M.P.S.I.2 2014 **Un jour en janvier** 2015 Charlemagne — 1-phô

MPSI 2/2014 GÉNÉALOGIE 1-phô

♡1 Je vous donne l'arbre généalogique de la famille *(codé comme l'arborescence des dossiers/répertoires d'un disque dur)*. Complétez *(éventuellement sur ce document)* ce qui manque : •1 pt.•

nom	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
indice montant	8	6	5	21	19	25	0	1	18	14	2	3	27	29	26		10
indice descendant	9	7	12	22	20	32		24	23		13	4	28	30	31	16	11

♡2 Qui est l'ancêtre commun à tout le monde ? •1 pt.•
Combien a-t-il de descendants directs et indirects ? •1 pt.• Combien a-t-il d'enfants ? •1 pt.•

♡3 Qui sont les individus sans descendants ? •1 pt.•

♡4 Combien `10` a-t-il d'enfants, et combien de descendants ? •1 pt.•

♡5 Donnez le père, le grand-père, et les aïeux successifs de `3`. •1 pt.•

♡6 Si `8` a un nouvel enfant, pouvez vous indiquer le script qui va modifier les deux listes "indices montants" et "indices descendants". •1 pt.•

MPSI 2/2014 TUYAUX 1-phô

Un lemme mathématique pour commencer : on donne quatre réels r, R, b et B vérifiant $r \leqslant R$ et $b \leqslant B$. Montrez : $|r-b| + |R-B| \leqslant |r-B| + |R-b|$. •2 pt.•

```
-----+----+-----------+--------+---------------- ou -----+----+-+-----------------+----------------
.....r....b...........R........B..................  ou .....r....R.b.................B.................
```

Vous disposez d'un stock de `N` tuyaux rouges de tailles diverses stockées dans une liste nommée `R`, et de `N` tuyaux bleus de longueurs diverses aussi stockées dans une liste nommée `B`. Exemple : `R=[5, 12, 8, 16]` et `B=[6, 8, 13, 20]`.
Vous devez les vendre par lots de deux : un rouge un bleu. Mais dans chaque lot, les deux tuyaux doivent être de même longueur *(il peut s'agir de tuyaux eau chaude/eau froide)*. Il va donc faloir couper soit le bleu *(exemple : `R[1]` couplé avec `B[3]`)*, soit le rouge *(exemple : `R[3]` couplé avec `B[2]`)*, soit aucun *(exemple : `R[2]` couplé avec `B[1]`)*.
Votre objectif : avoir le moins possible de pertes, c'est à dire trouver σ pour minimiser $\sum_{k=0}^{N-1} |R[K] - B[\sigma(k)]|$.

♠1 Ecrivez la procédure `pertes` qui pour deux listes `R` et `B` et une permutation `sigma` *(donnée aussi sous forme de liste)* calcule la somme indiquée ci dessus. •2 pt.•
Dans la mesure du possible, votre procédure devra retourner le booléen `False` si les trois listes n'ont pas la même longueur.

♠2 S'il faut tester toutes les permutations possibles, si le calcul de $\sum_{k=0}^{N-1} |R[K] - B[\sigma(k)]|$ prend 10^{-10} seconde, mais que N vaut 20, combien de temps vous faudrait il pour étudier tous les cas ? •2 pt.•

2^{20}	20!	20^{20}	20^{10}	
10^6	10^{18}	10^{26}	10^{13}	

♠3 Prouvez que la perte minimale pour notre exemple est de 6. •3 pt.•

♠4 Prouvez que pour minimiser la perte, il faut placer apparier le tuyau rouge le plus court avec le tuyau bleu le plus court. •3 pt.•

♠5 Déduisez que la perte est minimale si les deux listes sont triées dans le même ordre. •2 pt.•

♠6 Ecrivez une procédure qui donne la liste des couples `[R[k], B[sigma(k)]]` pour `k` de `0` à

N-1 en supposant que vous disposez de la méthode `sort` qui trie une liste *(syntaxe : `tamairen.sort()` si la liste s'appelle `tamairen`)*[1]. •3 pt.•

♠7 Ecrivez une procédure *(même peu efficace)* qui donne cette liste des couples si vous n'avez pas la métode `sort`. •3 pt.•

MPSI 2/2014 **SCRIPT** 1-phô

♣1 Que va faire ce script : •5 pt.•

```
from random import randrange
def leurdeschamps(L) :
....if len(L) == 1 :
........return(L[0])
....else :
........demi = len(L)//2
........a = leurdeschamps(L[0 :demi])
........b = leurdeschamps(L[demi, len(L)])
........if a > b :
............return(a)
........else :
............return(b)
LL = [randrange(1000) for k in range(10000)]
print(leurdeschamps(LL))
```

Votre note sur cette question dépendra de votre rédaction, explication, justification.

