TP 1 – Rappel et présentation de Python

Introduction

Un algorithme est une suite d'instructions élémentaires qui s'appliquent dans un ordre déterminé à des données et fournissant en un nombre fini d'étapes des résultats.
Une recette de cuisine est un algorithme. En effet il s'agit de :

- réunir les ingrédients ;
- préparer le plat ;
- arriver au plat prêt à être servi.

On peut considérer un algorithme comme une « boîte noire » fonctionnant selon le schéma suivant :

Entrée (données) → **Traitement** (instructions) → **Sortie** (résultats

Voici un exemple d'algorithme : « choisir un entier, le multiplier par 3, ajouter 1, donner le résultat ».
Si l'entier choisi est 4, l'algorithme donne . . .

Les algorithmes devront être écrits en **pseudo-code** ; c'est une écriture « en français » standardisée, indépendamment d'un langage spécifique comme Python, C, Java…

L'algorithme ci-dessus s'écrira ainsi.

```
Saisir a
b ← 3a + 1
Afficher b
```

Algorithme 1

La flèche ← désigne l'affectation des variables. L'algorithme ci-dessus fonctionne avec deux variables, a et b. Pour suivre l'état des variables à chaque étape, on construit ce tableau :

	a	b
Saisir a	4	
$b \leftarrow 3a + 1$	4	13
Afficher b	4	**13**

L'algorithme affiche b, c'est-à-dire 13.

1. Que retourne l'algorithme suivant s'il est exécuté avec 5 comme valeur de a ?

```
Saisir a
b ← 2a − 1
a ← 2b − 1
b ← 2a − 1
Afficher b
```

Algorithme 2

Compléter le tableau ci-dessous.

	a	b
Saisir a	5	

2. Même question avec l'algorithme suivant.
Construire le tableau pour suivre l'état des variables.

```
Saisir a
a ← a − 1
b ← a − 1
c ← b − 1
a ← a² + c
b ← a/b
Afficher b
```

Algorithme 3

Les ordinateurs ne sont fondamentalement capables de comprendre que quatre catégories d'instructions :

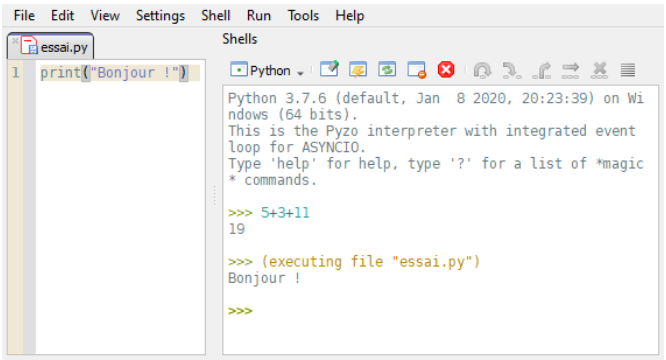
- la lecture / écriture ;
- l'affectation de variables ;
- les tests ;
- les boucles.

Nous venons d'utiliser les deux premières catégories dans les algorithmes 1, 2 et 3.

Présentation de Python

Python est un langage qui va nous permettre de programmer nos algorithmes.
Il existe plusieurs versions, parmi lesquelles

- version en ligne : https://repl.it/languages/python3 ;
- version à installer sur un ordinateur (EduPython, Pyzo…). Je préconise Pyzo.



Deux fenêtres sont essentielles pour travailler.

- La fenêtre de gauche (« **éditeur** ») où vous pouvez écrire et enregistrer votre code dans un fichier (dit « fichier source »). Elle possède des lignes numérotées sur la gauche qui vous permettent de vous repérer facilement dans le fichier, notamment quand Python vous signalera l'existence d'un bug dans votre programme.
- La fenêtre de droite (« **console** ») où vous pouvez rentrer des commandes pour tester leur comportement sans que ce soit sauvegardé dans un fichier (elle doit donc être cantonnée au rôle de test rapide).

Elle est facilement reconnaissable au symbole >>> qui commence chacune des lignes où vous pouvez rentrer du code. *C'est aussi à l'intérieur de celle-ci que sera exécuté le code de votre fichier principal.*

Affectation et comparaison

3. Quelle valeur retourne l'algorithme suivant (à la main) ? . . . . . .

$a \leftarrow 3$
$b \leftarrow a + 2$
$c \leftarrow a \times b$
$b \leftarrow c - 10$
Afficher b

L'affectation d'une valeur à une variable, que l'on note ← en pseudo-code, se fait par le symbole = en Python.
Traduire l'algorithme ci-dessus en Python puis le programmer.

```
a=3



print(  )
```

4. Compléter les réponses de Python ci-dessous.

```
>>> a=5
>>> a*10

>>> b=1.2
>>> b*10

>>> 2**3

>>> 10/5
```

Que permet de faire l'opérateur ** ? . . . . . .

5. Dire si les affirmations suivantes sont vraies ou fausses.

$5 = 6$: . . . . . $5 \neq 6$: . . . . .
$5 > 7$: . . . . . $2 \leq 2$: . . . . .

En python, les commandes pour effectuer des tests sont :

- comparaison : . . . .
- différent de : . . . .
- plus grand ; plus grand ou égal : . . . .
- plus petit ; plus petit ou égal : . . . .

La réponse est vrai (True) ou faux (False)

```
>>> 5==6
False
>>> 5!=6
True
```

6. On pose $A = 600^{700}$, $B = 700^{600}$ et $C = 650^{650}$.

a. À l'aide de Python, déterminer parmi les réels A et B lequel est le plus petit : . . . . . .

