

# Tutorat de Programmation

## L1 Informatique

### 1 – Comprendre et Corriger les Erreurs

#### A) - Erreurs de Syntaxe

Pour commencer, utilisez votre terminal pour vous déplacer dans le dossier **tuto2/exemples** :

```
ouroumov@Danaan: ~/Desktop/tuto2/exemples
ouroumov@Danaan:~/Desktop$ cd tuto2/exemples/
ouroumov@Danaan:~/Desktop/tuto2/exemples$ ls
e1.py e2.py e3.py e4.py e5.py e6.py e7.py e8.py e9.py
ouroumov@Danaan:~/Desktop/tuto2/exemples$
```

Exécutez le premier script **e1.py**, vous obtenez :

```
ouroumov@Danaan: ~/Desktop/tuto2/exemples
ouroumov@Danaan:~/Desktop/tuto2/exemples$ ./e1.py
File "./e1.py", line 3
    print 3 +
            ^
SyntaxError: invalid syntax
ouroumov@Danaan:~/Desktop/tuto2/exemples$
```

L'interpréteur Python s'est arrêté brutalement et vous a fourni un message d'erreur. Quand vous tombez sur des erreurs, il est important de savoir comment les lire. Vous avez ici trois informations importantes :

- Le fichier dans lequel se trouve l'erreur
- La position dans le fichier où l'erreur a été détectée (Ligne 3, et le symbole « ^ » vous indique où dans la ligne)
- Le type de l'erreur. Ici : il s'agit d'une erreur de Syntaxe

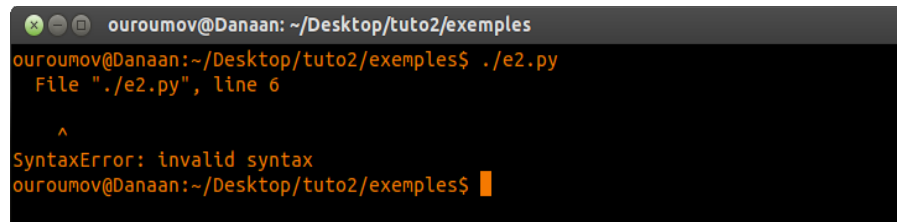
La Syntaxe d'un langage définit les combinaisons de symboles qui composent du code correct. Une erreur de Syntaxe indique qu'une combinaison illégale de symboles a été détectée. Si vous ouvrez le fichier pour regarder le code, vous pouvez tout de suite voir d'où vient l'erreur.

```
1 #! /usr/bin/env python
2
3 print 3 +
```

À la ligne trois, il manque une opérande pour l'opération « addition ». Une telle opération doit toujours avoir deux opérandes, et sur cette ligne se trouve donc une combinaison de symboles qui ne constitue pas du code Python valide.

Les erreurs de Syntaxe sont celles que vous verrez le plus souvent.

Exécutez maintenant le fichier e2.py



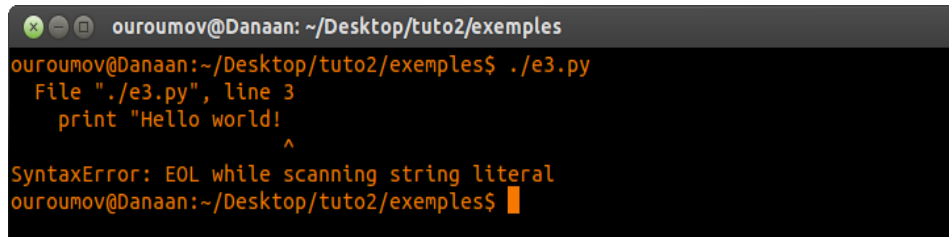
```
ouroumov@Danaan: ~/Desktop/tuto2/exemples
ouroumov@Danaan:~/Desktop/tuto2/exemples$ ./e2.py
File "./e2.py", line 6
    ^
SyntaxError: invalid syntax
ouroumov@Danaan:~/Desktop/tuto2/exemples$
```

