W4-S6-C2-exception-unboundlocalerror

December 14, 2014

1 L'exception `UnboundLocalError`

1.1 Complément - niveau intermédiaire

Nous résumons ici quelques cas simples de portée de variables

1.1.1 Variable locale

Les **arguments** attendus par la fonction sont considérés comme des variables **locales**.

Pour définir une autre variable locale, il suffit de la définir (l'affecter), elle devient alors accessible en lecture

```
In []: def ma_fonction():
           variable1 = "locale"
           print variable1
       ma_fonction()
```

et ceci que l'on ait ou non une variable globale de même nom

```
In []: variable2 = "globale"

       def ma_fonction():
           variable2 = "locale"
           print variable2
       ma_fonction()
```

1.1.2 Variable globale

On peut accéder **en lecture** à une variable globale sans précaution particulière

```
In []: variable3 = "globale"
       def ma_fonction():
           print variable3
       ma_fonction()
```

1.1.3 Mais il faut choisir !

Par contre on ne **peut pas** faire la chose suivante dans une fonction. On ne peut pas utiliser **d'abord** une variable comme une variable **globale**, **puis** essayer de l'affecter localement - ce qui signifie la déclarer comme une **locale**:

```
In []: # cet exemple ne fonctionne pas et leve UnboundLocalError
       variable4 = "globale"
       def ma_fonction():
           # on référence la variable globale
```

```
        print variable4
        # et maintenant on crée un variable locale
        variable4 = "locale"

    # on "attrape" l'exception
    try:
        ma_fonction()
    except Exception as e:
        print type(e)
        print e
```

1.1.4 Comment faire alors ?

L'intérêt de cette erreur est d'interdire de mélanger des variables locales et globales de même nom dans une même fonction. On comprend bien que ça serait vite incompréhensible. Donc une variable dans une fonction peut être soit locale si elle est affectée dans la fonction ou sinon globale, mais jamais les deux à la fois. Si vous avez une erreur `UnboundLocalError` c'est forcément que vous avez fait cette confusion.

Vous vous demandez peut-être à ce stade, mais comment fait-on alors pour modifier une variable globale depuis une fonction ?

Pour cela il faut utiliser l'instruction `global` comme ceci

```
In []: # Pour résoudre ce conflit il faut explicitement
       # déclarer la variable  comme globale
       variable5 = "globale"
       def ma_fonction():
           global variable5
           # on référence la variable globale
           print "dans la fonction", variable5
           # et maintenant on crée un variable locale
           variable5 = "changée localement"

       ma_fonction()
       print "après la fonction", variable5
```

Nous reviendrons longuement sur l'instruction `global` dans la prochaine vidéo.

1.1.5 Bonnes pratiques

Cela étant dit, l'utilisation de variables globales est généralement considérée comme une mauvaise pratique.

Le fait d'utiliser une variable globale en *lecture seule* peut rester acceptable, lorsqu'il s'agit de matérialiser une constante qu'il est facile de changer. Mais dans une application aboutie, ces constantes elles-mêmes peuvent être modifiées par l'utilisateur via un système de configuration, donc on préférera passer en argument un objet *config*.

Et dans les cas où votre code doit recourir à l'utilisation de l'instruction `global`, c'est très probablement que quelque chose peut être amélioré au niveau de la conception de votre code.

Il est recommandé, au contraire, de passer en argument à une fonction tout le contexte dont elle a besoin pour travailler; et à l'inverse d'utiliser le résultat d'une fonction plutôt que de modifier une variable globale.