TP 1 - Utilisation de Python

L'objectif de ce TP est d'apprendre à faire réaliser des calculs et des tâches répétitives à un ordinateur.

Pour faire cela, il est nécessaire de communiquer avec l'ordinateur en utilisant un langage compris par la machine. Il existe plusieurs langages que l'ordinateur est capable d'interpréter. Nous avons choisi le langage appelé "Python".

Dans la première partie de ce TP, nous expliquons comment il est possible d'écrire et d'envoyer des ordres écrit en Python à l'ordinateur.

Dans la seconde partie, nous montrons comment écrire et exécuter une succession de plusieurs ordres. En informatique, on appelle cette succession de plusieurs ordres un programme. On appelle cela aussi du code.

Dans une troisième partie, nous expliquons comment il est possible de réutiliser dans un nouveau programme du code existant en utilisant les modules.

Enfin, nous terminons par des exemples avancées d'utilisation de modules.

Donner des ordres à un ordinateur

Pour envoyer des ordres en Python à un ordinateur, nous utilisons le programme `idle`. Le programme `idle` permet à la fois d'écrire et de communiquer des ordres à la machine.

Lancez le programme `idle` :

— Sous Linux, il suffit d'ouvrir un terminal et de taper la commande :

```
idle
```

— Sous Windows, allez dans le menu `Démarrer/Tous les programmes/Python2.6` et cliquez sur `IDLE(Python GUI)`.

Si tout se passe bien, vous devriez obtenir la fenêtre de la figure 1.

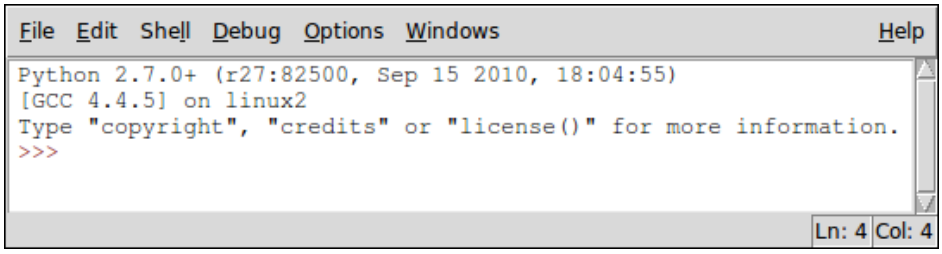

FIGURE 1 – Démarrage du programme idle

Vérifiez que la version de python utilisée est 2.X où X est un entier. Si ce n'est pas le cas, relancez IDLE avec la bonne version de python.

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

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

```
>>> 1 > 2
>>> 3.0 / 2
>>> 3 / 2
>>> 3 % 2
>>> 3 / 2.0
>>> 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
```

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

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

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`. Une nouvelle fenêtre devrait apparaître. Cette fenêtre doit ressembler à celle présentée à la figure 2.

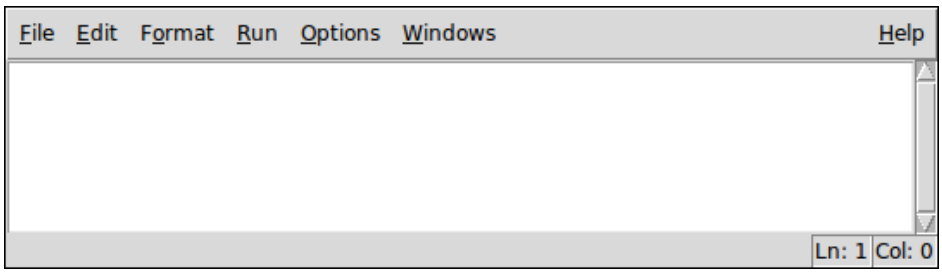

FIGURE 2 – Éditer un programme

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 `mon_premier_programme.py` en allant dans le menu `File/save`. Vous prendrez soin d'enregistrer votre programme dans le dossier `cpbx/python/tp1` que vous devez créer.

Vous pouvez maintenant exécuter votre programme en cliquant sur l'onglet `Run/Run Module`. Observez et décrivez ce qu'il se passe.

Exercice 2

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 `exercice2/calcul_polynome.py`.
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 `exercice2/calcul_moyenne.py`.
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 `exercice2/calcul_arithmetique.py`.
4) Écrire une fonction `est_paire` qui prends en paramètre un entier et qui renvoie vrai si l'entier est paire, 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 `exercice2/calcul_arithmetique.py`.
5) Écrire une fonction `est_compris_dans` qui prends en paramètres trois entiers, `a`, `b` et `c` et qui renvoie vraie si `a` est compris entre `b` et `c`.

