Introduction à Spyder pour python

1 Spyder

1.1 Introduction

Ce document est divisé en deux parties: d'abord une présentation des principales fonctionnalités de Spyder, puis des exercices pour essayer Spyder soi-même.

"Spyder" est un "environnement de développement intégré" ou IDE en anglais pour "Integrated Development Environment". Spyder est fait spécifiquement pour python appliqué aux sciences (Spyder pour "The scientific Python Development Environment"). Spyder est disponible gratuitement sous Windows, Mac et Linux:

`https://www.spyder-ide.org/`

Spyder est complémentaire aux Jupyter-notebooks et fournit des outils à la programmation non disponibles sous Jupyter, comme notamment son explorateur de variables (voir section 1.6) et son débogueur graphique intégré (voir section 1.10).

Spyder est développé depuis plus de 10 ans en projet open source et utilisé par environ un demi-million de personnes dans le monde, d'après cet article:

`https://labs.quansight.org/blog/2021/04/a-step-towards-educating-with-spyder/`

1.2 Installation

Spyder 4 est déjà fournit avec Anaconda:

`https://www.anaconda.com/products/individual`

donc pas besoin de l'installer à part. La dernière version, Spyder 5, doit par contre être installée pour l'instant à part, voir ici: `https://docs.spyder-ide.org/current/installation.html`

Sur les ordinateurs à l'Atrium, il y a actuellement uniquement la version 3 de Spyder, mais cela suffit largement pour cette UE. Capsule de Sorbonne Université fournit également un accès à distance via le bureau de l'UTES, donc on peut aussi utiliser Spyder en ligne sans rien installer sur son ordi:

`https://lutes.upmc.fr/bdl-ext.php`

Une fois connecté avec votre login habituel, il faut chercher "Spyder" comme application (cliquer sur l'icon "Loupe" en haut à droite pour rechercher) et choisir la dernière version pour python 3 (actuellement Spyder 3.2.3 pour python 3.6).

1.3 Jupyter notebooks dans Spyder

Même si Spyder n'est pas développé à la base pour utiliser des Jupyter notebooks, c'est possible de les ouvrir avec Spyder, mais uniquement en installant ce Plugin pour Spyder :

`https://docs.spyder-ide.org/current/plugins/notebook.html`

Ce Plugin n'est pas installé à l'Atrium et s'installe uniquement avec Spyder 4 sous Anaconda chez soi. Pour Spyder 5 ce Plugin ne fonctionne pas encore à ce jour.

1.4 Console IPython

Dans Spyder, dans la fenêtre en bas à droite il y a la "Console IPython". Cliquer sur l' invite de commande marquée avec "In [1]:" pour pouvoir taper des commandes python (voir figure 1)

Pour agrandir cette fenêtre de la console, utiliser sur la barre de menu principal de Spyder **Affichage - Agrandir le volet courant**. Sur ce menu il y a aussi indiqué le raccourci clavier pour cette opération: **Ctrl+Alt+Maj+M**.

