CPES Scientifique

Informatique

TP 2

Variables

Résumé

Ce TP introduit la notion fondamentale de *variable*.

1 Variables et identificateurs

Concrètement, chaque objet Python est stocké à un emplacement mémoire et cet emplacement a un numéro (*identifiant*) unique. Ce numéro est accessible grâce à la fonction Python « `id` ».

Attention : Les identifiants peuvent varier d'une machine à l'autre !

Question 1.1.

(a) Dans la console Python, trouver les identifiants de « `1` » et de « `True` ».

(b) Tester l'égalité de « `1` » et de « `True` ». Commenter.

Heureusement, on ne programme pas en manipulant directement des identifiants d'emplacements mémoire. Pour manipuler des objets stockés en mémoire, on utilise des *variables*. Une *variable* est une étiquette portant un nom (appelé *identificateur*) et pointant vers un emplacement mémoire. Les règles de construction des identificateurs Python sont données dans l'annexe A.

On peut se représenter une variable selon le schéma de la figure 1. Ce schéma décrit une variable dont l'identificateur est « `x` » et qui pointe vers l'emplacement mémoire contenant l'entier `1`. Par abus de language, dans le cas décrit fig. 1, on dit que la variable « `x` » « *contient* » la valeur `1`.

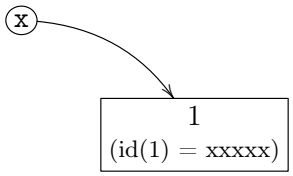


FIGURE 1 –

2 Utilisation des variables

Pour stocker une valeur « dans » une variable (c'est-à-dire pour qu'une variable pointe vers une case mémoire contenant cette valeur), on utilise l'*affectation*. Par exemple, si on veut qu'une variable de nom « `x` » « contienne » la valeur `1`, on fait

```
x = 1
```

Plus généralement, l'affectation est une instruction de la forme suivante :

identificateur = *expression*

Question 2.1. *Dans la console, affecter la valeur « `42` » à la variable « `y` », et comparer les* identifiants *de ces deux objets.*

Remarque 2.1 (L'affectation n'est pas symmétrique). Les expression

```
>>> x = y
```

et

```
>>> y = x
```

ont des effets différents !

Par exemple :

```
>>> x = 5
>>> y = 10
>>> x = y
>>> x
10
>>> y
10
>>>
>>>
>>> x = 5
>>> y = 10
>>> y = x
>>> x
5
>>> y
5
```

Lorsque l'on veut utiliser le contenu d'une variable (c'est-à-dire le contenu de l'emplacement mémoire vers lequel pointe cette variable), on utilise simplement son nom. Par exemple, dans la console, pour appeller le contenu d'un emplacement nommé par l'identificateur « `x` », on fait :

```
>>> x
1
```

Question 2.2. *Affecter les valeurs* 5, −10, `True` *respectivement aux variables « `a` », « `b` » et « `c` ». Afficher les contenu de ces variables.*

Quand on fait plusieurs affectations à la suite sur la même variable, on change l'emplacement désigné par cette variable.

Question 2.3. *Expliquer ce qu'il se passe lorsque l'on exécute les instructions suivantes :*

```
>>> x = 1
>>> x = True
>>> x
```

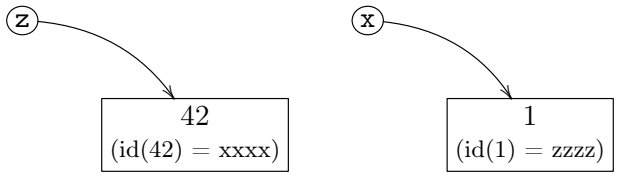
Plusieurs variables peuvent porter sur le même objet mémoire. De plus, on peut bien entendu affecter à une variable le contenu d'une autre variable.

Question 2.4. *Dans la console affecter la variable « `x` » à « `42` » et ensuite affecter la variables « `z` » à « `x` ». Comparer les identifiants de « `x` » et « `z` ».*

Attention : Il n'y a aucun lien entre « `x` » et « `z` » (si ce n'est que ces deux variables pointent vers la même case mémoire). En particulier, changer le contenu de « `x` » n'affectera pas la valeur de « `z` » (et vice-versa). Par exemple, dans la console :

```
>>> x = 1
>>> z
42
```

Ce qui correspond à la situation suivante :



3 Définition des noms et initialisation des variables

Lorsque l'on consulte le contenu d'une variable, celle-ci doit préalablement avoir été « *initialisée* », c'est-a-dire qu'une valeur doit lui avoir été donnée. Par exemple, dans la console, si on n'a jammais affecté la variable « `u` » alors on obtient :

```
>>> u
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'u' is not defined
```

La dernière ligne du message nous dit, en Anglais :

`NameError: name 'u' is not defined`

Ce que l'on pourrait traduire en Français par :

Erreur de nom : le nom `u` *n'est pas défini*

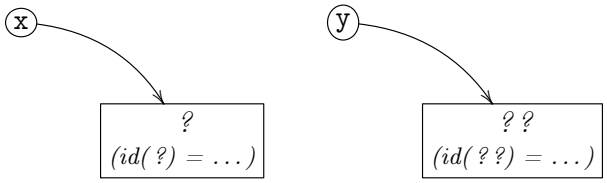
Pour qu'un nom soit défini il faut qu'il ait été préalablement initialisé.

Exemple 3.1. Dans la console :

```
>>> u
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'u' is not defined
>>> u = 1
>>> u
1
```

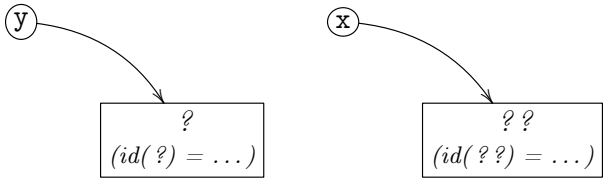
Nous verrons au prochain TP qu'un identificateur peut aussi désigner une fonction.

Question 3.1 (Échange)**.** *Supposons que les variables « `x` » et « `y` » ont déjà été initialisées, c'est à dire qu'on soit dans une situation ressemblant à*



avec $\texttt{id}(?) \neq \texttt{id}(??)$.

Donner une suite d'opérations permettant d'échanger le contenu de « `x` » et « `y` », c'est-à-dire d'arriver à la situation



A Règles de construction des identificateurs

Les identificateurs utilisés (en particulier) pour nommer les variables doivent être construits selon des règles précises. Chaque langage de programmation a ses propres règles pour la constructions des identificateurs.

En Python, les noms utilisés pour les identificateurs doivent suivre les règles suivantes :

— Ils doivent commencer par une *lettre* ou par un tiret bas « _ » (*underscore* en Anglais),

— suivi éventuellement d'une suite consituée de lettres, chiffres ou tirets bas.

Par exemple, dans la console :

```
>>> _x_9
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name '_x_9' is not defined
>>> _x-9
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name '_x' is not defined
```

Remarque A.1 (`True` et `False`). Les constantes Python `True` et `False` sont en particulier désignées par des identificateurs.

Il ne faut absolument pas les modifier, cela peut amener à des situations problématiques car difficiles à comprendre :

```
>>> True = 5
>>> True
5
>>> True == False
False
>>> False = 5
>>> True == False
True
>>> (True == False) == True
False
```