ESIR

CUPGE 1

Algorithmique

P. Rannou

2021-2022

Travaux Dirigés

1 Instructions de base

Exercice 1

1.1. Donner les valeurs des variables x, y et z à la fin de l'exécution du pseudo–code suivant :

Données : x,y,z : entiers
x ← 4
y ← 2+x
z ← x-y
y ← y+z

1.2. Échange de deux valeurs
On suppose que deux variables `a` et `b` ont déjà été initialisées. On note a_i la valeur initiale de `a` et b_i la valeur initiale de `b`.

a. Donner les valeurs de `a` et de `b` après la séquence d'instructions suivante :

Données : a,b,c : entiers
c ← a
a ← b
b ← c

b. Quelles seraient les valeurs de `a` et de `b` si à la place on avait effectué la séquence d'instructions suivante :

Données : a,b : entiers
a ← b
b ← a

Exercice 2

2.1. Donner le type (`int`, `float`, `bool` ou `str`) de chacune des valeurs suivantes :

a. `4.5` **b.** `'4'` **c.** `False` **d.** `0`

2.2. Donner le type (`int`, `float`, `bool` ou `str`) de chacune des expressions suivantes. Préciser aussi la valeur de cette expression.

a. `float(4)`
b. $4 > 5.5$
c. $(4 + 5.5) * 2$
d. `str`$(4 + 5)$
e. `str`$(4) +$ `str`(5)
f. $((3 < 4)$ and $(4 < 3))$ or $3 \mathrel{!}= 4$
g. $9/3$
h. $7//$`int`(3.14)
i. `'zebre'<'lion'`

2.3. On dispose d'une variable entière `n` prédéfinie.
On souhaite créer une variable `t` de type `str` précisant le double de `n` : si par exemple `n` vaut 4 alors `t` doit valoir :
`'Le double de 4 est 8.'`

Attention : il s'agit de créer une nouvelle variable, et pas d'afficher un message avec `print`.

Exercice 3

3.1. Le savant Sunisoc a écrit un programme permettant avec une instruction `input` de rentrer un nombre et d'afficher son double. Le programme est donné ci–dessous.

```
a = input("Rentrez un nombre entier : ")
print ("Le double de ce nombre est : ", a+a)
```

Il y a une erreur dans ce programme. Si un utilisateur écrit ce programme et rentre la valeur 4, quel affichage surprenant obtiendra–t–il ?

Exercice 4 Intersection de deux rectangles

Le plan est rapporté à un repère orthonormé direct $(O, \vec{\imath}, \vec{\jmath})$.
On se donne deux rectangles R_1 et R_2 dont les côtés sont parallèles aux axes de coordonnées.
Soient A le sommet inférieur gauche de R_1 et B son sommet supérieur droit.
Soient C le sommet inférieur gauche de R_2 et D son sommet supérieur droit.
On suppose connues les coordonnées `xA`, `yA`, `xB`, `yB`, `xC`, `yC`, `xD`, `yD` de ces 4 sommets.
4.1. Définir un test permettant de savoir si ces deux rectangles s'intersectent.

Exercice 5

5.1. Donner dans chacun des cas la valeur de la variable `a` après l'exécution du programme :

a.
```
a = 4
if a*2 > 10 :
    a = a-8
else :
    a = a+8
```

b.
```
a=int(input("Rentrez un entier"))
a=a*2
a=a**2
a=a**3
```

c.
```
a = 5
if a < 4 :
    a += 2
a -= 2
```

d.
```
a = 5
if a < 4 :
    a += 2
    a -= 2
```

e.
```
a = 5
if a%2 == 0 :
    a += 3
else :
    a //= 2
```

f. *Traiter les différents cas*

```
b=int(input("Rentrez un entier"))
a=2*b+4
if a == b :
    a = 0
```

Exercice 6 Imbrication d'embranchements conditionnels

Une compagnie de bus propose des tarifs de groupe pour un trajet :
- 10€ le ticket pour un seul passager
- 8€ le ticket s'il y a entre 2 et 3 passagers

- 7€ le ticket s'il y a entre 4 et 5 passagers
- 6€ le ticket s'il y a 6 passagers ou plus

6.1. Définir un algorithme demandant à l'utilisateur d'entrer le nombre de personnes dans un groupe et affichant le prix **total** que paiera ce groupe pour le trajet.

Un informaticien propose de coder ce calcul en `python` de la manière suivante :

```
if passagers < 4 :
    if passagers >= 2 :
            prix = passagers * 8
    else :
            prix = 10
else :
    if passagers < 6 :
            prix = passagers * 7
    else :
            prix = passagers * 6
```

6.2. Est-ce que son code est correct ?

6.3. Donner une autre façon de calculer ce prix en utilisant des `elif`.

Exercice 7

7.1. Préciser dans chacun des cas suivants l'éventuel message affiché par le programme (ou un message d'erreur si le programme est incorrect) :

a.
```
a = 6
t = (a%2 == 0)
if True == t :
    print ("Le nbr",a,"est pair")
```

b.
```
a = 6
t = (a%2 == 0)
if t :
    print ("Le nbr",a,"est pair")
```

c.
```
a = 6
if a%2 :
    print ("Le nbr",a,"est pair")
```

d.
```
a = 6
t = (a%2 == 0)
if t == True :
    print ("Le nbr',a,'est pair")
```

Exercice 8 Programme mystère

8.1. Préciser en fonction de l'entier `n` la valeur de `y` après l'exécution du programme suivant. On précisera les différents cas.

```
if n >= 3:
    y = n*n
    if n <= 4 :
        y = y-1
        y = 2*y
    else :
        y += 1
else :
    if n % 2 == 0 :
```

```
        y = 3*n
    else :
        y = n-5
    y = 3*y
```

Exercice 9 Minimum de trois valeurs

On suppose qu'on a déjà défini trois variables réelles `a`, `b` et `c`.

9.1. Écrire une conditionnelle utilisant uniquement des tests $<$ qui permet d'affecter à la variable `d` la plus petite des trois valeurs entre `a`, `b` et `c`
(remarque : il existe une solution plus simple : `d=min(a,b,c)`*).*

Exercice 10

10.1. On suppose qu'une variable `a` est entrée au clavier par l'utilisateur. Qu'affiche l'algorithme suivant ?

Données : a,b : entiers
lire (a)
$b \leftarrow 0$
tant que $b * b < a$ **faire**
  $b \Leftarrow b + 1$
fin
afficher b

Exercice 11

On définit une suite (u_n) par $u_0 = 0$ et $\forall n \in \mathbb{N}, u_{n+1} = \cos(u_n)$.
Un entier naturel n est lu au clavier et stocké dans une variable `n` (noter la différence entre l'entier mathématique n écrit en italique et qui ne change pas, et la variable informatique `n` écrite en police à chasse fixe).

11.1. À la fin de l'exécution de l'algorithme suivant, la variable `u` contient–t–elle u_n, u_{n+1} ou autre chose ?

Données : n : entier, u : réel
lire (n)
$u \leftarrow 0$
tant que $n > 0$ **faire**
  $u = \cos(u)$
  $n \leftarrow n - 1$
fin

Exercice 12

La suite de Fibonacci (F_n) est définie par $F_0 = 0$, $F_1 = 1$ et $\forall n \in \mathbb{N}, F_{n+2} = F_{n+1} + F_n$.
Un entier naturel n est lu au clavier et stocké dans une variable `n`.
12.1. À la fin de l'exécution de l'algorithme suivant, la variable `f` contient–t–elle F_n, F_{n+1} ou autre chose ? Et la variable `b` ?

```
Données : a,b,f,n : entiers
lire (n)
f ← 0
a ← 1
tant que n > 0 faire
  | b ← a + f
  | f ← a
  | a ← b
  | n ← n − 1
fin
```

Exercice 13

Le programme suivant permet de calculer le plus grand diviseur impair d'un entier positif n :

```
Données : n : entier
lire (n)
tant que n%2 = 0 faire
  | n ← n//2
fin
afficher (n)
```

13.1.

a. Que se passe–t–il si la valeur initialement rentrée dans `n` est 0 ?

b. Pourquoi le programme précédent est–il dangereux ?

c. Comment le modifier pour éviter les risques de “plantage” de l'ordinateur ?

Exercice 14

14.1. Écrire un algorithme utilisant une boucle `while` qui lit un entier n au clavier et qui affiche la somme des entiers naturels k tels que $k^2 + 3k <$ `n`.
Par exemple si `n` $= 20$ on doit trouver $0+1+2+3 = 6$ car 3 est le plus grand entier naturel pour lequel $k^2 + 3k < 20$.
14.2. Serait–il possible de résoudre ce problème sans utiliser de boucle `while` ?

Exercice 15 Punition

L'instruction `print ("Je dois ranger ma chambre")` provoque l'affichage du message :

```
Je dois ranger ma chambre
```

15.1. Écrire un programme avec une boucle qui affiche 50 fois le message `Je dois ranger ma chambre`.

Exercice 16 Chanson traditionnelle bretonne

La séquence d'instructions :

```
n = 10
print ("C'est dans", n, "ans je m'en irai j'entends le loup le renard chanter")
```

permet d'afficher le message :

```
C'est dans 10 ans je m'en irai j'entends le loup le renard chanter
```

16.1. Écrire une boucle `python` qui permet d'afficher :

```
C'est dans 10 ans je m'en irai j'entends le loup le renard chanter
C'est dans 9 ans je m'en irai j'entends le loup le renard chanter
C'est dans 8 ans je m'en irai j'entends le loup le renard chanter
...
C'est dans 1 ans je m'en irai j'entends le loup le renard chanter
```

On ne s'occupera pas de la faute d'orthographe de la dernière ligne.

Exercice 17

17.1. Écrire un algorithme lisant un entier `n` au clavier et affichant $H_{\mathtt{n}} = \prod_{k=1}^{\mathtt{n}} k^k$.

17.2. Écrire un algorithme lisant un entier `n` au clavier et affichant $\sum_{i=1}^{\mathtt{n}} H_i$.

Exercice 18

18.1. Écrire un algorithme demandant à l'utilisateur d'entrer un entier n au clavier et permettant de calculer le n–ième terme de la suite (u_n) définie par

$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, u_{n+1} = u_n + \cos(u_n) \end{cases}$$

18.2. Écrire un algorithme demandant à l'utilisateur d'entrer un entier n au clavier et permettant de calculer le n–ième terme de la suite (u_n) définie par

$$\begin{cases} u_0 = 1 \\ u_1 = 0 \\ \forall n \in \mathbb{N}, u_{n+2} = u_{n+1} + \sin(u_n) \end{cases}$$

Pour aller plus loin

Exercice 19

On modélise une population de bactéries dans une solution de culture de la façon suivante : chaque jour, chaque bactérie :

- ou bien consomme une unité de nourriture, elle libère alors une quantité de toxine égale à $(t+1)e^{-t}$ où t est la quantité de toxine présente au début de la journée, puis elle se duplique ;
- ou bien elle meurt sans produire de toxine s'il n'y a plus de nourriture disponible.

Initialement il y a une seule bactérie, un nombre (entier) d'unité de nourriture mémorisé dans une variable `n`, et une quantité nulle de toxine.

19.1. Concevoir un algorithme permettant de calculer la quantité totale de toxine produite.

Exercice 20 Le problème de Josephus

Cette histoire est issue de la vie de l'historien antique Flavius Josèphe, qui a vécu sous l'empereur romain Vespasien (même si sa véracité n'est pas garantie).
Soient $k, n \in \mathbb{N}^*$. n personnes sont positionnées en cercle, numérotées de 0 à $n-1$. On élimine itérativement un survivant sur k en tournant, jusqu'à ce qu'il n'en reste plus qu'un (ainsi, lorsque $k \leqslant n$, la première personne éliminée a pour numéro $k-1$). Le but est de déterminer le numéro $S(n,k)$ du dernier survivant.

20.1.

a. Démontrer que $\begin{cases} \forall n \geqslant 2, S(n,k) \equiv S(n-1,k) + k \bmod n, \\ S(1,k) = 0. \end{cases}$

Pour la première identité on pourra par exemple raisonner sur le numéro j de la première personne éliminée, puis se ramener au problème de Josephus à $n-1$ personnes.

b. En déduire un programme permettant de calculer $S(n,k)$.

Exercice 21

En 2004 on a découvert que le nombre $n = 28433 \times 2^{7830457} + 1$ est un nombre premier (son écriture décimale est composée de 2 357 207 chiffres). Ce nombre est beaucoup trop grand pour pouvoir être calculé en temps raisonnable par l'interpréteur Python.

21.1. Comment calculer les 10 derniers chiffres de son l'écriture décimale en temps raisonnable ?

Exercice 22 La constante de Champernowne

La constante de Champernowne est obtenue en concaténant derrière la virgule (ou le point) la suite des écritures décimales des entiers consécutifs :

$$0.123456789101112131415161718192021\ldots$$

On note d_n la n–ième décimale de ce nombre : par exemple $d_1 = 1$ et $d_{15} = 2$.

22.1. Écrire un algorithme permettant de calculer d_n.

Exercice 23

Il est classique de coder un rationnel $\dfrac{p}{q}$ par le couple d'entiers $(\texttt{p}, \texttt{q})$.
Dans cet exercice on définit la fonction $f : \mathbb{N} \to \mathbb{Q}^+$ définie par :

$$f(0) = 0 \text{ et } \forall n \in \mathbb{N}^*, f(2n) = \frac{1}{f(n)+1} \text{ et } \forall n \in \mathbb{N}, f(2n+1) = f(n) + 1$$

et on admettra que f est une bijection de $\mathbb{N}$ sur $\mathbb{Q}^+$.

23.1. Écrire la fonction `f(n)` qui renvoie la valeur de $f(n)$ sous forme d'un couple d'entiers.

Remarque : si vous connaissez la récursivité, c'est plus facile à programmer, mais on peut s'en passer.

23.2. On note $g = f^{-1}$ la bijection réciproque. Programmer cette fonction `g(p,q)`. On cherchera une solution plus efficace que de tester un à un tous les entiers...

2 Types de données structurés

Exercice 24

24.1. À la fin de ce code, quelle est la valeur de `len(a)` ?

```
a = "a"
a *= 2
a = a + a + "a"
```

24.2. Donner la valeur de l'expression :

```
('mon '+2*'dou'+' '+2*"nou"+'rs')[1:18]
```

Exercice 25 Recherche d'un mot dans un texte

On dispose d'une chaîne de caractères courte mémorisée dans une variable `mot` de type `str`, ainsi que d'une chaîne (probablement plus longue) de caractères mémorisée dans une variable `texte` de type `str` aussi. La longueur de `mot` est `p` et la longueur de `texte` est `n`.

On souhaite savoir si la chaîne `mot` se trouve à l'intérieur de la chaîne `texte` : plus formellement on souhaite savoir si :

$$\exists\ 0 \leqslant \mathtt{i} \leqslant \mathtt{j} \leqslant \mathtt{n} \mid \mathtt{mot} == \mathtt{texte}[\mathtt{i} : \mathtt{j}]$$

On considère que si `a` et `b` sont deux chaînes de caractères de même longueur $\mathtt{q} \in \mathbb{N}$, le test $\mathtt{a} == \mathtt{b}$ nécessite pour l'interpréteur Python au plus la comparaison de `q` lettres.