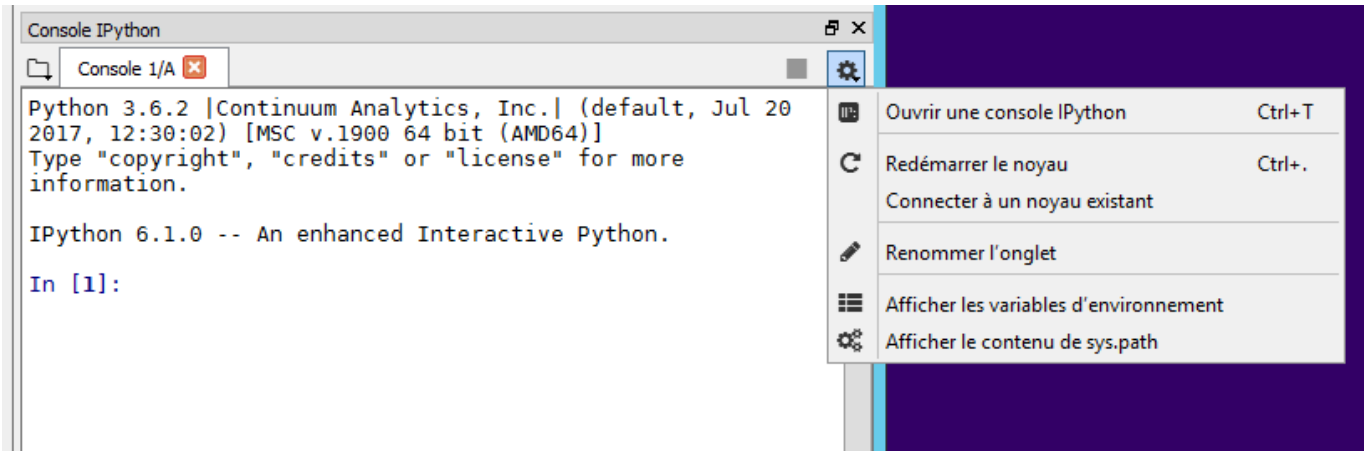

Figure 1: Console IPython avec menu local ouvert (bouton coin en haut à droite). On peut y redémarrer le noyau de python par exemple, pour démarrer de zéro.

1.5 Explorateur de fichiers

Utiliser l'explorateur de fichiers de Spyder pour créer un nouveau dossier au début de chaque session.

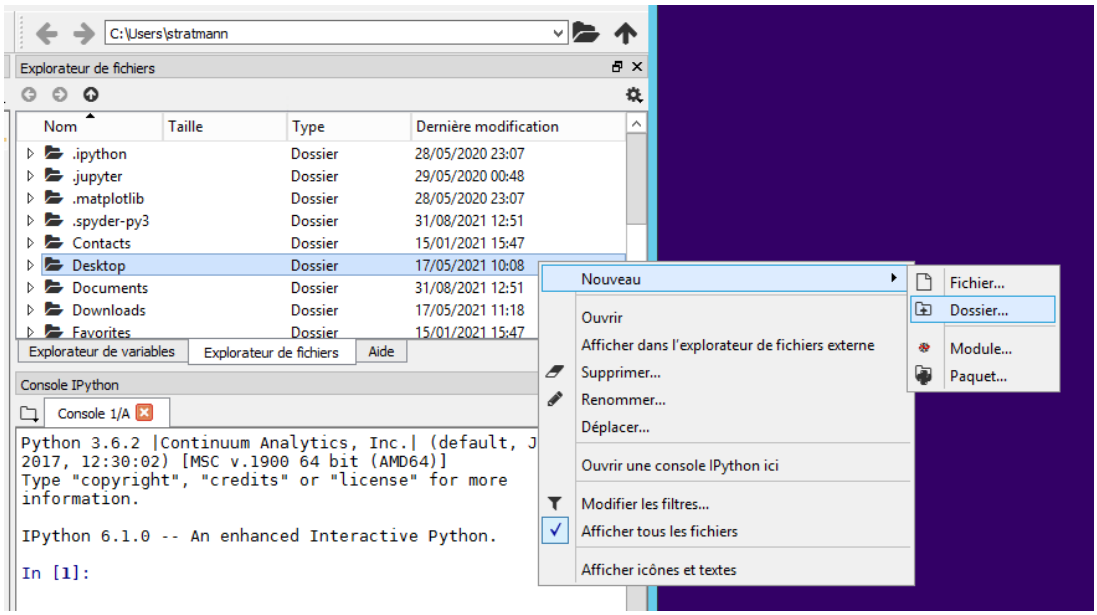

Figure 2: L'onglet en bas ouvre l'explorateur de fichiers. En haut on voit le chemin du répertoire actuel. Les flèches permettent de naviguer. Le menu local, ouvert avec un clique-droit de la souris, permet de créer un nouveau dossier ou fichier.