Recopier les commandes saisies.

```
>>> A=600**700
```

b. Classer par ordre croissant les nombres A, B, C.

Types de variables

Il existe plusieurs types de variables en Python.

- Le type **int** désigne les entiers (2, -3, ...) ;
- Le type **float** désigne les nombres à virgule ;
- Le type **string** désigne les chaînes de caractères, *toujours écrites entre guillemets ou apostrophes* : « toto », 'bonjour', ... ;
- Le type **bool** qui désigne les variables booléennes ne pouvant prendre que les valeurs « vrai » ou « faux » (généralement ce sont des résultats de tests, voir ci-dessus).

7. Exécuter les commandes ci-dessous et compléter les trous.

```
>>> a=5
>>> type(a)

>>> a*3

>>> b='bonjour'
>>> type(b)

>>> b*3
```

8. Donner le type des variables suivantes :

- 5-28+1 : . . . . .
- 6/3 : . . . . .
- 'abc' : . . . . .
- 5!=7 : . . . . .

TP 2 – Les fonctions print et input

Exécuter le script suivant. Que se passe-t-il ? . . .

```
a=8
b=2*a
```

Et pour celui-ci ? . . . . . .

```
a=8
b=2*a
print(b)
```

La fonction **print** permet aussi d'afficher plusieurs variables, à condition de les séparer par des virgules :

```
a=8
b=2*a
print("le double de",a,"vaut",b)
```

Exécuter le script ci-dessous.

```
nom=input("Comment vous appelez-
vous ?")
print("Bonjour",nom)
```

Que permet de faire la fonction **input** ? . . . . . .
. . . . . .

Exécuter le script ci-dessous avec le nombre de votre choix.

```
a=input("Choisissez un entier")
b=2*a
print("le double de",a,"vaut",b)
```

Le résultat est-il celui attendu ? . . . . .
Donner le type de la variable **a** . . . . .
Résoudre le problème rencontré à l'aide de la fonction **int** qui convertit une variable en type entier (quand c'est possible).
Recopier le script ci-dessous.

```

```

On retiendra que la fonction input renvoie une variable de type str et que par conséquent selon l'usage que l'on veut en faire, il faut la convertir en réel, avec la float, ou en entier, avec int.

TP 3 – Instructions conditionnelles

La résolution de certains problèmes nécessite la mise en place d'un test pour effectuer une tâche :

- si le test est positif, on effectue la tâche ;
- sinon, c'est-à-dire si le test est négatif, on effectue une autre tâche.

Le « sinon » n'est pas obligatoire. En son absence, lorsque le test est négatif, la tâche ne sera pas effectuée et l'algorithme passera à l'instruction suivante.

Le programme ci-dessous détermine, à partir de l'âge a d'une personne, si elle est mineure ou majeure.

```
Si a < 18 alors
    afficher « vous êtes mineur »
Sinon
    afficher « vous êtes majeur »
Fin Si
```

Voici sa traduction en Python.

```
age=input("Entrez votre âge")
age=float(age)
if age < 18:
    print(" .   .   .   .   .  ")
else:
    print(" .   .   .   .   .  ")
```

Le « alors » s'écrit avec deux points en Python. L'indentation délimite une instruction conditionnelle.

1. Compléter le programme, saisir et exécuter le script avec deux âges pour s'assurer du bon fonctionnement, l'un inférieur à 18, l'autre supérieur.

2. Écrire un programme qui détermine si un entier est pair ou impair.
On rappelle que le reste de la division euclidienne de n par 2 s'obtient avec $n\%2$.

3. On considère un algorithme qui affiche

- « normal » si la température est strictement supérieure à 5° ;
- « froid » si elle est comprise entre $-5°$ et 5° ;
- « grand froid » sinon.

On propose deux algorithmes, prenant en entrée la température T.
Compléter le second et le programmer.

```
Si T > 5 alors
    afficher « normal »
Fin Si
Si (T ≥ −5) et (T ≤ 5) alors
    afficher « froid »
Fin Si
Si T < −5 alors
    afficher « grand froid »
Fin Si
```

```
Si . . . alors
    afficher « normal »
Sinon
    Si . . . alors
        afficher « froid »
    Sinon
        afficher « grand froid »
    Fin Si
Fin Si
```

TP 4 – La boucle « pour »

Il est aussi possible de demander à un ordinateur de répéter une même tâche autant de fois que l'on veut. Cela se fait grâce à ce qu'on appelle une **boucle**. Regardons de suite un exemple pour voir l'intérêt de cette notion.

L'algorithme ci-contre affiche les entiers de 8 à 15.
Le principe de fonctionnement est le suivant : l'ordinateur affecte à K la valeur 8 comme il lui est demandé puis effectue les instructions comprises entre « Pour » et « Fin Pour ». Il augmente ensuite automatiquement K de 1 (donc K vaut 9) puis effectue à nouveau le bloc d'instruction. K est alors augmenté de 1 et ainsi de suite jusqu'à ce que K vaille 15.

```
Pour K de 8 à 15
    afficher K
Fin Pour
```

En Python, la syntaxe est la suivante.

```
for k in range(8,16):
    print(k)
```

Notez qu'il faut terminer à 16 et non à 15 (l'explication est que Python comprend ceci : « pour k entier dans l'intervalle $[8; 16[$ », par conséquent si l'on écrit `range(8,15)`, la boucle s'arrêtera à $k = 14$ qui est le dernier entier de l'intervalle $[8; 16[$).

