chapitre 3

Instructions élémentaires de programmation

Table des matières

I Tests (Instructions conditionnelles) 2
II Boucles 3
2.1 Boucle `for` 4
2.2 Boucle `while` 5
2.3 Boucles Imbriquées 6
III D'autres exercices 6

Le but de ce TP/cours est de découvrir comment travailler avec les structures élémentaires de programmation en python.

Avant de rentrer dans le cœur du sujet, faisons un petit « écart ». L'exemple qui suit montre deux manières distinctes d'écrire un programme qui permet de calculer la valeur absolue d'un nombre.

```
x=float(input("Entrez un réel : "))
if x>=0:
    print(x)
else:
    print(-x)
```

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

La première méthode est interactive, c'est celle que nous avons utilisée lors du premier TP. Dans la deuxième on crée une fonction que nous avons découvert dans le chapitre précédent. C'est le même type d'objet que la fonction `exp` ou `cos` qu'on peut trouver dans le module `math`.

Dans les exercices qui suivent, si rien n'est spécifié (« écrire un programme ») on utilisera au choix, mais avec pertinence, une fonction ou un script.

I Tests (Instructions conditionnelles)

Une structure conditionnelle est composée d'une d'instruction de contrôle puis d'un bloc d'instructions.

En Python, l'instruction de contrôle commence par le mot clef `if` suivi d'une condition à valeur booléenne (True ou False) et se termine par le symbole « : ».

Le bloc d'instructions qui suit s'exécute si et seulement si la condition de l'instruction de contrôle a pour valeur `True`. En Python il est délimité par son indentation qui doit être supérieure à celle du `if`.

Si la valeur de la condition est `False`, on peut proposer l'exécution d'un bloc d'instruction alternatif après le mot clef `else` (sinon) suivi de « : » .

Syntaxe Python

```
if condition:
    Bloc si condition == True
else:
    Bloc si condition == False
```

Il est possible d'avoir plusieurs conditions, comme on l'a vu dans l'exemple sur le programme de résolution des équations du second degré dans le TP précédent. Pour cela, on rajoute un bloc `elif` (contraction de `else if` (sinon si)) pour offrir une autre alternative de traitement conditionnel. Cela fonctionne de la manière suivante : le bloc 2 n'est exécuté que si la condition 1 n'est pas vérifiée et que la condition 2 l'est. Si aucune des deux condition n'est vérifiées alors on exécute le bloc 3.

Syntaxe Python

```
if condition_1:
    Bloc_1
elif condition_2:
    Bloc_2
else:
    Bloc_3
```

Les `elif` et le `else` sont optionnels. Les conditions sont des expressions qui doivent être évaluées en des booléens. Elles sont en général construites avec les opérateurs logiques et les symboles de comparaison.

Exemple 3.1.

```
from math import *
x = float(input('Entrez un réel : \n'))
if x>=-pi and x<=pi:      # -pi<=x<=pi marche aussi
    res = sin(x)
    print('Le résultat est {}'.format(res))
elif x==4 or x==-4:
    res = x**2
    print('Le résultat est {}'.format(res))
else:
    res = 0
    print('Le résultat est {}'.format(res))
print('Les tests sont terminés')
```

Commentez chacune des lignes du programme précédent, comme si vous mettiez un `#`. Que fait ce programme ? Était-ce judicieux d'écrire le programme ainsi ou une fonction aurait-elle était préférable ?

Remarques :

- ⋆ Une condition peut être formée à partir de plusieurs conditions, par exemple `x>=-pi and x<=pi`
- ⋆ Un bloc d'instruction peut tout à fait contenir lui-même des instructions `if` par exemple.
- ⋆ Attention : les `elif` et `else` sont indépendamment facultatifs.
- ⋆ Il peut y avoir plusieurs `elif`
- ⋆ Vous devez impérativement indenter comme dans l'exemple ci-dessus. Cela dit l'éditeur que vous utiliserez indentera automatiquement si vous n'oubliez pas les `:`

Exercice 3.1 L'Indice de Masse Corporelle se calcule par la formule IMC $= \frac{\text{masse}}{\text{taille}^2}$ où la masse est en kilogrammes et la taille en mètres. Un IMC est considéré comme normal s'il est compris entre 18,5 et 25. En dessous de $18,5$, la personne est en sous-poids et au-dessus de 25 elle est en sur-poids.