1.6 Explorateur de variables

L'explorateur de variables permet d'afficher le contenu de la mémoire en temps réel, ce qui est très utile pour déboguer son code (voir section 1.10).

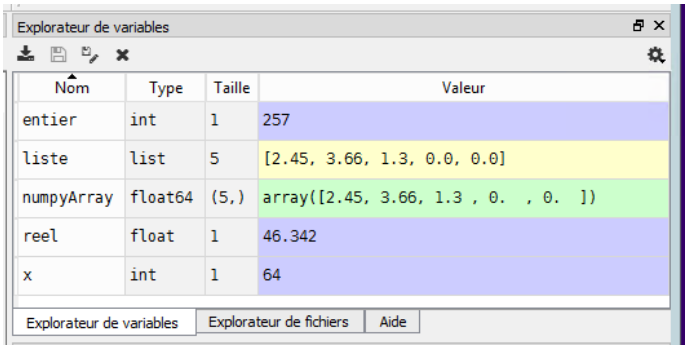

Figure 3: L'explorateur de variables affiche les valeurs et types des variables actuellement en mémoire que ce soit un script qui est exécuté ou la console IPython.

1.7 Aide

Pour obtenir de l'aide sur une fonction de python ou un module, on peut utiliser l'onglet "Aide". Il suffit de cliquer sur la fonction en question puis de faire **Ctrl+I**, ceci marche dans la console et dans les scripts. S'il s'agit d'une fonction d'un module, alors il faut d'abord importer ce module (ou écrire "import module" dans le script) pour que l'aide s'affiche.

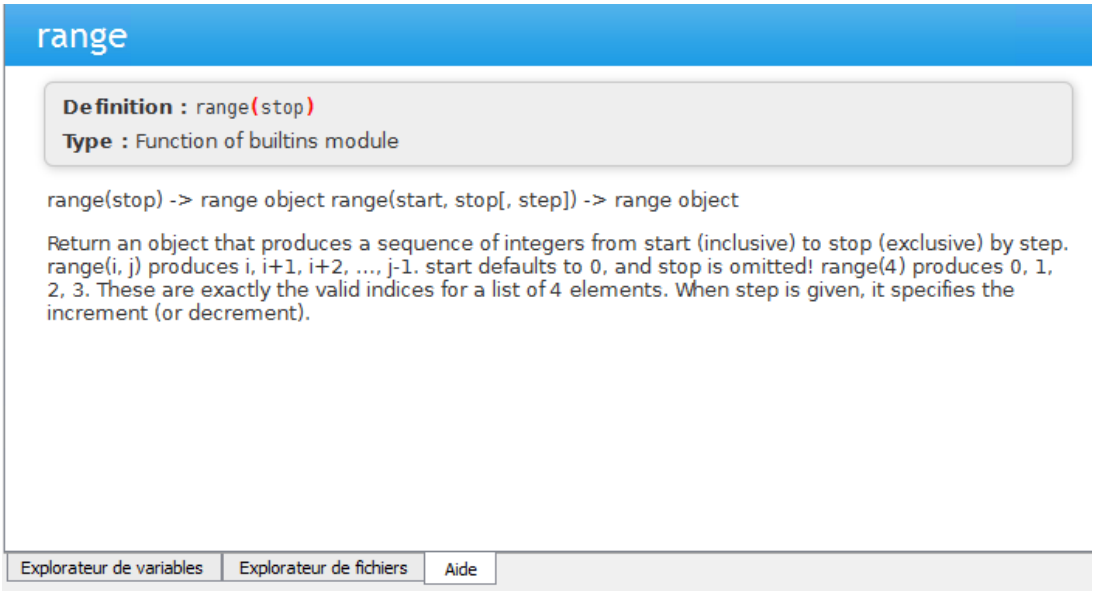

Figure 4: Panneau "Aide": une aide python intégrée à Spyder (en anglais)

1. *Exercice:* Lire la documentation de range() dans l'onglet "Aide" de Spyder. Décrire ce que fait range() et ses différentes variantes avec un, deux ou trois arguments.