1. Saisir le script précédent et l'exécuter.

2. Faire afficher les nombres pairs de 30 à 42

3. Faire afficher les nombres impairs de 11 à 25.

4. Que fait l'algorithme ci-contre ? Combien retourne-t-il ? . . . . . .

```
S ← 0
Pour K de 30 à 42
    S ← S + K
Fin Pour
Afficher S
```

5. Au premier 1er janvier 2020, Sophie a placé 1000 € sur un compte rémunéré à 2 %. Elle verse chaque 1er janvier une somme de 100 € sur ce compte.
   On suppose qu'aucun retrait n'est effectué sur ce compte.
   Le 1er janvier 2021, sa banque lui versera donc
   $$0{,}02 \times 1000 = 20 \text{ euros,}$$
   ce qui lui fera un capital de 1120 €.
   Écrire un algorithme pour afficher le tableau des capitaux de 2021 à 2040.

```
2021 1120.0
2022 1242.4
2023 1367.248
2024 1494.5929600000002
…
```

6. La suite de **Fibonacci** est la suite des nombres
   0 ; 1 ; 1 ; 2 ; 3 ; 5 ; 8 ; 13 ; 21 ; 34 ; …
   dans laquelle un terme est la somme des deux précédents.
   Écrire un algorithme qui demande à l'utilisateur un entier $n \geq 1$ et qui renvoie le n^{e} terme de la suite. Par exemple pour $n = 9$, l'algorithme renverra 21.

```
choix d'un entier ? 1
0

Choix d'un entier ? 9
21
```

7. Depuis le 1er janvier 2000, Robert fume 20 cartouches de cigarettes par an. En 2000, cela représentait un budget de 700 € par an.
   Chaque année, le prix du tabac augmente de 6 %.
   Compléter l'algorithme ci-dessous pour calculer l'argent dépensé par Robert en cigarettes du 1er janvier 2000 au 31 décembre 2020, puis le programmer.

```
A ← 700  # dépense annuelle
S ← 700  # dépense cumulée
Pour k de . . . à . .
    A ← . . .
    S ← . . .
Fin Pour
Afficher . . .
```

TP 5 – La boucle « tant que »

En algorithmique, on peut être amené à répéter un bloc d'instructions tant qu'une condition est vérifiée. En langage naturel, une telle répétition en boucle peut se formuler par « tant que », traduit par « while » dans la plupart des langages.

Si la condition qui suit l'instruction « tant que » est vérifiée, alors le programme exécute toutes les instructions du bloc qui suit ; à la fin de cette exécution il regarde à nouveau si la condition est vérifiée et ainsi de suite jusqu'à ce qu'elle ne le soit plus.

```
K ← 8
Tant que K ≤ 15
   afficher K
   K ← K + 1
Fin Tant que
```

1. Toute boucle « pour » peut être remplacée par une boucle « tant que » à l'aide d'un compteur.
L'algorithme ci-contre affiche lui aussi les entiers de 8 à 15.
Voici sa traduction en Python. L'expérimenter.

```
k=8
while k<=15:
    print(k)
    k=k+1
```

En revanche il existe des utilisations de « tant que » qui ne peuvent pas être remplacée par celle de « pour ».

2. À l'aide d'une boucle tant que, afficher les multiples de 23 compris entre 100 et 1000.

3. L'effectif d'une population de bactéries double chaque jour. Si la population est de 100 bactéries le premier jour, au bout de combien de jours dépassera-t-elle 2 000 bactéries ? (à la main) . . . . . .

Si la question avait été « au bout de combien de temps la population dépassera-t-elle 6 milliards de bactéries ? », le calcul à la main aurait été fastidieux. Compléter l'algorithme ci-dessous et le programmer.

```
J ← . . .
P ← . . .
Tant que P ≤ . . .
   P ← . . .
   J ← . . .
Fin Tant Que
Afficher . . .
```

J compte le nombre de jours et P est le nombre de bactéries le J-ième jour.

4. Une balle est lâchée de 100 mètres de haut et rebondit aux 4/5ème de sa hauteur à chaque rebond.

a. Calculer la hauteur du premier et du second rebond.

b. On considère que la balle ne rebondit plus si le rebond est inférieur à 1 cm. Au bout de combien de rebonds la balle ne rebondit plus ?

5. À l'aide d'un algorithme, déterminer en quelle année Sophie (voir TP 4) aura dépassé les 10 000 € sur son compte.

6. Déterminer le premier terme de la suite de Fibonacci (voir TP 4) plus grand que 1000.

7. On veut creuser un puits. Le 1er mètre coûte 100 €, le 2e mètre coûte 120 €, le 3e mètre 140 € et ainsi de suite.

a. Combien coûtera un puits de 33 mètres ?

b. On dispose d'un budget de 35 000 €.
Quelle profondeur maximale peut-on creuser ?

TP 6 – Les fonctions en Python

Considérons la fonction f qui a un nombre associe son double. Cela s'écrit $f: x \mapsto 2x$.
En Python pour définir une fonction, on utilise la commande **def** ; pour afficher la valeur à retourner, on utilise **return**.
La définition de la fonction s'arrête lorsque l'indentation se termine.
Exécuter le script suivant.

```
def double(nombre):
    return 2*nombre
```

Dans la console, il suffira de saisir :

```
>>> double(10)
20
>>> double(-8)
-16
```

Un des avantages des fonctions est de pouvoir les « enchaîner ».

```
>>> double(double(10))
40
```

Essayez donc de faire la même chose en remplaçant **return** par **print**, vous aurez un joli message d'erreur !

