Pense bête 1

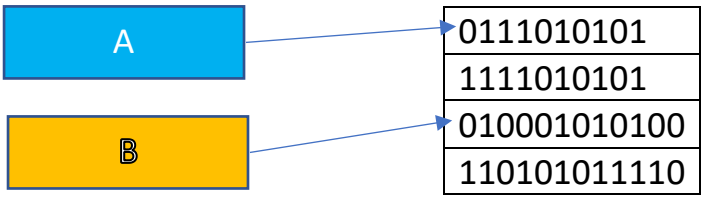
Variables

Un programme d'ordinateur manipule un grand nombre de données. Ces données sont mémorisées sous forme de 1 et de 0, dans la mémoire de l'ordinateur.

Dans un programme, pour pouvoir retrouver les données dont nous avons besoin, nous devons connaitre **leur adresse en mémoire** : où sont-elles stockées en mémoire ?

Ces adresses, appelées **références**, sont stockées dans ce qu'on appelle une **variable**, que l'on repère par son **nom**.

Le programme ne connait que le nom de la variable. Il n'a pas besoin de connaitre l'adresse physique de la donnée. C'est le travail du compilateur de faire le lien entre l'adresse contenue dans une variable et la donnée réelle en mémoire.



A et B sont deux variables contenant chacune une référence vers un emplacement de la mémoire de l'ordinateur.

Ces emplacements contiennent une donnée (un nombre, un caractère, une chaine de caractères etc.) que le programme pourra manipuler en utilisant les variables A et B.

En Python, la **déclaration** d'une variable et son **initialisation** (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps.

x = 2

Dans cet exemple, nous avons déclaré (donné un nom à la variable), puis initialisé la variable x avec la valeur 2.

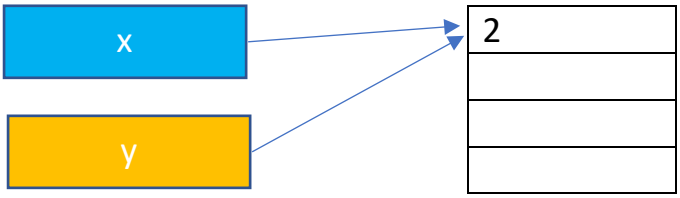
En réalité, il s'est passé plusieurs choses :

- Python a déterminé que la valeur 2 était un entier.
- Python a alloué (réservé) un espace en mémoire pour y accueillir un entier. Chaque type de donnée prend plus ou moins d'espace en mémoire. Python place la valeur 2 dans l'espace mémoire réservé. On dit que python a créé un **objet de type entier**.
- Python crée le nom de variable x.
- Python associe ce nom de variable x, à l'objet qu'il vient de créer : il place dans la variable x la **référence** à l'objet contenant la valeur 2.

Que se passe t'il lorsqu'on écrit : y = x ?

- Si y est un nouveau nom de variable, python crée ce nom
- Il copie ensuite dans la variable de nom y, la référence contenue dans la variable x. Il ne crée pas en mémoire, un nouvel objet entier.

Les 2 variables x et y pointent vers le même objet.

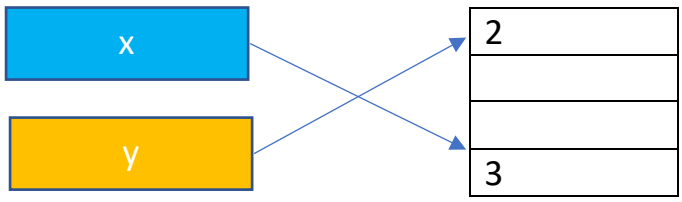


Si maintenant nous écrivons : x = x + 1 que fait python ?

- Il va rechercher la valeur contenue dans l'objet vers lequel x pointe : 2
- Il ajoute 1 à cette valeur
- Il alloue un **nouvel espace** en mémoire pour y stocker le résultat de l'opération : la valeur 3. Il crée **un nouvel objet** de type entier.
- Il place dans la variable x, la référence à ce nouvel objet.

Et y alors ?

y ne change pas et pointe toujours vers l'objet contenant la valeur 2.



Mettons ce processus en évidence :

Il existe une fonction dans python qui fournit la référence contenue dans une variable : **id(nom_variable)**

```
x = 1
print ("Référence contenue dans x :", id(x))
y = x
print ("Référence contenue dans y :", id(y))
x = x + 1
print ("Référence contenue dans x après l'instruction x = x + 1 : ", id(x))
print ("Référence contenue dans y après l'instruction x = x + 1 : ", id(y))
```

Résultat :

Référence contenue dans x : 1864741088

Référence contenue dans y : 1864741088

Référence contenue dans x après l'instruction x = x + 1 : 1864741120

Référence contenue dans y après l'instruction x = x + 1 : 1864741088

Nous pouvons constater que avant l'instruction x = x + 1 , les référencences dans x et y sont bien les même.

Après l'instruction x = x + 1 , la référence dans x a changé, alors que celle de y est resté la même.

Types de données :

Pour distinguer les uns des autres les divers contenus possibles, le langage de programmation utilise différents **types de variables**.

