Sophie Schüpp

Licence

Python en licence

Algorithme de base et algorithme scientifique

ellipses

Chapitre 1 : Analyse séquentielle

1.1 L'interpréteur : IDLE

Nous utiliserons IDLE qui est spécialement dédié à Python (et qui se télécharge automatiquement en même temps). On peut utiliser Python directement (non conseillé ensuite…). Le symbole « >>> » précise qu'on a la main. Notre premier écran Python :

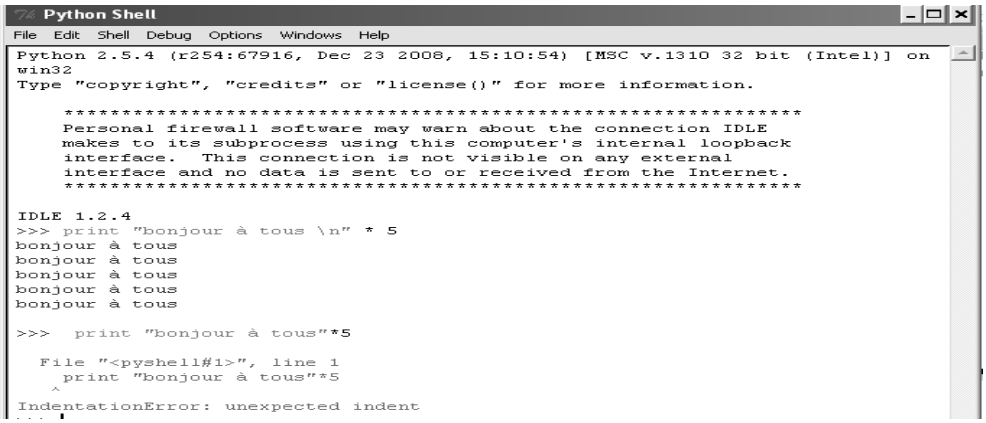

Fig. 1 Interface IDLE partie interpréteur

La particularité de Python est que le début et la fin de bloc d'instructions ne sont pas indiqués par un mot comme dans les autres langages structurés (C++, Java, …). Les instructions du bloc imbriqué dans le précédent doivent être indentées. i.e. des espaces en nombre fixe et paramétrable doivent être écrits en début de ligne, sinon il peut y avoir un message d'erreur. Le message d'erreur contient le nom du fichier, le numéro de ligne, l'instruction et le type d'erreur. Les erreurs de syntaxe sont gérées l'une après l'autre et provoque un message d'erreur bloquant.

Il est plus pratique et efficace d'utiliser un éditeur (comme emacs ou autres...). L'éditeur IDLE qui est téléchargé avec Python est très fonctionnel. Il permet de récupérer le numéro de ligne et de colonne du curseur. Les erreurs sont immédiatement localisées. Il permet d'ouvrir des fichiers pour y écrire des scripts (des programmes) variés, de les sauvegarder, puis de les exécuter. L'utilisation est très simple :

- On ouvre un fichier (ouverture ou création d'un nouveau fichier avec le menu déroulant)
- On écrit son script

- On le sauvegarde (« CTRL » S ou « enregistrer » ou « enregistrer sous » dans le menu déroulant)
- On l'exécute (F5 ou « **run** » dans le menu déroulant).

Chaque appel de « **run** » remet la mémoire à zéro (affichage de RESTART).
Le caractère # ouvre une ligne de commentaire qui ne sera pas lue par le programme (pas de symbole de fin de commentaire).

