Cours

Introduction

Un **algorithme** est une suite d'instructions à exécuter.
Celui-ci peut être écrit en **langage naturel**, c'est-à-dire en donnant les instructions en français, mais pour être exécuté, l'algorithme doit être traduit dans un langage de programmation.

Un même algorithme peut être **implémenté** dans plusieurs langages de programmation différents. Par exemple, le langage utilisé par les calculatrices TI n'est pas le même que celui utilisé par les calculatrices Casio. Ainsi, si on souhaite implémenter un algorithme sur sa calculatrice, on ne saisira pas le même **programme** selon la marque de celle-ci.

Variables et affectation

Une variable est désignée par un nom (une lettre, un mot, plusieurs mots séparés par « _ », sans espace) et contient une valeur d'un certain type. Les types que nous utiliserons ici sont :

- `int` : nombre entier relatif (positif ou négatif).
  Exemple : `4` et `-5` sont de type int.

- `float` : nombre flottant (à virgule).
  Exemple : `2.56` et `-1.627` sont de type float.

- `str` : chaîne de caractères, entourée d'apostrophes (`'...'`) ou de guillemets (`"..."`).
  Une chaîne de caractères correspond souvent à du texte, chaque caractère étant un symbole (lettre, chiffre, ponctuation...).
  Exemple : `'une chaine'` est de type `str` et contient 10 caractères (9 lettres et un espace qui est aussi un caractère).

- `bool` : booléen, qui peut avoir deux valeurs possibles seulement : `True` (vrai) et `False` (faux)

- `tuple` : suite d'éléments de types éventuellement différents, séparés par des virgules et entourés par des parenthèses (`(...)`).
  Exemple : `(3, 'triplet', True)` est un tuple contenant trois éléments.

- `list` : liste d'éléments, souvent de même type, mais ce n'est pas obligatoire, séparés par des virgules et entourés par des crochets (`[...]`).
  Exemples : - `[0, 2, 4, 6, 8]` est une liste contenant 5 éléments.
  - `[]` est une liste vide (ne contenant aucun élément).
  - `[[3, 6, 9], [4, 8], [5, 10]]` est une liste dont les éléments sont eux-mêmes des listes.
  - `[(64, 'pair'), (15, 'impair'), (12, 'pair')]` est une liste de couples.

⚠ **Remarque :** Attention, le symbole utilisé pour la virgule est « . » tandis que le symbole « , » sert à séparer des éléments, d'une liste par exemple.

En langage Python on utilise l'instruction `a = ...` pour affecter une valeur à une variable `a`.

Exemple : Voici un exemple de saisie dans une console d'exécution (le symbole `#` indique un commentaire, qui sera ignoré par Python) :

```
>>> a = 3   # on saisit les commandes après le prompt >>>
>>> b = 5   # a contient la valeur 3, b contient la valeur 5
>>> a + b
8       # résultat obtenu
>>> a - b
-2
>>> a * b
15
>>> a / b
0.6
>>> a ** 2  # a puissance 2, c-a-d a au carré
9
>>> a // 5, a % 5   # quotient et reste de la division euclidienne de a par 5
(28, 4)
```

Remarque : `a+b`, `a-b` et `a*b` sont des entiers (type `int`), `a/b` est un flottant (type `float`). Enfin, le dernier résultat est un couple d'entiers (type `tuple`).

Remarque : Attention, Python n'est pas un logiciel de calcul formel. Nous ne nous arrêterons pas dans ce cours sur la représentation des flottants, mais il faut avoir en tête qu'un ordinateur utilise une représentation binaire des nombres. On peut donc parfois obtenir des résultats surprenants comme par exemple...

```
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
```

Pour accéder à davantage de fonctionnalités on peut importer certains modules comme par exemple le module `math` (accès aux fonctions mathématiques), le module `random` (génération de nombres aléatoires) ou le module `turtle` (module graphique).