Entier : **int**. Ce type est utilisé pour stocker un entier, en anglais **integer** : 5, 20, -4

Flottant : **float**. Ce type est utilisé pour stocker des nombres à virgule flottante : 0.5, 12.,3.2e3 (3200)

Chaîne de caractères : **str**. Une chaîne de caractères (**string**) est une suite de caractères entre guillements ou apostrophes.

Booléen : **bool**. Une variable booléenne ne peut prendre que deux valeurs : vraie (True) ou fausse (False)

Liste : **list**. Collection d'éléments séparés par des virgules. Cet ensemble est délimité par des crochets : mois =[" janvier ","février","mars"]

La fonction **type**() permet de connaître le type d'une variable. C'est bien utile en python, car avec python, une variable prend le type de l'objet vers lequel elle pointe. Le type de l'objet pointé peut changer.

Python est un language à **typage dynamique**, ce qui signifie qu'il n'est pas nécessaire de déclarer à l'avance : cette variable est de type entier et ne peut donc pointer que vers des entiers, cette autre variable est de type flottant et donc toutes les valeurs qui lui seront affectées seront considérées comme des nombres avec virgule.

Une variable python peut contenir la référence à n'importe quel objet : nombre, liste, chaine de caractères, fonctions, etc.

Type mutable et type non mutable (point important)

Nous avons vu plus haut comment les objets contenant des données et les variables étaient créées et gérées par python.

Cette façon de faire s'applique aux objets **non mutables**, c'est-à-dire aux objets de type entier (int), floattant (float), booléen (bool), chaine de caractères (str).

Pour ces objets non mutables, python ne modifie pas le contenu d'un emplacement mémoire pointée par une variable, lorsque vous modifiez l'affectation de cette variable. Il crée un autre objet pour contenir le résultat de la nouvelle affectation.

La séquence **x = 2 puis x = x + 1** provoque la réservation d'un nouvel emplacement pour stocker la valeur 3, et donc la création d'un nouvel objet et une modification de la référence dans la variable x.

La valeur 3, ne vient pas remplacer la valeur 2 en mémoire.

Que devient l'objet contenant la valeur 2 si plus aucune variable ne contient sa référence ?

Python possède un **ramasse miette (garbage collector)** qui récupère l'espace mémoire alloué aux objets orphelins, c'est-à-dire ceux que l'on ne peut plus utiliser, car plus aucune variable n'en possède la référence.

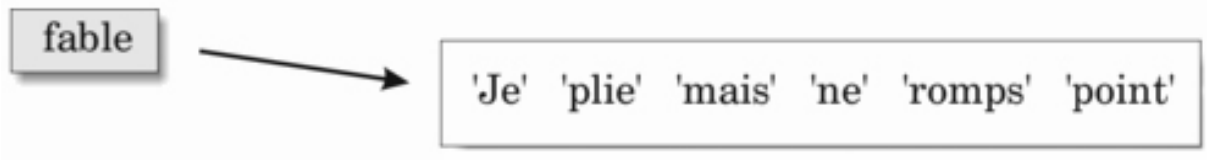
Si plus aucune variable ne pointe vers notre objet 2, le ramasse miette va récupérer la place mémoire occupée par cet objet.

Python traite d'une façon différente les **objets mutables** : les listes (list), les ensembles (set), les dictionnaires (dict).

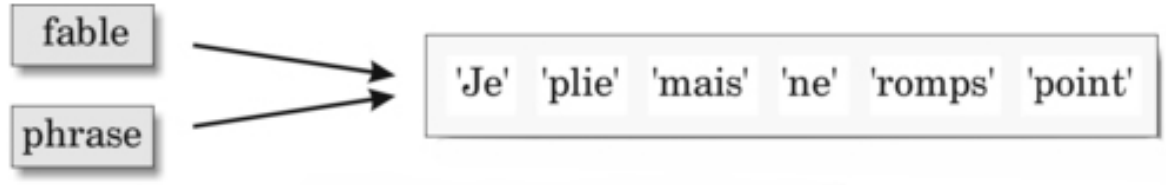
Exemple

Soit une variable fable déclarée comme suit :

fable = ['Je', 'plie', 'mais', 'ne', 'romps', 'point']



phrase = fable # les deux variables pointent vers la même liste.



Si maintenant on modifie le contenu de la liste en utilisant la variable **fable** :

fable[4] = ‘casse’ # ‘casse’ remplace le 5ième éléments de la liste

et que l’on fait afficher la liste vers laquelle fable pointe :

print(fable)

on obtient : [‘Je’, ‘plie’, ‘mais’, ‘ne’, ‘casse’, ‘point’]

Si on fait afficher le contenu de la liste vers laquelle **phrase** pointe :

print(phrase)

on obtient : [‘Je’, ‘plie’, ‘mais’, ‘ne’, ‘casse’, ‘point’]

On peut donc constater que Python **n’a pas modifié** la référence dans la variable **fable**. Il **n’a pas créé un nouvel objet** liste après la modification.

Les deux variables **fable** et **phrase** référencent toutes les deux la même liste après modification de cette dernière.