Informatique
TP1 : Découverte de Python
CPP 1A

Frederic Devernay, Matthieu Moy, Djamel Aouane

Mars - avril 2015

1 Découverte de l'IDE : Spyder

Spyder est un environnement de développement (Integrated Development Environment) pour Python, fourni avec toutes les versions de python.

L'utilisation de Spyder est très simple, et l'interface comporte par défaut une fenêtre avec ces panneaux :

- Sur la gauche, un éditeur de texte, dans lequel vous pouvez entrer votre programme (avec colorisation et indentation automatique), l'enregistrer et l'exécuter ;
- En bas à droite, un shell (ou interprète interactif) permettant de visualiser l'exécution du programme, et de lancer le débogueur ;
- En haut à droite, une fenêtre permettant d'afficher des informations sur le programme ;
- Éventuellement une fenêtre graphique (si on utilise le module "turtle").

Quelques présentations sur Spyder sont disponibles sur le site du cours : `http://www-verimag.imag.fr/~moy/cours/infocpp-S2/`.

1.1 L'interprète Python

Au lancement de Spyder, le shell apparaît en bas à droite de la fenêtre. Dans un premier temps, nous ignorerons tout le reste. Vous pouvez taper des commandes interactives.

Exercice 1 (Evaluation de quelques expressions) *Évaluez les expressions suivantes dans l'interprète :*

- `2 + 2`
- `2 - 2`
- `-2`
- `2 + 3 * 4`
- `(2 + 3) * 4`
- `2 ** 3`
- `2 ** 4`
- `10 / 3`
- `10 // 3`
- `10 % 3`

Que font les opérateurs `**` *et* `%` *? Quelle est la différence entre* `/` *et* `//` *?*

Exercice 2 (Utilisation de variables) *Évaluez maintenant les expressions suivantes (dans cet ordre) :*

- `x = 42`
- `y = x + 1`
- `x`
- `y`

1.2 Écriture d'un programme avec l'éditeur de texte, exécution

Au lancement, l'éditeur de texte (partie gauche de la fenêtre) est ouvert sur un fichier Python vide. Il est fortement déconseillé d'utiliser ce fichier ouvert par défaut : il se trouve caché dans un répertoire spécifique à Spyder, vous risquez de ne pas le retrouver si vous y mettez des choses importantes.

La première action à faire après le lancement de Spyder est donc de créer un nouveau fichier (menu « Fichier », « Nouveau Fichier ») et de l'enregistrer pour lui donner un nom.

Il y a également la possibilité d'ouvrir un fichier .py directement avec Spyder depuis l'explorateur (Windows) ou le Finder (Mac OS X).

Exercice 3 (Premier programme) *Entrez dans l'éditeur le programme suivant (vu dans le premier cours) :*

```
a = 0
b = 1
while b < 10:
    print(b)
    c = a + b
    a = b
    b = c
```

Pour l'exécuter, tapez la touche F5 (en haut du clavier). Spyder vous proposera d'enregistrer le fichier, faites-le et si ce n'est pas déjà fait, donnez-lui un nom explicite, par exemple ***`fibonacci.py`****.*

Vous devriez voir apparaître dans le panneau « shell » (en bas à droite de la fenêtre Spyder) :

```
>>> runfile('fibonacci.py', wdir=r'/tmp')
1
1
2
3
5
8
>>>
```

Essayez d'ajouter une erreur dans ce programme, par exemple remplacer `print(b)` *par* `print(b` *(suppression de la parenthèse fermante).*

Enregistrez le fichier. Spyder vous signale une erreur avec un symbole ⚠ dans la marge à côté du `print`*.*

On peut tout de même tenter une exécution : Spyder vous signale alors une erreur de syntaxe dans le panneau « shell ». En cliquant sur le message d'erreur, le curseur se positionne à la bonne ligne du fichier concerné. Corrigez l'erreur (la parenthèses fermante manquante du `print`*) et exécutez le programme.*

2 Rappels et compléments sur les bases de Python

2.1 Affichage : print

Exercice 4 (Entrées sorties avec `print` et `input`) *Créez un nouveau fichier dans l'éditeur de texte de Spyder (menu « Fichier », « Nouveau Fichier »), et entrez le programme suivant :*

```
print("Bonjour, quelle est ton année de naissance ?")
annee = int(input())
print("Tu as", 2015 - annee, "ans.")
```

Exécutez ce programme. Essayez avec une vraie année de naissance. Essayez avec autre chose qu'un nombre entier. Que se passe-t-il ?

