Sommaire

I. La boucle "pour"

La boucle pour est utilisée quand on connait le nombre de fois où l'on doit répéter une suite d'instructions.

Structure de la boucle "pour" en pseudo code :

Pour i allant de 0 à n-1
  faire :
    ...
    les différentes instructions à répéter
    ...
Fin pour

Pour illustrer ce cours nous allons partir sur un premier exemple : Obtenir la somme des n premiers entiers naturels.
Pour n=4, on veut écrire un programme qui renvoie la valeur de 0+1+2+3=6.

Ecriture du programme de l'exemple en pseudo code :

1. S← 0
2. Pour i allant de 0 à n-1
3. faire :
4. S ← S+i
5. Fin pour

La structure de la boucle "pour" en python

La première ligne fait que la suite du code va être répétée n fois en incrémentant de manière automatique la variable i de 0 à n-1.
Dans un premier temps l'attribut dans `range` sera un entier, nous verrons dans la suite qu'il peut en être autrement.

---

for i in range(n):
    ...
    les différentes instructions à répéter
    ...

On remarque ici l'absence de "faire" et de "fin pour". C'est en fait l'indentation qui sert de limitateur de boucle. L'indentation en python n'est pas uniquement un élèment d'hygiène de programmation mais un élément faisant partie intégrante des structures.

Ecriture du programme de l'exemple en python :

In [6]:

```
def somme(n:int)->int: #les int sont ici pour indiquer le typage des variables d'entrée et de sortie. Ici l'attribut est de type int et la sortie également
    """La fonction somme renvoie la somme des n+1 premiers entiers naturels.
    la fonction somme a pour attribut un entier (int) : n et renvoie un autre entier :S
    Pour n=4 : S=0+1+2+3=10"""
    S=0 #initialisation de la somme à 0
    for i in range(n):

        S=S+i
    return S
somme(20)
```

Out[6]:

```
190
```

In [35]:

```
somme(20)
```

Out[35]:

```
190
```

Cliquer sur cette adresse : http://pythontutor.com/visualize.html#mode=edit
Copier/coller votre code dans la fenêtre
puis cliquer sur `Visualize Execution` pour observer la trace d'execution de votre code pas à pas puis par des clics succéssifs sur le bouton `forward`.

Exercices

Exercice

Modifier la fonction `somme(n)` pour qu'en sortie on ait la somme des n+1 premiers entiers naturels.
pour n=4, S=0+1+2+3+4=10

In [1]:

```
def somme(n:int)->int: #les int sont ici pour indiquer le typage des variables d'entrée et de sortie. Ici l'attribut est de type int et la sortie également
    """La fonction somme renvoie la somme des n+1 premiers entiers naturels.
    la fonction somme a pour attribut un entier (int) : n et renvoie un autre entier :S
    Pour n=4 : S=1+2+3+4=10"""
    S=0 #initialisation de la somme à 0
    for i in range(n+1):

        S=S+i
    return S
```

In [2]:

```
#test de la fonction somme avec l'attribut 4
somme(4) #la réponse doit être 10
```

Out[2]:

```
6
```

Exercice

Ecrire une fonction `TableMulti()` qui reçoit en entrée un nombre entier `n` compris entre 1 et 10 et affiche en sortie la table de multiplication de ce nombre. Par exemple, si l'algorithme reçoit en entrée le nombre 7, il affichera la table de multiplication.

1 fois 7 = 7
2 fois 7 = 14
3 fois 7 = 21
etc...

In [27]:

```
#Aide à l'affichage du résultat:
i=5
n=8
print("La multiplication de", i, "par", n, "est",i*n   )
```

```
La multiplication de 5 par 8 est 40
```

In [86]:

```
def tableMulti(n:int)->None:
    """Renvoie la table de multiplcation de n"""
    for i in range(10):
        print("La multiplication de", i, "par", n, "est",i*n )
    return None
```

In [87]:

```
#test de la fonction :
tableMulti(5)
tableMulti(9)
```

```
La multiplication de 0 par 5 est 0
La multiplication de 1 par 5 est 5
La multiplication de 2 par 5 est 10
La multiplication de 3 par 5 est 15
La multiplication de 4 par 5 est 20
La multiplication de 5 par 5 est 25
La multiplication de 6 par 5 est 30
La multiplication de 7 par 5 est 35
La multiplication de 8 par 5 est 40
La multiplication de 9 par 5 est 45
La multiplication de 0 par 9 est 0
La multiplication de 1 par 9 est 9
La multiplication de 2 par 9 est 18
La multiplication de 3 par 9 est 27
La multiplication de 4 par 9 est 36
La multiplication de 5 par 9 est 45
La multiplication de 6 par 9 est 54
La multiplication de 7 par 9 est 63
La multiplication de 8 par 9 est 72
La multiplication de 9 par 9 est 81
```

Exercice

A la naissance d'Apolline, son grand-père Joël, lui ouvre un compte bancaire. Ensuite, à chaque anniversaire, le grand père d'Apolline verse sur son compte 100 €, auxquels il ajoute le double de l'âge d'Apolline.

