Cours – Les bases de la programmation

Objectifs :

- ⇨ Découvrir les concepts de base des langages de programmation
- ⇨ Apprendre les principaux éléments de syntaxe de python

I – Langage et code source

1) Nécessité d'un langage

Un ordinateur est une machine très polyvalente qui, comme le disent beaucoup d'informaticiens en plaisantant, peut faire à peu près tout sauf le café.

Mais un ordinateur est totalement dénué d'imagination, d'intuition ou de capacité d'apprentissage naturelle : il ne sait qu'exécuter une série d'ordres prévus à l'avance : son programme.
Il faut donc être capable de lui indiquer ce qu'il faut faire et comment le réaliser, ce qui implique un moyen de communication, c'est-à-dire un **langage**.

> *Les ordinateurs sont inutiles. Ils ne savent donner que des réponses.*
> – Pablo Picasso

2) Types de langages

Les ordinateurs actuels fonctionnent tous en logique binaire, c'est-à-dire qu'ils ne sont capable d'appréhender les données que sous forme de 0 et de 1.
Cela vient du fait qu'ils sont constitués de circuits électroniques où les tensions électriques ne peuvent prendre que deux états : 0 V ou +Vcc (où Vcc est la tension d'alimentation du circuit électronique).
Toutes les données que doit traiter l'ordinateur (instructions, nombres, sons, images, textes, …) doivent donc être convertis en binaire : c'est le codage ou la numérisation.

Ainsi c'est sous forme binaire que sont codées les instructions. Chaque microprocesseur connait ainsi un nombre réduit de commandes très élémentaires (son « jeu d'instruction ») qui mises bout à bout vont permettre d'effectuer des tâches plus élaborées.
Dans ces conditions, les langues utilisées par les humains ne sont pas directement compréhensibles par un ordinateur ; de même, des phrases complexes exprimées sous forme d'une suite de 0 et de 1 ne sont pas intelligible pour un être humain.

Il a donc fallu trouver des compromis : ce sont les différents **langages de programmation**.

Il en existe beaucoup, certains langages sont dits de ..... car ils sont plus proches des langues humaines (BASIC, JAVA, C/C++, FORTRAN, PYTHON, …), d'autres sont dits de ..... car plus proches de la syntaxe des machines (LANGAGE MACHINE, ASSEMBLEUR, …).

Langage machine (opcode)		Assembleur	
Code binaire	Code hexadécimal	Mnémonique	Signification
`01101010 00010100`	`6A 14`	`PUSH #0x14`	Place la valeur hexadécimale 14 sur la pile (mémoire de travail du programme)
`01110101 00011101`	`75 1D`	`JNZ _suite`	Saute à l'instruction située à l'adresse _suite (29 octets plus loin) si le résultat du dernier calcul n'est pas zéro.

Exemple d'instructions pour microprocesseur Intel 8086

Dans tous les cas, il y aura une phase intermédiaire que l'on pourrait qualifier de « traduction » qui convertira les instructions exprimées par le programmeur en opérations directement exécutables par l'ordinateur.
Selon la nature de cette phase intermédiaire on distingue à nouveau deux catégories de langages :

les **langages** ..... (Pascal, C/C++, Delphi, Fortran …) et les **langages** ..... (Logo, Basic, Python, Javascript…).

Dans le cas des langages interprétés, le texte du programme est exécuté par un logiciel déjà installé sur la machine (l'**interpréteur**). Par exemple, dans le cas du HTML[1], le texte est converti au moment de l'exécution en page internet par un navigateur (Firefox, Internet Explorer, Google Chrome, …). Les avantages essentiels de ces langages sont la portabilité (le même code peut s'exécuter sur plusieurs architectures machine différentes pourvu qu'un interpréteur soit disponible) et la simplicité de développement (pas de phase de compilation, débogage plus simple).

Dans le cas des langages compilés, le texte du programme est converti à l'avance en un fichier d'instructions exécutables par la machine au moyen d'un logiciel appelé **compilateur**. Ce type de langage a trois caractéristiques principales :

- la phase de compilation permet de détecter certaines erreurs : tant que le programme comporte certains types d'erreurs, il est impossible de le convertir en instructions exécutables.
- Si la compilation réussi, le programme sera exécuté plus rapidement que le programme équivalent écrit avec un langage interprété (certaines phases d'analyse du code et même d'optimisation ayant été faites au moment de la compilation).
- Le fichier généré par le compilateur est directement compréhensible par la machine (pas besoin de composant logiciel supplémentaire) et incompréhensible par l'homme. Ceci permet de protéger la confidentialité du programme dont le fonctionnement interne ne sera pas compréhensible.

Dans tous les cas (sauf pour le langage machine), le programme est donc écrit dans une langue humaine (l'anglais) sous forme de texte sans mise en forme particulière.