1.8 Créer un script python

Une fois qu'on a expérimenté avec les commandes de python dans la console IPython, il est très important de garder une trace de la suite de commandes retenues au final pour une tâche donnée. Cela permet de reproduire très facilement les résultats du calcul, juste en lançant le script, mais surtout de faire des petits modifications au besoin en modifiant des parties du script.

Les scripts sont édités dans Spyder dans la fenêtre "éditeur" en haut à gauche. Pour créer un nouveau script utiliser le menu **Fichier - Nouveau fichier...** ou bien le raccourci clavier **Ctrl+N** après avoir cliqué dans la fenêtre éditeur. Un nouvel onglet avec un nom de fichier "sanstitre0.py" sera ouvert. Dans ce fichier on copie+colle les commandes retenues depuis la console IPython (utiliser les raccourcis clavier **Ctrl+C** pour copier puis **Ctrl+V** pour coller).

Par rapport à la console IPython on doit ajouter deux choses dans un script:

1. La ligne `# -*- coding: utf-8 -*-` indique qu'on utilise l'encodage UTF-8, ce qui est nécessaire pour pouvoir utiliser des caractères spéciaux comme les accents. Le signe # est utilisée pour un commentaire ou encore pour indiquer ce genre d'options. Spyder insère automatiquement cette ligne à chaque nouveau fichier.

2. Importer les modules ou fonctions dont on a besoin.

1.9 Exécuter un script python

Pour exécuter un script dans Spyder on affiche d'abord les "Options d'exécution" avec **Ctrl+F6**, puis on choisit dans la rubrique **Console** la deuxième option "Exécuter dans une nouvelle console Python dédiée" (voir figure 5). Ceci a l'avantage de ne pas mélanger vos commandes et variables du script avec l'état actuel de votre console IPython. Ensuite démarrer votre script avec la touche **F5** ou encore avec les flèches bleus pour le mode débogage (voir menus Exécution et Déboguer). Les commandes **print** font un affichage dans la fenêtre de la console (voir figure 6).

