STS SIO 1°année
Lycée Merleau-Ponty
Rochefort

Initiation à l'algorithmique et à la programmation structurée

Ressources langage Python

Bertrand BOUCHEREAU

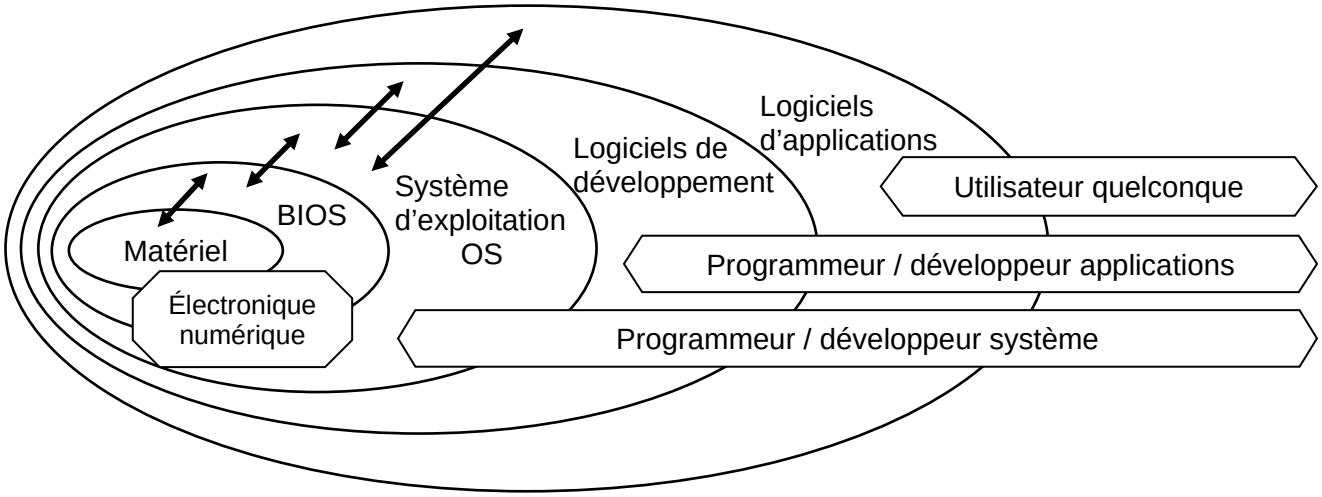
SOMMAIRE

Le BIOS

BIOS = **Basic Input Output System**
On désigne par BIOS un circuit intégré reprogrammable (EEPROM) qui contient les informations nécessaires au bon fonctionnement de la carte mère (carte supportant le microprocesseur).
Son rôle est de gérer les opérations de base (initialisation et vérifications matérielles, contrôle des fréquences, horloges, mémorisation de la date, heure…).
Ces données sont utilisées par le système d'exploitation pour les opérations élémentaires de communication avec les périphériques internes (disques, écran, clavier...) ou externes (USB...).
On peut y accéder lorsque la machine démarre **(Boot)** pour y imposer des réglages **matériels (Hard)**. Toute modification est fortement déconseillée au profane !

Depuis les années 2000, une évolution du BIOS nommée **UEFI** (Unified Extensible Firmware Interface) s'est généralisée. Elle apporte diverses fonctionnalités plus ou moins utiles à ce stade du démarrage d'un ordinateur (interface graphique, formats de disques étendus, prise en charge de pilotes spécifiques…).

Interactions matériel / logiciel



Un système informatique se décompose en deux grandes parties :

- Les composants **matériels (Hardware)** : unité centrale, mémoires, interfaces, disques, écran, clavier, souris, scanner, imprimante…
- Les composants **logiciels (Software)** : ensemble des programmes et applications stockées dans une mémoire sous forme de données binaires directement exécutables par le processeur.

Un ensemble matériel privé de logiciel exécutable serait complètement inutile !