Exemple : Les instructions suivantes permettent d'importer la fonction `sqrt` (racine carrée), du module `math`, puis de déterminer $\sqrt{144}$.

```
>>> from math import sqrt   # import de la fonction racine, du module math
>>> a = 144
>>> sqrt(a)      # racine carrée de a
12.0
```

Remarque : Si l'on souhaite directement importer toutes les fonctionnalités du module math on pourra directement saisir `from math import *`.

Pour afficher des valeurs et/ou du texte on utilise la fonction `print` :

```
>>> b = 8
>>> print('la racine carrée de ', b, ' vaut ', sqrt(b))
la racine carrée de 8 vaut 2.8284271247461903
```

Dans le cas des **séquences**, c'est-à-dire des listes, des chaînes de caractères et des tuples, l'opérateur + sert à la concaténation de deux séquences de même type, ce qui consiste à les mettre bout à bout, tandis que `len` détermine la longueur.

Cependant dans le cas des listes on préférera ajouter des éléments à l'aide de la méthode `append`. Enfin, pour accéder au i-ème élément d'une variable on ajoute `[i]` au nom de la variable.

⚠ **Remarque :** Attention, le premier élément est d'indice 0 et non 1 !
Pour accéder au dernier élément on peut aussi indiquer `[-1]`.

Exemple :

```
>>> puis_2 = [2, 4, 8, 16]  # type list
>>> puis_2
[2, 4, 8, 16]
>>> len(puis_2)
4
>>> puis_2[0]
2
>>> puis_2[-1]
16
>>> puis_2.append(64)        # on ajoute la valeur 64 à la liste puis_2
>>> puis_2
[2, 4, 8, 16, 64]
>>> len(puis_2)
5
>>> ch = 'bon' + 'jour'      # type str
>>> ch
'bonjour'
>>> len(ch)
7
>>> ch[3]
'j'
>>> ch[-1]
'r'
>>> couple = puis_2, ch      # type tuple
>>> couple
([2, 4, 8, 16, 64], 'bonjour')
>>> len(couple)
2
>>> t = couple + couple
>>> t
([2, 4, 8, 16, 64], 'bonjour', [2, 4, 8, 16, 64], 'bonjour')
>>> t[1]
'bonjour'
```

On peut aussi réaffecter un élément d'une liste existante avec une instruction du type `L[i] = ...`

Exemple :

```
>>> L = [0, 3, 7, 9, 12, 15, 18, 21]
>>> len(L)
8
```

Cours

```
>>> L[2]
7
>>> L[2] = 6
>>> L
[0, 3, 6, 9, 12, 18, 21]
```

Et les booléens ? Lorsque l'on teste une condition, à l'aide de l'un des opérateurs suivant, le résultat est un booléen.

Opérateur	Signification
==	« est égal à »
<	« est strictement inférieur à »
>	« est strictement supérieur à »
<=	« est inférieur ou égal à »
>=	« est supérieur ou égal à »
!=	« est différent de »

Exemple : Testons si 584 est divisible par 3 et si 255 est divisible par 5, puis vérifions que $584 > 255$ et que $2 \times 584 = 255 - 74$.

```
>>> a = 584
>>> b = 255
>>> a % 3 == 0      # le reste de la division eucl. de 584 par 3 vaut 0?
False           # 584 n'est donc pas divisible par 3
>>> b % 5 == 0      # le reste de la division eucl. de 255 par 5 vaut 0?
True            # 255 est donc divisible par 5
>>> a > b
True
>>> 2*b != a-74     # l'opérateur "!=" signifie "est différent de"
False
```

La valeur stockée dans une variable est susceptible de changer au fur et à mesure de l'exécution des instructions d'un algorithme. L'image mentale qu'on peut avoir d'une variable est celle d'un tiroir dans lequel on stocke une valeur, à laquelle on peut accéder quand on le souhaite et qu'on peut éventuellement remplacer par une autre valeur. Si on réaffecte une variable, l'ancienne valeur n'est pas gardée en mémoire.