Cet exemple est moins facile à comprendre. La flèche de l'interpréteur pointe sur du blanc et la ligne a changée. Observez le code ayant produit cette erreur :

```
1  #!/usr/bin/env python
2
3  print (3 +
4
5
```

Il n'y a pas de ligne 6. Comme indiqué plus haut, Python ne vous indique pas toujours où l'erreur se trouve (il ne peut pas faire ça), uniquement où l'erreur a été détectée. Ici, en essayant de lire la ligne 6, il trouve la fin du fichier. C'est la parenthèse ouvrante qui provoque ce comportement. Il est possible en Python d'écrire des expressions sur plusieurs lignes implicitement quand on utilise des parenthèse pour grouper les éléments de ces expressions.)

Python vous fournit parfois des indications supplémentaires :



```
ouroumov@Danaan: ~/Desktop/tuto2/exemples
ouroumov@Danaan:~/Desktop/tuto2/exemples$ ./e3.py
File "./e3.py", line 3
    print "Hello world!
    ^
SyntaxError: EOL while scanning string literal
ouroumov@Danaan:~/Desktop/tuto2/exemples$
```

Ici, il vous indique qu'il s'agit toujours d'une erreur de Syntaxe, mais d'un type particulier.

```
1  #!/usr/bin/env python
2
3  print "Hello world!
4
```

« EOL while scanning string literal » veut dire que l'interpréteur a trouvé la fin d'une ligne (End Of Line) alors qu'il essayait de lire une chaîne de caractères : c'est parce que qu'il manque un « " » à la fin du « Hello World ! »

Python supporte des chaînes sur plusieurs lignes : elles commencent par trois symboles « `"""` » ou « `'''` » et se terminent par les mêmes trois symboles.

## B) – Erreurs Sémantiques / Runtime Errors

Exécutez le script **e4.py** :

```
ouroumov@Danaan: ~/Desktop/tuto2/exemples
ouroumov@Danaan:~/Desktop/tuto2/exemples$ ./e4.py
Hello world!
Traceback (most recent call last):
  File "./e4.py", line 5, in <module>
    a = 4 / 0
ZeroDivisionError: integer division or modulo by zero
ouroumov@Danaan:~/Desktop/tuto2/exemples$
```

Il s'agit ici d'une classe d'erreurs différente : les erreurs de sens.

```
1 #!/usr/bin/env python
2
3 print "Hello world!"
4
5 a = 4 / 0
6
```

À la différence des erreurs syntaxiques, ces erreurs ne sont détectées que quand le programme est exécuté. C'est la raison pour laquelle le « `print "Hello world!"` » de la ligne 3 s'exécute correctement avant que le programme ne crashes.

L'erreur en question ici est une division par zéro. Notez que la ligne 5 est du code Python parfaitement valide sur le plan Syntaxique : on assigne à une variable le résultat d'une expression mathématique à deux opérandes. En enlevant le zéro, vous auriez introduit une erreur de Syntaxe et le programme n'aurait même pas commencé à s'exécuter. L'interpréteur Python ne commence à exécuter un programme qu'après en avoir fait une première lecture pour vérifier qu'il ne comporte pas d'erreurs de syntaxe.

Exécutez maintenant **e5.py** :

```
ouroumov@Danaan: ~/Desktop/tuto2/exemples
ouroumov@Danaan:~/Desktop/tuto2/exemples$ ./e5.py
1 2 3
Traceback (most recent call last):
  File "./e5.py", line 8, in <module>
    display([1, 2, 3])
  File "./e5.py", line 5, in display
    print l[i],
IndexError: list index out of range
ouroumov@Danaan:~/Desktop/tuto2/exemples$
```

Il s'agit ici d'un autre type d'erreur sémantique.

```
1 #!/usr/bin/env python
2
3 def display(l):
4     for i in range(len(l) + 1):
5         print l[i],
6
7
8 display([1, 2, 3])
```

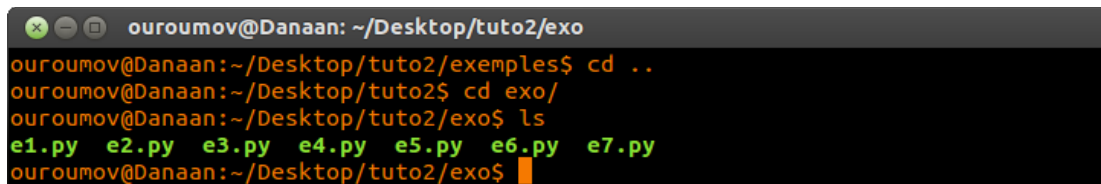
La fonction « **display**(1) » a une erreur dans son implémentation. À la ligne 4 le « + 1 » est de trop et à cause de ça un tours de trop dans la boucle **for** se produit.