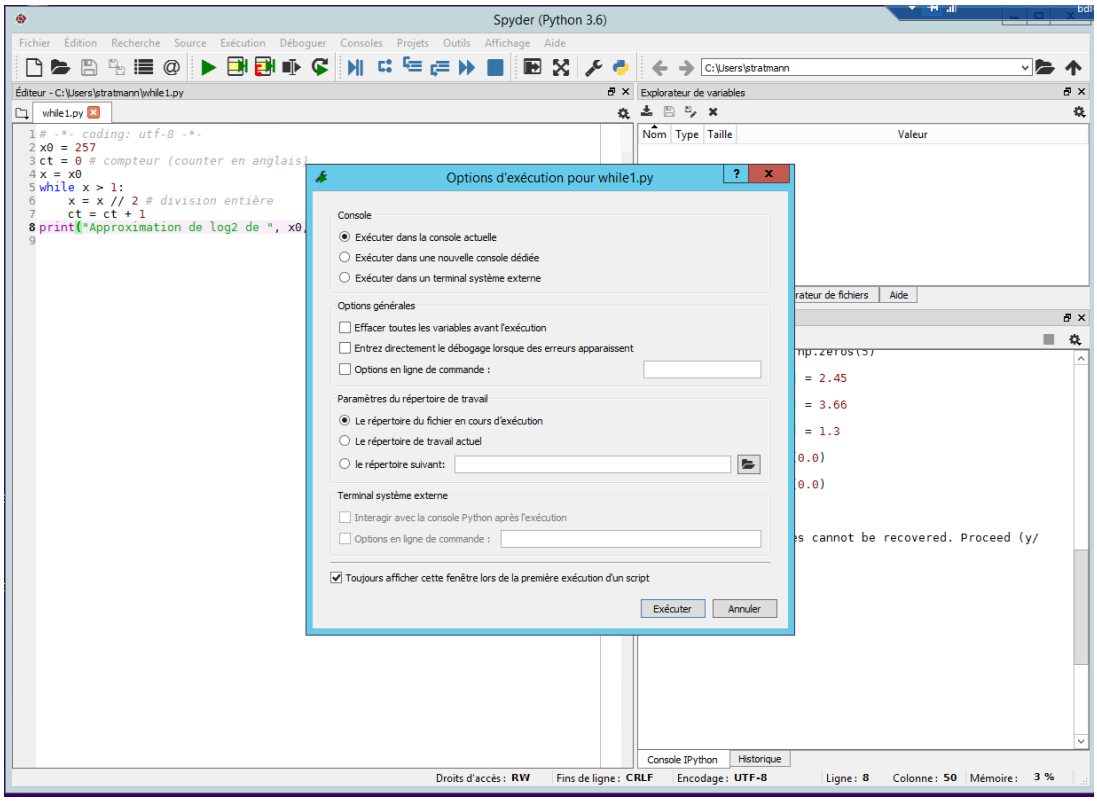

Figure 5: Options d'exécution (**Ctrl+F6**)

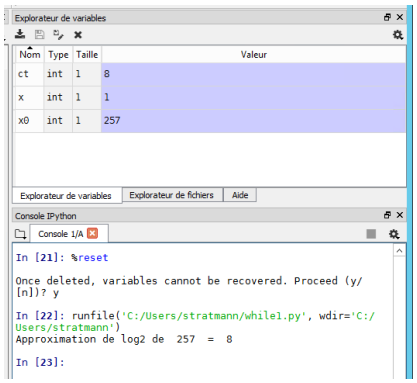

Figure 6: Après l'exécution de while1.py le résultat est affiché dans la console IPython et la mémoire n'est pas effacée comme on peut voir dans l'explorateur de variables.

