1 Les opérateurs de calcul simples

Addition, Soustraction :

```
>>> 2 + 3
5
>>> 5 - 2
3
```

Multiplication :

```
>>> 2 * 3
6
```

Division flottante :

```
>>> 5 / 2
2.5
```

Division entière :

```
>>> 5 // 2
2
```

Reste de la division euclidienne (modulo) :

```
>>> 5 % 2
1
```

Puissance :

```
>>> 2 ** 3
8
```

2 Affectation de variables

L'opérateur d'affectation en Python est le signe égal (=).

```
# Exemple :
n = 27
tup = (3, 7, -12)
```

Il est possible d'effectuer des affectations multiples sur une seule ligne :

```
# Exemple :
n1, n2, ch = 24, 36, "bonjour"
```

Il est possible de permuter le contenu de deux variables en une seule instruction :

```
# Exemple :
a, b = 5, 17
a, b = b, a
```

À l'issue de ces instructions `a` vaut 17 et `b` vaut 5.

Sans cette possibilité, il faudrait passer par une variable tampon intermédiaire :

```
# Exemple :
a, b = 5, 16    # a vaut 5 et b vaut 16
tampon = a      # tampon vaut 5 (a et b n'ont pas changé)
a = b           # a vaut 16 (b n'a pas changé)
b = tampon      # b vaut 5 (et a vaut 16)
```

3 Structure conditionnelle

Les structures conditionnelles permettent de « choisir » quelles instructions exécuter en fonction du résultat d'un test (ou d'une série de tests).

Cela correspond à la structure algorithmique : **Si ... Sinon ... Alors**

En Python, on peut avoir par exemple :

```
if age >= 18:
  acces = "majeur"
elif age > 10:
  acces = "mineur accompagné"
elif age > 6:
  acces = "entrée déconseillée"
else:
  acces = "interdit"
```

Lorsqu'il y a une série de tests comme dans l'exemple ci-dessus, les tests sont évalués dans l'ordre jusqu'à avoir un résultat positif et le bloc de code qui suit est alors exécuté, en revanche aucun autre test ne sera effectué par la suite et le programme se poursuit après l'ensemble de la structure conditionnelle.

Bien remarquer que chaque ligne de test se termine par un **double point** (:) et que **le bloc de code à la suite est indenté**, càd décalé par rapport à la marge.

Remarque : les blocs de code sont toujours identifiés par l'indentation en Python, mais la plupart des autres langages ont des caractères spéciaux pour délimiter les blocs de code (avec des accolades par exemple en JavaScript ou en C).

4 Boucle bornée : Pour (for)

4.1 Boucle

L'utilisation d'une boucle bornée permet d'effectuer une action un certain nombre connu de fois : c'est la boucle « pour » (for en Python).

Exemple : quel capital obtient-on au bout de 5 ans avec un gain de 10% annuel, en partant d'un capital de 100 € ?

```
capital = 100
for i in range(5):
  capital = capital * 1.1 # augmentation du capital de 10%
```

4.2 Complément sur l'instruction range de Python

La syntaxe générale est la suivante :
`for i in range(debut, fin, intervalle)`

Exemple : `for i in range(5, 23, 6)` :
La variable `i` prend les valeurs de 5 (inclus) à 23 (exclu) par pas de 6 : 5, 11, 17.

Par défaut, `debut` vaut 0 et `intervalle` vaut 1.

Exemple : `for i in range(5, 23)` :
La variable `i` prend les valeurs de 5 (inclus) à 23 (exclu) par pas de 1 : 5, 6, 7, 8, ..., 21, 22.
Exemple : `for i in range(23)` :
La variable `i` prend les valeurs de 0 (inclus) à 23 (exclu) par pas de 1 : 0, 1, 2, 3, ..., 21, 22.
Exemple : `for i in range(0, 23, 6)` :
La variable `i` prend les valeurs de 0 (inclus) à 23 (exclu) par pas de 6 : 0, 6, 12, 18.

5 Boucle non bornée : Tant que (while)

L'utilisation d'une boucle non bornée permet d'effectuer une action tant qu'une condition reste vraie : c'est la boucle « tant que » (while en Python).

Exemple : combien faut-il d'années pour atteindre un capital de 200 € avec un gain de 10% annuel, en partant d'un capital de 100 € ?

```
capital = 100
annee = 0
while capital < 200:
  capital = capital * 1.1 # augmentation du capital de 10%
  annee = annee + 1
```

6 Terminaison : variant de boucle

Lorsqu'on construit un algorithme, il est important de s'assurer qu'il terminera toujours quelques soient les valeurs de ses entrées.

Un algorithme reposant sur des boucles bornées est certain de terminer car une boucle bornée s'effectue un nombre défini de fois. En revanche, un algorithme reposant sur des boucles non bornées pourrait ne pas terminer si la condition de boucle reste toujours vraie.

On peut prouver la **terminaison** d'un algorithme en exhibant un **variant de boucle** : c'est une **grandeur qui reste positive et qui décroît à chaque tour de boucle**. Ainsi si une telle grandeur existe, il est certain que la boucle s'arrêtera.

Dans l'exemple précédent, la grandeur `(200 - capital)` est un variant de boucle : c'est une valeur positive décroissante (puisque `capital` est croissant) : lorsqu'elle devient négative, on quitte la boucle, prouvant ainsi sa terminaison.