Par exemple, lorsqu'elle a deux ans, il lui verse 104 € ce qui lui fait un total de 100+102+104=306 € à la deuxième année.

Ecrire une fonction `TresorApo()` qui en entrée reçoit un entier `n` et en sortie la somme présente sur le compte d'Apolline à sa

nième année.

In [92]:

```
def tresorApo(n:int)->int:
    tresor=0 #initialisation du trésor d'Apolline
    for i in range(n+1):
        tresor=tresor+100+2*i
    return tresor
```

In [91]:

```
#teste de la fonction :
tresorApo(2) #la valeur de sortie doit être 306
tresorApo(10) #la valeur de sortie doit être 1210
```

Out[91]:

```
1210
```

Exercice
L'instruction `range` peut prendre différents attributs. Dans cet exercice nous allons utiliser l'instruction :

```
for i in range(depart,arrivée,pas):
```

qui permet de repéter la boucle et d'incrémenter `i` de `départ` à `arrivée-1` avec une incrémentation de `pas`.
Les finances de Joël ne sont pas au mieux, il décide de ne donner à Apolline qu'une année sur deux en conservant la même règle.
A deux ans Apolline aura donc :100+104=204 €
Reprendre l'énoncé de l'exercice I.5.3 en écrivant une fonction `tresorApoB()` qui prend en entrée un entier `n` et renvoie un entier.

In [102]:

```
def tresorApoB(n:int)->int:
    tresor=0 #initilisation du trésor d'Apolline
    for i in range(0,n+1,2):
        tresor=tresor+100+2*i
    return tresor
```

In [104]:

```
#Test de la fonction
tresorApoB(3) #la valeur de sortie doit être 204
```

Out[104]:

```
204
```

Exercice

Dans cet exercice nous allons donner à l'instruction `range` un attribut de type `list`.
Nous allons écrire une fonction `estDansListe` qui prend en entrée une liste `lst` et un entier `n` et qui renvoie `True` si l'entier est dans la liste et `False` sinon.
Pour cela nous allons utiliser l'instruction :

```
for i in lst:
```

Qui va répéter les instructions autant de fois que d'éléments dans la liste et qui à chaque étape va donner à `i` une valeur de la liste `lst` en commencant par la première.

Indication : Vous devez utiliser une boucle conditionnelle "Si"

In [79]:

```
def estDansListe(lst:list,n:int)->bool:
    test=False
    for i in lst:
        if test==True or i==n :
            test=True
    return test
```

In [80]:

```
#test de la fonction
l=[0,1,2,3,4,5]
estDansListe(l,8) #doit renvoyer False
estDansListe(l,5) #doit renvoyer True
```

Out[80]:

```
True
```

La boucle "tant que"

La boucle "tant que " est utilisée quand on ne connait pas le nombre de fois où l'on doit répéter une suite d'instructions.
Elle est souvent utilisée également pour remplacer une boucle dans un souci d'optimisation du code.
Une autre différence majeure avec la boucle pour est l'absence d'incrémentation automatique.

Structure de la boucle "tant que" en pseudo code :

Tant que "condition : booleen"
  faire :
    ...
    les différentes instructions à répéter
    ...
Fin pour

Pour illustrer ce cours nous allons partir sur le premier exemple vu précédemment: Obtenir la somme des n premiers entiers naturels
Pour n=4, on veut écrire un programme qui renvoie la valeur de 0+1+2+3=6.

Ecriture du programme de l'exemple en pseudo code :

1. S← 0
2. i← 0
3. Tant que i<4
4. faire :
5. i ← 1+i
6. S ← S+i
7. Fin tant que

La structure de la boucle "tant que" en python

La première ligne fait que les instruction dans la boucle vont être répéter tant que les conditions indiquées derrière le `while` sont vérifiées.
L'écriture des conditions est ici simple, elle peut être composée de plusieurs conditions. Cela offre des possibilités de codes intéressantes et parfois complexes.
Remarque : Dans les premiers exemples la condition est simple et on pourrait remplacer cette boucle `while` par une boucle `for`.

While i<4:
    ...
    les différentes instructions à répéter

les différentes instructions à répéter
...