L'usage de la fonction print est interdit dans une fonction (sauf pour débuguer).

1. Écrire une fonction **age** dont l'appel donnera par exemple :

```
>>> age(15)
'vous êtes mineur'

>>> age(19)
'vous êtes majeur'
```

2. Écrire une fonction **pair(n)** qui prend en argument un entier n et retourne 1 s'il est pair, 0 sinon.

```
>>> pair(10)
1
>>> pair(11)
0
```

3. Écrire une fonction **sophie(n)** qui renvoie l'argent disponible au 1er janvier de l'année n (voir TP 4).

```
>>> sophie(2021)
1120
```

4. Écrire une fonction **fibo(n)** qui renvoie le n^{e} nombre de la suite de Fibonacci (voir TP4).

```
>>> fibo(9)
21
```

5. Écrire une fonction **somme(n)** qui calcule la somme
$$1 + 2 + 3 + \cdots + n$$
où n est un entier donné en argument à la fonction.

```
>>> somme(4)
10
```

6. Écrire une fonction **sommefibo(n)** qui renvoie la somme des n premiers termes de la suite de Fibonacci.

```
>>> sommefibo(5)
7
```

7. Écrire une fonction qui prend en argument un entier *à deux chiffres* et renvoie la somme des ses chiffres.
On rappelle que
   - `a%b` renvoie le reste de a par b ;
   - `a//b` renvoie le quotient de a par b.

```
>>> sommechiffres(59)
14
```

Généraliser à un nombre quelconque de chiffres.

```
>>> sommechiffres(25155)
18
```

8. Écrire une fonction **binversdec(n)** qui convertit un entier n écrit en binaire vers son écriture décimale.

```
>>> binversvec(1001)
9
```

TP 7 – Les listes en Python

1. Une liste est une variable qui contient plusieurs variables, elle se note entre crochets : […]
   - La longueur d'une liste s'obtient par `len(L)`,
   - Le terme numéro n s'obtient par `L[n]` (en commençant à 0). Par exemple `L[0]` désigne le premier terme de la liste.

```
>>> L = [1, 'a', 6, -7]
>>> len(L)
4
>>> L[0]
1
```

   On peut **concaténer** deux listes en une seule en utilisant l'opérateur + ; cela permet par exemple d'ajouter un élément à la fin d'une liste :

```
>>> L + [9]
[1, 'a', 6, -7, 9]
```

2. Écrire une fonction **dernier(L)** qui renvoie le dernier élément d'une liste L.

```
>>> dernier([7,8,3])
3
```

3. Écrire la liste des nombres pairs entre 10 et 100.

4. Écrire une fonction **moyenne(L)** qui calcule la moyenne des éléments d'une liste L.

```
>>> moyenne([3,2,7])
4
```

5. Écrire une fonction **doublon(L)** qui supprime les doublons dans une liste L.

```
>>> doublon([4,5,1,5,7,1])
[4,5,1,7]
```

6. Écrire une fonction qui renvoie la liste des premiers termes de la suite de Fibonacci.

```
>>> fibonacci(8)
[0, 1, 1, 2, 3, 5, 8, 13]
```

7. Donner la liste des entiers compris entre 1 et 200 qui sont des multiples de 7, mais qui ne sont ni des multiples de 5, ni de 2.

8. Écrire une fonction **listeversnombre(L)** qui convertit une liste de chiffres L en le nombre composé de ces chiffres.

```
>>> listeversnombre([2,3,1,5])
2315
```

9. Écrire la fonction **nombreversliste(L)**, inverse de la précédente.

10. On lance un pièce qui fait pile (1) ou face (0).
    a. Écrire une fonction qui compte le nombre de « double pile » obtenus. Attention, un P ne peut être que dans un seul « double pile ».

```
>>> doublepile([1,1,1,0,1,1])
2
```

    b. Écrire une fonction qui répond True si le tirage se termine par un « double pile », False sinon.

```
>>> doublepilefin([0,1,0,1,1,1])
False
```

    c. Sur 1000 lancers de longueur 400, compter le nombre de ceux qui se terminent par un « double pile ». Recommencer plusieurs fois. Que peut-on dire de ce nombre ?

Distributeur de billets et pièces

Un automate permet de retirer de l'argent liquide.
Cet appareil contient un nombre fixé de billets de 200, 100, 50, 20, 10 et 5 euros ainsi que des pièces de 2 et 1 euro. On ne peut retirer qu'un nombre entier d'euros.
Écrire un programme pour rendre la monnaie avec le moins de billets et pièces possibles.
On pourra stocker le contenu de l'appareil sous forme d'une liste, par exemple :
[(200,5),(100,4),(50,12),(20,15),(10,12),(5,7),(2,10),(1,23)]
(5 billets de 200 euros, 4 billets de 100 euros…). Sur l'exemple précédent, on obtiendra :

```
Somme ? 448
2 billets de 200
2 billets de 20
1 billet de 5
1 pièce de 2
1 pièce de 1
```

TP 8 – Les chaînes de caractères

Une chaîne de caractère (type str) est une liste de caractère délimitée par des guillemets.
On utilise alors les mêmes commandes que pour les listes.

```
>>> ch='merci'
>>> len(ch)
5
>>> ch[2]
'r'
>>> ch+'catrice'
'mercicatrice'
```

1. Écrire une fonction **unsurdeux** qui renvoie une chaîne en ne prenant qu'un caractère sur deux.

```
>>> unsurdeux('Bonjour !!!')
'Bnor!!'
```

