UNIVERSITÉ PARIS8 VINCENNES-SAINT-DENIS

Méthodologie de la Programmation

Palus Jean-Pascal

Licence IV - L1 - Semestre I

Bureau A184 - jpp@up8.edu

Séance VI

La mémoire en CPython

Les variables dans les langages proches du C

```
int a = 5
int b = 5
```

variable	adresse	valeur
a	0x3E8	101
b	0x3E9	101

Les variables dans les langages proches du C

```
int a = 5
int b = 5
```

variable	adresse	valeur
a	0x3E8	101
b	0x3E9	101

- Ces valeurs existent dans une case de **taille fixe**.

Les variables dans les langages proches du C

```
int a = 5
int b = 5

// changer la valeur de la variable
a = 6
```

variable	adresse	valeur
a	0x3E8	**110**
b	0x3E9	101

- La valeur de la variable est changée en mémoire.

Les variables en Python

- Python a des **noms**, pas des variables au sens usuel.

Comment les objets sont-t-il stockés en mémoire en Python ?

- Python a des **noms**, pas des variables au sens usuel.
- Le code python est compilé en bytecode avant d'être exécuté par la PVM (Python Virtual Machine).

Comment les objets sont-t-il stockés en mémoire en Python ?

- Python a des **noms**, pas des variables au sens usuel.
- Le code python est compilé en byte code avant d'être exécuté par la PVM (Python Virtual Machine).
- Python est dit Heap-based (basé sur le Tas), du point de vue de la PVM.

la Pile (Stack)

- La pile est la plage de mémoire où toutes les variables déclarées avant l'initialisation du programme sont stockées.

la Pile (Stack)

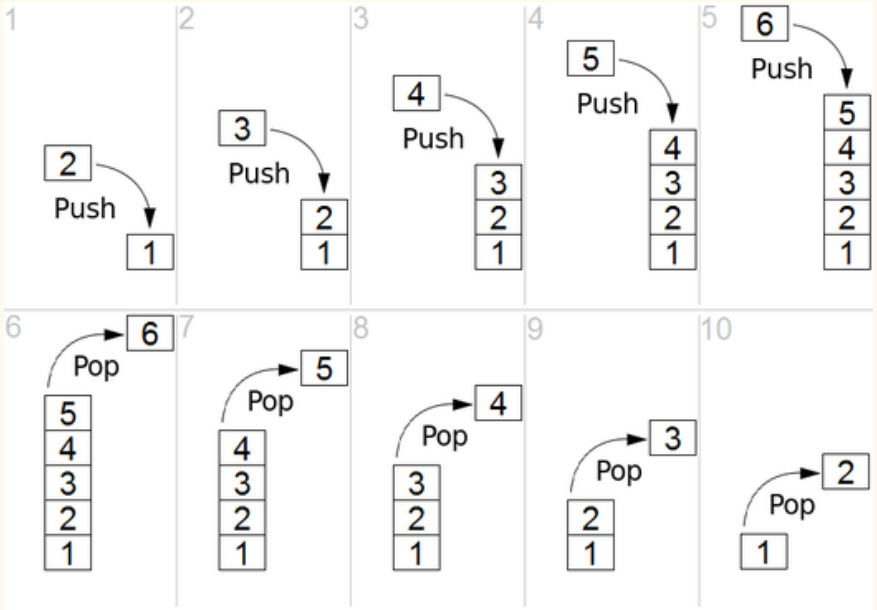
- La pile est la plage de mémoire où toutes les variables déclarées avant l'initialisation du programme sont stockées.
- Les données qui y sont stockées le sont temporairement ; une fois sorti du programme celle-ci est réalouée.

la Pile (Stack)

- La pile est la plage de mémoire où toutes les variables déclarées avant l'initialisation du programme sont stockées.
- Les données qui y sont stockées le sont temporairement ; une fois sorti du programme celle-ci est réalouée.
- Ce qui est en haut de la pile disparaît quand on sort du scope courant.

la Pile (Stack)

- La pile a un ordre de priorité **LIFO** (Last In First Out).
- Et deux opérations : **push** et **pop**.



