Introduction à l'informatique

1 Notion d'ordinateur

On appelle aujourd'hui *ordinateur* toute machine éléctronique qui se décompose en *architecture de Von Neumann* [*] :

- une unité arithmétique et logique, qui effectue les opérations de bases, séquencée par une unité de contrôle (processeur)
- une mémoire, divisée en RAM (très rapide, mais volatile) et ROM (très lente, mais permanente) (mémoire vive et disque dur, disque ssd, clef usb, *etc.*)
- des entrées/sorties pour communiquer (souris, clavier, écran,*etc.*)

Tous ces éléments sont mis en relation par des bus.

Le processeur sert à exécuter les instructions élémentaires : faire une addition, accéder à une case mémoire, *etc.* Ce processeur est cadencé par une horloge, dont la fréquence est précisée ; un processeur cadencé à 2GHz fera donc 2 milliards d'opérations élémentaires par seconde.

Toutes les données traitées par un ordinateur sont manipulées sous forme binaire, *i.e.* sous forme d'une suite de 0 et 1. Chaque 0 et 1 s'appelle bit. Une série de 8 bits s'appelle un octet, qui est stocké dans une case mémoire à laquelle on accède par son adresse.

Les données stockées dans la RAM servent à exécuter directement un programme : elles sont effacées dès que l'ordinateur est éteint, ou peuvent être remplacées par d'autres si la mémoire est pleine.

Les données stockées dans la ROM sont elles persistantes. Elles sont stockées par exemple de façon éléctromagnétique dans un disque dur. En revanche, l'accès à ce type de mémoire es très lent (de l'ordre de 100 000 fois plus lent que la RAM).

2 Notion de programme

Les opérations qu'un processeur doit effectuer sont détaillés dans un programme, lui-même stocké dans la ROM. Au moment de le lancer, les opérations transitent par la RAM.

Le processeur ne sait effectuer que des opérations sous forme binaire : c'est ce qu'on appelle le langage machine. L'humain, lui, ne sait pas programmer directement en langage machine (sauf exception).

Le premier langage de programmation a été l'assembleur. Il est très proche du langage machine, et permet de manipuler directement les cases mémoires (registres). Voici un exemple de programme écrit en assembleur :

```
org  0x0100

; ceci est un programme affichant
; simplement "Hello World"
```

∗. John Von Neumann, 1903 – 1957, Américano-Hongrois

```
mov  dx, texte
mov  ah, 0x09
int  0x21      ; afficher le hello world
ret            ; fin du programme

texte: db 'Hello, World !!', 10, 13, '$'
```

L'avantage de l'assembleur est d'être très proche du langage machine, et donc d'être très rapide ; par contre, il est très difficile à manipuler, et n'a pas de commandes de haut niveau.

On utilisera donc en général plutôt des langages haut niveau : Python, C++, OCaml, Haskell,

Un programme écrit dans un de ces langages est transformé en programme assembleur par un compilateur ou un interprète, puis en langage machine.

Le langage choisi sera Python : c'est un langage très utilisé, créé en 1989. C'est un langage interprété, et qui sera donc plutôt lent, mais dont l'apprentissage est rapide. On utilisera l'IDE Pyzo pour travailler.

3 Pyzo

Pyzo est un éditeur pour Python, qui permet d'écrire du code de façon pratique, et de l'interpréter directement. En cas d'erreur, il pourra même nous dire où elle se situe.

Pyzo se divise en trois parties :

- à gauche se trouve l'éditeur ; c'est là qu'on écrira nos programmes
- en haut à droite se trouve la console, ou shell ; c'est ici qu'on pourra donner directement des commandes à traiter, ou interpréter notre programme
- en bas à droite se trouve un navigateur de fichier, qu'on pourra le plus souvent fermer.

Pyzo a quelques raccourcis utiles :

- Ctrl-c pour copier
- Ctrl-x pour couper
- Ctrl-r pour commenter
- Alt-Enter pour exécuter la ligne ou la sélection
- Ctrl-i pour interrompre une commande en cours (en cas de boucle infinie)
- Ctrl-v pour coller
- Ctrl-a pour tout sélectionner
- Ctrl-t pour décommenter
- Ctrl-e pour exécuter le fichier entier
- Ctrl-k pour vider la mémoire (sinon, on garde en mémoire toutes les variables utilisées, ce qui peut être une cause d'erreur

4 Initiation à Python

Pour l'instant, on va se contenter de travailler dans le shell. Nous passerons à l'éditeur quand nous aurons à travailler avec des choses plus avancées.

4.1 Types de données

Python est ce qu'on appelle un langage typé : tout élément utilisé en Python a un type qui doit être respecté.

Pour obtenir le type d'un objet, on peut taper `type(objet)`.

EXERCICE

En tâtonnant avec la commande `type`, compléter le tableau suivant :

Description	Python	Exemples
Nombres entiers		1,2
Nombres réels		
		"Hello world", "Python"
		`1+2j, 1.2 + 4j`

Nous découvrirons d'autres types plus tard.

Ces mots-clef permettent aussi de convertir un objet d'un type dans un certain autre type.