25.1. Résoudre le problème posé avec une solution nécessitant au plus la comparaison de $\mathtt{p} \times (\mathtt{n} - \mathtt{p} + 1)$ lettres.

25.2. Modifier l'algorithme précédent, de façon à renvoyer la liste des indices où `mot` apparaît dans `texte`. Plus précisément, renvoyer la liste des indices :

$$\mathtt{i} \in [\mathtt{0}, \mathtt{n}] \text{ tels que } \exists \mathtt{j} \in [\mathtt{i}, \mathtt{n}] \,|\, \mathtt{mot} == \mathtt{texte}[\mathtt{i} : \mathtt{j}]$$

Exercice 26

26.1. Générer la liste $\ell = [\ell_0, \ell_1, \ldots, \ell_{\mathtt{n}-1}]$ avec $\ell_{\mathtt{k}} = 2\mathtt{k} + 1$.

26.2. Générer la liste $\ell = [\ell_0, \ell_1, \ldots, \ell_{\mathtt{n}-1}]$ avec $\ell_{\mathtt{k}} = \mathtt{k}^2$.

Exercice 27

On définit une suite (u_n) par $u_0 = 2$ et $\forall n \in \mathbb{N}, u_{n+1} = u_n + \sqrt{u_n} + 1$.

27.1. Écrire un algorithme qui permet de calculer la liste $[u_0, u_1, \ldots, u_{n-1}]$.

Exercice 28 *List comprehension*

28.1. Quel est le résultat de l'expression suivante (essayer de deviner avant de l'écrire) :

```
[i**2+1 for i in range (5)]
```

Est–ce qu'on obtient la même liste dans la variable `res` si on exécute le programme suivant :

```
res = []
for i in range (5) :
    res.append (i**2+1)
```

Exercice 29

Le savant Sunisoc crée une liste de listes nommée `x` :

```
>>> x = [[3, 4], [5, 9]]
```

Il souhaite dupliquer cette liste de listes dans une variable nommée `y` de façon à ce que toute modification subséquente de `x` ne modifie pas `y`.
Il utilise donc le constructeur de listes :

```
>>> y = list (x)
```

Pourtant une surprise l'attend :

```
>>> x[1][1] = 10
>>> x
[[3, 4], [5, 10]]
>>> y
[[3, 4], [5, 10]]
```

29.1. Expliquer ce qui s'est passé et corriger son code pour arriver au résultat voulu.

Exercice 30

30.1. Écrire un programme qui permet de supprimer toutes les occurrences d'un élément dans un tableau. Par exemple si on part de la liste `l=[2,6,2,2,4,5,2,3]` et qu'on supprime toutes les occurrences de 2, à la fin du programme la liste `l` doit être égale à `[6,4,5,3]`.

30.2. En utilisant la fonction `len` et une *list comprehension*, écrire un programme qui calcule la liste miroir d'une liste donnée (c'est–à–dire la liste des éléments dans l'autre sens).
Tester : si la liste d'entrée est `[3,6,1,8,2,4,0]`, son miroir est `[0,4,2,8,1,6,3]`.

Exercice 31

Une liste est un *palindrome* si c'est la même qu'on la lise de gauche à droite ou de droite à gauche.

31.1. Écrire un algorithme permettant de savoir si une liste est un palindrome.
Tester : pour `[4,2,4,2]` on doit renvoyer `False`, pour `[4,2,4]` ça doit être `True`.

Exercice 32 Second plus petit élément

32.1. Écrire un algorithme qui, à partir d'une liste d'entiers *distincts*, calcule le second plus petit élément de cette liste.

Exercice 33

Un tableau à double entrée (matrice) à valeurs entières est codé par une liste de listes.

Par exemple la matrice

2	3	7
1	2	0

est codée par :

$$[[2,3,7],[1,2,0]]$$

33.1. Écrire une fonction calculant le nombre de lignes d'une matrice. Sur notre exemple, la valeur de retour doit être 2.

33.2. Écrire une fonction calculant le nombre de colonnes d'une matrice. Sur notre exemple, la valeur de retour doit être 3.

33.3. Écrire une fonction renvoyant le nombre d'éléments impairs de la matrice. Sur notre exemple, la valeur de retour doit être 3.

33.4. Écrire une fonction renvoyant la liste des sommes des éléments de chaque colonne. Sur notre exemple, la valeur de retour doit être $[3,5,7]$.

33.5. Écrire une fonction renvoyant la liste des sommes des éléments de chaque ligne. Sur notre exemple, la valeur de retour doit être $[12,3]$.

Exercice 34

Un tableau `t` à double entrée à n lignes et n colonnes est dit symétrique si, lorsqu'on note $c_{i,j}$ le coefficient à la ligne i et à la colonne j : $\forall i,j \in [0,n-1], c_{j,i} = c_{i,j}$

34.1. Écrire une fonction dont le paramètre est le tableau `t` et qui renvoie un booléen indiquant si `t` est symétrique ou non.

Exercice 35

35.1. Écrire une fonction permettant de transposer un tableau à double entrée, c'est–à–dire que si `t` est un tableau à p lignes et q colonnes, on doit calculer le tableau à q lignes et p colonnes tel que pour tout $i \in [0,q-1]$, pour tout $j \in [0,p-1]$, l'élément de la ligne i et de la colonne j du tableau final est l'élément ligne j colonne i du tableau de départ.

Exercice 36 Élection : max du min

p électeurs $e_0,\ldots,e_{p-1}$ participent à une élection. Il y a q candidats $c_0,\ldots,c_{q-1}$. Le vote se déroule ainsi : chaque électeur e_i ($i \in [0,p-1]$) attribue une note $n_{i,j}$ à chaque candidat c_j ($j \in [0,q-1]$). Le candidat retenu est celui dont la note la plus basse (parmi les notes qui lui sont attribuées) est plus haute (parmi les différents candidats). Remarquer qu'un tel candidat n'est pas nécessairement unique. Dit autrement, un candidat c_j est retenu si :

$$\forall k \in [0,q-1], \min\{n_{0,j},\ldots,n_{p-1,j}\} \geqslant \min\{n_{0,k},\ldots,n_{p-1,k}\}$$

On mémorise dans un tableau à double entrée les notes $n_{i,j}$ attribuées (avec i en indice de ligne et j en indice de colonne).

36.1. Écrire une fonction `python` dont le paramètre est le tableau à double entrée des notes et dont la valeur de retour est la liste des candidats retenus.

36.2. Écrire une fonction `python` `ajout_colonne` dont les deux paramètres sont un tableau à double entrée `t` à p lignes, et une liste `c` de longueur p, et qui modifie le tableau `t` en rajoutant `c` comme une nouvelle colonne apparaissant à droite de `t`. Il n'y a pas de valeur de retour ; c'est le tableau `t` qui est modifié.

Exercice 37 Carré magique

Un *carré magique* est une matrice de n lignes et n colonnes, dont les coefficients sont les entiers appartenant à $[1, n^2]$ (chaque entier apparaissant une et une seule fois) et telle que les sommes des coefficients de chaque ligne, de chaque colonne et des deux diagonales du carré sont toutes les mêmes.

Par exemple

4	9	2
3	5	7
8	1	6

est un carré magique : les sommes des coefficients de chaque ligne, de chaque colonne et des deux diagonales sont toutes égales à 15, et chaque entier de $[1, 9]$ apparaît une et une seule fois.

37.1. Écrire une fonction qui permet de savoir si les sommes de chaque ligne, de chaque colonne et des deux diagonales sont toutes les mêmes.

37.2. Écrire une fonction qui permet de savoir si chaque élément de $[1, n^2]$ apparaît une et une seule fois.

3 Terminaison et correction

Exercice 38

Soit l'algorithme suivant :

```
f(n : entier)
  Données : x,y : entiers
  x ← 0
  y ← n
  tant que y > 0 faire
  | x ← x+3
  | y ← y-1
  fin
  retourner x
```

38.1. Qu'effectue cet algorithme ?
38.2. Prouver sa terminaison.
On considère l'invariant :

$$(I) : \text{à chaque tour de boucle } x = (n - y) * 3.$$

38.3. Prouver la correction de cet algorithme.

Exercice 39

Soit l'algorithme suivant :

```
g(n,b : entier)
  Données : x,y : entiers
  x ← 0
  y ← n
  tant que y ⩾ b faire
  | x ← x+1
  | y ← y-b
  fin
  retourner x
```

39.1. Qu'effectue cet algorithme ?
39.2. Prouver sa terminaison.
On considère l'invariant :

$$(I) : \text{à chaque tour de boucle } n = x * b + y.$$

39.3. Prouver la correction de cet algorithme.

Exercice 40

40.1. Écrire une fonction permettant de calculer le minimum d'un tableau d'entiers passé en paramètre.
40.2. Prouver la terminaison et la correction de votre algorithme.

Exercice 41

41.1. Écrire une fonction prenant en paramètre un tableau d'entiers et retournant `Vrai` si le tableau est trié par ordre croissant.

41.2. Prouver la terminaison et la correction de votre algorithme.

Exercice 42

```
ra(n : entier)
  Données : d,s : entiers
  d ← 0
  s ← 1
  tant que s ⩽ n faire
  |  d ← d+1
  |  s ← s + 2*d + 1
  fin
  retourner d
```

42.1. Quel résultat renvoie cet algorithme si on lui passe en paramètre 17, 25 et 48 ?

42.2. Quel résultat renvoie cet algorithme si on lui donne en paramètre un entier positif n ?

42.3. Justifier sa terminaison.

On considère l'invariant : $(I) : s = (d+1)^2$.

42.4. Montrer la correction de cet algorithme.

Exercice 43

Soit l'algorithme suivant :

```
fusion(t1 : entier[], t2 : entier[])
  Données : i : entier, res : entier[]
  res ← entier[taille(t1)+taille(t2)]
  i ← 0
  i1 ← 0
  i2 ← 0
  tant que i1 < taille(t1) et i2 < taille(t2)
   faire
     si t1[i1] < t2[i2] alors
        res[i] ← t1[i1]
        i ← i+1
        i1 ← i1+1
     sinon
        res[i] ← t2[i2]
        i ← i+1
        i2 ← i2+1
     fin
  fin
  pour j de i1 à taille(t1)-1 faire
     res[i] ← t1[j]
     i ← i+1
  fin
  pour j de i2 à taille(t2)-1 faire
     res[i] ← t2[j]
     i ← i+1
  fin
  retourner res
```

43.1. En supposant que $t1$ et $t2$ sont deux tableaux d'entiers triés par ordre croissant, que calcule cet algorithme ?

43.2. Prouver la terminaison de ce programme.

4 Récursivité

Exercice 44

44.1. Écrivez une fonction récursive `multiplie(n,m)` qui retourne le produit de 2 entiers **positifs** `m` et `n` (vous n'avez le droit d'utiliser que l'opération +).

44.2. Écrivez une fonction récursive `puissance(n,m)` qui retourne `n` à la puissance `m`.

44.3. Écrivez une fonction `pgcd(n,m)` prenant en paramètre 2 entiers positifs `n` et `m` et retournant leur plus grand diviseur commun.

44.4. Écrivez une fonction récursive `factorielle(n)` prenant en paramètre un entier positif `n` et retournant `n!`.

Rappel : $0! = 1$ *et* $n! = n \times (n-1) \times \cdots \times 1$.

Exercice 45

45.1. Écrivez une fonction récursive `est_pair` prenant un paramètre un entier et retournant `Vrai` si cet entier est pair et `Faux` sinon.

45.2. Écrivez une fonction récursive `est_multiple` prenant un paramètre deux entiers `n` et `m`, et retournant `Vrai` si `n` est multiple de `m` et `Faux` sinon.

Exercice 46

Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = \dfrac{3}{8u_n + 2} \end{cases}$$

46.1. Écrivez une fonction `calcul_u` récursive prenant en paramètre un entier `n` et renvoyant la valeur de u_n.

Exercice 47

Soit v_n définie par $\begin{cases} v_0 = 1 \\ v_1 = 3 \\ \forall n \in \mathbb{N}, v_{n+1} = v_{n-1} + 7 \end{cases}$.

47.1. Écrire une fonction récursive `calcul_v` qui prend en paramètre un entier `n` et qui renvoie la valeur de v_n.

Exercice 48

Soit la suite $(w_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} w_0 = 0 \\ w_{n+1} = \dfrac{w_n + 1}{w_n + 2} \end{cases}$$

48.1. Écrivez une fonction `calcul_w` récursive prenant en paramètre un entier `n` et renvoyant la valeur de w_n.

Exercice 49

49.1. Écrire une fonction récursive `sommeCarres` qui prend en paramètre un entier `n` et qui renvoie la somme $S_n = \sum_{i=0}^{n} i^2$.

Indication : que vaut S_0 ? Que vaut S_n en fonction de S_{n-1} ?

Exercice 50

Pour trier un tableau d'entier (on notera `n` sa taille), on propose la méthode suivante :

1. on commence par déterminer $i_{\mathtt{M}}$, l'indice du maximum du tableau (entre les indices `0` et `n-1`)
2. on échange les valeurs des cases d'indice $i_{\mathtt{M}}$ et `n-1` (le maximum du tableau se trouve alors dans la case d'indice `n-1`)
3. puis on trie le tableau entre les indices `0` et `n-2`

50.1. Écrire une fonction `recherche_indice_maximum(tab,m)` prenant en paramètre un tableau d'entier `tab` et un indice maximum `m` et retournant l'indice du maximum de `tab` entre les indices `0` et `m` (inclus).

50.2. Écrire une fonction récursive `tri_tableau_rec(tab,m)` prenant en paramètre un tableau d'entier `tab` et un indice `m` et triant `tab` par ordre croissant entre les indices `0` et `m`.

Ce tri s'effectuera en place, c'est à dire que la fonction ne retournera rien, mais modifiera le tableau.

50.3. Écrire une fonction `tri_tableau(tab)` prenant en paramètre un tableau d'entier `tab` et triant `tab` par ordre croissant (on fera appel à `tri_tableau_rec` avec les bons paramètres).

Exercice 51

51.1. Écrivez une fonction `recherche` prenant en paramètre un tableau d'entiers `tab` et un entier `el` et retournant `Vrai` si l'élément `el` est contenu dans `tab`.

On considère maintenant que `tab` est un tableau trié.
Pour accélérer la recherche, on décide d'utiliser un algorithme utilisant le principe de recherche suivant : on cherche `el` dans une portion du tableau (entre les indices `i` (inclus) et `j` (exclus)) :

- si `i` = `j-1`, alors il n'y a qu'un seul élément à tester.
- sinon :
  — on calcule `m` ← `(i+j)//2`
  — si `tab[m]` > `el` alors on cherche entre `i` et `m`
  — sinon on cherche entre `m` et `j`

51.2. Écrivez une fonction récursive `recherche_trie_aux(tab,el,i,j)` qui retourne `Vrai` si `el` est présent dans `tab` entre les indices `i` (inclus) et `j` (exclus).

51.3. Écrivez une fonction `recherche_trie(tab,el)` qui retourne `Vrai` si `el` est présent dans `tab` (cette fonction appellera la fonction précédente avec les bons paramètres).

51.4. Justifier de la terminaison de cette fonction.

51.5. En supposant que le tableau contient 2^n éléments, combien d'appel à `recherche_trie_aux` provoque `recherche_trie(tab,el)` ?