Quand cette fonction est appelée en lui passant comme argument la liste « [1, 2, 3] » la valeur « len(1) + 1 » vaut 4 et la fonction « range(4) » produit la liste des indices : « [0, 1, 2, 3] ».

Quand Python essaye alors d'accéder à la valeur « l[3] » (soit la 4ème valeur dans la liste) à la ligne 5, une erreur se produit parce que la liste n'a que trois éléments.

## C) – Exercices

Déplacez vous avec votre terminal dans le dossier **exos** :



```
ouroumov@Danaan: ~/Desktop/tuto2/exo
ouroumov@Danaan:~/Desktop/tuto2/exemples$ cd ..
ouroumov@Danaan:~/Desktop/tuto2$ cd exo/
ouroumov@Danaan:~/Desktop/tuto2/exo$ ls
e1.py e2.py e3.py e4.py e5.py e6.py e7.py
ouroumov@Danaan:~/Desktop/tuto2/exo$
```

Attention, avant de commencer les exercices, configurez votre éditeur de texte pour utiliser 4 espaces par niveau d'indentation, et pas des tabulations.

Voir comment faire sur le forum du cours Algo & Prog :

<https://e-uapv2015.univ-avignon.fr/mod/forum/discuss.php?d=853>

Exécutez chacun des programmes, et corrigez les erreurs qui surviennent.

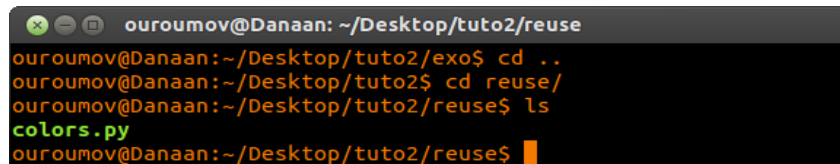
Vous allez voir des erreurs qui n'ont pas été expliquées précédemment. Quand cela se produit, utilisez vos compétences parfaites en maîtrise de la langue Anglaise pour déchiffrer ce que Python vous dit.

## 2 – Retour vers le futur les fonctions

Revenons sur la différence « `return` » Vs « `print` ».

- « `print` » ne doit être utilisé dans le corps d'une fonction que si le but explicite de la fonction est d'afficher quelque chose.
- « `return` » est utilisé dès que le code qui appelle une fonction a besoin de stocker son résultat pour l'utiliser plus tard.
- Si vous écrivez une fonction sans inclure un « `return` », alors la valeur par défaut de retours de la fonction est toujours `None`.

Dans le terminal, déplacez vous dans le dossier « reuse » :

