I Branchement conditionnel

I.1 Le test Si ··· alors ···

Le test si est l'instruction conditionnelle la plus simple. Il permet d'exécuter ou non une liste d'instructions selon qu'une condition s'avère vraie ou fausse.
En Python, il se traduit avec l'instruction **if**.

I.1.1 Syntaxe

La syntaxe ci-dessous permet un test conditionnel :

```
if condition :
    Bloc d'instructions à exécuter      # Indentation obligatoire
# On sort du test en stoppant l'indentation
```

I.1.2 Exemple

Dans l'éditeur, ouvrez une nouvelle page, puis créez la fonction suivante :

```
1 def parite(n) :
2     """ Cette fonction teste si un entier donné est pair et renvoie True dans ce cas."""
3     if n%2 == 0 :
4         print(str(n)+' est pair')
5         return True
6     print(str(n)+' est impair')
```

Enregistrez cette fonction dans un fichier, puis retournez dans la console, exécutez le fichier et tapez les instructions suivantes :

☞ $parite(6)$ ☞ $parite(61)$ ☞ $test = parite(6)$ ☞ $type(test)$
☞ $test2 = parite(61)$ ☞ $type(test2)$ ☞ $test2 == True$ ☞ $test2 == False$

Dans notre fonction, il n'y a qu'un seul **return** et celui-ci se trouve à l'intérieur du test. Si la condition n'est pas réalisée, la fonction ne rencontre pas de **return**. Du coup lorsqu'on l'exécute, le résultat n'a pas de type. L'affichage ne suffit pas, il faut conclure la fonction par un **return**

I.2 Le test Si ··· alors ··· sinon ···

Le test si/sinon enrichit le test simple du paragraphe précédent en offrant une alternative, c'est-à-dire la possibilité d'exécuter un bloc d'instructions si la condition est vraie ou un autre bloc d'instructions si cette condition est fausse.

I.2.1 Syntaxe

Si est traduit par **if** en Python tandis que sinon l'est par **else**

```
if condition :
    Bloc d'instructions 1 à exécuter      # Indentation obligatoire
else :
    Bloc d'instructions 2 à exécuter      # Indentation obligatoire
# On sort du test en stoppant l'indentation
```

I.2.2 Exemple :

Complétez la fonction *parite* précédente pour qu'elle renvoie désormais **True** si l'entier n donné est pair et **False** sinon.

I.3 Le test Si ··· alors ··· sinon si ··· alors ··· ··· sinon ··· alors ···

Le test si/sinon s'utilise lorsqu'on doit choisir entre une condition et son contraire. Mais, lorsqu'on doit faire un choix entre plus de deux possibilités, il est nécessaire d'utiliser une structure polyconditionnelle plus riche où le sinon est remplacé par un ou plusieurs sinon si.

I.3.1 Syntaxe

Sinon si est traduit par **elif** en Python, abréviation de **else if**

```
if condition1 :
    Bloc d'instructions 1 à exécuter      # Indentation obligatoire
elif condition2 :
    Bloc d'instructions 2 à exécuter      # Indentation obligatoire
⋮
else :
    Bloc d'instructions n à exécuter      # Indentation obligatoire
# On sort du test en stoppant l'indentation
```

Idéalement, les conditions booléennes qui apparaissent derrière si et les sinon si sont mutuellement exclusives (c'est-à-dire qu'elles ne se recoupent pas).

I.3.2 Exemple

Ecrire et testez la fonction suivante :

```
1  def competence(note) :
2      """ Cette fonction traduit en français les compétences évaluées en
       lettre ou chiffre."""
3      if note =='A'  or  note == 1 :
4          return 'ACQUIS'
5      elif note =='AR'  or  note == 2 :
6          return 'A RENFORCER'
7      elif note =='ECA'  or  note == 3 :
8          return "EN COURS D'ACQUISITION"
9      else :
10         return 'NON ACQUIS'
```

Ecrire ce script dans l'éditeur, puis exécutez-le :

```
1 print("Entrer un nombre positif plus petit que 100.")
2 nb = float(input())
3 if nb < 0 :
4     print("Ce nombre ne convient pas car il est négatif.")
5 elif nb < 100 :
6     print("Ce nombre convient.")
7 else  :
8     print("Ce nombre est trop grand.")
```

A noter que dans le dernier **sinon** il n'est pas nécessaire de donner la condition puisque c'est la négation de toutes les autres.