Exercice 52

Un avion décolle à la date T. Ensuite, à chaque date T, $T + dt$, $T + 2dt, \ldots$ il envoie le signal `'OK'`. Subitement la réception s'interrompt : l'avion n'émet plus de signal !
La tour de contrôle a reçu la liste :

$$['\texttt{OK}','\texttt{OK}','\texttt{OK}','\texttt{OK}','\texttt{OK}','\texttt{OK}',\texttt{None},\texttt{None},\texttt{None}]$$

On souhaite déterminer à quelle date ont cessé les émissions, c'est à dire l'indice du premier `None`.

52.1. Écrire une fonction `date` qui prend en entrée la liste reçue par la tour de contrôle et qui renvoie le premier entier k pour lequel on a enregistré `None` à la date $T + k.dt$ en agissant par *dichotomie*.

Exercice 53

53.1. Écrivez une fonction récursive `nombre_de_chiffres(n)` retournant le nombre de chiffre de `n` (on considérera que `n` est un entier positif ou nul).
Exemple : nombre_de_chiffre(4235) doit retourner 4.
Montrez la terminaison de votre fonction.

53.2. Écrivez une fonction récursive `nombre_d_element(tab,i,el)` qui retourne le nombre d'éléments de `tab[0:i]` égaux à `el`.

Exemple : nombre_d_element([0,1,2,1,3,1],3,1) doit retourner 2.
Montrez la terminaison de votre fonction.

53.3. Écrivez en `python` une fonction récursive `fusion` prenant en paramètre deux tableaux `tab1` et `tab2` triés par ordre croissant et retournant la fusion triée des deux tableaux.
Exemple : `fusion([1,2,7],[4,5,8,9]` retourne [1,2,4,5,7,8,9]
Montrez la terminaison de votre fonction.

Travaux Pratiques

1 Découverte de Python

Exercice 1

L'objectif de ce TP est de vous familiariser avec Python.
`python` peut être lancé de deux manières :

- via la console : tapez `python3` dans un terminal
- en exécutant un fichier : tapez `python3 nom_fichier.py` dans un terminal

1.1. Dans la console, tapez les commandes suivantes :

```
>>> 5+7
```

```
>>> 34/9
```

```
>>> min(3,5)
```

Les variables s'affectent par le signe `=` :

```
>>> a = 3+4
```

L'affichage d'une variable dans la console se fait juste en tapant son nom ou `print(nom_variable)`

```
>>> a
```

```
>>> print(a)
```

En `python`, les deux valeurs binaires sont `True` et `False`.

1.2. (Opérateurs binaires) Exécutez les commandes suivantes :

- `True and False`
- `True or False`
- `not(True) or False and True`

$==$	teste l'égalité entre deux valeurs
$!=$	teste la différence entre deux valeurs
$<, <=, >, >=$	teste les relations d'ordre entre deux valeurs

Comparaisons

Exercice 2 `if`, `while`, `for`, `range` et fonctions

Voici un exemple d'utilisation en python de l'instruction `if` :

```python
if x>0 :
        y=x
        print("Cas positif")
elif x<0 :
        y=-x
        print("Cas négatif")
else :
        y=0
        print("Cas nul")
print("Fin")
```

Noter l'absence d'accolades et l'utilisation de l'indentation : les blocs de l'instruction conditionnelle ne sont distinguables que par l'indentation. Il en est de même pour les boucles `for`, `while` et pour l'écriture des fonctions.
Voici un exemple de fonction écrite en python :

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

2.1. Dans un fichier `tutoriel.py`, écrivez une fonction g permettant de calculer :

$$g(x) = \begin{cases} -x^2 & \text{si } x < 0 \\ x^2 & \text{sinon} \end{cases}$$

`python` est surtout un langage de script. Si vous souhaitez tester votre fonction, il vous suffit de rajouter, par exemple,

```
print(g(-4))
```

à la fin de votre fichier et de le lancer dans le terminal par la commande :

```
$ python3 tutoriel.py
```

En cours, nous avons vu que que la commande `for` s'utilise de la manière suivante :

```python
for i in range(a,b) :
        ...
```

2.2. Afficher ce que donne les commandes :

-
```python
for i in range(5,10) :
        print(i)
```

-
```python
for i in range(6) :
        print(i)
```

-
```python
for i in range(30,3,-8) :
        print(i)
```

2.3. Écrivez une fonction h prenant en paramètre un entier positif x et affichant tous les entiers multiples de 7 compris entre 0 et x inclus.

Remarque : dans la suite des TPs, on n'utilisera pas `python` *directement dans le terminal. Les fonctions seront à écrire dans des fichiers et seront testées et utilisées à la fin de ces fichiers, qu'on exécutera en entier.*
Exemple :

```python
def f(a,b) :
    ...

def g(a) :
    ...

...

print("f(3,\"chat\") =",f(3,"chat"))
...
```

2 Instructions de bases, entrées-sorties

Exercice 3 Ou exclusif

Il n'y a pas dans le cours de test "ou exclusif" (ou xor) qui renvoie `VRAI` si et seulement si l'un des deux paramètres est vrai, mais pas les deux simultanément.

3.1. Écrivez une fonction `python` prenant en paramètres 2 booléens et renvoyant le "ou exclusif" de ces deux paramètres.

Exercice 4

4.1. Chacun de ces programmes est faux. Préciser l'erreur.

a.
```
a = 5
if a%0==2 : print ("Nbr pair")
```

b.
```
a == 5
if a : print ("Nbr non nul")
```

Exercice 5

Considérons le petit programme suivant :

```
a = 'False'
if a :
    print(4)
else :
    print(8)
```

5.1. À l'exécution le programme affiche 4. Pourquoi ?

Exercice 6

6.1. Écrire une fonction prenant en paramètre un entier n et qui permet de savoir si n est pair ou non : si l'entier n est pair, l'ordinateur doit afficher "n est pair", sinon l'ordinateur doit afficher "n est impair" (la fonction ne doit rien renvoyer).

Tester ce code lorsque `n` = `46` puis lorsque `n` = `47`.

Exercice 7

7.1. Écrire une fonction prenant en paramètre un entier n, qui affecte à une variable `res` la valeur 1 si n est un multiple de 3 et 0 si n n'est pas un multiple de 3, et qui renvoie `res`.

Tester votre programme avec $n = 15$ puis avec $n = 19$.

Exercice 8 Payer ses impôts

Lorsqu'on ne dépasse pas la tranche à $5,5\%$, le montant de l'impôt que l'on doit payer, en fonction de son revenu pour une personne ne possédant qu'une seule part, est défini par la formule :

$$impot = \begin{cases} 0 & \text{si } revenu < 5875 \\ \frac{5.5}{100}(revenu - 5875) & \text{sinon} \end{cases}$$

8.1. Écrire une fonction `calcul_impot_1` prenant en paramètre le revenu et renvoyant le montant de l'impôt à payer.

Tant qu'on ne dépasse pas la tranche à 14%, le montant de l'impôt est

$$impot = \begin{cases} 0 & \text{si } revenu < 5875 \\ \frac{5.5}{100}(revenu - 5875) & \text{si } 5875 \leqslant revenu < 11\,720 \\ \frac{14}{100}revenu - 1\,319.33 & \text{sinon} \end{cases}$$

8.2. Écrire une fonction `calcul_impot_2` prenant en paramètre le revenu et renvoyant le montant de l'impôt à payer selon ce nouveau calcul, en utilisant une instructions `elif`.

8.3. Faire des tests dans chacun des trois cas suivants :

- Le revenu vaut 3 000 (l'impôt calculé doit valoir 0)
- Le revenu vaut 7 000 (l'impôt calculé doit valoir 61.875)
- Le revenu vaut 20 000 (l'impôt calculé doit valoir 1 480.67)

8.4. Écrire une fonction `calcul_impot_3` effectuant le même calcul que `calcul_impot_2`, mais sans utiliser d'instruction `elif`.

Refaire les trois tests de la question ci–dessus et vérifier la validité.

Pour un revenu quelconque (toujours avec une seule part), l'impôt est :

$$impot = \begin{cases} 0 & \text{si } revenu < 5875 \\ \frac{5.5}{100}(revenu - 5875) & \text{si } 5875 \leqslant revenu < 11\,720 \\ \frac{14}{100}revenu - 1\,319.33 & \text{si } 11\,720 \leqslant revenu < 26\,030 \\ \frac{30}{100}revenu - 5\,484.13 & \text{si } 26\,030 \leqslant revenu < 69\,783 \\ \frac{40}{100}revenu - 12\,462.43 & \text{au–dessus} \end{cases}$$

8.5. Écrire une fonction `calcul_impot_4` prenant en paramètre le revenu et renvoyant le montant de l'impôt à payer selon ce nouveau calcul.

Sur Mars, le calcul de l'impôt dépend du revenu et d'un booléen permettant de savoir si la personne imposée habite dans un cratère ou non.

Le calcul de l'impôt est :

	`cratere`	
	`True`	`False`
`revenu` $< 4\,800$	$0.12 *$ `revenu`	0
`revenu` $\geqslant 4\,800$	$0.25 *$ `revenu` $- 624$	$0.12 *$ `revenu` $- 576$

8.6. Écrire une fonction `calcul_impots_mars` prenant en paramètre un entier `revenu` et un booléen `cratere` et renvoyant le montant de l'impôt.

Effectuer les tests suivants :

- un martien habitant un cratère et touchant un revenu de 4 000 doit payer 480 ;
- un martien habitant un cratère et touchant 6 000 doit payer 876 ;
- un martien n'habitant pas de cratère et touchant 6 000 doit payer 144.

Exercice 9 Importance de l'indentation

On cherche à calculer $\sum_{i=1}^{10} \frac{1}{i}$. Considérons les deux programmes suivants :

```
s = 0
for i in range (1, 11) :
    s += 1/i
print (s)
```

```
s = 0
for i in range (1, 11) :
    s += 1/i
    print (s)
```

9.1. Quelle différence constate–t–on lors de l'exécution entre les deux programmes ?

Exercice 10 Calculs de sommes et de produits

Soient $S(n) = \sum_{k=1}^{n} \frac{1}{k}$ et $H(n) = \prod_{k=2}^{n} \left(1 - \frac{1}{k^2}\right)$ (pour $n \geqslant 2$).

10.1. Nous allons calculer $S(n)$ avec l'algorithme suivant :

Algorithme 5 : somme_S(n : entier)

Données :
k : entier
s : réel
s ← 0
k ← 0
tant que $k < n$ **faire**
  k ← k+1
  s ← s+1/k
fin
retourner s

10.2. Programmer cet algorithme en `python`.

10.3. Test : vérifier que $S(10) \simeq 2.9289$

10.4. Si dans le test de la boucle `while` on avait remplacé le test $k < n$ par $k \leqslant n$, qu'aurait calculé ce programme ?

10.5. Modifier l'algorithme de la question précédente pour calculer le produit $H(n)$.

Test : vérifier que $H(5) = 0.6$.

10.6. Déterminer le plus petit entier n tel que $S(n) \geqslant 10$. On pourra par exemple appliquer l'algorithme suivant :

Données :
n : entier
s : réel
n ← 0
s ← 0
tant que $s <$ *10* **faire**
  n ← n+1
  s ← s+1/n
fin
retourner n

Élement de réponse : on doit trouver un entier compris entre 10 000 et 100 000.

Le savant Sunisoc conjecture que $\lim_{n\to+\infty} H_n = \frac{1}{2}$ (cette conjecture est vraie).
On fixe un réel $\varepsilon > 0$ "petit" et on cherche à déterminer le plus petit entier $n \geqslant 2$ tel que $\frac{1}{2} - \varepsilon \leqslant H_n \leqslant \frac{1}{2} + \varepsilon$.
10.7. Écrire un programme permettant de résoudre automatiquement ce problème.
Test : pour $\varepsilon = 0.002$ on doit trouver $n = 250$.

Exercice 11 Itérés d'une suite récurrente d'ordre 1

On définit une suite u_n par $u_0 = 0$ et $\forall n \in \mathbb{N}, u_{n+1} = \dfrac{6+u_n}{6-u_n}$.
Nous allons calculer u_n avec l'algorithme suivant :

Algorithme 6 : calcul_u(n : entier)

```
Données : u : réel
u ← 0
tant que n > 0 faire
    n ← n-1
    u ← (6+u)/(6-u)
fin
retourner (u)
```

11.1. Programmer cet algorithme en `python`.
Test : vérifier que $u_5 \simeq 1.812$.
Si dans le test de la boucle `while` on avait remplacé le test $n > 0$ par $n \geqslant 0$, qu'aurait calculé ce programme ?
Et si on avait mis le test $n < 0$?
11.2. Calculer $\sum_{k=0}^{n} u_k$.
Test : pour $n = 5$ on doit trouver environ 7.5534

Exercice 12 Gestion de matchs

On souhaite gérer les matchs au cours d'une année d'une ligue de sport.
Rappel : usage d'une commande d'impression : Si `i = 3` et `j = 2` sont définis, la commande :

```
print ("L'equipe",i,"se deplace contre l'equipe",j)
```

permet d'afficher le message : `L'equipe 3 se deplace contre l'equipe 2`
12.1. Une équipe doit jouer contre toutes les autres équipes. Écrire une fonction prenant en paramètre un entier `k` représentant le numéro de l'équipe et un entier `n` représentant le nombre total d'équipes et qui permet d'afficher tous les messages :

```
L'equipe 2 se deplace contre l'equipe 1
L'equipe 2 se deplace contre l'equipe 2
L'equipe 2 se deplace contre l'equipe 3
...
```

12.2. On cherche maintenant à écrire un message pour chacun des matchs à jouer. Écrire une fonction prenant en paramètre un entier `n` représentant le nombre total d'équipes et qui affiche les messages :

```
L'equipe 1 se deplace contre l'equipe 1
L'equipe 1 se deplace contre l'equipe 2
L'equipe 1 se deplace contre l'equipe 3
...
L'equipe 2 se deplace contre l'equipe 1
L'equipe 2 se deplace contre l'equipe 2
L'equipe 2 se deplace contre l'equipe 3
...
L'equipe 3 se deplace contre l'equipe 1
L'equipe 3 se deplace contre l'equipe 2
L'equipe 3 se deplace contre l'equipe 3
...
```

12.3. Dans un vrai tournoi de ligue une équipe ne se déplace pas pour jouer contre elle–même. Rajouter une commande `if` pour n'afficher le message que si les deux numéros d'équipe sont différents. On doit obtenir un résultat du type :

```
L'equipe 1 se deplace contre l'equipe 2
L'equipe 1 se deplace contre l'equipe 3
L'equipe 1 se deplace contre l'equipe 4
...
L'equipe 2 se deplace contre l'equipe 1
L'equipe 2 se deplace contre l'equipe 3
L'equipe 2 se deplace contre l'equipe 4
...
L'equipe 3 se deplace contre l'equipe 1
L'equipe 3 se deplace contre l'equipe 2
L'equipe 3 se deplace contre l'equipe 4
...
```

12.4. Pour cause de budget restreint la ligue de sport décide que deux équipes ne joueront pas un match aller et un match retour mais un seul match sur terrain neutre. Les messages à afficher sont maintenant :

```
Matchs de l'equipe 1

Matchs de l'equipe 2
L'equipe 2 joue contre l'equipe 1

Matchs de l'equipe 3
L'equipe 3 joue contre l'equipe 1
L'equipe 3 joue contre l'equipe 2

Matchs de l'equipe 4
L'equipe 4 joue contre l'equipe 1
L'equipe 4 joue contre l'equipe 2
L'equipe 4 joue contre l'equipe 3
...
```

Pour les plus rapides

Exercice 13

Regardons tous les nombres $1 \leqslant k < 10$ multiples de 3 ou de 5 : il y a 3, 5, 6 et 9. Leur somme vaut 23.