MPSI 2/2014 **BASES DE BONNETS** 1-phô

On envisage une base de données pour la gestion des Prépas du lycée faite de deux tables : `eleves` et `professeurs`. Les champs de la table `eleves` sont les suivants :

champ	nom	prenom	date_naissance	classe	LV1	LV2	option	regime
type	char(20)	char(20)	date	char(5)	char(8)	char(8)	char(8)	char(12)
exemple	HARTY	Kenty	data error	MPSI2	anglais	francais	Info	externe

Ceux de la table `professeurs` sont les suivants :

champ	nom	prenom	MPSI1	MPSI2	PCSI	MP	MP*	PC	PC*	matieres	date_naissance
type	Char(20)	char(20)	int(1)	int(1)	int(1)	int(1)	int(1)	int(1)	int(1)	char(30)	date
exemple	Papanico	Bob	1	1	1	1	1	0	0	SII	

On vous demande de saisir diverses requêtes, à vous de les formuler en langage SQL :

nombre d'élèves de sup faisant anglais langue vivante 1
nom des élèves de PCSI externes
nom, prénom et classe des élèves de Spé ayant moins de 18 ans
liste des élèves prénommés Alexandre ou même Alex, Alexis..., triés par classe
liste des professeurs de la MP*
liste des professeurs de la filière MP (quatre classes) triés par âge
liste nominale des professeurs de mathématiques
liste des professeurs de l'élève Lebanc
vérifier qu'aucun élève de PCSI n'a pris option informatique
envoyer tout de suite en MP* les élèves prénommé(e)s Lucille

Rappel des mots du langage SQL : `SELECT, ORDER BY, WHERE, FROM, COUNT, MAX, MIN, AVG, UPDATE, DELETE.`

[1] oui, "ta mère en short"