II Boucles for et while

II.1 La boucle for

La boucle pour est l'instruction itérative qui permet de répéter un bloc d'instructions un nombre de fois prédéfini. Algorithmiquement, cela donne :

pour k décrivant *un ensemble de valeurs* faire
    ce bloc d'instructions
fin du pour

La lettre k est appelée la variable de boucle. Le bloc d'instructions peut faire intervenir ou non la variable de boucle k. Dans le cas où elle n'y intervient pas, elle permet simplement de comptabiliser combien de fois le bloc d'instructions est répété.

II.1.1 La boucle pour avec un compteur

La syntaxe ci-dessous est utilisée pour une boucle pour :

```
for  k  in range(a,b) :
    Bloc d'instructions à exécuter      # Indentation obligatoire
# On sort de la boucle en stoppant l'indentation
```

À la compilation, le bloc d'instructions est ainsi effectué pour toutes les valeurs de k comprises entre a et b-1. On dit alors que la variable de boucle est un compteur. Le

nombre d'itérations est, dans ce cas, connu : il vaut b-a.
Rappelons, d'une part, que range(n) parcourt les entiers entre 0 et n-1 et, d'autre part, que range(a,b,r) permet de parcourir, avec le pas r, les entiers compris entre a et b-1.

Exemples :

1. Dans l'éditeur, tapez le script suivant, puis exécutez-le :

```
for k in range(4) :
    print(k)
```

2. Dans l'éditeur, créez la fonction suivante :

```
def sommecube(n) :
    """ Cette fonction calcule et renvoie la somme des n premiers cubes, 1**3+2**3+...+n**3 """
    s=0 # Initialisation
    for k in range(n+1) :
        s=s+k**3 # ou s+=k**3
    return s
```

Enregistrez cette fonction, puis retournez dans la console, compilez le fichier et tapez les instructions suivantes :

☞ *sommecube*(3) ☞ *s* ☞ $t = sommecube(3)$ ☞ *t*

II.1.2 La boucle pour avec un itérable

Un itérable est une donnée informatique susceptible d'être parcourue. On pense bien sûr aux intervalles d'entiers (range) mais aussi aux chaînes de caractères (parcourues caractère par caractère) ou aux listes (parcourues élément par élément).
Au paragraphe précédent, nous avons vu que la variable d'une boucle for peut parcourir un intervalle d'entiers. Mais ce n'est pas la seule possibilité ! En fait, de façon générale, cette variable de boucle peut parcourir n'importe quel itérable. La syntaxe, très naturelle, est alors de la forme :

```
for k in iterable :
    Bloc d'instructions à exécuter      # Indentation obligatoire
# On sort de la boucle en stoppant l'indentation
```

L'itérable peut être une chaîne de caractère ou une liste.
Exemple : Dans l'éditeur, créez la fonction suivante :

```
def presencelettredansmot(lettre,mot) :
    """ Cette fonction renvoie True si la lettre appartient au mot, False sinon """
    for le in mot :
        if le==lettre : # on teste si le lettre le en cours est égale à la lettre cherchée
            return True # on peut sortir de la fonction, la lettre appartient bien au mot
    return False
```

Exécutez le fichier et tapez les instructions suivantes :

☞ *presencelettredansmot*('y',' *sapristy*') ☞ "*y*" *in* "*sapristy*"

II.2 La boucle Tant que :

La boucle tant-que est l'instruction conditionnelle qui permet de répéter un bloc d'instructions tant qu'une condition donnée reste vraie. Elle peut ainsi être considérée comme une exécution répétée du test si. Algorithmiquement, cela donne

Tant que *une condition reste vraie* faire
    ce bloc d'instructions
fin du tant que

Au contraire de la boucle pour, la boucle tant que ne nécessite pas de connaître à l'avance le nombre d'itérations du bloc d'instructions que l'on souhaite effectuer. Toutefois, sans être a priori connu, ce nombre d'itérations doit rester fini, afin d'éviter que le processus ne tourne indéfiniment. Pour cela, il faut s'assurer qu'une instruction provoque à terme l'arrêt de la boucle. Dans la plupart des cas, le bloc d'instructions agit sur la condition, de sorte que celle-ci, vraie au départ, devienne fausse (après une ou plusieurs itérations).