13.1. Calculer la somme de tous les entiers $1 \leqslant k < 1000$ multiples de 3 ou de 5.
Réponse : 233 168

Exercice 14 Transmission imparfaite avec code auto–correcteur

Deux dispositifs informatiques échangent des données : l'émetteur envoie cinq entiers a_1, a_2, a_3, $b = a_1 + a_2 + a_3$ et $c = a_1 + 2a_2 + 3a_3$.
À cause d'imperfections techniques, il y a un risque faible mais non négligeable que l'un de ces entiers ait été modifié pendant la transmission de données : le récepteur reçoit des entiers ar_1, ar_2, ar_3, br et cr dont l'un est potentiellement différent de l'entier émis. Le risque que deux entiers (ou plus) aient été mal transmis est en pratique trop faible pour qu'on en tienne compte.

Votre but est, connaissant les entiers ar_1, ar_2, ar_3, br et cr, de déterminer s'il y a eu ou non une erreur de transmission et, si c'est le cas, de retrouver les valeurs des entiers émis.

14.1. Commençons par envisager le cas où l'un des entiers a_1, a_2 ou a_3 a été mal transmis (mais b et c ont été correctement transmis). Montrer que dans ce cas $br \neq ar_1 + ar_2 + ar_3$ et $cr \neq ar_1 + 2ar_2 + 3ar_3$. Expliquer comment déterminer lequel parmi a_1, a_2 ou a_3 a été mal transmis, et comment corriger l'erreur.

14.2. Que se passe–t–il si a_1, a_2 et a_3 ont été correctement transmis, mais qu'il y a eu une erreur de transmission ou bien sur b ou bien sur c ?

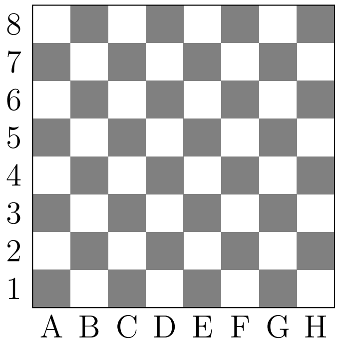
14.3. Écrire une fonction `code_correcteur` prenant en paramètres 5 entiers `ar1`, `ar2`, `ar3`, `br` et `cr`, et qui affiche suivant les cas l'un des messages suivants :

- `Les cinq entiers ont été correctement transmis`
- `a1 a été mal transmis : sa vraie valeur est :` *suivi de la valeur de* a_1
- *(ou un message similaire avec* a_2*,* a_3*, b ou c)*

Tester votre programme dans chacun des cas suivants :

- $ar1 = 3$, $ar2 = 2$, $ar3 = 6$, $br = 10$, $cr = 25$. On doit obtenir le message :
  `b a été mal transmis, sa vraie valeur est : 11`
- $ar1 = 3$, $ar2 = 1$, $ar3 = 6$, $br = 11$, $cr = 25$. On doit obtenir le message :
  `a2 a été mal transmis, sa vraie valeur est : 2`
- $ar1 = 3$, $ar2 = 2$, $ar3 = 6$, $br = 11$, $cr = 25$. On doit obtenir le message :
  `Les cinq entiers ont été correctement transmis`
- $ar1 = 3$, $ar2 = 2$, $ar3 = 5$, $br = 11$, $cr = 25$. On doit obtenir le message :
  `a3 a été mal transmis, sa vraie valeur est : 6`

Exercice 15 Mouvement du cavalier sur un échiquier



Un échiquier est composé de 8×8 cases, chaque case étant repérée par son abscisse (A, B, C,. . ., H) et par son ordonnée (1, 2, 3,. . ., 8).
Nous allons coder chaque case par une chaîne de 2 caractères : par exemple la valeur `"F4"` désignera une case (ne pas oublier les guillemets).

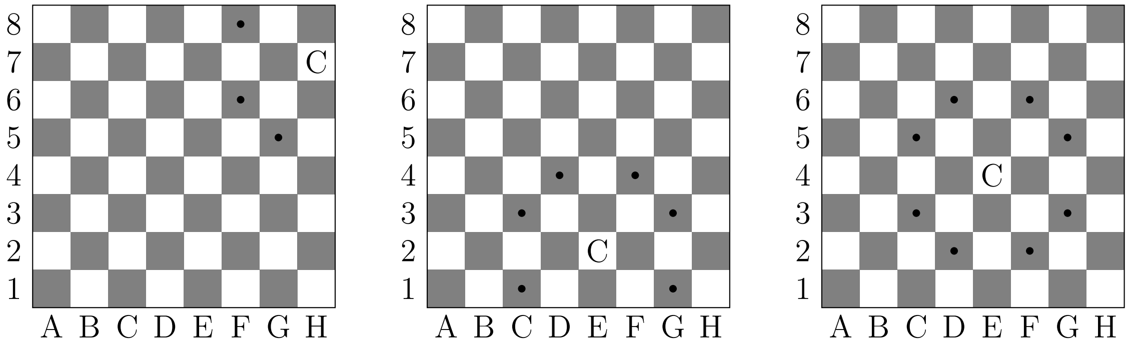
15.1. Tester les instructions suivantes (inutile de recopier les commentaires derrière le symbole `#`) :

```
>>> c1 = "F4"           # definition d'une case c1
>>> x1 = c1[0]          # abscisse de c1 : attention aux crochets
>>> y1 = c1[1]          # ordonnee de c1
>>> type (c1)           # type str (string) : chaine de caracteres
>>> x1
>>> type (x1)
>>> y1
>>> type (y1)           # encore de type str
>>> c2 = "G2"           # definition d'une autre case c2
>>> x2 = c2[0]
>>> y2 = c2[1]
>>> y1 - y2             # attention : erreur
>>> y1 = int (y1)       # convertion en int (integer : entier)
>>> type (y1)           # la convertion a bien eu lieu
>>> y2 = int (y2)
>>> y1 - y2             # maintenant ca marche
>>> x1 - x2             # erreur, bien sur
>>> ord (x1)            # code entier du caractere "F"
>>> ord (x1) - ord (x2) # calcul de la difference des ordonnees
>>> chr (66)            # reciproque de la fonction ord
>>> str (8)             # convertion d'un entier en str
>>> chr(67) + str (8)   # concatenation (mise bout-a-bout)
```

Un cavalier peut se déplacer (en un seul mouvement) d'une case c_1 vers une case c_2 exactement dans chacun des deux cas suivants :

- ou bien leur abscisse diffère de 2 et une ordonnée diffère de 1 ;
- ou bien leur abscisse diffère de 1 et leur ordonnée diffère de 2.

Bien sûr il est nécessaire que les deux cases c_1 et c_2 soient des cases qui existent sur l'échiquier ! Voici ci–dessous trois cavaliers (repérés par la lettre “C”) ainsi que, à chaque fois, les cases sur lesquelles il peut se déplacer.



15.2. Écrire une fonction `deplacement_possible` prenant en paramètre deux chaînes de caractères `c1` et `c2` représentant des cases de l'échiquier et qui renvoie `True` si un cavalier peut passer de la case `c1` à la case `c2`. On pourra utiliser la fonction `abs` qui calcule la valeur absolue.

Effectuer les tests suivants :

- Pour les cases `"F4"` et `"G2"` le déplacement est possible ;
- pour les cases `"A3"` et `"B4"` le déplacement est impossible ;
- pour les cases `"G7"` et `"I8"` le déplacement est impossible (la case `"I8"` n'existe pas).

15.3. Écrire une fonction `cases_possible` prenant en paramètre une chaîne de caractères `c1` représentant une case existante et qui affiche les cases possibles accessibles par un cavalier en un seul mouvement depuis `c1`.

Tester votre programme de déplacement du cavalier :

- Depuis la case `"H1"` votre programme doit afficher les cases `"G3"` et `"F2"` ;
- depuis la case `"D2"` votre programme doit afficher les cases `"B1"`, `"B3"`, `"C4"`, `"E4"`, `"F3"` et `"F1"`.

Exercice 16

Soient $k, n \in \mathbb{N}$ tels que $k \leqslant n$.

Le coefficient binomial $\binom{n}{k}$ peut se calculer avec la récurrence :

$$\binom{n}{0} = 1 \text{ et } \binom{n+1}{k+1} = \frac{n+1}{k+1}\binom{n}{k}$$

16.1. Écrire une fonction `binomial` prenant en paramètre deux entiers positif `n` et `k` et retournant la valeur de $\binom{\mathtt{n}}{\mathtt{k}}$.

On ne demande pas de prouver cette formule, mais de l'appliquer pour calculer $\binom{15}{10}$ *(réponse : 3003).*

Exercice 17

Les nombres triangulaires T_n, pentagonaux P_n et hexagonaux H_n sont définis ainsi :

Triangulaires : $T_n = \dfrac{n(n+1)}{2}$ (on trouve $1, 3, 6, 10, 15 \ldots$)

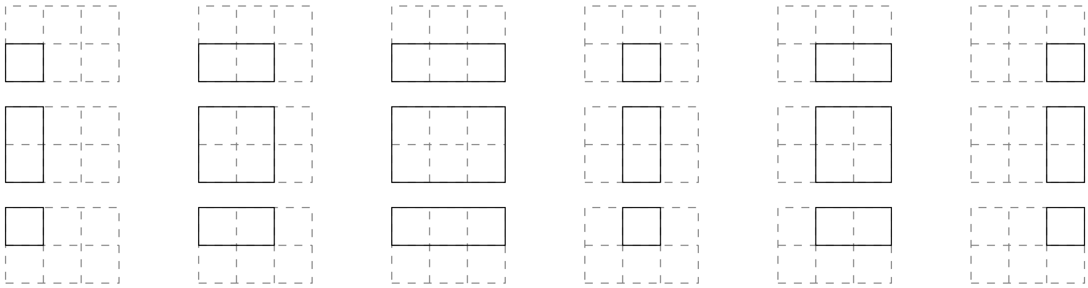
Pentagonaux : $P_n = \dfrac{n(3n-1)}{2}$ (on trouve $1, 5, 12, 22, 35, \ldots$)

Hexagonaux : $H_n = n(2n-1)$ (on trouve $1, 6, 15, 28, 45, \dots$)
Il est possible de vérifier que $T_{285} = P_{165} = H_{143} = 40755$.

17.1. Trouver le nombre suivant qui est à la fois triangulaire, pentagonal et hexagonal (il est supérieur à un milliard...)

Exercice 18 Dénombrer les rectangles

Constatons que sur une grille de taille 3×2 on peut positionner 18 rectangles :



Il n'est pas possible de trouver de grille contenant exactement 2 000 000 (2 millions) de rectangles.

18.1. Calculer cependant les dimensions d'une grille qui s'en approche le plus.

3 Tableaux

Exercice 19 Aidons Rakham le Rouge

Le pirate Rakham le Rouge a trouvé une carte au trésor écrite en ces termes : “50 pas au nord, 20 pas à l'est, 30 pas au sud, 82 pas à l'est, 48 pas au nord, 43 pas à l'ouest, 51 pas au sud, 18 pas au nord, 46 pas à l'est, et tu trouveras le trésor.”

Il est clair qu'il lui suffit d'effectuer 35 pas au nord et 105 pas à l'est pour trouver le trésor.

19.1. Écrire un programme qui, si on lui donne comme paramètre la liste :

`[[50,'n'], [20,'e'], [30,'s'], [82,'e'], [48,'n'], [43,'o'], [51,'s'], [18,'n'], [46,'e']]`

calcule la liste `[[35,'n'], [105,'e']]`.

Exercice 20 Jean Petit qui danse

20.1. Une chaîne de caractères possède un et un seul caractère “espace”. Écrire une fonction qui renvoie la partie de la chaîne se trouvant derrière cet espace.

Par exemple si la valeur d'entrée est `"sa main"` la fonction doit renvoyer `"main"`.

Remarque : il est demandé de gérer ceci avec une boucle, en parcourant les caractères jusqu'à trouver l'espace. Nous verrons dans un autre chapitre qu'il existe des fonctions permettant de faire ce genre de traitement automatiquement, mais ici il est demandé de trouver une solution “à la main”.

20.2. “Jean Petit qui danse” est une chanson populaire. À chaque couplet on ajoute une partie du corps de Jean Petit avec laquelle il danse.

On fournit la liste des parties du corps de Jean Petit, il s'agit de créer le texte de la chanson en une seule chaîne de caractères. Si par exemple on fournit en entrée la liste `["son doigt", "sa main", "ses cheveux"]` il s'agit de créer une chaîne dont l'affichage avec l'instruction `print` donnera :

```
Jean Petit qui danse (bis)
De son doigt il danse (bis)
De son doigt, doigt, doigt
Ainsi danse Jean Petit.

Jean Petit qui danse (bis)
De sa main il danse (bis)
De sa main, main, main
De son doigt, doigt, doigt
Ainsi danse Jean Petit.

Jean Petit qui danse (bis)
De ses cheveux il danse (bis)
De ses cheveux, cheveux, cheveux
De sa main, main, main
De son doigt, doigt, doigt
Ainsi danse Jean Petit.
```

Rappelons que le retour à la ligne se code avec "\n" ; et que la concaténation de chaînes se fait avec le symbole +.

Exercice 21 Sommes simples ou doubles

21.1. Écrire une fonction prenant en paramètre un entier n, qui contient une boucle `for` et qui calcule $\sum_{k=1}^{n} \frac{k^2}{k^2+1}$.

Test : pour $n = 5$ on doit trouver environ 4.10271

21.2. Écrire une fonction prenant en paramètre un entier n et qui retourne la liste des valeurs $\frac{n+i}{n^2+i^2}$ pour toutes les valeurs de i comprises entre 1 et n inclus. Dit autrement il s'agit de générer la liste :

$$\left[\frac{n+1}{n^2+1}, \frac{n+2}{n^2+4}, \ldots, \frac{n+n}{n^2+n^2}\right]$$

Test : pour $n = 5$ on doit trouver (aux arrondis près)

$$[0.23077, 0.24138, 0.23529, 0.21951, 0.2]$$

21.3. Écrire une fonction prenant en paramètre un entier n, qui affiche avec une commande `print` la liste $\left[\frac{k+1}{k^2+1^2}, \frac{k+2}{k^2+2^2}, \ldots, \frac{k+k}{k^2+k^2}\right]$, et ceci itérativement pour toutes les valeurs $k \in [1, n]$.

Test : avec $n = 5$ on doit obtenir l'affichage (aux arrondis près)

```
[1.0]
[0.6, 0.5]
[0.4, 0.38462, 0.33333]
[0.29412, 0.3, 0.28, 0.25]
[0.23077, 0.24138, 0.23529, 0.21951, 0.2]
```

21.4. Écrire un programme prenant en paramètre un entier n, et qui calcule $\sum_{k=1}^{n} \sum_{i=1}^{k} \frac{k+i}{k^2+i^2}$.

On imbriquera deux boucles `for` l'une dans l'autre.