1. Écrire un programme qui demande la taille et retourne l'intervalle de masse correspondant à un IMC normal.
2. Écrire un programme qui demande la masse et retourne l'intervalle de taille correspondant à un IMC normal.
3. Écrire un programme qui demande la taille et le poids et renvoie si la personne est en sous-poids, sur-poids, ou si son IMC est normal.

Exercice 3.2 *Vérification*

Aller chercher le code de l'exercice sur l'IMC fait en classe à l'endroit qui convient sur le réseau. Exécuter le et vérifier que tout fonctionne correctement.

Exercice 3.3 *A propos des fonctions*

On considère le code suivant (à gauche) et on l'exécute (à droite)

```
1 def mystere(x):
2     if x>15:
3         print("Super ! ")
4     print("Tu peux mieux faire.")
```

```
>>>mystere(10)
...
>>>mystere(18)
...
```

1. Prévoir ce qui va être retourner dans la fenêtre de droite.
2. Vérifier que c'est bien cela en rentrant le code.
3. Conclusion.

Exercice 3.4 *Échange conditionnel de valeurs*

Étant données deux variables a et b dont les valeurs sont entrées par l'utilisateur, écrire un script qui attribue à a la plus petite des deux valeurs et attribue à b la plus grande.

Exercice 3.5 *Maximum*

1. Écrire une fonction `max2` renvoyant le maximum de deux réels.
2. Écrire une fonction `max3` renvoyant le maximum de trois réels.

II Boucles

Une boucle est une construction permettant de répéter de nombreuses fois la même chose. Pour certaines boucles, on sait à l'avance combien de fois il faudra répéter la même opération (boucle inconditionnelle), pour d'autres, ce nombre n'est pas connu à l'avance, il faut imposer une condition d'arrêt (boucle conditionnelle).

2.1 Boucle for

Exemple 3.2.

```
>>> for i in range(5):
        print(i)
0
1
2
3
4
```

```
>>> for x in "BCPST":
        print(x)
B
C
P
S
T
```

```
>>> for a in [4, 9, 12, 17]:
        print(a**2)
16
81
144
289
```

La boucle `for` de Python est assez particulière comparativement aux autres langages. Elle permet de parcourir des objets « itérables », comme des listes ou des chaînes de caractères.

Syntaxe Python

```
for variable in [objet itérable]:
    Bloc
```

Exemple 3.3.

```
1 prenom=input('Entrez votre prénom : ')
2 for lettre in prenom:
3     print(lettre+"/",end=" ")
```

```
>>> (executing lines 1 to 3 of "<tmp 1>")
Entrez votre prénom : Adrien
A/ d/ r/ i/ e/ n/
```

Néanmoins dans la pratique on se servira de la boucle `for` de la manière suivante.

Si on veut répéter un bloc d'instructions un nombre déterminé `n` de fois, on utilise une boucle `for` avec une variable compteur `i` qui va prendre successivement les valeurs de `m` à `n` avec un pas de `s`.

En Python, la fonction `range` crée un itérateur, qui est une sorte de distributeurs d'entiers consécutifs.

Syntaxes Python

```
for i in range(m,n,s):
    Bloc

for i in range(n):
    Bloc
```

- ⋆ `for i in range(n)` : `i` parcourt tous les entiers consécutifs entre `0` et `n-1`. (`n` exclu)
- ⋆ `for i in range(m,n)` `i` parcourt tous les entiers consécutifs entre `m` compris et `n-1`. (`n` exclu)
- ⋆ `range(m,n,s)` `i` parcourt tous les entiers les entiers consécutifs entre `m` compris et `n` exclu avec un pas de `s`..

Exercice 3.6 *Mystère*

Quelle valeur renvoie la fonction suivante si on tape la commande `mystere(3)` ? Que fait ce programme ?

```
1 def mystere(n):
2     u=0
3     for i in range(1,n+1):
4         u=3*u+2
5     return u
```

Exercice 3.7 *Sommes*

Soit $n \in \mathbb{N}^*$. On considère les deux sommes suivantes :

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

1. Déterminer dans chaque cas quelle est la relation de récurrence. Autrement dit exprimer S_{n+1} en fonction de S_n puis I_{n+1} en fonction de I_n.
2. Écrire un programme donnant le terme de rang n des deux suites.

```
>>>S(100)
1.634983...
```

```
>>>I(100)
0.23122...
```

