Informatique III
Types, variables, fonctions & expressions

Laurent Kaczmarek

PCSI^2 2013-2014
Lycée Louis Le Grand

Lundi 16 septembre 2013

La notion de type en informatique

Nécessité du typage

- Nous avons vu que l'information est stockée sous forme binaire dans un ordinateur, quelque soit sa nature (nombre entier relatif, décimal, chaîne de caractères, etc).
- Par exemple, selon l'interprétation qu'on en fait, la séquence binaire `00111010` peut désigner le caractère de ponctuation `:` (selon le code ASCII, *American Standard Code for Information Interchange*) ou l'entier `58`.
- Pour lever toute ambiguité sur l'interprétation d'une donnée sous forme binaire, on lui associe un *type*, ie une interprétation.

Les types du langage Python

Liste des principaux types

- Le type `int` pour les *entiers relatifs*.
- Le type `float` pour les *flottants* (sous-ensemble de l'ensemble des nombres décimaux permettant la représentation informatique des nombres réels, cf. le cours sur la représentation informatique des nombres).
- Le type `bool` pour les booléens $\{0, 1\} = \{\texttt{False}, \texttt{True}\}$.
- Le type `str` pour les chaînes de caractères.

La fonction type

Elle permet d'identifier le type d'une donnée.

```
>>> type(5)
<type 'int'>
>>> type(5.6)
<type 'float'>
>>> type('A')
<type 'str'>
>>> type(False)
<type 'bool'>
```

Le type int de Python

Les opérateurs entre nombres entiers sont conformes aux usages mathématiques :

- L'*addition* +, la *soustraction* -.
- La *multiplication* *, la *fonction puissance* **.
- Le *quotient* de la division euclidienne //, le *reste* de la division euclidienne %.

Exemples

```
>>> 45-4
41
>>> 45+34
79
```

```
>>> 45*4
180
>>> 45**2
2025
```

```
>>> 45//4
11
>>> 45%4
1
```

Le type `float` de Python

Les opérateurs entre nombres réels sont conformes aux usages mathématiques :

- L'*addition* `+`, la *soustraction* `-`.
- La *multiplication* `*`, la *fonction puissance* `**`.
- Le *quotient* `/`.

Exemples

```
>>> 7.6*4.7
35.72
>>> 4.3**5.4
2634.6908314601055
>>> 7/4
1.75
```

```
>>> 6.6-4.5
2.0999999999999996
>>> 6.5+9.5
16.0
```

Le type bool de Python

Ce type correspond aux valeurs de vérité `Vrai` et `Faux`. Il est au fondement des tests en informatique. Il sera étudié plus spécifiquement dans le cours sur les Tests.

Le type str de Python

Nous reviendrons sur ce type important ultérieurement.

- La *concaténation* +.

Exemples

```
>>> (-1)**4==1
True
```

```
>>> 'ABC'+'DEFG'
'ABCDEFG'
```

Le transtypage

Changement de type

- Les fonctions `int`, `float` et `str` permettent des changements de type.
- Cela peut être utile. Par exemple, une donnée entrée par l'utilisateur au moyen du clavier est traitée comme une chaîne de caractères. Pour effectuer des opérations sur elle, il faudra donc la convertir au préalable en un type numérique.

Exemples

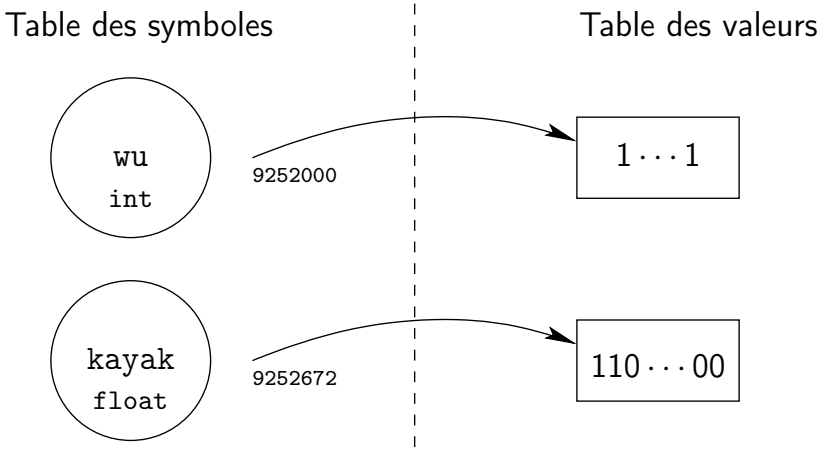
```
>>> str(4)
'4'
>>> int('3')
3
```

```
>>> int(6.7)
6
>>> float(5)
5.0
```

Manipulation des données

Les données manipulées par Python sont stockées dans des *tables de noms* (ou *namespace*). Ces données peuvent se représenter comme divisées en deux parties :

