Portee_variables_scalaires

February 12, 2020

1 Portées des variables

Les variables en python sont créées au moment de leur première affectation.

Variables locales ou globales :

1. Une variable *locale* (à une fonction) est créée lorsque la fonction s'exécute mais n'est pas conservée une fois la fonction terminée.
2. Une variable est *globale* (à un module/fichier) lorsqu'elle est définie en dehors d'une fonction. Une fois définie (par sa première affectation), elle reste active en mémoire jusqu'à la fin de l'exécution du module/fichier.

Une variable définie dans une fonction est *locale* par défaut. Une variable seulement référencée dans une fonction est *globale* par défaut.

La fonction `nombre_a()` utilise deux variables locales: `c` et le paramètre formel `s`. A chaque exécution de la fonction, une variable `c` est créée au moment de l'initialisation (de même qu'une variable `s` est créée et reçoit la valeur du paramètre avec lequel la fonction est appelée).

[1]:
```
def nombre_a(s):
    c = 0
    for lettre in s:
        if lettre == 'a':
            c += 1
    print("Il y a %d occurrences de la lettre %s dans '%s'" % (c,'a',s))
```

Bien que la fonction `nombre_a()` utilise une variable `c` (et la modifie), la variable globale n'est pas affectée par cette modification.

[2]:
```
c = 80
nombre_a("maharadjah")
print(c)
```

```
Il y a 4 occurrences de la lettre a dans 'maharadjah'
80
```

Si une variable est référencée (ie utilisée) mais non modifiée, elle est considérée comme globale. La variable `lettre_glob` est dans ce cas dans la fonction suivante.

[3]:
```
def nombre_var(s):
    c = 0
```

```
    for lettre in s:
        if lettre == lettre_glob:
            c += 1
    print("Il y a %d occurrences de la lettre %s dans '%s'" % (c,lettre_glob,s))
```

Si la variable n'est pas déjà connue, son emploi déclenche une erreur dans la fonction:

[4]:
```
nombre_var("maharadjah")
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-4-c47a7ca57c2e> in <module>
----> 1 nombre_var("maharadjah")


<ipython-input-3-128fb1781c3f> in nombre_var(s)
      2     c = 0
      3     for lettre in s:
----> 4         if lettre == lettre_glob:
      5             c += 1
      6     print("Il y a %d occurrences de la lettre %s dans '%s'" %␣
↪(c,lettre_glob,s))


NameError: name 'lettre_glob' is not defined
```

C'est la valeur affectée "à l'extérieur de la fonction" qui est utilisée:

[5]:
```
lettre_glob = 'h'
nombre_var("maharadjah")
```

```
Il y a 2 occurrences de la lettre h dans 'maharadjah'
```

Le mot-clé `global` permet de changer le comportement par défaut. Pour éviter la création automatique d'une nouvelle variable dans une fonction, on peut déclarer cette variable comme globale.

[6]:
```
def nombre_a_cglob(s):
    global c
    c = 0
    for lettre in s:
        if lettre == 'a':
            c += 1
    print("Il y a %d occurrences de la lettre %s dans '%s'" % (c,'a',s))
```

Dans cette nouvelle version de la fonction, la variable globale c est utilisée comme compteur (clairement une mauvaise idée!). Donc la valeur définie à l'extérieur (80 dans l'exemple) est perdue:

[7]:
```
c = 80
nombre_a_cglob("maharadjah")
print(c)
```

```
Il y a 4 occurrences de la lettre a dans 'maharadjah'
4
```

On peut aussi utiliser `global` pour la cas d'une variable qu'on ne fait que "référencer" mais ce n'est pas considéré comme pertinent car ça ne change rien au comportement par défaut. Le seul avantage potentiel pourrait concerner des situations où l'on veut rendre particulièrement visible le fait que la variable utilisée doit exister préalablement.

[8]:
```
def nombre_var_glob(s):
    global lettre_glob
    c = 0
    for lettre in s:
        if lettre == lettre_glob:
            c += 1
    print("Il y a %d occurrences de la lettre %s dans '%s'" % (c,lettre_glob,s))
```

[9]:
```
lettre_glob = 'r'
nombre_var("maharadjah")
```

```
Il y a 1 occurrences de la lettre r dans 'maharadjah'
```

On peut aussi utiliser `global` à l'intérieur d'une fonction pour rendre globale une variable définie dans la fonction. Dans l'exemple suivant, on crée une variable `lettres` qui va exister de manière globale après l'exécution de la fonction. (*Attention pour le test: une fois que vous avez essayé une fois la fonction, la variable est créée. Il faut donc stopper le noyau et récommencer pour produire l'effet ci-dessous*)

[10]:
```
# création d'une var glob dans une fonction
def nombre_var_cre_glob(s):
    global lettres
    lettres = ""
    c = 0
    for lettre in s:
        lettres += lettre
        if lettre == lettre_glob:
            c += 1
    print("Il y a %d occurrences de la lettre %s dans '%s'" % (c,lettre_glob,s))
```

[11]:
```
print(lettres)
```

```
␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-11-f898038084ad> in <module>
   ----> 1 print(lettres)


       NameError: name 'lettres' is not defined
```

[12]:
```
lettre_glob = 'r'
nombre_var_cre_glob("maharadjah")
print(lettres)
```

```
Il y a 1 occurrences de la lettre r dans 'maharadjah'
maharadjah
```