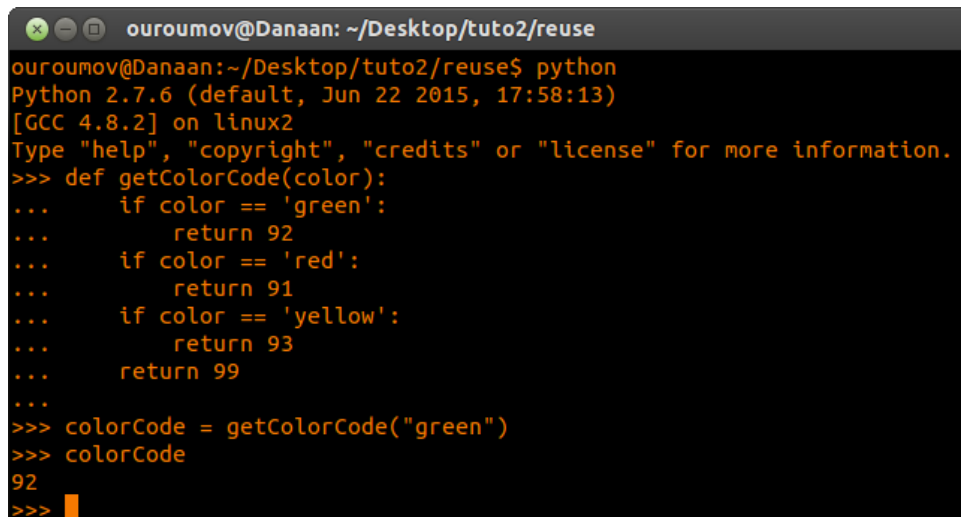


```
ouroumov@Danaan: ~/Desktop/tuto2/reuse
ouroumov@Danaan:~/Desktop/tuto2/exo$ cd ..
ouroumov@Danaan:~/Desktop/tuto2$ cd reuse/
ouroumov@Danaan:~/Desktop/tuto2/reuse$ ls
colors.py
ouroumov@Danaan:~/Desktop/tuto2/reuse$
```

Ouvrez le fichier **colors.py** et regardez la première fonction :

```
1  #! /usr/bin/env python
2
3  def getColorCode(color):
4      if color == 'green':
5          return 92
6      if color == 'red':
7          return 91
8      if color == 'yellow':
9          return 93
10     return 99
```

Cette fonction renvoie une valeur. Dans le terminal, lancez un shell Python et saisissez cette fonction. (Vous pouvez coller du texte dans le terminal en utilisant la combinaison « CTRL+ MAJ + V ») Puis, appelez la fonction et stockez sa valeur de retours dans une variable.



```
ouroumov@Danaan: ~/Desktop/tuto2/reuse
ouroumov@Danaan:~/Desktop/tuto2/reuse$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> def getColorCode(color):
...     if color == 'green':
...         return 92
...     if color == 'red':
...         return 91
...     if color == 'yellow':
...         return 93
...     return 99
...
>>> colorCode = getColorCode("green")
>>> colorCode
92
>>>
```

Vous pouvez voir que la variable `colorCode` reçoit bien la valeur correspondante à la couleur « verte » suite à l'appel de la fonction. C'est exactement ce qu'il se passe dans la seconde fonction définie dans le fichier :

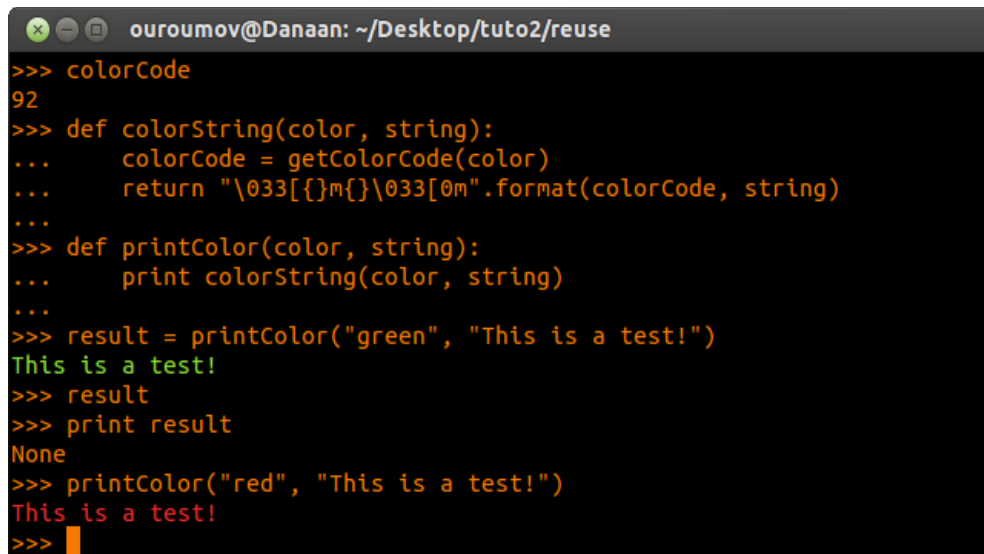
```
14 def colorString(color, string):
15     colorCode = getColorCode(color)
16     return "\033[{}m{}\033[0m".format(colorCode, string)
```

À la ligne 15, si on avait utilisé des « `print` » au lieu de « `return` » dans le corps de la fonction « `getColorCode` », `colorCode` aurait reçu la valeur « `None` » et le code n'aurait pas pu être correctement utilisé dans la suite de la fonction.

