

Algorithmique et programmation

Leçon

Remarque. Cette partie est transversale : les algorithmes sont travaillés dans tous les chapitres (encadrement de $\sqrt{2}$, multiples, balayage, simulations...). Cette leçon rassemble les notions de base.

1. Variables et affectation

Définition

Une **variable** est une case mémoire désignée par un nom, qui contient une valeur. Son **type** dépend de la valeur :

Type	Python	Exemples
entier	int	5, -12
flottant (nombre à virgule)	float	2.5, 7/2, 1e-3
chaîne de caractères	str	"bonjour", "42"
booléen	bool	True, False

Définition

L'**affectation** donne une valeur à une variable. En langage naturel on écrit $a \leftarrow 5$; en Python `a = 5`.

Remarque. Le signe `=` de Python n'est pas l'égalité mathématique : `x = x + 1` signifie « la nouvelle valeur de x est l'ancienne augmentée de 1 ». Pour tester une égalité, on écrit `==`.

Propriété – Opérations utiles

`+` `-` `*` `/`; puissance `**`; quotient entier `//`; reste `%`; racine carrée `sqrt` (module `math`).
Exemples : `7 / 2` vaut 3.5; `7 // 2` vaut 3; `7 % 2` vaut 1; `2**10` vaut 1024.

2. Instruction conditionnelle

Définition

Une instruction conditionnelle exécute des instructions seulement si une condition est vraie.

Langage naturel

Si $x > 0$ **alors**

afficher « positif »

Sinon si $x < 0$ **alors**

afficher « négatif »

Sinon

afficher « nul »

Fin Si

Python

```

1 if x > 0:
2     print("positif")
3 elif x < 0:
4     print("négatif")
5 else:
6     print("nul")

```

Remarque. Comparaisons : `==`, `!=` (différent), `<`, `<=`, `>`, `>=`. Connecteurs logiques : `and`, `or`, `not`. En Python, l'**indentation** (décalage) délimite les blocs d'instructions : elle est obligatoire.

3. Boucles

Définition – Boucle bornée

On l'utilise quand on connaît à l'avance le nombre de répétitions. `for i in range(n)` : répète le bloc pour i valant $0, 1, \dots, n-1$ (n fois). `range(a, b)` donne les entiers de a à $b-1$.

Exemple. Calcul de $1 + 2 + \dots + 100$:

```
1 s = 0
2 for k in range(1, 101):
3     s = s + k
4 print(s)      # 5050
```

Définition – Boucle non bornée

On l'utilise quand on ne connaît pas le nombre de répétitions : `while condition` : répète le bloc **tant que** la condition est vraie.

Exemple. Plus petit entier n tel que $2^n > 1000$:

```
1 n = 0
2 while 2**n <= 1000:
3     n = n + 1
4 print(n)      # 10
```

Remarque. Dans une boucle `while`, il faut s'assurer que la condition finira par devenir fausse, sinon le programme ne s'arrête pas (boucle infinie).

4. Fonctions

Définition

Une **fonction** est un bloc d'instructions nommé, qui reçoit des **arguments** (paramètres) et **renvoie** un résultat avec `return`. On l'**appelle** en écrivant son nom suivi des valeurs des arguments.

Exemple.

```
1 def f(x):
2     return x**2 - 3*x + 1
3
4 def aire_rectangle(longueur, largeur):
5     return longueur * largeur
6
7 print(f(2))          # -1
8 print(aire_rectangle(5, 3)) # 15
```

Remarque. `print` affiche une valeur à l'écran ; `return` renvoie une valeur qui peut être réutilisée dans un calcul.

Définition – Fonctions aléatoires

Le module `random` fournit `random()` (flottant au hasard dans $[0; 1[$) et `randint(a, b)` (entier au hasard entre a et b inclus). En répétant l'appel, on obtient une série statistique (simulation).

Exemple.

```
1 from random import randint
2
3 def lancer_de():
4     return randint(1, 6)
5
6 def nombre_de_six(n):
7     c = 0
8     for i in range(n):
9         if lancer_de() == 6:
10             c = c + 1
11     return c
```

5. Bonnes pratiques

Méthode – Écrire et vérifier un programme

1. Écrire d'abord l'algorithme en langage naturel.
2. Choisir des noms de variables explicites.
3. Tester le programme sur des cas dont on connaît le résultat.
4. En cas d'erreur, faire un **tableau d'état des variables** (« trace ») à la main.