La syntaxe ci-dessous est utilisée pour une boucle tant que :

```
while condition :
    Bloc d'instructions à exécuter      # Indentation obligatoire
# On sort de la boucle en stoppant l'indentation
```

En lançant une telle boucle, on obtient l'exécution répétée du bloc d'instructions tant que condition reste une variable booléenne vraie. Dès que condition tombe en défaut, Python s'arrête !
Comme signalé ci-dessus, le nombre de tours de boucle dans un while est a priori inconnu. Nous verrons que cela offre à while plus de souplesse qu'à for mais que cela nécessite aussi plus de maîtrise (pour éviter les " boucles infinies ").

Exemple : Dans l'éditeur, tapez les instructions suivantes puis exécutez les :

```
1 a=0
2 while a< 100 :
3       a+=3
4       print(a)
```

Vous pouvez constatez que la boucle continue jusqu'à ce que la variable a dépasse 100.
⚠ le risque est d'écrire une condition qui reste toujours vraie.

Exemple : Dans l'éditeur, tapez les instructions suivantes puis exécutez les :

```
1 a=0
2 while a>= 0 :
3       a=2*a+1
4       print(a)
```

Aïe, aïe, aïe, la boucle ne s'arrête pas. Pas de panique, Python a tout prévu. Il faut appuyer simultanément sur les touches **Ctrl** + **I** pour interrompre le script.

II.3 for ou while ?

La boucle itérative **for** est une boucle "active" au contraire de son homologue **while**. Cela signifie qu'en plus de la répétition d'un bloc d'instructions (action commune aux deux types de boucle), la boucle **for** gère l'évolution d'une variable dans un itérable.
Ainsi, la commande **for k in itérable :** s'occupe, de façon autonome, de l'évolution de k alors que **while k in itérable :** se contente de vérifier que k est présent dans l'itérable sans jamais prendre l'initiative de faire évoluer la valeur de k.
Cette opposition de style entre **for** et **while** a deux conséquences essentielles. L'une est, disons, à l'avantage du **for** ; l'autre à l'avantage du **while**.

•Simplicité du **for** : L'autonomie du **for** dans la gestion de la variable de boucle (incrémentation automatique et assurance de terminaison) sont des arguments de simplicité en faveur de l'utilisation du **for**.

> lorsqu'on connaît à l'avance le nombre de répétitions à effectuer, on utilise la boucle for. Dans le cas contraire, on utilise la boucle while

•Souplesse du **while** : La boucle **for** ne peut être utilisée que dans le cas où le nombre d'itérations à effectuer est connu à l'avance. Au contraire, la boucle **while** s'utilise indifféremment dans tous les cas, que l'on connaisse ou non le nombre de tours de boucle à accomplir.
Mais cette souplesse a un prix : mettre en place une boucle **while** est généralement plus compliqué que dans le cas d'un **for**.
En particulier, en cas d'utilisation d'un compteur dans une boucle **while**, il est nécessaire de gérer "à la main" ce compteur. Autrement dit, il convient d'initialiser le compteur en amont de la boucle et d'incrémenter ce compteur à l'intérieur de la boucle.
Voici deux fonctions calculant la somme des n premiers carrés, la première version est ici plus simple :

```
1 def sommecarre(n) :
2     """ Cette fonction calcule et renvoie la somme des n premiers carrés, 1**2+2**2+...+n**2 """
3     s=0 # Initialisation
4     for k in range(n+1) : # k va prendre les valeurs de 0 à n
5         s=s+k**2 # ou s+=k**2
6     return s
```

```
1 def sommecarre2(n) :
2     """ Cette fonction calcule et renvoie la somme des n premiers carrés, 1**2+2**2+...+n**2 """
3     s=0 # Initialisation de la somme s
4     k=0 # Initialisation de l'indice de la somme k
5     while k<=n : # k va prendre les valeurs de 0 à n
6         s=s+k**2 # ou s+=k**2
7         k+=1 # il faut penser à incrémenter k
8     return s
```