2. Écrire une fonction **joie** qui renvoie une chaîne en répétant les caractères comme ci-dessous.

```
>>> joie(ch)
'meerrrccccciiiii'
```

3. Écrire une fonction **voyelle** qui ne renvoie que les voyelles d'une chaîne de caractère.

```
>>> voyelle('Bonjour !!!')
'oou'
```

4. Donner une nouvelle version de la fonction **sommeschiffres** sans utiliser la division euclidienne.

```
>>> sommechiffres(25155)
18
```

5. Écrire une fonction **hexversdec** qui convertit un entier écrit en binaire vers son écriture décimale.

```
>>> hexversdec('3A')
58
```

6. Écrire une fonction **decvershex** inverse de la précédente.

```
>>> decvershex(58)
'3A'
```

7. On dit qu'un nombre est « croissant » si les chiffres qui le composent apparaissent dans l'ordre croissant quand on lit le nombre de gauche à droite.
   Par exemple 125 et 358 sont des nombres croissants (car $1 < 2 < 5$ et $3 < 5 < 8$) en revanche 4356 et 1563 ne sont des nombres croissants.
   Écrire une fonction qui renvoie True ou False selon que le nombre entier entré en argument est croissant ou pas.

```
>>> est_croissant(1489)
True

>>> est_croissant(14892)
False
```

Calendrier perpétuel

La formule de Zeller permet de calculer le jour de la semaine (lundi, mardi, …) à partir de sa date. Cette formule fonctionne pour le calendrier grégorien, c'est-à-dire pour les dates postérieures au 15 octobre 1582.
On note

- j la date du jour ;
- m le numéro du mois : 3 pour mars, 4 pour avril, …, 13 pour janvier et 14 pour février ;
- a le numéro de l'année. Dans la formule, janvier et février sont comptés pour les mois 13 et 14 de l'année précédente. Par exemple, pour calculer le jour du 12 janvier 2007, on cherchera le jour du « 12/13/2006 » ;
- h le numéro du jour avec 0 pour samedi, 1 pour dimanche, 2 pour lundi, …, 6 pour vendredi.

On note $\lfloor x \rfloor$ la partie entière d'un nombre x, par exemple $\lfloor 13{,}52 \rfloor = 13$ et $\left\lfloor \frac{5}{3} \right\rfloor = 1$. Ainsi $\left\lfloor \frac{a}{b} \right\rfloor$ est le quotient de la division euclidienne de a par b.
La formule de Zeller affirme que h est le reste de la division euclidienne par 7 de

$$j + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + a + \left\lfloor \frac{a}{4} \right\rfloor - \left\lfloor \frac{a}{100} \right\rfloor + \left\lfloor \frac{a}{400} \right\rfloor.$$

Par exemple, pour le 12 janvier 2007, c'est-à-dire le « 12/13/2006 », on a $j = 12$, $m = 13$ et $a = 2006$.
Puisque $\left\lfloor \frac{13(m+1)}{5} \right\rfloor = \left\lfloor \frac{13\times 14}{5} \right\rfloor = \lfloor 36{,}4 \rfloor = 36$, $\left\lfloor \frac{2006}{4} \right\rfloor = 501$, $\left\lfloor \frac{2006}{100} \right\rfloor = 20$ et $\left\lfloor \frac{2006}{400} \right\rfloor = 5$, le calcul est

$$12 + 36 + 2006 + 501 - 20 + 5 = 2540.$$

Comme $2540 = 7 \times 362 + 6$, on en déduit $h = 6$ et le jour cherché est vendredi.

Écrire une fonction **zeller** qui prend en argument la date du jour, le mois et l'année et qui renvoie

```
>>> zeller(12,1,2007)
'vendredi'
```

La modifier pour qu'elle prenne la date souhaitée sous forme d'une chaîne de caractère : jj-mm-yyyy

```
>>> zeller('12-01-2007')
'vendredi'
```

TP 9 – Matrices

On rappelle qu'une matrice peut être représentée en Python par la liste de ses lignes.

1. Par un algorithme, écrire :
   a. la matrice de taille 7×3 remplie de 0 ;
   b. la matrice carrée identité I_{10} ;
   c. la matrice carrée de taille 10 dans laquelle tous les coefficients sont égaux à 1, sauf ceux de la diagonale « montante » égaux à 7 :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 7 \\ 1 & 1 & 1 & 7 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 7 & 1 & 1 & 1 \\ 7 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

2. À quelle condition une liste de listes représente-t-elle une matrice ? En admettant que cette condition soit satisfaite, écrire une fonction **dim(L)** qui renvoie la dimension d'une matrice représentée par la liste L.

```
>>> dim([[2,1,-3],[7,0,0]])
(2, 3)
```

3. Écrire une fonction **est_valide(L)** qui renvoie True si la liste saisie est une matrice et False sinon.

```
>>> est_valide([[2,1,-3],[7,0]])
False
```

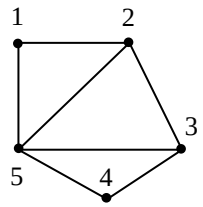
4. Écrire une fonction **est_sym(L)** qui renvoie True ou si liste saisie est une matrice carrée symétrique et False sinon.

```
>>> est_sym([[2,1,0],[1,0,0],[0,0,0]])
True
```

5. Écrire une fonction **transposee(L)** qui renvoie la matrice symétrique de L par rapport à sa diagonale descendante (appelée « transposée » de L).

```
>>> symetrique([[1,2,4],[3,0,0]])
[[1,3],[2,0],[4,0]]
```

6. Chaque matin, parmi cinq élèves d'une même classe, certains se serrent la main, et d'autres non, selon le schéma ci-contre. On définit la matrice A des poignées de mains comme suit : l'élément a_{ij} de A vaut 1 si l'élève i a serré la main de l'élève j et 0 sinon.



   a. Saisir la matrice A sur Python.
   b. Écrire une fonction **sont_amis(a,b)** qui retourne True si les élèves a et b se serrent la main et False sinon.