Test : pour $n = 5$ on doit trouver environ 5.46902

Exercice 22 Codes de Gray

Pour tout $n \in \mathbb{N}^*$, on définit G_n qui est une liste de listes par récurrence :

- $G_1 = [[0], [1]]$
- Soit $n \geqslant 2$, soient ℓ_1 et ℓ_2 deux copies de G_{n-1}. À la fin de chacun des éléments de ℓ_1 (qui sont des listes), on concatène un 0. À la fin de chacun des éléments de ℓ_2 on concatène un 1. Ensuite on renverse ℓ_2 et on concatène ℓ_1 et ℓ_2 : le résultat est G_n.

Exemple pour $n = 2$.

- On part de $\ell_1 = [[0], [1]]$ et $\ell_2 = [[0], [1]]$.
- On concatène 0 à la fin de chaque élément de ℓ_1 et 1 à la fin de chaque élément de ℓ_2 : on obtient $\ell_1 = [[0, 0], [1, 0]]$ et $\ell_2 = [[0, 1], [1, 1]]$.

- On renverse ℓ_2 et on concatène : il vient $G_2 = [[0,0],[1,0],[1,1],[0,1]]$.

Vérifier que $G_3 = [[0,0,0],[1,0,0],[1,1,0],[0,1,0],[0,1,1],[1,1,1],[1,0,1],[0,0,1]]$.

22.1. Écrire une fonction prenant en paramètre un entier `n` qui permet de calculer $G_{\mathtt{n}}$.

Remarque : un *hypercube de dimension* 2 est un carré, un *hypercube de dimension* 3 est un cube. Le code de Gray G_n permet de parcourir les sommets de l'hypercube de dimension n en passant une et une seule fois par chaque sommet et en suivant uniquement des arêtes.

Exercice 23 Diviseurs d'un nombre

23.1. Écrire une fonction prenant en paramètre un entier n et qui permet de déterminer le plus petit diviseur $k \geqslant 2$ de n. On pourra implanter l'algorithme suivant :

```
début
  k ← 2
  tant que k ne divise pas n faire
    k ← k + 1
  fin
fin
```

Vérifier que si $n = 35$ la valeur calculée de k est 5.

23.2. Adapter l'algorithme précédent de façon à calculer le plus grand diviseur $k < n$ de n. Par exemple si $n = 35$ on doit trouver $k = 7$.

On suppose connu un autre entier $p > 2$, non premier avec n (c'est–à–dire que p et n ont des diviseurs communs supérieurs ou égaux à 2).

23.3. Écrire une fonction prenant en paramètre deux entiers p et n (que l'on supposera non premiers entre eux) qui calcule le plus petit entier $k \geqslant 2$ qui divise à la fois n et p.

Vérifier que si $n = 30$ et $p = 25$ alors l'entier k vaut 5.

Cette fois–ci on ne sait pas si p et n sont premiers entre eux ou non.

23.4. Réécrire le programme précédent, mais cette fois–ci on affichera (avec `print`) la valeur de k si un tel plus petit diviseur commun existe, ou "n et p sont premiers entre eux" si un tel diviseur commun n'existe pas.

<u>Indication</u> : on pourra remarquer que si un tel entier k existe alors nécessairement $k \leqslant n$ et $k \leqslant p$.

Vérifier que si $n = 30$ et $p = 25$ alors k vaut 5, alors que $n = 48$ et $p = 35$ sont premiers entre eux.

23.5. Écrire une fonction prenant en paramètre un entier n et qui calcule la liste de tous les diviseurs positifs de l'entier n.

Test : si $n = 12$ on doit trouver la liste $[\mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{6}, \mathtt{12}]$.

23.6. Écrire une fonction prenant en paramètre un entier n et qui calcule la somme des diviseurs positifs de l'entier n. Par exemple pour $n = 12$ on doit trouver 28.

Un nombre est dit *parfait* s'il est égal à la somme de ses diviseurs positifs (sauf lui–même). Par exemple 6 est parfait car $6 = 1 + 2 + 3$.

23.7. Écrire une fonction prenant en paramètre un entier n et qui détermine si l'entier n est parfait ou non.

23.8. Écrire une fonction prenant en paramètre deux entiers p et q, et qui calcule la liste de tous les nombres parfaits entre p et q. On vérifiera qu'entre 2 et 40 seuls 6 et 28 sont parfaits.

Exercices plus difficiles

Exercice 24 Étude d'un arbre généalogique

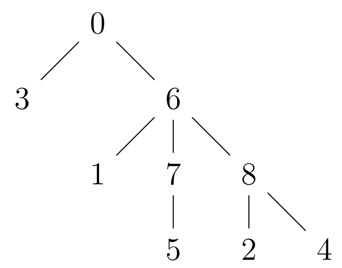
Dans un palais en ruine de la forêt inexplorée de Murzanie, vous découvrez une stèle antique retraçant avec précision la descendance de l'empereur Grot'zuub, le célèbre fondateur de la dynastie des Grot'zuubiens. La stèle ne mentionne pas les noms des conjoints des personnages nommés, et vous savez que les descendants de l'empereur Grot'zuub ne se sont jamais mariés entre eux. Le récit de la stèle s'arrête à l'effondrement de la dynastie.

Vous décidez de représenter l'arbre généalogique de l'empereur Grot'zuub par un tableau : chacun des n personnages nommés dans la stèle se voit désigné par un numéro entre 0 et $n-1$, l'empereur Grot'zuub ayant le numéro 0. On définit un tableau `parent` de longueur n ainsi : pour tout `i` $\in [0, n-1]$, `parent[i]` est le numéro du père ou de la mère dans l'arbre du personnage de numéro `i`. Par convention `parent[0]` vaut -1 (on ne connaît pas les parents de Grot'zzub). On rappelle que dans cet arbre généalogique, il n'y a aucun conjoint, seulement les descendants de Grot'zuub.

Exemple : supposons que la stèle mentionne 9 personnages. Supposons les données suivantes :

- Grot'zuub (le numéro 0) a eu deux enfants de numéros respectifs 3 et 6.
- 6 a eu trois enfants de numéros 8, 7 et 1.
- 8 a eu deux enfants de numéros 4 et 2.
- 7 a eu un enfant de numéro 5.
- Il n'y a pas d'autres enfants.

personne	0	1	2	3	4	5	6	7	8
parent	-1	6	8	0	8	7	0	6	6



Cet arbre généalogique est donc codé par le tableau : `parent=[-1,6,8,0,8,7,0,6,6]`

Attention : l'exemple donné ci–dessus sert juste à comprendre ce que l'on fait, mais vos programmes doivent être capables de fonctionner même avec d'autres exemples de dynastie (donc avec des tableaux `parent` différentes).

Dans la suite, le terme *enfant* désigne un enfant direct, c'est–à–dire que `i` est un enfant de `j` si et seulement si `j` est le père ou la mère de `i`.

On définit récursivement la notion de *descendant* :

- tout personnage est un descendant de lui–même ;
- un personnage `j` est un descendant d'un personnage `i` s'il existe un enfant `k` de `i` dont `j` est un descendant.

Dans la suite on suppose que deux variables `i` et `j` on déjà été définies, avec pour chacune une valeur entière comprise entre 0 et $n-1$.

24.1. Écrire une fonction `premiere_generation (p, i)` dont le paramètre `p` doit être la liste `parent`, et qui calcule le booléen `Vrai` si `i` est un enfant (direct) de Grot'zuub, et le booléen `Faux` sinon.

Sur l'exemple de famille donné, l'appel `premiere_generation (parent, 6)` doit renvoyer `Vrai` et l'appel `premiere_generation (parent, 5)` doit renvoyer `Faux`.

24.2. Écrire une fonction qui calcule le nombre d'enfants de `i`.

Sur l'exemple de famille donné, et si la valeur de `i` est 6, on doit trouver 3 enfants.

Lorsque Grot'zuub prit le pouvoir, un sorcier puissant lança une terrible malédiction : un descendant sur deux de Grot'zuub devait être frappé par la malédiction. Vous avez pris soin d'assigner un numéro pair aux descendants maudits (Grot'zuub lui–même était maudit).

Selon les termes de la malédiction, tous les enfants d'une personne maudite avaient les cheveux roses.

Sur l'exemple donné, les personnes maudites avaient les numéros 0, 2, 4, 6 et 8 ; et les personnes ayant les cheveux roses avaient les numéros 1, 2, 3, 4, 6, 7 et 8 (les deux seuls n'ayant pas de cheveux roses sont ceux de numéro 0 et 5, ce sont les seuls n'ayant pas un parent maudit).

24.3. Écrire un programme permettant de calculer le nombre de personnes avec des cheveux roses.

Intuitivement, Grot'zuub est de la génération 0, ses enfants de la génération 1, ses petits–enfants de la génération 2,etc...

Formellement on définit la fonction *generation* ainsi :

$$\begin{cases} generation(0) = 0 \text{ (la génération de Grot'zuub est 0)} \\ \text{si } generation(\mathtt{i}) = k \text{ alors la génération de chacun des enfants de } \mathtt{i} \text{ est } k+1 \end{cases}$$

Sur l'exemple de famille donné ci–dessus, $generation(7) = 2$.

24.4. Écrire une fonction `generation (p,i)` qui calcule $generation(\mathtt{i})$.

24.5. Écrire une fonction qui calcule un booléen qui précise si `j` est un descendant de `i`.

Sur l'exemple de famille donné ci–dessus :

- si `i` a la valeur 6 et `j` a la valeur 2 on doit trouver `Vrai` ;
- si `i` a la valeur 3 et `j` a la valeur 1 on doit trouver `Faux`.

24.6. Écrire une fonction qui permet de connaître le nombre de descendants de `i`. On pourra utiliser les fonctions précédentes.

Test : sur l'exemple de famille donné ci–dessus, 8 possède 3 descendants.

L'observation de l'arbre généalogique permet de constater le fait suivant :

Tous individus `i` *et* `j` *possèdent un plus proche ancêtre commun : c'est un individu* `k` *de génération maximum et dont* `i` *et* `j` *sont des descendants. Le plus proche ancêtre commun de* `i` *et de* `j` *est* `i` *si et seulement si* `j` *est un descendant de* `i`.

Sur l'exemple, le plus proche ancêtre commun de 5 et de 8 est 6.

Dans les cryptes du palais en ruine, vous retrouvez des tombes avec des momies de descendants de Grot'zuub. Malheureusement, à l'effondrement de la dynastie des Grot'zuubiens, le fondateur de la dynastie suivante fit effacer tous les noms sur les tombes : les momies retrouvées sont donc anonymes. Vous décidez malgré cela de poursuivre vos recherche archéologiques en étudiant de l'ADN prélevée sur les momies.

Pour tous individus `i` et `j` on définit le *chemin* de `i` à `j` ainsi : notons α le plus proche ancêtre commun de `i` et de `j`, notons $a_0 = \mathtt{i}$, notons par récurrence a_{k+1} le parent (père ou mère) de a_k, jusqu'à trouver un indice $p \in \mathbb{N}$ tel que $a_p = \alpha$.
Notons aussi $a'_0 = \mathtt{j}$, notons par récurrence a'_{k+1} le parent (père ou mère) de a'_k, jusqu'à trouver un indice $q \in \mathbb{N}$ tel que $a'_q = \alpha$. Alors la liste $[a_0 = i, a_1, \ldots, a_p = \alpha = a'_q, a'_{q-1}, \ldots, a'_0 = \mathtt{j}]$ s'appelle le chemin de `i` vers `j`.
Le nombre $p + q$ s'appelle la *distance génétique* entre `i` et `j`.
Avec l'arbre généalogique donné en exemple, le chemin de 4 à 1 est $[4, 8, 6, 1]$ et la distance génétique de 4 à 1 est égale à 3.

24.7. Écrire une fonction qui permet de calculer le plus proche ancêtre commun de `i` et de `j`.
Sur l'exemple de famille donné ci–dessus, le plus proche ancêtre commun de 8 et de 5 est 6 ; alors que le plus proche ancêtre commun de 2 et de 8 est 8 (toute personne est son propre ancêtre).

Indication : on pourra par exemple envisager deux cas suivant que `i` et `j` soient de la même génération ou non.

24.8. Écrire une fonction qui permet de calculer la distance génétique entre deux personnes `i` et `j`. Vérifier sur l'exemple qu'on trouve bien une distance génétique entre 4 et 1 égale à 3.

Exercice 25 Surveiller ses courtisans

Les questions suivantes n'ont pas grand chose à voir avec les questions précédentes. Pour la suite, vous pouvez oublier l'arbre généalogique. Les deux dernières questions sont d'un niveau difficile

Grot'zuub craignait les coups d'état. Pour s'en prémunir, il avait instauré un système de délation entre ses courtisans. Chaque courtisan était chargé de surveiller (et le cas échéant de dénoncer) un et un seul autre courtisan. Le système garantissait que réciproquement chaque courtisan était surveillé par un et un seul autre courtisan.

Notons `c` le nombre de courtisans, chaque courtisan est repéré par un entier compris entre 0 et $\mathtt{c} - 1$.

On dispose de la liste surveillance définie ainsi : pour tout courtisan $\mathtt{i} \in [0, \mathtt{c} - 1]$, `surveillance[i]` désigne le courtisan surveillé par `i`.

Dans les questions suivantes, nous prendrons le tableau de surveillance suivant :

courtisan surveillant	0	1	2	3	4	5	6	7	8	9	10
courtisan surveillé	3	7	4	5	1	2	8	0	10	6	9

ce qui signifie que le courtisan 0 surveille le courtisan 3, le courtisan 1 surveille le courtisan 7, etc...

Nous définissons donc `surveillance` $= [3, 7, 4, 5, 1, 2, 8, 0, 10, 6, 9]$. Bien sûr, vous devrez fournir à chaque question une solution qui fonctionne dans le cas général et pas juste sur cet exemple.

On souhaite calculer une liste `delateur` tel que pour tout $\mathtt{i} \in [0, \mathtt{c}-1]$, `delateur`$[\mathtt{i}]$ est le numéro du courtisan qui surveille `i`.

25.1. Écrire une fonction qui permet de calculer la liste `delateur` à partir de la liste `surveillance`.

Sur notre exemple, il s'agit de calculer la liste $[7, 4, 5, 0, 2, 3, 9, 1, 6, 10, 8]$.

Remarque : un programme est dit de *complexité linéaire* par rapport à `c` s'il existe une constante K telle que pour tout entier $\mathtt{c} > 0$, pour toute liste `surveillance` de longueur `c`, le nombre d'opérations qu'effectue le programme est inférieur ou égal à $K\mathtt{c}$.

Il est possible de répondre à cette question avec un programme de complexité linéaire par rapport à `c`. Si votre programme contient deux boucles imbriquées, il n'est très probablement pas de complexité linéaire : essayez alors de l'améliorer (pour améliorer la complexité d'un programme, il est souvent nécessaire de changer d'algorithme).

Un *cycle* de courtisans est une liste de courtisans $[\mathtt{i}_0, \ldots, \mathtt{i}_{\mathtt{p}-1}]$ telle que $\mathtt{i}_0$ surveille $\mathtt{i}_1$, $\mathtt{i}_1$ surveille $\mathtt{i}_2$,..., $\mathtt{i}_{\mathtt{p}-2}$ surveille $\mathtt{i}_{\mathtt{p}-1}$ et $\mathtt{i}_{\mathtt{p}-1}$ surveille $\mathtt{i}_0$.

