Lycée Charlemagne

.2018.– *MPSI2* –.2019.

PYTHON ET MATHS

PYTHON

Que va afficher l'exécution du script suivant :

```
a, b, c = 0, 1, 2
for k in range(3) :
....a, b, c = b, c, b
print(a, b, c)
```

Le script commence par affecter des valeurs à `a`, `b` et `c` en simultané.

D'entrée, `a` vaut `0`, `b` vaut `1` et `c` vaut `2`.

Ensuite, on fait une boucle où le nombre `k` sert juste de compteur. On va effectuer trois fois une boucle qui modifie en simultanée les valeurs. On s'attendrait à `a, b, c = b, c, a` qui fait tourner les trois valeurs. Mais c'est pire, avec `a, b, c = b, c, b`. La valeur a est perdue !

On va suivre pas à pas les variables

k	a	b	c
avant	0	1	2
0	1	2	1
1	2	1	2
2	1	2	1

L'ordinateur affiche ensuite a, b et c, c'est à dire `1, 2, 1`

On définit pour a et b entiers naturels : $a \boxdot b = \left\{ \begin{array}{ll} a+b & si \quad a \times b \ pair \\ a \times b & si \quad a \times b \ impair \end{array} \right|$. Montrez que cette loi est interne sur $\mathbb{N}$, commutative. Est elle associative ? A-t-elle un élement neutre ? Si oui, quels éléments ont un symétrique ? Si non, calculez $2 \boxdot 2 \boxdot 2 \ldots \boxdot 2$ *(n fois)*.

Écrivez une procédure `Python` qui prend en entrée deux entiers `a` et `b` et retourne l'entier `a⊡b`.

Dès lors que a et b sont dans $\mathbb{Z}$, les deux nombres $a+b$ et $a \times b$ y sont aussi.

Qu'on prenne l'un ou l'autre en fonction d'un critère sur $a \times b$, $a \boxdot b$ est à son tour un entier relatif. La loi est interne.

Si on échange les rôles de a et b, la condition reste la même, et le résultat aussi, cette loi est commutative.

Pour l'associativité, on peut tester différents triplets. Si l'un d'entre eux plante, on aura un contre-exemple. Si tous conviennent, on se dit que c'est bien parti pour que la loi soit associative, et on tente une démonstration en étudiant les différentes parités possibles pour a, b et c (puisque le critère est en fait $a \boxdot b = \left\{ \begin{array}{ll} a+b & si \quad a \ ou \ b \ pair \\ a \times b & si \quad a \ et \ b \ impairs \end{array} \right|$).

On voit que si les trois sont pairs, on a $(a \boxdot b) \boxdot c = a+b+c = a \boxdot (b \boxdot c)$.

Si les trois sont impairs, on a $(a \boxdot b) \boxdot c = a \times b \times c = a \boxdot (b \boxdot c)$.

Mais ces deux lignes ne prouvent rien.

Le problème est si a est pair, mais pas b ni c.

On regarde un exemple *(qui va acquérir le statut de **contre-exemple**)* : $a=2$, $b=1$ et $c=3$:

$2 \boxdot 1 = 2+1$	$(2 \boxdot 1) \boxdot 3 = 3 \boxdot 3 = 3.3$	$(1 \boxdot 3) = 1.3$	$2 \boxdot (1 \boxdot 3) = 2 \boxdot 3 = 5$
car 2.1 pair	car 3.3 impair	car 1.3 impair	car 2.3 pair

Il n'y a pas égalité.

L'entier 0 est neutre. En effet, que a soit pair ou impair, le produit $a.0$ est pair *(il vaut 0)* et la définition donne $a \boxdot 0 = a+0 = a$.

On se donne à présent a et on veut $a \boxdot b = 0$ *(le neutre)*.

Si a est pair, $a \boxdot b$ vaut $a+b$ quoi qu'y fasse b. On prend alors $b=-a$ et c'est gagné.
On a par exemple et pour comprendre : $6 \boxdot (-6) = 6+(-6) = 0$.
Les entiers pairs ***(y compris évidemment*** 0***)*** **ont tous un symétrique.**
Si a est impair, tout dépend de b. Si b est pair, l'entier $a \boxdot b$ est $a+b$ qui est impair et ne peut pas être nul. Si b est impair, l'entier $a \boxdot b$ est $a \times b$ qui est impair et ne peut pas être nul.
Bref, **aucun entier impair n'a de symétrique.** Tant pis pour eux, ça leur apprendra.

Si vous avez mal répondu à la question "*y a-t-il un neutre*", vous avez calculé $2 \boxdot 2 \boxdot 2 \ldots \boxdot 2$. Comme 2×2 est pair, on a $2 \boxdot 2$ vaut 4, qui est pair. On poursuit $(2 \boxdot 2) \boxdot 2 = 4 \boxdot 2 = 6$. On poursuit autant de fois qu'il faut avec des nombres tous pairs : $\boxed{2 \boxdot 2 \boxdot 2 \ldots \boxdot 2 = 2.n}$

On définit une procédure. Le test de parité d'un entier `N` est `N%2 == 0`.
En effet, il suffit de tester le nombre après réduction modulo 2.

```
def loi(a, b) :
....produit = a*b
....if produit % 2 == 0 :
........c = a+b
....else :
........c = a*b
....return(c)
```

Si `L` est une liste, on peut la lire dans le sens direct : `L[k]` est l'élément d'indice k (et non pas le k^{ieme}, on commence à 0, bien prendre l'habiude de dire « élément d'indice k »). Que donne `L[len(L)]` ?
On peut aussi lire la liste à partir de la fin : `L[-1]` est le dernier élément de la liste. Que donne `L[-len(L)]` ?

Que va donner
.
.

```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
for k in range(len(L)) :
....print(L[-k])
```

`L[len(L)]`est une erreur. Il n'y a pas d'élément d'indice `len(L)`.
Un exemple : `L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']`

k	0	1	2	3	4	5	6	7	8
L[k]	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	out of range

On peut donc aussi lire la liste à partir de la fin, de l'indice `-1` à l'indice `-len(L)` *(inclus)*.
`L[-len(L)]` fait retomber sur l'élément d'indice 0, le premier de la liste.

k	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8
L[k]	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	?

Comparons deux affichages

sens direct		sens rétrograde raté	sens rétrograde réussi
`for k in range len(L) : ....print(L[k])`	len(L)=8	`for k in range len(L) : ....print(L[-k])`	`for k in range len(L) : ....print(L[-1-k])`
'a'	k=0	'a'	'h'
'b'	k=1	'h'	'g'
'c'	k=2	'g'	'f'
'd'	k=3	'f'	'e'
'e'	k=4	'e'	'd'
'f'	k=5	'd'	'c'
'g'	k=6	'c'	'b'
'h'	k=7	'b'	'a'

Pour un affichage rétrograde réussi, la bonne instruction est

```
for k in range(len(L)) :
....print(L[-1-k])
```

L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
for k in range(len(L)) :
....L[k] = L[-k]
print(L)
Ca donne quoi ?

Réponse : `['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']`

Allez, on justifie. La boucle est impérative, et effectuée `len(L)` fois, c'est à dire 8 fois. `k` va prendre les valeurs `0`, `1`, `2`, `3`, `4`, `5`, `6` et `7` (on en a bien 8).
Et la liste est modifiée au fur et à mesure.
`L[-k]` va chercher l'élément d'indice `-k` *(on lit la liste à partir de la fin, `L[-1]` est le dernier élément, `L[-2]` est l'avant dernier).*
Petit détail : L[-0] est quand même le premier élément de la liste.
On prend donc l'élément d'indice `-k` de la liste *(dans son état actuel)*, et on l'affecte en position `k`.
Et on passe au `k` suivant.
Regardons l'exécution pas à pas :

k	L	L[-k]	L[k] reçoit L[-k]	
0	**['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']**	`'a'`	**['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']**	sans effet
1	**['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']**	`'h'`	**['a', 'h', 'c', 'd', 'e', 'f', 'g', 'h']**	'b' est perdu
2	**['a', 'h', 'c', 'd', 'e', 'f', 'g', 'h']**	`'g'`	**['a', 'h', 'g', 'd', 'e', 'f', 'g', 'h']**	'c' est perdu
3	**['a', 'h', 'g', 'd', 'e', 'f', 'g', 'h']**	`'f'`	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	'd' est perdu
4	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	`'e'`	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	sans utilité
5	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	`'f'`	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	'f' écrase 'f'
6	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	`'g'`	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	c'était déjà un 'g'
7	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	`'h'`	**['a', 'h', 'g', 'f', 'e', 'f', 'g', 'h']**	

On affiche `L`. C'est bien cette liste, dans laquelle des éléments ont été perdus.
Pour renverser la liste, il fallait faire autre chose !

Je crois que $832! + 1$ est un mutliple de 2017. Écrivez un script Python qui va le prouver.
Mon fils ne sait pas si $8321! + 1$ est un multiple de 2017. Mais vous, vous avez la réponse ! sans ordinateur.

On doit calculer 832!+1. Python réussit à manipuler des entiers aussi grands qu'on veut. On va calculer cette factorielle par une boucle `for`. Attention, on rappelle que `range(832)` donne tous les entiers de `0` à `831` (et il y en a 832). Et nous on veut les entiers de `1` à `832`. On va ajouter `1`.

```
facto =1 #initialisation
for k in range(832) :
....facto = facto*(k+1) #ou facto*=k+1
```

Le nombre `facto` est alors égal à 832!. On lui ajoute 1 :

```
facto = facto+1 #ou facto+=1
```

Il reste à voir si il est multiple de 2017.
Le réflexe de calculette : on calcule le quotient et on regarde si il est entier. C'est gentil, mais ça ne fait de vous que des élèves de Terminale, et pas des scientifiques.
Le réflexe d'ordinateur : on fait une division euclidienne. Et on ne regarde que le reste :

```
facto%2017
```

Si ce nombre vaut bien 0, on a gagné.
Le test est même `(facto%2017)==0`
Juste pour le plaisir :

```
11716384800371466108976291641693592665628440569160942802854221347977634631408509362753955052967459014157191814117612417331431770578801961043507494544512090799281926985036103443873445298164185559368135622237957572567445696798578264002036284682224929503321104605824016830686012037055107889268176163660991491776358133848514058571074515804007550961749039044956177541713865534264563703297341416217263895854112029377041
```

```
3228471996800859740889341995263741363489459333634160843117861153143276957863788867044674286666529955900
9582104913569493786419393786839599809753016138673856698435821666260815251952831092259211396155606782225
7697888789863310108839132508926551902009571329487009556307452202945350052314349410659070729478751142091
5473117791699102626045194241305979933982726271751566091774051063484171238053349727417633432155777602902
2607985909868300407486910722672281095164513265754822917512037255360427712542315025902251857998392217911
8948666173918746879823468426122078224960077147884676644191202380752708192460738839235891814831844122355
5675173631218118388208628551818236999678629353324948595320523531898310401914632348143450691445927371471
4549964066263757574388841193083568619686783442637483474476153947114020459837892322788376902251043152981
3123218811262444068246469343389301810439265146345904383128480101605776131066574016466653731009370667208
0247123384533204626155443685733814339787310418700409184416162820298896987759180684847173969539596889938
7099181793908498374047799614002480390160824619520701271132276558433712169402901918519540961148853673320
6434013000040217637798407013918063333392624854093723221837501794996597262894273654112964598410916108115
7390540527420594240237975978261020909762020899014241283871908382936951783778751303364348154972322903847
8552249157152046595115475209161616576378818766630811101928297021498377494951297017308077351840769654341
7858051276800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000001
```
C'est bon, i n'y a plus qu'à poser la division.

Mais qu'en est il de $8321! + 1$?
Le nombre 8321! est un produit d'entiers parmi lesquels il y a 2017. C'est donc un multiple de 2017.
On lui ajoute 1, c'est fini ; ce n'est plus un multiple de 2017.

L'algorithme de tri `QuickSort` a besoin de la procédure suivante sur une liste `L` :
prendre le premier élément de la liste, noté `a` et créer deux listes :
- `Lbas` : les éléments de `L` plus petits que `a`
- `Lhaut` : les éléments de `L` plus grands *(strictement)* que `a`

(on ne reprend pas `L[0]` *dans ces listes).* Écrivez cette procédure.

On supposera que la liste `L` n'est pas vide, afin de pouvoir prendre son premier élément `L[0]` :

```
def coupe(L) : #on déclare
....a = L[0] #on prend l'élément initial
....Lbas, Lhaut = [], [] #on initialise deux listes vides
....for element in L[1 :] : #on ne regarde L qu'à partir de la position 1
........if element > a : #on teste
............Lhaut.append(element) #on ajoute à la bonne liste
........else :
............Lbas.append(element)
....return(Lbas, a, Lhaut) #on retourne ce qu'il faut
```

La liste `Temp` est de longueur 365 car l'année ne fut pas bisextile. `Temp[n]` indique la température qu'il faisait le jour d'indice `n` de l'année 2016 à 8 heure du matin dans la cour du lycée.
Écrivez une procédure qui indique combien de jours dans l'année la température à huit heure a dépassé 17 degrés.
Affinez la pour qu'elle vous donne la liste des jours où la température a dépassé 17 degrés.
Et si vous n'avez que ça à faire, débrouillez vous pour que ce soit un affichage "quantième et mois".

On va explorer les différents termes de la liste, par un
`for k in range(365) : lire Temp[k]` *ou un* `for temp_du_jour in Temp` *: si on est informaticien.*
Pour chacun, on va regarder si la température a dépassé `18`. dans ce cas, on va incrémenter un `compteur` (qu'on aura initialisé à `0`) :

```
compteur = 0
for k in range(365) :
....if Temp[k] > 18 :
........compteur+=1
print(compteur)
```

```
compteur = 0
for tdj in Temp :
....if tdj > 18 :
........compteur+=1
print(compteur)
```

```
compteur = 0
for tdj in Temp :
....compteur+=(tdj > 18)
print(compteur)
```

Dans la version `compteur+=(temp_du_jour > 18)`, on profite de ce qu'est `(temp_du_jour > 18)` pour `Python`. C'est un `booléen`, qui vaut `True` ou `False`. Mais `True` vaut 1 et `False` vaut 0. On ajoute donc 1 ou 0 au `compteur`, selon que la température a ou non dépassé 18. Astucieux.

On évitera le gentillet

```
for k in range(365) :
....if Temp[k] > 18 :
........compteur=compteur+1
....else :
........compteur = compteur
```

Ce n'est pas faux, mais c'est quand même une façon tordue de dire à `Python` *"ne fais rien". Pour moi, c'est le signe d'une forme d'esprit que j'aime assez*[a].

a. car je suis bien conscient qu'il faut aussi former des physiciens et pas seulement des informaticiens

---

Si on veut la liste des dates, on crée une liste vide au départ, dans laquelle on colle les dates qui nous plaisent.
On n'a en effet plus besoin de la variable compteur, il suffit de mesurer la longueur de la liste.
Pour extraire la date JJ/MM à partir de l'indice du jour de l'année, il faut un programme qui ne sera pas détaillé ici.

```
Dates_chaleur = []
for k in range(365) :
....if Temp[k] > 18 :
........Date_chaleur.append(k)
print(len(Date_chaleur), Date_Chaleur)
```

On exécute :
.
Que lirez vous ?
.
.
.

```
L=[k for k in range(11)]
while len(L) > 1 :
....a = L.pop(0)
....b = L.pop(0)
....L.append(b)
print(L)
```

On initialise une liste. Etape par étape on en enlève des éléments par la méthode pop. Avec l'instruction `L.pop(0)`, on enlève le premier élément de la liste.
Plus précisément avec `a = L.pop(0)`, on le sort, et on le perd puisque nulle par on n'exploite `a`. Avec `b = L.pop(0)`, on prend le suivant. Et enfin, avec `L.append(b)`, on va le recoller en fin de liste.
C'est finalement une histoire de file d'attente un peu spéciale.
Vingt individus numérotés de 0 à 10 font la queue devant la cantine/un cinéma/une caisse de supermarché/le père Noel...
Un vigile laisse passer une personne sur deux (a) et renvoie la suivante en fin de file.
Il finira par revenir en tête de liste, et se fera peut être à nouveau refouler.
On se demande qui seront les derniers à sortir.
On étudie pas à pas :

`L=[0,1,2,3,4,5,6,7,8,9,10]`
0 sort et 1 va en fin de file `L=[2,3,4,5,6,7,8,9,10,1]`
2 sort et 3 va en fin de file `L=[4,5,6,7,8,9,10,1,3]`
4 sort et 5 va en fin de file `L=[6,7,8,9,10,1,3,5]`
jusqu'à 8 sort et 9 va en fin de file `L=[10,1,3,5,7,9]`
10 sort et 1 retourne en fin de file `L=[3,5,7,9,1]`
3 sort enfin et 5 retourne en fin de file `L=[7,9,1,5]`
7 sort enfin et 9 retourne en fin de file `L=[1,5,9]`
1 sort enfin et 5 retourne en fin de file `L=[9,5]`

Le processus s'arrête. On note que 5 sera le dernier à sortir, mais le programme affiche `[9, 5]`

La suite de Fiboonacci est définie par "F_0 et F_1 donnés dans $\mathbb{N}$" et $F_{n+2} = (F_{n+1} + F_n)\ mod\ 100$.
Écrivez un script `Python` qui prend en entrée les deux premiers termes puis qui calcule et affiche les termes de la suite, jusqu'à ce qu'il y ait un terme en double *(en l'occurence ici pour* 1 *et* 2*, ici ce sera* `[1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 44, 33, 77, 10, 87, 97, 84, 81, 65, 46, 11, 57, 68, 25, 93, 18]` *car après, c'est...).*
Pourquoi votre programme est il sûr de s'arrêter ?

