J.Courtin *PC* LVH Caen*

INFORMATIQUE

REVISIONS

Types de données
Opérations et Expressions
Boucles

TD : Etude de la densité de probabilité de l'orbitale 1s de l'atome Hydrogène

- **Etude de fonction**
- **Recherche de maximum**
- **Calcul d'intégrale**

Les types de données primitives en Python

— opérations de base —

Scalaire : int, float, complex — bool

Séquentielle : **Non modifiable :** str, tuple, range **modifiable** : list

Ensemble : set & dict

Fonction : function

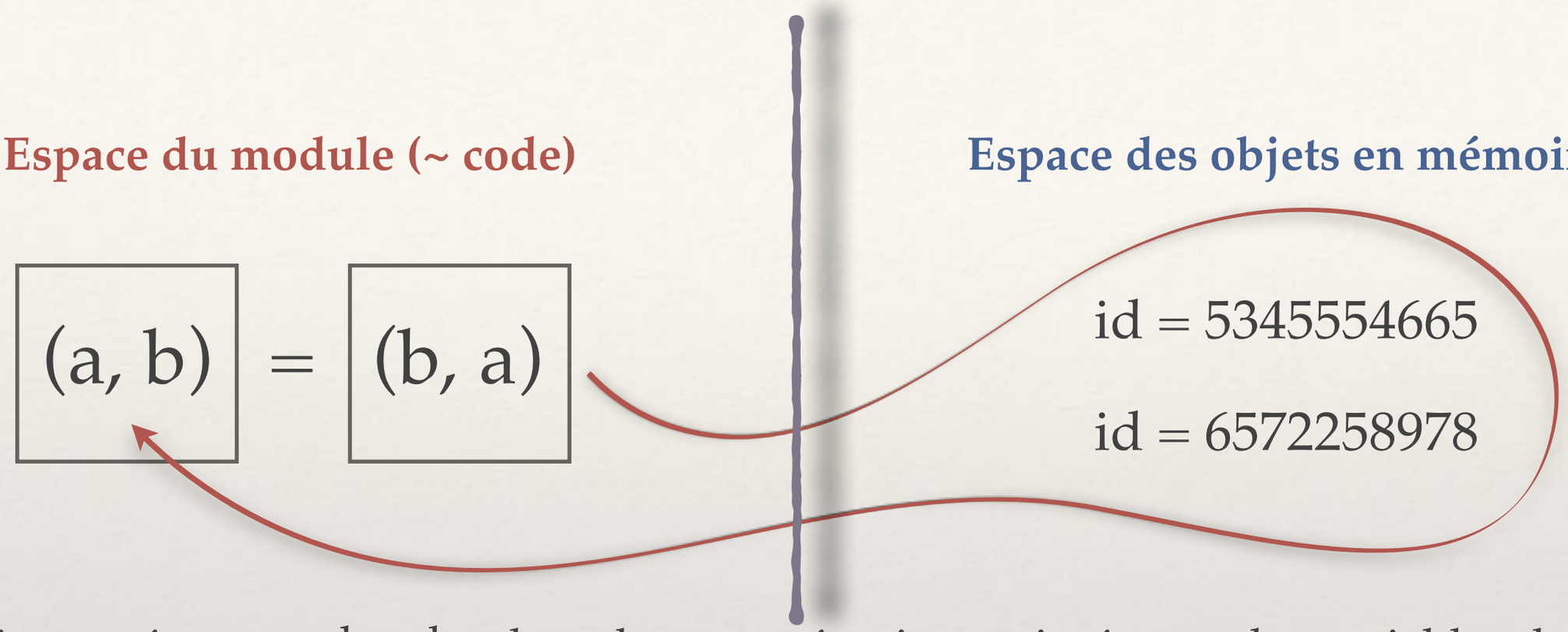
et bien d'autres ….

Le typage est dynamique :

- La même variable peut pointer vers tout type d'objet au cours d'un programme

Comment comprendre l'affectation en Python

Variable et objet deux choses bien distinctes



1 - L'interpréteur va chercher les adresses mémoires pointées par les variables de (b, a)
il crée un objet provisoire de même structure : ici tuple de taille 2. [Rq: il n'a pas de id]

2 - L'interpréteur affecte les adresses mémoires collectées dans le membre de droite
dans le tuple (a,b) ré-aiguillant ainsi les adresses mémoires pointées par a et b.