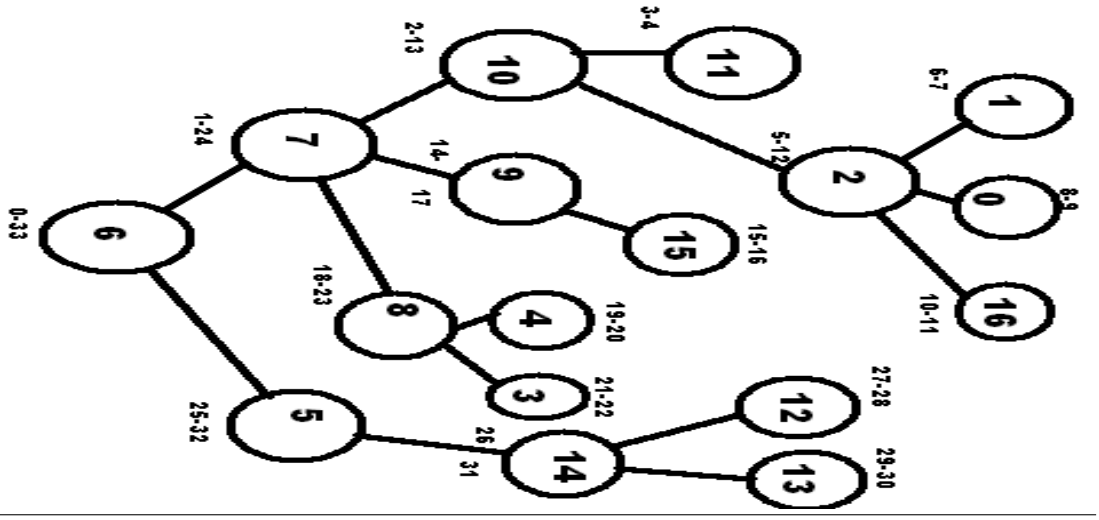
2014-15

Un arbre généalogique, ou l'arborescence d'un disque dur. 1-phô

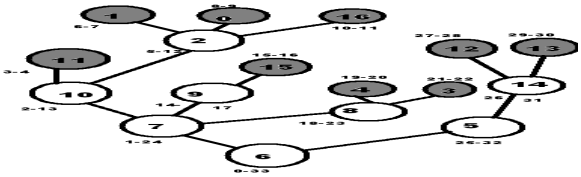
L'ancêtre est évidemment celui qui a l'indice ascendant égal à 0. C'est 6.
Son indice descendant est le double du nombre de personnes dans la famille puisque chaque peronne a deux indices au total. On complète donc : 33 comme indice descendant *(et on ne descend pas plus bas)*.

Il manque encore dans le tableau deux indices : 15 et 17
On a deux cases où les mettre.
Comme l'indice ascendant est plus petit que l'indice descendant, la seule possibilité est

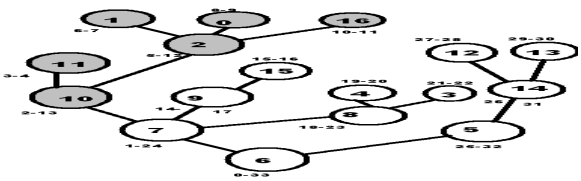
nom	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
indice montant	8	6	5	21	19	25	0	1	18	14	2	3	27	29	26	15	10
indice descendant	9	7	12	22	20	32	33	24	23	17	13	4	28	30	31	16	11



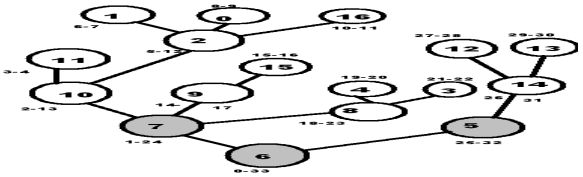
Les individus n'ayant pas de descendants sont ceux pour lesquels la différence d'indices est de 1. Il s'agit donc de 0, 1, 3, 4, 11, 12, 13, 15, 16.



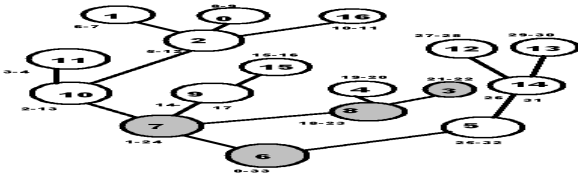
L'individu 10 a des enfants puisque la différence d'indices est plus grande que 1. On la calcule : $13 - 2 = 11$. Il a cinq descendants.
On les détecte à leurs indices compris entre 2 et 13 : 0, 1, 2, 11 et 16.
Parmi eux, seul 2 n'est pas en fin de branche, et a des enfants : 0, 1 et 16.
Par élimination, 11 est descendant direct de 10. On résume : 10 a deux enfants : 2 et 11. Et il a trois petits enfants : 0, 1 et 16 tous descendants de 2.



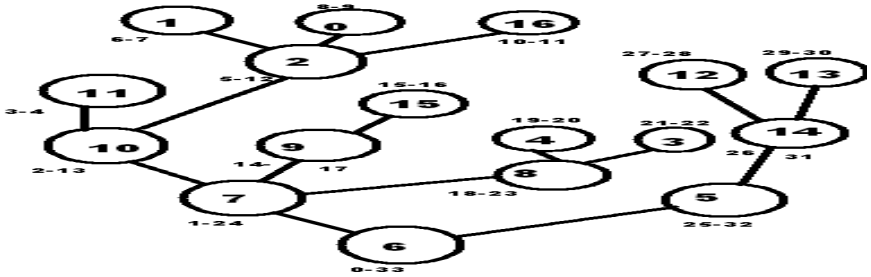
On cherche les descendants directs de la racine 6.
Déjà, il y a 7 *(indice montant 1)*. Son indice descendant est 24. Ceux d'indice entre 2 et 23 sont dans la brache de 7.
On cherche qui a pour indice montant 25. C'est le deuxième enfant de 6. Et c'est 5.
Son indice descendant est 32. On est donc en fin de liste. Il n'y a plus personne au delà.
Les deux enfants du grand aïeul 6 sont 7 et 5.