On va écrire une procédure (`def ...() :` jusqu'à `return(...)`), quitte à ensuite l'utiliser avec des `a = int(input('Entrez le premier entier'))` qui se ra suivi d'un `print(Fiboo(a,b))`.
On va utiliser une liste qu'on va grandir peu à peu. Pour aller chercher les deux derniers termes de la liste : `L[-1] et L[-2]`. C'est ainsi que pour la suite de Fibonacci, on peut utiliser

```
L=[0, 1]
for k in range(n) :
....L.append(L[-1]+L[-2])
```

plus concis que

```
a, b, L = 0, 1, [0, 1]
for k in range(n) :
....a, b = b, a+b
....L.append(b)
```

Ici, on va même faire une boucle `while` avec un argument de type négatif "ne pas être dans la liste" :
Dans cette version, on sort avant d'avoir un terme en double.
On pourra faire un test initial `a!=b` pour éviter d'avoir des suites un peu idiotes comme `[2, 2]` qui s'arrête tout de suite.

```
def Fiboo(a, b) :
....L = [a, b]
....x = a+b
....while not(x) in L) :
........L.append(x)
........x = (L[-1]+L[-2])%100 #c'est le modulo
....return(L)
```

Pourquoi le script s'arrête ? Les valeurs prises par la suite modulo `100` sont des entiers entre `0` et `99`.
On ne peut pas avoir une infinité d'entiers distincts entre `0` et `99`.
La suite ne peut pas avoir plus de cent termes distincts. Elle s'arrête donc avant le rang `100`.
C'est avec la graine `0, 74` *qu'on a la suite la plus longue.*
`[1, 74, 75, 49, 24, 73, 97, 70, 67, 37, 4, 41, 45, 86, 31, 17, 48, 65, 13, 78, 91, 69, 60, 29, 89, 18, 7, 25, 32, 57]`

Le même type de raisonnement sur les couples formés de deux termes consécutifs permet de montrer que la suite de Fibonacci modulo 100 *(ou modulu tout entier) est périodique à partir d'un cerain rang.*

Écrivez un script `Python` qui pour `n` et `p` donnés indique combien de termes de la ligne d'indice `n` du triangle de Pascal sont des multiples de `p`.

On reçoit `n` et `p`. On va calculer un par un les coefficients de la ligne, en utilisant la relation $\binom{n}{k+1} = \frac{n-k}{k+1}.\binom{n}{k}$ qu'on applique sans cesse :

	$\times n$		$\times(n-1)$		$\times(n-2)$		$\times(n-3)$
1		n		$\frac{n.(n-1)}{2}$		$\frac{n.(n-1).(n-2)}{6}$	...
	$\div 1$		$\div 2$		$\div 3$		$\div 4$

On va obtenir des entiers, et à chaque fois, on va diviser par `p`. Si le reste est nul, on enregistre. C'est un test `(binomial%p)==0`.

```
def likripou(n,p) :
....binomial = 1
....L = []
....for k in range(n+1) :
........binomial=binomial*(n-k)/(k+1)
........if (binomial%p)==0 :
............L.append(binomial)
....return(len(L))
```

Ici, on crée inutilement une liste `L`, mais ce n'est pas si mal d'avoir les nombres.
Mais on pourra se contenter de

```
def likripou(n,p) :
....binomial, compt = 1, 0
....for k in range(n+1) :
........binomial=binomial*(n-k)/(k+1)
........compt += ((binomial%p)==0)
....return(compt)
```

en profitant de ce qu'un booléen comme `(binomial%p)==0` vaut 0 ou 1.
Les plus futés s'arrêteront en milieu de ligne et multiplieront par 2 *(en prenant garde de ne pas le faire si on est sur le terme pivot du milieu de la ligne).*

*Soyons franc, les élèves qui auront crée une foncion factorielle et utiliseront binomial=fact(n)/(fact(k)*farct(n-k)) ne pourront pas prétendre avoir les points. Ils seront assimilés à des gentils "programmeurs au pied de la lettre", mais ni à des matheux ni à des informticiens.*

```
def Mystere(n) :
....N, S = n, 0
....while N != 0 :
........c, N = N%10, N/10
........S = 10*S+c
....return(abs(n-S))
```

```
def Mimystere(a) :
....L = [a]
....while a != 0 :
........a = Mystere(a)
........L.append(a)
....return(L)
```

Que fait `Mystere` ?
Que va donner `Mimystere(2017)` ?
Que va donner `Mimystere(18)` ?

Le premier script prend en entrée un entier `n` (il y a des modulo et divisions, donc on travaille sur des entiers). Il en fait une copie `N` qu'il va fare fondre `N=N/10` Il va en extraire à chaque fois le chiffre des unités jusqu'à ce qu'il ne reste plus rien

```
while N != 0 :
....c = N%10
....N = N/10
```

Mais il fait remonter dans un certain `S` les chiffres en question.
Le premier chiffre sorti de `N` est le chiffre des unités de `n`. Peu à peu, il est multiplié par 10, il monte et reste le chiffre de tête de `S`.
Finalement, `S` crée une sorte de copie de `n`, mais "lue à l'envers".
C'est ce qu'on peut appeler le renversé de `n`.
Prenons un exemple :

N	test	c	N	S
2017				0
2017	continue	7	201	7
201	continue	1	20	71
20	continue	0	2	710
2	continue	2	0	7102
0	stop			7102

On effectue la différence n-S du nombre et de son renversé et c'est ce qu'on retourne. On en prend la valeur absolue. Par exemple `2017 -> 7102-2017=5085`

Ensuite, on prend un nombre `a`, qu'on met dans une liste `L` *(on va agrandir cette liste au fur et à mesure).* On le remplace par son `mystère` *(valeur absolue de a moins son renversé).* Et on continue jusqu'à ce qu'on tombe sur 0.

a	revense(n)	mystere(a)	L
2017	7102	5085	[2017,5085]
5085	5805	720	[2017,5085,720]
720	27	693	[2017,5085,720,693]
693	396	297	[2017,5085,720,693,297]
297	792	495	[2017,5085,720,693,297,495]
495	594	99	[2017,5085,720,693,297,495,99]
99	99	0	[2017, 5085, 720, 693, 297, 495, 99, 0]

On fait de même avec 18

a	revense(n)	mystere(a)	L
18	81	63	[18,63]
63	36	27	[18, 63, 27]
27	72	45	[18, 63, 27, 45]
45	54	9	[18, 63, 27, 45, 9]
9	9	0	[18, 63, 27, 45, 9, 0]

Une liste `Deci` contient les N premières décimales de π. Que fait ce script :
.
.
.
.

```
for k in range(N-5) :
....c = 0
....for r in range(1,6) :
........if Deci[k+r] == Deci[k] :
............c +=1
....if c == 5 :
........print(k,Deci[k, k+6])
```

On va parcourir la liste `Deci` case par case, peut être pas jusqu'au bout.
Pour chaque `k`, on prend l'élément `Deci[k]`, et on initialise un compteur `c` à `0`.
On lit alors les cinq termes suivants de la liste `for r in range(1, 5) : Deci[k+r]` Pour chacun de ceux qui est égal à `Deci[k]`, on incrémete le compteur de une unité `c+=1`
Au final, si ce compteur `c` vaut `5`, c'est que chaque `Deci[k+r]` est égal à `Deci[k]`. C'est donc que les six décimales `Deci[k]`, `Deci[k+1]`, `Deci[k+2]`, `Deci[k+3]`, `Deci[k+4]` et `Deci[k+5]` sont égales.
Dans ce cas, on affiche `k` et la sous-liste `L[k : k+6]` faite de ces six nombres.
On cherche donc si dans la liste L il y a six chiffres consécutifs égaux.
Pour information, à la 762^{eme} décimale, on croise le motif `99999`.
à la 19 446^{eme} on croise encore `99999`
à la 24 446^{eme} on croise cette fois `55555`.

Ce script `Python` étudie une suite récurrente sur le modèle de celle de Stern (avec ou sans Brocott) :
Que donneront

`recurrente(5,0,10)`
`recurrente(0,0,200)`
`recurrente(1,1,20)`

J'ai trouvé $a_5 = 6$ et $a_6 = 2$ retrouvez a_1 et a_0.

```
def recurrente(a,b,n) :
....L = [a,b]
....for k in range (2,n+1) :
........if k%2 == 0 :
............c = 2*L[k/2]
........else :
............c = L[k/2]-L[k/2-1]
........L.append(c)
....return(L)
```

La procédure crée une liste qu'elle initialise aux deux nombres a et b. Ensuite, elle construit pas à pas une suite, du terme d'indice 2 au terme d'indice n *(le range $n+1$ s'arrête à n)*. La suite est définie par $a_k = 2.a_{k/2}$ si k pair et $a_k = a_{k/2} - a_{k/2-1}$ si k impair. On donne ensuite tous les termes de la suite.
Il est plus aisé de distinguer les cas en écrivant $k = 2.p$ ou $k = 2.p+1$

$a_{2.p} = 2.a_p$	$a_{2.p+1} = a_p - a_{p-1}$

Une récurrence évident dit que si les deux premiers termes de la liste sont nuls, tous le sont.
On intialise ensuite à 5 et 0 :

k	0	1	2	3	4	5	6	7	8	9	10
formule	5	0	$2.a_1$	$a_1 - a_0$	$2.a_2$	$a_2 - a_1$	$2.a_3$	$a_3 - a_2$	$2.a_4$	$a_4 - a_3$	$2.a_5$
valeur			0	-5	0	0	-10	-5	0	5	0

`[5, 0, 0, -5, 0, 0, -10, -5, 0, 5,0]`

On initialise à 1 et 1 :

k	0	1	2	3	4	5	6	7	8	9	10
formule	1	1	$2.a_1$	$a_1 - a_0$	$2.a_2$	$a_2 - a_1$	$2.a_3$	$a_3 - a_2$	$2.a_4$	$a_4 - a_3$	$2.a_5$
valeur			2	0	4	1	0	-2	8	4	2

k	11	12	13	14	15	16	17	18	19	20
formule	$a_5 - a_4$	$2.a_6$	$a_6 - a_5$	$2.a_7$	$a_7 - a_6$	$2.a_8$	$a_8 - a_7$	$2.a_9$	$a_9 - a_8$	$2.a_{10}$
valeur	-3	0	-1	-4	-2	16	10	8	-4	4

`[1, 1, 2, 0, 4, 1, 0, -2, 8, 4, 2, -3, 0, -1, -4, -2, 16, 10, 8, -4]`

On reprend nos formules et nos résultats : $a_5 = 6$ et $a_6 = 2$ donc $a_2 - a_1 = 6$ et $a_3 = 1$. Or, $a_2 = 2.a_1$.
On a donc $a_1 = 6$. On remonte : $a_1 - a_0 = a_3 = 1$, donc $a_0 = 5$. On résume le début de la suite :

n	0	1	2	3	4	5	6
a_n	5	6	12	1	24	6	2

Que fait ce script ? Trouvez une liste de longueur `10` pour lequel la réponse sera `True`.

```
def mystere(L) :
....n = len(L)
....delta = n/2
....for k in range(n-1) :
........if abs(L[k+1]-L[k]) < delta :
............return(False, k)
....L.sort() #tri de la liste
....if L != range(n) :
........return(False, 'tricheur')
...return(True)
```

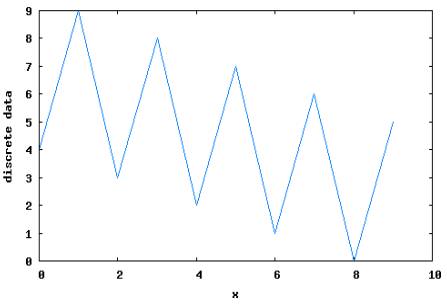
La procédure prend une liste, en mesure la longueur et même la demi longueur `n = len(L)` `delta = n/2` . Ensuite, elle prend les éléments un à un, jusqu'à l'avant dernier `for k in range(n-1) :` *(car on va comparer chaque terme au suivant).* On regarde la différence entre un terme et le suivant. Si elle est plus petite que `delta`, on sort abruptement avec la réponse `False` *(plus l'indice* `k` *du terme posant problème)*
Si on a passé cette boucle sans sortir, c'est que toutes les différences $|L[k+1] - L[k]|$ sont "grandes". Mais on n'a pas gagné. Il reste le test

```
L.sort()
if L != range(n) :
...return(False, '...')
```

on trie la liste et on la compare à un range $[0, 1, \dots n-1]$. Si il n'y a pas égalité, on sort *(avec* `False` *et une insulte).*
Comment peut il y avoir égalité ? C'est que les éléments de `L` sont les entiers de 0 à $n-1$. Bref, pour sortir par le `return(True)` tout au bout, il n'y a qu'une solution :

la liste `L` est une permutation de la liste de `0` à `n-1`, dans laquelle les écarts $|L_{k+1} - L_k|$ sont plus grands que $n/2$.

On trouve une solution avec une permutation de `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`. Mais il faut que les différences soient au moins égales à `5`. Par exemple `[0, 15, 30, 20, 12, 6, 1, 17, 23]` est convenable sur les écarts mais pas sur les termes présents. En revanche, `[6, 1, 9, 3, 4, 5, 0, 7, 8, 2]` commence bien, mais plante dès `k=3`.
On peut proposer `[4, 9, 3, 8, 2, 7, 1, 6, 0, 5]`



Que fait cette procédure :

```
def Harmo(a, epsi) :
....S = 0
....L = []
....k=1.
....sens = a>0
....while abs(S-a)>epsi :
........while sens*(S-a)<0 :
............L.append(sens)
............S += sens/k
............k+=1
............#print(S)
........sens *=-1
....return(L)
```

Pourquoi ça marche ? Que donne-t-elle pour $a = \ln(2)$.
Pourquoi est ce qu'une procédure écrite en maths ne commence pas par `from physic import *` alors que toute procédure écrite en physique commence par `from math import *` ? Quel serait le risque si on rempla `S += sens/k` par `S += sens/k**2` ou `S += sens/k*k` ?
.
.
.

Laissé au lecteur.

Simplifiez $\sum_{k=-n}^{n} k^2$ et calculez $\sum_{\substack{0\leqslant k\leqslant 2016\\ k\ impair}} k^3$. Montrez que la somme $\sum_{\substack{0\leqslant k\leqslant 2016\\ k\ impair}} k^3$ est divisible par 1008^2.
N'ayant pas confiance dans votre calcul, écrivez une procédure `Python` qui prend en entrée N et retourne $\sum_{\substack{0\leqslant k\leqslant N\\ k\ impair}} k^3$. [a]

a. une "procédure qui prend en entrée N" c'est `def nom_fonction(N) :` et pas un programme gentillet qui fait `N=int(input('Veuillez vous essyer les pieds et me donner un nombre'))` ; on programme pour mettre en commun, et pas pour juste dire à son voisin, "tiens, tiens, je vais te montrer un truc"

La première est une question coeur : $\sum_{k=-n}^{n} k^2 = 2.\sum_{k=0}^{n} k^2$ car chaque terme est présent deux fois : $(-k)^2 = k^2$. On simplifie $\boxed{\sum_{k=-n}^{n} k^2 = \frac{n.(n+1).(2.n+1)}{3}}$

Pour $\sum_{\substack{0\leqslant k\leqslant 2016\\ k\ impair}} k^3$, on a deux approches. On dit que k impair s'écrit $k = 2.p+1$ avec p pouvant aller de 0 *(pour $k=1$)* à 1007 *(pour $k=2015$)*. On somme alors $\sum_{p=0}^{1007}(2.p+1)^3$ qu'on développe en $8.\sum_{p=0}^{1007} p^3 + 12.\sum_{p=0}^{1007} p^2 + 6.\sum_{p=0}^{1007} p + \sum_{p=0}^{1007} p$.
On se contente de $2.N^2.(N+1)^2 + 2.N.(N+1).(2.N+1) + 3.N.(N+1) + N + 1$ avec $N = 1007$ *(la fin du travail, vous la confiez à votre technicien).*

On peut aussi lui adjoindre la somme $\sum_{\substack{0\leqslant k\leqslant 2016\\ k\ pair}} k^3$ qui n'est autre que $\sum_{p=0}^{1008}(2.p)^3$ c'est à dire $2.\frac{(1008.1009)^2}{4}$. En les sommant, on récupère $\sum_{k=0}^{2016} k^3$ c'est à dire $\frac{(2016.2017)^2}{4}$. On soustrait : $\boxed{1008^2.(2017)^2 - 2.1008^2.(1009)^2}$ Tous calculs *(idiots)* faits : 2 064 771 088 128

Sous cette forme, on voit que ce nombre est bien factorisable par 1008^2.

