Chapitre 1 : Introduction à Python

Axel Carpentier

Informatique

Python

Table des matières

1. Interface

2. Représentation en Python
   2.1 Notion de type
   2.2 Conversion de type
   2.3 Booléens

3. Variables

Interface

1. Interface

2. Représentation en Python
   2.1 Notion de type
   2.2 Conversion de type
   2.3 Booléens

3. Variables

Interface

Python est un langage de programmation polyvalent et modulaire, utilisé dans de très nombreux domaines, scientifiques ou non se présentant sous la forme suivante : Une partie interactive et un script.



Interface

Dans le premier cas, il y a un dialogue entre l'utilisateur et l'interprète : les commandes entrées par l'utilisateur sont évaluées au fur et à mesure.
Par exemple ci-dessous dans le mode interactif :

```
>>> 1+1
2
```



Interface

Cette première solution est pratique pour réaliser des prototypes, ainsi que pour tester tout ou partie d'un programme ou plus simplement pour interagir aisément et rapidement avec des structures de données complexes.

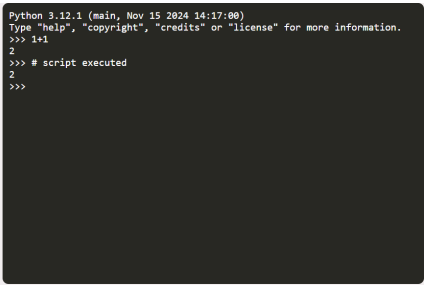
Pour une utilisation en mode script, les instructions à évaluer par l'interprète sont sauvegardées, comme n'importe quel programme informatique, dans un fichier. Dans ce second cas, l'utilisateur doit saisir l'intégralité des instructions qu'il souhaite voir évaluer à l'aide de l'éditeur de texte, prévoir des effets pour afficher les résultats souhaités s'il y en a, puis demander leur exécution à l'interprète en utilisant la commande `print`.

Interface

Par exemple ci-dessous dans le mode script :

```
1 a=1
2 b=1
3 print(a+b)
```





Représentation en Python

1. Interface

2. Représentation en Python
   2.1 Notion de type
   2.2 Conversion de type
   2.3 Booléens

3. Variables

Un aspect essentiel des langages de programmation est la notion de type : tout objet manipulé par Python en possède un, qui caractérise la manière dont il est représenté en mémoire, et dont vont dépendre les fonctions qu'on peut lui appliquer.
On peut faire l'analogie en mathématiques, pour la fonction $f : x \mapsto 3x + 2$, $f(Antoine)$ n'a aucun sens. Ici c'est le même principe. La commande `type()` permet de déterminer le type d'un élément.

Définition 1.

Nous travaillerons principalement avec les types suivants :

- `int` : les nombres entiers.
- `float` : les nombres réels (ou flottants).
- `str` : les chaines de caractère (délimitées par des guillemets "...").
- `bool` : les booléens (`True` ou `False`).
- `list` : Les listes (délimités par des crochets [..., ...]).

```
1  >>> type(5)
2  'int'
3
4  >>> type(5.3)
5  'float'
6
7  >>> type("Antoine de Saint-Exupery")
8  'str'
9
10 >>> type(True)
11 'bool'
12
13 >>> type([4,-0.967,"M.Carpentier"])
14 'list'
```

Exercice 1.
Déterminer les types des éléments suivants :

- -17
- `False`
- 4.78
- $[4, 7.9, -6, \text{"Bonjour"}]$
- "Seconde"

Remarque 1.

En mathématiques, un nombre entier est aussi un nombre réel. Ce n'est pas le cas en informatique, chacun des deux types `int` et `float` étant représenté d'une façon spécifique en mémoire. par exemple, le nombre 2 peut être représenté :

- Dans le type `int` sous la forme `2`.
- Dans le type `float` sous la forme `2.0`.

