Entrée[2]:
```
1 import numpy as np
```

TP Python 3 : boucles bornées et non bornées

Corrigé

Les boucles en informatique permettent de **répéter une instruction autant de fois que nécessaire.**
Il y a **deux types de boucles** en informatique :

1. les boucles **bornées**, ou **boucles *for*, où l'on précise dès le début combien de fois il faut répéter l'opération,**
2. **et les boucles non bornées , ou boucles *while*, où l'on donne une condition d'arrêt.**

Partie 1 : Prérequis sur Range et les listes

Une **liste** est une collection **ordonnée** d'éléments (on parle aussi de tableau, dit *array*). On peut coder des listes (ou tableaux) dans Python en utilisant les crochets. Leur type est : `list` .

Entrée[ ]:
```
1 L = [3,2,7]
```

Pour a et b deux entiers, `range(a,b)` permet d'obtenir la liste des nombres entre a et $b-1$ sous forme de liste. `range(b)` est un raccourci désignant `range(0,b)` . On peut aussi utiliser `range(a,b,c)` pour préciser le pas (l'écart entre les valeurs). Ces listes sont fréquemment utilisées et ont un type spécialisé : le type `range` .

Entrée[ ]:
```
1 list(range(2,9))
```

Entrée[ ]:
```
1 list(range(9))
```

Entrée[ ]:
```
1 list(range(2,15,3))
```

Nous détaillons peu les listes ici car nous y reviendrons au cours du TP, mais il

s'agit d'une structure importante. Quelques commandes sur les listes :

1. `L[k]` renvoie le k-ième élément de la liste `L`
   ATTENTION! La numérotation des indices commence à 0 : on dit dans le jargon que *array starts at 0*. Selon le langage de programmation utilisé, les listes sont initialisées soit à 0, soit à 1.
2. `len(L)` renvoie le nombre d'éléments de la liste `L` .
3. `L.append(a)` rajoute l'élément `a` à la fin de la liste `L` .

Entrée[1]:

```
L = [1,23,7,2]
print(len(L))
L.append(7)
L.append(3)
print(L)
print(len(L))
```

```
4
[1, 23, 7, 2, 7, 3]
6
```

Partie 2 : Boucle bornée *for*

La boucle for permet de répéter une commande pour chaque valeur d'une variable muette dans un ensemble, de manière similaire au symbole $\sum$ en mathématique où la commande que l'on répète est une addition. La structure est la suivante :

Entrée[ ]:

```
for variable in liste:
    instruction1
    instruction2
    instruction3
```

Exemple :

Entrée[ ]:

```
for k in range(1,10):
    print(k)
```

Les mots clefs sont `for` et `in` . Comme en mathématique, la variable muette n'a pas forcément à intervenir dans les commandes à répéter. On attire l'attention sur les deux choses suivantes :

1. **Il ne faut pas oublier `:`** dans le code.
2. **L'indentation est primordiale.** C'est elle qui signifie à Python comment sont organisés les blocs de codes à exécuter et leur agencements entre eux, à la manière des parenthèses pour organiser un calcul en mathématiques.

Exemple :
Pourquoi les codes suivants ne fonctionnent-ils pas comme attendu ?

Entrée[ ]:
```
# dans cet exemple, on souhaitait augmenter une valeur de 1 5 fo

l = 10
for k in range(3,8):
    l = l+1
    print(l)
```

Entrée[ ]:
```
for k in range(3,8)
    print(k)
```

Entrée[ ]:
```
l = 10
for k in range(7):
    a = l+1
   a = a+1
l
```

Exemple :
Que renvoie le code suivant ?

Entrée[ ]:
```
a = 0
for k in range(50):
    a += k
a
```

L'ensemble qui suit le mot `in` est donc celui où la variable muette prend ses valeurs. On peut utiliser pour cela la commande range comme on a vu, mais on peut aussi utiliser une liste de manière générale.