Pour le script `Python` on va cumuler dans une variable `S`, et avancer

soit de deux en deux

```
def Somme(N) :
....S = 0
....k = 1
....while k < N :
........S = S + k*k*k
........k = k+2
....return(S)
```

soit par boucle `for` précalculée

```
def Somme(N) :
....S = 0
....maxi = int(N/2)
....for p in range(maxi) :
........S = S+(2*p+1)**3
....return(S)
```

Petit test amusant à faire :
comparez la durée d'exécution des deux programmes suivants :

```
def Somme(N) :
....S = 0
....maxi = int(N/2)
....for p in range(maxi) :
........S = S+(2*p+1)**3
....return(S)
```

```
def Somme(N) :
....S = 0
....maxi = int(N/2)
....for p in range(maxi) :
........Nb = 2*p+1
........S = S+Nb*Nb*Nb
....return(S)
```

Il n'y aura pas photo, le programme qui calcule le cube de `a` par `a*a*a` plutôt que `a**3` va plus vite.

On tape
.
.
.
.
Que va donner
:

```
couleur=['pique', 'coeur', 'trefle', 'carreau']
hauteur=['as']+[str(k) for k range(2, 11)]+['valeur',
'dame, 'roi']paquet = [ ]
for i in range(13) :
....for j in range(4) :
........paquet.append(hauteur[i]+' de '+couleur[j])
```

`print(hauteur)`	`print(len(paquet))`	`print(paquet[12])`	`print(paquet[20])`
`print(paquet[52])`	`print(paquet[-3])`	`print(paquet[8 :12])`	

Que fait
.

```
c = int(input('Entrez une valeur entre 1 et 52'))
paquet = paquet[c :]+paquet[ :c]
```

On a crée deux listes de mots, et on va puiser dedans pour créer des chaines de caractère par concaténation (addition) :
`hauteur[i] + ' de ' + couleur[j]`
La liste hauteur est faite de trois listes collées bout à bout :
• une liste très courte `['as']` : `L[0]` vaudra donc `'as'`
• une liste "*en compréhension*" comme elle dit Solène : `k` va de `2` à `10`, et pour chaque valeur de `k`, on a `str(k)`, c'est à dire `k` converti en mot : `['2', '3',...'10']`
• une liste de trois mots qui font penser vreiment à un jeu de cartes.

```
hauteur=['as', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'valet', 'dame', 'roi']
```

il ne leur manque que les couleurs ♠, ♢, ♡, ♣.
On crée une liste vide `L`, et on fait deux boucles imbriquées :
une boucle sur `i` de `0` à `12` ; et au sein de cette boucle, `j` va de `0` à `3`.
Pour `i` égal à `0`, `hauteur[i]` est le mot "`as`". On lui colle ensuite pour chaque valeur de `j` la petite chaine ' `de` ' puis chacune des quatre couleurs.
Avec `i=0` et le passage d'une boucle entière sur `j`, la liste `L` contient

`['as de pique', 'as de coeur', 'as de carreau', 'as de trefle']` avec bien des espaces de chaque côté du `de`.
On recommence ensuite de même avec `hauteur[1]`. On `append` quatre nouveaux termes : `'2 de pique'`, `'2 de coeur'`, `'2 de carreau'`, `'2 de trefle'`.
On continue ainsi pour chaque valeur de `i`.
Combien de termes a-t-on `append`é à la liste `L` initialement vide : quatre à chaque valeur de `i`. La liste `L` a `52 termes`, de `'as de pique'` à `'roi de carreau'`.
Qui est le terme d'indice `12` ? Le treizième.
On a épuisé la série des `as`, des `2`, des `3`, et on attaque tout juste la série des `4` : `'4 de pique'`
Le terme d'indice 20 a vu passer cinq séries de 4 ; on attaque les `6` avec `'6 de pique'`
Le terme d'indice 52 n'existe pas, puisque le liste a 52 termes. Réponse `"erreur"`
Le terme d'indice `-3` est à `3` de la fin, c'est un `roi`, l'avant avant dernier : `'roi de coeur'`

Au fait, dans votre réponse, vous avez pensé aux guillemets ? Python y a pensé, lui !

La découpe paquet[8 :12] donne les termes d'indices 8 à 11 (quatre termes en effet) :
`'3 de pique', '3 de coeur', '3 de trefle', '3 de carreau'`
`paquet(23)` n'a pas de sens, l'indice doit être entre crochets.
Après, on prend un indice `c` que tape l'utilisateur *(on en fait un entier).*
On récupère deux morceaux de la liste : tout de l'indice `c` à la fin, et tout jusqu'à l'indice `c`.
Le terme d'indice `c` sera dans `paquet[c :]` et pas dans `paquet[ :c]`.
On les colle dans le "mauvais ordre" : le paquet de la fin avant le paquet du début.
C'est ce qu'on appelle "couper un paquet de cartes", que ce soit pour jouer à la belote, ou pour faire un tour de magie.

Écrivez un script Python qui crée des listes de noms pour les meubles Ikea.

Cet exercice est laissé au lecteur.

Voici un script Python :
Quelle sera sa réponse pour `n` égal à `6` ? pour `n` égal à `12` ?
Que fait il en toute généralité ?
Rappel : `a/b` et `a%b` sont le quotient et le reste de la division euclidienne de `a` par `b`.

```
def entoibordel(n) :
....F, c = 1, 0
....for k in range(1, n+1) :
........F *= k
....while F > 0 :
........c += ((F%10)==0) #un booléen vaut 0 ou
1
........F = F/10
....return(c)
```

Dans la partie
```
....F, c = 1, 0
....for k in range(1, n+1) :
........F *= k
```
on reconnaît le calcul de `n !` que l'on place dans `F`. Ensuite, avec
```
.while F > 0 :
.....F = F/10
```
on fait "fondre" `F` en le divisant peu à peu par 10, jusqu'à ce qu'il ne reste plus rien.
Mais à chaque tout, on extrait `c += ((F%10)==0)` qu'on ajoute à un compteur `c` qu'on a initialisé à `0`. L'instruction `c += ((F%10)==0)` augmente c de une unité chaque fois que le chiffre des unités de `F` vaut `0`.
On compte donc le nombre de `0` dans l'écriture décimale de `n !`
Sachant `6 ! = 720`, la réponse sera `1`.
Sachant `12 ! = 479 001 600` (à la main), on trouve `4`.

Écrivez un script `Python` qui pour `N` donné (a priori grand) trouve le premier indice `n` vérifiant $\binom{\mathbf{2.n}}{\mathbf{n}} \geqslant \mathbf{N}$.

On va créer une boucle `while :` tant que `binomial` est plus petit que `N`.
Et à chaque étape on va calculer le nouveau binomial : $\binom{2.(n+1)}{n+1} = \frac{(2.n+2)!}{((n+1)!)^2} = \frac{(2.n)!.(2.n+1).(2.n+2)}{(n!.(n+1))^2} = \frac{4.n+2}{n+1}.\frac{(2.n)!}{(n!)^2} = \frac{4.n+2}{n+1}.\binom{2.n}{n}$.
On va donc initialiser la suite à la valeur 1 et faire avancer n.

```
def auconmaltais(N) :
....n = 0
....binomial = 1
....while binomial<N :
........binomial =
(4*n+2)*binomial/(n+1)
........n += 1
....return(n)
```

Si on est mauvais en programmation :

```
def factorielle(n) :
....f = 1
....for k in range(1,
n+1) :
........f *= k
....return(f)
```

```
def binomial(n,k) :
....B = factorielle(n)
....B /= factorielle(k)
....B /=
factorielle(n-k)
....return(B)
```

```
def aitiste(N) :
....n = 0
....while
binomial(2*n,n)<N :
........n += 1
....return(n)
```

Cette méthode est correcte, mais qu'est ce qu'elle va faire comme calculs inutiles...

Vous avez été 45 à voter. Le résultat des urnes est une liste de listes, nommée `Vote`. Chaque élément de `Vote` est le vote de l'un d'entre vous, tel que `['Hugo', 'Nathan']` ou `['Matéo']` ou même `[]`. Écrivez un script qui compte les votes blancs. Écrivez un script qui rend une liste `BonsBulletins` où l'on a éliminé les personnes ayant voté deux fois. Écrivez un script qui compte le nombre de voix pour `Liam` (attention, un bulletin `['Liam', 'Liam']` ne doit pas compter double !). Bonus : écrivez un script qui déclare qui de Nathan, Matéo, Hugo et Liam a le plus de voix. Super bonus : classez les quatre candidats.

```
blancs = 0
for bulletin in Vote :
....if len(bulletin) == 0 :
........blancs +=1
print('Il y a '+str(blancs)+' bulletins blancs.')
```

ou même

```
for bul in Vote :
....blanc += (len(bul)==0)
```

```
BonBulletins = []
for k in range(len(Vote)) :
....if len(Vote[k])<3 :
........BonsBulletins.append(Vote[k])
```

```
VoteLiam = 0
for bul in BonsBulletins :
....if bul[0]=='Liam' or bul[1]=='Liam' :
........VoteLiam +=1
print(str(VoteLiam)+' élève(s) ont voté pour Liam')
```

Et quand même pas `if bul[0] or bul[1] == 'Liam'`.
On compte les voix de chacun :

```
VoteLiam, VoteMateo, VoteNathan, VoteHugo = 0, 0, 0, 0
for bul in BonsBulletins :
....if bul[0]=='Liam' or bul[1]=='Liam' :
........VoteLiam +=1
if bul[0]=='Mateo' or bul[1]=='Mateo' :
........VoteMateo +=1
if bul[0]=='Hugo' or bul[1]=='Hugo' :
........VoteHugo +=1
if bul[0]=='Nathan' or bul[1]=='Nathan' :
........VoteNathan +=1
```

On crée deux listes :

```
Verdict = [VoteLiam, VoteMateo, VoteHugo, VoteNathan]
Noms = ['Liam', 'Mateo', 'Hugo', 'Nathan']
```

On cherche le maximum des voix, et l'indice :

```
MaxiVote, IndexWinner = 0, -1
for k in range(4) :
....if Verdict[k] > MaxiVote :
........MaxiVote = Verdict[k]
........IndexWinner = k
print('Le gagnant est '+Noms[IndexWinner]+' avec '+str(MaxiVote)+' voix')
```

On efface :

```
Verdict.pop(IndexWinner)
Noms.pop(IndexWinner)
```

On recommence :

```
MaxiVote2, IndexFollower = 0, -1
for k in range(3) :
....if Verdict[k] > MaxiVote2 :
........MaxiVote2 = Verdict[k]
........IndexFolower = k
print('Le deuxieme est '+Noms[IndexFollower]+' avec '+str(MaxiVote2)+' voix')
```

Et ainsi de suite.

Pour tout entier naturel n, on note $n\natural$ la fucktorielle de n. C'est quoi cette connerie ? C'est juste que quand on calcule $n!$, on effectue le produit des entiers de 1 à n, alors que pour $n\natural$, quand on tombe sur un multiple de 7, on le remplace par 2017. Ne me demandez pas pourquoi, c'est moi qui décide ! Par exemple, $9\natural = 1.2.3.4.5.6.2017.8.9$. Écrivez un script qui demande à n à l'utilisateur et lui calcule $n\natural$.
On note qu'on a $n\natural = n!$ pour des valeurs de n à préciser, mais qu'on a très vite $n\natural > n!$. Peut on espérer avoir $n\natural < n!$ à partir d'un certain rang ?

On écrit donc un script `Python` dans lequel on va initialiser un produit à `1` *(valeur qu'il y a avant de multiplier, et surtout pas `0`)*. On fait ensuite une boucle avec un `range`. On connaît les moeurs du `Python` : pour les entiers de `1` à `n` inclus, on doit donner `range(1, n+1)`. Ensuite, on fait un teste pour savoir si l'entier est multiple de `7`. Si oui, on multiplie par `2017` parce que c'est ce qu'on a décidé. Sinon, on multiplie par `k` lui même comme dans la factorielle.

```
n=int(input('Entrez n'))
p = 1
for k in range(1, n+1) :
....p = p*k
print(p)
```

```
n=int(input('Entrez n'))
p = 1
for k in range(1, n+1) :
....if (k%7) == 0 :
........p = p*2017
....else :
........p = p*k
print(p)
```

```
def fucktorielle(n) :
....p = 1
....for k in range(1, n+1) :
........if k%7==0 :
............p = p*2017
........else :
............p = p*k
....return(p)
```

Le premier script est la factorielle classique. Le second est celui attendu. Le troisième est sous forme "procédure", à utiliser tant qu'on veut.

Tant que n n'a pas atteint 7, les deux définitions coïncident : $\forall n \in range(0,7),\ n! = n \uparrow\uparrow$.

Dans le calcul de $20 \uparrow\uparrow$ par exemple, on dépasse largement la factorielle. En effet, on a remplacé dans le produit 7 et 14 par 2017.

Et ça ne fait qu'empirer. Entre 1 et n, il y a à peu près $n/7$ multiples de 7 qui sont convertis en des 2017.

La fucktorielle grimpe bien bien plus vite que la factorielle.

Mais quand même. Que se passe-t-il quand on a atteitn 2017 ?

On a par exemple $2023! = (2022!).2023$ mais $2023 \uparrow\uparrow = (2022 \uparrow\uparrow).2017$.

Cette fois, la croissance est en défaveur de la fucktorielle. Certes de peu, mais quand même.

Et ensuite : $2030! = (2029!).2030$ mais $2030 \uparrow\uparrow = (2029 \uparrow\uparrow).2017$.

Et à chaque fois qu'on croise un multiple de 7, la factorielle regagne un peu de terrain.

Et même de plus en plus quand on s'éloigne de 2017 par le haut.

Et on a combien de multiples de 7 pour refaire ce retard énorme ? Une infinité.

La factorielle finira par regagner...

J'ai défini le programme ci-contre.
Explicitez `SommePM(n)` pour `n` de `0` à `5`.
Qui est `SommePM(n)[-1]` ?
Justifiez que `0` est dans `SommePM(15)`.
`2018` est dans `SommePM(64)`. Prouvez le. (indication : $\sum_{k=1}^{63} k =?$ et $\sum_{k=1}^{64} k =?$).
.
.
.

```
def SommePM(n) :
....if n== 0 :
........return([0])
....LL = []
........for nombre in SommePM(n-1) :
............if nombre+n not in LL :
................LL.append(nombre+n)
............if nombre-n not in LL :
................LL.append(nombre-n)
....LL.sort()
....return(LL)
```

Voici ce que m'a répondu le `Python` :

0	[0]
1	[-1, 1]
2	[-3, -1, 1, 3]
3	[-6, -4, -2, 0, 2, 4, 6]
4	[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10]
5	[-15, -13, -11, -9, -7, -5, -3, -1, 1, 3, 5, 7, 9, 11, 13, 15]

Que fait ce programme ? Il construit de manière récursive une liste de nombres.
Quand il a trouvé la liste au rang `n-1`, il passe au rang suivant en reprenant tous les éléments de la liste précédente et en leur ajoutant et soustrayant `n`.
Sommairement : `ListePM(n)=[nombre ±n avec nombre dans ListePM(n-1)]`.
Et les nombres de `ListePM(n-1)` étaient des nombre $\pm(n-1)$ avec nombre dans `ListePM(n-2)`.
de proche en proche, on remonte à des $\pm 1 \pm 2 \pm 3 \ldots \pm n$.
Et comme `ListePM(0)` ne contient que 0 (et même ± 0 si vous y tenez), on a la liste triée (et sans

doublons) de tous les nombres de la forme $\pm 1 \pm 2 \pm 3 \ldots \pm n$.
Quitte à ne garder que les nombres positifs, pour n égal à 5, on a
$1+2+3+4+5$, $-1+2+3+4+5$, $1-2+3+4+5$, $1+2-3+4+5$, $1+2+3-4+5$, $-1-2+3+4+5$ et ainsi de suite.