Propriété 1. *Les pincipales opérations sur les nombres*

- `x+y` : somme de `x` et de `y`.
- `x-y` : différence de `x` et de `y`.
- `x*y` : produit de `x` et de `y`.
- `x / y` : quotient de `x` et de `y`.
- `x**y` : `x` puissance `y`.
- `x // y` : quotient de la division euclidienne de `x` par `y`.
- `x % y` : reste de la division euclidienne de `x` par `y`.
- `abs(x)` : valeur absolue (ou module) de `x`.

```
1  >>> 2+3
2  5
3  >>> 9-11
4  -2
5  >>> 5*4
6  20
7  >>> 24/4
8  6.0
9  >>> 2**3
10 8
11 >>> 23//3
12 7
13 >>> 23%3
14 2
```

Représentation en Python

1. Interface

2. Représentation en Python
   2.1 Notion de type
   2.2 Conversion de type
   2.3 Booléens

3. Variables

Nous savons qu'en mathématiques on a l'inclusion d'ensemble suivante : $\mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$.
Par ailleurs, il a été évoqué qu'en informatique ce n'était pas le cas de par la représentation de chaque nombre en mémoire.

Il est cependant possible de "forcer" un nombre à rentrer dans un type spécifique.
Une analogie mathématiques serait d'écrire $2,0$, $\frac{4}{2}$ ou $2+0i$ à la place de 2 pour bien avoir que 2, pour les écritures précédentes, est un élément de $\mathbb{D}$, $\mathbb{Q}$ et $\mathbb{C}$.

```
1  >>> int(6.78)
2  6
3
4  >>> int(-6.78)
5  -6
6
7  >>> float(13)
8  13.0
9
10 >>> complex(2)
11 2+0j
```

Représentation en Python

1. Interface

2. Représentation en Python
   2.1 Notion de type
   2.2 Conversion de type
   2.3 Booléens

3. Variables

Le type `bool` est un type qui ne possède que deux éléments : `True` ou `False` représentant très intuitivement le vrai et le faux.
Leur intérêt principal, nous le verrons lorsque l'on évoquera la notion d'instruction, est de tester la véracité d'assertions.
Au type `bool` sont associés trois opérateurs : `not`, `and` et `or`.

Par ailleurs, un certain nombre d'opérateurs sont définis sur d'autres types (en particulier les types de nombres) et à valeurs dans le type `bool`.

Propriété 2. *Les pincipaux opérateurs booléens*

- `x<y` : `x` strictement plus petit que `y`.
- `x>y` : `x` strictement plus grand que `y`.
- `x<=y` : `x` inférieur ou égal à `y`.
- `x>=y` : `x` supérieur ou égal à `y`.
- `x==y` : `x` est égal à `y`.
- `x!=y` : `x` est différent de `y`.

Exemple 1:

```
1 >>> 7<8
2 True
3 >>> 7<8 and 4<3
4 False
5 >>>  7<8 or 4<3
6 True
```

Exercice 2:

Que vont renvoyer les tests suivants ?

```
1 >>> (1+3==4 and 5+8<=16) or 10+3>= 9
2
3 >>> (1+3!=4 or 5+8<=16) and 10+3<= 9
```

Variables

1. Interface

2. Représentation en Python
   2.1 Notion de type
   2.2 Conversion de type
   2.3 Booléens

3. Variables

Variables

Les données calculées peuvent être mémorisées à l'aide de variables. Pour ce faire, il faut attribuer un nom à cette variable et lui associer une valeur à l'aide de l'opérateur d'affectation $=$. Le nom de la variable est une suite de lettres (minuscules ou majuscules) et de chiffres, qui doit toujours commencer par une lettre. Il est d'usage de choisir des termes explicites pour faciliter la lecture du code.

Une fois affectée, la valeur de la variable peut être utilisée dans un calcul en faisant référence à son nom. Voici un exemple de calcul de l'aire d'un rectangle dans lequel les noms de variables ont été choisis de manière à rendre le code limpide :

```
1 longueur = 5.67
2 largeur = 2.53
3 aire = longueur * largeur
4 print("l'aire␣du␣rectangle␣est␣egale␣a", aire)
```

Exercice 3:
Soit un triangle de base $3,89$ cm et de hauteur $8,42$ cm. Déterminer sur Python l'aire du triangle.

```
1 ...
2 ...
3 ...
4 ...
```

Nous verrons également qu'il peut être souvent très utile de modifier le contenu d'une variable à partir de sa propre valeur :

```
1 longueur = 5.67
2 longueur = longueur+1
3 >>> longueur
4 6.67
```