Système d'exploitation

Souvent désigné par l'acronyme OS (**O**perating **S**ystem), le **système d'exploitation** constitue la première couche logicielle et permet de gérer les échanges entre le processeur, les périphériques et l'utilisateur. Il est habituellement stocké sur le disque dur ; un ensemble de fichiers cachés sur le secteur de démarrage (primary bootstrap) permet le lancement automatique du système.

Les logiciels de développement et d'applications prennent appui sur le système d'exploitation pour pouvoir accéder aux ressources matérielles.

Du point de vue de l'utilisateur final, toutes les couches inférieures à celle du logiciel utilisé sont invisibles. L'utilisation d'un ordinateur par une personne non spécialiste formée seulement au logiciel final est alors possible.

Langage informatique

Un logiciel ou programme directement exécutable se résume à une suite d'informations binaires regroupées dans un ensemble de fichiers. Pour que ces informations prennent un sens pour l'homme, il a fallu inventer la notion de langage informatique.
Il existe aujourd'hui un grand nombre de **langages évolués** qui permettent une programmation plus ou moins aisée d'une machine informatique. Constitués pour la plupart de mots empruntés à la langue anglaise, ils sont régis par une **syntaxe** très **stricte** et impérativement **structurée**.

Notion de programme : texte constitué d'une **suite ordonnée d'instructions**.

Un langage informatique évolué est donc un logiciel comprenant :

- Un **éditeur** de textes (pour écrire le **programme ou code source**)
- Un **compilateur** (ou interpréteur dans certains cas) pour transcrire le programme source en codes binaires exécutables (parfois appelé **programme ou code objet**)
- Un ensemble plus ou moins fourni d'aide au développement (aide en ligne, debugger, bibliothèques, modules prêts à l'emploi…).

Un programme est donc une suite d'instructions (appelée **code**) organisée suivant la structure et la syntaxe du langage utilisé.

Programmation structurée

Le but de tout programme est de commander l'exécution d'un processus déterminé à la machine. L'analyse indispensable du processus est menée méthodiquement à l'aide de **l'approche algorithmique**.

Cette approche débouche sur la production d'un **algorithme** réalisable (codable) avec n'importe quel **langage structuré**.

Il est démontré (et nous admettrons) que l'analyse d'un processus quelconque peut être réalisée en s'appuyant sur seulement 3 structures universelles.
La programmation structurée ou **procédurale** repose sur ces trois structures de base, énoncées ci-dessous :

- ⇨ **La séquence**
- ⇨ **L'alternative**
- ⇨ **L'itération**

Choix d'un premier langage de programmation

Le choix d'un langage de programmation doit être conduit par le projet visé (plate-forme, performances recherchées, interactions utilisateurs, type de modélisation, etc...). Ainsi, il n'existe pas d'échelle de classement des langages. Chacun possède ses propres caractéristiques qui peuvent apparaître comme avantages ou inconvénients suivant l'objectif poursuivi.

Voici quelques arguments en faveur du langage Python pour débuter :

- portabilité : il fonctionnera sur toutes les plate-formes courantes (Windows, Apple, GNU/Linux)
- gratuité : aucune contrainte pour le travail personnel
- syntaxe simple, génératrice de bonnes pratiques et dotée de fonctions très riches
- multi-paradigme : programmation procédurale ou modélisation « orientée objet »
- extensible et en évolution : de nombreuses librairies d'objets existent et continuent d'être développées.

Programmation structurée avec Python 3

Dans un premier temps, nous allons nous limiter à la description des éléments de base (communs à beaucoup d'autres langages) qui donnent accès aux fonctions essentielles d'un langage de programmation.

Commentaires : Il est possible d'insérer des commentaires dans le code d'un programme. En Python, ils doivent être précédés du caractère # (hash). Tout le texte entré après sera ignoré par le compilateur. Un programme commenté sera plus facile à maintenir et garantit un travail possible à plusieurs. **Commenter son code est obligatoire et correspond à une norme de bonnes pratiques dans le domaine du développement.**

I – La manipulation des données élémentaires

L'ordinateur n'est avant tout qu'un puissant calculateur. Il ne traite donc exclusivement que des données chiffrées. A l'aide d'un code de représentation (**code ASCII** pour la base), les lettres qui forment les textes sont transcrites en nombres binaires pour être traitées. Les données manipulées sont alors contenues dans des variables.

