Conditionnelles et boucles

Les commandes Python vues jusqu'à présent ne sont évidemment pas suffisantes pour écrire des programmes. On va donc utiliser des conditionnelles et des boucles.

1 La conditionnelle `if`

Cette instruction permet de n'effectuer une action que si une certaine condition est vérifiée ; plus précisemment si un certain booléen vaut `true`. La syntaxe est la suivante :

```
if booléen:
   instructions 1
instructions 2
```

On note que les deux points " :" et l'indentation avant les instructions sont indispensables.

Quand cette instruction est executée, Python ne traitera les instructions 1 que si le booléen vaut `true` ; sinon, il passera directement aux instructions 2.

Exercice 1. Écrire un programme qui demande un nombre réel à l'utilisateur, et affiche sa racine carrée s'il est positif (on n'affiche rien s'il est strictement négatif).

Exercice 2. Modifier le programme précédent en faisant afficher un message d'erreur dans le cas négatif.

On peut utiliser les variantes suivantes :

```
if booléen:
   instructions 1
else:
   instructions 2
instructions 3
```

qui traite les isntructions 1 si le booléen est vrai, les instructions 2 s'il est faux, et enchaîne dans tous les cas sur les instructions 3.

La syntaxe

```
if booléen1:
   instructions 1
elif booléen2:
   instructions 2
else:
   instructions 3
instructions 4
```

traite les instructions 1 si le booléen1 est vérifié, 2 si le booléen2 est vérifié, et 3 sinon. Dans tous les cas, on passe ensuite aux instructions 4.

Exercice 3. Pour calculer la clef de contrôle d'un numéro de sécurité sociale, on calcule le numéro de sécurité sociale modulo 97 qu'on soustrait à 97.

Écrire un script qui demande un numéro de sécurité sociale et une clef de contrôle, et qui vérifie qu'elle est correcte, et affiche un message d'erreur si elle ne l'est pas.

Exercice 4. Écrire un script qui résout l'équation $ax^2 + bx + c = 0$ en demandant les valeurs de a, b et c à l'utilisateur.

2 La boucle `for`

On peut aussi vouloir effectuer une action plusieurs fois : on utilisera alors une boucle `for`. On l'utilise de la façon suivante :

```
for element in ensemble:
    instructions 1
instructions 2
```

Cette instruction traite la suite d'instructions 1 pour chaque élément de l'ensemble spécifié, puis la suite d'instructions 2.

Il y a différentes façons pour créer des ensembles : une chaîne de caractère est un ensemble de caractères, une liste est un ensemble de valeurs. Un ensemble qu'on utilisera beaucoup sera l'ensemble $\{p, p+1, p+2, \dots, n-1\}$ qu'on obtient avec la commande `range(p,n)`.

Attention au décalage d'indice !

Exercice 5. Écrire un programme qui calcule la factorielle d'un entier demandé à l'utilisateur.

Exercice 6. Écrire un programme qui calcule le n-ième terme de la suite de Fibonacci.

3 La boucle `while`

On souhaite parfois exécuter un programme non pas un nombre défini de fois, mais tant qu'une certaine condition n'est pas vérifiée. On utilise alors la syntaxe suivante

```
while booléen:
    instructions 1
instructions 2
```

Ce programme exécute la suite d'instructions 1 tant que le booléen est vrai, et passe aux instructions 2 sinon.

Attention : il faut bien penser à s'assurer qu'à un moment, le booléen deviendra faux ; sinon on se trouve dans un boucle infinie. On utilisera Ctrl+C pour l'arrêter.

Exercice 7. Écrire un programme qui choisit un entier au hasard entre 0 et 10, et demande au joueur de le deviner. Le jeu continue tant que l'entier n'a pas été deviné.

Exercice 8. Modifier le jeu précédent pour qu'il choisisse un entier entre 0 et 100, et précise à chaque tentative s'il est inférieur ou supérieur au nombre proposé.

Exercice 9. Écrire un programme qui teste si un entier est premier.

Exercice 10. On veut calculer la fraction

$$1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\dots}}}}.$$

On utilise alors la suite définie par

$$u_0 = 1 \text{ et } u_{n+1} = 1 + \frac{1}{u_n}.$$

Écrire un script qui calcule le n-ième terme de la suite. Comparer avec le nombre d'or φ.

Écrire un script qui affiche le nombre d'itérations nécessaire pour approximer φ avec cette fractions à 10^{-n} près.