Cours Info - 4

Types et opérateurs de base

D.Malka

MPSI 2016-2017

Sommaire

Sommaire

Expressions

Expressions

Une expression est une suite de caractères définissant une valeur. Pour calculer cette valeur, la machine évalue l'expression. En machine, une expression est toujours un mot binaire (ex : 000110010).

Exemples d'expressions

- **-42**
- **3.0**
- **1+4**
- **3*8/10.0**
- **"Toto"**
- **x+3**
- **not True**

Types des expressions

Types des expressions

Le type d'une expression permet de l'interpréter. C'est la syntaxe qui indique le typage de l'expression.

Types des expressions

Ainsi le mot binaire `1000001` peut signifier :

- 65 interprété comme un entier non signé,
- -63 interprété comme un entier signé,
- *A* interprété comme un caractère,
- *True* interprété comme un booléen.

Types des expressions

La fonction (type(*expression*)) permet de connaître le type de l'expression en argument :

CODE PYTHON

```
In [1]: type(2)
Out[1]: int

In [2]: type(2.0)
Out[2]: float

In [3]: type(1//3)
Out[3]: int

In [4]: type('a')
Out[4]: str

In [5]: type(True)
Out[5]: bool
```

Différentes expressions

Dans la plupart des langages de programmation, une expression est :

- **soit une constante comme `42` ;**
- **soit un nom de variable comme `x` ou `position` ;**
- **soit une expression entre parenthèse comme en mathématiques `(2-3)` ;**
- **soit composée de plusieurs expressions réunies par des opérateurs comme `1+(4-6)*3/2` ;**
- **soit composé d'une fonction appliquée à d'autre expression comme `fact(4)`**

Le type entier `int`

- **Type `int`**
- **Ecriture en base 10 : `15, 9, 5, -2...`**

CODE PYTHON

```
In [6]: 121165497
Out[6]: 121165497
```

- **A priori codé sur 32bits ou 64bits suivant l'architecture :** $-9223372036854775808 < N < 9223372036854775807$ **;**
- **Python : codage particulier pour les grands entiers. Taille en mémoire illimitée (sauf par la machine).**
- **Arithmétique juste !**

Exemples d'entiers

CODE PYTHON

```
In [7]: 42
Out[7]: 42

In [8]: -12
Out[8]: -12

In [9]: 156987
Out[9]: 156987

In [10]: 1216516541698544
Out[10]: 1216516541698544L
```

Le type `float`

Les entiers flottant : le type `float`.

- **Flottant = nombres à virgules**
- **Symbole « . » à la place de « , »**
- **Ecriture en base 10 : `3.0`, `-15.2`, `3.1451...`,`10e-4`**
- **Taille ou capacité** $10^{-324} < x < 10^{309}$

CODE PYTHON

```
In [11]: 1.0e-350
Out[11]: 0.0

In [12]: 1.0e309
Out[12]: Inf
```

- **Ensemble des flottants $\mathbb{F}$ approximation de l'ensemble des réels $\mathbb{R}$.**
- **Arithmétique fausse !**

Entier ou flottant ?

Entier ou flottant ?

CODE PYTHON

```
In [13]: type(3)
Out[13]: int

In [14]: type(3.0)
Out[14]: float
```

Entier ou flottant ?

La présence du point `.` signe le type `float` de la valeur de l'expression.

Exemples de flottants

CODE PYTHON

```
In [15]: 12.0
Out[15]: 12.0

In [16]: -3.4
Out[16]: -3.4

In [17]: 12e-4
Out[17]: 0.0012
```

Le type booléen : bool

- **Type `bool`**
- **Uniquement 2 constantes :**
  - **`True`**
  - **`False`**
- **Le résultat d'un test (i.e valeur d'une expression logique). Par exemple 1<2 ou True or False ou x !=3.**

Sommaire

Opérateurs numériques

Les opérateurs numériques.

+	addition
-	soustraction ou inversion de signe
*	multiplication
/	division décimale
//	division entière
%	modulo (reste de la division entière)
**	exponentiation

Opérateurs logiques

Les opérateurs logiques.

==	égalité (test)
!=	inégalité (test)
<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à
not	négation (non logique)
or	disjonction (ou logique)
and	conjonction (et logique)

Précédence des opérateurs

- **Les expressions entre parenthèses sont exécutées prioritairement les parenthèses intérieures vers des parenthèses extérieures.**
- **En l'absence de parenthèse, la priorité des opérateurs est la suivante (du plus au moins prioritaire) :**
  - **
  - *,/,//,%
  - +,-
  - ==, !=,<,<=,>,>=
  - not
  - and
  - or
- **Entre opérateurs de priorité identique, le calcul s'exécute de gauche à droite.**

Opérations sur les entiers

CODE PYTHON

```
In [18]: 4-2
Out[18]: 2

In [19]: 8/4
Out[19]: 2

In [20]: 8/3
Out[20]: 2

In [21]: 8%3
Out[21]: 2

In [22]: 2*4+4**5
Out[22]: 1032

In [23]: 2<1
Out[23]: False

In [24]: 2==1+1
Out[24]: True
```

Opérations sur les flottants

CODE PYTHON

```
In [25]: 2.0*5.3
Out[25]: 10.6

In [26]: 8.0/3.0
Out[26]: 2.6666666666666665

In [27]: 10.0/3.0*3.0
Out[27]: 10.0

In [28]: 10**2>2**10
Out[28]: False
```

Opérations sur les booléens

CODE PYTHON

```
In [29]: True and True
Out[29]: True

In [30]: True and False
Out[30]: False

In [31]: False and False
Out[31]: False

In [32]: True or True
Out[32]: True

In [33]: True or False
Out[33]: True

In [34]: False or False
Out[34]: False

In [35]: not True
Out[35]: False

In [36]: not False
Out[36]: True
```

Opérations sur plusieurs types

CODE PYTHON

```
In [37]: True and 2<=1
Out[37]: False

In [38]: True and 0
Out[38]: 0

In [39]: 8/3
Out[39]: 2

In [40]: 8/3.0
Out[40]: 2.6666666666666665

In [41]: 8.0/3
Out[41]: 2.6666666666666665

In [42]: (2*3+2>=8 and 7**1.5/4)==(7/4+3>5 or 2==1+1)
Out[42]: False
```

Sommaire

float → int

Fonction `float(int n)` et `int(float f)` :

CODE PYTHON

```
In [43]: float(14)
Out[43]: 14.0

In [44]: int(14.7)
Out[44]: 14
```

float ou int ⟷ bool

Fonction `float(bool b), int(bool b) , bool(int n), bool(float f)` :

CODE PYTHON

```
In [45]: bool(2.0)
Out[45]: True

In [46]: bool(2)
Out[46]: True

In [47]: bool(0)
Out[47]: False

In [48]: int(True)
Out[48]: 1

In [49]: int(False)
Out[49]: 0
```

float ou int ⟷ bool

Tester :

CODE PYTHON

```
In [50]: 0 and True
Out[50]: False

In [51]: 0 or False
Out[51]: False

In [52]: 0 or 1
Out[52]: True

In [53]: 0 and 8.0
Out[53]: False
```

Interprétation booléenne des valeurs numériques

Dans de nombreux langage dont Python, l'interprétation logique d'une expression numérique `expr` est la suivante :

- si expr est nulle, elle interprétée comme `False`,
- si expr est non nulle, elle interprétée comme `True`

Prochain chapitre

Comment stocker en mémoire la valeur d'une expression ?