On retrouve la généalogie de 3. Son indice descendant est 22. Or, un individu a pour indice descendant 23. C'est donc son père *(et il était le cadet de ce père nommé 8)*.
On cherche si quelqu'un a pour indice descendant 24. C'est 7, qui est donc son grand père.
On continue : 25 n'est indice descendant de personne, mais au contraire indice montant de 5. C'est donc que 5 est le frère de 7.
On est donc au même rang. Tous ceux dont l'indice est entre 25 et 32 *(indice descendant de 5)* ne sont donc que des cousins lointaints de 3.



On pouvait aussi reconstituer l'arbre par ses indices et tout lire visuellement.



L'élément 8 a un nouvel enfant. Celui ci aura donc pour indice montant 23 et pour indice descendant 24.
8 garde son indice montant, et son indice descendant devient 25.
Il en va de même de tous les indices qui suivaient, ils sont tous augmentés de 2.
Par exemple, 14 aura pour nouveaux indices 28 et 33.
Par exemple, 7 garde son indice montant 1 et adopte pour indice descendant 26.
Si l'on considère que les indices montants et descendants sont codés dans deux listes de noms naturels :

```
for individu in arbre :
....if montant[individu] > 22 :
........montant[individu] += 2
....if descendant[individu] > 22 :
........descendant[individu] += 2
```

Et on n'oublie pas `montant[17], descendant[17] = 23, 24` pour le petit nouveau.

2014-15

Des tuyaux. 1-phô

On commence par le lemme mathématique qui servira sans doutes plus loin.

On a donc quatre réels $r \leqslant R$ et $b \leqslant B$. On mesure des distances.
On peut utiliser l'inégalité triangulaire, mais il y a quand même deux termes dans chaque membre, alors que l'inégalité triangulaire crée des termes en plus dans un membre.
On va étudier les différents cas de figure sur l'ordre de ces quatre nombres. Fort heureusement il n'y a pas 24 cas puisque certains ordres sont imposés.

	$d = \lvert r-b\rvert + \lvert R-B\rvert$	$\delta = \lvert r-B\rvert + \lvert R-b\rvert$	différence $\delta - d$
$r \leqslant R \leqslant b \leqslant B$	$b-r+B-R$	$B-r+b-R$	0
$r \leqslant b \leqslant R \leqslant B$	$b-r+B-R$	$B-r+R-b$	$2.(R-b) \geqslant 0$
$r \leqslant b \leqslant B \leqslant R$	$b-r+R-B$	$B-r+R-b$	$2.(R-b) \geqslant 0$
$b \leqslant B \leqslant r \leqslant R$	$r-b+R-B$	$r-B+R-b$	0
$b \leqslant r \leqslant B \leqslant R$	$r-b+R-B$	$B-r+R-b$	0
$b \leqslant r \leqslant R \leqslant B$	$r-b+B-R$	$B-r+R-b$	$2.(R-r) \geqslant 0$

On constate que dans tous les cas, la différence est positive.
On a bien $\lvert r-b\rvert + \lvert R-B\rvert \leqslant \lvert r-B\rvert + \lvert R-b\rvert$.
Si il s'agit de mesurer des distances, mieux vaut regrouper les deux petits entre eux et les deux grands entre eux.

Si les listes des longueurs et de la permutation ont la même longueur, on a juste à cumuler une somme :

```
def pertes(R, B, sigma) :
....S = 0
....for k in range(len(L)) :
........S = S+abs(R[k]-B[sigma[k]])
....return(S)
```

La fonction `abs` est d'accès direct. Si vous aviez des doutes, il fallait l'importer du module `math`.
On sollicite un indice $\sigma(k)$ en allant chercher un terme d'indice k dans une liste `sigma`, d'où `sigma[k]` avec des crochets et non `sigma(k)` qui correspondrait à l'appel d'une fonction ou procédure.
Enfin, on rappelle que le résultat d'une procédure *(introduite par def)* est retourné par `return`. On en fait ensuite ce qu'on veut. Ce n'est pas du tout un print qui affiche un résultat mais qu'on ne peut exploiter, sauf si on a des yeux et un crayon, ce qui n'est pas le cas de l'ordinateur.