⇨ **Variables** : objet numérique ou alphanumérique défini au sens d'une variable mathématique dont le contenu n'est pas connu *a priori*. Une variable est un espace réservé en mémoire pour y placer une donnée élémentaire. Son nom doit être vu comme une étiquette associée à cet espace.

Dans beaucoup de langages, il convient de **déclarer** ces objets en tête d'un module de code en précisant son **type** et son nom. En Python, c'est inutile car une variable est automatiquement créée avec le type qui correspond le mieux à la valeur fournie ; c'est un processus de **typage dynamique**.

Exemple :
```
nb = 12                    # donner à nb la valeur entière 12
pi = 3.14159               # assigner à pi la valeur décimale 3,14159
message = "Voici le résultat :"   # affecter à message la chaîne de
                                    caractères "Voici le résultat :"
```

⇨ Identificateur et affectation

Le **nom** donné à une variable est aussi appelé « **identificateur** » puisqu'il permet d'identifier une donnée. Au cours du déroulement du traitement, une **donnée est placée dans une variable** : cette opération porte le nom d'**affectation**.

⇨ Types de données

Le type d'une donnée définit quelle forme de donnée sera mémorisée dans cette variable et fixe ainsi la place réservée dans la mémoire de travail. Il existe de nombreux types ; les principaux en Python sont donnés ci-dessous :

- `integer` : nombres entiers signés sur 32 bits de -2^{31} à $(2^{31}-1)$
- `float` : nombres en virgule flottante (séparateur point décimal), notation scientifique (mantisse + exposant), codage sur 64 bits : de 10^{-308} à 10^{308}
- `string` : chaîne de caractères délimitées par simple quote (') ou double quote ("), échappement par \
- `boolean` : type logique (valeur `True` ou `False`).

Types de données avancés

Dans un langage moderne comme Python, il existe des types de données évolués associés à de nombreuses primitives. Ces types de données complexes donnent accès à de puissantes fonctions qui permettent la production d'un code simplifié et performant. Il sera nécessaire de consulter la documentation pour plus de détails sur leur mise en œuvre.

Type avancé	**Mutable** (ajout, supp. éléments...)	**Immuable**
Itérable (opérateur for..in.. applicable)	**list** : liste ordonnée d'éléments multiples de types variés (apparenté à un tableau) `list_ex = [123,'abcd',10.2,'d',...]`	**tuple** : liste constante `tuple_ex = (123,'hello')`

Non ordonné (1 seule instance conservée d'un même élément)	**set** : collection non ordonnée d'objets uniques
	`set_ex = {'a','b',...}`
	dictionnaire : suite de paires clé : valeur
	`dict_ex = {'name':'red','age':10,...}`
	accès aux données par `dict_ex[name] → 'red'`
		frozenset : set constant
		définition par `frozenset(set)`
		`fs_ex = frozenset({12,'h'})`

II – Trois instructions de base

Il existe 3 instructions universelles présentes dans tous les langages. Ces instructions permettent de manipuler le contenu des variables.

① Lire identificateur

→ permet à l'utilisateur d'**entrer** généralement au clavier **une valeur** qui va affecter la variable dénommée par l'identificateur. En Python, on peut faire appel à la **fonction `input`** qui attend que l'utilisateur entre une réponse.

Exemple : `reponse = input('Entrez votre nom :')`

A l'issue de l'exécution, la variable nommée `reponse` contiendra la saisie clavier de l'utilisateur (ici le nom de l'utilisateur).

Note 1 : La chaîne de caractères passée en argument « Entrez votre nom : » sera affichée préalablement à l'invite de saisie utilisateur.
Note 2 : Le type de donnée renvoyé par la fonction input est toujours de type `string`.

② Donner à identificateur la valeur

→ permet de donner directement une valeur à une variable.