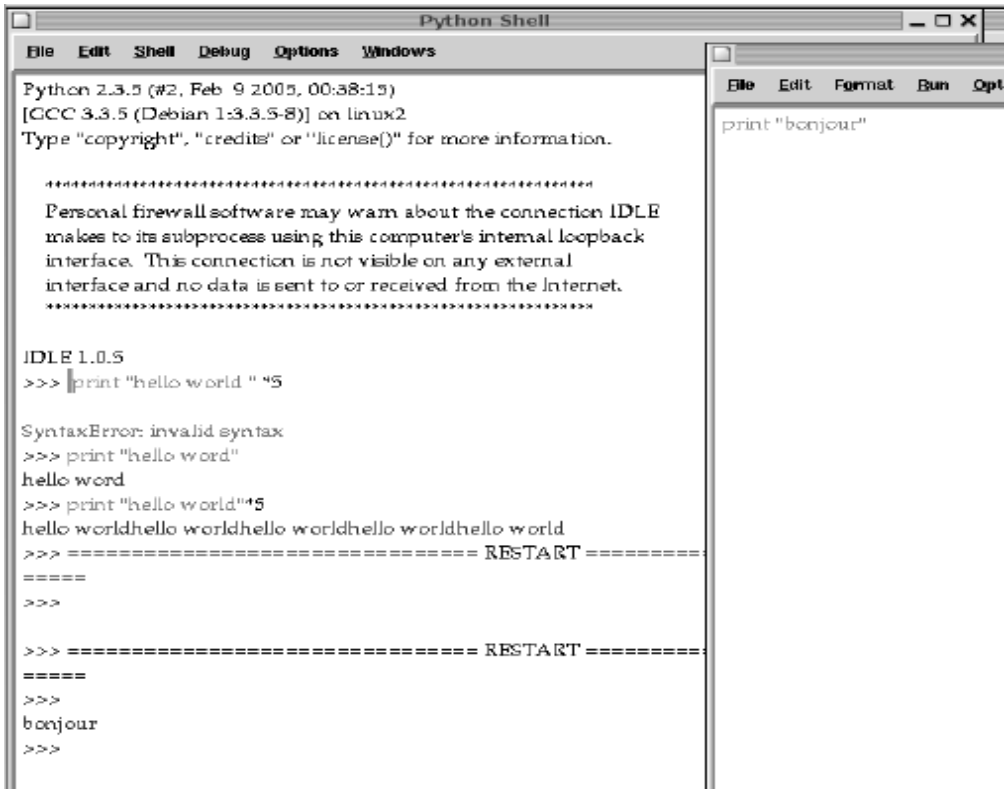

*Fig. 2 Exemple de d'utilisation d'un fichier *.py*

1.2 Les calculs

Division entière opérateur

Python et l'éditeur associé IDLE peuvent être utilisé comme calculatrice. Une fois le calcul entré, on fait « enter » au lieu de taper « = ». Dans les versions supérieures à 3.0, la division entre deux nombres entiers ou réels est réelle. Pour avoir une division entière il faut utiliser « // ». Le résultat est l'entier inférieur le plus proche. Si un des deux nombres est de type réel, le résultat est de type réel, sinon de type entier.

```
x = 9/2
n = 9//2
z = 9.0//2
print("x = ", x, "n = ", n, "z = ", z)
x = 4.5 y = 4 z = 4.0
```

Ordre de priorité

En Python, les règles de priorité sont les mêmes que celles des mathématiques. On les mémorise à l'aide de l'acronyme PEMDAS : P pour parenthèses, E pour exposants, M et D pour multiplication et division, qui ont la même priorité, A pour addition, S pour soustraction.
Les parenthèses ont la plus haute priorité. Elles permettent donc de « forcer » l'évaluation d'une expression dans l'ordre désiré. Ainsi (1+1)**(5–2) = 8 et 2*(3–1) = 4. Les puissances sont évaluées ensuite (après les expressions entre parenthèses), avant les autres opérations. Ainsi 2**1+1 = 3 (et non 4), et 3*1**10 = 3 (et non 59049!). Les multiplications et les divisions ont la même priorité et sont évaluées avant l'addition A et la soustraction S, lesquelles sont donc effectuées en dernier lieu. Ainsi 2*31 = 5 (plutôt que 4), et 6/31 = 1
Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression 59*100/60, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer 5900/60, ce qui donne 98. Si la division était effectuée en premier, le résultat serait 59 (puisqu'il s'agit d'une division entière).

1.3 Les variables

Règles du choix des noms

En Python, les noms de variables doivent obéir à quelques règles simples. Un nom de variable est une séquence de lettres (a ... z, A ... Z) et de chiffres (0…9), qui doit toujours commencer par une lettre (il est conseillé de prendre des noms explicites !!!). Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (tiret du bas). En particulier – est interdit. La casse est significative (les caractères majuscules et minuscules sont distingués).
Attention : largeur, Largeur et LARGEUR sont donc des variables différentes, x et X peuvent cohabiter (mais ce n'est pas conseillé !!!).
On écrit généralement les noms de variables en caractères minuscules (y compris la première lettre, mais ce n'est pas une obligation). On n'utilisera généralement les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans « **`tableDesMatieres`** », par exemple.

En plus de ces règles, certains mots sont réservés, ils ne peuvent donc pas servir de nom de variable. En particulier, les mots du langage Python sont interdits. Ils sont nombreux car Python est un langage Orientés Objets et possède de plus de nombreuses bibliothèques de fonctions prédéfinies. Les mots ci-dessous sont les plus courants :

```
and assert class def del elif else for from global if import in is lambda not
or pass print raise return while as
```

Il faut aussi tenir compte de toutes les fonctions prédéfinies qui ne peuvent pas servir de nom de variables. Un message d'erreur plus la coloration du mot en orange vous prévient de la confusion possible.