Exercice 3.8

1. Écrire une fonction `puissance` prenant en entrée (x, n) et renvoyant x^n.
   (*Bien entendu il est interdit d'écrire* `x**n`)
2. Écrire une fonction `factorielle` prenant en entrée un entier n et renvoyant $n! = 1.2.3 \dots (n-1)n$.

```
>>>puissance(5,20)
95367431640625
```

```
>>>factorielle(10)
3628800
```

2.2 Boucle `while`

Le propre d'une boucle `for` est que l'on sait à l'avance combien d'itérations il faudra effectuer. Lorsque ça n'est pas le cas, il faut utiliser une boucle `while`.

C'est une structure itérative dont la première instruction est `while condition:` suivie d'un bloc d'instruction. Le bloc est exécuté si la condition est vérifiée (la valeur booléenne de la condition est vraie (`True`)). Puis l'interpréteur remonte à la ligne du `while`, vérifie de nouveau si la condition est réalisée, auquel cas il exécute de nouveau le bloc, etc. Si la condition n'est pas réalisée (sa valeur est `False`) la boucle est achevée.

Syntaxe Python

```
while condition:
    Bloc
```

Attention au risque de boucle infinie avec les boucles `while` si le test d'arrêt n'est jamais vérifié ... Pour que la boucle se termine, il faut nécessairement que la condition devienne fausse à un moment donné. Ainsi, les instructions dans le bloc doivent modifier une variable intervenant dans la condition, sinon, celle-ci ne changera jamais de valeur et le code bouclera alors indéfiniment (sauf si l'on utilise `return` ou `break` mais cela n'est pas naturel).

Exemple 3.4. Que fait le programme suivant ?
(*indic.* `floor` signifie partie entière et `random` donne un nombre au hasard dans $[0, 1[$)

```
1 from math import floor
2 from random import random
3 de=0
4 while de!=6:
5     de=floor(6*random())+1
```

Remarque : Boucle infinie avec `break`.
Parfois (dans les jeux), on utilise des boucles infinies qui tournent tant qu'un événement particulier ne s'est pas produit. En Python, le mot clef `break` permet de sortir d'une boucle `while` directement à partir du corps de boucle.

```
1 password = 'secret'
2 while True:
3     entree = input('Entrez le mot de passe')
4     if entree == password:
5         break
```

Exercice 3.9 Choisissez un nombre à 4 chiffres correspondant au code secret contenu sur la puce d'une carte bancaire par exemple. Écrire un script qui demande à l'utilisateur de rentrer son code de carte bancaire. Le script affiche « `code bon` » si le code donné par l'utilisateur correspond à celui inscrit sur la puce et « `code faux, veuillez recommencer` » si le code n'est pas le bon. Au bout de 3 essais, si le code n'est toujours pas le bon, le script affiche « `code faux, carte inutilisable` ».

Exercice 3.10 *Limite d'une suite*

On sait que la suite définie par $S_n = \sum_{k=1}^{n} \frac{1}{k}$ est croissante et a pour limite $+\infty$.

1. Quelle relation existe-t-il entre S_n et S_{n+1} ?
2. Écrire une fonction `seuilharmonique(M)` qui détermine le plus petit entier n à partir duquel $S_n \geqslant M$, où M est un réel entré par l'utilisateur.

```
>>>seuilharmonique(5)
83
```

```
>>>seuilharmonique(10)
12367
```

2.3 Boucles Imbriquées

Exercice 3.11 On considère la fonction suivante :

```
def imbriquee(n):
    for i in range(n):
        for j in range(n):
            print(i, j)
        print("*")
    print("*")
```

1. Prévoir précisément l'affichage qui sera produit si l'on appelle `imbriquee(2)`.
2. Vérifier !

Exercice 3.12 Écrire des fonctions prenant en entrée un entier n et renvoyant :

1. $\sum_{i=1}^{n}\sum_{j=1}^{n} ij$
2. $\sum_{1\leqslant i<j\leqslant n} (i+j)$

```
>>>rectangle(100)
25502500
```

```
>>>triangle(100)
499950
```

III D'autres exercices

Exercice 3.13

Écrire deux fonctions calculant la somme des carrés des entiers entre 1 et n. L'une utilisera un `while`, l'autre un `for`.