6) (Optionnel, difficile : il faut être inventif !) Écrire une fonction `max` qui prends en paramètres deux entiers et qui renvoie l'entier le plus grand. Vous devez écrire cette fonction SANS utiliser les sauts conditionnels (if then else), NI les boucles (while, for).

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. Si vous souhaitez placer le module dans un répertoire différent, nous vous invitons à lire la section "Utilisation avancée des modules".

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 3

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`.

2) Ajoutez à ce module, la fonction `afficher_moyenne` qui prends en paramètres deux entiers et qui affiche leurs moyennes.
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 fréquement utilisées. Dans ce cas, il est plus sympatique 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 *
```

Exercice 4

Reprendre l'exercice 3 et importer dans le programme les fonctions à la place du module.

Utilisation avancée des modules

Supposons que l'on exécute avec `idle` un programme situé dans le fichier `prog.py` lui même situé dans le répertoire `cpbx/python/tp1`. Dans la section précédente, nous avons vu que l'ordinateur était capable d'importer n'importe quel module situé dans le répertoire `cpbx/python/tp1`. C'est en effet possible, car `idle` a informé la machine que le répertoire où se trouve le programme à exécuter est susceptible de contenir des modules. Pour connaître la liste de tous les répertoires dans lequel l'ordinateur doit chercher des modules, il suffit d'importer le module `sys` et d'afficher la liste `sys.path` :

```
>>> import sys
>>> sys.path
```

Sous Linux, on obtient la réponse suivante :

```
['/home/schur/cpbx/python/tp1', '/usr/bin', '/usr/lib/python2.7',
 '/usr/lib/python2.7/plat-linux2', '/usr/lib/python2.7/lib-tk',
 '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
 '/usr/local/lib/python2.7/dist-packages',
 '/usr/lib/python2.7/dist-packages']
```

Comme prévu, on trouve le dossier courant `cpbx/python/tp1`. On remarque qu'il y a aussi d'autres dossiers. Ces dossiers contiennent diverses bibliothèques.

Si vous voulez maintenant pouvoir ajouter des modules situés dans de nouveaux répertoires, il vous suffit d'inclure ces nouveaux répertoires dans la liste `sys.path`. Par exemple, imaginons que vous ayez un module graphique situé dans le dossier `/home/schur/mes_programmes_python`, alors, pour pouvoir accéder à ce dossier, il vous suffit de taper la commande :

```
>>> import sys
>>> sys.path.append('/home/schur/mes_programmes_python')
```

Il existe une autre manière de pouvoir importer un module sans avoir à ajouter des dossiers dans `sys.path`. Pour cela, il faut que :

— le dossier en question soit un sous-dossier d'un des dossiers de `sys.path` ;
— le dossier contienne un fichier `__init__.py` qui peut être vide ;
— que la commande qui importe le module soit de la forme :

```
import SOUS–DOSSIER.SOUS–DOSSIER. ... .SOUS–DOSSIER.MODULE
```

Pour illustrer cette dernière solution, nous allons créer un programme qui affiche une approximation de Pi en écrivant un module `constantes.py` qui calcule Pi et que l'on mettra dans le sous-dossier `geometrie`.

Commencez par créer un nouveau dossier `geometrie` dans `cpbx/python/tp1`. Ajoutez un fichier `__init__.py` vide dans ce dossier. Ajoutez le module `constantes.py` contenant le code suivant :

```
def pi(n):
    result = 0.0
    for i in xrange(n):
        result = result + 2.0/((4*i+1)*(4*i+3))
    return 4 * result
```

Créez le programme `afficher_approximation_pi.py` dans le dossier `cpbx/python/tp1` avec le code suivant :

```
import geometrie.constantes
print("Voici une approximation de la constante pi:")
print( geometrie.constantes.pi(100) )
```

Normalement, l'arborescence de votre TP devrait ressembler à :

```
cpbx/pyton/tp1
|-- afficher_approximation_pi.py
`-- geometrie
    |-- constantes.py
    `-- __init__.py
```

Exécutez le programme `afficher_approximation_pi.py` pour afficher une approximation de Pi.

Dans un programme, on n'utilise jamais la première solution car le contenu de la variable 'sys.path' dépend du système d'exploitation et de votre environnement de travail. Si vous utilisez la première solution, vous obtiendrez un programme qui ne marche que sur votre ordinateur. C'est pour cela que l'on utilise toujours la deuxième solution qui elle ne dépend pas du système d'exploitation, ni de votre environnement de travail.