Si l'on veut tester que les listes ont la même longueur :

```
def pertes(R, B, sigma) :
....if not(len(R) == len(B) and len(L) == len(sigma)) :
........return(False)
....else :
........S = 0
........for k in range(len(R)) :
............S = S+abs(R[k]-B[sigma[k]])
........return(S)
```

Un programmeur zélé vérifiera que tous les indices `sigma[k]` sont aussi compris entre `0` et `N`.

Un petit calcul d'ordre de grandeurs avant de se ruer dans la programmation.
Si on a 20 tuyaux de chaque sorte, les listes sont de longueur 20. On a 20! permutations possibles.
Chaque calcul prend 10^{-10} seconde(s). On a donc besoin de $20!/10^{10}$ secondes.
On trouve un nombre de l'ordre de 10^8 secondes. On divise par 60 pour avoir des minutes, et encore par 60 pour avoir des heures, puis par 24 et 365. On trouve un résultat de l'ordre de $\boxed{7 \text{ ans}}$
Ce serait un peu idiot de lancer un ordinateur sur un tel calcul pour vendre des bouts de tuyau...

On regarde notre exemple :

rouges	=====	==========	========	================
bleues	======	========	==========	==================

On prend la permutation identité et on mesure les chutes :

rouges	=====	==========	========	================
bleues	======	========	==========	==================
14	=	====	=====	====

On trie les deux listes et on apparie :

rouges	=====	========	==========	================
bleues	======	========	==========	==================
6	=		=	====

On constate que la valeur 6 est une valeur possible. Pourquoi ne peut on avoir moins ?

Une solution consiste à tester les vingt quatre permutations et à calculer pour chacune la perte totale. C'est évidemment une méthode un peu lourde.

On comprend qu'il vaut mieux apparier le tuyau rouge 5 avec le tuyau bleu 6 pour optimiser les pertes. On se doute bien que mettre le 5 avec le 20 créerait une chute de 15 fort peu judicieuse *(même si finalement on pourrait peut être tenter d'apparier cette chute avec le morceau de* 13*, mais on ne part pas dans cette voie de redécoupages).*
Il faut quand même le prouver.
Supposons que le 5 rouge soit avec un autre tuyau bleu α *(*$\alpha \in \{8,\ 13,\ 20\}$*).* Le 6 bleu est alors avec un rouge a *(*$a \in \{8,\ 12,\ 16\}$*).* On va montrer que la perte est alors plus grande que dans le cas 5 avec 6 *(et* a *avec* α*)* : $\begin{array}{|c|c|c|}\hline 5 & a & \text{autres}\\ \hline \alpha & 6 & \text{autres}\\ \hline\end{array}$ contre $\begin{array}{|c|c|c|}\hline 5 & a & \text{autres}\\ \hline 6 & \alpha & \text{autres}\\ \hline\end{array}$

Les deux pertes sont $|5-\alpha|+|a-6|+\sum_{autres}|rouge-bleu|$ et $|5-6|+|a-\alpha|+\sum_{autres}|rouge-bleu|$.
Comme on a deux fois le même terme $\sum_{autres}|rouge-bleu|$ il suffit de comparer $|5-\alpha|+|a-6|$ et $|5-6|+|a-\alpha|$.

Notre lemme permet de conclure : la configuration $\begin{array}{|c|c|c|}\hline 5 & a & \text{autres}\\ \hline 6 & \alpha & \text{autres}\\ \hline\end{array}$ minimise les pertes.

On recommence en comparant la liste $\begin{array}{|c|c|c|}\hline 5 & 8 & 12,\ 16\\ \hline 6 & 8 & 13,\ 20\\ \hline\end{array}$ avec $\begin{array}{|c|c|c|}\hline 5 & 8 & 12,\ 16\\ \hline 6 & 13 & 8,\ 20\\ \hline\end{array}$ ou $\begin{array}{|c|c|c|}\hline 5 & 8 & 12,\ 16\\ \hline 6 & 20 & 13,\ 8\\ \hline\end{array}$.
C'est le même lemme qui montre qu'il vaut mieux choisir le premier modèle.

On termine avec $\begin{array}{|c|c|c|c|}\hline 5 & 8 & 12 & 16\\ \hline 6 & 8 & 13 & 20\\ \hline\end{array}$ meilleure que $\begin{array}{|c|c|c|c|}\hline 5 & 8 & 12 & 16\\ \hline 6 & 8 & 20 & 13\\ \hline\end{array}$.

