TP2 - Instructions de contrôle et exercices

1 Instructions de contrôle

Jusqu'à présent, nous avons regardé une exécution **linéaire** (dans l'ordre) des instructions données. Mais ce n'est évidemment pas très utile en pratique sauf peut-être pour faciliter des calculs. On aimerait en effet que l'ordinateur puisse prendre des décisions, qu'il puisse sauter des instructions ou en répéter certaines. C'est à ça que servent les instructions de **contrôle**.

Une instruction de contrôle est une instruction spéciale qui permet d'éviter l'exécution ou au contraire qui permet de forcer l'exécution d'un bloc d'instructions sous certaines conditions. Il y en a trois à retenir :

- **L'instruction conditionnelle `if` :**

```
if condition:
    # instructions si la condition est vraie
else:
    # instructions si la condition est fausse
```

La partie **`else`** est optionnelle.

L'instruction **`if`** permet de conditionner l'exécution d'un bloc d'instructions à la réalisation d'une condition précise. Typiquement, la condition est une comparaison entre deux nombres. On peut utiliser les opérateurs `==` (égal), `!=` (différent), `<` (strictement plus petit), `>` (strictement plus grand), `<=` (plus petit ou égal), `>=` (plus grand ou égal). Par exemple :

```
if delta >= 0:
    print("Il y a au moins une solution.")
else:
    print("Il n'y a pas de solution.")
```

Les blocs sont, comme pour les fonctions, introduits par deux points `:`. Ils doivent être indentés correctement.

Exercice 1

Écrire une instruction conditionnelle permettant d'afficher l'unique solution à $ax^2 + bx + c = 0$ lorsque cette solution existe et est bien unique.

Utilisation avancée : Techniquement, la condition doit être une expression dont le résultat est un booléen (`True` ou `False`). On peut donc écrire des choses du genre :

```
def fonction_compliquee():
    # Plein d'instructions compliquees
    return True

if fonction_compliquee():
    print("La fonction compliquee a reussi.")
```

Ces instructions afficheront `"La fonction compliquee a reussi."` dès lors que `fonction_compliquee` renvoie `True`.

- **La boucle `for` :**

```
for k in sequence:
    # instructions à répéter
```

La boucle **`for`** permet de répéter un bloc d'instructions. Le bloc est répété autant de fois qu'il y a d'éléments dans `sequence`. À chaque itération, la variable indiquée, ici `k`, contient la valeur d'un élément de la séquence. Ainsi `k` va prendre toutes les valeurs de la séquence successivement.

On peut utiliser pleins de types d'objets en tant que séquences (des chaînes de caractères, des listes, des tableaux `numpy`), mais en pratique on utilisera les séquences qui sortent de la fonction **`range`**. **`range`** s'utilise sous 3 formes principales :

- Pour rapidement répéter n fois : **`range(n)`** renvoie la séquence de n nombres de 0 à $n-1$. Ce code affiche les nombres 0 à 9 :

```
for k in range(10):
    print(k)
```

- Pour parcourir les nombres entre deux bornes : **`range(i,n)`** renvoie la séquence de $n-i$ nombres de i à $n-1$. Ce code affiche les nombres 2 à 15 :

```
for k in range(2,16):
    print(k)
```

- Pour parcourir des nombres avec un pas défini : **`range(i,n,s)`** renvoie la séquence de $n-i$ nombres de i à $n-1$ en avançant avec des pas de s. Ce code affiche les nombres impairs de 1 à 17 :

```
for k in range(1,18,2):
    print(k)
```

Exercice 2

Écrire une boucle qui permet de lister l'ensemble des carrés de $1 = 1^2$ à $100 = 10^2$.

- **La boucle `while` :**

```
while condition:
    # instructions à répéter tant que condition est vérifiée
```

Comme la boucle **`for`**, la boucle **`while`** permet de répéter un bloc d'instructions. Mais contrairement à la boucle **`for`**, et de manière semblable à **`if`**, le nombre de répétitions n'est pas définie à l'avance mais est soumis à une condition. Tant que la condition est remplie, le bloc s'exécute en boucle. La condition est vérifiée à chaque fois avant le bloc, et si la condition est vraie, alors le bloc est exécuté en entier. Puis la condition est de nouveau vérifiée et ainsi de suite.