Si un courtisan désire fomenter un coup d'état, il est obligé de convaincre tous les membres de son cycle pour éviter les dénonciations.

Dans notre exemple, il y a 2 cycles : $[0, 3, 5, 2, 4, 1, 7]$ et $[6, 8, 10, 9]$.

$$\begin{array}{cccccccccccc}
 & \nearrow & 3 & \rightarrow & 5 & & & & & 8 & & \\
0 & & & & & \searrow & & & \nearrow & & \searrow & \\
\uparrow & & & & & & 2 & 6 & & & & 10 \\
7 & & & & & \swarrow & & & \nwarrow & & \swarrow & \\
 & \nwarrow & 1 & \leftarrow & 4 & & & & & 9 & &
\end{array}$$

25.2. Écrire une fonction calculant la liste des cycles (c'est-à-dire une liste des listes). Sur notre exemple, le programme pourra calculer $[[0, 3, 5, 2, 4, 1, 7], [6, 8, 10, 9]]$ (mais d'autres réponses sont aussi acceptables car on ne précise pas dans quel ordre figurent les cycles, ni à quel courtisan commence chaque cycle).

L'esprit de Grot'zuub vous contacte par magie à travers le temps pour faire appel à vos services. Pour minimiser les risques de conspiration, il souhaite que tous ses courtisans forment un seul cycle.

25.3. Calculer la liste de toutes les listes `surveillance` possibles telles que les courtisans forment un seul cycle. Le seul paramètre connu est le nombre `c` de courtisans.

Par exemple, pour $\mathtt{c} = 4$, votre programme calculera :

$$[[3, 0, 1, 2], [3, 2, 0, 1], [2, 0, 3, 1], [2, 3, 1, 0], [1, 3, 0, 2], [1, 2, 3, 0]]$$

(ou une permutation de ces 6 listes de surveillance).

4 Dictionnaires

Exercice 26

Pour cet exercice, vous pouvez récupérer le fichier `TP04.py` *(sur Moodle) qui contient des déclarations de dictionnaires sur lesquelles vous pourrez effectuer vos tests.*

Les noms des étudiants d'une école sont stockés dans un dictionnaire, avec comme clé leur numéro d'étudiants.

Exemple :

```
{ 208978 : "AGHALI", 206780 : "AGUILLLON", 207525 : "BAUDOIN",  ... }
```

26.1. Écrivez une fonction `get_nom` prenant en paramètre un dictionnaire `dict` et un entier `num` représentant un numéro d'étudiant et retournant le nom qui lui est associé.

Avec l'exemple fourni, l'appel de `get_nom(dict_etudiants1, 204909)` *doit retourner* `"BOUVIER"`.

26.2. Écrivez une fonction `get_numero` prenant en paramètre un dictionnaire `dict` et une chaîne de caractères `nom` représentant un nom d'étudiant et retournant le numéro qui lui est associé.

Avec l'exemple fourni, l'appel de `get_numero(dict_etudiants1, "PERRIN")` *doit retourner* `219843`.

26.3. Écrivez une fonction prenant en paramètre un dictionnaire `dict` et retournant la liste des noms dont le numéro associé est un multiple de 10.

Avec l'exemple fourni, l'appel de votre fonction sur `dict_etudiants1` *doit retourner* `['AGUILLLON', 'CHAUVEL', 'GULLIENT', 'CATLEY', 'HOUCKE', 'JOSSO', 'LE SCORNET', 'LUCE', 'PORZIER']`.

26.4. Écrivez une fonction prenant en paramètre un dictionnaire `dict` et retournant la liste des numéros des étudiants dont le nom commence par "B".

Avec l'exemple fourni, l'appel de votre fonction sur `dict_etudiants1` *doit retourner* `[207525, 216928, 194001, 191632, 204909, 191772, 196648]`.

Exercice 27

27.1. Écrivez une fonction `occurrences_lettre` prenant un paramètre une chaîne de caractère `texte` et retournant un dictionnaire dont les clés sont les caractères présents dans `texte` et les valeurs sont le nombre d'occurrence de ces caractères dans le texte.

L'appel à `occurrences_lettre("Le chat chasse !")` *doit retourner :*

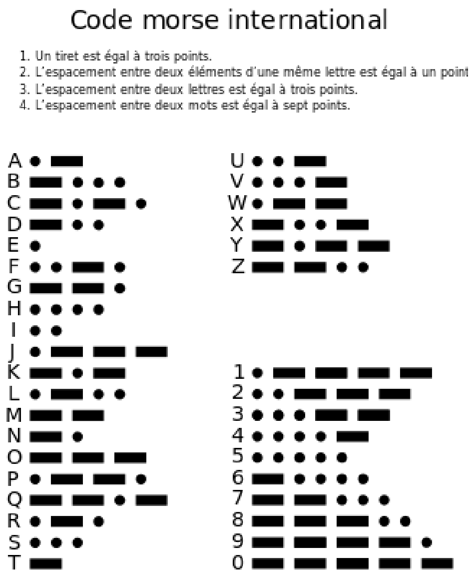
```
{'L': 1, 'e': 2, ' ': 3, 'c': 2, 'h': 2, 'a': 2, 't': 1, 's': 2, '!': 1}
```

27.2. Écrivez une fonction prenant en paramètre un une chaîne de caractère `texte` et affichant les fréquences d'apparition de chaque lettre.

Exercice 28 Code Morse

Le code Morse est un système de transcription permettant de coder des messages textuels en série d'impulsions (courte pour les "points", longues pour les "tirets").
Dans cet exercice, nous allons coder un codeur et un décodeur de message écrit en Morse, avec les conventions suivantes :

- Les messages en clair ne contiendront que des lettres non accentuées en majuscule ainsi que des espaces.
- Le codage des "points" sera le caractère "`.`"
- Le codage des "tirets" sera le caractère "`-`"
- Dans le texte codé, le passage au codage de la lettre suivante du mot sera représenté par un espace.
- Dans le texte codé, le passage au codage du mot suivant sera représenté par deux espaces.



Exemple : Le message "`LE MOT`" sera codé par : `.-..␣.␣␣--␣---␣`

Le dictionnaire de transcription lettre claire vers caractère codé vous est fourni dans le fichier `TP04.py`.

28.1. Écrivez une fonction `traduire_texte_vers_morse` prenant en paramètre une chaîne de caractère en clair `texte` et un dictionnaire de conversion `dict`, et retournant le texte codé suivant la méthode ci-dessus.

L'appel de `traduire_texte_vers_morse("LE CHAT EST NOIR", dict_morse)` doit retourner :

`.-..␣.␣␣-.-.␣....␣.-␣-␣␣.␣...␣-␣␣-.␣---␣..␣.-.`

28.2. Écrivez une fonction `reverse_dict` prenant en paramètre un dictionnaire et retournant le dictionnaire dont les clés et les valeurs ont été inversées.

L'appel de `reverse_dict(dict_morse)` devra vous retourner le dictionnaire :

`{'.-': 'A', '-...': 'B', '-.-.': 'C', ... }`

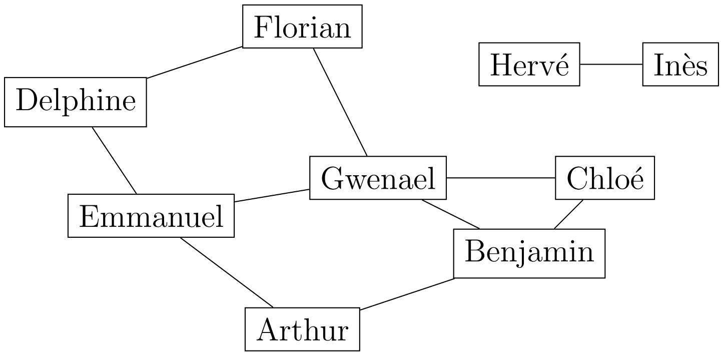
Est-ce que cette fonction est applicable sur tout dictionnaire ?

28.3. Écrivez une fonction `traduire_morse_vers_texte` prenant en paramètre une chaîne de caractère représentant le texte codé `texte_code` et le dictionnaire utilisé pour la conversion `dict`, et retournant le texte décodé.

L'appel de `traduire_morse_vers_texte(".-.. . -.-. .... .- - . ... - -. --- .. .-. ", dict_morse)` doit retourner : "LE CHAT EST NOIR"

Exercice 29

On représente les liens d'un réseau social à l'aide d'un dictionnaire.
Exemple : le réseau d'amis suivant :



est représenté par :

```
reseau = {
    "Arthur" : ["Benjamin", "Emmanuel"],
    "Benjamin" : ["Arthur", "Gwenael", "Chloé"],
    "Chloé" : ["Benjamin", "Gwenael"],
    "Delphine" : ["Emmanuel", "Florian"],
    "Emmanuel" : ["Arthur", "Gwenael", "Delphine"],
    "Florian" : ["Delphine", "Gwenael"],
    "Gwenael" : ["Florian", "Benjamin", "Emmanuel", "Chloé"],
    "Hervé" : ["Inès"],
    "Inès" : ["Hervé"]
}
```

On considérera que la relation d'amitié est symétrique : si a est dans la liste des amis de b, alors b est dans la liste des amis de a.

29.1. Écrivez une fonction prenant en paramètre un dictionnaire `reseau` représentant un réseau social et retournant la personne ayant le plus grand nombre d'amis.

Sur l'exemple précédent, votre fonction doit retourner `Gwenael`.

29.2. Écrivez une fonction `amis_en_commun` prenant en paramètre un dictionnaire `reseau` et deux noms de personne du réseau, et retournant la liste des amis commun à ces deux personnes.

Sur l'exemple précédent, l'appel de `amis_en_commun(reseau, "Florian", "Emmanuel")` doit retourner `['Delphine', 'Gwenael']`.

On cherche à déterminer tous les amis de nos amis (et ceux des amis de nos amis, etc.). Pour ce faire, on propose d'implémenter l'algorithme suivant (le pseudo-code autorise l'utilisation de dictionnaires et de tableaux à longueur variable) :

```
amis_des_amis(reseau : dictionnaire, nom : chaîne
  Données :
      res : chaîne[]
      marqués : dictionnaire[chaîne :booléen]
      aTraiter : chaîne[]
      n : chaîne
  début
    res ← chaîne[0]
    marqué ← {}
    aTraiter ← reseau[nom]
    pour n clé de reseau faire
      marqué[n] ← Faux
    fin
    marqué[nom] ← Vrai
    tant que aTraiter ! = [] faire
      n ← enlever un élément de aTraiter
      si non marqué[n] alors
        ajouter n à res
        marqué[n] ← Vrai
        ajouter reseau[n] à aTraiter
      fin
    fin
    retourner res
  fin
```

29.3. Implémentez cet algorithme.

Sur l'exemple précédent, l'appel de `amis_des_amis(reseau, "Arthur")` *doit retourner* `['Benjamin', 'Emmanuel', 'Gwenael', 'Chloé', 'Delphine', 'Florian']`.

29.4. *(Difficile)* En vous inspirant de l'algorithme précédent, proposer une fonction permettant de calculer la distance entre deux amis (c'est à dire le nombre minimal de lien d'amitié qui les unit).

Sur l'exemple précédent, l'appel de votre fonction sur `"Arthur"` *et* `"Florian"` *doit retourner 3, et celui sur* `Gwenael` *et* `Inès` *doit renvoyer* `inf`.

En `python`*, vous pouvez affecter à une variable* `float("inf")`*, qui sera supérieur à tous les autres nombres auquel vous le comparerez.*

5 Utilisation de modules sous python

Exercice 30

Dans sa version de base, `python` possède un certains nombre de fonctions prédéfinies :

- `min`,
- `max`,
- `abs`,
- `sort`,
- ...

Mais certaines fonctions ne sont pas implémentées. Par exemple, il n'existe pas de commande dans la version de base de `python` pour calculer directement $\sqrt{x}$.

En revanche, un grand nombre de fonctions sont implémentées dans des bibliothèques appelées **modules**. Un module est un fichier écrit en `python` que l'on va **importer** depuis notre programme en écrivant :

```
import nom_module
...
```

Par exemple, pour utiliser le module `math`, on utilise les commandes :

```
import math

print(math.sqrt(9))
```

Pour utiliser la fonction `sqrt` du module `math`, il y a 3 possibilités :

- importer le module `math` puis utiliser la commande `math.sqrt`
- importer le module `math` en lui donnant un alias :
  ```
  import math as mt
  ```
  puis utiliser la commande `mt.sqrt`. Ceci permet d'éviter des noms de module trop longs.
- importer directement certaines ou toutes les fonctions du module `math` :
  ```
  >>> from math import *
  ```
  puis les utiliser directement sans préfixe : `sqrt`. ATTENTION !! : ceci est risqué si vous importez plusieurs modules avec des fonctions portant le même nom.

30.1. En utilisant le module `math`, calculer les valeurs suivantes :

- $\sqrt{5}$
- e^3
- $cos(4)$
- $tan(\pi/4)$
- $ln(10)$

Parmi les modules `python` les plus utilisés, on trouve :

- `math` : pour les opérations mathématiques classiques
- `numpy` : pour le calcul matriciel et numérique
- `scipy` : pour le calcul symbolique
- `matplotlib` : pour l'affichage (de courbes, images, dessin pixelisés, etc.)
- `random` : pour la génération de nombres aléatoires
- `os` et `sys` : pour l'utilisation de commandes liées au système d'exploitation

- `time` ou `datetime` : pour mesurer la durée d'exécution d'un programme
- `socket` : pour l'utilisation du réseau
- ...

Exercice 31 Utilisation de `numpy`

`numpy` est une bibliothèque `python` utilisée notamment pour le calcul matriciel et le calcul numérique.
Elle s'utilise en l'important au début de votre fichier par la commande :

```
import numpy
```

ou plus généralement :

```
import numpy as np
```

31.1. Déterminer ce qu'effectuent les commandes suivantes :

- `np.zeros(7)`
- `np.ones(6)`
- `np.identity(3)`
- `np.array([3,7,-1,2])`
- `np.array([[3,7],[-1,2]])`
- `np.arange(10,30,5)`
- `np.linspace(0,2,9)`
- `np.sin(np.linspace(0,2*np.pi,20))`
- `math.sin(np.linspace(0,2*np.pi,20))`

On définit deux matrices :

- `a = np.array([[1,3],[0,4]])`
- `b = np.array([[4,0],[-1,1]])`

31.2. Déterminer ce qu'effectuent les commandes suivantes :

- `a+b`
- `a+4`
- `a*b`
- `3*a`
- `a*3`
- `np.add(a,b)`
- `a.dot(b)`
- `a @ b`
- `a.transpose()`
- `np.linalg.matrix_power(a,2)`
- `a.shape`

31.3. Toujours avec les matrices `a` et `b` :

- `a.sum()`
- `a.sum(axis=0)`
- `a.sum(axis=1)`
- `a.min()`
- `a.max()`
- `a[1]`
- `a[0,1]`
- `a[0][1]`

31.4. Déterminer une suite de commandes permettant de calculer un tableau `numpy` contenant les nombres $[f(0), f(1), \ldots, f(10)]$ où f est définie par :

$$f(x) = x^2 \sin(x) + 4$$

Notez que `numpy` est une librairie très utilisée et très efficace en calcul numérique. Elle permet de faire beaucoup d'autres choses, mais dont nous ne parlerons pas ici.