Exercice 5 (Importance de `print`) *Essayez d'entrer les expressions ou instructions suivantes dans l'interprète interactif :*

- `"Bonjour"`
- `print("Bonjour")`
- `2 + 2`
- `print(2 + 2)`

Essayez maintenant d'entrer ces 4 lignes dans un programme, dans la fenêtre de l'éditeur de texte. Exécutez ce programme. Que voyez-vous ? Pourquoi ?

2.2 Les commentaires

Un commentaire est une portion du programme qui est ignorée par Python. Les commentaires sont là pour aider la relecture du programme par un être humain.

En Python, un commentaire commence par le caractère `#` et va jusqu'à la fin de la ligne.

Exercice 6 (Commentaires) *Modifiez le programme précédent pour en faire, par exemple :*

```
# Cette ligne n'affiche rien :
"Bonjour"

print("Bonjour")  # Encore un commentaire
2 + 2
print(2 + 2)

# Commentaire
# sur plusieurs
# lignes.
```

Vérifiez en exécutant le programme que son comportement n'a pas changé.

2.3 Les erreurs

Les programmeurs sont des êtres humains comme les autres, et ils ont droit à l'erreur ! Plusieurs types d'erreurs peuvent se produire :

- Les **erreurs de syntaxe**, qui correspondent à des programmes qui ne sont pas "grammaticalement corrects". Dans ces cas, Python refuse de commencer l'exécution du programme (il ne comprend pas ce qu'on lui demande).
- Les **erreurs à l'exécution** peuvent arriver après le début de l'exécution du programme (Python comprend ce qu'on lui demande, mais il y a un problème lors de l'exécution du programme).

Exercice 7 (Erreur de syntaxe) *Essayez d'entrer le programme suivant (dans l'interprète interactif ou dans l'éditeur de texte) :*

```
print("Début du programme")
x = 2 +
print("Fin du programme")
```

Essayez d'exécuter le programme. Que se passe-t-il ?

Exercice 8 *Modifiez le programme ci-dessus comme ceci :*

```
print("Début du programme")
x = 1 / 0
print("Fin du programme")
```

Essayez d'exécuter le programme. Que se passe-t-il ?

En Python, les vérifications d'existence et de types des variables sont faites à l'exécution. Les erreurs suivantes sont donc des erreurs à l'exécution :

```
x = 'Bonjour' + 42  # Can't convert 'int' object to str implicitly
x = y + z  # name 'y' is not defined
```

(D'autres langages comme C ou Java font ces deux vérifications au même moment que les vérification de syntaxe)

3 Applications

3.1 Calcul simples sur un circuit électrique

On considère le circuit de la figure 1. L'objectif de cet exercice est de calculer le courant I_2 en fonction de la tension V et des résistances R_1, R_2 et R_3.

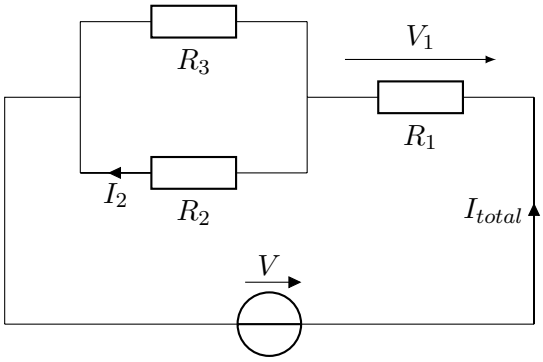

FIGURE 1 – Circuit électrique

Mathématiquement, on peut écrire :

$$
\begin{aligned}
R_{total} &= R_1 + \frac{1}{\frac{1}{R_2} + \frac{1}{R_3}} \\
I_{total} &= \frac{V}{R_{total}} \\
V_1 &= R_1 I_{total} \\
V_2 &= V - V_1 \\
I_2 &= \frac{V_2}{R_2}
\end{aligned}
$$

On va maintenant traduire ces formules en un programme Python. Le programme ressemble donc à :

```
R1 = 2
R2 = 4.2
R3 = 10
V = 12

R_total = ...
I_total = ...
V1 = ...
V2 = ...
I2 = ...

print("L'intensité I2 vaut", I2)
```

Il faut bien sûr remplacer les `...` par les expressions correctes. Quelques remarques :

– Par défaut, on manipule des nombres sans unités. C'est au programmeur de choisir sa convention (par exemple, si les résistances sont en megohm et les tensions en Volts, alors le résultat sera en milliampères).

– Les notations mathématiques doivent être exprimés en Python donc en texte sans mise en forme. On ne peut pas écrire I_{total} mais on écrit par exemple `I_total`.
– La multiplication doit être explicitée avec l'opérateur `*`, on ne peut pas écrire `R1 I_total` par exemple.

