# Sujet n°1

## Algorithmique & Programmation

### Découverte de l'IDE

L'objectif principal de ce cours est l'apprentissage de l'algorithmique. Afin de mettre en œuvre les différents algorithmes que vous allez créer, nous utiliserons le langage *Python* comme support d'implémentation. L'apprentissage de ce langage n'est donc pas un objectif en soi pour ce cours.

## 1 Environnement de développement

*IDLE* est un environnement de développement (Integrated Development Environment - IDE) pour Python, fourni avec toutes les versions de Python. Nous l'utiliserons pour sa simplicité d'installation et d'utilisation. Il vous permettra de constituer rapidement votre environnement de développement sur vos machines personnelles.

L'utilisation d'*IDLE* est très simple, et l'interface se divise en plusieurs fenêtres :

— Un *shell* permettant d'exécuter des instructions Python ligne à ligne, de visualiser la sortie standard d'un programme, et de lancer le débogueur

— Un éditeur de texte dans lequel vous pouvez entrer votre programme (avec colorisation et indentation automatique), l'enregistrer et l'exécuter

— Éventuellement une fenêtre de débogage

— Éventuellement une fenêtre graphique (si on utilise le module "turtle").

### 1.1 L'interpréteur Python

Pour démarrer *IDLE*, il vous suffit de taper la commande `idle3` sous Linux ou aller dans le menu *Démarrer/Tous les programmes/Python xxx* et de lancer *IDLE(Python GUI)* sous Windows.

Au lancement d'*IDLE*, seule la fenêtre *Shell* apparaît. Vous pouvez taper des commandes interactives.

Vérifiez que la version de python utilisée est 3.X où X est un entier.

Cette fenêtre contient une ligne de commande symbolisée par

```
>>>
```

Dans cette ligne de commande, vous pouvez écrire un ordre. Puis envoyer cette ordre à l'ordinateur en tapant sur la touche entrée. Par exemple, tapez la commande suivante :

```
>>> 1 + 1
```

Validez l'ordre avec la touche entrée et observez le résultat que vous obtenez. Essayez maintenant d'exécuter les ordres suivants :

```
>>> print("Bonjour !")
>>> 23.2 * (2 + 6.3)
```

### Exercice 1

QUESTION 1. Prévoyez le résultat que l'ordinateur va donner en exécutant les ordres suivants :

```
>>> 1 > 2
>>> 3.0 / 2
>>> 3 / 2
>>> 3 // 2
>>> 3 % 2
>>> 3 / 2.0
>>> 3.0 / 2.0
>>> i = 2
>>> i = i + 4
>>> print( i )
>>> j = 5
>>> i > j
>>> i != 9
>>> i == 9
>>> True and False
>>> True or False
>>> print( non_defini )
>>> True and non_defini
>>> False and non_defini
>>> non_defini and False
```

QUESTION 2. Faites exécuter par l'ordinateur ces commandes et observez le résultat.

QUESTION 3. Avez-vous obtenu le résultat souhaité ? Si non, essayez d'expliquer le résultat obtenu.

## 1.2 Variables et affectations

Pour manipuler des données, il peut être utile d'utiliser des variables qui représentent ces données dans la mémoire de l'ordinateur. Un nom de variable est une suite de caractères qui renvoie à une adresse mémoire où a été créé un objet.

Pour affecter (ou assigner) une valeur à une variable on utilise le signe =. La commande ci-dessous signifie que désormais la variable $x$ renvoie à la valeur 1.

```
>>> x=1
>>> x
1
```

Le signe = doit être distingué de l'égalité mathématique (== en python).

### Exercice 2

Examiner la série de commandes ci-dessous. Prédire le résultat puis confirmer-le à l'aide d'*IDLE*.

```
>>> x=1
>>> y=2
>>> x=x+y
>>> y=x**y
>>> y
```

## 1.3 Les fonctions d'entrée / sortie

Afin que l'utilisateur puisse entrer des données dans les variables du programme et visualiser les résultats calculés par le programme, le programmeur utilise des fonctions d'entrée et de sortie.

La fonction `print` est une fonction de sortie : elle affiche à l'écran (à l'attention de l'utilisateur) une donnée ou le contenu d'une variable.

La fonction `input` est une fonction d'entrée : le programme affiche une question à l'attention de l'utilisateur et attend que l'utilisateur tape sur des touches du clavier en réponse à cette question. L'utilisateur doit terminer sa saisie par la touche "*Entrée*" pour que le programme continue son exécution. La suite de caractères saisie par l'utilisateur est récupérée par le programme où elle peut être utilisée. Elle est généralement affectée à une variable du programme.

## 1.4 Écrire des programmes

Dans la section précédente, nous avons vu comment écrire des ordres et les envoyer à l'ordinateur pour qu'il les exécute.

Exécuter les ordres un à un est fastidieux. Il est plus pratique d'écrire un programme et d'exécuter ensuite le programme d'une seule traite, sans interruption. *IDLE* permet d'écrire des programmes dans des fichiers et de les faire exécuter par l'ordinateur. Pour cela, vous devez cliquer sur l'onglet *File/New Window* et une nouvelle fenêtre apparaitra.

Cette fenêtre contient une zone de texte vide dans laquelle vous pouvez écrire votre programme.

Écrivez le programme suivant :

```
print("Bonjour !")
print("Les entiers de 0 à 4 sont :")
for i in range(5):
    print(i)
```

Enregistrez votre programme dans le ficher `programme.py` en allant dans le menu *File/save*. Vous pouvez maintenant exécuter votre programme en cliquant sur l'onglet *Run/Run Module*. Observez et décrivez ce qu'il se passe.

## 1.5 Déboguage avec IDLE

Pour activer le débogueur, placez le curseur dans la fenêtre *shell*, puis sélectionnez le menu *Debug/Debugger*. La fenêtre du débogueur apparaît.