Exercice 32 Utilisation de `matplotlib`

`python` propose un module, `matplotlib`, permettant d'afficher simplement des graphiques.

Pour afficher des fonctions mathématiques, on peut utiliser la commande `plot` présente dans `matplotlib.pyplot` Dans (quasiment) tous les programmes `python` dans lesquels nous ferons de l'affichage graphique, nous utiliserons :

- `numpy` que l'on importera par la commande :

```python
import numpy as np
```

- `matplotlib.pyplot` que l'on importera par la commande :

```python
import matplotlib.pyplot as plt
```

Premier exemple On souhaite afficher le graphique de la fonction

$$f : x \mapsto 3x^4 + 8x^3 - 6x$$

sur l'intervalle $[-3, 2]$:

a. on commence par discrétiser l'intervalle $[-3, 2]$:

```python
x = np.linspace(-3,2,100)
```

b. on calcule les ordonnées correspondantes :

```python
y = 3*x**4+8*x**3-6*x
```

Notez que l'on utilise la fonction puissance et la fonction multiplication entre tableaux `numpy`, ce qui ne fonctionnerait pas avec des tableaux classiques.

c. on calcule l'image qui va être affichée :

```python
plt.plot(x,y,'b:',linewidth=5)
```

d. enfin, on affiche l'image :

```python
plt.show()
```

Remarques :

- Pour afficher un graphique, `matplotlib` utilise 2 tableaux :
  — un tableau d'abscisses : ici représenté par `x`
  — un tableau d'ordonnées : ici représenté par `y`

- Rappel : la commande `linspace(a,b,n)` du package `numpy` permet de placer `n` valeurs régulièrement réparties dans l'intervalle `[a,b]`.
- Rappel : les tableaux de base de python ne permettent pas de faire des opérations terme à terme :

```
[1,2,3] * 2 -> [1,2,3,1,2,3]
```

  On utilise donc des tableaux `numpy`, qui se manipulent facilement.
  Attention cependant, lorsque vous devez utiliser des fonctions comme cos, ln, exp, ... à utiliser les fonctions correspondantes dans `numpy` : `np.cos`, `np.log`, `np.exp`
- l'affichage se fait en deux temps :
  1. calcul de l'image à afficher (avec une ou plusieurs commandes `plot`)
  2. affichage de l'image à l'écran avec la commande `show`
- la commande `plot` peut prendre plusieurs arguments [1]. En considérant que `x=[x0, x1, ... xn]` et `y=[y0, y1, ... yn]` sont deux tableaux (`numpy` ou non) de même taille, les principales utilisations sont :
  — `plot(x,y)` : affiche un ensemble de points `(x0,y0), (x1,y1), ... (xn,yn)` avec le style de ligne par défaut.
  — `plot(y)` : affiche un ensemble de points `(0,y0), (1,y1), ... (n-1,yn)` avec le style de ligne par défaut.
- plusieurs options sont disponibles :
  — `color`, qui correspond à la couleur de la ligne et des marqueurs, qui peut valoir `'blue'` (ou `'b'`), `'red'` (ou `'r'`), `'green'` (...), `'orange'`, `'yellow'`, ... et aussi d'autres représentations (en hex RGB par exemple : `'#f6a154'`)
  — `marker`, pour mettre en évidence les points dessinés, qui peut valoir `'o'` (cercles), `'x'` (croix en 'x'), `'+'` (croix en '+'), `'s'` (carrés), `'v'` (triangles vers le bas), `'^'` (triangles vers le haut) ...
  — `linestyle`, qui correspond au style de ligne, qui peut valoir `'solid'` ou `'-'`, `'dashed'` ou `'--'`, `'dashdot'` ou `'-.'`, `'dotted'` ou `':'`
  — `linewidth`, qui correspond à l'épaisseur de la ligne,
  — `markersize`, qui correspond à la taille des marqueurs
  — ainsi on pourra par exemple exécuter la commande :

```
plot(x, y, color = 'r', marker = '+', linestyle = ':',
        linewidth = 4, markersize = 20)
```

- certaines options peuvent être utilisées en raccourci : si l'on souhaite une ligne en pointillés, rouge avec des marqueurs en `'x'`, on peut écrire : `plot(x,y,'rx:')`

32.1. Affichez, avec le style de votre choix, et sur **un seul** graphique, les courbes représentant les fonctions suivantes sur l'intervalle $[-5, 5]$:

- $f(x) = x + \sin(x)$
- $g(x) = \dfrac{x^3 + 5}{x^2 + 2}$
- $h(x) = \ln(x^4 + 1) - 4$

1. voir le lien https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.plot.html pour une description exhaustive

32.2. Dans notre cas, étant donné que nous souhaitons afficher des fonctions, il est souvent utile d'ajouter des informations au graphique :

- la commande `plt.grid()` permet l'affichage d'une grille adaptée en fonction de la taille de la fenêtre, on peut lui passer en argument une couleur, un style de ligne et une épaisseur de ligne :

```
plt.grid(color='grey',linestyle = '--')
```

- lorsque plusieurs fonctions sont dessinées, on peut dessiner une petite légende avec le nom de chaque fonction :
  - récupérez la valeur de retour du `plot` dans une variable :

    ```
    c_f, = plt.plot(x,f,...)
    ```

    On ne récupère pas toute la valeur de retour de `plot`*, seulement la première partie, d'où la ',' après le* `c_f`*.*
  - ajoutez une étiquette à la courbe (ici `'f'`) :

    ```
    c_f.set_label('f')
    ```

  - affichez la légende :

    ```
    plt.legend()
    ```

Dans certains cas, il peut être intéressant d'afficher les fonctions les unes à côté des autres. Pour cela, on peut utiliser la commande `subplot`.
Avant le `plot` de chacune des fonctions, on fait appel à une commande `subplot`. `subplot` permet de diviser la fenêtre d'affichage en une 'grille' de cases dans lesquelles on dessinera les fonctions. Par exemple, la commande :

```
plt.subplot(231)
```

signifie qu'on divise la fenêtre d'affichage en une grille de 2 lignes par 3 colonnes et que le `plot` qui suivra sera dessiné dans la case n°1.

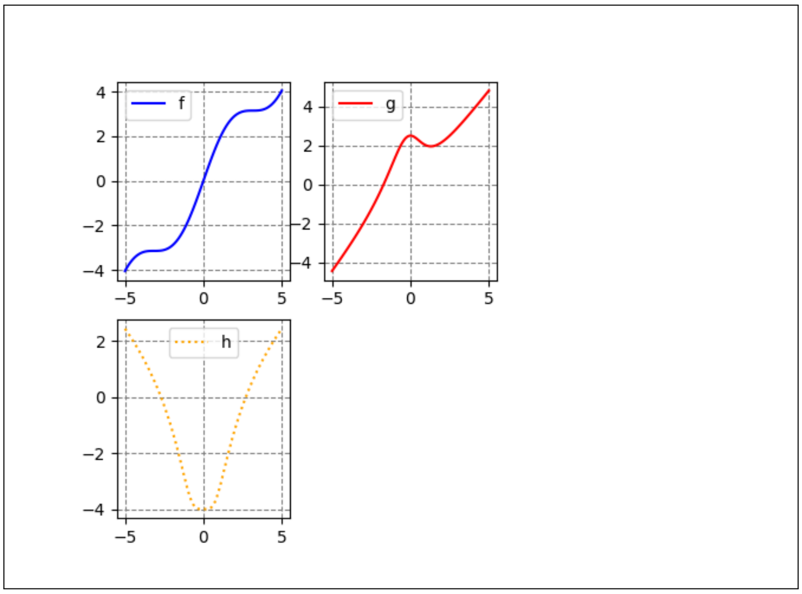
1	2	3
4	5	6

Ordre du `subplot`

Ainsi si l'on souhaite prendre une grille de 2 par 3, et afficher les fonctions f, g et h respectivement dans les cases 1, 2 et 4, on écrira :

```
plt.subplot(231)
plt.plot(x,f,...)
plt.subplot(232)
plt.plot(x,g,...)
plt.subplot(234)
plt.plot(x,h,...)
```

Et on obtient un graphique ressemblant à :



Exercice 33 Création et utilisation de modules personnels

On peut créer ses propres modules `python`.
Supposons que l'on souhaite coder une fonction `fonction1` qui sera utile dans plusieurs projets. Pour éviter de recopier son code à plusieurs endroits, on peut la coder dans un fichier, appelé par exemple `mon_module.py` et importer ce module depuis les autres fichiers en écrivant :

```
import mon_module

...
```

33.1. Dans un fichier `bibliotheque_tableaux.py`, écrivez une fonction `recherche_maximum` prenant un paramètre un tableau (qu'on supposera non-vide) et retournant le maximum des éléments de ce tableau.

33.2. Dans un fichier `mon_programme.py`, définissez un tableau `tab` contenant des entiers (choisissez les différents les uns des autres), puis déterminez-en le maximum grâce à la fonction écrite précédemment.

Exercice 34 Documentation du code

Lorsque les projets de programmation durent dans le temps, sont développés par plusieurs personnes, et/ou sont partagés, il est important qu'ils soient **documentés**, c'est à dire que l'on puisse avoir une description des fonctions que l'on utilise, avec par exemple :

- le nombre et le type des arguments
- le type de retour
- l'action effectuée par la fonction
- les restrictions sur les paramètres (tableau non-vide, entier positif, etc.)
- ...

`python` propose une manière de documenter les fonctions qui permet :

- d'accéder à la documentation d'une fonction depuis la console `python`,
- de générer un document HTML contenant la documentation de tout un module (en utilisant la commande `pydoc3`).

34.1. Dans une console `python` (en tapant `python3` dans un terminal), afficher la documentation de la fonction `min` en tapant

```
>>> help(min)
```

Quittez en appuyant sur `q`.

et en tapant

```
>>> print(min.__doc__)
```

On peut commenter les fonctions `python` de telle manière qu'elles soient prises en compte dans la documentation. Pour ce faire, on utilise des **docstring** selon le modèle suivant :

```python
def fonction(arg1, arg2) :
    """Fonction faisant ... - description de la fonction,
    de ce qu'elle affiche/renvoie, etc.

    Arguments :
    - arg1 : ...
    - arg2 : ...
    """
    code de la fonction
```

Il n'y a pas de syntaxe fixée (comme il peut exister dans d'autres langages comme java) sur la façon de commenter des fonctions, seulement des recommandations (voir [2]).

34.2. Reprenez le code de votre fonction `recherche_maximum` du fichier `bibliotheque_tableaux.py` (exercice précédent) et commenter le.

Une fois les *docstring* écrites, on peut produire une documentation de notre module en tapant la commande :

```
pydoc3 -w nom_module(sans .py)
```

qui produira le fichier `nom_module.html` contenant les commentaires des fonctions mis en forme.

34.3. Générez la documentation de votre module `bibliotheque_tableaux.py` et ouvrez le fichier produit.

Exercice 35 `doctest`

Il est possible de spécifier des tests dans les *docstring* des fonctions, et d'exécuter ces tests à l'aide d'une commande. Pour ce faire, on peut utiliser un module appelé **doctest**.

Notez que l'écriture des tests dans les commentaires de la fonction ne nécessite pas l'import du module, seulement l'exécution de ces tests.

Par exemple, si l'on souhaite tester le code de la fonction factorielle, on peut écrire dans les commentaires :

2. Pour une documentation complète, voir https://www.python.org/dev/peps/pep-0257/

```
def factorielle(n):
    """Renvoie la factorielle de n.

    >>> factorielle(3)
    6
    >>> factorielle(6)
    720
    """
    ... (code de la fonction)
```

et pour exécuter ces tests, écrire dans le même fichier :

```
import doctest

doctest.testmod(verbose=True)

# testmod() pour 'test module'
```

Remarque : l'argument **`verbose=True`** *est optionnel, mais s'il n'est pas présent, l'exécution du programme n'affiche que les tests échoués, et n'affiche rien si tous les tests sont réussis.*

35.1. Reprenez le code de votre fonction `recherche_maximum` du fichier `bibliotheque_tableaux.py` (exercices précédents) et ajoutez-y des tests. Testez ensuite votre programme avec `doctest`.

35.2. Exécutez ensuite le code de votre fichier `mon_programme.py`, que remarquez-vous ?

Il est possible de spécifier que la commande

`doctest.testmod(verbose=True)`

ne s'exécute que si le fichier courant est exécuté (et ne s'exécute donc pas lorsqu'il est importé).

Pour cela, on remplacera la commande par :

```
if __name__=='__main__' :
    doctest.testmod(verbose=True)
```

35.3. Modifier votre fichier `bibliotheque_tableaux.py` et vérifier que les tests ne sont relancés lors de l'exécution de `mon_programme.py`.

Dans la suite du module, vous devrez écrire les tests de vos TPs dans les *dosctring* des fonctions correspondantes.

Pour davantage de documentation sur **`doctest`**, *voir https://docs.python.org/3.4/library/doctest.html*

6 Récursivité

N'oubliez pas d'écrire les tests de vos fonctions à l'aide de *doctest.*

Exercice 36 Dessins avec des étoiles

36.1. Écrire la fonction suivante :

```
def triangle_haut (n) :
    if n > 0 :
        triangle_haut (n-1)
        print (n * "*")
```

Essayer de comprendre ce que fait cette fonction avant de la tester, puis tester `triangle_haut (5)` pour vérifier.

36.2. Écrire une fonction récursive `triangle_bas` prenant en paramètre un entier positif `n` et qui dessine le même triangle que `triangle_haut`, mais avec une pointe vers le bas.

Test : `triangle_bas (5)` provoque l'affichage

```
*****
****
***
**
*
```

36.3. Écrire une fonction récursive `sablier` prenant en paramètre un entier positif `n` et qui dessine en même temps les triangles haut et bas.

Test : `sablier (4)` provoque l'affichage

```
****
***
**
*
**
***
****
```

Le savant Sunisoc propose la fonction suivante :

```
def triangle_milieu (n) :
    if n > 0 :
        triangle_milieu (n-1)
        print (n * "*")
        triangle_milieu (n-1)
```

36.4. Quel est l'affichage provoqué par l'appel de `triangle_milieu (4)` ? Essayer de deviner le résultat, puis tester !

Test : `triangle_milieu (10)`, `triangle_milieu (13)`.

Remarque (explosion exponentielle) : derrière son apparence élémentaire, cette fonction a une complexité exponentielle qui la rend inutilisable, même pour de petites valeurs.

Exercice 37 Exponentiation rapide

37.1. Écrivez une fonction **itérative** `expo_normale(x,n)` prenant en paramètre une valeur `x` et un entier `n` (qu'on supposera positif) et retournant la valeur de `x`$^{\mathtt{n}}$ (en n'utilisant que la multiplication `*`).

On peut améliorer la complexité d'une telle fonction en utilisant l'algorithme suivant, pour calculer x^n :

- si $n = 0$ alors retourner 1
- sinon si $n\%2 = 0$, alors calculer $a = x^{\frac{n}{2}}$ et retourner $a * a$
- sinon calculer $a = x^{\frac{n-1}{2}}$ et retourner $x * a * a$.

37.2. Écrivez une fonction **récursive** `expo_rapide(x,n)` prenant en paramètre une valeur `x` et un entier `n` (qu'on supposera positif) et calculant la valeur de `x`$^{\mathtt{n}}$ selon l'algorithme précédent.