Exemple :
```
C = 0       # on place la valeur 0 dans C
C = I + 1   # on place le résultat de I+1 dans C
I = I + 1   # nouvelle valeur de I = (ancienne valeur de I) +1 → opération d'incrémentation
```

③ Écrire valeur ou expression ou identificateur

→ permet d'**écrire** généralement sur l'écran, mais aussi sur l'imprimante ou tout autre périphérique une **valeur** en clair ou contenue dans une variable ou définie par une expression.
En Python, on peut faire appel à la **fonction `print`** :

Exemple :
```
print('Bonjour') # on affiche la chaîne Bonjour dans la fenêtre d'exécution
print('Le résultat est :', I)   # on affiche le message suivant : Le résultat est : 5
                                # si I vaut 5.
```

Il existe bien sûr de nombreux outils pour communiquer avec l'utilisateur dans une fenêtre système (comme le contrôle *zone de texte* ou le contrôle *button*) mais cela nécessite de savoir utiliser les objets et fait appel à une certaine expérience.

III – Les trois structures de base

① La séquence

La séquence ou **structure linéaire** est simplement une **suite chronologique** d'instructions (d'actions simples) ; elle constitue le corps de tout programme. Elle est matérialisée par l'écriture d'**une instruction par ligne**.

② L'alternative

Cette structure permet d'**orienter** le déroulement du programme vers le traitement d'un ensemble d'instructions en fonction du résultat d'un **test comparatif** entre deux grandeurs ou objets de **même type**.
Elle est généralement composée de 3 mots réservés : IF … THEN … ELSE.

Représentation littérale	Écriture en code Python

Si <condition> **Alors**
  Instruction A
Sinon
  Instruction B
FinSi
	If (<condition>):
	␣␣␣␣Instruction A
	Else:
	␣␣␣␣Instruction B

Remarque : Dans le cas de plusieurs instructions `if` imbriquées, `else: … if` se contracte en `elif`.

Contrairement à beaucoup d'autres langages, il n'existe pas en Python de mot délimiteur (comme FinSi) pour marquer les blocs d'instructions. La délimitation des blocs est réalisée par l'**indentation obligatoire** (décalage des instructions par des **espaces** ␣␣␣␣), ce qui correspond à une excellente pratique améliorant grandement la lisibilité du code !

A noter :
→ Un bloc d'instruction débute après le caractère (**:**) et se termine après la **dernière instruction indentée**.
→ Cette règle de syntaxe est valable pour toutes les structures du langage Python.

③ L'itération

L'itération ou structure itérative permet la **répétition** de tout ou partie d'un processus (une ou plusieurs instructions). Cette structure aussi appelée boucle, constitue l'essence même de **l'intérêt du calculateur** : effectuer très **rapidement** et de manière **fiable** la même tâche un **grand nombre de fois**.

Dans la plupart des langages structurés, il existe 3 grands types de boucle en fonction de l'endroit où l'on teste la condition (au début ou à la fin) et du nombre de passages (connu ou pas). En Python, une seule écriture est disponible pour l'itération classique.

Boucle Tant que…Faire

Cette boucle n'utilise pas de compteur et teste la condition au début de son exécution. Ainsi, le groupe d'instruction peut ne pas être exécuté du tout.

Représentation littérale	Écriture en code Python

Tant que <condition vraie> **Faire**
  Instruction A
  Instruction B
FinFaire
	while (<condition>):
	  Instruction A
	  Instruction B

A noter :
Le traitement réalisé par le bloc d'instruction doit obligatoirement modifier les opérandes du test sans quoi la sortie de boucle ne se produira jamais !

Exemple :
```
while (I<10):
    print I
    I=I+1
```

IV – Écriture des tests et conditions

Les tests doivent être exécutés avec des objets de types compatibles (on ne compare pas une chaîne de caractères avec un nombre !) et suivant la syntaxe donnée dans le tableau suivant :

Syntaxe	Signification
==	égal
!=	différent
>	Supérieur
<	Inférieur
>=	Supérieur ou égal
<=	Inférieur ou égal
and	ET logique
or	OU logique
not	Négation logique
is	Objet identique
is not	Objet différent
Item in	Item existe dans type collection set, tuple, list, string

Le **résultat** d'un test est toujours de **type logique** (0 ou 1), c'est-à-dire `boolean`.