Exercice 9 (Calcul de I_2) *Complétez le programme ci-dessus pour calculer I2. Exécutez-le et vérifiez qu'il fonctionne (avec les valeurs ci-dessus, le résultat doit être 1.7045...). Essayez de modifier les valeurs d'entrée et de ré-exécutez le programme : on peut bien sûr refaire le calcul plusieurs fois.*

3.2 Manipulations de variables

On considère le squelette de programme suivant :

```
x = 42
y = 3

print("avant : x =", x, " y =", y)

# ...

print("après : x =", x, " y =", y)
```

Exercice 10 (Échange des valeurs de deux variables) *Écrire (à la place du `# ...`) un programme qui échange les valeurs de `x` et de `y`. Vérifiez que votre programme fonctionne correctement.*

4 Structure conditionnelle : `if/else`

Tous les programmes réalisés jusqu'ici sont *inconditionnels* : les lignes du programmes sont toutes exécutées, les unes après les autres. On peut bien sûr écrire des programmes qui n'exécutent certaines parties que quand une condition est vraie. La syntaxe générale est :

```
if condition :
    une ou plusieurs instructions
suite du programme
```

La partie *une ou plusieurs instructions* ne sera exécutée que si *condition* est vraie.
Par exemple :

```
print("Entrez un nombre")
x = int(input())

if x >= 0:
    print("x est positif ou nul")
if x <= 0:
    print("x est négatif")
    print("ou bien x est nul")
if x == 0:
    print("x est nul")
print("fin du programme")
```

Attention :
– L'indentation est importante : il faut des espaces (par convention, 4 espaces) avant *une ou plusieurs instructions*. La fin du `if` correspond à la fin du bloc indenté.
– L'opérateur `==` est l'opérateur de comparaison (« est-ce que `x` est égal à 0 ? »), différent de l'opérateur d'affectation (`x = 0`, pour « `x` devient égal à zero »).

Exercice 11 *Entrez ce programme dans l'éditeur de texte, et exécutez-le. Essayez avec des valeurs de `x` négative, positives, et avez 0. Essayez de remplacer `>=` et `<=` par `<` et `>`. Quelle est la différence ?*

5 Solution des exercices

Exercice 1 : `**` est l'opérateur « puissance » (`x ** y` calcule x^y), et `%` est l'opérateur « modulo » (ou reste de la division entière). L'opérateur `/` est la division flottante, alors que `//` est la division entière (le résultat est un nombre entier).

Exercice 4 : si on entre autre chose qu'un entier, la conversion `int(...)` échoue avec le message « invalid literal for int() with base 10 : '...' ».

Exercice 5 : entrer une expression dans l'interprète évalue l'expression et affiche son résultat. Dans un vrai programme, le fait de faire un calcul n'implique pas d'afficher son résultat (imaginez si les logiciels que vous utilisez vous affichaient tous les résultats de tous les calculs qu'ils font !). Depuis un programme, il faut donc demander explicitement, avec l'instruction **`print`**, si on veut faire un affichage. A noter qu'évaluer `"Bonjour"` dans l'interprète affiche `'Bonjour'`, avec les guillemets (pour que l'utilisateur puisse voir qu'il s'agit de « la chaîne de caractère `'Bonjour'` »), alors que l'instruction **`print`** n'affiche pas les guillemets : l'utilisateur final du programme n'a pas besoin de savoir comment sont représentées les chaînes en Python !

Exercice 7 : il n'y a aucun affichage, l'erreur de syntaxe arrive avant le premier **`print`**.

Exercice 8 : cette fois, on voit bien le premier affichage `Début du programme`, mais la division par zéro entraîne un arrêt du programme : on voit un message d'erreur (qui doit indiquer la ligne 2 de notre programme), et la ligne `print("Fin du programme")` n'est jamais exécutée.

Exercice 9 :

```
R1 = 2
R2 = 4.2
R3 = 10
V = 12

R_total = R1 + 1 / (1/R2 + 1/R3)
I_total = V / R_total
V1 = R1 * I_total
V2 = V - V1
I2 = V2 / R2

print("L'intensité I2 vaut", I2)
```

Exercice 10 :

```
x = 42
y = 3

print("avant : x =", x, " y =", y)

tmp = x
x = y
y = tmp

print("après : x =", x, " y =", y)
```

Exercice 11 : `<=` et `>=` sont les opérateurs d'inégalités larges (supérieur/inférieur ou égal), alors que `<` et `>` sont les inégalités strictes.