Exemple :

```
n = 1
while n**2 <= 100:
    n = n + 1
print("Le premier carre apres 100 est {}.".format(n))
```

Exercice 3

Écrire une boucle qui permet d'afficher la plus grande solution entière à $x^3 - 5x^2 - 35 \leq 0$.

2 Nombres premiers

Exercice 4

Écrire une fonction d'en-tête :

```
1 def divisible_par_2(n):
```

qui prend un entier naturel `n` *en paramètre et qui renvoie* `True` *si le nombre est divisible par 2 et* `False` *sinon.*

Indication : l'opérateur `%` *permet de calculer le reste d'une division euclidienne.*

Exercice 5

Écrire une fonction d'en-tête :

```
1 def premier(n):
```

qui prend un entier naturel `n` *en paramètre et qui renvoie* `True` *si le nombre est premier et* `False` *sinon.*

Exercice 6

Écrire une fonction d'en-tête :

```
1 def liste_premiers():
```

qui ne prend aucun paramètre et qui affiche la liste des nombres premiers entre 1 *et* 100*.*

3 La conjecture de Syracuse

Exercice 7

Soit $m \in \mathbb{N}^\star$*. On considère la suite* (u_n) *définie par :*

$$\begin{cases} u_0 &= m, \\ \forall n \in \mathbb{N},\ u_{n+1} &= \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases} \end{cases}$$

Écrire une fonction d'en-tête :

```
1 def syracuse(m,n):
```

qui prend un entier naturel non nul m *et un entier naturel* n *en paramètres et qui affiche les* n *premiers termes de la suite* (u_n) *avec* $u_0 = m$*.*
Exécuter la fonction pour différentes valeurs de m *et* n*. Faites une conjecture sur le comportement de la suite* (u_n)*.*

Exercice 8

En admettant la conjecture précédente, écrire une fonction d'en-tête :

```
1 def mesures_syracuse(m):
```

qui prend en paramètre la valuer m *de* u_0 *et qui envoie le couple* (t, a) *constitué du temps de vol* t *(le nombre d'itérations avant d'atteindre* 1*) et de l'altitude maximale* a *(la plus haute valeur atteinte par la suite).*

4 Exercices en vrac

Exercice 9 ⋆

Définir une fonction `surface_trapeze(b1,b2,h)`*. Cette fonction doit renvoyer la surface (l'aire) d'un trapèze de hauteur* `h` *et de bases* `b1` *et* `b2`*.*

Exercice 10 ⋆⋆

1. *Définir une fonction* `maximum2(a,b)` *qui renvoie le plus grand nombre entre* a *et* b*.*
2. *Définir une fonction* `maximum3(a,b,c)` *qui renvoie le plus grand nombre entre* a*,* b *et* c*.*

Exercice 11 ⋆

Écrire une fonction `table(n)` *qui prend en paramètre un entier naturel* n *et qui affiche la table de multiplication de* n*.*

Exercice 12 ⋆⋆

Écrire une fonction `combien_div_2(n)` *qui prend en paramètre un entier naturel* n *et qui renvoie combien de fois ce nombre est divisible par* 2*.*

Exercice 13 ⋆⋆

Écrire une fonction `monnaie(centimes)` *qui prend un entier naturel non nul centimes en paramètre et qui affiche le nombre de pièces de* 2 *euros, de pièces de* 1 *euro et de pièces de* 50, 20, 10, 5, 2 *et* 1 *centimes il faut pour rendre la quantité de centimes* `centimes` *passée en paramètre.*

Exercice 14 - Fizz Buzz ⋆⋆

Écrire une fonction `FizzBuzz(n)` *qui prend en paramètre un entier naturel non nul* n *et qui affiche la liste des entiers de* 1 *à* n *maus en remplaçant les multiples de* 3 *par* `Fizz`*, les multiples de* 5 *par* `Buzz` *et les multiples de* 3 *et* 5 *simultanément par* `FizzBuzz`*.*