Comme on a la possibilité d'additionner tout, et comme la liste est triée, le dernier terme de `ListePM(n)` est $1+2+3+\ldots+n$, c'est à dire $\frac{n.(n+1)}{2}$.

Pour prouver que `0` est dans `SommePM(12)`, on doit écrire `0` sous la forme d'une somme alternée des entiers de `1` à `12`. On a plusieurs solutions. On les regroupe pour faire des sommes de `13`

1+12	2+11	3+10	4+9	5+8	6+7

et on met des signes moins à une somme sur deux :

1+2+3-4-5-6-7-8-9+10+11+12

Mais j'ai compté 248 solutions... comme `1-2-3-4+5-6-7-8-9+10+11+12` ou `-1-2+3+4-5-6-7-8+9-10+11+12`...

La dernière question demande d'écrire 2018 sous la forme d'une some alterné $\pm 1 \pm 2 \pm 3 \ldots \pm 64$. C'est jouable puisque la somme $1+2+\ldots+64$ atteint 2080.
Il faudra quand même un bon nombre de signes plus. Avec $1+2+\ldots+64$, on atteint 2017.
Entre 2018 et 2080, il y a une différence de 62. Si au lieu de mettre $+31$ on met -31, on a gagné :

$$1+2+3+\ldots+29+30-31+32+33+\ldots+64=2018$$

Il y a d'autres solutions. Rien qu'en jouant sur les 20 premiers signes, j'ai 297 solutions...

On peut montrer que la suite u_0 donné plus grand que 1 $\forall n,\ u_{n+1}=u_n-\ln(u_n)$ converge vers 1 en décroissant. On ne demande pas de le redémontrer. On vous demande un script qui prend en entrée `uzero` et `epsilon` et donne le premier indice `n` vérifiant $|u_n-1|\leqslant\varepsilon$.

Pour le script Python, on sait que la suite converge en décroissante.
La condition d'arrêt est donc juste « plus bas que $1+\varepsilon$ ».
On prend le soin d'importer la fonction `log` du module `math`.

```
from math import log
```

```
def Attente(uzero, epsilon) :
....n, u = 0, uzero
....while u > 1+epsilon :
........n += 1
........u = u-log(u)
....return(n)
```

Écrivez un script qui prend en entrée un entier naturel `n` et retourne la somme des inverses des diviseurs de `n`.

```
def SommeInvDivi(n) :
....S = 0
....for k in range(1, n+1) : #la difficulté est là
........if n%k == 0 : #test de divisibilité
............S += 1./k
....return(S)
```

Écrivez un script `Python` qui prend en entrée une liste `L` et retourne la « matrice circulante » associée. par exemple, pour `L=[1, 4, 3, 2]` il retournera la liste de listes `[[1, 4, 3, 2], [2, 1, 4, 3], [3, 2, 1, 4], [4, 3, 2, 1]]`.

En taille 5, une circulante est de la forme $\begin{pmatrix} a & b & c & d & e \\ e & a & b & c & d \\ d & e & a & b & c \\ c & d & e & a & b \\ b & c & d & e & a \end{pmatrix}$. Si vous aimez les mathématiques, donnez la forme du terme a_i^k de ligne i et colonne k.

On peut copier la liste *(par sécurité)*, et la coller dans la liste de listes en décalant peu à peu :

```
def Circulante(L) :
....LC = L[ : ] #la copie pour ne pas modifier L
....M = [LC]
....for k in range(len(L)-1) : #on a déjà une ligne
........LC = [LC[-1]]+LC[ : -1] #on fait circuler
........M.append(LC) #on valide la nouvelle ligne
....return(M)
```

On peut aussi aller chercher la formule explicite pour le coefficient de position ligne `i` colonne `k` : `L[i-k]`. Ou plutôt `L[i-k modulo n]` où n est la longueur de la liste

```
def Circulante(L) :
....n = len(L) #pour ne la lire qu'une fois
....M = [[L[(i-k) % n for k in range(n)] for i in range(n)]
....return(M)
```

C'est facile !

Sous le nom d'auteur d'*Ellery Queen* se cachaient deux frères qui écrivirent de nombreux romans policiers et fondèrent la revue "*Mystere Magazine*". C'est donc encore une procédure mystère. Que fait elle ? Quel sera sa réponse si n est votre numéro de Sécurité Sociale ?

```
def ElleryQueen(n) :
....N = n
....while n>0 :
........c = n%10
........n = n/10
........N = 10*N+c
....return(N)
```

La division est euclidienne.

Regardons sur un exemple, on prend $n = 2017$. On étudie étape par étape

n	2017	201	20	2	0
c		7	1	0	2
N	2017	20177	201771	2017710	20177102

La procédure prend un nombre et lui colle sa copie miroir. Le nombre $\overline{abcdef}$ donnera $\overline{abcdeffedcba}$.

Voici un programme de multiplication matricielle. Corrigez les erreurs.

.
.
.
.
.
.
.
.

```
def produit(A, B) :
....if len(A[0]) =! len(B) : #pourquoi ce test ?
........return([])
....for i in range(len(A)) :
.......for k in range(len(B)) :
...........S = 0
...........for j in range(len(B)) :
..............S =+A[i][k]*B[j][k]
...........L.append(S)
........M.append(L)
....print(M)
```

Les deux matrices sont sous des formes `A=[[a,b,c,d],[a',b',c',d'], [a",b",c",d"]]`.
`len(A)`, c'est ici `3`, le nombre de lignes de `A`.
`len(A[1])`, c'est ici `4` (`len([a,b,c,d])`) le nombre de colonnes de `A`.
Le test initial `len(A[1]) != len(B)` vérifie si `A` a autant de colonnes que `B` a de lignes. C'est le test de compatibilité des formats. S'il n'est pas rempli, on ressort une matrice vide.
Mais attention, le test, c'est `if len(A[1]) != len(B) :` et pas `if len(A[1]) =! len(B) :`

Ensuite, on crée la matrice ligne par ligne

```
M = []
for i in range(len(A)) :
....L = []
....for k in range(len(B) :
........création du coefficient c[i][k]
```

La formule est `c[i][k] = sum(a[i][j]*b[j][k] for j in range ...)`.
Quel est le range de j ? Le range commun à `A` et `B`, c'est à dire `len(A[0])` ou `len(B)` (les deux sont égaux, puisqu'on a passé le test du début sans sortir par le `return`).
On crée une somme vide, du nom de S à chaque fois, on y accumule les `a[i][j]*b[j][k]`. Mais il y a un piège. On doit accumuler par `S += a[i][j]*b[j][k]` Et ici, on a écrit `S = +a[i][j]*b[j][k]`
A chaque passage, on n'accumule rien, on remplace `S` par la valeur `+a[i][j]*b[j][k]` ce qui fait qu'à la fin de la boucle, on y trouvera juste le nombre `+a[i][n-1]*b[n-1][k]` en notant `n` le range commun. [1]

On colle ensuite ce nombre dans L `L.append(S)`
Quand la ligne `L` est finie, on la colle à la matrice en construction : `M.append(L)` (au sein de la boucle `i`).
Et quand tout est fini, on retourne la matrice `M`, par un `return`, et pas par un `print`.

Si vous n'avez toujours pas compris la différence entre ***`print`*** *et* ***`return`****, n'allez pas passer les concours. Ce sont des concours pour devenir ingénieur, c'est écrit noir sur blanc... Faites une fac d'arts plastique, ou un D.E.A. de force de vente, ou une thèse de philosophie, ou un de la recherche en chimie, mais ne faites pas de sciences...*
.
.
.
.

```
def produit(A, B) :
....if len(A[0]) != len(B) :
#compatibilité
........return([])
....M = [ ]
....for i in range(len(A)) :
.......L = [ ]
.......for k in range(len(B[0])) :
...........S = 0
...........for j in range(len(B)) :
..............S += A[i][j]*B[j][k]
...........L.append(S)
........M.append(L)
....return(M)
```

On constate : $153 = 1^3 + 5^3 + 3^3$, $371 = 3^3 + 7^3 + 1^3$, $370 = 3^3 + 7^0 + 0^3$. Écrivez un programme `Python` qui cherche les nombres à trois chiffres égaux à la somme des cubes de leurs chiffres.

On veut $\overline{abc} = a^3 + b^3 + c^3$, ou encore $100.a + 10.b + c = a^3 + b^3 + c^3$. Pour les trouver, le plus simple est de faire une boucle impérative sur a, une sur b et une sur c :

```
for a in range(10) :
....for b in range(10) :
........for c in range(10) :
............SommeCubes = a*a*a + b*b*b + c*c*c
............Nombre = 100*a+10*b+c
............if SommeCubes == Nombre :
................print(Nombre)
print('Fini')
```

Et on trouve le nombre 407 comme dernière solution (quoique il y a aussi 0 et 1).
Et plus digne d'un MPSI2 :

1. Merci, Clotilde, je n'aurais jamais inventé cette erreur tout seul !

```
Cube = [n*n*n for n in range(10)]
for a in range(10) :
....for b in range(10) :
........for c in range(10) :
............sommeCubes = Cube[a]+Cube[b]+Cube[c]
............Nombre = 100*a+10*b+c
............if SommeCubes == Nombre :
................print(Nombre)
print('Fini')
```

Pensez à une dernière instruction du type `print('Fini')` pour être spur que votre programme n'est pas « encore en train de tourner ».

La notation de la factorielle a longtemps été $\boxed{\text{n}}$, jusqu'à ce qu'en 1808, Christian Kramp invente la notation $n!$, plus pratique. Comme on est en MPSI2, il faut qu'on invente autre chose : $n♪$ est le produit des entiers plus petits que n, mais premiers avec n. Par exemple, 10! vaut 1.2.3.4.5.6.7.8.9.10, alors que 10♪ vaut 1.3.7.9. Calculez $n♪$ pour n de 1 à 13. La suite $(n♪)$ est elle monotone ? Calculez $p♪$ pour p premier. Calculez pour tout n le *p.g.c.d.* de n et $n♪$. Montrez : $2018♪ = \dfrac{2018!}{2^{1009}.1009!.1009}$ *(en sachant que* 1 009 *est premier).* Décomposez 30♪ en produit de facteurs premiers. Résolvez $n♪ = 3^3.7.11.13.17.19$. Trouvez le premier entier n tel que $n♪$ soit un multiple de 1 000.
Écrivez un script Python qui prend en entrée n et retourne $n♪$.

Il ne sera abordé ici que la partie programmation.
On va avoir besoin de savoir si des entiers sont premiers entre eux, on écrit donc une procédure qui calcule le p.g.c.d. de deux entiers (« classique » (?)) :

```
def pgcd(a, b) :
....if a%b == 0 :
........return(b)
....return(pgcd(b, a%b))
```

Il ne reste plus qu'à créer une procédure avec une boucle impérative (`for`) sur une variable qui va de `1` à `n` (inclus, mais ça ne sert à rien).
Pour chaque `k`, on calcule le p.g.c.d. et s'il est convenable (`if`), on multiplie, sinon on n'en fait rien :

```
def fucktorielle(n) :
....p = 1 #on initialise le produit
....for k in range(1, n+1) : #on parcourt
........if pgcd(n, k) == 1 : #on teste
............p *= k #le facteur k est dans le produit
....return(p) #on a fini la boucle
```

Il est toujours amusant de voir des élèves écrire

```
for k in range(1, n+1) : #on parcourt
....if pgcd(n, k) == 1 : #on teste
........p = p*k #le facteur k est dans le produit
....else : #sinon
........p = p #le produit ne change pas
```

On se demande à quoi sert une instruction `p=p` !

Une matrice carrée `M` *(tableau donné sous forme de liste de listes)* est dite magique si chaque somme en ligne est égale à chaque somme en colonne, elle même égale à chaque somme des diagonales.
Écrivez un script Python qui vérifie si une matrice passée en argument est un carré magique.
Par exemple `test([[16, 3, 9, 6], [2, 13, 7, 12], [5, 10, 4, 15], [11, 8, 14, 1]])` devra répondre `True`.
Complétez en carré magique $\begin{pmatrix} 16 & 3 & 2 & 13 \\ & 8 & & 5 \\ & & 6 & \\ & 14 & & \end{pmatrix}$.

Pour tester si une matrice `A` est magique, il faut déjà calculer la somme en question. On la calcule sur la première ligne. Ensuite, on fait défiler les lignes une à une. Si l'une n'a pas le bon total, on sort brutalement par `return(False)`. Et la suite du programme n'est pas exécutée.

> Rappelons que dans une procédure (`def Toto( ) :`), le premier `return` croisé vous fait sortir définitivement de la procédure et n'exécute plus rien de ce qui suit.
> Ici donc, sortir par un `return(False)` dès le premier test invalidé est donc la bonne démarche.
> En revanche, si un test est réussi, ne tapez surtout pas `return(True)` ; vous sortiriez alors dès le premier test.

.

Si on a passé le test des lignes, on passe à celui des colonnes. Si l'un d'entre eux tombe en défaut, on sort brutalement par `return(False)`.
Si on a passé aussi ce test, on calcule la somme d'une diagonale *(somme des `A[i][i]`)*, et on regarde. Si on est en échec on sort, sinon, on passe à l'autre diagonale *(celles des `A[i][n-i-1]`)*.
Si tous les tests ont été passés sans erreur, on valide la matrice par un `return(True)`. Il peut être sans test, puisque pour accéder à la dernière ligne du programme, il faut avoir tout traversé sans faute.

```
def Test(A) : # définition de la procédure
....n = len(A) # pour ne pas aller le cherche à chaque fois
....Somme = sum(A[0][k] for k in range(n)) # somme des éléments de la ligne d'indice 0
....# ligne par ligne
....for i in range(n) :
........Somme_i = sum(A[i][k] for k in range(n)) # somme des éléments de la ligne
........if Somme_i != Somme : # test pour la ligne
............return(False) # échec
....# on passe aux colonnes
....for k in range(n) :
........Somme_k = sum(A[i][k] for i in range(n)) # on somme les A[i][k] avec i qui bouge
........if Somme_k != Somme : # on teste
............return(False) # on invalide
....# la somme en diagonale
....Somme_diag = sum(A[i][i] for i in range(n))
....if Somme_diag != Somme : # son test
........return(False) # raté
....# l'autre somme
....Somme_antidiag = sum(A[i][ni-1] for i in range(n)) #attention à n-i-1
....if Somme_antidiag != Somme :
........return(False) # dernier test : raté, dommage !
....# tout s'est bien passé, on la félicite
....return(True) # cette fois, on sort
```

On a intérêt à poser pour ne pas solliciter à chaque fois des `for in in range(len(A))` qui alourdissent la frappe et font qu'à chaque boucle, le compilateur doit aller rechercher la longueur de la liste qu'on lui avait pourtant déjà demandée au tour précédent.
On détermine donc une fois pour toutes `len(A)` qu'on appelle `n` et on effectue des `range(n)`. C'est à ce genre de détails qu'on reconnaît le vrai programmeur.

La matrice de l'énoncé faisant l'objet du test est $\begin{pmatrix} 16 & 3 & 9 & 6 \\ 2 & 13 & 7 & 12 \\ 5 & 10 & 4 & 15 \\ 11 & 8 & 14 & 1 \end{pmatrix}$ et on a dix sommes valant toutes 34.
Sinon, l'autre exercice est laissé au lecteur matheux.

Une matrice carrée de taille n est dite centro-symétrique si on a $a_i^k = a_{n+1-i}^{n+1-k}$ pour tout couple d'indices $(i,\ k)$ entre 1 et n *(indexation naturelle de 1 à n)*. Exemples ci contre.

$$\begin{pmatrix} 1 & 3 & 5 \\ 4 & 2 & 4 \\ 5 & 3 & 1 \end{pmatrix},$$

$$\begin{pmatrix} a_1^1 & a_1^2 & a_1^3 & a_1^4 \\ a_2^1 & a_2^2 & a_2^3 & a_2^4 \\ a_2^4 & a_2^3 & a_2^2 & a_2^1 \\ a_1^4 & a_1^3 & a_1^2 & a_1^1 \end{pmatrix}$$

La matrice unité I_n est elle centro-symétrique ?

Montrez que le produit de deux matrices centro-métriques de taille n est encore une matrice centro-symétrique de taille n.

