Premiers programmes et structures usuelles

Nous avons construit nos premiers programmes sous la forme de fonctions qui pour des arguments donnés, renvoyaient un ou plusieurs résultats. On rappelle d'ailleurs que pour éviter les messages d'erreur, il a fallu rester vigilant sur la structure physique du langage Python : ce sont les indentations, les sauts de lignes et autres symboles dédiés qui remplacent les balises souvent utilisées dans d'autres langages de programmation...

On verra donc ici comment gérer les données de programmation, avant de définir les **principaux blocs d'instructions** qui structureront nos programmes.

La gestion des données de programmation

La plupart du temps, on définira nos procédures sous la forme de **programmes fonctionnels**, dont les arguments seront renseignés à l'appel de la fonction.
Par exemple, si on souhaite construire le programme *equation* qui pour tout nombre $a \geq 0$, renvoie les solutions de l'équation $x^2 = a$:

```python
from math import *

def equation(a):
    x1=sqrt(a)
    x2=-x1
    return x1,x2
```

```
In : equation(4)
                                        2,-2
```

Mais pour d'autres raisons, on peut décider de rendre le **programme intéractif** et attendre que les données soient renseignées par l'utilisateur au cours de l'exécution du programme.
Concrètement, on peut adapter le programme précédent de sorte que :

```python
from math import *

def equation():
    print("on cherche à résoudre l'équation x**2=a")
    a=float(input('que vaut a ?'))
    x1=sqrt(a)
    x2=-x1
    return x1,x2
```

On retiendra alors que la fonction est définie sans argument, mais que les données seront obtenues à l'aide de la commande`input`. Et on fera très attention puisqu'il est nécessaire de **convertir la chaine de caractère entrée par l'utilisateur**, que ce soit en nombre entier à l'aide de la commande `int` ou en nombre réel à l'aide de la commande `float`.

Présentation des structures usuelles

Parmi les blocs d'instructions les plus courants, on distingue :

1. **les blocs d'instructions conditionnelles**
   Il s'agit des structures conditionnelles qui permettent d'effectuer une série d'instructions selon qu'une certaine condition soit réalisée ou non. Dans le langage Python, on aura recours aux commandes `if`, `elif`, `else`, et la syntaxe d'un tel bloc sera toujours la même :

   `if` *condition 1*:
   indentation Instructions
   `elif` *condition 2*:
   indentation Instructions
   `elif` *condition 3*:
   indentation Instructions
   (...)
   `else`:
   indentation Instructions
   (...)

   Pour séparer les instructions, on pourra encore utiliser le symbole ; ou bien pour faciliter le **debogage**, on pourra préférer un **saut de ligne**.
   De plus, on retiendra que l'instruction `else` n'est pas obligatoire et on veillera à ce que les *conditions* énoncées soient bien des **conditions booléennes**, c'est à dire des tests logiques qui utilisent les opérateurs usuels et ne renvoient que les valeurs `True` ou `False` :

commande Python	Interprétation
`x == y`	x est égal à y
`x != y`	x est différent de y
`x > y`	x est strictement supérieur à y
`x >= y`	x est supérieur ou égal à y
`x in y`	x apparient à y
`x and y`	x et y (logique)
`x or y`	x ou y (logique)
`not x`	non x (logique)

Par exemple, si on souhaite définir la fonction *vabsolue* qui renvoie la valeur absolue d'un nombre réel, on entrera :

```python
def vabsolue(x):
    if x>=0:
        return x
    else:
        return -x
```

Remarque Cette fonction est en fait déjà intégrée au langage, il s'agit évidemment de la fonction `abs`.

2. **les blocs d'instructions répétitives**
Il s'agit des structures itératives qui permettent d'effectuer une série d'instructions un nombre de fois donné ou tant qu'une condition est réalisée. C'est le cas des boucles suivantes :

- la boucle `while` (boucle "tant que") dépendant d'une **condition booléenne** :

  while *condition*:
  indentation Instructions

- la boucle `for` (boucle "pour") associée à une liste donnée, qu'elle soit constituée d'entiers obtenus par la commande `range`, ou constituée de valeurs quelconques :

  for k in range(1,n+1):
  indentation Instructions

  ou encore

  for x in L:
  indentation Instructions

Bien entendu, quand le nombre d'itérations n'est pas déterminé à l'avance, on préfèrera choisir la boucle `while` mais on veillera à ce qu'on puisse sortir de la boucle, c'est à dire qu'à un moment la *condition* énoncée ne doit plus être vraie... On sera donc parfois amené à justifier mathématiquement la **terminaison de nos programmes**.

Remarque Dans la pratique, on fera très attention à ce genre d'erreur, car si la condition d'arrêt n'est jamais atteinte, et qu'un programme tourne sur lui-même, il nous faudra interrompre l'exécution de celui-ci en fermant par exemple la console interactive.

Exercice 1 - Somme des premiers carrés

On souhaite vérifier la formule donnée par Candice :

$$S_n = \sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

Dans le langage Python, construire la fonction *somme* qui pour tout entier n donné, affiche la liste des carrés de 0 à n, puis renvoie le couple $S_n, \frac{n(n+1)(2n+1)}{6}$. On essaiera de procéder de deux façons, à l'aide des boucles `while` et `for`.

Exercice 2 - Un premier jeu

On souhaite construire le jeu suivant : le programme affiche le produit de deux nombres au hasard grâce à la commande `randint` du module **random**, puis l'utilisateur doit renseigner le résultat. Si celui-ci est juste, l'utilisateur gagnera un point.
Le jeu s'arrête après n calculs et renvoie le score du joueur.

1. Dans le langage Python, construire la fonction *jeu* qui pour tout entier n donné, affiche n calculs puis en fonction des réponses de l'utilisateur, renvoie le score obtenu.

2. Modifier votre programme afin que celui-ci renvoie le temps de jeu. On pourra consulter l'aide sur la librairie **time**.

Exercice 3 - Un second jeu

On souhaite construire le jeu suivant : le programme choisit un nombre au hasard x entre 1 et 100. Le joueur doit alors deviner le nombre choisi et tant que celui-ci n'est pas trouvé, l'ordinateur indique si x est au dessus ou en dessous de sa proposition. Le jeu s'arrête quand l'utilisateur a trouvé la valeur de x.

1. Dans le langage Python, construire la fonction *jeu* qui ne prend pas d'argument, mais invite l'utilisateur à deviner la valeur de x tant que celle-ci n'a pas été trouvée.

2. Modifier votre programme afin que celui-ci renvoie le nombre de coups joués par l'utilisateur.