Bref, c'est bien la configuration $\begin{array}{|c|c|c|c|}\hline 5 & 8 & 12 & 16\\ \hline 6 & 8 & 13 & 20\\ \hline\end{array}$ qui optimise, avec perte $1+0+1+4$.

On reprend le raisonnement ci dessus pour montrer qu'on a intérêt à mettre ensemble le plus petit rouge et le plus petit bleu.

On note r le plus petit rouge et b le plus petit bleu. Si on ne les met pas ensemble $\begin{array}{|c|c|c|}\hline r & a & \text{autres}\\ \hline \alpha & b & \text{autres}\\ \hline\end{array}$.

Or, avec $\begin{array}{|c|c|c|}\hline r & a & \text{autres}\\ \hline b & \alpha & \text{autres}\\ \hline\end{array}$, le lemme donne $|r-b|+|a-\alpha|+autres \leqslant |r-\alpha|+|a-b|+autres$.
On comprend qu'il vaut mieux coupler les deux plus petits (Julie et Iqbal ?).
On mes met de côté, et on recommence avec ce qu'il reste.
Etape par étape, par récurrence, on montre qu'il faut apparier le i^{eme} tuyau rouge r_i (par ordre croissant) avec le i^{eme} tuyau bleu.

On passe à l'algorithme dans le cas où l'on dispose de la méthode `sort`.

```
R.sort()
B.sort()
S = 0
for k in range(len(R)) :
....S += abs(R[k]-B[k])
print(S)
```

Dans le cas où on ne dispose pas de la procédure `sort`, on va la reconstruire. Et tant pis si la manière n'est pas efficace, puisque on ne va pas travailler sur des listes de grande longueur.
On créer une petite procédure qui prend une liste, cherche le plus petit élément et l'efface de la liste :

```
def extrait_min(L) :
....LL = [] #on initialise une liste vide qu'on va remplir peu à peu
....mini = L[0]
....for k in range(1, len(L)) : #on va parcourir la liste L (sans son premier terme déjà regardé)
........elt = L[k] #on prend donc un élément de L d'indice non nul
........if elt < mini : #si on détecte une descente...
............LL.append(mini) #on oublie le minimum précédent, donc on le met dans la liste LL
............mini = elt #on mémorise le nouveau minimum
........else : #si on n'a pas détecté une descente
............LL.append(elt) #on transfère dans LL le nouveau nombre sans intérêt
....return(mini, LL) #le minimum est mémorisé, et la liste est un peu plus courte
```

On note qu'avec cette procédure, la liste `LL` contient bien tous les éléments de `L` sauf son minimum, mais l'ordre a été modifié.

On peut alors mettre en boucle cette procédure pour `R` et `B` :

```
n = len(R) #on mémorise la longueur des listes car on va les modifier
pertes_min = 0 #on initialise une somme nulle qu'on va augmenter au fur et à mesure
for k in range(n) : #on va solliciter extrait_min n fois
....a, R = extrait_min(R) #on sort le minimum des listes déjà réduites R et B
....b, B = extrait_min(B) #et on racourcit ces deux listes
....perte_min = perte_min + abs(b-a) #on calcule un terme de plus de la somme
print(perte_min) #on affiche la perte totale du cas optimisé
```

2014-15

Un script sur des listes aléatoires. 1-phô

On analyse diverses parties du script qui est formé d'une importation, de la définition d'une procédure visiblement récursive *(elle se rappelle elle même)*, de la création d'une liste et de la sollicitation de la procédure.

```
from random import randrange
```

On importe la fonction qui engendre des nombres "aléatoires".

```
LL = [randrange(1000) for k in range(10000)]
```

On crée justement une liste aléatoire de 10.000 termes entre 0 inclus et 1000 exclu.

```
print(leurdeschamps(LL))
```

On sollicite la procédure sur la liste qu'on vient de créer. On aurait pû compacter en

```
print(leurdeschamps([randrange(1000) for k in range(10000)]))
```

On définit donc une procédure qui prendra en variable un certain objet `L` qui sera une liste si l'on en croit la suite du programme mais aussi le test `len(L) == 1`.

```
def leurdeschamps(L) :
....if len(L) == 1 :
........return(L[0])
```