C'est ce que l'on appelle le ..... ou fichier source ou plus simplement *le* « source ».

On désigne également le programme comme le « code » et le fait de programmer comme le « codage ».

Le source étant un fichier texte sans mise en forme, il pourra être affiché et modifié avec n'importe quel éditeur de texte (comme le bloc-notes de Windows).
Cependant les programmeurs utilisent en général un EDI (voir TP) bien plus confortable à l'usage et qui contient des commandes spécifiques facilitant la saisie ou la vérification du programme.

II – Principe généraux

1) Syntaxe

Syntaxe : façon dont les mots se combinent pour former des phrases ou des énoncés dans une langue.

Casse : fait de distinguer les lettres CAPITALES des lettres minuscules.

La *sensibilité à la casse* (traduction de l'anglais *case sensitivity*) est une notion informatique signifiant que dans un contexte donné, le sens d'une chaîne de caractères (par exemple un mot, un nom de variable, un nom de fichier, un code alphanumérique, …) dépend de la casse (capitale ou bas de casse) des lettres qu'elle contient.

[1] Le HTML est un langage de description de page et non de programmation comme nous le verrons en détail plus tard dans l'année.

Indentation : ajout de tabulations ou d'espaces dans un fichier texte, généralement en début de ligne. En informatique cela permet d'améliorer grandement la lisibilité du programme. Dans certain langages, elle est même obligatoire pour définir les blocs de code.

La syntaxe des langages de programmation est très stricte. La moindre modification dans la casse, la ponctuation, ou l'indentation peut changer complétement le fonctionnement d'un programme.

`i=0`
`While i<10 :`
`  Print(i)`
`  i=i+1`
`Print(i)`
	`i=0`
	`while i<10 :`
	`  print (i)`
	`  i=i+1`
	`print( i )`
		`i = 0`
		`while i < 10:`
		`  print(i)`
		`  i = i + 1`
		`print(i)`
			`i = 0`
			`while i < 10:`
			`  print(i)`
			`i = i + 1`
			`print(i)`

⇨

2) Traitement séquentiel

Un processeur d'ordinateur exécute les instructions les unes après les autres, ne passant à la suivante que lorsque la précédente est terminée. On dit qu'il traite les instructions **séquentiellement**. Il en est de même dans les langages de haut niveau où les instructions s'exécutent à la suite, de haut en bas, une ligne après l'autre. Certaines instructions des langages permettent cependant de changer la prochaine ligne à être exécutée.

3) Familles d'instructions

Il existe 4 grandes familles d'instruction dans un code source :

- ✓ Les .....
  Elles correspondent à la mise en place de structures de données par le langage (création d'une variable, d'une fonction, importations de fonctions d'une bibliothèque, …)
  Ex : `import`

- ✓ Les .....
  Ils permettent de calculer un résultat à partir d'une ou plusieurs données (généralement deux).
  Ex : `+` (addition), `**` (mise à la puissance), `not` (négation)

- ✓ Les .....
  Lorsque le programme rencontre une fonction, il suspend son traitement pour aller exécuter un autre bout de code (la fonction) et en retirer un résultat.
  Ex : `math.cos(x)` (calcule le cosinus de x), `turtle.reset()` (initialise l'affichage pour turtle)

Les opérateurs et les fonctions permettent de former ce que l'on appelle des ..... . L'essentiel du temps d'un programme se passe donc à ....., c'est-à-dire calculer leur valeur.

Parfois l'évaluation d'une expression ne donne pas lieu à une valeur (mais l'évaluation a pu avoir l'effet recherché comme par exemple afficher un texte). Dans ce cas l'évaluation donne la valeur spéciale « `None` ».

- ✓ Les .....
  Ces instructions permettent de modifier le déroulement séquentiel des instructions en changeant l'endroit du code où l'exécution du programme va se continuer.
  Ex : `for` (répétition d'une portion de code), `if` (passe une portion de code si une condition n'est pas satisfaite)

III – Le langage Python

Le langage Python a été créé en 1991 par Guido van Rossum (un programmeur néerlandais). Il ne commence à se développer vraiment qu'à partir des années 2000 avec la version 2.0 et trouve un public très large à partir de la version 3 et des années 2010. Il a l'avantage d'être simple et possède énormément de bibliothèques de fonction ce qui en fait un langage très utilisé. Voir une présentation plus complète sur https://www.lebigdata.fr/python-langage-definition.

Fait important et rare en informatique : les développeurs de python ont décidé de *casser la compatibilité* entre les versions 2.x et 3.x de python. Ainsi un programme écrit pour python 2 ne fonctionnera pas correctement (sauf pour les cas les plus simples) avec l'interpréteur python 3.
La **version 3** de python est largement la plus utilisée et c'est celle que nous utiliserons.