1.10 Déboguer

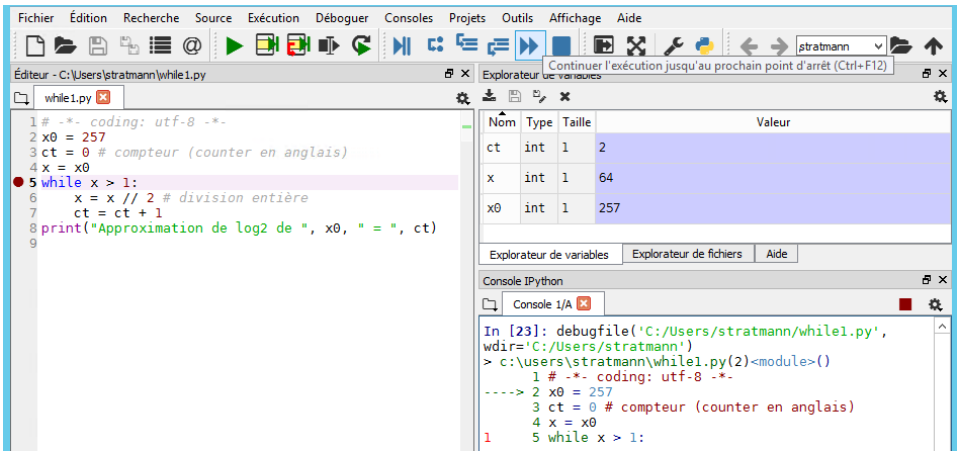

Figure 7: Le débogueur permet d'exécuter un script pas à pas et de voir avec l'explorateur de variables le contenu de celles-ci en temps réel. Pour démarrer et contrôler le débogueur on peut utiliser les flèches bleus. En restant avec la souris sur une des flèches on obtient sa fonction et le raccourci clavier associé, par exemple ici Ctrl+F12 pour continuer l'exécution jusqu'au prochain point d'arrêt. Alternativement on peut aussi utiliser le menu "Déboguer". Pour mettre ou enlever un point d'arrêt (breakpoint en anglais) sur une ligne de code, on double-clique simplement sur cette ligne à gauche du numéro. Un point d'arrêt est indiqué avec un point rouge. Pour démarrer son script avec le débogueur il faut utiliser la première flèche bleu ou alors Ctrl+F5. La ligne d'exécution actuel est indiqué avec un fond rose, ici l'exécution s'est arrêtée momentanément à cause du point d'arrêt à la ligne 5. Dans l'explorateur de variables à droite on voit le contenu actuel des variables après deux passages dans la boucle while.

1.11 Raccourcis clavier principaux

Pour l'utilisation de Spyder voici les raccourcis clavier les plus importants (penser à activer les touches "fonction" F1-F12 avec la touche "F Lock", "Fn" ou similaire si c'est nécessaire sur votre clavier):

- Ctrl+N : nouveau fichier (ou pomme à la place de Ctrl sous Mac OS X)
- Ctrl+O : ouvrir un fichier, Ctrl+S : sauvegarder un fichier
- Ctrl+Z : annuler, Ctrl+C : copier, Ctrl+V : coller, Ctrl+X : couper
- Ctrl+Alt+Maj+M Agrandir/Réduire le volet courant
- F5 lancer votre programme (voir menu "Exécution")
- Ctrl+F6 Options d'exécution
- Ctrl+F5 Activer le débogueur (voir menu "Déboguer")
- Ctrl+F10 *Déboguer:* pas à pas
- F12 *Déboguer:* Ajouter un point d'arrêt
- Ctrl+F12 *Déboguer:* Continuer jusqu'au prochain point d'arrêt

2 Exercices avec Spyder

2.1 Console IPython

Dans cette partie on utilisera la console IPython de Spyder pour tester les exemples et pour répondre aux questions. Pensez à agrandir le volet de la console avec Ctrl+Alt+Maj+M une fois cliqué dedans.

L'invite de commande de IPython est précédé de "In [XX]: " avec XX le compteur des commandes exécutées par le noyau ("Kernel" en anglais) de python. Une commande est exécutée simplement avec la touche "Entrée" ou "Return" (touche "retour à la ligne"), pour avoir plusieurs lignes de code il faut utiliser "Ctrl" + "Return" au début, puis "Maj" + "Return" (ou "Return" après une ligne vide) pour exécuter le bloc de lignes de code.

2.1.1 Calculatrice

2. Tester ces opérations dans la console IPython de Spyder avec une opération par invite de commande "In [XX]:" :

```
20 + 3
20 - 3
20 * 3
20 ** 3
20 / 3
```

```
20 // 3  # division entière
20 % 3   # modulo
abs(3 - 20)  # valeur absolue
```

2.2 Explorateur de variables

2.2.1 Introduction

Les variables en programmation stockent des valeurs. Comme leur nom le dit, leur valeur peut changer au cours de l'exécution d'un programme. La valeur d'une variable est stockée dans une case de la *mémoire vive* ou *mémoire système* (*RAM* en anglais pour Random Access Memory) de l'ordinateur. La mémoire vive contient juste une suite de zéros et de uns, par exemple: 011001110010101000... La conversion de ce *code binaire* en une valeur d'une variable dépend du **type** de la variable. Par exemple, la même suite de zéros et de uns donne une valeur différente si on la convertit en une valeur entière ou en une valeur réelle. C'est pour cela que chaque variable en python a un type, même si on ne le déclare pas explicitement. Le type d'une variable détermine à la fois la manière de convertir le code binaire et la longueur de la suite des zéros et des uns, c'est-à-dire la taille de la case mémoire.

2.2.2 Types de variables en python

Les deux types qu'on utilise principalement dans cette UE sont **int** et **float**.