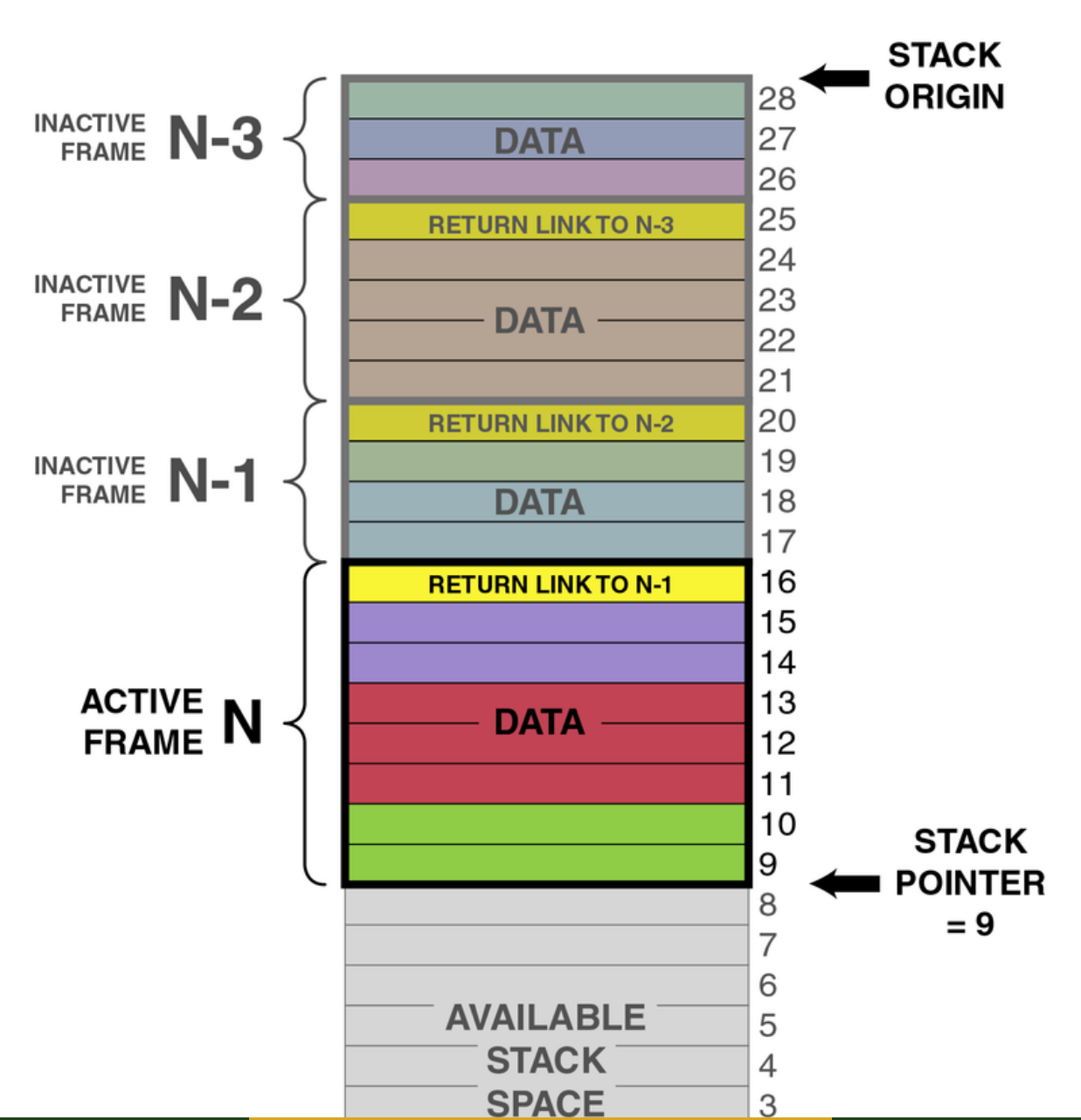
la Pile (Stack)

- La pile à une taille fixe, assignée au lancement du programme par l'OS.
- Certains langages permettent une manipulation directe de la pile ce qui peut se conclure en bugs (stack overflow).
- Le fait d'être directement empilée la rend rapide d'accès (car pas besoin de se rapeller de l'emplacement de l'élément suivant).

la Pile (Stack)

- La pile est donc :
  - Un espace temporaire de stockage mémoire.
  - Alloué par l'OS au lancement du programme.
  - Stocke les variables locales.

la Pile (Stack)



le Tas (Heap)

- Le tas est une plage de mémoire servant à l'allocation dynamique.

le Tas (Heap)

- Le tas est une plage de mémoire servant à l'allocation dynamique.
- Les variables globales y sont stockées.

le Tas (Heap)

- Le tas est une plage de mémoire servant à l'allocation dynamique.
- Les variables globales y sont stockées.
- Contrairement à la pile, la mémoire alouée doit être procéduralement libérée.

le Tas (Heap)

- Les données qui y sont stockées y résident de manière parmanente.

le Tas (Heap)

- Les données qui y sont stockées y résident de manière parmanente.
- La quantité de mémoire allouée au tas par l'OS n'est limité que par la capacité physique de la RAM.

le Tas (Heap)