Entrée[ ]:
```
x = 0
for s in [3,5,9,7]:
    x = x+s
x
```

Entrée[ ]:
```
for j in "test":    # un exemple pour constater que cela fonctio

    print(j)
```

Partie 3 : Boucle non bornée *while*

Avec une boucle while, on répète la ou les commandes associées tant que la condition d'arrêt n'est pas vérifiée.
Donnons deux exemples :

Entrée[ ]:
```
k = 0
while k < 10:
    print(k)
    k += 1
k
```

Entrée[ ]:
```
k = 2
while 3*k <= 14:
    k = k+1
k
```

Quelques remarques :

1. Il ne faut pas oublier `:` dans le code.
2. L'indentation est là encore primordiale!
3. **La condition d'arrêt est cruciale**. Si la condition d'arrêt est mal choisie, le code pourrait s'arrêter au mauvais moment ou ne jamais s'arrêter. Dans ce dernier cas, on parle de **boucle infinie**, ce qui fait planter votre machine.

Exemple :
Quel est le plus petit multiple de 73 supérieur à 4000 ?

Entrée[ ]:
```
a = 0
while a < 4000:
    a = a+73  # on parcourt les multiples de 73
a
```

Entrée[ ]:
```
a = 4000
while a%73 != 0:      # on part de 4000 et on s'arrête dès qu'on
    a = a+1
a
```

Remarque :
On peut utiliser la commande `break` pour sortir d'une boucle, mais nous n'en aurons pas besoin.

Exercices -- Corrigé

Entrée[ ]:
```
import numpy as np
```

Exercice 1

1. Coder un programme demandant à l'utilisateur la valeur de n (avec `input`) et qui renvoie en sortie la somme des nombres entiers entre 1 et n.
2. Faire la même chose en utilisant plutôt une fonction informatique

Entrée[6]:

```
# La commande int() est un garde-fou, au cas où l'utilisateur
# se trompe et rentre une valeur non entière comme 7.8 par exempl
# La commande \n est une commande de texte et permet un saut à la
n = int(input('Rentrer une valeur entière \n'))

# On déclare et initialise la variable somme
s = 0
for i in range(1,n+1):
    s = s + i
print(' La somme des ',n,' premier entiers vaut ',s)
```

```
Rentrer une valeur entière
4
 La somme des  4  premier entiers vaut  10
```

Exercice 2
Calculer le produit $123\times 124 \times \dots \times 212 \times 213$.

Entrée[19]:

```
# On définit le point de départ du produit
a = 123
b = 213

# On initialise le produit /!\ il faut débuter à 1 pas à 0,
# ce n'est pas une somme
prod  = 1

# On bouge de a à b, en prenant garde au fait que range
# boucle à partir de a inclu jusqu'à b exclu
for k in range(a,b+1):
    prod = prod * k

print('Le produit 123x124x...x212x213 vaut ',prod,'\n')
# HP : on peut afficher un nombre flottant selon différent format
# à l'aide de la fonction 'format'.
# Pour plus de renseignements https://touron.tech/doc_info/Fiches
print('L''écriture scientifique de ce produit est ',format(prod,
```

```
Le produit 123x124x...x212x213 vaut  102103701457377947454966533816
963402118177810861414895278395996317296053527613432978185577552591 1
856783036183931696516934514205218282081991660761634678065744317465 0
504344436001996800000000000000000000000

Lécriture scientifique de ce produit est  1.02104e+202
```

Exercice 3
Créer une fonction `fact(n)` permettant de calculer $n!$.

Entrée[22]:

```
def fact(n):
    # On initialise le produit à 1
    y = 1

    # On boucle à partir de 0 inclus jusqu'à n inclus!
    # Il fallait ici faire attention au cas 0! = 1.
    for k in range(0,n):
        y = y*(k+1)

    print( n,'! = ',y,'\n')
    return(y)

#Test :
fact(0)
fact(1)
fact(2)
fact(5)

```

```
0 ! =  1

1 ! =  1

2 ! =  2

5 ! =  120
```

Sortie[22]:

```
120
```

Exercice 4

1. Créer une fonction qui prend en entrée un entier n et renvoie : $u_n=\underset{k=2}{\overset{n}{\prod}}\left(1-\dfrac{1}{k^2}\right)$.
2. Tester cette fonction pour des valeurs de n et conjecturer le sens de variation de la suite.
3. Prouver cette conjecture (sans Python).

