qu'une seule valeur.

On forme des **expressions** en combinant les objets avec des **opérateurs**; par exemple `5.2+3.4` représente l'objet `8.6` de type `float`. L'opérateur == est utilisé pour déterminer si deux expressions sont égales et l'opérateur != pour déterminer si deux expressions ne sont pas égales.

La fonction `type` permet de déterminer le type d'un objet :

```
>>>type(5.2)
<class 'float'>
>>> type(-18)
<class 'int'>
```

Les opérateurs sur les types `int` et `float` sont :
- l'addition + , la soustraction - , la multiplication * , l'exponentiation ** dont le résultat est un `float` si l'un des termes est un `float` et sinon un `int` ;
- la division / donc le résultat est toujours un `float` ;
- la division entière // , l'opération modulo % : utilisés avec des nombres entiers, on obtient le quotient et le reste, de type `int`, dans la division euclidiennne : 17//5=3 et 17%5=2 ou (-17)//5=-4 et (-17)%5=3. Autre exemple : 7.4//3=2.0 (le résultat est un nombre entier mais de type `float`).

Les opérateurs précédents suivent les règles de priorité habituelles et on peut naturellement utiliser des parenthèses.
Il existe aussi des opérateurs de comparaison qui ont le sens usuel : < , <= , > , >= .

Pour le type `bool` les opérations sont :
- **a and b** donne `True` si a et b sont `True` et `False` sinon ;
- **a or b** donne `False` si a et b sont `False` et `True` sinon ;
- **not a** donne `True` si a est `False` et `False` si a est `True`.

4.2 Variables et affectation

Une **variable** permet d'associer un nom avec un objet. Si on écrit `pi = 3.14159`, on lie le nom `pi` à un objet de type `float` ; on a alors une variable dont le nom est "pi" et la valeur est 3,14159.
Une **affectation** associe le nom à gauche du signe = avec l'objet représenté par l'expression qui est à droite du signe = ; par exemple `aire=pi*5**2`.

Afin de faciliter la lecture d'un programme, il est important de bien choisir les noms utilisés. On peut écrire :

```
a=3.14
b=5.3
c=a*b**2
```

mais il est préférable d'écrire, en ajoutant des commentaires si nécessaire :

```
# Calcul de l'aire d'un disque
pi=3.14
rayon=5.3
aire=pi*rayon**2
```

Il est important d'ajouter des commentaires qui seront ignorés par le compilateur mais qui seront bien utiles à la compréhension si quelqu'un lit le programme ou si on le relit dans trois mois. Pour cela on utilise le symbole # et le texte qui suit jusqu'à la fin de la ligne n'est pas interprété par Python.

Python autorise des affectations multiples : `x, y = 3, 7` associe x à 3 et y à 7.
Ceci permet de faire l'échange : `x, y = y, x` de manière très simple (et bien utile !).

4.3 Type str et fonction input

4.3.1 Type str

Python n'a pas de type pour les caractères. Le type `str`, (abréviation de l'anglais string, chaîne en français), est utilisé pour représenter les chaînes de caractères. (Un caractère est une chaîne de longueur 1). On peut écrire `'bonjour'` ou `"bonjour"`.

Si on tape `3*'a'`, on obtient `'aaa'` et si on tape `'a'+'b'` on obtient `'ab'`. On dit que les opérateurs + et `*` sont **surchargés**, cela signifie qu'ils ont une action différente en fonction du type des objets auxquels on les appliquent. Ici, ce n'est plus l'addition et la multiplication des nombres mais + permet la concaténation de deux chaînes, `*` permet la répétition d'une chaine.

La fonction `len` permet de trouver la longueur d'une chaîne : `len('bonjour')` donne 7.

L'indexation permet d'extraire un caractère d'une chaîne. Attention, le premier caractère est indexé par 0, donc `'bonjour'[2]` donne `'n'`, soit le troisième caractère. On peut aussi indexer à partir de la fin de la chaîne : par exemple `'bonjour'[-1]` donne `'r'`.

Pour extraire une sous-chaîne, on utilise `'chaîne'[début:fin]` ; l'extrait commence à l'index `début` et se termine à l'index `fin-1` : par exemple, `'bonjour'[2:5]` donne `'njo'`. Et donc `'bonjour'[0:len('bonjour')]` donne `'bonjour'`.

4.3.2 Fonction input

La fonction `input` permet d'obtenir une chaîne de caractères entrée par l'utilisateur. Le code suivant explique l'utilisation :

```
>>>nom=input('Entrer votre nom :')
Entrer votre nom : toto
>>> print(nom)
toto
```

Afin de récupérer un nombre, il est nécessaire de procéder à une conversion :

```
>>> a=input('Entrer un nombre entier : ')
Entrer un nombre entier 5
>>> print(3*a)
'555'
>>> a=int(a) #conversion d'un type str en type int
>>> print(3*a)
15
```

La ligne `a=int(a)` permet de convertir le caractère "5" en un nombre entier égal à 5. On peut aussi écrire plus rapidement : `a=int(input('Entrer un nombre entier : '))`.

5 Instruction conditionnelle

Dans tous les exemples précédents, chaque instruction est exécutée dans l'ordre d'apparition. Les programmes avec branchement apportent un intérêt plus important. L'un des plus simples utilise l'instruction conditionnelle : un test est effectué (une expression qui a la valeur `True` ou `False`) ; si la valeur est `True`, alors un bloc de code est exécuté et un bloc optionnel peut être effectué si la valeur est `False`.

On utilise l'instruction "Si ... alors ... " , (If ... then ...),
ou bien l'instruction "Si ... alors ... Sinon" (If ... Then ... Else).
L'algorithme correspondant s'écrit ainsi :

```
Si <condition> alors
    ... Instructions A ...
Sinon
    ... Instructions B ...
```

La traduction de l'algorithme en Python est :

```
if (condition):
    action1
    action2
    action3
else:
    action4
    action5
```

La condition peut s'écrire avec des opérateurs logiques, par exemple `(a > 10 and a < 20)`.
Remarque : l'**indentation** est sémantiquement importante ; Python est très particulier dans cette utilisation (la plupart des autres langages utilisent des parenthèses ou des accolades). Toutes les instructions au même niveau d'indentation appartiennent au même bloc.
Les programmes suivants ne sont donc pas identiques ; les commenter.

```
if x%2==0:
    print("x est pair")
    if x%3==0:
        print("x est divisible par 6")
    else:
        print("x est impair")
```

```
if x%2==0:
    print("x est pair")
    if x%3==0:
        print("x est divisible par 6")
else:
    print("x est impair")
```

Note : il existe une syntaxe plus compacte d'une alternative, par exemple :
`mini=x if x<y else y`