Chapitre 1: Variables, types et boucles

Brice Mayag
brice.mayag@dauphine.fr

M1 SIREN: maj. SIEE/GSI

Dauphine | PSL
UNIVERSITÉ PARIS

Sommaire

Sommaire

Les instructions d'un algorithme ont généralement pour but de traiter des données d'entrée (numériques, textes, images, ...) qui peuvent être saisies par l'utilisateur, résulter d'un calcul de l'ordinateur, ...

Les espaces mémoire où ces données sont conservées sont appelés variables.

Les variables sont essentielles pour pouvoir écrire un programme informatique.

Définition

Une variable est définie par trois composantes :

- un identifiant (le nom de la variable)
- un type de données
- une valeur courante, dans le domaine des valeurs du type

En langage algorithmique, on utilise le symbole $\longleftarrow$ pour indiquer qu'une valeur est affectée à une variable : $a \longleftarrow 1$.

En Python, on utilise le symbole "=", qui s'appelle dans ce contexte opérateur d'affectation.

Les mots clé réservés du langage ne peuvent pas être utilisés comme noms de variables. Par exemple and, as, break, class, def, if, else, while, for, return, None, True, False

Sommaire

Les structures de données

Structure de données

Une structure de données est définie par 3 choses :

1. Un **type** qui est simplement un nom permettant de classifier les données relevant de ce nom.
2. Un **ensemble** qui définit avec précision les éléments appartenant au type.
3. des **opérations** utilisées par les programmes pour calculer sur ces données

On parle de structures algébriques.

Les booléens

Ce type de données est prédéfini dans Python par :

- son nom bool
- ses valeurs {True,False}
- ses opérations
  - not : $bool \rightarrow bool$,
  - and : $bool \times bool \rightarrow bool$, et
  - or : $bool \times bool \rightarrow bool$

Nom vient de George Boole [1815-1864] mathématicien anglais intéressé par les propriétés algébriques des valeurs de vérité.

Les entiers relatifs

Ce type de données est prédéfini dans Python par :

- son nom int
- son ensemble de valeurs est théoriquement $\mathbb{Z}$ mais est en fait borné selon la machine utilisée.
- ses opérations
  - +, -, *, /, %, ** : $int \times int \rightarrow int$,
  - ==, >, <, >=, <=, != : $int \times int \rightarrow bool$

Les opérations de calcul (+,-,** ...) sont prioritaires sur les opérateurs de comparaison (==,<=, ...).

Quelques exemples d'utilisation des entiers

```
>>> type(46)  #type : fonction pr\'{e}d\'{e}finie fournissant
type 'int'              #le type d'une expression
>>> print 5 * 7   #print : fonction pr\'{e}d\'{e}finie d'impression
35
>>> 5 / 2   # / quotient de la division euclidienne
2
>>> 5 % 2 #  % reste de la division euclidienne
1
>>> 5 ** 3 # fonction d'exponentation
125
>>> 5 < 3 # pr\'{e}dicat de comparaison "strictement inf\'{e}rieur"
False
>>> 5 ==  3 + 2 # op\'{e}ration + est prioritaire sur ==
True
>>> 5 = 3 + 2
SyntaxError: can't assign to literal
```

Les nombres réels

En fait, les nombres réels ne sont pas représentables en machine. On les approxime par des nombres rationnels (i.e. avec des virgules).
Ce type de données est prédéfini dans Python par :

- son nom float
- son ensemble de valeurs est théoriquement $\mathbb{R}$ mais est en fait un sous-ensemble de $\mathbb{Q}$. De plus, il est borné selon la machine utilisée.
- ses opérations
  - $+$, -, *, /, ** : $float \times float \rightarrow float$,
  - int : $float \rightarrow int$, et float : $int \rightarrow float$
  - $==$, $>$, $<$, $>=$, $<=$, $!=$: $float \times float \rightarrow bool$

Les flottants s'écrivent en utilisant le point pour séparer la partie décimale de la partie entière.

Quelques exemples d'utilisation des nombres réels

```
>>> type(46.8)
<type 'float'>
>>> 5.0 / 2.  # le point distingue un flottant d'un entier,
2.5
>>> 5. / 2  # 2 est converti en flottant
2.5
>>> 7.5 * 2
15.0
>>> int (7.5 * 2)
15
>>> int (7.75)
7
>>> float (3)
3.0
```

Les chaînes de caractères

Type immuable (on ne peut pas changer les valeurs)

Description formelle

Une chaîne de caractères est une application $s : S \to char$ où S est une séquence finie d'entiers positifs $\{0, \ldots, n\}$ et char est l'ensemble des 256 caractères ascii.