1) Caractéristiques

- Python est un langage ..... . Il est permet aussi bien la **programmation impérative** (ce que l'on utilisera cette année) que la programmation objet ou fonctionnelle (dont nous aurons un aperçu l'an prochain).
- C'est un langage à typage ..... . Ce qui veut dire que le type des données utilisé est toujours bien défini et que ce type se détermine au moment de la création de la structure de donnée.
- Il incorpore des mécanismes de **gestion automatique de la mémoire**.
- Il est **multiplateforme** (fonctionne sur tout type de système d'exploitation et d'architecture matérielle).
- C'est un langage ..... , ce qui signifie qu'il est gratuit, librement distribuable et que son code source est public (« open source »).

2) Syntaxe

Python est un langage sensible à la casse. Il convient donc d'être très prudent dans l'utilisation des majuscules et des minuscules et de se fixer des règles strictes en ce sens.

a. Lignes de programme

Chaque ligne du fichier source de python correspond à une instruction. Dans le cas d'instruction très longue à écrire, il est possible de la fractionner sur plusieurs lignes :

- soit à l'aide du caractère de continuation de ligne « `\` »,
- soit de manière implicite si on passe à la ligne après une parenthèse, un crochet ou une accolade ouvrante.

`a = math.exp(12) \`
`  + 3*x \`
`  - 5`
	`turtle.reset()`
	`turtle.color(determination_couleur_stylo(),`
	`  determination_couleur_arriereplan() )`

Continuation de ligne explicite avec « `\` »	Continuation de ligne implicite

Pour aérer la présentation, il est possible de rajouter des espaces entre les opérateurs, avant ou après la ponctuation, mais ..... (avant le premier caractère non blanc) sauf dans le cas où la ligne est fractionnée.

On peut également rajouter des lignes vides pour aérer la présentation.

b. Commentaires

Pour faciliter la compréhension des codes sources et les documenter, il est possible d'insérer des **commentaires** dans le code source. Ceux-ci sont uniquement à destination des lecteurs humains et sont totalement ignorés par l'interpréteur python.

Il existe deux types de commentaires en python :

- Les commentaires monolignes qui commencent par le caractère « # » : tout le reste de la ligne jusqu'au passage à la ligne suivante est ignoré par python.
- Les commentaires multilignes qui commencent et finissent par des triples guillemets anglais : « """ » et peuvent s'étendre sur plusieurs lignes.

```
print(resultat)   # Affichage de la solution

""" Ici on peut décrire ce que le programme
    va faire par la suite ou documenter une
    structure de donnée, le rôle d'une
    variable, … """

resultat = nouveau_traitement(a, b)
a = 3
# a = b
print(a)
```

Les commentaires permettent également d'éviter qu'une ligne s'exécute.

⇨

c. Indentation

L'indentation est capitale en python car elle permet de repérer les différents blocs de programme : toutes les lignes ayant la même indentation (même nombre d'espaces entre le début de la ligne et le premier caractère non blanc) situées entre 2 lignes d'indentations différentes forment un bloc.

```
a = 1
if a < b :
    c = a + b
    print(c)
d = 8 + b
```

Bloc de code

d. Noms

Les noms d'objets (variables, fonctions) dans python suivent les règles suivantes :

- Ils ne doivent être constitués que de lettres (`A-Z` et `a-z`), de chiffres (`0-9`) et du caractère de soulignement (« _ »). Pas d'espace.
- Ils ne doivent pas commencer par un chiffre (`variable_1` est correct, mais `1_variable` ne l'est pas).
- Ils sont sensibles à la casse (`Mavariable`, `mavariable` et `MAVARIABLE` sont traités comme trois variables différentes).
- Ils ne peuvent pas correspondre à un des mots-clés du langage :

Liste des mots-clés de python :

and	**assert**	**break**	**class**	**continue**	**def**	**del**	**elif**	**else**
except	**exec**	**finally**	**for**	**from**	**global**	**if**	**import**	**in**
is	**lambda**	**not**	**or**	**pass**	**print**	**raise**	**return**	**try**
while	**yield**	**True**	**False**	**None**				

- Ils peuvent avoir n'importe quelle longueur

3) Utilisation de bibliothèques externes

Python permet d'utiliser des fonctions qui ont été écrites ailleurs que dans le fichier source sur lequel on travaille. Il y a trois cas de figure possible :

- Il s'agit d'une fonction « ..... », c'est à dire intégrée directement à la base du langage et dont le code réside dans l'interpréteur lui-même. Ex : `print()`, `input()`, `dir()`, `help()`, …
- Les fonctions de bibliothèques tierces. Elles peuvent être écrites en python ou dans d'autres langages et résident dans les répertoires de l'installation de python.
- Les fonctions écrites dans d'autres fichiers source (.py) et appartenant au même projet.