Entrée[43]:

```
def u(n):
    y = 1
    for k in range(2,n+1):
        tg = 1-(1/(k**2))
        y = y * tg

    return y

# TEST
print('u_0 = ',u(0),' car on ne rentre pas dans la boucle.\n')
print('u_1 = ',u(1),' car on ne rentre pas dans la boucle.\n')
t2 = 1-(1/(2**2))
print('u_2 = ',u(2),' et le produit directement vaut ',t2)
t3 = 1-(1/(3**2))
print('u_3 = ',u(3),' et le produit directement vaut ',t2*t3)

# Conjecture
# On a déjà affiché u_2 et u_3 pour tester notre fonction.
# Affichons les premières valeurs de u_n pour conjecture que ceti

for i in range(7,13):
    print('u_',i,' = ',u(i))

# La preuve de la stricte décroissance est laissée en exercice
```

```
u_0 =  1  car on ne rentre pas dans la boucle.

u_1 =  1  car on ne rentre pas dans la boucle.

u_2 =  0.75  et le produit directement vaut  0.75
u_3 =  0.6666666666666666  et le produit directement vaut  0.666666
6666666666
u_ 7  =  0.5714285714285714
u_ 8  =  0.5625
u_ 9  =  0.5555555555555556
u_ 10  =  0.55
u_ 11  =  0.5454545454545455
u_ 12  =  0.5416666666666667
```

Exercice 5
On considère la suite définie par $u_0=2$ et : $\forall n \in \mathbb{N},~ u_{n+1} =2u_n+1$.

1. Écrire une fonction permettant de calculer la valeur du terme d'indice n de cette suite.
2. Modifier la fonction pour qu'elle affiche les 20 premiers termes de la suite.
3. Conjecturer la limite de cette suite puis prouver votre conjecture (sans Python).
4. Écrire une fonction permettant de calculer la somme des n premiers termes de la suite.

Entrée[53]:

```
def suite1(n):
    u = 2
    print('Le premier terme vaut ',u,'.\n')
    for k in range(1,n+1):
        u = 2*u+1
        print('Au rang ',k, ' le terme de la suite est ',u,'.\n')

    return
# On affiche les 20 premiers termes.
suite1(20)

# Conjecture : la limite semble être +infini. La preuve est laiss

# Calculons les n premiers termes de la suite
def somme_suite1(n):
    # suite initialisée
    u = 2
    # somme initialisée
    s = 0 +u
    # on boucle pour calculer ...
    for k in range(1,n+1):
        # le terme u_k
        u = 2*u+1
        # la somme des k premiers termes
        s = s + u

    # on s'intéresse à la somme s, c'est elle qu'on renvoie
    return s

# on teste
print(' La somme des ',4,' premiers termes de la suite vaut ',sor
```

```
Le premier terme vaut  2 .
Au rang  1  le terme de la suite est  5 .
Au rang  2  le terme de la suite est  11 .
Au rang  3  le terme de la suite est  23 .
Au rang  4  le terme de la suite est  47 .
Au rang  5  le terme de la suite est  95 .
Au rang  6  le terme de la suite est  191 .
Au rang  7  le terme de la suite est  383 .
Au rang  8  le terme de la suite est  767 .
Au rang  9  le terme de la suite est  1535 .
Au rang  10  le terme de la suite est  3071 .
Au rang  11  le terme de la suite est  6143 .
Au rang  12  le terme de la suite est  12287 .
Au rang  13  le terme de la suite est  24575 .
Au rang  14  le terme de la suite est  49151 .
Au rang  15  le terme de la suite est  98303 .
Au rang  16  le terme de la suite est  196607 .
Au rang  17  le terme de la suite est  393215 .
Au rang  18  le terme de la suite est  786431 .
Au rang  19  le terme de la suite est  1572863 .
Au rang  20  le terme de la suite est  3145727 .
 La somme des  4  premiers termes de la suite vaut  88
```

Exercice 6
On considère la suite définie par $u_0=1$, $u_1=-1$ et : $\forall n \in \mathbb{N}, ~u_{n+2}=4u_{n+1}-4u_n$.

1. Écrire une fonction permettant de calculer la valeur du terme d'indice n de cette suite.
2. Conjecturer la limite de cette suite.
3. Écrire une fonction `S(n)` permettant de calculer la somme $S(n)=\underset{k=0}{\overset{n-1}{\sum}} u_k$.