Écrivez une procédure `Python` qui vérifie si une matrice *(liste de listes avec indexation de 0 à n − 1)* est centro-symétrique.

$$\begin{pmatrix} a & b & c & d & e \\ a' & b' & c' & d' & e' \\ a" & b" & c" & b" & a" \\ e' & d' & c' & b' & a' \\ e & d & c & b & a \end{pmatrix}$$

Il doit donc y avoir une symétrie par rapport au centre. Les termes de la ligne 1 en a_1^k se retrouvent en ligne n : a_{n+1-1}^*, mais en plus les colonnes sont renversées selon le même principe.

1	2	3	...	i	...	$n-1$	n
n	$n-1$	$n-2$	...	$n+1-i$	...	2	1

La matrice unité vérifie $a_i^k = 0$ sauf pour $i = k$. On étudie a_{n+1-i}^{n+1-k}. Si i n'est pas égal à k, on trouve 0 et sinon, on trouve 1. Comme pour a_i^k.

On prend deux matrices centro-symétriques de taille n. On en note une A de terme général a_i^k, et on note l'autre B de terme général b_i^k *(les deux familles à double indice vérifient la propriété de l'énoncé)*.

La matrice produit a pour terme général $c_i^k = \sum_{j=1}^{n} a_i^j.b_j^k$.

On calcule le terme général de position $(n+1-i,\ n+1-k)$ *(objectif : $c_{n+1-i}^{n+1-k} = c_i^k$)* :

$c_{n+1-i}^{n+1-k} = \sum_{j=1}^{n} a_{n+1-i}^j.b_j^{n+1-k} = \sum_{jj=0}^{jj=n} a_{n+1-i}^{n+1-jj}.b_{n+1-jj}^{n+1-k}$ en posant $\boxed{jj = n+1-j}$ *(qui va de n à 1 quand j va de 1 à n)*.

On utilise la centro-symétrie de a et de b : $c_{n+1-i}^{n+1-k} = \sum_{jj=0}^{jj=n} a_i^{jj}.b_{jj}^k = c_i^k$ car la variable jj est muette...

On voit ici que le calcul formel (reposant sur le simple changement d'indice $jj = n+1-j$) est plus simple que la narration par des colonnes qui tombent sur des lignes.

Si la matrice est passée sous forme de liste de listes, on appelle ses termes par `A[i][k]` à la place de a_i^k. Et les indices `i` et `k` vont de `0` à `n-1` et non plus de 1 à n.

La formule qui compare "*premier et dernier*", "*deuxième et avant dernier*", "*troisième et antépénultième*" n'est plus $i \longmapsto n+1-i$ mais $i \longmapsto n-1-i$:

0	1	2	...	i	...	$n-2$	$n-1$
$n-1$	$n-2$	$n-3$	...	$n-1-i$	...	1	0

On va faire un test par double parcours :

```
for i in range(n) :
....for k in range(n) :
........test
```

Si ne serait ce qu'un test est un échec, on sort tout de suite et définitivement des boucles et de la procédure par un `False`.

Si tous les tests sont positifs, on est allé au bout des boucles, on peut dire que la matrice est centro-symétrique, on répond `True`.

```
def CentroSym(M) :
....for i in range(len(M)) :
........for k in range(len(M[0])) :
............if A[n-1-1][n-1-k] != A[i][k] :
................return(False)
........#une ligne de faite
....#on a fait le tour
....return(True)
```

On notera qu'il n'est pas nécessaire de parcourir toutes les lignes. Si on a `M[0] == M[n-1].reverse()` *(pour parler en* ***Python****)*, alors on a aussi `M[n-1] == M[0].reverse()`. Il suffit donc de faire le teste pour la première moitié des valeurs de `i` : `for i in range((len(M)+1)/2)` : *(si* ***len(L)*** *est pair comme* ***8****, on s'arrête au milieu avec* ***0****,* ***1****,* ***2****,* ***3*** *; si* ***len(M)*** *est impair, comme* ***7****, on explore les lignes* ***0****,* ***1****,* ***2****,* ***3*** *en comparant les termes de la ligne* ***3*** *avec ceux de la même ligne).*

Calculez $\sum_{0\leqslant i+j\leqslant n} i.j$.
Vous n'avez pas confiance dans votre résultat. Alors vous écrivez une procédure `Python` qui prend en entrée `n` et retourne la valeur de cette somme.

Dans la somme $\sum_{0\leqslant i+j\leqslant n} i.j$, on a certes un produit, mais les variables sont dépendantes. On n'a qu'une partie de tableau.

On va fibrer sur i : $\sum_{0\leqslant i+j\leqslant n} i.j = \sum_{i=0}^{n}\Big(\sum_{k=0}^{n-i} i.k\Big)$. La condition $i+k\leqslant n$ entraine en effet que k (évidemment positif) est plus petit que $n-i$.

Pour i fixé, on le sort de la somme $\sum_{k=0}^{n-i} i.k$ où la variable de sommation est k :

$$\sum_{0\leqslant i+j\leqslant n} i.j = \sum_{i=0}^{n}\Big(i.\sum_{k=0}^{n-i} k\Big).$$

On rappelle : $\sum_{k=0}^{p} k = \dfrac{p.(p+1)}{2}$ *(cours)* :

$$\sum_{0\leqslant i+j\leqslant n} i.j = \sum_{i=0}^{n}\Big(i.\frac{(n-i).(n-i+1)}{2}\Big) = \frac{1}{2}.\sum_{i=0}^{n}\Big(i^3-(2.n+1).i^2+(n^2+n).i\Big).$$

On sépare en trois sommes :

$$\sum_{0\leqslant i+j\leqslant n} i.j = \frac{1}{2}.\Big(\frac{n^2.(n+1)^2}{4}-(2.n+1).\frac{n.(n+1).(2.n+1)}{6}+(n^2+n).\frac{n.(n+1)}{2}\Big).$$

Le calcul final donne $\boxed{\sum_{0\leqslant i+j\leqslant n} i.j = \frac{(n-1).n.(n+1).(n+2)}{24}}$ et on se dit qu'il doit y avoir un coefficient binomial caché.

On peut définir

```
def Somme(n) :
....S = 0
....for i in range(n+1) : #et pas n
........for j in range(n+1) :
............if i+j <= n :
................S += i*j
....return(S)
```

A eux trois, les trois nombres A, $2.A$ et $3.A$ utilisent les neuf chiffres de 1 à 9 une fois et une seule. Par exemple 192, 384 et 576. Il parait qu'il y a d'autres solutions. Plus vous en trouvez, plus ça fait de points. Si vous prouvez qu'il n'y a que celles de votre liste, vous engrangez encore. Et si finalement vous préférez un script `Python`, ça fait aussi des points.

Les quatre solutions

A=	1	9	2	A=	2	1	9	A=	2	7	3	A=	3	2	7
2.A=	3	8	4	2.A=	4	3	8	2.A=	5	4	6	2.A=	6	5	4
3.A=	5	7	8	3.A=	6	5	7	3.A=	8	1	9	3.A=	9	8	1

On utilisera la méthode `sort` pour trier une liste. Cette liste sera faite des trois nombres A, $2.A$ et $3.A$ transformés en chaines. On comparera avec la liste de référence des neuf entiers de 1 à 9 *(crée une fois pour toutes au début)*.

```
Test = list('123456789')
for A in range(100, 400) : #inutile de chercher trop petit ou trop grand
....Mot = str(A)+str(2*A)+str(3*A) #le mot
....LMot = list(Mot) #sous forme de liste pour pouvoir trier
....LMot.sort() #justement, on trie
....if LMot == Test : #on compare
........print(A, 2*A, 3*A) #si c'est bon, on affiche
```

On savait que A était plus grand que 100 pour ne pas utiliser de 0 ; et qu'il était plus petit que 333 pour que $3.A$ ne dépasse pas les trois chiffres.

Démontrez $\displaystyle\sum_{k=0}^{n} k^5 = \frac{n^2.(n+1)^2.(2.n^2+2.n-1)}{12}$ puis calculez $\displaystyle\sum_{\substack{k\leqslant 100\\ k\ impair}} k^5$ notée S.

Vous n'avez pas confiance ? Écrivez un script `Python` qui va calculer cette somme S.
Vous avez confiance en vous pour la programmation : ajoutez le script qui va décomposer S *(ou tout autre nombre)* en produit de facteurs premiers.

La formule $\displaystyle\sum_{k=0}^{n} k^5 = \frac{n^2.(n+1)^2.(2.n^2+2.n-1)}{12}$ est vraie au rang 0.

Pour un n quelconque donné, supposons la vraie au grand n et calculons $\displaystyle\sum_{k=0}^{n+1} k^5 = \sum_{k=0}^{n+1} k^5 + (n+1)^5$.

On remplace grâce à l'hypothèse : $\displaystyle\sum_{k=0}^{n+1} k^5 = \frac{n^2.(n+1)^2.(2.n^2+2.n-1)}{12} + \frac{(n+1)^2.12.(n+1)^3}{12}$.

Il nous reste à prouver :

$(n+1)^2.\Big(n^2.(2.n^2+2.n-1)+12.(n+1)^3\Big) = (n+1)^2.(n+2)^2.(2.(n+1)^2+2.(n+1)-1)$.

On développe d'un côté $2.n^4+2.n^3-n^2+12.(n^3+3.n^2+3.n+1)$ et de l'autre $(n^2+4.n+4).(2.n^2+6.n+3)$.
Dans les deux cas, on trouve $2.n^4+14.n^3+35.n^2+36.n+12$. Il y a bien égalité.
Le résultat a été prouvé par récurrence sur n.

On pouvait aussi faire un télescopage, en introduisant $P(X) = X^2.(X+1)^2.(2.X^2+2.X-1)$ et en vérifiant $P(X)-P(X-1) = 12.X^5$. Il ne restait alors plus qu'à télescoper $\displaystyle\sum_{k=0}^{n} P(k)-P(k-1)$.

Pour calculer la somme des impairs, on va calculer la somme de tous et la somme des pairs. Il ne restera qu'à soustraire. Et pourquoi la somme des pairs est plus simple ? Mais parce que

$$\sum_{\substack{k\leqslant 100\\ k\ pair}} k^5 = \sum_{k=0}^{50} (2.p)^5 = 32.\sum_{p=0}^{50} p^5 = 32.\frac{50^2.51^2.(2.2500+2.50-1)}{12}\ .$$

Par soustraction : $\displaystyle\sum_{\substack{k\leqslant 100\\ k\ impair}} k^5 = \sum_{k=0}^{100} k^5 - \sum_{\substack{k\leqslant 100\\ k\ pair}} k^5$.

Le résultat est $\displaystyle\frac{100^2.101^2.(2.10000+2.100-1)}{12} - 32.\frac{50^2.51^2.(2.2500+2.50-1)}{12}$ et on s'en contente.

On pouvait aussi penser à $\sum_{p=0}^{49}(2.p+1)^5$ et utiliser la formule du binôme.
La valeur est 83 291 672 500, mais on s'en moque.

Pour un calcul explicite avec `Python` :

```
def Somme(n) :
....S = 0
....for k in range(n+1) :
........if k%2 == 1 : #test de parité
............S += k**5
....return(S)
```

Et ayant crée cette fonction, on peut demander `S(100)`.
Il faut ensuite décomposer en produit de facteurs premiers. Ça c'est moins évident.
On peut espérer qu'il n'y ait pas dans cette cochonnerie de grands facteurs premiers. Il suffit donc de le diviser par 2 tant qu'on peut, puis par 3 puis par 5 et ainsi de suite.
C'est d'ailleurs l'idée qu'on va exploiter. On avance pas à pas un diviseur possible. Tant que l'entier est divisible par ce diviseur, on le fait et on compte l'exposant. Si il n'est pas ou plus divisible, on passe au diviseur suivant.

```
def Decomposition(N) :
....NN = N #on crée une copie de sécurité car on va le détruire
....L = [] #contiendra la liste [facteur, exposant]
....divi = 1
....while NN != 1 : #tant qu'il lui reste des diviseurs
........divi += 1
........exposant = 0
........while (NN % divi) == 0 : #tant qu'il y a des facteurs divi dans NN
............NN /= divi
............exposant += 1 #on compte l'exposant
........if exposant != 0 #pas la peine d'afficher des 3^0 ou autres
............L.append([divi, exposant]) #comme promis
....return(L)
```

Le résultat final est ici $2^2.5^4.523.63703$ impossible à deviner.
Il est écrit sous la forme `[[2, 2], [5, 4], [523, 1], [63703, 1]]`.

. On notera que notre programme ne va effectivement donner que des diviseurs premiers. Si `N` est par exemple divisible par `10` comme c'est le cas ici (ou même 10^2), quand on arrivera pas à pas à `divi=10`, on sera déjà passé par `2` et par `5` et on aura effacé de `NN` tous ses `2` et tous ses `5`.

Pourquoi ne valorise-t-on pas des programmes comme ça dans les épreuves d'I.P.T. des concours ?

Écrivez un script qui prend en entrée une matrice de taille n sur n et vérifie qu'elle est symétrique, que ses termes diagonaux sont positifs et que ses mineurs de taille 2 *(centrés sur la diagonale)* sont positifs.

On récupère le format de la matrice *(on ne vérifie pas si elles est carrée, ce n'est pas demandé, on espère que le concepteur ne s'est pas trompé)* :

```
def Gram(A) :
....n = len(A)
```

On va tester la symétrie de la matrice : `A[i][k]==A[k][i]` pour `i` de `0` à `n-1` et pour `j` de `i+1` à `n` (pas besoin de vérifier deux fois, pas besoin de vérifier pour les termes diagonaux).
Si un test lors d'une boucle tombe en défaut, on sort brutalement par `return(False)` *(mais on ne sort pas par* ***`return(True)`*** *si un est correct, sinon, on sort dès le premier).*

```
....for i in range(n-1) :
........for k in range(i+1, n) :
............if A[i][k] != A[k][i] :
................return(false)
```

On prend ensuite un à un les termes diagonaux et on teste aussi.

```
....for i in range(n) :
........if A[i][i]<0 :
............return(False)
```

On teste enfin les mineurs `A[i][i]*A[k][k]-A[i][k]*A[k][i]`.

```
....for i in range(n-1) :
........for k in range(i+1, n) :
............mineur = A[i][i]*A[k][k]-A[i][k]*A[k][i]
............if mineur < 0 :
................return(False)
```

Une matrice qui a passé avec succès tous ces teste sans casser la moindre boucle et retourner `False` peut enfin retourner `True`.
On peut même compacter :

```
def Gram(A) :
....n = len(A)
....for i in range(n) :
........if A[i][i] < 0 :
............return(False)
........for k in range(i+1, n) :
............mineur = A[i][i]*A[k][k]-A[i][k]*A[k][i]
............if mineur < 0 :
................return(False)
............if A[i][k] != A[k][i] :
................return(false)
....return(True)
```

N député(e)s présentes [a] dans l'Assemblée sont priées de se regrouper en p goupes *(qui peuvent même ne contenir qu'une personne)*. Chaque personne serre la main des membres de son groupe *(et seulement des personnes de son groupe)*. Combien cela fait il de poignées de main au minimum ? Combien cela fait il de poignées de main au maximum.
Écrivez une procédure `Python` qui prend en entrée N et p et donne la liste des valeurs possibles pour le nombre de poignées de main.
Par exemple, pour $(N, p) = (6, 3)$, *il devra répondre* $[3,\ 4,\ 6]$, *voyez vous pourquoi ?*

a. j'ai décidé que le féminin l'emporte sur le masculin, na !

Exercice laissé au lecteur.

On a donné à Léo les dix chiffres de 0 à 9. Il en a fait quatre nombres : un nombre à un chiffre, un nombre à deux chiffres, un à trois chiffres, un à quatre chiffres. Les quatre sont des carrés parfaits. Pouvez vous refaire la même chose ? Et si on veut toutes les solutions, on prend `Python` ?