Si la liste est de longueur 1, on renvoit son élément d'indice 0 c'est à dire son unique élément.
Sinon, on se lance dans une suite d'instructions.

```
........demi = len(L)//2
........a = leurdeschamps(L[0 :demi])
........b = leurdeschamps(L[demi, len(L)])
```

On sollicite la procédure sur les deux demi-listes `L[0 :demi]` et `L[demi, len(L)]`, et on stocke les résultats dans deux variables `a` et `b`.

```
........if a > b :
............return(a)
```

```
........else :
............return(b)
```
On retourne le maximum de `a` et de `b` par simple test.
On comprend qu'on commence donc à parler de maximum.
Si la liste a deux éléments `[x, y]`, on coupe en deux et on sollicite la procédure pour les deux listes `[x]` et `[y]`. Elle renvoit les deux nombres `x` et `y`, qu'elle compare et elle renvoit alors le plus grand des deux.
Si la liste a quatre éléments `[x, y, z, t]`, elle coupe en deux et calcule `leurdeschamps([x, y])` ainsi que `leurdeschamps([z, t])`. Elle retourne donc $Max(x, y)$ et $Max(z, t)$ en vertu de l'étude précédente. Elle les compare et retourne finalement $Max(Max(x, y), Max(z, t))$, c'est à dire $Max(x, y, z, t)$.
On comprend qu'elle détermine donc le maximum des éléments d'une liste.
Si il convient de vraiment démontrer ce résultat, on le fait par récurrence forte sur les nombre d'élements de la liste.
C'est initialisé. Supposons ensuite que la propriété soit vraie pour toute liste de longueur inférieure ou égale à `n`. On donne une liste de longueur `n+1`. Elle se coupe en deux listes `L[0 :demi]` et `L[demi,n+1]`. Comme chacune est de longueur inférieure ou égale à `n`, les deux nombres `a=leurdeschamps(L[0 :demi])` et `b=leurdeschamps(L[demi,n+1])` sont les deux maxima `Max(L[0],...L[demi-1])` et `Max(L[demi],...L[n])`. Le test if `a>b` va donc retourner au final le plus grand des deux, c'est à dire `Max(L[0],...L[n])`.
Ceci achève la récurrence.
C'est ce type de démonstration qu'on vous demandera en option informatique.
Et le principe de cette démarche qui coupe le tas en deux et recommence s'appelle "*diviser pour régner*". Bref, on tire une liste aléatoire et on en cherche le maximum.

nombre d'élèves de sup faisant anglais langue vivante 1
`SELECT COUNT(*) FROM eleves WHERE LV1="Anglais" OR LV1="ANGLAIS"`
nom des élèves de PCSI externes
`SELECT nom FROM eleves WHERE classe="PCSI" AND regime="externe"`
nom, prénom et classe des élèves de Spé ayant moins de 18 ans
`SELECT nom, prenom, classe FROM eleves WHERE date_naissance<(calcul) AND classe classe = "MP" OR classe = "MP*" OR classe = "PC"OR classe = "PC*"`
liste des élèves prénommés Alexandre ou même Alex, triés par classe
`SELECT nom, prenom FROM eleves WHERE prenom LIKE "%Alex%" SORT BY classe`
liste des professeurs de la MP*
`SELECT nom FROM professeurs WHERE MP*=1`
liste des professeurs de la filière MP (quatre classes) triés par âge
`SELECT nom, prenom FROM professeurs WHERE classe IN {MPSI1, MPSI2, MP, MP*} SORT BY date_naissance`
liste nominale des professeurs de mathématiques
`SELECT nom FROM professeurs WHERE matiere LIKE "%math%"`
liste des professeurs de l'élève Lebanc
`SELECT nom, prenom, classe FROM eleves WHERE nom="Leblanc" or nom="LEBLANC"` `SELECT nom FROM professeurs WHERE (...)=True`
vérifier qu'aucun élève de PCSI n'a pris option informatique
`SELECT COUNT(*) FROM eleves WHERE classe="PCSI" AND option="Info"`
envoyer tout de suite les élèves prénommées Lucille en MP*
`UPDATE eleves SET classe="MP*" WHERE prenom="Lucille"`

MPSI 2/2014

SPEED DATING

1-phô

Explication de la situation modélisée :
Vous êtes un employeur qui doit embaucher un nouvel employé, à partir d'un groupe de N candidats.
Ou alors un séducteur *(une séductrice)* qui doit choisir l'âme soeur dans un panel de prétendantes *(prétendants)*.
Chaque candidat a une valeur entre 0 et 1000, et vous devez choisir dans la mesure du possible celui qui a la valeur la plus élevée.
Facile allez vous me dire : il suffit de prendre celui dont la note est la plus haute.
Oui, c'est d'ailleurs ce que nous allons faire dans un premier temps.

