# Les fonctions `globals` et `locals`

## Complément - niveau intermédiaire

### Un exemple

python fournit un accès à la liste des noms et valeurs des variables visibles à cet endroit du code. Dans le jargon des langages de programmation on appelle ceci **l'environnement**.

Cela est fait grâce aux fonctions *builtin* `globals` et `locals`, que nous allons commencer par essayer sur quelques exemples. Nous avons pour cela écrit un module dédié:

```
import env_locals_globals
```

---

Dont voici le code

```
from modtools import show_module
show_module(env_locals_globals)
```

---

et voici ce qu'on obtient lorsqu'on appelle

```
env_locals_globals.temoin(10)
```

### Interprétation

Que nous montre cet exemple ?

D'une part la fonction **`globals`** nous donne la liste des symboles définis au niveau de **l'espace de noms du module**. Il s'agit évidemment du module dans lequel est définie la fonction, pas celui dans lequel elle est appelée. Vous remarquerez que ceci englobe **tous** les symboles du modules et non pas seulement ceux définis avant `temoin`, c'est-à-dire la variable `globale`, les deux fonctions, et la classe `Foo`.

D'autre part **`locals`** nous donne les variables locales qui sont accessibles **à cet endroit du code**, comme le montre ce second exemple qui se concentre sur `locals` à différents points d'une même fonction.

```
import env_locals

# le code de ce module
show_module(env_locals)
```

```
env_locals.temoin(10)
```

## Usage pour le formatage de chaînes

Ces deux fonctions ne sont pas d'une utilisation très fréquente. Elles peuvent cependant être utiles dans le contexte du formatage de chaînes, comme ou peut le voir dans les deux exemples ci-dessous.

#### Avec `format`

On peut utiliser `format` qui s'attend à quelque chose comme

```
"{nom}".format(nom="Dupont")
```

que l'on peut obtenir de manière équivalente, avec le passage d'arguments en **, comme on l'a vu en fin de semaine passée:

```
"{nom}".format(**{'nom': 'Dupont'})
```

En versant la fonction `locals` dans cette formule on obtient une forme relativement élégante

```
def format_et_locals(nom, prenom, civilite, telephone):
    return "{civilite} {prenom} {nom} : Poste {telephone}".format(**locals())

format_et_locals('Dupont', 'Jean', 'Mr', '7748')
```

#### Avec l'opérateur `%`

De manière similaire, avec l'opérateur `%` - dont nous rappelons qu'il est obsolète - on peut écrire

```
def pourcent_et_locals(nom, prenom, civilite, telephone):
    return "%(civilite)s %(prenom)s %(nom)s : Poste %(telephone)s"%locals()

pourcent_et_locals('Dupont', 'Jean', 'Mr', '7748')
```