On doit prendre dix chiffres et fabriquer une liste de carrés parfaits a, $\overline{bc}$, $\overline{def}$ et $\overline{ghij}$.
Les valeurs possibles pour a sont 0, 1, 4 et 9.
Les valeurs possibles pour b sont 16, 25, 36, 49, 64 et 81. Certaines sont incompatibles avec a.
Pour c la liste s'allonge, même si on élimine 100, 121 et quelques autres qui utilisent plusieurs chiffres en double.
On trouve les douze solutions suivantes :

0, 16, 784, 5329	0, 25, 784, 1369	0, 25, 784, 1936	0, 25, 841, 7396
0, 36, 729, 5184	0, 81, 324, 7569	0, 81, 576, 3249	0, 81, 729, 4356
1, 36, 784, 9025	9, 16, 784, 3025	9, 81, 324, 7056	9, 81, 576, 2304

```
Carres = [[ ] for long in range(5)] #on va créer la liste des carrés
for k in range(100) : #le plus grand 99^2
....carre=str(k*k) #on en fait une chaine
....long=len(carre) #on va voir dans quelle liste l'insérer
....Carres[long].append(carre) #on l'insère
print(Carres) #on vérifie
Tout = list('0123456789') #l'anagramme à trouver
for a in Carres[1] : #on l'appelle a
....for b in Carres[2] : #et il y a b
........for c in Carres[3] : #puis c
............for d in Carres[4] : #et enfin d
................Mot=a+b+c+d #on met les quatre carrés bout à bout
................if sorted(Mot) == Tout : #on teste si c'est un anagramme de 0123456789
....................print(a,b,c,d) #si oui, on l'affiche
print('Fini') #pour savoir si le programme a fini
```

Voici deux scripts

```
def truc(a, b) :
....if a%b == 0 :
........return(b)
....return(truc(b, a%b))
```

```
def machin(n, k, p) :
....S = 0
....for i in range(1, n+1) :
........if truc(i, p) == 1 :
............S += i**k
....return(S)
```

Que donnera l'exécution de `machin(100, 3, 101)` (oui, 101 est premier), `machin(2018, 2, 6)` ?

Le premier script est du cours : calcul du p.g.c.d. de deux entiers.
Ensuite, `machin` prend trois arguments visiblement entiers : `n` *(qui servira de* `range` *pour une somme* $\sum_{k=0}^{n}$*)*, `k` qui sera un exposant dans `i**k` et `p` qui sert de condition : on ne somme que quand le `i` et `p` sont premiers entre eux. On calcule donc $\sum_{\substack{0\leqslant i\leqslant n\\ i\wedge p=1}} i^k$

Quand on calcule comme demandé `machin(100,3,101)`, on somme des cubes de 0 à 100, à condition que `i` et `101` soient premiers entre eux. Mais ils le sont tous.
On calcule donc $\sum_{k=0}^{100} k^3$ et le cours dit que c'est $\frac{100^2.101^2}{4}$.

En revanche, pour `machin(2018,2,6)`, on somme des carrés de 0 à 2018 (inclus) mais seulement pour les nombres premiers avec 6.
On doit donc enlever les multiples de 2, de 3 et de 6.
Si on les avait tous : $\sum_{i=0}^{2018} i^2 = \frac{2018.2019.4037}{6}$.
Si on a les multiples de 2 : $\sum_{k=0}^{1009} (2.k)^2 = 4.\frac{1009.1010.2019}{6}$.
Si on a les multiples de 3 : $\sum_{k=0}^{672} (3.k)^2 = 9.\frac{672.673.1345}{6}$.
Si on a les multiples de 6 : $\sum_{k=0}^{336} (6.k)^2 = 36.\frac{336.337.673}{6}$.

Alors, ensuite, on calcule quoi ? Tout moins ces trois là ? $\sum_{i=0}^{2018} i^2 - \sum_{k=0}^{1009} (2.k)^2 - \sum_{k=0}^{672} (3.k)^2 - \sum_{k=0}^{336} (6.k)^2$.
Non, les multiples de 6 sont déjà enlevés en tant que multiples de 2.
Ils sont d'ailleurs aussi enlevés en tant que multiples de 3. On les a donc enlevés deux fois au lieu d'une. La bonne formule est donc
$$\sum_{i=0}^{2018} i^2 - \sum_{k=0}^{1009} (2.k)^2 - \sum_{k=0}^{672} (3.k)^2 + \sum_{k=0}^{336} (6.k)^2 = \frac{2018.2019.4037}{6} - 4.\frac{1009.1010.2019}{6} - 9.\frac{672.673.1345}{6} + 36.\frac{336.337.673}{6}$$
Si vous y tenez : 914 462 305

Écrivez un script `Python` qui crée la matrice de taille n de la forme suivante facilement généralisable pour n de 2 à 6 :
$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 2 \\ 0 & 1 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 1 & 0 & 0 \\ 3 & 0 & 2 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 2 & 0 & 0 \\ 0 & 3 & 0 & 3 & 0 \\ 0 & 0 & 2 & 0 & 4 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 2 & 0 & 0 & 0 \\ 0 & 4 & 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 & 4 & 0 \\ 0 & 0 & 0 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$

On commence par créer une matrice avec des 0 partout.
Puis on met à `k` les termes de ligne `k`, colonne `k+1` (indexation non pythonienne), et les autres de l'autre côté :

```
def Matrice(n) :
....M = [[0 for k in range(n)] for i in range(n)]
....for k in range(n-1) :
........M[k][k+1] = k+1
........M[k+1][k] = n-1-k
....return(M)
```

On remplit une matrice sous forme de triangle de Pascal écrit de travers par rapport à nos habitudes et coupé en carré, comme $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{pmatrix}$ venant de $\begin{array}{ccccccccc} & & & & & 1 & & & \\ & & & & 1 & & 1 & & \\ & & & 1 & & 2 & & 1 & \\ & & 1 & & 3 & & 3 & & 1 \\ & 1 & & 4 & & 6 & & 4 & & 1 \end{array}$, puis $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{pmatrix}$. Écrivez un script `Python` qui prend en entrée n et fabrique la matrice de taille n sur n.

On crée la matrice de Pascal en ne calculant aucun coefficient binomial. Il suffit d'utiliser la relation dite de Pascal : $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.
Appliquée ici, elle dit qu'un terme est la somme de deux termes : l'un issu de la ligne au dessus, l'autre de sa propre ligne $\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{pmatrix}$: A[i][j]=A[i-1][j]+A[i][j-1].
La méthode consistera donc à remplir déjà une ligne de 1.
Ensuite, on initialise chaque ligne par la valeur 1 et on exploite la formule A[i][j]=A[i-1][j]+A[i][j-1]

```
def Pascal(n) :
....L = [1 for k in range(n)] #premiere ligne
....M = [L] #une ligne
....for i in range(1, n) : #on n'a plus que n-1 lignes à créer
........L = [1] #on amorce
........for k in range(1, n-1) : #il reste n-1 termes
............L.append(L[-1]+M[-1][k]) #calcul
........M.append(L) #la ligne est faite, on la colle
....return(M)
```

Sinon, si vous avez un comportement qui n'a pas la fibre info, vous comprenez qui est le terme d'indice i, k de la matrice $a_i^k = \binom{i+k}{k} = \binom{i+k}{i}$.
Si vraiment vous êtes le roi des relou :

```
def Fact(i) :
....p = 1
....for k in range(i) :
.......p *= k+1 #et pas k
....return(p)
```

```
Def Binom(n, k) :
....Haut = Facto(n)
....Bas = Fact(k)*Fact(n-k)
....return(Haut/Bas)
#on sait que la division tombe juste
```

```
def Pascal(n) :
....M = []
....for i in range(n) :
........L = [ ]
........for k in range(n) :
............L.append(Binom(i+k,k))
........M.append(L)
....return(M)
```

Si vous faites ça, normalement, un correcteur d'Informatique Pour Tous vous donne tous les points. Et un correcteur d'informatique vous en donne le tiers, par générosité.

On définit la suite $\left(\sqrt{1},\ \sqrt{1.\sqrt{2}},\ \sqrt{1.\sqrt{2.\sqrt{3}}},\ \sqrt{1.\sqrt{2.\sqrt{3.\sqrt{4}}}},\ \sqrt{1.\sqrt{2.\sqrt{3.\sqrt{4.\sqrt{5}}}}},\dots\right)$. Écrivez un script `Python` qui prend n et retourne le terme d'indice n. Justifiez que cette suite est croissante.
Variante $\left(\sqrt{1!},\ \sqrt{1!.\sqrt{2!}},\ \sqrt{1!.\sqrt{2!.\sqrt{3!}}},\ \sqrt{1!.\sqrt{2!.\sqrt{3!.\sqrt{4!}}}},\ \sqrt{1!.\sqrt{2!.\sqrt{3!.\sqrt{4!.\sqrt{5!}}}}},\dots\right)$.

On n'oublie pas d'importer la fonction racine carrée.
On profite des range inversés.
On compte bien combien de racines on a pris au total.

```
from math import *
def racines(n) :
....x=1
....for k in range(n,0,-1) :
........x=k*sqrt(x)
....return(sqrt(x))
```

Pour le même avec des factorielles, on a deux approches possibles.
On peut calculer toutes les factorielles par un programme à part :

```
from math import *
def factorielle(p) :
....facto = 1
....for i in range(1, p+1) :
........facto *= i
....return(facto)
def racines(n) :
....x=1
....for k in range(n,0,-1) :
........x=factorielle(k)*sqrt(x)
....return(sqrt(x))
```

On calcule bien des fois la même chose (dès la première boucle sur k, on calcule $n!$, on l'oublie, et au tour suivant, on recalcule $(n-1)!$. On l'oublie à son tour, et au rang suivant on refait un gros produit : $(n-2)!$.
Si on ne veut pas faire n fois le même calcul, mais qu'on a de la place en mémoire, on calcule d'abord la liste de toutes les factorielles dont on va avoir besoin.

```
def ListeFact(n) : #liste[n !, (n-1) !, (n-2) !, ...6, 2, 1, 1]
....L = [1]
....for k in range(1, n+1) :
........Nouvelle Factor = L[0]*k
........L = [NouvelleFactor]+L
def Racines(n) :
....x = 1
....L = ListeFact(n) #longueur n+1
....for k in range(n+1) :
........x = L[k]*sqrt(x)
....return(x)
```

Pour n donné, on a $n.\sqrt{n+1} \geqslant n$.
On passe à la racine et on multiplie par $n-1$ (positif) : $(n-1).\sqrt{n.\sqrt{n+1}} \geqslant (n-1).\sqrt{n}$.
On passe à la racine (croissante), on multiplie : $(n-2).\sqrt{(n-1).\sqrt{n.\sqrt{n+1}}} \geqslant (n-2).\sqrt{(n-1).\sqrt{n}}$.
On remonte ainsi pas à pas, par récurrence sur le nombre de racines, jusqu'à

$$\sqrt{1.\sqrt{2\sqrt{\ldots(n-2).\sqrt{(n-1).\sqrt{n.\sqrt{n+1}}}}}} \geqslant \sqrt{1.\sqrt{2\sqrt{\ldots(n-2).\sqrt{(n-1).\sqrt{n}}}}}$$

Si récurrence il y a (et il y en a une), elle porte à n fixé sur le nombre de remontées.
On ne dire donc pas "par récurrence sur n", sans risquer de passer pour un idiot.

Réfléchissez : comment allez vous justifier que vous passez de
$\sqrt{1.\sqrt{2.\sqrt{3}}} \leqslant \sqrt{1.\sqrt{2.\sqrt{3.\sqrt{4}}}}$ *à* $\sqrt{1.\sqrt{2.\sqrt{3.\sqrt{4}}}} \leqslant \sqrt{1.\sqrt{2.\sqrt{3.\sqrt{4.\sqrt{5}}}}}$
Il faut foutre à la poubelle le réflexe "il y a un entier n, donc je dis "récurrence sur n".
Il faut aussi foutre à la poubelle le réflexe "il y a un entier, donc je dis récurrence sans préciser sur qui".
Bref, encore et toujours, je suis votre prof de maths, pas votre prof de calcul.

Le nombre de Chapernowne est le nombre qui s'écrit $0,12345678910111213141516171819 20...$ en collant bout à bout les entiers naturels. Il ne sert a priori à rien hormis à créer des exercices tels que celui ci :
Ecrire une procédure qui, pour n donné retourne la n^{ieme} décimale du nombre de Chapernowne.
Ecrire une procédure qui, pour n et N donnés retourne la liste des décimales du nombre de Chapernowne, de la n^{ieme} à la N^{ieme}.
Ecrire une procédure qui, pour n donné retourne la n^{ieme} décimale du carré du nombre de Chapernowne.

L'exercice ne sera pas traité ici.

Écrivez un script `Python` qui prend en entrée une suite *(sous forme de liste)* et retourne sa moyenne de Cesàro (suite), et estimez la complexité de votre programme en fonction de la longueur n de la liste entrée.
La moyenne de Cesaro de $(a_0,\ a_1,\ a_2,\ a_3,\ a_4, \ldots a_n \ldots)$ est la suite $\left(a_0,\ \dfrac{a_0+a_1}{2},\ \dfrac{a_0+a_1+a_2}{3}, \ldots\right)$ de terme général $c_n = \dfrac{a_0+a_1+\ldots+a_n}{n+1}$ (oui, il y a bien $n+1$ termes au numérateur).

On a plusieurs solutions, plus ou moins lourdes, suivant qu'on est simple lecteur de formules ou plus programmeur :

```
def Cesaro(S) :
...n = len(S)
...C = [ ]
...for k in range(n) :
......somme = 0
......for i in range(k+1) :
.........somme += S[k]
......terme = suite/(k+1)
......C.append(terme)
...return(C)
```

Complexité : k addition et un quotient pour chaque k de 0 à n
total $\displaystyle\sum_{k=0}^{n-1}(k+1) \sim \frac{n^2}{2}$

```
def Cesaro(S) :
...n = len(S)
...cumul = 0
...C = [ ]
...for k in range(n) :
.......cumul += S[k]
.......terme = cumul/(k+1)
.......C.append(terme)
....return(C)
```

Complexité : $2.n$.

```
def Cesaro(S) :
...n = len(S)
...C = [S[0]]
...for k in range(1, n) :
......c =
(C[-1]*k+S[k])/(k+1)
......C.append(c)
...return(C)
```

Complexité : $3.n$.

La première est à peine digne d'un (M)PSI[2], car elle recalcule des sommes déjà calculées.

La seconde comporte un risque : la somme cumul risque de devenir très grande puisque c'est $\displaystyle\sum_{i=0}^{k} u_i$, et si les u_i sont grands et n aussi, on risque le dépassement.
La dernière n'a pas ce risque, puisque tous les nombres calculés sont de l'ordre de u_i et c_i, par la formule $c_k = \dfrac{u_0 + \ldots + u_{k-1} + u_k}{k+1} = \dfrac{k.c_k + u_k}{k+1}$. Mais les petits arrondis à chaque étape finissent par s'accumuler aussi.

Le saviez vous : $88^2 + 33^2 = 8\ 833$; $12^2 + 33^2 = 1\ 233$; $588^2 + 2\ 353^2 = 5\ 882\ 343$, en tout cas, Lenstra, van der Poorten, Weibe et Thomsen l'ont constaté et généralisé. Tenez, j'ai aussi $9412^2 + 2353^2 = 94\ 122\ 353$. Écrivez un script pour le Python suprême qui cherche les exemples à deux fois quatre chiffres $(abcd)^2 + (efgh)^2 = abcdefgh$.

On a donc $88^2 + 33^2 = 88.100 + 33$. On cherche des identités de la forme $a^2 + b^2 = 10^n.a + b$ avec n égal au nombre de chiffres de b.
Cherchons des exemples à deux chiffres : $a^2 + b^2 = 100.a + b$. On bascule : $a.(100 - a) = b.(b - 1)$.

Peut on avoir un exemple à deux chiffres (ou même nombres) a et b vérifiant $a^2 + b^2 = 10.a + b$?
Dans ce cas $a.(10 - a) = b.(b - 1)$.
On peut demander que tout soit nul de chaque côté : $a = 10$ et $b = 1$: $\boxed{10^2 + 1^2 = 101}$

Python m'a trouvé $9\ 412^2 + 2\ 353^2 = 94\ 122\ 353$. Mais aussi $-34^2 + 68^2 = 3\ 468$.
Comment ? Par deux boucles rapides :

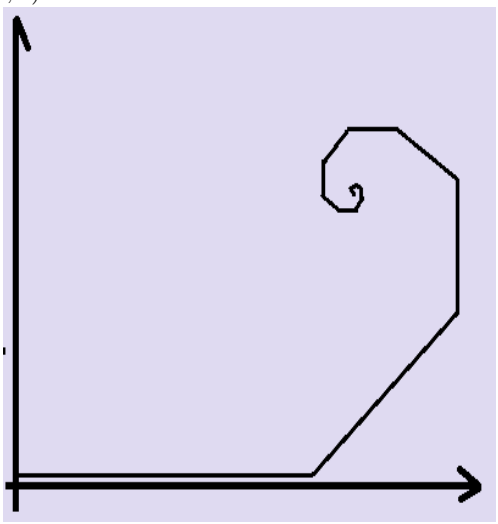
```
for a in range(1000):
....for b in range(1000):
........if a*a+b*b = 1000*a+b:
............print(a,b)
print('fini')
```

2. sans le M de maths ni le I si c'est Informatique

Si `N` est suffisament grand pour que le physicien le trouve infini, où est arrivée la tortue ? 3 pt.

```
from turtle import * #module turtle qui déplace un pointeur à l'écran
dist=200.
for k in range(N) :
....forward(dist) #avancer de dist droit devant
....left(45) #rotation de 45 degrés sens trigonométrique
....dist *= 0.8
```

La règle : on avance de `dist`, on tourne, on divise remplace `dist` par `dist*0.8`, et on recommence.
On avance avec des pas de plus en plus petits, et en changeant de direction à chaque fois.
Le plus simple est de modéliser avec des affixes représentant chaque nouveau déplacement.
Les affixes ont un module et un argument à chaque étape.
Le module est multiplié par 0.8 et l'argument augmente de 45 degrés ($\pi/4$).
A chaque nouvea pas, l'affixe du déplacement est $200.(0,8)^n.e^{i.n.\pi/4}$.
On somme tous les délacements successifs : on a une série géométrique $\sum_{k=0}^{N-1} 200.(0,8)^n.e^{i.n.\pi/4}$.