Affectation

L'affectation se fait avec le symbole « = », pour une définition ou une modification, avec le nom en premier. Une affectation multiple est possible. Les variables sont séparées par des virgules, ainsi que les valeurs respectives dans le même ordre. Si l'affectation est mal faite, il y a un message d'erreur.

```
x = 5
phrase = ”bonjour tout le monde”
x, y = 2.4, 8
z = x + y
```

L'affectation crée et mémorise un nom de variable, attribue un type donné, crée et mémorise une valeur particulière, établit un lien (système de pointeurs) entre le nom et l'emplacement mémoire de la valeur correspondante. En Python : la déclaration de type est inutile. Le typage se fait automatiquement (typage dynamique). La fonction prédéfinie permet de connaître le type affecté à la variable.
Une **affectation** utilise le signe « = » et elle signifie que l'on veut mettre un nombre dans une variable. L'ordinateur détermine (évalue) tout d'abord la valeur à droite du signe « = », en faisant tous les calculs nécessaires, elle est transférée dans la mémoire dont le nom est indiqué à gauche du « = ». Cela n'a pas grand-chose à voir avec l'égalité mathématique.

On peut écrire « a = 5 » mais pas « 5 = a »
On ne peut pas écrire « a + b = 0 »
« a = a+1 » signifie : regarder combien vaut a, ajouter 1, puis stocker le résultat dans « a » (son ancienne valeur étant alors écrasée par la nouvelle).

Type de variables

```
x = 4
y = 15.3
ph = ”bonjour”
print( type(x)) → <type 'int'>
```

```
print(type (y)) → <type 'float'>
print(type(ph)) → <type 'str'>
```

- int : un entier entre –2^31 +1 et 2^31 –1, arrondi au plus près à un entier.
- long : taille de l'entier non limitée x = 2345655555555
- float : nombre à virgule
- complex

1.4 Quelques opérateurs

- « **x**n** » ou pow(x, n) renvoie « x » à la puissance « n ».

```
x = 4
print(4**3) → 64
print(4**0.5) → 2.0
print(4**(1/2)) → 2.0
n1 = 3
n2 = 1/2
n3 = –1
y1 = x**n1
y2 = x**n2
y3 = x**n3
print(x, "à la puissance", n1, " = ", y1, "–", x, "à la puissance", n2, " = ",
y2, "–", x, "à la puissance", n3, " = ", y3)
➔ 4 à la puissance 3 = 64 – 4 à la puissance 0.5 = 2.0 – 4 à la
   puissance –1 = 0.25
```

- « **a%b** » Renvoie le reste de la division euclidienne de « a » par « b »:

  si a = b q + r alors a%b renvoie r

```
print(24%5) → 4
```

- La fonction « **min** » renvoie le minimum de la liste des paramètres, de même « **max** » renvoie le maximum.

```
a = 1
b = 2
c = 1
delta = b*b–4*a*c
valmin = min(a, b, c, delta)
valmax = max(a, b, c, delta)
print("a=",a,"b=",b,"c=",c) → a= 1 b= 2 c= 1
print("delta=",delta,"minimum=",valmin,"max=",valmax)
 → delta= 0 minimum= 0 max= 2
```

1.5 Les nombres complexes

Python gère automatiquement les nombres complexes. Mais attention, le complexe « a + ib » se note « a + jb » et il faut obligatoirement écrire « b » (même si b = 1); on peut considérer les imaginaires purs : « 1j ». Python gère : addition, soustraction, multiplication et division entre complexes, puissance…

```
print((3+5j)3+7j)        → 12j
print((3+5j)(3+7j) )     → 2j
print( (3+5j)/(3+7j) )   → (0.758620689655172380.10344827586206896j)
print( (3+5j)*( 2+7j) ) →(29+31j)
print( (1+2j)**2 )       →(3+4j)
```

1.6 Dialogue interactif : saisie au clavier, affichage écran

1.6.1 L'instruction « print » : affichage à l'écran

L'instruction « **print** » permet l'affichage de données (n'importe quel type de données). Pour afficher le contenu d'une variable, il faut indiquer le nom de la variable sans guillemet. Si le nom de la variable est encadré de guillemet, Python interprète le paramètre comme une chaîne de caractères et n'affiche pas le contenu de la variable mais le nom de la variable. Une virgule entre les données à afficher permet de mettre plusieurs affichages sur une même ligne. A partir de la version 3.0, il faudra mettre les arguments dans des parenthèses.
A partir de la version 3, la fonction print possède des paramètres qui permettent de spécifier le séparateur affiché (« **sep** ») et d'empêcher le passage à la ligne (« **end** »). Par défaut, le paramètre « **sep** » a la valeur « espace » et le paramètre « **end** » a pour valeur « **\n** » qui est un passage à la ligne.

```
print(2+3) → 5
print( "résultat du calcul" , "2+3 = ", 2+3) → résultat du calcul 2+3 = 5
ph1 = "marquise"
ph2 = 'vos beaux yeux'
ph3 = "d'amour"
ph4 = 'mourir'
ph5 = 'me font'
ph7 = 'hello world'
print( ph7) → hello world
print( ph1, ph3, ph2, ph4, ph5) → marquise d'amour vos beaux yeux
mourir me font
x = "premier cours de Python"
y = x[7:18]
```

```
print("x", x, sep = " = ", end = " ## ")
print("y = ", y)          → x = premier cours de Python ## y =  cours de P
```