Entrée[77]:

```
def suite2(n):
    u0 = 1
    u1 = -1
    print('u_0 = ',u0,'\n')
    print('u_1 = ',u1,'\n')
    a = u0
    b = u1
    c = 0

    for k in range(2,n+1):
        # on définit le terme u_{k+2}
        c = 4 * b -4 * a
        print('u_',k,'= 4*(',b,')-4*(',a,')=',c,'\n')
        # le terme u_{k} prend la valeur de u_{k+1}
        a = b
        # le terme u_{k+1} prend la valeur de u_{k+2}
        b = c


    return c

# Affichons les 15 premiers termes
print(suite2(3))

# On conjecture que la suite tends vers -infini. Preuve laissée e

# Somme des n premiers termes.

def somme_suite2(n):
    # la somme de u0 et u1 fait 0. On peut donc directement somme
    s = 0
    u0 = 1
    u1 = -1

    # Si n vaut 0, la somme n'est pas définie, on sort de la fonc
    # après avoir affiché un message d'erreur
    if n == 0 :
        print(' La somme n''est pas définie.')
        return

    # Si n vaut 1, la somme vaut 0, rien à boucler, on la renvoie
    if n == 1 :
        return s

    # Autrement, i.e si n>=2 :
    else :
        a = u0
        b = u1
        c = 0

        for k in range(2,n+1):
            # on définit le terme u_{k+2}
            c = 4 * b -4 * a
            s = s + c
            # le terme u_{k} prend la valeur de u_{k+1}
            a = b
            # le terme u_{k+1} prend la valeur de u_{k+2}
            b = c
        return s
```

```
60
61 # TEST
62 print('La somme des ',1, 'premiers termes est',somme_suite2(0))
63  print('La somme des ',2, 'premiers termes est',somme_suite2(1))
64  print('La somme des ',3, 'premiers termes est',somme_suite2(2))
65  print('La somme des ',4, 'premiers termes est',somme_suite2(3))
```

```
u_0 =  1

u_1 =  -1

u_ 2 = 4*( -1 )-4*( 1 )= -8

u_ 3 = 4*( -8 )-4*( -1 )= -28

-28
 La somme nest pas définie.
La somme des  1 premiers termes est None
La somme des  2 premiers termes est 0
La somme des  3 premiers termes est -8
La somme des  4 premiers termes est -36
```

Exercice 7

1. Programmer une fonction `aux(a,b)` prenant en entrée deux entiers naturels a,b avec $a\leq b$ et renvoyant le nombre maximal de fois que l'on peut soustraire a à b sans obtenir une quantité négative.
   Par exemple, pour $a=3$ et $b=23$, la réponse est 7.
2. Coder une fonction prenant en entrée une somme d'argent n (entière) exprimée en euros, et renvoyant une décomposition de cette somme en un nombre minimal de pièces et de billets.
   Par exemple, pour $n=543$ euros, la décomposition sera : 1 billet de 500, 2 billet de 20, 1 pièces de 2, 1 pièce de 1.

Entrée[4]:

```
def retrancher(a,b):

    if a>b :
        return 0

    # on initialise un compteur
    compt = 0

    # on initialise un index pour soustraire a à b
    k = 1
    while (b-k*a >=0):
        # on peut soustraire k fois a à b sans être dans le négat
        # on augmente le compteur
        compt = compt +1
        # on passe à k+1 fois a.
        k = k+1

    return compt

# TEST
retrancher(56,10)
print(' On peut retrancher',retrancher(3,23),'fois le nombre a=',

def billets(n):

    liste_monnaie = [500,200,100,50,20,10,5,2,1]
    monnaie_rendue = []
    r = n
    k = 0

    while k in range(np.size(liste_monnaie)):

        montant = liste_monnaie[k]

        b = retrancher(montant,r)
        r = r - b * montant

        if b != 0 :
            if (montant - 5) <0 :
                print("==> {:4d} pièce(s) de {:4d} euros, reste
            else :
                print("==> {:4d} billet(s) de {:4d} euros, reste

        k = k+1

    return

billets(543)
```

```
 On peut retrancher 7 fois le nombre a= 3 au nombre b= 23 .


==>    1 billet(s) de  500 euros, reste 43 euros
==>    2 billet(s) de   20 euros, reste 3 euros
==>    1 pièce(s) de    2 euros, reste 1 euros
==>    1 pièce(s) de    1 euros, reste 0 euros
```

Exercice 8

Chargement des fichiers auxiliaires...

On considère la suite définie par $u_0=2$ et $u_{n+1}=2u_n+1$.
Quel est le rang du premier terme supérieur à 1 000 000 ?

