

Représentation mémoire

Skander Zannad et Judicaël Courant

2013-10-06

1 Modèle mémoire

Représentation des données en mémoire :

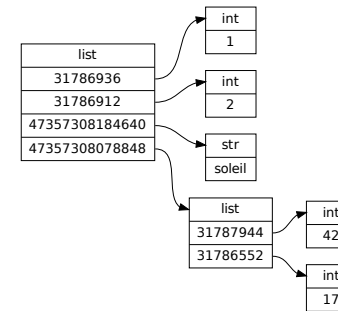
- Relativement complexe.
- Mais on peut travailler sur un modèle relativement simple.

Notre modèle :

- Python manipule des données qu'on appelle des *objets*.
- Un objet est une suite de cases mémoires contiguës contenant
 1. Le type de l'objet.
 2. Des données propres à l'objet.
 3. Des liens vers d'autres objets (adresses des autres objets).

Exemple :

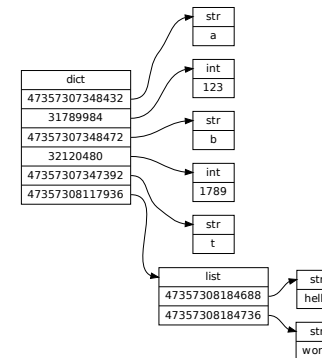
[1, 2, 'soleil', [42, 17]]



2 Variables globales

Quand on définit des variables (globales), celles-ci sont stockées dans une table appelée *dictionnaire* associant à chaque nom de variable sa valeur :

[1, 2, 'soleil', [42, 17]]



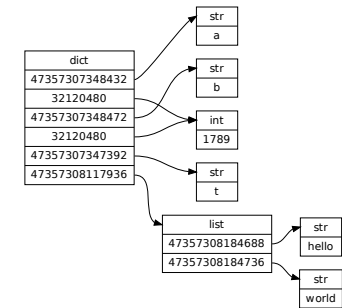
3 Affectation

Modifions a :

[1, 2, 'soleil', [42, 17]]

Que s'est-il passé ?

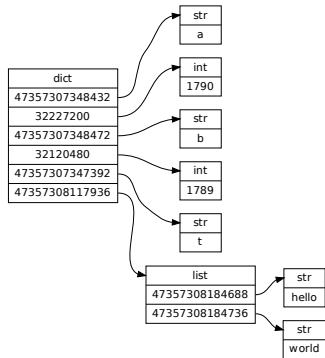
1. On a évalué l'expression b. Le résultat est une adresse mémoire, celle de l'entier 1789.
2. Ce résultat est mis dans a.



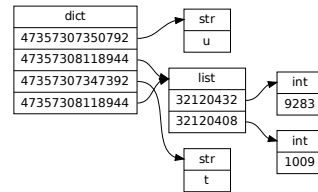
4 Expressions

[1, 2, 'soleil', [42, 17]]

Pour évaluer $1789 + 1$, Python construit un nouvel objet, y met le résultat et retourne son adresse. *L'évaluation d'une expression peut créer de nouveaux objets.*

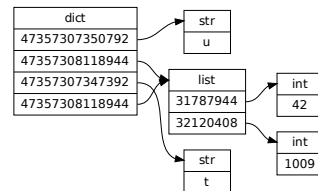


En python, elle peut se tester avec le mot-clé `is` («`t is u`» vaut ici `True`).



Cela peut avoir des conséquences inattendues :

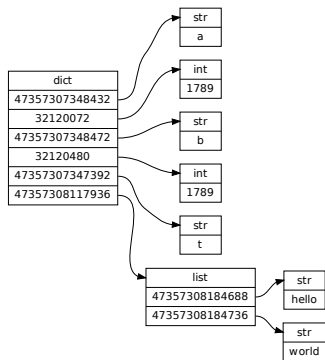
[1, 2, 'soleil', [42, 17]]



Remarque : Python n'est pas très malin

[1, 2, 'soleil', [42, 17]]

Python a construit un nouvel objet pour évaluer `1789 + 0...`



Attention : deux objets peuvent avoir le même contenu mais ne pas être physiquement égaux.

[1, 2, 'soleil', [42, 17]]

On parlera d'égalité *structurelle*. En python, elle se teste avec le symbole `==` («`t == u`» vaut ici `True` alors que «`t is u`» vaut `False`)

5 Aliasing

Deux variables peuvent représenter le même objet :

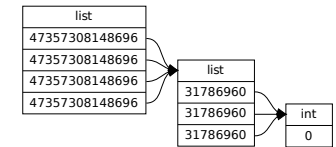
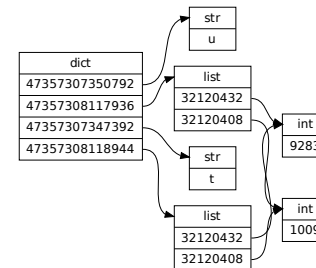
[1, 2, 'soleil', [42, 17]]

On parlera d'égalité *physique* pour dire que deux objets ont la même adresse mémoire.

