La fonction `id()`

En Python, la fonction `id()` ne renvoie pas l'adresse mémoire d'un objet. Au lieu de cela, elle renvoie un identifiant unique pour l'objet, qui est utilisé par le système de gestion de la mémoire de Python. L'adresse mémoire réelle d'un objet n'est pas directement accessible via les fonctions de bibliothèque standard de Python, car Python supprime les détails de la gestion de la mémoire pour fournir un environnement de programmation de niveau supérieur, plus sécurisé et plus portable.

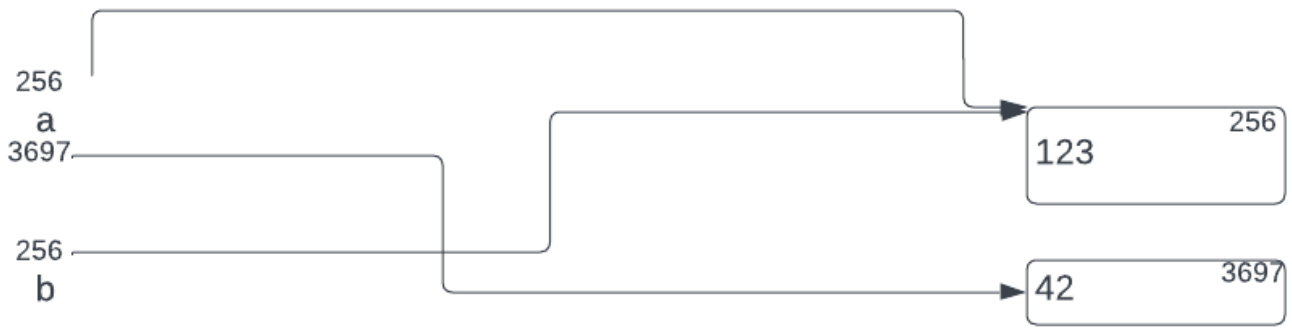
Dans une optique pédagogique, on peut considérer que ce que retourne `id()` est l'identifiant de l'espace mémoire dans lequel est stockée la valeur de la variable.

```
ma_variable = 55
print("id de ma_variable: ", id(ma_variable)) # l'adresse mémoire de ma variable
```

Ensuite grâce à l'exemple suivant, on peut comprendre que les passages par référence font pointer les espaces mémoire au même endroit, alors que les réaffectations font changer l'espace en mémoire.

```
a = 123
b = a # passage par référence
print("id de a: ", id(a))
print("id de b: ", id(b)) # On obtient la même adresse mémoire

# La variable "a" va changer d'espace mémoire lors de l'assignation alors que b
pointe toujours sur le même espace mémoire.
a = 42
print("id de a: ", id(a))
print("id de b: ", id(b))
# Les id ne renvoient donc plus la même valeur pour a et b
```



Les affectations multiples

Les affectations multiples permettent de déclarer et affecter plusieurs varaibles au même résultat, parfois utile au début des programme pour définir toutes les variables à des valeurs par défaut.

```
a = b = c = 0
a = b = 0 = c # synthaxe invalide
```

Les affectations parallèles

Très pratique pour les tuples, les dictionnaires, les tableaux et tous les élément qui possedent plusieurs informations. L'affectation parallèle permet d'affecter plusieurs variables en même temps.

- exemple entiers :

```
a, b = 2, 3
a, b = b, a
```

- exemple tuple :

```
mon_tuple = ("2", 3)

a, b = mon_tuple
print(a, type(a)) # 2 <class 'str'>
print(b, type(b)) # 3 <class 'int'>
```

- exemple dictionnaire :

```
mon_dict = {
    1: "toto",
    "element": 42,
    "valide": True
}

a, b, c = mon_dict["element"], mon_dict[1], mon_dict["valide"]
print(a, type(a)) # 42 <class 'int'>
print(b, type(b)) # toto <class 'str'>
print(c, type(c)) # True <class 'bool'>
```

⚠️ À utiliser dans la limite du raisonnable, sous peine de vous faire maudire par vos collègues.

```
infos, name, address = message.content.split(':')[0], message.content.split('[')
[0], message.content.split(';')[4]
```