On remarque ici l'absence de "faire" et de "fin tant que". Comme pour la boucle pour c'est en fait l'indentation qui sert de limitateur de boucle. On rappelle encore que l'indentation en python n'est pas uniquement un élèment d'hygiène de programmation mais un élément faisant partie intégrante des structures.

Ecriture du programme de l'exemple en python :

In [36]:

```
def somme(n:int)->int:
    """"La fonction somme renvoie la somme des n+1 premiers entiers naturels.
    la fonction somme a pour attribut un entier (int) : n et renvoie un autre entier :S
    Pour n=4 : S=0+1+2+3=10"""
    S=0 #initialisation de la somme à 0
    i=0
    while i<n:
        i+=1 #on aurait pu écrire aussi i=i+1; incrémente i de 1 à chaque passage dans la boucle

        S=S+i
    return S
```

In [ ]:

```
#Test de la fonction :
somme(4)
```

Cliquer sur cette adresse : http://pythontutor.com/visualize.html#mode=edit
Copier/coller votre code dans la fenêtre
puis cliquer sur `Visualize Execution` pour observer la trace d'exécution de votre code pas à pas puis par des clics succéssifs sur le bouton `forward`.

Exercices

Exercice

Modifier la fonction `somme(n)` pour qu'en sortie on ait la somme des n+1 premiers entiers naturels.
pour n=4, S=0+1+2+3+4=10

In [70]:

```
def somme(n:int)->int:
    """"La fonction somme renvoie la somme des n+1 premiers entiers naturels.
    la fonction somme a pour attribut un entier (int) : n et renvoie un autre entier :S
    Pour n=4 : S=0+1+2+3+4=10"""
    S=0 #initialisation de la somme à 0
    i=0
    while i<n:
        i+=1 #on aurait pu écrire aussi i=i+1; incrémente i de 1 à chaque passage dans la boucle

        S=S+i
    return S
```

In [71]:

```
#Tester la fonction avec n=4, vous devriez avoir 10 en sortie
somme(4)
```

Out[71]:

```
10
```

Exercice

Une entreprise vend un récupérateur d'eau de pluie qui a une capacité de 150 litres d'eau.
Elle voudrait proposer à ses clients un programme qui permettrait au client de savoir en combien de jours le réservoir sera plein en fonction du remplisage moyen quotidien.
Ecrire une fonction `full` qui prend en argument une entrée du type `float` indiquant le volume en litre du remplisage moyen quotidien `a` et en sortie un entier indiquant le nombre de jours nécessaire au remplissage.

In [51]:

```
def full(a:float)->int:
    j=0 #initialisation du nombre de jours
    v=0 #volume d'eau dans le récupérateur
    while v<150 :
        v=v+a #le volume d'eau augmente de a
        j=j+1 #le nombre de jorus augmente de 1
    return j
```

In [56]:

```
#test de la fonction
full(3.1) #la réponse devrait être 49
```

Out[56]:

```
49
```

Exercice

Ecrire une fonction qui permet d'écrire une liste d'entiers aléatoires qui s'arrête dès que la somme des nombres dépasse une certaine valeur.
la fonction `makeList` prend en entrée 3 valeurs entières `deb`, `fin` et `max` correspondant respectivement à la plus petite valeur des nombres aléatoires, à la plus grande et à la valeur à ne pas dépasser en faisant la somme de ces nombres.

In [5]:

```
#à interpréter en premier
from random import  *
```

Des instructions nécessaires à la réalisation du programme :

In [6]:

```
#obtenir un nombre entre deux entiers, 10 et 25 par exemple
randint(10,25)
```

Out[6]:

```
17
```

In [3]:

```
#pour créer une liste vide qui s'apelle lst on écrit :
lst=[]
#pour rajouter un élément à cette liste par exemple 5 on écrit :
lst.append(5)
print(lst)
```

```
[5]
```

In [74]:

```
def makeList(deb:int,fin:int,max:int)->list:
    lst=[]
    s=0
    while s<max:
        a=randint(deb,fin)
        lst.append(a)
        s=s+a
    return lst
```

In [78]:

```
makeList(2,50,250)
```

Out[78]:

```
[3, 29, 16, 15, 13, 36, 2, 25, 36, 10, 39, 49]
```

Exercice
Reprendre l'exercice I.5.5 en utlisant une boucle while

In [111]:

```
l=[0,1,2,3,4,5]
#Pour déterminer la longueur d'une liste on écrit :
len(l)
```

Out[111]:

```
6
```

In [112]:

```
#Attention aux indices !
l[5]
```

Out[112]:

```
5
```

In [113]:

```
def estDansListe(lst:list,n:int)->bool:
    i=0 #initialise le pointeur dans la liste lst
    test=False
    while i<=len(lst)-1 and lst[i]!=n :
        i+=1
    if i==len(lst)-1:
        test=True
    return test
```

In [115]:

```
#test de la fonction
l=[0,1,2,3,4,5]
estDansListe(l,8) #doit renvoyer False
estDansListe(l,5) #doit renvoyer True
```

Out[115]:

```
False
```

Exercice
Écrire une fonction qui demande à l'utilisateur d'entrer des notes, sans que l'on sache à l'avance combien de notes il va saisir : l'arrêt de la saisie se produit dès que l'utilisateur saisit un nombre négatif.