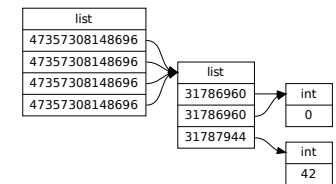
6 Copier une liste

Le *slicing* (tranchage) se fait par copie. C'est un moyen simple de copier une liste dans une autre :

[1, 2, 'soleil', [42, 17]]

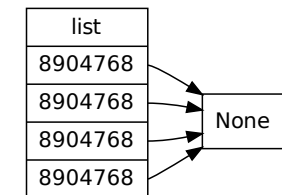


[1, 2, 'soleil', [42, 17]]



Pour construire A correctement, on crée une liste de 4 objets bidons :

[1, 2, 'soleil', [42, 17]]



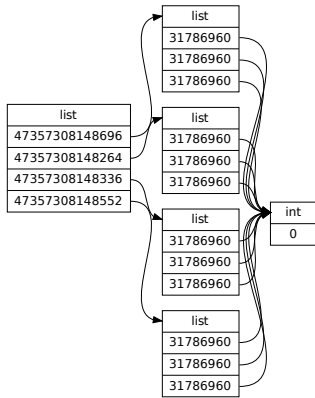
NB : couper une tranche de taille n demande la copie de n adresses. Temps : $\Theta(n)$.

Puis on remplace chacun par une nouvelle rangée de 0.

[1, 2, 'soleil', [42, 17]]

7 Un problème d'aliasing

[1, 2, 'soleil', [42, 17]]



8 Appel de fonctions

Selon les langages de programmation, les valeurs des objets peuvent être :

- Les données contenues dans les objets manipulés (cas de Scilab)
- Les *adresses en mémoire* de ces objets (cas de Python)

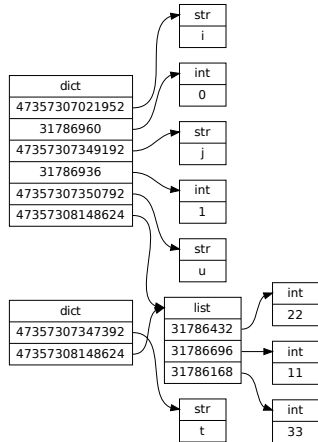
Appel de $f(t)$, où t tableau à n éléments :

- En Scilab, *copie* des n éléments : temps $\Theta(n)$. Impossibilité pour f de modifier le contenu de t .
- En Python, passage de l'*adresse mémoire* de t à f : temps $\Theta(1)$. Possibilité pour f de modifier le contenu de t .

Exemple :

[1, 2, 'soleil', [42, 17]]

Lors de l'exécution de swap, deux dictionnaires de variables : locales et globales.



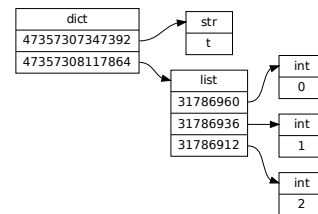
[0, 1, [3, 4, [...]]]

10 Ramasse-miettes

Pour exécuter

[1, 2, 'soleil', [42, 17]]

Python prend de la place mémoire pour représenter la liste :



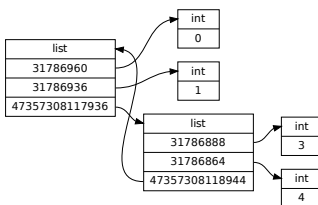
Si on affecte la variable avec une autre valeur :

[1, 2, 'soleil', [42, 17]]

9 Graphe des données

Lors de l'exécution du programme, les données peuvent former un graphe arbitrairement complexe, il peut même y avoir des cycles.

[1, 2, 'soleil', [42, 17]]



NB : `print(u)` donne

Récupération de la place inutilement utilisée :

- Gérée par un mécanisme interne à l'implantation de Python.
- Appelé *Garbage Collector* (GC), en français *Glaneur de Cellules* ou *Ramasse-miettes*.
- Le déclenchement du GC est non spécifié : il peut aussi bien se déclencher dès qu'un objet est inaccessible qu'attendre le moment où la mémoire est pleine et où il faut faire de la place pour de nouveaux objets.