Les variables booléennes ne peuvent contenir que 2 constantes prédéfinies :

- `True` pour vrai
- `False` pour faux

On différencie :

- les tests simples : `A==B, A>=B, cpt!=0…`
- les tests composés : `(A>7) and (A<50)…`

On peut aussi tester directement des variables booléennes sans avoir à préciser la valeur d'égalité recherchée :

```
If (var_test==True):      est équivalent à :    If var_test:
If (var_test==False):     est équivalent à :    If not var_test:
```

V – Les sous-programmes

Lorsqu'une partie d'un **algorithme** se reproduit **plusieurs fois** au cours du même traitement, il est fortement conseillé d'écrire **un sous-programme** (SP).

Le SP est constitué de l'ensemble d'instructions qui se répète que l'on **écrit une seule fois**. On peut ensuite utiliser le SP à tout moment dans le traitement en procédant simplement à un **appel de SP** (en général par son nom). Cette technique simplifie beaucoup la structure du programme et facilite sa **mise au point et sa maintenance** en le rendant plus lisible.
Dans un langage structuré comme Python, on déclare les sous-programmes en tête du programme ou du module de code. Le module qui contient toutes les déclarations et permet le lancement de l'application ainsi créée est appelé programme ou module principal (**Main**).
En général, on distingue 2 types de sous-programmes : les procédures et les fonctions.

① Les procédures

C'est un simple sous-ensemble composé d'un ou plusieurs groupes d'instructions structurées.
En Python, une procédure se déclare par le mot **`def`** pour *define* en anglais.

Exemple :
```
def affiche(prenom):
    print('Bonjour', prenom)
```
Utilisation :
```
affiche('le monde')
```

→ On obtient donc *Bonjour le monde* à l'écran lors de l'exécution de l'appel de la procédure (code de droite).
A la lecture du mot `affiche`, le compilateur exécute les instructions constituant le corps de la procédure nommée `affiche`.

② Les fonctions

Les fonctions sont des procédures d'un type particulier dont le nom est une variable capable de renvoyer une valeur. L'appel d'une fonction est en fait une opération d'affectation ou d'affichage direct.
En Python, une fonction se déclare de la même façon et se termine par le mot-clé `return` suivi de la valeur ou de l'objet à renvoyer.

Exemple :
```
def calcul(I):
    result=(I+2)/3
    return result
```

Utilisation :
```
N=calcul(7)
print(N)
```
ou bien directement : `print(calcul(7))`

→ Résultat à l'écran : 3.0

A la lecture du mot `calcul`, le compilateur exécute les instructions constituant le corps de la fonction nommée `calcul` et se sert de son nom pour passer une valeur (appelée argument de sortie) au programme principal.

③ Portée des variables

Il est possible d'utiliser des variables dans le corps des SP. Dans ce cas, ces variables n'existent que pendant l'exécution du SP seulement. Elles ne sont donc pas accessibles en dehors : on parle de **variables locales**.
Une **variable** utilisée dans le **bloc principal** est accessible en **lecture seule** depuis les SP et existe pendant toute la durée de l'exécution : on parle de **variable globale**.

④ Échanges d'arguments avec les sous-programmes

Afin de rendre un SP le plus **indépendant** possible (ce qui permet de le réutiliser dans plusieurs programmes), il est intéressant de lui fournir des **valeurs de travail** (appelées **arguments**) au moment de son appel.
Ces arguments prennent la forme de variables locales au SP et doivent être déclarés entre parenthèses accolées au nom du SP.
→ dans les exemples précédents, les arguments sont `prenom` pour la procédure `affiche` et `I` pour la fonction `calcul`.
Il est possible de passer plusieurs arguments au même SP en les séparant par des virgules.

Conclusion

L'écriture d'un programme performant passe par une analyse algorithmique soignée et rigoureuse, un découpage modulaire et/ou en sous-programmes astucieux, une application sans failles des structures et un respect infini de la syntaxe du langage employé.