Le programme doit calculer et afficher la moyenne (Attention: le nombre negatif saisi ne doit pas compter dans la moyenne !). On nommera cette fonction `moy` elle ne prendra aucune entrée et donnera un élément de type `float` en sortie.

In [ ]:

```
#demander au joueur un nombre et l'affecter à une variable :
a=input("Donner un nombre")
print(a)
#la valeur renvoyée par input est de type str, il faudra donc écrire :
a=int(input("Donner un nombre"))
```

In [63]:

```
def moy()->float:
    note=0 #initialisation de la variable note
    eff=0 #initialisation de l'effectif
    lst=[] #création d'une liste vide qui contiendra les notes
    moy=0 #intialisation de la moyenne
    while note>=0:
        note=int(input("Donner une note"))
        if note>=0:
            moy=(moy*eff+note)/(eff+1)
            eff+=1
            lst.append(note)
    print("la moyenne de la liste des notes :", lst, "est" , moy)
    return None
```

In [64]:

```
#test de la fonction
# essayer 10,12,14,8,12,6,8
#la sortie devrait être 10
moy()
```

```
Donner une note10
Donner une note12
Donner une note14
Donner une note8
Donner une note12
Donner une note6
Donner une note8
Donner une note-1
la moyenne de la liste des notes : [10, 12, 14, 8, 12, 6, 8] est 10.0
```

Exercice

Pour commencer l'ordinateur va choisir au hasard un nombre compris entre 1 et 100.
L'utilisateur doit alors deviner ce nombre comme ceci :
L'utilisateur propose un nombre. L'ordinateur lui dit s'il est trop petit ou trop grand, et ainsi de suite jusqu'à ce que l'utilisateur ait trouvé le bon nombre.

Des instructions nécessaires à la réalisation du programme :

In [79]:

```
#la condition n'est pas égale s'écrit :
!=
#le test d'égalité s'écrit :
==
```

Ecrire la fonction `plusOuMoins()` qui en entrée ne reçoit rien et en sortie renvoie le nombre d'essais qu'il a fallu pour trouver l'entier.

In [14]:

```
def pluOuMoins()->str:
```

```
    p=int(input("Donner un entier entre 1 et 100")) #initialisation de la  proposition
    r=randint(1,100) #initialisation de la valeur réponse
    print(r)
    while p!=r:
        if p<r:
            p=int(input("La réponse est supérieure à ta propositon. Essaie encore !"))
        elif p>r:
            p=int(input("La réponse est inférieure à ta propositon. Essaie encore !"))
    print("Gagné")
    return None
```

In [15]:

```
#tester le programme
pluOuMoins()
```

```
Donner un entier entre 1 et 10050
85
La réponse est supérieure à ta propositon. Essaie encore !85
Gagné
```

Exercice

On reprend l'exercice précédent en ajoutant une règle. Vous devez ajouter une entrée `nbEssais` de type int à la fonction qui correspond au nombre d'essais maximum. On nommera cette fonction `plusOuMoinB`

In [34]:

```
def plusOuMoinB(nbEssais:int)->str:
    p=int(input("Donner un entier entre 1 et 100")) #initialisation de la  proposition
    r=randint(1,100) #initialisation de la valeur réponse
    n=1 #initialisation du nombre d'essais
    print(r)
    while p!=r and n<nbEssais:
        n=n+1
        if p<r:
            p=int(input("La réponse est supérieure à ta propositon. Essaie encore !"))
        elif p>r:
            p=int(input("La réponse est inférieure à ta propositon. Essaie encore !"))
    if p==r and n<nbEssais:
        print("Gagné en" , n, "essais")
    else :
        print("perdu la réponse était :",r)
    return None
```

In [36]:

```
plusOuMoinB(10)
```

```
Donner un entier entre 1 et 10050
65
La réponse est supérieure à ta propositon. Essaie encore !60
La réponse est supérieure à ta propositon. Essaie encore !65
Gagné en 3 essais
```