Le membre de gauche n'est ré-aiguillé que si sa structure est compatible avec la structure mémoire de l'objet donné dans le membre droit

Tout passe par référence : à aucun moment l'interpréteur ne connait les valeurs de a et b

Ces principes se retrouvent partout, et parfois de façon moins explicite

```
>>> for (x,y) in [(1,2), (2,2), (3,2)]:
...     print(x, " et ", y)
...
...
```

```
1 et 2
2 et 2
3 et 2
```

```
>>> a=b=2
puis
>>> a=b==2
```

```
>>> a,b
????
>>> a,b
????
```

<u>On les retrouve souvent :</u>

- **dans les blocs d'itération**
- **dans les blocs conditionnels**
- **ou dans les simples opérations conditionnelles**

Les opérations et affectations se font de la droite vers la gauche

Une expression (à droite) qui est affectée (à gauche), devient ainsi …
une expression (à droite) qui est affectée (à gauche), devenant ainsi …
une expression etc ….

```
>>> A=B=C=True
>>> (a,b)=(1,2)
```

```
>>> A == B != C == a<b
???
>>> A,B,C
???
```

<u>Important :</u>
Refaire les exemples dans une console python.

<u>Rq :</u>
L'affectation ou l'opération ne peut se faire qu'une fois que l'expression a été complètement évaluée et l'objet résultat créé.

Du point de vue algorithmique il s'agit d'**une structure de pile**

<u>exemple précédent :</u>

exemple :

```
>>> a,b,c=1,2,3
>>> a=b=c
```

```
>>> a,b,c
(3, 3, 3)      #et pas (1,1,1)
```

Ceci prouve le sens d'affectation : de la droite vers la gauche !

Les boucles en Python

Comment choisir la bonne structure itérative ?

Je connais le nombre d'itération ?

OUI → **FOR IN**

NON → **WHILE**

Structure :
while (condition):

J'ai besoin des indices ?

OUI →

Structure : **i est un entier**
for i in range(N):

NON →

Structure : **elem est un objet du séquentiel**
for elem in sequentiel:

Exemples de syntaxes « for in » : for itérateur in itérable

En python toutes les boucles for se font sur séquentiels,
pas sur des entiers (C, Fortran, Java)

```
for i in range(10)
```

« faux ami » →

engendre la liste itérable [0,1,2,3,4,5,6,7,8,9]
(c'est un cas particulier)

```
for  (x,y)  in  [(1,2), (2,2), (3,2)]:
```

```
for nom in {'Einstein': 'Albert', 'Newton': 'Isaac', 'Gauss': 'Carl-Friedriech', 'Tesla': 'Nicolas'}
```

```
for key in dictionnaire.keys()
```

→ méthode qui engendre la **[liste des clefs]**

Rq : Privilégier des noms explicites

```
for char in ''maChaineDeCaractères''
```

```
for tupNoeud in noeudsAViseter
```

→ Chaque noeud est un objet tuple

Quelques remarques :

Rq 1 : **Ne jamais changer la valeur de l'itérateur dans une boucle for !**

[ça ne changerais que pour l'itération en cours mais c'est une très mauvaise pratique]

L'itérateur et surtout l'itérable doivent être anticipés dès la conception du code :

Rq 2 : Toujours se demander au sujet de l'itérateur :
« Quel est l'objet que j'ai dans la main ?» —> quelle information ?

Rq 3 : Toujours se demander au sujet de l'étirable :
« Quel objet est-ce que je voudrais avoir ? »
—> c-à-d anticiper : de quelle information ai-je besoin ?

Rq 4 : La structure **for in** est beaucoup utilisée avec les **listes par compréhension** :

```
[k**2 for k in range(1,6)]   ————————>   engendre  [1, 4, 9, 16, 25]
```

```
for monCarré in [k**2 for k in range(1,6)]
```

Modification de l'itérateur

```
>>> for i in range(10):
...     print(i, end=" ")
...     i=7  #porte bonheur…
...     print(i)
...
...
0 7
1 7
2 7
3 7
4 7
5 7
6 7
7 7
8 7
9 7
```

Don't U do that !

TD : Etude d'une densité de probabilité

On se donne la fonction d'onde 1s de l'atome d'hydrogène : $\varphi_{1s} = \kappa_0 \, e^{-\frac{r}{a_0}}$

On veut tracer la densité de probabilité de présence de l'électron en fonction du rayon sur [0, 5.a_0] et trouver les coordonnées de son maximum.

Pour cela il faut déterminer la fonction densité de probabilité et réaliser une **intégration numérique** en python pour calculer le facteur de normalisation ρ_0.

On rappelle que : $dP = \left|\varphi_{1s}\right|^2 d\tau = \rho(r)dr =$

<u>Attention :</u>

Le problème est coordonnées sphériques et possède la symétrie de révolution.
L'intégrale doit être calculée en Python de façon <u>numérique</u>.
[On utilisera le module matplotlib.pyplot pour le tracé]