Exercice 3.14

Écrire une fonction `approx_expo` prenant en entrée (x, n) et renvoyant $S_n = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$. (*sans utiliser d'exposant*)

```
>>>approx_expo(4,17)
54.598
```

Exercice 3.15 *Devinette*

On considère le programme suivant :

```
from random import randint
def devinette(n):
    cible = randint(1, n)
    ## choisit la cible au hasrard entre 1 et n
    reponse = int(input("Votre essai ? "))
    ## affiche "Votre essai ?", attend que l'utilisateur rentre une
    ## valeur validée par entrée et convertit cette valeur en entier.
    if reponse == cible:
        print("Gagné !")
    else:
        print("Perdu !")
        print("La réponse était", cible)
```

À propos de `randint` *du module* `random`, *voilà ce qu'on trouve dans l'aide*

`randint(a,b) method of Random instance. Return random integer in range [a,b], including both end points.`

1. Modifier cette fonction pour que l'ordinateur réponde «trop grand» ou «trop petit» suivant les cas et permette à l'utilisateur de jouer jusqu'à ce qu'il trouve.
2. Faire en sorte que la fonction affiche le nombre d'essais utilisés par le joueur (quand il finit par trouver).
3. Faire une dernière modification pour que l'utilisateur puisse choisir au début le nombre d'essais maximum dont il disposera. S'il ne trouve pas au bout de ce nombre d'essais, le jeu s'arrête.

Exercice 3.16 *Test de primalité*

1. On rappelle qu'un nombre (supposé entier et positif) est dit **premier** s'il a exactement deux diviseurs distincts (1 et lui-même). Les premiers nombres premiers sont donc $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots$.
   Écrire une fonction `est_premier(n)` qui renvoie `True` si n est premier, `False` sinon.

   indic. On pourra utiliser l'opérateur `%` qui donne le reste de la division euclidienne et remarquer que a est divisible par b ($a, b \in \mathbb{N}$) si et seulement si `a % b` vaut 0.
2. 51 552 127 est-il premier ? Et 9 999 991 ?
3. Combien de divisions votre fonction effectue-t-elle pour déterminer si 24 est premier ? si 17 est premier ? Pouvez-vous faire mieux (c'est-à-dire moins...) ?

Exercice 3.17 *Quel jour sommes nous ?*

Dans cet exercice on ne considèrera que des années postérieures à 1582 [1].

1. Les années bissextiles sont les années non séculaires divisibles par 4 ou les années séculaires divisibles par 400. Écrire une fonction `bissextiles` qui prend en entrée une année n et qui affiche en sortie si elle est bissextile.

```
>>>bissextile(2012), bissextile(2100)
(True,False)
```

   indic. Réfléchissez bien aux conditions, et pensez à utiliser l'opérateur `%`.
2. Sachant que le premier janvier 2013 est tombé un mardi, écrire une fonction retournant le jour de la semaine où tombe le premier janvier d'une année. On prendra comme convention : 0 pour lundi, 1 pour mardi, ...jusqu'à 6 pour dimanche.

```
>>>premierjanvier(1950), premierjanvier(2050)
(6,5)
```

   indic. Regardons ce qui se passe si on demande pour une année supérieure à 2013 (on adaptera dans le cas contraire). $365 = 7 \times 52 + 1$ et $366 = 7 \times 52 + 2$, conclusion une partie du code pourrait ressembler à cela :

```
if n>2013:
    for annee in range(2013,n):
        if bissextile(annee):
            j+=2
        else:
            j+=1
    return (j % 7)
```

   Bien entendu il y a des lignes de code avant après cela...
3. Écrire une fonction prenant en entrée une date (triplet d'entiers (jour,mois,année)) et retournant le jour dans la semaine correspondant à cette date.

   indic. On pourra créer un tableau donnant le nombre de jours pour chaque mois de l'année (en faisant attention au mois de février) une partie du code pourrait ressembler à cela :

```
.......
durees=[31,28,31,30,31,30,31,31,30,31,30]
    if bissextile(annee):
        durees[1] += 1 # On rajoute un jour au mois de février
    for m in range(mois-1):
        j+=durees[m]
.......
```

   Le 12 juillet 1998 était un dimanche, le 8 mai 1945 était un mardi.

1. https://fr.wikipedia.org/wiki/Calendrier_gregorien