Entrée[18]:

```
def suite3(n):
    u0=2
    u = u0

    for k in range(n):
        u = 2*u+1

    return u

N = 1000000
terme = suite3(0)
k = 0
while terme < N :
    terme = suite3(k+1)
    k = k+1

print('À partir du rang ',k,'les termes de la suite dépassent ',N
```

```
À partir du rang  19 les termes de la suite dépassent  1000000
```

Exercice 9
Déterminer le rang du dernier terme strictement positif de la suite $(u_n)_{n \in \mathbb{N}}$ définie par $u_0=890$ et pour tout $n \in \mathbb{N}$, $u_{n+1} =\dfrac{u_n}{2}-3n$.

Entrée[33]:

```
def suite4(n):
    u0 = 890
    u = u0
    for k in range(1,n+1):
        u = (u/2)-3*(k-1)
    return u

terme = suite4(0)
k = 0
while terme >=0 :
    terme = suite4(k+1)
    k = k+1

print('Le rang du dernier terme strictement positif est ',k-1,' a


```

```
Le rang du dernier terme strictement positif est  5  avec u_ 5 = 9.
4375
 et u_ 6 = -10.28125
```

Exercice 10
Un appartement vaut 100000 euros. Sa valeur est multiplié par 1.2 chaque année. Au bout de combien d'années la valeur de l'appartement aura doublé ? (on résoudra le problème informatiquement mais aussi mathématiquement, sans

Python)

Entrée[45]:
```
val_ini = 1000000
annee = 0
val_annee = val_ini
objectif = val_ini *2
while val_annee < objectif:
    val_annee = val_annee * 1.2
    annee = annee+1

print('Au bout de ',annee, 'ans, la valeur de l''appartement aura
```

```
Au bout de  4 ans, la valeur de lappartement aura doublé.
```

Exercice 11
Que fait le programme suivant ?

Entrée[35]:
```
def precision(epsilon):
    u = 14
    n = 0
    while abs(u-4) > epsilon:
        u = 6-0.5*u
        n = n+1
    return n

# TEST
precision(0.1)
```

Sortie[35]:
```
7
```

On définit la suite $(u_n)_{n \in \mathbb{N}}$ par : $\begin{cases} u_0 = 14 \\ u_{n+1} = 6- \dfrac{u_n}{2} \quad \forall n \geq 1\end{cases}$. Ce code renvoie le rang à partir duquel les termes de la suite $(u_n)_{n \in \mathbb{N}}$ se trouvent dans l'intervalle fermé $\left[4-\varepsilon :4+\varepsilon\right]$, cf définition de la convergence d'une suite vers 4.

Exercice 12
Renvoyer la plus petite valeur de $n\in \mathbb{N}$ tel que $1-\left(\dfrac{5}{6}\right)^n \geq 0.99$.

Entrée[46]:
```
n =0
q = 5/6

while 1-q**n <0.99 :
    n = n+1

print('La plus petite valeur pour laquelle le terme est supérieu
```

```
La plus petite valeur pour laquelle le terme est supérieur ou égal
à 0.99 est 26 avec le terme égal à  0.9912645033246699 et le terme
précédent valant 0.9895174039896039
```

Exercice 13
Écrire une fonction dont l'entrée est un entier positif n et la sortie est le nombre de fois que cette entier est divisible par 2.

Entrée[48]:
```
def parité(n):
    p = bool
    if n%2 == 0:
        print('L''entier considéré est pair')
    else :
        print('L''entier considéré est impair')
    return p

parité(56)
parité(13)
```

```
Lentier considéré est pair
Lentier considéré est impair
```

Exercice 14
On considère la suite de Syracuse $(x_n)_{n \in \mathbb{N}}$ définie par la valeur de son premier terme x_0 et par : $$\forall n \in \mathbb{N}, ~ x_{n+1} =\left\{ \begin{array}{rcl} \dfrac{x_n}{2}&\text{si}& \text{x_n est pair}\\ 3x_n+1 &\text{sinon} & \end{array}\right. $$
La conjecture de syracuse dit : $\forall x_0 \in \mathbb{N}^*, ~ \exists n \in \mathbb{N}, ~ x_n=1$.

1. A. Écrire une fonction `Syracuse(x0)` qui renvoie le premier indice n tel que $x_n=1$ en fonction de la valeur de x_0.
   B. Tester votre fonction en $x_0=27$.
2. Vérifier la conjecture de Syracuse pour les 20 premiers entiers x_0

Entrée[ ]:
```
def syracuse(x0):

```