- La table des symboles, où sont stockés les noms des données. *À chaque nom de donnée est associé un type et une adresse dans la table des valeurs.*
- La table des valeurs, où sont stockées les valeurs des données. *Ce stockage se fait sous forme binaire.*
- Représentation schématique des tables :



La notion de variable

- L'association d'un type au nom de la donnée permet d'interpréter sans ambiguité le contenu binaire dans la case correspondante de la table des valeurs.
- Lors de l'exécution d'un programme, différents espaces de noms sont créés puis supprimés par l'interpréteur. La durée de vie d'une donnée est appelée sa *portée* (ou *scope*).
- On appelle *variable* l'association (selon le principe décrit ci-dessus) d'un nom (table des symboles) à une valeur (table des valeurs).
- Sous Python, le nom d'une variable est formé de lettres non accentuées, de chiffres, du caractère tiret bas _ et commence toujours par une lettre.

Affectation d'une variable sous Python

- Cette association de fait par une instruction appelée *affectation* au moyen du symbole =.
- Considérons l'exemple suivant : `wu=10+1`
- Voici comment Python traite cette instruction :
  1. Python commence par *évaluer* l'expression `10+1`, et note le *type* du résultat.
  2. Python stocke le résultat obtenu dans la table des valeurs, et note l'adresse utilisée.
  3. Python inscrit le *nom* dans la table des symboles, en précisant le type et l'identité associés.
- Si on affecte une valeur à une variable dont le nom existe déjà, Python supprime d'abord le nom de la table des symboles puis le réinscrit. Il ne s'agit donc plus de la même variable.

Les fonctions

En programmation impérative, une *fonction* est un algorithme qui prend des *arguments* en entrée, effectue une séquence d'instructions et renvoie un résultat.

Deux types de fonctions

On distingue deux types de fonctions :

- Les fonctions pré-définies, par exemple *cos*. Il faudra le plus souvent charger une *bibliothèque* contenant sa définition avant de pouvoir l'utiliser. Ce mécanisme permet d'économiser de la mémoire en laissant de côté les programmes dont l'utilisateur ne se sert pas.
- Les fonctions définies par les utilisateurs du langage. Nous reviendrons ultérieurement sur ce sujet.

Utilisation d'une fonction sous Python

- Une fonction est considérée par Python comme un objet de type *function*.
- On charge en mémoire la bibliothèque `nombibliotheque` contenant la définition de la fonction `nomfonction` avant de pouvoir l'utiliser sous la forme `nombibliotheque.nomfonction`.
- Si `f` est le nom de la fonction, on obtient le résultat de `f` pour un jeu de d'arguments `arg1`, ..., `argN` par l'appel suivant, conforme à l'usage mathématique :

  `f(arg1,...,argN)`

- Un exemple :

```
>>> import math
>>> from math import cos
>>> math.cos(pi/2)
6.123031769111886e-17
```

Construire et évaluer des expressions sous Python

- Une *expression* est formée à partir : de constantes, d'opérateurs, de noms de variables, de noms de fonctions (suivis d'arguments entre parenthèses), ...
- Pour évaluer une expression, Python remplace chaque nom de variable par la valeur correspondante, et chaque nom de fonction (suivi d'arguments) par la valeur renvoyée par la fonction.
- Écrire une expression *dans l'interpréteur* puis appuyer sur la touche *Return* a pour effet l'évaluation et l'affichage de la valeur de l'expression.
- En revanche, dans le cas d'un programme écrit dans l'éditeur de pyzo, il faudra utiliser la commande `print` pour forcer l'affichage d'une expression au cours de l'exécution du programme.

Les expressions

Construire et évaluer des expressions sous Python

- Une expression est toujours évaluée *avant* d'être stockée dans une variable.
- Ce mécanisme d'évaluation permet d'utiliser les opérateurs usuels (+,*,etc) sur les variables (pourvu que leurs types soient corrects !) et d'utiliser des variables comme arguments d'une fonction.

Exemples

```
>>> x=3
>>> y=2*x+1
>>> x=1000
>>> z=y**2+1
>>> x,y,z
(1000, 7, 50)
```

```
>>> x*y
7000
>>> type(z)
<type 'int'>
>>> y==(y//x)*x+y%x
True
```

Les affectations multiples sous Python

- Sous Python, on peut réaliser des *affectations multiples*. Ces dernières sont très spécifiques au langage Python et ne pourront pas être utilisées dans d'autres langages. Toutes les expressions sont évaluées (de gauche à droite) *avant* la première affectation.
- Exemple d'affectations multiples :

```
>>> x=2
>>> y=3
>>> x,y=2*x+3*y,4*x
>>> x,y
(13,8)
```