- Les données qui y sont stockées y résident de manière parmanente.
- La quantité de mémoire allouée au tas par l'OS n'est limité que par la capacité physique de la RAM.
- Mais contrairement à la pile elle n'est pas linéaire, peut être fragmentée et donc est plus lente d'accès.

le Tas (Heap)

- Elle sert à stocker les variables qui ne peuvent être initialisées à la compilation où au chargement du programme (runtime).
- Certains langages permettent de la manipuler directement (malloc(), ...)

le Tas et la Pile en Python

- Elle sert à stocker les variables qui ne peuvent être initialisées à la compilation où au chargement du programme (runtime).
- Certains langages permettent de la manipuler directement (malloc(), new()...)
- Le tas est concurrentiel.

le Tas et la Pile en Python

- Python étant compilé en bytecode et exécuté par une machine virtuelle, la gestion des vrais pile et tas ne sont pas possible par le développeur.
- La PVM est une **stack machine** qui ne fait qu'empiler et dépiler des instructions.

La mémoire en Python

- En Python un **nom** sert à désigner un objet.
- Un objet peut avoir plusieurs noms.

```
# a est un nom de l'objet 5
a = 5
```

La mémoire en Python

- Chaque objet a un type :
  - simples (int, float, str, bool)
  - conteneurs (dict, list, classes définies par l'utilisateur, ...)
- Les conteneurs peuvent contenir des références vers d'autres objets (simple ou conteneurs).

Qu'est ce qu'une référence

- Une référence est un nom ou un objet conteneur qui pointe vers un autre objet.

Qu'est ce qu'une référence

- Une référence est un nom ou un objet conteneur qui pointe vers un autre objet.
- La PVM garde en mémoimoire le compte de chaque référence pointant vers chaque objet (ob_refcnt).

Compteur de références

- Une référence est un nom ou un objet conteneur qui pointe vers un autre objet.
- La PVM garde en mémoire le compte de chaque référence pointant vers chaque objet (ob_refcnt).

```
x = 42
# 42.ob_refcnt = 1
```

Compteur de références

- Le compteur de références peut s'incrémenter :

```
x = 42
# 42.ob_refcnt = 1

y = 42
# 42.ob_refcnt = 2
```

Compteur de références

- Le compteur de références peut s'incrémenter :

```
x = 42
# 42.ob_refcnt = 1

y = 42
# 42.ob_refcnt = 2

z = [42, 42]
# 42.ob_refcnt = 4
```

Compteur de références

- Le compteur de références peut s'incrémenter :

```
x = 42
# 42.ob_refcnt = 1

y = 42
# 42.ob_refcnt = 2

z = [42, 42]
# 42.ob_refcnt = 4
```

Compteur de références

- Le compteur de références peut aussi se décrémenter :

```
x = True
y = None
# 42.ob_refcnt -= 2
```

```
x = 42

del x
# 42.ob_refcnt -= 1
# Pas nécessaire. Ne supprime pas l'objet.
```

Compteur de références

- Le compteur de références peut aussi se décrémenter :

```
def print_hello():
    greeting = 'Hello'
    # Hello.ob_refcnt += 1
    print(greeting)

print_hello()
# Hello.ob_refcnt -= 1
```

Compteur de références

- Lorsque le compteur de références d'un objet arrive à 0, cet objet est supprimé de la mémoire.
- Attention aux variables globales.

Vérifier que deux noms pointent vers le même objet

```
x = 42
y = 42

print(id(x))
print(id(y))

print(x is y)
```

Le Garbage Collector

- C'est la partie de la PVM dont la fonction est de libérer automatiquement la mémoire des objets qui ne sont plus utilisés.

Le Garbage Collector

- Le **garbage collector** de Python utilise deux mécanismes :
  - Le comptage de références
  - Le traçage

Le comptage de références

- Supprime l'objet quand son compteur de référence == 0.
- A potentiellement un effet en cascade quand l'objet supprimé pointait vers un autre objet.

Le comptage de références

- Pose certains problèmes :
  - Couteux en mémoire
  - Couteux en cycle d'exécution
  - Thread-unsafe
  - Incapable de détuire les références cycliques.

Le traçage

- Consiste a faire tourner un algorithme de coloriage de graph (Mark-and-Sweep).
- Les noeuds du graph inatteignable depuis le scope sont marqués pour destruction.

Le traçage

- Consiste a faire tourner un algorithme de coloriage de graphe (Mark-and-Sweep).
- Les noeuds du graphe inatteignable depuis le scope sont marqués pour destruction.
- la PVM maintient des listes de "générations", contenant les objets par date de création pour optimiser Mark-and-Sweep.