III Exercices

III.1 Tests

Exercice 1: Dans un club de rugby, il y a quatre groupes :
•celui des 8-9 ans, nommé "U9",
•celui des 10-11 ans, nommé "U11",
•celui des 12-13 ans, nommé "U13"
•celui des 14-16 ans. nommé "U16".
Écrire un script qui demande à l'utilisateur quel est l'âge de la personne et qui affiche alors à quel groupe il appartient.

Exercice 2: Ecrire une fonction qui étant donné un réel a renvoie sa valeur absolue.

Exercice 3: Ecrire une fonction qui étant donné un entier n correspondant à une année renvoie **True** si cette année est bissextile et **False** sinon.

Exercice 4: L'Indice de Masse Corporelle est le rapport de votre poids (exprimée en kg) par le carré de votre taille (exprimée en m). Il permet de déterminer la corpulence d'une personne. En dessous de 18, vous souffrez de maigreur ; entre 18 et 25, votre corpulence est idéale ; entre 25 et 30, vous avez du surpoids et au delà de 30, vous souffrez d'obésité. Écrire une fonction qui calcule votre IMC et précise votre corpulence.

Exercice 5: Ecrire une fonction $maximum(a, b)$ qui détermine la valeur maximale de 2 réels a, b.
Les variables d'entrée sont a et b, des réels quelconques.
La variable de sortie contient $max(a, b)$.

Exercice 6: Ecrire une fonction $degre1(a, b)$ qui discute et résoud l'équation de degré 1 : $ax + b = 0$
Les variables d'entrée sont a et b, les coefficients de l'équation.
La variable de sortie est x qui contient la solution éventuelle.
S'il n'y a pas de solution, on l'écrira et l'on ne retournera rien **return**().
Idem, si tous les réels sont solutions.

Exercice 7: Ecrire une fonction $degre2(a, b, c)$ qui discute et résout l'équation de degré 2 dans $\mathbb{R}$: $ax^2 + bx + c = 0$
Les variables d'entrée sont a, b et c, les coefficients de l'équation.
On retournera la (les) solutions éventuelles.

III.2 Boucles

Exercice 8: Ecrire un script affichant tous les entiers pairs de 0 à 20.

Exercice 9: Écrire un script demandant à l'utilisateur de rentrer un nombre n et affiche

1. la liste des carrés des nombres inférieur ou égaux à ce nombre
2. la somme des carrés des nombres inférieur ou égaux à ce nombre.

Exercice 10: Ecrire un script qui permet d'afficher un à un les 64 codons rencontrés en génétique (3-listes de lettres prises dans l'ensemble $\{A, C, G, T\}$).

Exercice 11: Soit m un entier naturel non nul. Écrire un programme qui affiche la valeur de la somme : $\sum_{k=1}^{n} k^3$

Exercice 12: Ecrire une fonction factorielle(n) qui étant un entier n donné, renvoie la valeur de $n!$.

Exercice 13: Soit a et b deux entiers naturels.

1. Écrire une boucle affichant et incrémentant (+1) la valeur de a tant qu'elle reste inférieure à b.
2. Écrire une boucle décrémentant (-1) la valeur de b, affichant sa valeur si elle est impaire, jusqu'à ce que b soit nul.

Exercice 14: Un jeu. Ecrire un script qui génère un nombre au hasard entre 0 et 100 et qui demande à l'utilisateur de trouver ce nombre. On affichera "Trop petit" ou "Trop grand" selon le cas et l'on demandera à l'utilisateur de chercher tant qu'il n'a pas trouvé.

Exercice 15: Ecrire un script qui simule les lancers successifs d'un dé à 6 faces jusqu'à obtenir la face 6. On pourra afficher à chaque étape le résultat du dé ainsi que le numéro du lancer.
Ecrire alors une fonction apparitionmoyenne(n) qui simule n fois l'expérience précédente et renvoie le rang d'apparition moyen du premier 6.