EXERCICE

Quels sont le type et la valeur de `int(2.2)` ? et de `int(1+2j)` ?
Exécuter et commeneter les commandes suivantes :

```
print("Nous sommes en "+2018)
print("Nous sommes en "+float(2018))
print("Nous sommes en "+str(2018))
```

Il est donc impossible de

EXERCICE

Essayer les opérations suivantes avec différents types de nombres, et compléter le tableau :

Python	Mathématiques
+	
*	
-	
/	
**	
//	
%	

En cas de mélange de type, Python se permet de modifier le type de certains objets. Si besoin, il pourra transformer des `int` en `float` ou `complex`, ou des `float` en `complex`, *etc.*

De même, les priorités opératoires sont les mêmes qu'en mathématiques. Il est cependant conseillé de parentheser autant que possible pour améliorer la lisibilité du code.

Il existe un dernier type qu'on va voir pour l'instant : le type des booléens `bool`. C'est un type qui contient seulement deux éléments : `True` et `False`. Ils servent à donner le résultat de certaines opérations.

Relation	Python
égal	==
différent	!=
inférieur strict	<
supérieur strict	>
inférieur ou égal	<=
supérieur ou égal	>=

Il existe aussi trois opérations sur les booléens : `and`, `or` et `not`.

EXERCICE

Compléter le résultat des commandes suivantes :

and	True	False
True		
False		

or	True	False
True		
False		

not	
True	
False	

5 Variables

Pour pouvoir sauvegarder et réutiliser une valeur, on utilise des variables. Une variable est la donnée de

- son nom, qui doit commencer par une lettre, puis peut contenir des lettres, des chiffres ou _ (attention, certains noms sont réservés à Python)
- sa valeur, qu'on donne par la commande *`nom = valeur`*
- son type, qui est automatiquement inféré par Python.

Attention à ne pas confondre l'affectation d'une valeur à une variable (=) et la comparaison de booléens (==).

Si on rentre par exemple la commande `a=42`, il y a deux cas possibles :

- si aucune variable ne s'appelait `a`, alors une nouvelle variable est créee portant ce nom, et ayant pour valeur 42

- si une variable s'appelant `a` avait déjà été définie, alors sa valeur et son type sont effacés et remplacés par 42 et `int`.

EXERCICE

Sans taper ces commandes, donner le type et la valeur de `a,b,c` à la fin du programme :

```
a = 42
b = a
c = "Hello world!"
b=a+b
c=a
b=c+b
b=b+1
```

EXERCICE

À l'aide d'une seule variable, calculer u_4 pour la suite définie par $u_0 = 1$, $u_{n+1} = 2u_n + 1$.
On définit la suite de Fibonacci par $f_0 = 0$, $f_1 = 1$ et pour tout n, $f_{n+2} = f_{n+1} + f_n$. À l'aide de trois variables, calculer la valeur de f_5.

Dans l'exercice précédent, on voit qu'il faut rajouter une variable intermédiaire. Python a une fonctionnalité puissante qui permet de l'éviter : l'affectation multiple.

Par exemple, la commande `a,b=1,2` permet d'affecter les deux variables en même temps. L'affectation de toutes les variables se fait avec leurs versions avant la commande.

EXERCICE

Comparer la suite de commandes

```
a=1
b=2
a=b
b=a
```

et

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

Réécrire alors le calcul de la suite de Fibonacci avec l'affectation multiple, avec seulement deux variables.

On peut aussi vouloir affecter à une variable un nombre choisi par l'utilisateur du programme. On utilise la commande `input`, qui s'utilise de la manière suivante :

```
nom_variable = type_voulu(input("phrase_a_afficher"))
```

À l'inverse, pour afficher quelques chose à l'écran, on utilisera la commande `print`. On peut affichier plusieurs choses à la suite en séparant les arguments par des virgules.

EXEMPLE

Ce programme calcule l'année de naissance en demande l'âge :

```
x=int(input("Quel âge aurez-vous le 31 décembre ?"))
x=2018-x
print("Vous êtes né en ",x)
```

EXERCICE

Écrire un programme qui demande à l'utilisateur le nombre de chiens, fourmis et poules dans une ferme, et affiche le nombre de pattes.

6 Modules

Pour l'instant, Python a très peu de fonctions accessibles. On peut en importer de nouvelles en utilisant des modules. Pour importer des fonctions d'un module, on a plusieurs choix

- `from` *`nom_module`* `import *` : importe toutes les fonctions du module, qui seront accessibles directement
- `from` *`nom_module`* `import` *`nom_fonction`* : importe seulement la fonction *nom_fonction* du module
- `import` *`nom_module`* : importe le module, dont les fonctions seront accessibles par *`nom_module.nom_fonction`*
- `import` *`nom_module`* `as` *`nom_choisi`* : importe le module, dont les fonctions seront accessibles par *`nom_choisi.nom_fonction`*

On utilisera beaucoup le module `math`, qui permet d'accéder à des fonctions mathématiques, comme `pi,e,sin,cos,tan,exp,log,sqrt,floor,ceil`*,etc.*

On utilisera aussi en probabilités le module `random`, qui génère des nombres aléatoires.

La commande `random()` renvoie un `float` aléatoire entre 0 et 1, et la commande `randint(`*`début`*`,`*`fin`*`)` renvoie un entier entre *début* et *fin* (inclus).

EXERCICE

Écrire un programme qui simule la commande `randint(1,6)` en utilisant la commande `random()`.