37.3. On va à présent tester la durée d'exécution de nos fonctions en utilisant le module `time`.

a. Ajoutez la commande

```
import time
```

au début de votre fichier.

b. Définissez une variable `n = 10000`

c. Utilisez les commandes :

```
t1 = time.time()
# calcul dont on souhaite mesurer la durée
t2 = time.time()
print("durée :",t2-t1)
```

pour mesurer le temps de calcul de :

- `expo_normale(2,n)`
- `expo_rapide(2,n)`
- `2**n`

d. faites varier `n` et observez la différence de temps de calcul entre les méthodes.

Exercice 38 Recherche dichotomique

38.1. Écrivez une fonction `recherche(tab,el)` prenant en paramètre un tableau `tab` et une valeur `el` et retournant `True` si `el` appartient à `tab`.

On considère à présent que le tableau dans lequel on cherchera l'élément est trié.

38.2. En vous inspirant de ce que vous avez vu en TD, écrivez :

- une fonction récursive `recherche_trie_aux(tab,el,i,j)` qui retourne `True` si l'élément `el` se trouve entre les bornes `i` (inclus) et `j` (exclus) de `tab`.

- une fonction `recherche_trie(tab,el)` utilisant la fonction précédente et retournant `True` si `el` appartient à `tab`.

  38.3. Comparez les temps d'exécutions des deux fonctions.
  La commande `list(range(n))` permet de créer le tableau `[0,1,...,n-1]`

Exercice 39

39.1. Écrivez une fonction récursive `somme_cube_chiffre(n)` prenant en paramètre un nombre entier `n` et retournant la somme des cubes de ses chiffres.
Exemple : `somme_cube_chiffre(643)` retourne $6^3 + 4^3 + 3^3 = 216 + 64 + 27 = 307$

On cherche à déterminer un *point fixe* de cette fonction, c'est à dire un entier `n` tel que `somme_cube_chiffre(n)=n`.
39.2. Écrivez une fonction récursive `iterer_somme_cube_chiffre(n)` qui :

1. calcule `a=somme_cube_chiffre(n)`
2. affiche `a`
3. si `a` $\neq$ `n`, appelle `iterer_somme_cube_chiffre(a)`

Exemple : `iterer_somme_cube_chiffre(2)` doit afficher :

```
8
512
134
92
737
713
371
371
```

39.3. Testez votre fonction pour des multiples de 3 ($\neq 0$). Que remarquez-vous ?
39.4. Conjecturez le résultat de la fonction `iterer_somme_cube_chiffre(n)` lorsque `n` est un multiple de 3.

Nous allons maintenant montrer cette conjecture en utilisant l'ordinateur.
39.5.

a. *(difficile)* Montrez que lorsque `n`$\geqslant 10\,000$, `somme_cube_chiffre(n)<n`.

b. En utilisant un programme informatique, déterminez s'il existe des nombres multiples de 3 ($\neq 0$) et $\leqslant 9999$ ne vérifiant pas votre conjecture.
(Vous pouvez modifier votre fonction `iterer_somme_cube_chiffre` de telle sorte qu'elle renvoie le nombre sur lequel elle s'arrête, et qu'elle évite les affichages.)

c. *(difficile)* Conclure.

Exercice 40 Suite de Syracuse

La suite de Syracuse d'un nombre entier $N > 0$ est définie par récurrence de la façon suivante :

$$\begin{cases} u_0 & = N \\ u_{n+1} & = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3 * u_n + 1 & \text{si } u_n \text{ est impair} \end{cases} \end{cases}$$

40.1. Définissez en `python` une fonction `prochain_terme` prenant en paramètre un entier `u` et retournant :
- $\frac{u}{2}$ si u est pair.
- $3 \times u + 1$ si u est impair.

40.2. Définissez en `python` une fonction récursive `syracuse` prenant en paramètre deux entiers `u0` et `n` et retournant les `n+1` premiers termes (de u_0 à u_n) de la suite de Syracuse avec `u0` comme premier terme.

Exemple : `syracuse(25,10)` doit retourner :

```
[25, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34]
```

40.3. Testez votre fonction pour différentes valeur de `u0` (avec un `n` suffisamment grand). Que remarquez-vous ?

La conjecture de Syracuse est la suivante : « quel que soit le nombre N de départ, la suite de Syracuse atteindra 1 ».

Étant donné un nombre de départ `u0`, on appelle *durée de vol* le plus petit entier n pour lequel la suite de Syracuse atteint 1.

40.4. Définissez en `python` une fonction récursive `duree_de_vol` prenant en paramètre un entier `u0` et retournant la durée de vol correspondante.

Exemple : `duree_de_vol(25)` doit retourner 23.

40.5. Définissez en `python` une fonction `liste_durees_de_vol` prenant en paramètre un entier `p` et retournant la liste des durées de vol pour `u0` de 1 à `p`.

Exemple : `liste_durees_de_vol(11)` doit retourner :

```
[0, 1, 7, 2, 5, 8, 16, 3, 19, 6, 14]
```

40.6. (Bonus) On souhaite représenter graphiquement les durées de vol pour les p premiers entier.

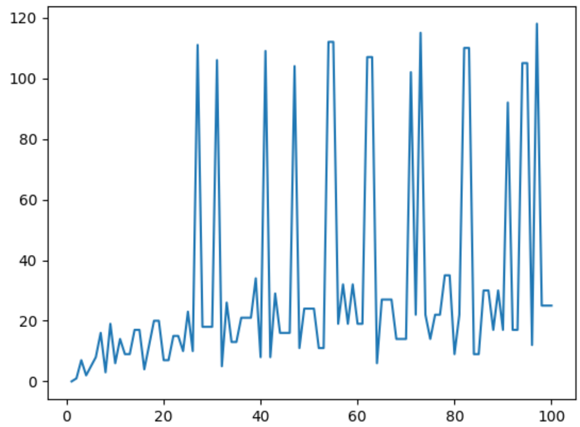
a. Ajoutez la commande `import matplotlib.pyplot as plt` au début de votre fichier.

Pour dessiner des points en `python`, vous pouvez utiliser la commande `plt.plot(Lx,Ly)` où :
- `Lx` représente la liste des abscisses des points à afficher
- `Ly` représente la liste des ordonnées des points à afficher

puis utiliser la commande `plt.show()` pour afficher le graphique.

b. Affichez les durées de vols des entiers de 1 à 100.

Vous devez obtenir un graphique ressemblant à :



Exercice 41 Nombres miroirs

Dans cet exercice, on s'intéresse aux *miroirs* de nombres entiers (par exemple, le miroir de 452 est 254).

41.1. Définissez en `python` une fonction `miroir` prenant en paramètre un entier `n` et retournant le nombre miroir de `n`.

(Il est possible de coder cette fonction en utilisant une sous-fonction récursive, mais la version itérative est plus simple.)

Exemples :

- `miroir(9)` *doit retourner 9*
- `miroir(0)` *doit retourner 0*
- `miroir(10)` *doit retourner 1*
- `miroir(1204)` *doit retourner 4021*
- `miroir(2100)` *doit retourner 12*

Un palindrome est nombre égal à son miroir (exemple : 121, 1001, 8, ...).

41.2. Écrivez une fonction `est_palindrome(n)` qui retourne `True` si l'entier `n` en paramètre est un palindrome.

Soit $N \in \mathbb{N}$, on définit la suite de Lychrel de la manière suivante :

$$\begin{cases} u_0 & = N \\ u_{n+1} & = \begin{cases} u_n & \text{si } u_n \text{ est un palindrome} \\ u_n + miroir(u_n) & \text{sinon} \end{cases} \end{cases}$$

41.3. Définissez en `python` une fonction `prochain_terme` prenant en paramètre un entier `u` et retournant :

- u si u est un palindrome.
- $u + miroir(u)$ sinon.

41.4. Définissez en `python` une fonction récursive `lychrel` prenant en paramètre deux entiers `u0` et `n` et retournant les `n+1` premiers termes (de u_0 à u_n) de la suite de Lychrel avec `u0` comme premier terme.

Exemple : `lychrel(97,8)` *doit retourner :*

```
[97, 176, 847, 1595, 7546, 14003, 44044, 44044, 44044]
```

41.5. Testez votre fonction pour différentes valeur de `u0` (avec un `n` suffisamment grand). Que remarquez-vous ?

41.6. Testez votre fonction avec `u0`= 196. Que remarquez-vous ?

41.7. Définissez en `python` une fonction `nb_iteration_lychrel` prenant en paramètre deux entiers `u0` et `n` et retournant le nombre d'itérations nécessaires pour que la suite `lychrel` devienne stationnaire, la fonction retourne -1 s'il y a plus de `n` itérations.

41.8. Définissez en `python` une fonction `liste_nb_iteration_lychrel` prenant en paramètre un entier `p` et retournant la liste des nombres d'itérations nécessaires pour que la suite `lychrel` devienne stationnaire, pour `u0` de 1 à `p` (on fixera le nombre d'itération maximal à 200).

Exemple : `liste_nb_iteration_lychrel(20)` doit retourner :

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 2, 1]
```

Testez votre fonction sur des nombres plus importants.

41.9. (Bonus) On souhaite représenter graphiquement cette liste pour les p premiers entier.

a. Ajoutez la commande `import matplotlib.pyplot as plt` au début de votre fichier.

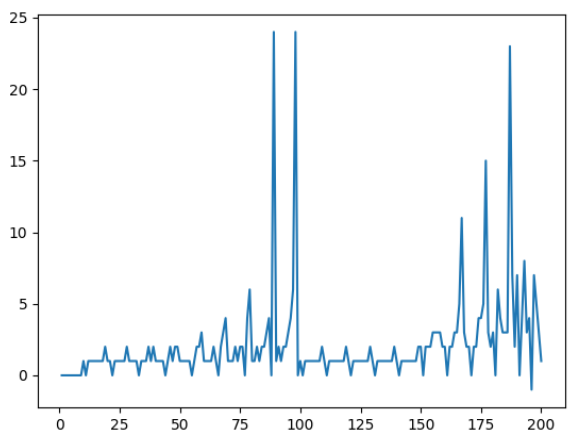
Pour dessiner des points en `python`, vous pouvez utiliser la commande `plt.plot(Lx,Ly)` où :

- `Lx` représente la liste des abscisses des points à afficher
- `Ly` représente la liste des ordonnées des points à afficher

puis utiliser la commande `plt.show()` pour afficher le graphique.

b. Affichez les nombres d'itérations calculés pour les entiers de 1 à 200.

Vous devez obtenir un graphique ressemblant à :



7 Manipulation de fichiers texte

Exercice 42 Lecture et écriture de fichiers texte

La commande :

```
f = open("#nom_du_fichier#","#option_d_ouverture#")
```

permet d'ouvrir le fichier *#nom_du_fichier#* se trouvant dans le répertoire d'où est lancé le programme `python`.

Il existe plusieurs valeurs possibles pour *#option_d_ouverture#* selon comment on souhaite manipuler le fichier :

- `"r"` : si on souhaite **lire** le fichier,
- `"w"` : si on souhaite **écrire** dans le fichier.

(Il existe d'autres options dont nous ne nous servirons pas dans ce TP.)

Pour la lecture, il y a deux commandes principales :

- `f.read()` qui retourne le contenu entier du fichier sous forme d'une chaîne de caractère.
- `f.readlines()` qui retourne un tableau contenant les lignes du fichiers (possédant le caractère de terminaison de ligne `\n` à la fin de chaque ligne, sauf éventuellement la dernière).

Pour l'écriture, il y a deux commandes principales :

- `f.write(chaine)` qui écrit dans le fichier le contenu de *chaine* (de type chaîne de caractères).
- `f.writelines(tab)` qui prend en paramètre un tableau de chaînes de caractères et écrit chaque élément de `tab` comme une ligne du fichier (attention à bien finir chaque élément de `tab` par le caractère de terminaison de ligne `\n`).

Attention : à la fin de la manipulation d'un fichier, n'oubliez pas de **fermer** ce fichier dans `python` par la commande :

```
f.close()
```

42.1.

a. Écrivez quelques lignes dans un fichier `test_fichier.txt` (à l'aide d'un éditeur de texte comme gedit ou vscode) et sauvegardez-le dans votre répertoire de travail.

b. À l'aide de `python`, lisez et affichez ce que contient votre fichier.

c. Toujours à l'aide de `python`, ouvrez un deuxième fichier en mode écriture et écrivez-y :

`"Voici le nouveau\ncontenu du fichier !"`

d. Lisez votre deuxième fichier et vérifiez que le contenu a bien été écrit.

Quelques remarques :

- Tous les fichiers ne sont pas des fichiers texte :
  - les fichiers .txt, .csv, .java, .py, .sh, .html ... sont des fichiers texte : vous pouvez les éditer avec un éditeur de texte (comme gedit ou vs code)
  - les fichiers .png, .jpg, .odt, .mp3, .doc, .pdf, .exe ... sont des fichiers en binaire : vous les consultez/éditez avec des logiciels dédiés (libreOffice et Word **ne sont pas** des éditeurs de texte).

- La commande d'ouverture en lecture d'un fichier non-existant provoque une erreur.
- La commande d'ouverture en écriture d'un fichier non-existant crée ce fichier.
- Lorsque vous souhaitez lire ou écrire dans un fichier qui ne se trouve pas dans votre répertoire de travail, il faut préciser le chemin jusqu'au fichier, en remplaçant l'argument *"#nom_du_fichier#"* par *"chemin_jusqu_au_fichier/#nom_du_fichier#"*.

Exercice 43 Dessin à partir d'un fichier

Un programme a produit un fichier `points.dat` dans lequel chaque ligne représente les coordonnées (flottantes) d'un point en 2 dimensions (séparées par un espace).

43.1. Écrivez une fonction `draw_points_from_file(nom_fichier)` prenant en paramètre le nom d'un fichier contenant des points en deux dimensions et es affichant sur un graphique en utilisant `matplotlib`.

Commandes utiles :

- `chaine.split(",")` : retourne un tableau contenant les portions de `chaine` entre `","`.
- `type(a)` : convertit la valeur de `a` en type `type`. Fonctionne notamment avec `int`, `float`, `str`.

Exercice 44 Lecture de fichiers .csv

Les fichiers .csv (*Comma-Separated Values*) sont des fichiers texte utilisés pour stocker des données, ils peuvent notamment être importés/exportés par des tableurs comme LibreOffice Calc ou Excel.

Certaines notes d'étudiants ont été stockées dans un fichier (pour les tests, vous pourrez utiliser le fichier `notes_etudiants.csv` à récupérer sur moodle), chaque ligne ayant le format suivant :

`num. étudiant ; note maths ; note info ; note élec ; note anglais \n`

```
195955;13.0;4.75;4.75;10.5
198806;10.25;7.0;8.75;19.5
198539;10.75;0.75;6.25;15.75
206701;14.25;0.25;19.0;18.25
215046;6.0;2.75;19.0;6.0
```

44.1. Écrivez une fonction `calcule_moyennes(nom_fichier)` prenant en paramètre le nom d'un fichier et retournant un dictionnaire dont les clés seront les numéros d'étudiants et les valeurs, les moyennes des étudiants sur ces 4 notes.

44.2. Écrivez une fonction `exporte_moyennes(dict_moyennes, nom_fichier)` permettant d'exporter un dictionnaire dont les clés sont des numéros d'étudiants et les valeurs, des nombres flottant représentant des moyennes dans un fichier `nom_fichier`.