En général on n'utilise pour les calculs que des variables de type **float**, à moins d'être certain que la variable en question ne prend que des valeurs entières, comme un compteur par exemple. Lors de calculs qui impliquent des variables de type **int**, il faut faire attention aux erreurs dues à l'arrondi.

2.2.3 Définir le type d'une variable

Le type d'une variable est déterminé dans Python lors de l'affectation de celle-ci. Pour connaitre de le type d'une variable on peut soit utiliser la fonction `type(nom_de_variable)` de python, soit ouvrir "l'Explorateur de variables" de Spyder (voir section 1.6).

3. Observer dans "l'Explorateur de variables" de Spyder ce qui se passe quand on entre ces lignes de code un par un dans la console IPython:

```
x = 10
y = 10.0
z = 10.
z = x
y = 2.5e4
y
y = 2.5e16
y
```

On doit avoir observé que dans cet exemple les variables sont soit de type **float** soit de type **int** en fonction de ce qu'elles doivent stocker. Noter aussi que le type d'une variable peut changer en cours de route, ce qui est impossible dans d'autres langages de programmation comme le C. Par contre dans les scripts python il est déconseillé de changer en cours de route le type d'une variable car cela peut porter à confusion.

4. Pour convertir le type d'une donnée on peut utiliser les fonctions **int()**, **float()**, **str()**, **complex()**, **bool()**. Voici quelques exemples à tester dans la console:

```
int("300")
str(300)
float("3.14")
float(10)
int(3.99)
bool(0)
bool(1)
bool(-12345)
```

2.2.4 Formules et variables

5. La formule de conversion d'une température en Fahrenheit en une température en Celsius est

$$T_C = (T_F - 32) * 5/9$$

Calculer la température en Celsius pour 189 degré Fahrenheit. Utiliser des variables pour T_C et T_F. Afficher leur valeur simplement en tapant leur nom. De quel type doivent être ces deux variables ? Vérifier leur type avec la commande `type(nom_de_variable)`. Vérifier leur type et valeur avec "l'explorateur de variables" de Spyder (voir section 1.6).

Correction.

```
In [1]: ( 189.0 - 32.0 ) * 5.0 / 9.0
Out[1]: 87.22222222222223

In [2]: T_in_F = 189.0
In [3]: T_in_C = ( T_in_F - 32.0 ) * 5.0 / 9.0

In [4]: T_in_F
Out[4]: 189.0
In [5]: T_in_C
Out[5]: 87.22222222222223

In [6]: type(T_in_F)
Out[6]: float
In [7]: type(T_in_C)
Out[7]: float
```

6. Changer juste la valeur de la variable pour T_F. Que vaut la valeur de la variable pour T_C après avoir changé la variable pour T_F ? Pourquoi ce résultat et comment faire pour le corriger ?

Correction. Python ne garde pas des formules en mémoire, mais effectue uniquement des opérations sur la mémoire. Il faut insister sur le fait que "=" est une affectation et non une formule mathématique.

```
In [8]: T_in_F = 100.0

In [9]: T_in_C
Out[9]: 87.22222222222223

In [10]: T_in_C = ( T_in_F - 32.0 ) * 5.0 / 9.0
In [11]: T_in_C
Out[11]: 37.77777777777778
```

2.3 Débogage sous Spyder des scripts python

2.3.1 Introduction

On utilisera ici l'éditeur des scripts python (fichiers *.py) de Spyder , qui est sur la fenêtre de gauche. Si la fenêtre de la console IPython est encore agrandie, utiliser le raccourci clavier Ctrl+Alt+Maj+M pour avoir l'ensemble des fenêtres de Spyder.

Contrairement aux humains qui peuvent comprendre une phrase contenant des fautes d'orthographe ou de grammaire, les ordinateurs ne sont pas dotés d'une telle intelligence et ont alors une tolérance nulle aux erreurs. Les instructions doivent être écrites sans faute, sinon un programme ne fonctionne pas. Par exemple les deux instructions du langage python:

```
write("Bonjour!")
print("Bonjour!")
```

ont l'air de vouloir dire la même chose pour un lecteur humain, mais pour un ordinateur il y a une grande différence.