Exemple :

```
>>> c = 2
>>> c = c ** 2      # c prend la valeur 2 au carré, donc 4
>>> c = c ** 2      # c prend la valeur 4 au carré, donc 16
>>> c = c ** 2      # c prend la valeur 16 au carré, donc 256
>>> print('c = ', c)
c = 256
```

On peut également affecter plusieurs variables en même temps, puis les réaffecter simultanément.

Exemple :

```
>>> a, b, c, d, e = 27, 12, 35, 4, 9
>>> a, b, c, d, e = d, e, b, a, c
>>> print(a, '<', b, '<', c, '<', d, '<', e)
4 < 9 < 12 < 27 < 35
```

■ Fonctions

Les fonctions Python sont définies à l'aide du mot-clef `def` : on donne un nom à la fonction et on précise ses éventuels paramètres (on indique () s'il n'y a pas de paramètre). On indique ensuite les différentes instructions à exécuter lors de l'appel de la fonction. Il n'y a aucun affichage si ce n'est pas explicitement demandé dans les instructions. On peut parfois utiliser `print` pour l'affichage.

Exemple : La fonction `dev` permet d'afficher le quotient et le reste de la division euclidienne d'un nombre a par un nombre b.
La fonction `affiche` n'a pas de paramètres et affiche simplement une chaîne de caractères.

```
>>> def div(a, b):
        print('le quotient de la division euclidienne de ',
            a, ' par ', b, ' vaut ', a // b)
        print('le reste de la division euclidienne de ',
            a, ' par ', b, ' vaut ', a % b)

>>> div(17, 5)
le quotient de la division euclidienne de 17 par 5 vaut 3
le reste de la division euclidienne de 17 par 5 vaut 2
>>> def affiche():
        print('texte a afficher')

>>> affiche()
texte a afficher
```

⚠ **Remarque :** On fera extrêmement attention à l'**indentation** (le retrait en début de ligne). Une erreur d'indentation peut empêcher le bon fonctionnement d'un programme.

On utilise `return` lorsque l'on souhaite interrompre l'exécution de l'algorithme et retourner un certain résultat.

Exemple : Fonction $x \longmapsto (5x-3)^2$.

```
>>> def fonction(x):
        v = 5 * x
        v = v - 3
        v = v ** 2
        return v
```

```
>>> fonction(0)
9
>>> fonction(2)
49
```

⚠ **Remarque :** Il est nécéssaire d'utiliser la commande `return` et non la fonction `print` afin de pouvoir exploiter les résultats.

```
>>> def fonction_bis(x):
        v = 5 * x
        v = v - 3
        v = v ** 2
        print(v)

>>> fonction_bis(0)
9
>>> fonction_bis(2)
49
```

Les résultats obtenus avec les fonctions `fonction` et `fonction_bis` semblent identiques, pourtant...

```
>>> fonction(2) < fonction(0)
False
>>> fonction_bis(2) < fonction_bis(0)
49
9
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    fonction_bis(2) < fonction_bis(0)
TypeError: '<' not supported between instances of 'NoneType' and 'NoneType'
>>> fonction(2) < fonction(0)
False
```

Lorsque l'instruction `fonction_bis(2) < fonction_bis(0)` est exécutée, l'appel `fonction_bis(2)` affiche la valeur 49, puis l'appel `fonction_bis(0)` affiche la valeur 9, enfin, les deux résultats sont comparés. Or la fonction `print` ne fait qu'afficher les résultats, sans les stocker, il n'y a donc rien à comparer !