Ce type de données est prédéfini dans Python par :

- son nom str
- son ensemble de valeurs est tout mot fini (i.e. une suite de caractères ascii).
- ses opérations
  - $+ : str \times str \to str$ (concaténation de deux chaînes de caractères),
  - $* : str \times int \to str$ l'entier doit être positif,
  - len : $str \to int$ (longueur d'une chaîne de caractères),
  - $==, >, <, >=, <=, != : str \times str \to bool$

Les chaînes de caractères - suite

En python, une chaîne se décrit en extension en écrivant le mot avec des guillemets : $s =$ "$a_1 \dots a_n$".

La notation $s[i]$ pour $i = 0, \dots, (n-1)$ dénote la lettre à la position $i+1$ dans la chaîne s.

On peut accéder à des sous chaînes de caractères d'une chaîne donnée. Soit s une chaîne de longueur n. s[i :j] avec $i < j$ et $i, j \leq n$, dénote la sous-chaîne $s[i]s[i+1] \dots s[j-1]$

Accès aux sous-chaînes par les (intervalles d')index

```
>>> test ='1234567'
>>> test[3:6]
'456'
>>> test[0]
'1'
>>> test[3]
'4'
>>> test[6]
'7'
>>> test[3:4]
'4'
```

Sommaire

Les expressions conditionnelles

Condition en python

Une expression conditionnelle python se déclare de la façon suivante

```
if expression_conditionnelle :
    Instructions
else :
    Instructions
```

Les expressions conditionnelles - suite

Dans le cas où l'on ne souhaite pas rien faire lorsque la condition est fausse, on peut omettre la partie "else".

Enfin, il arrive que l'on ait des "cascades" d'expressions conditionnelles. On peut alors utiliser le mot clé elif.

```
if expression_conditionnelle :
    Instructions
elif expression_conditionnelle :
    Instructions
else :
    Instructions
```

Les boucles While et for

Boucle while

Une boucle while python se déclare de la façon suivante

```
while expression_conditionnelle :
    Instructions
```

Boucle for

Une boucle for python se déclare de la façon suivante

```
for variable in collection :
    Instructions
```

Les collections dans les boucles "for" peuvent être de plusieurs formes :

- "range(ind_début,ind_fin,pas)". La variable va prendre pour valeur à chaque étape de la boucle "for" ind_début,ind_début+pas,...,ind_fin.
- une chaine de caractères ou une liste. Dans ce cas là, la variable va parcourir toute les valeurs se trouvant aux positions allant de 0 à $len(s) - 1$ où s est la chaîne de caractères ou la liste.

Les listes/tableaux

Description formelle

Une liste est une application $s : S \rightarrow TYPE$ où S est une séquences d'entiers positifs $\{0, \ldots, n\}$ et TYPE est l'union de tous les types de données.

Ce type de données est prédéfini dans Python par :

- son nom lists
- son ensemble de valeurs est tout mot fini sur l'ensemble TYPE pris comme alphabet.
- ses opérations
  - $+$: $list \times list \rightarrow list$ (concaténation de deux listes),
  - $*$: $list \times int \rightarrow list$ l'entier doit être positif,
  - len : $list \rightarrow int$ (longueur d'une liste),
  - $==, >, <, >=, <=, !=$: $list \times list \rightarrow bool$

En python, une liste se décrit en extension en écrivant le mot entre crochets : $l = [a_1, \ldots, a_n]$. La notation [] dénote la liste vide.

Les listes - suite

En python, les listes sont mutables, i.e., on peut changer les valeurs au différentes positions. On utilise la notation : $l[i] = e$ met l'élément e à la position i dans la liste l

On peut accéder à des sous listes d'une liste donnée. Soit l une liste de longueur n. l[i :j] avec $i < j$ et $i, j \leq n$, dénote la sous-liste $l[i]l[i+1] \ldots l[j]$

Convention : $l[: i]$ et $l[i :]$ dénotent les i premiers et les $n - i$ derniers éléments de la liste l, respectivement.

Quelques illustrations sur les listes

```
>>> type([1,2])
<type 'list'>
>>> [2+3,"toto",2.5]
[5, 'toto', 2.5]
>>> type([2+3,"toto",2.5])
<type 'list'>
>>> [5,[2,2],"toto"]
[5, [2, 2], 'toto']
>>> ["toto",5,"tutu",5.5][1]
5
>>> ["toto",5,"tutu",5.5][1:3]
[5, 'tutu']
>>> "tutu" in ["toto",5,"tutu",5.5]
True
>>> "tu" in ["toto",5,"tutu",5.5]
False
```