Ensuite, commencez l'exécution avec la touche *F5*. La fenêtre du débogueur affiche alors la première ligne du programme.

La barre de boutons du haut contient les actions :

- *Go* pour continuer l'exécution sans déboguer
- *Step* pour exécuter l'instruction suivante. Si l'instruction courante est un appel de fonction, le débogueur entre dans la fonction
- *Over* pour exécuter jusqu'à la ligne suivante du programme
- *Out* pour exécuter jusqu'à sortir de la fonction courante (très utile si on est entré par erreur dans une fonction)

Le panneau du milieu affiche ce qu'on appelle la *pile d'appel*, mais dans le cas d'un programme simple il y a seulement deux lignes :

- Une ligne 'bdb'.run() qu'on peut ignorer
- La ligne du programme à laquelle on est, et qui sera exécutée lors du prochain *Step* ou *Over*

Dans le panneau du bas se situe la liste des variables locales qui apparaissent au fur et à mesure de leur définition, ainsi que leurs valeurs.

### Exercice 3

Exécutez pas à pas le programme suivant jusqu'à sa fin, et observez les changements des valeurs $a$, $b$, $c$.

```
a = 0
b = 1
```

```
while b < 10:
  print b
  c = a + b
  a = b
  b = c
```

## 2 Premiers problèmes

### Exercice 4

QUESTION 1. Soit $f(x) = 2x^2 - x + 1$. Écrire un programme qui affiche le résultat de f(1), f(2) et f(3). Vous mettrez ce programme dans le fichier `polynome.py`.

QUESTION 2. Écrire une fonction `moyenne` qui prends en paramètres deux entiers et qui calcule la moyenne de ces deux entiers. Écrire un programme qui affiche la moyenne de 11 et 14, de 18 et 15, de 20 et 15 en utilisant la fonction `moyenne`. Vous mettrez ce programme dans le fichier `moyenne.py`.

QUESTION 3. Écrire une fonction `est_divisible_par` qui prends en paramètres deux entier $n$ et $k$ et qui renvoie *vrai* si $n$ est divisible par $k$, *faux* sinon.Écrire un programme qui affiche la divisibilité de 5 par 3, de 6 par 2 et de 9 par 3. Vous mettrez ce programme dans le fichier `arithmetique.py`.

QUESTION 4. Écrire une fonction `est_paire` qui prends en paramètre un entier et qui renvoie *vrai* si l'entier est pair, *faux* sinon. Vous utiliserez la question précédente pour réaliser cette fonction. Écrire un programme qui affiche la parité des entiers 2, 4, 3 et 7. Vous ajouterez ce programme au fichier `arithmetique.py`.

QUESTION 5. Écrire une fonction `est_compris_dans` qui prends en paramètres trois entiers, $a$, $b$ et $c$ et qui renvoie *vrai* si $a$ est compris entre $b$ et $c$.

## 3 Les modules

Lorsque l'on écrit des programmes, on souhaite pouvoir réutiliser du code déjà écrit. En python, cela est possible grâce aux modules. Un module est une bibliothèque contenant du code. Python contient de nombreux modules. Par exemple, il contient le module *math* qui permet de calculer des racines carrés, des cosinus et d'autres fonctions mathématiques. Pour réutiliser le code du module math, il vous suffit d'importer le module en tapant la commande :

```
>>> import math
```

Ensuite, pour calculer la racine carré, il vous suffit de taper :

```
>>> math.sqrt(3)
```

Si vous souhaitez obtenir de l'aide concernant ce module, vous devez taper :

```
>>> help(math)
```

Lorsque vous écrivez un programme dans un fichier, vous écrivez un module. Le nom du module est le nom du fichier sans son extension. Vous pouvez donc créer vous même vos propres modules et les importer de la même manière que vous avez importé la bibliothèque math. Cependant, le module à importer doit être situé dans le même répertoire que le programme qui importe le module.

Par exemple, avec *IDLE*, créez un nouveau module `util` en ajoutant un fichier `util.py` contenant le code suivant :

```
def somme(n):
    return n*(n+1)/2
```

Créez ensuite, le programme main.py contenant le code

```
import util

n=5
print("La somme des entiers allant de 0 à " + str(n) + " vaut:" )
print( util.somme(n) )
```

Le fichier `main.py` représente le programme que vous souhaitez exécuter. Ce programme utilise les fonctionnalités du module *util* codé dans le fichier `util.py`.

Vous pouvez maintenant exécuter le programme de main.py en utilisant l'onglet *Run/Run Module* de la fenêtre associée au fichier `main.py`.

## Exercice 5

QUESTION 1. Faire un module `affichage_divers` qui contient la fonction `afficher_somme` qui prends en paramètre un entier $n$ et qui affiche la somme de 0 à $n$.

QUESTION 2. Ajoutez à ce module, la fonction `afficher_moyenne` qui prends en paramètres deux entiers et qui affiche leurs moyennes.

QUESTION 3. Faire un programme qui utilise le module `affichage_divers` et qui affiche la somme des entier de 0 à 201, et qui affiche la moyenne entre 12.2 et 14.3.

Certaines fonctions de certains modules sont très fréquemment utilisées. Dans ce cas, il est plus sympathique de pouvoir utiliser directement le nom de la fonction sans avoir à rappeler le nom du module. C'est possible sous python, pour cela, il suffit d'importer la fonction en écrivant la ligne :

```
from MODULE import FUNCTION
```

Par exemple, si l'on reprend l'exemple de la racine carré, on obtient :

```
from math import sqrt
sqrt(3)
```

Si vous souhaitez importer toutes les fonctions d'un module donné, il vous suffit décrire :

```
from MODULE import *
```