R1.06 TP 2

Suites aliquotes

1. Découverte et construction

13 novembre 2025 – B. Colombel

Avant de commencer : introduction aux listes

Une *liste* est simplement une suite d'objets entre crochets. On peut faire appel au i^{e} élément de la liste L par la commande `L[i]`. Attention, la numérotation commence à 0 !
Taper les commandes suivantes. À quoi servent-elles ?

[1]:
```
L = [2 ,3 ,5 ,7 ,11 ,13 ,42 ,0]
print(L)
```

[2]:
```
L [0] , L [1] , L [ -1] , L [ -2] , len ( L ) , len ([])
```

[3]:
```
L [2:5]
```

[4]:
```
[1..3 , 7 , 10..13]
```

Les éléments d'une liste ne sont pas nécessairement du même type :

[5]:
```
M=[7,"x",["pi","b"]]
print(M[2])
print(M[2][1])
```

Les listes sont un type d'objet très utile. En particulier, lorsqu'on a besoin de renvoyer en sortie d'une procédure plusieurs éléments, il suffit d'en faire une liste et de renvoyer la liste.

On peut aussi affecter un objet dans une case particulière de la liste :

[6]:
```
L[7] = 99
L
```

Pour rajouter un élément à la fin de la liste, on peut utiliser la commande `append` :

[7]:
```
L.append("y")
print(L)
```

D'autres commandes sur les listes (liste vide, création de liste avec des entrées générées par une fonction) sont illustrées dans les exemples suivants :

[8]:
```
N = [i^2 for i in range(1,6)]
print(N)
```

[9]:
```
for i in range(0,len(N)) :
    N[i] = N[i]/(i+1)
print(N)
```

[10]:
```
[3,4,7]+[1,2]
```

1 Conjecture de Catalan

Dans la suite n désigne un entier positif strictement supérieur à 1 ($n > 1$).

Définition 1 (Diviseurs stricts)
Un *diviseur strict* de n est un diviseur de n strictement inférieur à n.
Un diviseur strict de n est aussi appelé *partie aliquote* de n.

Remarque. l'ensemble des parties aliquotes des n est donc l'ensemble des diviseurs de n auquel on a enlevé n.

Exemple 1. Les diviseurs stricts de 24 sont :

$$1,2,3,4,6,8 \text{ et } 12 \qquad \text{(on ne compte pas 24)}$$

Définition 2 (Suite aliquote)
La suite aliquote de l'entier n est construite de la manière suivante :
- le premier terme de la suite est n ;
- chaque terme est égal à la *somme des diviseurs stricts* du terme précédent.

Exemple 2. Si $n = 24$, alors :
- les diviseurs stricts de 24 sont 1, 2, 3, 4, 6, 8 et 12 ; leurs somme est $1+2+3+4+6+8+12 = 36$ donc le terme qui vient après 24 est 36.
- les diviseurs stricts de 36 sont 1, 2, 3, 4, 6, 9, 12 et 18 ; leur somme est 55 ; le troisième terme est donc 55.

et ainsi de suite. Finalement, le suite aliquote issue de 24 est :

$$24, \quad 36, \quad 55, \quad 17 \quad \text{et } 1$$

le processus s'arête, 1 n'ayant pas de diviseur strict.

Conjecture de Catalan Le mathématicien Eugène Catalan a conjecturé à la fin du XIX^{e} siècle que, quelque soit le nombre de départ, on finit toujours par se retrouver dans un des deux cas suivants :

1. On finit par tomber sur 1 et le processus s'arête car 1 n'a aucun diviseur strict.
2. On obtient un cycle dont la longueur peut varier.

2 Construction de suites aliquotes

Pour construire des suites aliquotes et étudier certaines de leurs propriétés, nous avons besoin d'une fonction qui détermine les diviseurs stricts d'un entier $n > 1$.

Exercice 1. Méthode naïve
Pour déterminer tous les diviseurs stricts de n, il suffit de tester si la division euclidienne a un reste nul pour tous les entiers p tels que $1 \leqslant p \leqslant n - 1$.
Avec *Sagemath*, le reste de la division euclidienne de a par b s'obtient avec la commande `a % b`. Par exemple,

[11]:
```
24 % 4
```

[11]:
```
0
```

[12]:
```
24 % 5
```

[12]:
```
4
```

1. Écrire une fonction `parties_aliquotes(n)` qui retourne la liste des diviseurs stricts de n.
2. Écrire une fonction `suivant_aliquote(n)` qui retourne l'élément suivant n dans le suite aliquote.

Exemples d'utilisation.

[14]:
```
n = 24
parties_aliquotes(n)
```

[14]:
```
[1, 2, 3, 4, 6, 8, 12]
```

[15]:
```
n = suivant_aliquote(n)
print(n)
```

[15]:
```
36
```

[16]:
```
n = suivant_aliquote(n)
print(n)
```

[16]:
```
55
```

[17]:
```
n = suivant_aliquote(n)
print(n)
```

[17]:
```
17
```

[18]:
```
n = suivant_aliquote(n)
print(n)
```

[18]:
```
1
```

3 Nombres parfaits, amicaux et sociables

3.1 Cycle de longueur 1 : nombres parfaits

> On appelle *nombre parfait* les nombres qui sont égaux à la somme de leurs parties aliquotes.
> Dans les suites que nous construisons, les nombres parfaits correspondent donc à des cycles de longueurs 1.

Exercice 2. Écrire une fonction `parfaits(n)` qui retourne tous la liste des nombres parfaits plus petits que n.
Commenter le temps des calculs.

Exemple d'utilisation.

[20]:
```
L = parfaits(1000)
```

[20]:
```
[6, 28, 496]
```

[21]:
```
L = parfaits(10000)
```

[21]:
```
[6, 28, 496, 8128]
```

3.2 Cycles de longueur 2 : nombres amicaux

Les *nombres amicaux* correspondent donc à des cycles de longueurs 2.

Exercice 3. Écrire une fonction `amicaux(n)` qui retourne la liste des nombres amicaux inférieurs à n.
Commenter le temps des calculs.

Exemple d'utilisation

[22]:
```
L = amicaux(1000)
```

[22]:
```
[220, 284]
```

[23]:
```
L = amicaux(10000)
```

[23]:
```
[220, 284, 1184, 1210, 2620, 2924, 5020, 5564, 6232, 6368]
```

3.3 Cycles longs : nombres sociables

À ce jour, on ne connait pas de cycle de longueur 3 mais on peut trouver des cycles de longueur 4. Malheureusement, nos méthodes sont trop lentes et il faut donc les améliorer et les rendre plus performantes.

Pour cela, nous auront besoin des nombres premiers pour déterminer plus rapidement les diviseur propres de l'entier n.