7. Écrire une fonction **symetrise(L)** qui renvoie la matrice symétrique construite à partir de la liste des coefficients situés au-dessus de la diagonale descendante. Par exemple pour construire la matrice

$$A = \begin{pmatrix} \mathbf{1} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ 3 & \mathbf{0} & \mathbf{1} & \mathbf{3} \\ 4 & 1 & \mathbf{2} & \mathbf{4} \\ 5 & 3 & 4 & \mathbf{7} \end{pmatrix}$$

seule la connaissance des termes en gras est nécessaire.

```
>>>syme
  trise([[1,3,4,5],[0,1,3],[2,4],[7]])

[[1, 3, 4, 5], [3, 0, 1, 3], [4, 1,
2, 4], [5, 3, 4, 7]]
```

Opérations sur les matrices

1. Écrire une fonction **add_mat(L,M)** qui ajoute les matrices L et M.
   On gérera les cas d'exception avec la fonction **raise Exception**.

```
>>> add_mat([[1],[3]],[[0],[-1]])
[[1],[2]]

>>> add_mat([[1],[3]],[[0,4],[-1]])
Exception: Matrice non valide

>>> add_mat([[1],[3]],[[0]])
Exception: Addition impossible
```

2. Écrire une fonction **mult_mat(L,M)** qui multiplie les matrices L et M, en gérant les deux mêmes cas d'exception que pour l'addition.

3. Écrire une fonction **puiss_mat(L,n)** qui calcule la puissance n^{e} de la matrice L.

TP 10 – Problèmes

Le Code EAN-13

(*European Article Numbering*) est un code-barres utilisé par le commerce et l'industrie.
Les 12 premiers chiffres sont propres à l'article, le treizième est la clé de contrôle calculée de la façon suivante.
Si les douze premiers chiffres sont $a_1a_2 \dots a_{12}$, on calcule le reste r dans la division euclidienne par 10 de
$S = a_1 + a_3 + \cdots + a_{11} + 3(a_2 + a_4 + \cdots + a_{12})$.

- si ce reste est 0, la clé est 0
- sinon, la clé est $10 - r$.

1. Écrire une fonction `cle` calculant la clé à partir des douze premiers chiffres.

```
>>> cle(590123412345)
7
```

2. Écrire une fonction `cle_valide` calculant la clé à partir des douze premiers chiffres qui renvoie `True` ou `False` selon qu'une clé saisie de treize chiffre est valide ou pas.

3. Modifier la fonction `cle` pour qu'elle admette en paramètre une clé avec des éventuels espaces ou tirets.

```
>>> cle('5 9012-3412-345')
7
```

Conjecture de Syracuse

On choisit un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. En répétant l'opération, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur.

Exemple 1. En partant de 1 on a successivement $1 \to 4 \to 2 \to 1 \to 4 \to 2 \to 1 \to \dots$ On constate que la suite des nombres obtenus se répète (c'est un cycle de longueur 3).

Exemple 2. On part ici de 3. On a alors
$$3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1,$$
et ce n'est pas la peine de continuer, puisque ayant obtenu 1, on sait que l'on va retomber indéfiniment dans le cycle $1 \to 4 \to 2 \to 1$ de l'exemple 1.

Les mathématiciens ont toutes les raisons de penser que quel que soit le nombre initial choisi, on retombe toujours sur 1, mais aucun n'a réussi à prouver ce résultat jusqu'à présent, ce qui justifie le statut de conjecture de ce résultat.

On appelle **durée de vol** d'un entier le nombre d'entiers calculés avant de tomber sur 1 et **hauteur de vol** la plus grande valeur prise par la suite des nombres obtenus avant de tomber sur 1. Ainsi la durée de vol de 3 est 7 et sa hauteur de vol est 16.

Compléter le tableau suivant.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
durée	1		7												
hauteur	1		16												

1. Écrire les fonctions `vol`, `duree` et `hauteur`.

```
>>> vol(7)
[7, 22, 11, 34, 17, 52, 26, 13,
40, 20, 10, 5, 16, 8, 4, 2, 1]
```

```
>>> duree(7)
16
```

```
>>> hauteur(7)
52
```

2. Écrire une fonction `dureemax` prenant en entrée deux entiers a et b et renvoyant l'un des entiers compris entre a et b ayant la plus grande durée de vol ainsi que cette durée de vol maximale.

```
>>> dureemax(8,14)
(9, 19)
```

3. Faire de même avec une fonction `hauteurmax` renvoyant l'un des entiers compris entre a et b et ayant la plus grande hauteur ainsi que cette hauteur maximale.

```
>>> hauteurmax(1,15)
(15, 160)
```

Somme de deux carrés

Écrire une foncttion `somme2carres(n)` qui détermine les décompositions distinctes comme somme de deux carrés d'entiers un entier naturel n donné.
Une telle décomposition d'un entier n peut toujours s'écrire $n = a^2 + b^2$ avec $0 \leq a \leq b \leq n$, c'est celle-ci que l'on donnera.

```
>>> somme2carres(25)
[[0, 5], [3, 4]]
>>> somme2carres(21)
[]
>>> somme2carres(325)
[[1, 18], [6, 17], [10, 15]]
```

Démontrer que le plus entier à avoir trois décompositions distinctes est 325.
Quel est le plus petit entier à avoir quatre décompositions distinctes ?