Sa somme au rang N est donc $200.\dfrac{1-((0,8).e^{\pi/4})^N}{1-0,8.e^{\pi/4}}$
(la raison de la série vaut $0,8.e^{i.\pi/4}$*).*
Comme N tend vers l'infini, il reste $\boxed{\dfrac{200}{1-0,8.e^{\pi/4}}}$
On peut mettre sous forme cartésienne $200.\dfrac{(1-0,4\times\sqrt{2})-i.(0,4\times\sqrt{2})}{\sqrt{(1-0,4\times\sqrt{2})^2+(0,4\times\sqrt{2})^2}}$ et même dire que le dénominateur vaut $\dfrac{\sqrt{41+20.\sqrt{2}}}{5}$. Application numérique 170-222*i.

Pour tout entier naturel n, on note $s(n)$ le nombre de chiffres premiers dans l'écriture de n (exemple : $s(2019)=1$ [a], $s(1789)=1$, $s(5435)=3$). Montrez que la série de terme général $\dfrac{s(n)}{n^2}$ converge.
Écrivez un script `Python` qui pour N donne calcule la valeur approchée de $\sum_{k=1}^{n}\dfrac{s_k}{k^2}$.

a. on rappelle que 0 et 1 ne sont ni premiers, ni composés)

Attention, il n'est pas utile de créer un test pour savoir si un nombre est premier !
En effet, seuls les entiers de 0 à 9 seront testés. Autant savoir dnc tout de suite qui parmi eux est premier, et face à un nombre tel que 2019, juste tester si 2, 0, 1 et 9 sont dans cette liste !

Bon, les chiffres, c'est de 0 à 9.

	0	1	2	3	4	5	6	7	8	9
premier			x	x		x		x		
composé					x		x		x	x
ni premier ni composé	x	x								

Bref, on doit compter combien il y a de 2, de 3, de 5 et de 7 dans l'écriture de n.
On va prendre n, le faire fondre en divisant par 10, et pour chaque chiffre extrait, on regarde si il vaut 2, 3, 5 ou 7.

```
def ComptePrem(n) :
....nn = n #on travaille sur une copie
....compt
....while nn > 0 :
........chiffre = nn%10
........nn = nn/10
........if chiffre in [2, 3, 5, 7] :
............compte += 1
....return(compt)
```

```
def ComptePrem(n) :
....compt = 0
....Mot = str(n)
....for chiffre in Mot :
........if chiffre in ['2','3','5','7'] :
............compt += 1
....return(compt)
```

Si on travaille sur entiers, le test est `chiffre in liste de chiffres`, si on travaille sur `string`, le test est sur `chiffre in liste de caractères`.

On a crée la petite procédure utile. Il vaut mieux découper le travail, et éviter l'étirement du programme.
La série est à termes positif, elle croit. Pour la faire converger, il suffit de la majorer.

```
def Somme(N) :
....S = 0
....for n in range(1, N+1) : #gare aux bornes
........S += ComptePrem(n)/(n*n)
....return(S)
```

Cadeau : preuve mathématique de la convergence de la série.

Mais on ne peut pas majorer $\frac{s(n)}{n^2}$ par $\frac{K}{n^2}$ puisque le nombre de chiffre de n peut être grand, et pas mal de ses chiffres peuvent être premiers (et même tous (écrivez des 2323235323...7, ce n'est pas ce qui manque).
Mais $s(n)$ est au maximum égal au nombre de chiffres de n. Et ce nombre de chiffres, c'est $log(n)$. Ou plus précisément : $\left[\frac{\ln(n)}{\ln(2)}\right] + 1$.

On est face à une série majorée par $\frac{\ln(n)}{n^2}$. Cette fois, c'est mieux.

Si on majore $\ln(n)$ par n, on ne peut rien faire : $\frac{s_n}{n^2} = O\left(\frac{1}{n}\right)$ n'est pas pertinent, puisque la série de terme général $\frac{1}{n}$diverge.

Si en revanche on écrit $\ln(n) = o(\sqrt{n})$, on a $\frac{s(n)}{n^2} = \frac{o(\sqrt{n})}{n^2} = o\left(\frac{1}{n.\sqrt{n}}\right)$.

La série de terme général $\frac{1}{n.\sqrt{n}}$converge (exposant strictement plus grand que 1). par domination, la série de terme général $\frac{s(n)}{n^2}$converge.

Une suite de Saias et Mazet [a] est une suite d'entiers naturels non nuls, distincts, où chaque terme est diviseur ou multiple du terme qui suit. Par exemple `[1, 2, 6, 3, 12, 4]`, ou `[5, 1, 4, 2, 6, 3]`. Écrivez un script qui prend en entier une liste de nombres et vérifie si elle est de Saias et Mazet. [b]
On peut construire une suite de Saias et Mazet qui contient tous les entiers naturels. En voici le début : `[1, 2, 6, 3, 12, 4, 20, 5, 35, 7, 56, 8, 72, 9, 90, 10, 110, 11, 143, 13]` Donnez les six termes suivants. Donnez l'algorithme qui en fournit les `n` premiers termes.

a. Eric Saias et Pierre Mazet, mathématiciens français contemporains
b. Pour le test de type de variable, vous ne connaissez pas : « `type(a) is int` » est un booléen qui dit si `a` est de type `int`, vous pouvez tester aussi `char`, `float`, `str`, `list`....

On va parcourir la liste `L` [3], vérifier que les éléments sont des entiers, positifs, pour chacun, on vérifiera qu'il divise le suivant ou est divisible par lui. En cas d'échec, on sortira tout de suite avec un `False`. Si on est allé jusqu'au bout *(attention au range, on ne doit pas aller trop loin)*, on sortira vainqueur. Au fur et à mesure, on vérifiera que l'élément ne fait pas déjà partie des précédents, puisqu'on a demandé « tous distincts ».

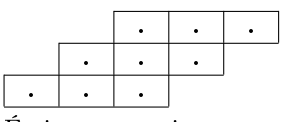
3. pas original d'appeler une liste `L`, mais `OirDeFrance` c'est trop long

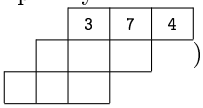
```
def TestMazet(L) :
....for elt in L :
........if not(type(elt) is int) :
............return(False) #l'un d'entre eux n'est pas entier
........if elt <= 0 :
............return(False) #ou s'il est entier, il est négatif !
....for k in range(len(L)-1) : #ne pas déborder
........Element, Suivant = L[k], L[k+1] #comme on va les utiliser plusieurs fois
........TestDiv = (Element%Suivant==0) or (Suivant%Element==0) #le test
........if not(TestDiv) : #le test a échoué
............return(False) #on sort brutalement
........if Suivant in L[ : k+1] :
............return(False) #la valeur est déjà prise
....return(True)
```

Pour remplir la suite qui va prendre tous les entiers un à un. On part de 1 et 2. On aimerait tout de suite atteindre 3, on va transiter par 6 (multiple de 2 et de 3), et on redescendra à 3.
On veut atteindre 4, on prend le *ppcm* de 3 et 4, c'est 12, on le place après 3, on redescend à 4.
On insère le couple 20, 5.
On n'a pas besoin ensuite d'accéder à 6, il est déjà apparu dans la suite. On passe donc à 7 qui n'y est pas encore, et on y accède par l'intermédiaire du produit 35 *(dont on est sûr qu'il n'y est pas encore).*

```
def Mazette(n) :
....L = [1, 2]
....while len(L) < n :
........Element = L[-1] #le dernier trouvé
........Suivant=Element+1 #celui qu'on veut
............while suivant in L : #mais s'il est déjà pris
................Suivant +=1 #et même tant qu'il est déjà pris
........Multiple = Element*Suivant #le produit pour monter descendre
........L.append(Produit) #on monte
........L.append(Suivant) #on descend
....return(L)
```

On effectue une boucle conditionnelle « tant que la longueur de la liste est plus petite que n.
Il est plus difficile d'effectuer une boucle impérative (for), car à chaque étape, on a plusieurs types de comportements possibles pour allonger la liste.
Bon, il est possible que la suite soit trop longue car on avance de deux en deux. Si `n` était impair, on a un élément de trop, on le détruit par `return(L[ : n])`
`[1, 2, 6, 3, 12, 4, 20, 5, 35, 7, 56, 8, 72, 9, 90, 10, 110, 11, 143, 13, 182, 14, 210, 15, 240, 16, 272, 17, 306, 18, 342, 19, 399, 21, 462, 22, 506, 23, 552, 24, 600, 25, 650, 26, 702, 27]`
La complexité est en `n` pour le parcours, mais à chaque valeur de `k`, il y a des tests sur une liste de longueur `k`. On a donc plutôt $\sum_{k=0}^{k} k$, ce qui crée un $O(n^2)$.



Remplissez cet escalier avec les neuf entiers de 1 à 9. La consigne : dans aucune ligne il ne peut y avoir deux entiers consécutifs (on refuse par exemple 

) [a].

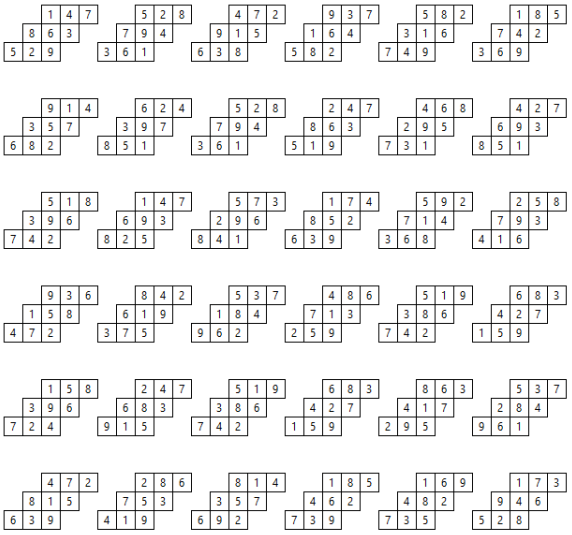
a. je précise car une année, un élève a poliment demandé : « c'est quoi un entier consécutif »

Écrivez un script `Python` qui prend en entrée la liste du type `[a, b, c, a', b', c', a'', b'', c'']` et vérifie si elle est valide.
Écrivez un script `Python` qui cherche toutes les solutions.

On peut proposer

		1	8	6
	7	5	3	
4	2	9		

mais il y en a d'autres.

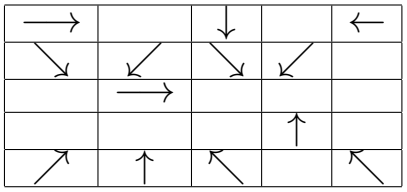


```
def Test(L) : #test brut sur une liste
....if Contigu(L[0],L[1],L[2]) : #ligne 0
........return(False)
....if Contigu(L[3],L[4],L[5]) : #ligne 1
........return(False)
....if Contigu(L[6],L[7],L[8]) : #ligne 2
........return(False)
....if Contigu(L[0],L[4],L[8]) : #colonne
........return(False)
....if Contigu(L[0],L[3],L[6]) : #diago-
nale 0
........return(False)
....if Contigu(L[1],L[4],L[7]) : #diago-
nale 2
........return(False)
....if Contigu(L[2],L[5],L[8]) : #diago-
nale 3
........return(False)
....if sorted(L) != liste(range(1,10)) :
........return(False)
....return(True)
```

```
def Contigu(a,b,c) : #test de conti-
guite
....if abs(a-b)==1 :
........return(True)
....if abs(a-c)==1 :
........return(True)
....if abs(b-c)==1 :
........return(True)
....return(False)
```

Il faut penser aussi à tester que la liste est bien formée des entiers de 1 à 9.
J'ai ensuite mis un boucle des listes aléatoires (`shuffle` sur `list(range(1,10))`) et affiché grâce à `Tkinter` les solutions trouvées.
Sinon, il faudrait par exemple générer toutes les listes de permutations de la liste [1, 2, ...9] et les tester, sachant que beaucoup de tests sont inutiles, dès que la première ligne est erronée par exemple.
Il faudrait ensuite éliminer les solutions qui se déduisent d'une même solution par renversement par exemple.

Voici un jeu des défi maths pour école primaire (et de la Fédération Française des Jeux Mathématiques) : Chaque flèche vise toute les cases vides de sa rangée (ligne, colonne ou diagonale), même « à travers » une autre flèche. Place un jeton dans chaque case vide visée par au moins trois flèches.



Niveau Bintou : résolvez cet exercice.

Niveau I.P.T. : les données sont un entier (`n`, la taille du carré) et huit listes : les cases où sont les flèches en fonction de leur direction (ici : `N = [[3,3], [4,1]]`, `S=[[0,3]]`, `E=[[0,0], [2,1]]`, `O=[[0,4]]`, `NE=[[4,0]]`, `SO=[[1,3], [1,1]]`, `NO=[[4,2], [4,4]]` et `SE=...`) Écrivez un script `Python` qui retourne alors la liste des cases répondant au critère « *case vide visée par au moins trois flèches* ».

Bonus : écrivez un script qu'il n'y a pas dans les données des incohérences du type « *une flèche hors du tableau* », « *une case avec deux flèches* ».

Super bonus : écrivez un script avec des `can.create_rectangle(...)`, `can.create_oval(...)`.

On commence par le truc qui teste quand même que les cases sont bien dans le bon range.

```
def TestOccupation( ) :
....Occupees = N+S+E+O+NE+SE+NO+SO #toutes les cases occupées
....NbOcc = len(Occupees) #nombre de cases occupées
....for k in range(NbOcc) :
........Case = Occupees[k] #on lit la case
........if Case[0]<0 or Case[0]>=n or Case[1]<0 or Case[1]>=n : #est elle hors tableau
............return(False)
........for i in range(k) : #on regarde les cases avant elle
............if Case == Occupees[i] : #était elle déjà dans la liste ?
................return(False)
....return(True) #tout s'est bien passé
```

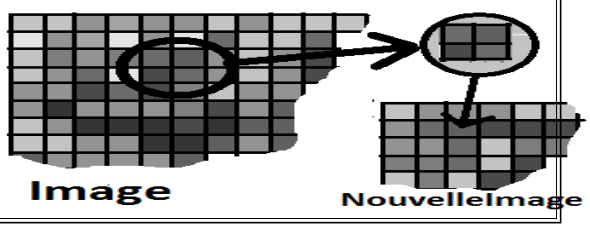
On considèrera que `n`, `N`, `S`, `E`, `O`... sont des variables globales, sinon, on les passe en variables dans la procédure qui suit.

```
def Resolution( ) :
....Occupees = N+S+E+O+NE+SE+NO+SO #toutes les cases occupées
....Positions = [ ] #liste des positions où on mettra un jeton
....for i in range(n) : #ligne à ligne
........for k in range(n) : #on avance sur la ligne
............if Test(i,k) :
................Positions.append([i, k])
....return(Positions)
```

Il faut penser aussi au test pour chaque case.

```
def Test(i,k) :
....if [i, k] in Occupees : #on ne regarde que les cases vides
........return(False)
....NbVisees = 0
....for Fleche in N : #les flèches visant le Nord
........if Fleche[1] == k : #même colonne
............if Fleche[0] > i : #et venant du bas
................NbVisees += 1 #la case est visée
....for Fleche in S : #les flèches visant le Sud
........if Fleche[1] == k if Fleche[0] < i : #même colonne et venant du haut
................NbVisses += 1 #la case est visée
....for Fleche in E : #les flèches visant l'Est
........if Fleche[0] == i and if Fleche[0] < k : #même ligne et venant de gauche
................NbVisses += 1 #la case est visée
....for Fleche in O : #les flèches visant l'Occident
........if Fleche[0] == i and if Fleche[0] > k : #même ligne et venant de gauche
................NbVisses += 1
....for Fleche in NE :
........if Fleche[0]+Fleche[1] == i+k : #même diagonale
............if Fleche[1] < k : #du bon coté
................NbVisses += 1
....#et encore trois tests de ce genre
....return(NbVisees >= 3) #le booleen est évalué, on répond donc True ou False.
```

Une image en noir, blanc et gris de taille 200 sur 100 est sauvegardée sous forme de matrice, pixel par pixel. C'est trop précis. Sauvegardez la sous la forme d'une matrice de taille 100 sur 50 où chaque pixel est la moyenne des quatre pixels dont il est le représentant. Chaque pixel est un niveau de gris codé par un entier. 3 pt.