Dans la troisième fonction, on affiche enfin quelque chose :

```
20 def printColor(color, string):
21     print colorString(color, string)
```

Copiez les fonctions 2 et 3 dans le shell Python, et utilisez la fonction « `printColor` » :

A screenshot of a terminal window titled 'ouroumov@Danaan: ~/Desktop/tuto2/reuse'. The terminal shows a Python interactive session. It starts with a prompt '>>>' followed by 'colorCode', which returns '92'. Then, the function 'colorString' is defined with two parameters: 'color' and 'string'. Inside the function, 'getColorCode(color)' is called, and its result is used in a string formatting operation: '\033[{}m{}\033[0m'.format(colorCode, string). Next, the function 'printColor' is defined with two parameters: 'color' and 'string'. Inside, it calls 'colorString(color, string)'. Then, 'result = printColor("green", "This is a test!")' is executed, which prints 'This is a test!' in green. After that, 'result' is printed, showing 'None'. Finally, 'printColor("red", "This is a test!")' is executed, which prints 'This is a test!' in red. The prompt '>>>' is shown at the end of the last line.

Comme vous pouvez le voir, si on essaye de stocker le résultat de l'appel de la fonction dans une variable, on obtient bien « `None` », malgré le fait que la chaîne de caractères soit affichée sur l'écran.

Le principe d'écrire des fonctions pour pouvoir réutiliser leur code et s'en servir pour résoudre des problèmes en les découpant en petites parties est un des fondements de la programmation. Ce principe est souvent désigné sous l'intitulé « **Code Reuse** »

### 3 – Modules

La librairie standard du langage Python est découpée en modules. Vous en avez déjà utilisé quelques uns, comme le module « **math** », ou le module « **random** ».

Ces modules regroupent des fonctions autour d'un thème commun. Par exemple le module **math** fournit les fonctions mathématiques les plus utiles qui ne font pas partie des fonctions de base du langage. Le module **random** fournit des fonctions permettant d'introduire de l'aléatoire dans votre code.

Les documentations pour ces modules sont disponibles ici :

**math** : <https://docs.python.org/2/library/math.html>

**random** : <https://docs.python.org/2/library/random.html>

Lorsque vous allez commencer à écrire beaucoup de code pour faire des programmes qui servent vraiment à quelque chose, vous allez vous aussi écrire vos propres modules. Une telle division du code est importante pour la maintenance, la clarté, la documentation.

En fait, vous avez déjà un module custom sous la main : le module **colors**.

Si vous positionnez votre terminal dans le même répertoire que le fichier **colors.py** vous pouvez le constater après avoir lancé un shell Python :

```
ouroumov@Danaan: ~/Desktop/tuto2/reuse
ouroumov@Danaan:~/Desktop/tuto2/reuse$ ls
colors.py
ouroumov@Danaan:~/Desktop/tuto2/reuse$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from colors import *
>>> printColor("yellow", "This works!")
This works!
>>>
```

**Exercice :**

Créez dans le même dossier un programme **main.py** qui utilise ce module pour afficher ça :

```
ouroumov@Danaan: ~/Desktop/tuto2/reuse
ouroumov@Danaan:~/Desktop/tuto2/reuse$ chmod +x main.py
ouroumov@Danaan:~/Desktop/tuto2/reuse$ ./main.py
this
is
cool
ouroumov@Danaan:~/Desktop/tuto2/reuse$
```

Puis, ajoutez une fonction dans le module **colors** : cette fonction doit accepter trois arguments de type chaîne de caractères (str) et afficher les trois chaînes sur la même ligne, chaque chaîne d'une couleur différente.