7. Dans la console IPython de Spyder entrer ces deux commandes et commenter les résultats.

Il existe des vérificateur de la syntaxe des programmes, tel qu'ils sont implémentés dans les compilateurs ou dans les environnements de développement intégrés (IDE en anglais pour "integrated development environment"), comme dans Spyder.

8. Mettre les deux commandes d'en haut dans un script dans Spyder (un fichier *.py dans la fenêtre à gauche). On doit voir quelque chose comme dans la figure 8.

Même un programme sans faute de syntaxe ne fait pas obligatoirement ce qu'on veut, car il reste souvent encore des erreurs de logique ou de conception à corriger. Cette dernière correction, aussi nommé "débogage"

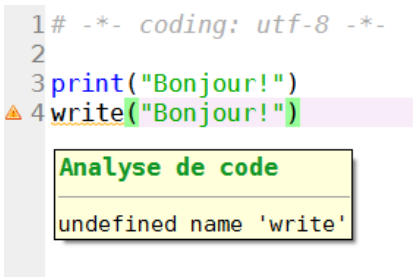

Figure 8: Erreur de syntaxe indiquée par Spyder directement dans l'éditeur. Si on positionne la souris sur le triangle jaune, le texte informatif dans le cadre apparaît.

(anglais "debugging" = trouver et enlever les "bugs"), demande souvent le plus du temps. Pour trouver les "bugs", on doit suivre l'exécution d'un programme pas à pas, soit en mettant des instructions d'affichage du contenu des variables après chaque ligne du programme, soit en utilisant un logiciel "debugger" ou "débogueur", tel qu'il est intégré dans une IDE, comme par exemple dans Spyder (voir section 1.10).

2.3.2 Exercices de débogage sur les boucles

Il existe deux sortes de boucles en python:

1. boucle **while** pour répéter un bloc d'instructions un nombre de fois qui n'est pas connu à l'avance
2. boucle **for** pour un des deux cas:
   (a) répéter un bloc d'instructions un nombre de fois qui est connu à l'avance
   (b) parcourir une liste ou autre objet itérable

9. Tester cet exemple avec **while**:

```
# -*- coding: utf-8 -*-
x0 = 257
ct = 0 # compteur (counter en anglais, d'où le nom "ct")
x = x0
while x > 1:
    x = x // 2 # division entière
    ct = ct + 1
print("Approximation de log2 de ", x0, " = ", ct)
```

10. Suivre l'exécution de ce script pas à pas avec le débogueur et l'explorateur de variables (voir section 1.10 et 1.6).

11. Comparer et expliquer les différences entre **for** et **while** dans cet exemple:

```python
# -*- coding: utf-8 -*-
for i in range(5):
    print(i)
print("après la boucle for, i=", i)

i = 0
while i<5:
    print(i)
    i = i + 1
print("après la boucle while, i=", i)
```

On voit que si on connait le nombre d'itérations à l'avance, la boucle **for** est plus simple à écrire et moins sujet aux erreurs qu'on peut faire avec une boucle **while**, où on peut oublier d'initialiser ou incrémenter la variable i.

12. Suivre l'exécution de ce script pas à pas avec le débogueur et l'explorateur de variables (voir section 1.10 et 1.6).

Pour parcourir les éléments d'une liste ou un autre objet itérable, on peut soit procéder avec un indice explicite, soit laisser **for** faire ce travail implicitement, ce qui est plus court et plus simple à écrire et lire:

```python
# -*- coding: utf-8 -*-
suite = ["a", "b", "c"]
print("première boucle:")
for i in range(len(suite)):
    print(suite[i])

print("deuxième boucle:")
#plus simple:
for x in suite:
    print(x)
```

13. Suivre l'exécution de ce script pas à pas avec le débogueur et l'explorateur de variables (voir section 1.10 et 1.6).