1.6.2 L'instruction « input() » : saisie au clavier

La saisie au clavier et l'affichage à l'écran font partie de la gestion des entrées/sorties. Un programme est une suite finie et ordonnée d'instructions (un algorithme) pour résoudre un problème. Le programme a besoin de données initiales – les entrées – et calcul et renvoie les résultats – les sorties –. Les entrées sont données aux programmes sous différentes formes : saisies au clavier en interactif, lues dans un fichier, résultats issus d'un autre programme, mesures de capteurs, … De même pour les sorties, il existe différentes formes, les résultats peuvent s'afficher à l'écran, s'écrire dans un fichier, être transmis à un autre programme, …
L'appel de « **input(message)** » va provoquer l'affichage de la chaîne message (ce doit être une chaîne) et attendre que l'utilisateur écrive quelque chose puis tape sur « **enter** » pour valider la saisie. Attention, la saisie de l'utilisateur est obligatoirement une chaîne de caractères qu'il faut transformer en nombre pour faire les calculs.
La plupart des scripts élaborés nécessitent à un moment ou un autre une intervention de l'utilisateur (entrée d'un paramètre, clic de souris sur un bouton, etc.). Dans un script simple en mode texte (comme ceux que nous avons créés jusqu'à présent), la méthode la plus simple consiste à employer la fonction intégrée « **input()** ». Cette fonction provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à terminer avec la touche « **enter** ». Lorsque cette touche est enfoncée, l'exécution du programme se poursuit et la fonction fournit en retour une valeur correspondant à ce que l'utilisateur a entré. Cette valeur peut alors être assignée à une variable quelconque. On peut invoquer la fonction « **input()** » en laissant les parenthèses vides. On peut aussi y placer en argument un message explicatif destiné à l'utilisateur.

Exemple :

```
print( 'Veuillez entrer un entier positif quelconque : '),
nn = int(input())
print( 'Le carré de', nn, 'vaut', nn**2) ou encore :
prénom = input('Entrez votre prénom : ')
print( 'Bonjour, ', prénom )
```

Remarques importantes :
• La fonction « **input()** » renvoie une valeur dont le type est une chaîne de caractères. Dans notre exemple, la variable « **nn** » contient donc une chaîne de caractères qui est transformée en entier avec la fonction « **int()** ». La fonction « **input()** » renvoie toujours *une chaîne de caractères*. Vous pouvez ensuite convertir cette chaîne en nombre à l'aide de « **int()** » ou de « **float()** ».

Exemple :

```
a = input('Entrez une donnée : ')
Entrez une donnée : 52.37
print(type(a))   → <type 'str'>
b = float(a) # conversion en valeur numérique
print( type(b))  → <type 'float'>
 x = int(input( “age”))
25
print(x)   → 25
 nom = input (“votre nom “)
“Dupont”
print(nom)   → “DuPont”
```

La fonction **raw_input(“message”)** n’existe plus à partir de la version 3.

1.7 Chaînes de caractères

Une chaîne de caractères (c'est le type « **string** ») est une suite de caractères délimitée par des guillemets ou des apostrophes (ces guillemets ou apostrophes ne faisant bien évidemment pas partie de la chaîne). On mettra obligatoirement des guillemets si la chaîne contient une apostrophe et des apostrophes si la chaîne contient des guillemets. On doit impérativement mettre le même symbole au début et à la fin de la chaîne.
Attention : 2 apostrophes ne font pas des guillemets.

1.7.1 Opérations élémentaires sur les chaînes

L’addition + et la multiplication * s'appliquent aux chaînes. L’addition permet de concaténer (mettre bout à bout) les chaînes de caractères ou les variables représentants des chaînes. La multiplication permet de répéter n fois une chaîne.

```
print(ph1+ph2+ph3+ph4+ph5)      →
"marquisevos beaux yeuxd'amourmourirme font"
print(ph7 * 2)   → 'hello worldhello world'
```

1.7.2 Longueur d'une chaîne : len(chaine)

La fonction « **len()** » renvoie la longueur (le nombre de caractères la composant) de la chaîne passée en paramètre.

```
print(len (“bonjour”))      → 7
print(len(ph2+ph3))         → 21
```