Somme des quatre carrés

Écrire un programme `somme4carres(n)` qui détermine les décompositions distinctes comme somme de deux carrés d'entiers un entier donné.

```
>>> somme4carres(49)
[[0, 0, 0, 7], [0, 2, 3, 6], [1,
4, 4, 4], [2, 2, 4, 5]]
```

Prouver que tous les entiers positifs inférieurs ou égaux à 1000 peuvent s'écrire comme la somme de quatre carrés (certains pouvant être égaux à 0).

Somme des trois carrés

Montrer qu'il y a 165 entiers inférieurs à 1000 qui ne peuvent pas s'écrire comme une somme de 3 carrés (comme par exemple 7, 15…).

Problème des soldats

41 soldats, numérotés de 1 à 41, sont disposées en cercle. On procède à la décimation (punition militaire romaine qui consistait à « éliminer » une personne dans un groupe) de la façon suivante : le numéro 2 est exécuté, puis le 4, le 6 ainsi de suite en exécutant un soldat sur 2.
Attention, après un tour complet, il y a des places vacantes, et ça se complique.
La question est : où se placer dans le cercle d'origine pour être le dernier survivant ? Autrement dit, quel sera le dernier numéro restant dans le cercle ?
Exemple avec 11 soldats, l'ordre des malheureux exécutés est 2, 4, 6, 8, 10, 1, 5, 9, 3, 11, donc 7 a la vie sauve.
Écrire un algorithme pour répondre et le généraliser à un nombre quelconque de soldats.

Pierre, feuille, ciseaux

On rappelle les règles :

- Les ciseaux coupent le papier ;
- la feuille recouvre la pierre ;
- la pierre bat les ciseaux.

On code ciseaux par 0, feuille par 1 et pierre par 2. On définit la matrice carrée $A = (a_{ij})$ de taille 3 par

- $a_{ij} = 0$ si i perd contre j ;
- $a_{ij} = 1$ si i gagne contre j ;
- $a_{ij} = 2$ s'il y a égalité ;

On a donc

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix}.$$

Voici une version avec 5 objets de ce jeu : ciseaux, feuille, lézard, pierre et Spock.

- Les ciseaux coupent la feuille ;
- La feuille recouvre la pierre ;
- la pierre écrase le lézard ;
- le lézard empoisonne Spock ;
- Spock écrabouille les ciseaux ;
- les ciseaux décapitent le lézard ;
- le lézard mange la feuille ;
- la feuille repousse Spock ;
- spock détruit la pierre ;
- la pierre bat les ciseaux.

1. Saisir la matrice du jeu dans Python.
2. Écrire une fonction **jeu(c)** qui prend en entrée ce que vous jouez et qui retourne si vous perdu, gagné ou s'il y a égalité contre l'ordinateur qui a choisi au hasard un objet qu'il indiquera.

```
>>> jeu("Spock")
('ciseaux', 'vous avez gagné')

>>> jeu("pierre")
('pierre', 'égalité')
```

3. On voit donc qu'un jeu correspond à une matrice. Quelles sont les conditions sur une matrice pour qu'elle donne un jeu valide ?
   Écrire une fonction **cpu_alea(n)** qui renvoie une matrice de taille n définissant un jeu.

```
>>> cpu_alea(3)
[[2,1,0],[0,2,1],[1,0,2]]
```

Fusionner deux listes ordonnées

On considère deux listes A et B d'entiers classés par ordre croissant. Écrire une fonction qui fusionne ces deux listes en une liste croissante.
On pourra utiliser la trame suivante et la méthode *pop()*.

```
def fusion(A, B) :
   C = []
   while A != [] and B != []:
      if A[0] < B[0]:

      else:

   return C + A + B
```

```
>>> fusion([2,5,11],[1,2,3,8])
[1,2,2,3,5,8,11]
```

Une suite logique…

Comment est constituée la suite logique suivante ? Écrire un algorithme pour générer les premiers termes.

```
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
11131221133112132113212221
```

Triangle de Pascal

Le triangle de Pascal est un triangle de nombres obtenu de la façon suivante :

- des 1 sur la première colonne et la diagonale ;

- les autres nombres s'obtiennent comme la somme « des deux du dessus ». Par exemple $10 = 4 + 6$.

1					
1	**1**				
1	2	**1**			
1	3	3	**1**		
1	4	6	4	**1**	
1	5	10	10	5	**1**

1. Écrire une fonction **pascal(n)** qui calcule le tableau de la première ligne (numérotée 0) à ligne n.

```
>>> pascal(3)
[[1],[1,1],[1,2,1],[1,3,3,1]]
```

2. Que remarque-t-on lorsque l'on fait la somme des nombres d'une ligne ?
Écrire un programme pour vérifier que votre conjecture est vraie jusqu'à $n = 50$.

3. On remarque que les deux nombres obtenus en faisant la somme par ligne d'un terme sur deux sont égaux (par exemple $1 + 6 + 1 = 4 + 4$).
Écrire un programme pour vérifier cela jusqu'à $n = 50$.