Les pixels sont stockés dans une matrice : le pixel $(i,\ j)$ *(ligne i et colonne j)* est dans `Image[i][j]`, indexé pour `i` de `0` à `199` et `j` de `0` à `99` *(dans la pratique pour généraliser `0` à `len(Image)` et `0` à `len(Image[i])`).*
On va créer une nouvelle image où chaque `NouvelleImage[i][j]` est une moyenne. Déjà, son `i` va aller de `0` à `99` et son `j` de `0` à `49`. *(on prendra `len(Image/2)`)* De qui sera-t-il la moyenne ? Des éléments de `Image` en lignes `2*i` et `2*i+1` et en colonnes `2*j` et `2*j+1`.

```
def reduction(Image) :
....NouvelleImage=[ ] #une liste vide qu'on va agrandir
....for i in range(len(Image)/2) : #division euclidienne
........NouvelleLigne=[ ] #on crée une ligne à agrandir
........for j in range(len(Image[i]/2) :
............Somme = Image[2*i][2*j]+Image[2*i][2*j+1]
....................+Image[2*i+1][2*j]+Image[2*i+1][2*j+1]
............NouvelleLigne.append(Somme/4) #division euclidienne
........NouvelleImage.append(NouvelleLigne) #la ligne est finie
....return(NouvelleImage) #l'image est finie
```

Les ensembles sont données sous formes de listes `L`. Et on donne la liste des ensembles `ListeEns`. Écrivez une procédure `Python` qui crée alors la matrice dont le terme général est $Card(ListeEns[i] \cap ListeEns[k])$. *Par exemple, pour `[[1, 2, 3, 5], [1, 2, 4], [2, 5, 6], [5, 6]]` elle donnera la matrice* $\begin{pmatrix} 4 & 2 & 2 & 1 \\ 2 & 3 & 1 & 0 \\ 2 & 1 & 3 & 2 \\ 1 & 0 & 2 & 2 \end{pmatrix}$. *Ces matrices font l'objet d'un exercice d'oral d'un grand concours. Elles sont du ressort de l'algèbre linéaire et des probabilités.*

On a une liste de listes, faite de `n` ensembles. la matrice est alors de taille `n` sur `n`. Le mieux est de la créer d'abord, puis de remplir les cases une à une. On va avoir besoin d'une procédure `Inter` qui calcule le cardinal de l'intersection, c'est à dire qui pour deux ensembles `A` et `B` donnés dit combien il y a d'éléments de `A` qui sont aussi dans `B`.

```
def Inter(A, B) :
....Compt = 0
....for a in A :
........if a in B :
............Compt+=1
....return(Compt)
```

```
def MatriceSym(ListeEns) :
....n = len(ListeEns)
....M = [[0 for k in range(n)] for i in range(n)]]
....for k in range(n) :
........A = ListeEns[k]
........M[k][k] = len(A)
........for i in range(k+1, n) :
............cardinal = Inter(A, ListeEns[i])
............M[k][i], M[i][k] = Cardinal, Cardinal
....return(M)
```

On évite de calculer plusieurs fois la même chose. Et on ne mesurera pas $Card(A \cap A)$ en utilisant la fonction `Inter(A,A)`, ce serait du gaspillage.

Écrivez un programme `Python` qui prend en entrée une matrice carrée `M` (liste de listes) et un vecteur `V` (liste) et répond `False, 0` si `V` n'est pas vecteur propre de `M` et `True, k`, si `V` est vecteur propre de `M` de valeur propre `k` (c'est donc au programme de trouver la valeur propre). Par exemple `[[1,1],[2,1]], [1,3]` -> `False,0` et `[[4,2],[1,3]], [1,-1]` -> `True,2`. Il faudra tester quand même que la matrice est bien carrée et les formats compatibles, pour déjà un point.

```
def VecteurPropre(M, V):
....if len(M) != len(M[0]):
........return(False,0)
....if len(M) != len(V):
........return(False, 0)
....n = len(M)
```

Les deux tests pour savoir si `M` a autant de lignes que la longueur de sa première ligne (c'est à dire que de colonnes), puis pour savoir si `V` a la même taille que cette longueur commune.
Inutile d'utiliser des `else` :
si la première condition n'est pas vérifiée, on sort tout de suite, mais si elle est vérifiée, on continue.

On donne un nom `n` à la longueur commune pour ne pas avoir à taper sans cesses `len(M)` et éviter à `Python` de faire des routines inutiles.
On va calculer le vecteur colonne `M.V` et regarder si les deux vecteurs sont proportionnels.
On peut calculer une bonne fois pour toutes `M.V` :

```
....MV = []
....for i in range(n):
........MV.append(sum(M[i][k]*V[k] for k in range(n))
```

ou

```
....MV = []
....for i in range(n):
........S = 0
........for k in range(n):
............S += M[i][k]*V[k]
........MV.append(S)
```

puis tester la proportionnalité ligne par ligne avec des petits déterminants de taille 2 $\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}$
Si ne serait ce qu'un seul des tests est invalide, on sort par un `return(False,0)`. Mais si un teste est valide, on continue, c'est tout !

```
....for i in range(n-1):
........if V[i]*MV[i+1] != V[i+1]*MV[i]:
............return(False)
....return(True, MV[0]/V[0])
```

Quand on a tout passé sans faute, on répond donc `True` *(`V` et `MV` sont colinéaires)*, et on lit le rapport de colinéarité sur la première ligne.

A un détail près. Si `V[0]` est nul... Après tout, $\begin{pmatrix} 7 & 3 & -3 \\ 8 & 6 & -4 \\ 26 & 14 & -12 \end{pmatrix} . \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix}$. *C'est un vecteur propre, mais le raport se trouve avec* $2/1$ *et pas avec* $0/0$.
Il faut donc normalement chercher le premier coefficient `V[i]` non nul de `V` pour calculer `MV[i]/V[i]`

```
....index = 0
....while V[index] == 0:
........index+=1
....return(True, MV[index]/V[index])
```

Et si le programme a planté par un `index out of range` parce que tous les coefficients de `V` sont nuls, c'est bien fait pour l'utilisateur : le vecteur nul n'est pas vecteur propre : il vérifie $M.O = \lambda.O$ pour tous les λ, ce qui n'est pas cohérent.

Un théorème affirme que $2^{(3^n)} + 1$ est toujours multiple de 3^{n+1}.
Vous avez deux missions :
• le démontrer (par exemple par récurrence sur n)
• et aussi écrire un script `Python` qui le vérifie pour les 1000 premiers n. Attention, si votre script utilise `**` ou une fonction `power`, je déduirai que vous avez appris l'informatique avec un physicien ou un sisyphien et je vous sanctionnerai d'un point pour demander à l'ordinateur des calculs répétitifs et inutiles.
(de toutes façons, $2^{3^8} + 1$ *a déjà près de deux mille chiffres* :

114235622173779374328348175648643910549218134760744031862028761863071947087591566201928157368569531607827046253207987994804937857835390714753081815589584098861391805077662557452688352956451173135043388192631

74032688332546457426755549620769428586957231464831651432921593012275157146431886225597936977286944214081623051215720172928118415772138687731430382795105524352709730894410749619582727395293560467726776607154653344689069748325520783793046278268417214857568022519720348777990549245307563007235878967239853404657432739248617369401613412255775972622651500848679828690159246202719195699050229683682647690402306373009376735213034424201965620226088889202645389546822920698807192029500887337968244655336992489435007048889170201007854170822579523483909521108909770911216208512009764733812213983322811060008298939259133843377761654405368479964650041306977068708658148713750248565274771452944193619618600322917775706796412033023123204096385880883918428954822560503117027658771871504304640715953595962621105525929760521047219435215441230817637830597872204276227006717347495180608781981515407966685857058701272066142263781770979668741627603606421012729301419859545051369662363361499407343582484295408233021569768619856161040462956607479008656905964395123459981749032349205208222479390949718889992474802831067358876241768252734863066520541720935908666237780468377635651379058927259197895617030716511710758951208859532331216917962485148868903951940963741942569442398629623960930125550273893112700090612569212891284684080534123067002494864017495376877222437065304517924908448453744316460335286068817912438888767176949085796149897255258544837481113654533742949300924798227274813869495004981983748787534958275753840598485222537599820503189459408587922911967815010044410261905772220380049944748945510761349792758719486505967472554586984628817361444157817603758272167513051936194901887798148365235165318898816719871506447972960710779524342675694498625468251969981542935236341641535611411739533 Il y a 216 chiffres 7 et seulement 188 chiffres 1... et tout le monde s'en fout.

On initialise : $2^{3^0}+1=3$ | $2^{3^1}+1=9$ | $2^{3^2}+1=513=27\times 19$

On met en forme au rang n, en notant $a_n=2^{3^n}+1$ et $b_n=3^{n+1}$. On suppose donc qu'il existe c_n vérifiant $a_n=b_n.c_n$.
On doit calculer $a_{n+1}=2^{3^{n+1}}+1=2^{3^n.3}+1=(2^{3^n})^3+1=(a_n-1)^3+1$.
On remplace $a_{n+1}=(3^{n+1}.c_n-1)^3+1=(3^{n+1})^3.(c_n)^3-3.(3^{n+1})^2.(a_n)^2+3.(3^{n+1}).(a_n)-1+1$.
Le terme $3^{3.n+3}.(c_n)^3$ se factorise par 3^{n+2}. Le terme $3^{2.n+3}.(a_n)^2$ se factorise par 3^{n+2}. Le terme $3^{n+2}.(a_n)$ se factorise aussi par 3^{n+2}. La somme est un multiple de 3^{n+2}. La récurrence s'achève.

3
9
513
134 217 729
2 417 851 639 229 258 349 412 353
14 134 776 518 227 074 636 666 380 005 943 348 126 619 871 175 004 951 664 972 849 610 340 958 209

J'ai appris avec un physicien	J'ai cru apprendre, mais c'est avec mes pieds, j'ai rien compris	J'ai pas besoin d'apprendre, je suis déjà matheux

```
def test(N):
....for n in range(N):
........an = 2**(3**n)+1
........bn = 3**(n+1)
........if an%bn != 0:
............return(False)
....return(True)
```

```
def test(N):
....for n in range(N):
........an = 2**(3**n)+1
........bn = 3**(n+1)
........if an%bn != 0:
............return(False)
........else:
............return(True)
```

```
def test(N):
....a = 3
....b = 3
....for n in range(N):
........a = a-1
........a = a*a*a+1
........b = b*3
........if a%b != 0:
............return(False)
....return(True)
```

Il existe aussi une méthode qui n'utilise pas la division euclidienne mais fait des divisions par 3 en boucle arretée à n.

Un fichier appelé FRENCH.TXT, situé dans le répertoire actif, contient tous les mots du dictionnaire français. Écrivez un script Python qui en profite alors pour afficher la liste de tous les mots contenant au moins une fois chaque voyelle 'a', 'e', 'i', 'o', 'u' (comme "OISEAU" ou "ABSOLUTISME" ou même "TAMBOURINER").

```
import os
Dico = open('FRENCH.TXT', 'r')
MotsVoyelles = []
for mot in Dico :
....if 'a' in Mot and 'e' in Mot and 'i' in Mot and 'o' in Mot and 'u' in Mot :
........MotsVoyelles.append(Mot)
print(MotsVoyelles)
```

ACCENTUATION, AUTOCUISEUR, SOUVERAIN, SAOUDITE, SAPEUR-POMPIER... mon dictionnaire en a trouvé 375, mais il n'est pas complet.
Et j'ai même APOCALYPTIQUE, AÉRODYNAMIQUE, YOUGOSLAVIE et TYPOGRAPHIQUE qui contiennent même le y.

Un ADONIS DANOIS veut ACHETER un HECTARE pour la NICHE de son CHIEN. Un AMATEUR de MEETINGS (ou un MARTEAU dans un GISEMENT ?) Un ARBITRE va s'ABRITER pour échapper à l'AIGLE AGILE qui fait ALLIANCE avec une CANAILLE. Ah il y en a des couples de mots anagramme l'un de l'autre.
Ouvrez le fichier `FRENCH.TXT` et cherchez les couples d'anagrammes. Vous avez le droit de convertir les chaines en listes (`list(Mot)`), de trier les listes (`sorted(L)`).

Comment savoir rapidement si deux listes sont égales sans être forcément dans le même ordre ? On les trie toutes les deux, et on regarde si les listes triées sont égales. Il y a aussi des solutions avec les ensembles, en utilisant une différence symétrique.
On va ouvrir le fichier, le lire. Comme on va le lire deux fois, on le stocke dans une variable avec `readlines()`.
On transforme les mots en listes une bonne fois pour toutes avec `list(Mot)`.
On ne testera ensuite que les mots de même longueur.
Comme la fonction `sorted` est un peu lourde, on évitera de trop la solliciter.
Et si finalement on la sollicitait une seule fois pour chaque mot, en créant une liste des mots triés, en gardant une liste des mots véritables.
On évitera aussi de dire qu'un mot est un anagramme de lui même. Et on évitera de tester ASPIRANT avec PARTISAN puis PARTISAN avec ASPIRANT, on ne va donc parcourir qu'une moitié de produit cartésien : `for k in range(len(L))` suivi de `for i in range(k)` tout court.

```
from os import * #tiens, que devient Julien ?
Dico = open('FRENCH.TXT', 'r')
DicoMots = [ ] #pour les mots proprement écrits
DicoSorted = [ ] #pour les mots triés
for mot in Dico : #on lit le dictionnaire
....DicoMots.append(mot) #on remplit le dictionnaire des mots lisibles
....MotListe = sorted(list(Mot)) #on crée l'anagramme trié
....DicoSorted.append(MotListe) #on le mémorise
Dico.close() #on est propre, on ferme
for k in range(len(DicoSorted)) : #on va lire le dictionnaire trié
....Mot1 = DicoSorted[k] #on prend le mot pour ne pas le lire et le relire
....for i in range(k) : #on ne regarde que les mots qui précèdent
........Mot2 = DicoSorted[i] #on lit le mot
........if len(Mot1) == len(Mot2) : #on ne regarde que si ils ont la même longueur
............if Mot1 == Mot2 : #on compare les mots triés
................print(DicoMots[i], DicoMots[k]) #on affiche les mots sous forme lisible
print('Et voila, fini !') #on prévient qu'on a fait le tour
```

Écrivez un script `Pipython` qui cherche dans `French.txt` le mot *(les mots)* contenant le plus de lettres `i`.

```
import os
Fichier = open('french.txt','r')
MotRecorc, Record = ' ', 0
for Mot in Fichier :
....Compt = Mot.count('i')
....if Compt > Record :
........MotRecord = Mot
........Recrod = Compt
print('record ', Record)
```

Du classique

Recors pour i : indivisibilité.

Et pour d'autres lettres : usurpateur, tartelette, baobab, concupiscence...

Si `count` est interdit :

```
Compt=0
for lettre in mot :
....Compt+=
int(lettre=='i')
```

Si on veut la liste des records, on fait deux tests :

```
if Compt == Record :
....MotsRecod.append(Mot)
if Compt > Record :
....MotsRecord = [Mot]
....Record = Compt
```

Un fichier contient un texte `MAVIEMONOEUVRE.TXT` que vous devez censurer. Un autre fichier `INDEX.TXT` contient le dictionnaire des mots interdits (ceux qu'on met à l'index). Ces mots sont au singulier, et masculin si il s'agit d'adjectifs. Écrivez un programme qui lit le texte, le découpe mot à mot (un mot est entre deux espaces ou deux ponctuations) et indique si plus de cinq pour cent des mots sont dans l'index de la censure. Variante : il remplace les mots à censurer par un mot tiré au hasard dans un fichier `NOMSDOISEAUX.TXT`.

Vous ne croyez quand même pas que je vais moi même me faire l'auxilliaire de la censure !