■ Instruction conditionnelle (If)

Pour effectuer des instructions différentes en fonction de certaines conditions à vérifier on utilise la structure conditionnelle `If`. En language naturel cela s'écrit :

Si *Condition 1*
    Instruction 1
Autre Si *Condition 2*
    Instruction 2
Sinon
    Instruction 3
Fin Si

En langage Python on utilise `if` pour « Si », on inscrit la condition à vérifier pour exécuter les instructions, puis « : » avant de donner les instructions à la ligne suivante.

Il n'y a pas de « Fin Si », c'est l'indentation qui indique si l'on est encore en train de donner des instructions ou non. Dans l'éventualité où il y aurait plusieurs cas à distinguer on utilise `elif` pour « Autre Si ». Enfin, on utilise `else` pour « Sinon ».

Exemple : Fonction qui teste la parité d'un nombre entier.

```
>>> def parite(n):
        if n % 2 == 0:      # reste de la div. eucl. de n par 2 vaut 0?
            print(n, " est pair")
        else:
            print(n, " est impair")

>>> parite(23)
23  est impair
>>> parite(16)
16  est pair
```

Remarque : Ce programme affiche un résultat très compréhensible mais celui-ci n'est pas utilisable par la suite puisque l'on a utilisé la fonction `print` (voir page 8). Pour pouvoir exploiter cette fonction dans d'autres programmes il est préférable d'utiliser `return` et de renvoyer un booléen.

```
>>> def est_pair(n):
        if n % 2 == 0:
            return True
        else:
            return False

>>> est_pair(23)
False
>>> est_pair(16)
True
```

L'utilisation d'instructions conditionnelles n'est en fait pas nécessaire dans ce cas particulier :

```
>>> def est_pair(n):
        return n % 2 == 0
>>> est_pair(23)
False
```

Exemple : Fonction qui teste deux entiers sont tous les deux impairs.

```
>>> def sont_impairs(n, m):
        if est_pair(n):
            return False
        elif est_pair(m):
            return False
        return True
```

```
>>> sont_impairs(5, 17) # aucune des deux conditions n'est verifiée,
True                    # on sort de la structure "if" et on retourne True
>>> sont_impairs(7, 12) # la deuxième condition est verifiée,
False                   # l'exécution s'arrête et on renvoie False
```

Exemple : Définissons une fonction qui, lorsqu'on saisit les coordonnées de 3 points A, B et C indique si le triangle ABC est rectangle ou non.
On définit une première fonction qui, lorsqu'on donne les coordonnées de deux points A et B, renvoie la longueur du segment $[AB]$.

```
>>> def norme(a, b):  # a et b ne sont pas des nombres mais les points A et B
        xa, ya = a     # on note (xa; ya) les coordonnées de A
        xb, yb = b     # on note (xb; yb) les coordonnées de B
        return sqrt((xb - xa)**2 + (yb - ya)**2)  # formule de la distance AB

>>> norme((-2, 2), (6, -4))
10
```

Remarque : L'utilisation de `sqrt` suppose que l'on a au préalable importé cette fonction du module `math` avec l'instruction `from math import sqrt` voire `from math import *` (voir page 4).

La fonction suivante, qui utilise la fonction `norme`, renvoie une chaîne de caractères indiquant si oui on non le triangle est rectangle :

```
>>> def est_rectangle(a, b, c):
        ab = norme(a, b)
        ac = norme(a, c)
        bc = norme(b, c)
        if bc**2 == ab**2 + ac**2:
            return 'ABC est rectangle en A'
        elif ac**2 == ab**2 + bc**2:
            return 'ABC est rectangle en B'
        elif ab**2 == ac**2 + bc**2:
            return 'ABC est rectangle en C'
        else:
            return "le triangle n'est pas rectangle"
            # utilisation des guillemets " pour permettre le caractère '
>>> a = (-2, 2)
>>> b = (2, 4)
>>> c = (6, -4)
>>> d = (-3, 5)
>>> est_rectangle(a, b, c)
'ABC est rectangle en B'
>>> est_rectangle(a, b, d)
"le triangle n'est pas rectangle"
```