Affichage de tableau de nombres

Écrire une fonction pour afficher « proprement » des tableaux de nombres saisis sous forme de listes de listes.

```
>>> M=[[25,1,875],[0,25],[78,476]]

>>> aff(M)
25 1   875
0  25
78 476
```

```
>>> aff(pascal(9))
1
1 1
1 2 1
1 3 3  1
1 4 6  4  1
1 5 10 10 5   1
1 6 15 20 15  6   1
1 7 21 35 35  21  7  1
1 8 28 56 70  56  28 8  1
1 9 36 84 126 126 84 36 9 1
```

Une équation

Montrer que l'équation $x^2 + 41y^2 = z^7$ admet au moins une solution en nombres entiers avec $x \geq 1$ et $y \geq 1$.

Chiffrement de César

À chaque lettre de l'alphabet, on associe son rang dans l'alphabet, en commençant à 0 et en rajoutant un 0 devant s'il n'y a qu'un chiffre (a → 00, b → 01, …, c → 25).

On choisit un entier **cle** compris entre 0 et 25. Le chiffrement de César consiste à décaler le rang des lettres dans l'alphabet de **cle**. Par exemple, avec la cle 2, la lettre z, dont le rang est 25, devient la lettre de « rang » 25 + 2 = 27, c'est-à-dire la lettre de rang 27 – 26 = 1, le b donc.

1. Écrire une fonction `codage(mot,cle)`.

```
>>> codage_cesar("gaz",2)
'icb'
```

2. Écrire une fonction `decodage_cesar(mot, cle)` en deux lignes.

```
>>> decodage_cesar("icb",2)
'gaz'
```

Chiffrement RSA

Nous supposerons que le message à coder ne comporte que des lettres minuscules. Chaque lettre est codée par son rang dans l'alphabet (avec un 0 devant s'il n'y a qu'un chiffre, et en commençant à compter à 0).

1. Créer les fonctions permettant l'encodage et le décodage d'un message sous forme d'un nombre.

```
>>> mess_vers_code('bob')
11401

>>> code_vers_mess(11401)
'bob'
```

Ouvrir l'article du chiffrement RSA de Wikipédia.

2. Le chiffrement RSA est-il une méthode symétrique ou asymétrique ?

Voici les cinq étapes du chiffrement RSA :

- **a.** Choisir p et q, deux nombres premiers distincts ;
- **b.** calculer leur produit $n = pq$, appelé *module de chiffrement* ;
- **c.** calculer $A = (p-1)(q-1)$;
- **d.** choisir un entier naturel *e* premier avec *A* et strictement inférieur à *A*, appelé *exposant de chiffrement* ;
- **e.** calculer l'entier naturel d strictement inférieur à *A* tel que $d \times e \equiv 1\ [A]$.
  On l'appelle *exposant de déchiffrement*. Son existence est assurée par un théorème de mathématiques (hors programme) et son calcul peut s'effectuer par l'algorithme donné à la question 10.

3. Quelle est la clé publique communiquée par Alice si elle veut qu'on lui écrive ?

4. Quelle est la clé privée que va utiliser Alice pour décrypter un message envoyé par Bob ?

5. Soit M un message (écrit sous forme d'un nombre, avec $M < n$) que Bob veut envoyer à Alice. Comment calcule-t-il M', le message codé qu'il envoie à Alice ?

6. Alice reçoit le message M' de la part de Bob. Quel calcul effectue-t-elle pour retrouver M ?

On choisi $p = 1009$, $q = 1013$, $e = 13$ et $d = 78469$ et on admet que ces entiers vérifient les hypothèses.
Comme $pq > 10^6$, on se contentera de coder des messages d'au plus trois lettres avec ce système (pourquoi ?)

7. Écrire les fonctions permettant de crypter et décrypter.
On utilisera la fonction `pow(x,y,n)` au lieu de `(x**y) % n`, elle est beaucoup plus performante.

```
>>> crypte('bts')
744856

>>> decrypte(744856)
'bts'
```

8. Décrypter « 289380 »

En choisissant deux nombres premiers supérieurs à 100 000, nous allons créer un système RSA permettant de crypter des messages de 5 lettres.

9. Écrire une fonction **premier(n)** qui renvoie True si n est premier, False sinon.

10. L'algorithme suivant permet de calculer le PGCD de deux entiers naturels a et b (algorithme d'Euclide).

```
Tant que b > 0
  (a, b) ← (a, a%b)
Retourner a
```

Écrire une fonction **premier_entre_eux(a,b)** qui renvoie True si a et b sont premiers entre deux, False sinon.

La fonction suivante, exécutée avec
$a = e$ et $b = (p-1)(q-1)$,
calcule d une fois p, q et e choisis.

```
def invmod(a,b):
      if b%a==0:
         return 1
      else:
       return b - invmod(b%a,a)*b//a
```

11. Utiliser les fonctions précédentes pour créer un système RSA permettant de crypter des messages à 5 lettres.

Saut de puces

Une puce est positionnée sur l'axe des réels en 0. Elle effectue des sauts : le premier de longueur 1, le deuxième de 2, etc. À chaque saut, elle peut aller vers la droite, ou vers la gauche, mais à condition de ne pas tomber dans les négatifs.
Étant donné un entier naturel n, déterminer les moyens les plus rapides d'arriver sur n.
Par exemple,

- $3 = 1 + 2$
- $4 = 1 + 2 - 3 + 4$
- $8 = 1 + 2 + 3 - 4 + 5 - 6 + 7$
  $= 1 + 2 - 3 + 4 + 5 + 6 - 7$

```
>>> dec(11)
[[1, 2, 3, 4, -5, 6]]
```

Quelle est le plus petit entier atteignable d'au moins 10 façons ?