

Chapitre 4: Instructions simples

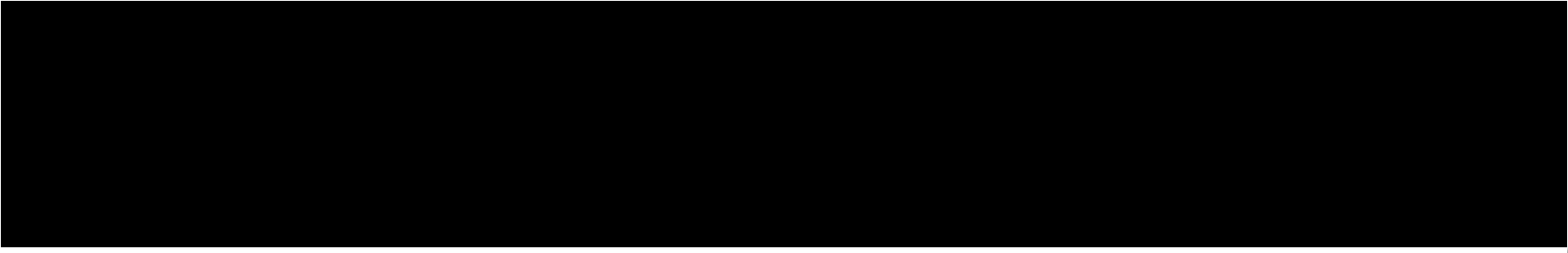
4.1 Affectation

On a déjà vu deux type s'affectation:

- Affectation directe: `<variable> = <littéral>`
- Affectation simple: `<variable> = <expression>`

Exemple:

```
>>> s = "Bonjour"
>>> n = 3 * 4
>>> x = 25.0 ** 0.5
>>> b = n < x or n % 2 == 0
```



Il y a trois autres formes d'affectations couramment utilisées qui ne sont que des abréviations de la forme générale

`<variable> = <expression>`

Ces trois formes d'affectations se nomment:
affectation composée.

- L'expression est une opération binaire et la variable affectée est la même que l'opérand gauche de l'expression.

L'instruction

<variable> = <variable> <opérateur binaire> <expression>

Est équivalente à l'instruction

<variable> <opérateur binaire> = <expression>

Exemple: Incrémenter une variable de 1 (très utilisé en pratique).

```
>>> v = v + 1    s'écrit généralement
```

```
>>> v += 1
```

```
>>> v = 5        # affectation
```

```
>>> v += 1       # incrémentation
```

```
>>> v
```

```
6
```

```
>>>
```

- **Affectation simultanée de plusieurs variables à une même expression.**

La suite d'instructions

```
<var_1> = <expression>  
<var_2> = <expression>  
...  
<var_n> = <expression>
```

est équivalente à l'instruction

```
<var_1> = <var_2> = ... = <var_n> = <expression>
```

Exemple: initialisation (i.e. affectation) de plusieurs variables à zéro.

```
>>> a = b = c = 0    # affectation
>>> a
0
>>> b
0
>>> c
0
>>>
```

- Affectation simultanée de plusieurs variables à différentes expressions.

La suite d'instructions

```
<var_1> = <expr_1>  
<var_2> = <expr_2>  
...  
<var_n> = <expr_n>
```

est équivalente à l'instruction

```
<var_1> , <var_2> , ... , <var_n> = <expr_1> , <expr_2> , ... , <expr_n>
```


Exemple:

```
>>> h,m,s = 10,33,14      # affectation
>>> print "secondes depuis minuit = " , \
      h*3600+m*60+s
secondes depuis minuit = 37994
>>> print "secondes depuis minuit = " + \
      str(h*3600+m*60+s)
secondes depuis minuit = 37994
```


Exemple: échange de variables (très utilisé en pratique).

```
>>> a = 4          # affectation
>>> b = 5          # affectation
>>> print a,b
4 5
>>> a,b = b,a      # réaffectation
>>> print a,b
5 4
>>>
```

Remarque: en fait, l'instruction « a,b = b,a » n'est pas strictement équivalente aux instructions séquentielles « a = b » puis « b = a ». En effet:

```
>>> a,b = 4,5          # affectation
>>> a,b = b,a          # réaffectation
>>> print a,b
5 4
```

```
>>> a,b = 4,5          # affectation
>>> a = b               # réaffectation
>>> b = a               # réaffectation
>>> print a,b
5 5
```



Ainsi, dans certains cas, l'*affectation simultanée* de variables n'est pas équivalente à l'*affectation séquentielle* de ces mêmes variables.

Exercice: Comment échanger le contenu de deux variables de manière purement séquentielle (i.e. sans utiliser l'affectation simultanée)?

```
>>> a = 4          # affectation
>>> b = 5          # affectation
>>> z = a          # nouvelle var z qui prend
                   # le contenu de a, i.e. 4
>>> a = b          # a vaut maintenant 5
>>> b = z          # b vaut z, i.e. l'ancien
                   # contenu de a, i.e. 4
>>> print a,b
5 4
```

4.2 Instructions composées ou bloc d'instructions

A la différence d'autres langages, Python n'entoure pas un groupe d'instructions par des accolades ou les mots réservés **begin** et **end**.

C'est grâce à l'*indentation* que l'on matérialise un groupe d'instructions. Donc, toutes les instructions qui commencent au même niveau d'indentation font partie du même groupe.

- La première instruction d'un fichier ne doit pas être indentée.
- Les instructions tapées en mode interactif (i.e. à l'invite >>>) ne doivent pas non plus être indentées.
- La fin d'une ligne physique marque la fin d'une instruction.
- Pour forcer une instruction à tenir sur plusieurs lignes, on emploie le caractère « \ » (*back-slash*).

4.3 Instructions d'entrée-sortie

■ Instruction d'entrée


Pour communiquer une valeur de l'utilisateur vers l'application, on utilise l'instruction:

```
raw_input (prompt )
```

Cette instruction retourne une chaîne de caractères tapée par l'utilisateur à l'invite du paramètre `prompt`.

Exemple:

```
>>> age = raw_input('Quel est votre age: ')
Quel est votre age: 23
>>> age+3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and
'int' objects
>>> int(age)+3
26
>>>
```



Remarque: la fonction « `raw_input ()` » convertit automatiquement ce qui est entré au clavier par l'utilisateur *en chaîne de caractères*.

La fonction « `input ()` » est similaire mais devine le type de ce qui est entré au clavier par l'utilisateur.

Pour plus de contrôle, on conseille d'utiliser « `raw_input ()` ».

Exemple:

```
>>> age = input('Quel est votre age: ')\nQuel est votre age: 23\n>>> age+3\n26\n>>>
```

Dans ce cas là, Python comprend directement que la valeur 23 est de type `int`.

■ Instruction de sortie

Pour communiquer une valeur de l'application vers l'utilisateur, on utilise l'instruction:

```
print expr_1, expr_2, ..., expr_n
```

Exemple:

```
>>> h, m, s = 10, 33, 14
>>> print "secondes depuis minuit =", \
      h*3600 + m*60 + s
secondes depuis minuit = 37994
>>>
```

opérateur de continuation
de ligne

4.4 Instruction vide

L'instruction **pass** est une instruction qui ne fait rien.

Elle peut apparaître partout où la syntaxe requiert une instruction.

Exemple:

```
>>> a = 7
```

```
>>> pass
```

```
>>> b , c = a > 5 , "bonjour"
```