Lorsque python essaye d'importer un module « toto », il commence par chercher si le fichier « toto.py » existe dans le même répertoire que le fichier source actuel. Si oui, il va chercher le fichier en question et l'exécute. Si non, il va chercher dans les bibliothèques standard de python. S'il ne trouve toujours pas, il affiche une erreur « `ModuleNotFoundError` ».

Application :
Créer dans le même répertoire 2 fichiers « test1.py » et « test2.py » :

test1.py	test2.py

`import test2`
`print("Début du programme")`
`print(a)`
	`print("Exécution du code de test2.py")`
	`a = 5`
	`print("a =", a)`

Exécuter le fichier test1.py et commenter. Utiliser l'exécution pas à pas (Step into – F7) pour essayer de comprendre ce qui se passe avec la variable a.

NB : Généralement les fichiers source que l'on importe ainsi ne contiennent que des déclarations de fonction et pas de code exécutable directement comme nous venons de le faire.

Il existe plusieurs syntaxes pour l'instruction import :

`import turtle` : syntaxe de base permettant d'importer toutes les fonctions de la bibliothèque turtle en les faisant précéder du nom du module et d'un point (ex : `turtle.forward(100)`).

`import turtle as tt` : permet de faire référence au module turtle non pas avec son nom complet, mais avec le nom abrégé qu'on écrit après le mot-clé as. Ainsi on pourra écrire `tt.forward(100)` plutôt que `turtle.forward(100)`.

`from turtle import *` : permet de faire référence à toutes les fonctions du module turtle directement sans avoir à préciser le module. Ainsi on pourra écrire `forward(100)` plutôt que `turtle.forward(100)`.

`from turtle import reset, forward, left, right, color` :
permet d'importer uniquement certaines fonctions du module turtle qui seront alors accessibles directement sans avoir à écrire « `turtle.` » avant le nom de fonction.

Application :
Créer un nouveau programme « turtle1.py » contenant juste les lignes ci-contre.
Créer ensuite « turtle2.py » et « turtle3.py » avec les deux dernières syntaxes d'`import`.
Exécuter les trois versions et noter les différences (regarder notamment la fenêtre « Variables » de Thonny)

```
import turtle as tt

tt.reset()
tt.forward(50)
```

4) Deux fonctions de base en python

Nous allons voir deux fonctions très utiles dans tous les programmes de base en python et qui concernent les entrées-sorties.

a. Affichage dans la console : la fonction `print()`

Cette fonction built-in prend un nombre d'argument quelconque et affiche des caractères dans la fenêtre de sortie standard (le Shell dans Thonny).

```
print(valeur1, valeur2,  ..., sep=' ', end='\n', file=sys.stdout)
```

Cette fonction affiche les textes correspondant aux valeurs `valeur1`, `valeur2`, … en les séparant par un séparateur (`sep`) qui peut être précisé mais qui est l'espace par défaut) et en rajoutant un dernier caractère (`end`) à la fin (par défaut c'est un saut de ligne `'\n'`).

Exemple :

Programme	Affichage

`a = 13`
`print("La valeur de a est :", a, end="")`
`print("en rajoutant 2, on obtient", a+2)`
`print()    # Saute une ligne (ligne vide)`
`print(1, 2, 3, 4, sep="-")`
	`La valeur de a est : 13en rajoutant 2, on obtient 15`
	
	`1-2-3-4`

NB : Cette fonction ne renvoie rien (ou plutôt elle renvoie la valeur « `None` »).

b. Interaction avec l'utilisateur : la fonction `input()`

On a souvent besoin d'interaction avec l'utilisateur dans un programme. La fonction built-in `input` permet de demander à l'utilisateur de saisir quelque chose au clavier.

```
texte_saisi = input(prompt)
```

Le paramètre prompt est facultatif. S'il est donné alors la fonction input affiche ce texte dans la fenêtre de sortie standard avant d'afficher le curseur et d'attendre la saisie de l'utilisateur. Lorsque l'utilisateur appuie sur la touche entrée, la variable `texte_saisie` reçoit l'ensemble des caractères tapés avant l'appui sur entrée.

La variable `texte_saisie` est donc une variable de type chaine de caractère et il peut être nécessaire de la convertir en nombre avec les fonctions `int()` (pour un entier) ou `float()` (pour un nombre à virgule).

Exemple :

Programme	Affichage

`saisie = input("Nombre de départ ?")`
`n = int(saisie)`
`print("Si on ajoute 1, cela fait", n + 1)`
`input("Appuyez sur <Entrée> pour quitter")`
	`Nombre de départ ?5`
	`Si on ajoute 1, cela fait 6`
	`Appuyez sur <Entrée> pour quitter`