Mais il y a un problème : vous recevez les candidats un par un, et vous devez donner votre réponse à chacun à tour de rôle : éliminé ou choisi *(et si vous optez pour "choisi" vous arrêtez alors la sélection)*. Comment alors savoir si vous avez en face de vous le meilleur postulant, sachant que vous ne pouvez le comparer qu'à ceux qui l'ont précédé et pas encore à ceux qui suivront. Et si vous vous rendez compte qu'il était meilleur que ceux qui suivent, c'est trop tard, vous l'avez éliminé...

On va alors recourir à l'algorithme qui est le plus classiquement utilisé :
• vous auditionnez une partie des candidats sans en prendre aucun *(tant pis pour eux)* [2]
• vous regardez dans cette liste de candidats alors éliminés qui avait la plus grande valeur notée S
• vous auditionnez alors les candidats suivants,
◦ vous retenez le premier qui dépasse la valeur de seuil S
◦ si aucun n'a dépassé cette valeur, vous prenez le dernier *(car il faut bien faire un choix)*

Il s'avère que cet algorithme est assez efficace dans la grande majoroté des cas *(donnez un exemple de liste de candidats pour lequel il ne le serait pas* •1 pt.• *)*. La démonstration ne sera pas faite ici, car elle dépasse les compétences d'un élève de Sup. Mais on va modéliser la situation et mettre en place l'algorithme.

MPSI 2/2014

Modélisation

1-phô

Il faut d'abord créer la liste `L` des valeurs des `N` candidats *(valeurs de* 0 *à* 1000*)*, tirées aléatoirement.
On rappelle qu'il existe en Python un module appelé `random` dans lequel il y a une fonction appelée `randrange` *(l'instruction `randrange(k)` tire un entier "au hasard" entre* 0 *et* $k-1$*)*. Au fait, que devez vous écrire pour importer cette fonction et l'utiliser ?
Pourquoi le programme suivant n'est il pas le bon ? •1 pt.•

```
L = []
a = randrange(1000)
for i in range(N) :
....L.append(a)
print(L)
```

sachant que `oirgéo.append(teuse)` ajoute l'élément `teuse` à la fin de la liste `oirgéo`.[3]
Donnez un script qui crée donc une liste `L` convenable. •1 pt.•

On va avoir besoin d'une procédure qui cherche le plus grand élément d'une liste, en la parcourant en entier. Je propose :

[2] en l'occurence, les $\sqrt{N}$ premiers candidats, sachant que N ne vaut pas 1
[3] le fait de détecter les jeux de mots stupides ne rapporte pas de points

```
def ilémilitair(liste) :
....for i in range(len(liste)) :
........if liste[i] > max :
............max = .... (*)
....return(max)
```

Qu'est ce qui ne va pas dans ce programme ? Que faut il préciser sur la ligne (*) ? •2 pt.•
Comment pouvez vous le perfectionner pour qu'il indique aussi en sortie le maximum et le nombre de fois où il est atteint ? •2 pt.•

MPSI 2/2014 **Algorithme** 1-phô

On crée donc comme ci dessus une liste `L` de `N` valeurs (`N` est fixé dès le début du programme, par exemple égal à 200).
Il faut trouver le maximum `S` des $\sqrt{N}$ premiers termes de la liste : écrivez ce script. •2 pt.•
Il faut ensuite parcourir la suite de la liste et prendre le premier indice `i` pour lequel on a `L[i]>S`.
Vous pourrez utiliser une boucle `for i in range(...,...)` *(sachant qu'avec* ***`range(d, f)`*** *on génére les indices de* ***`d`*** *à* ***`f-1`*** *)* dont vous pourrez sortir par un `return` en cours de boucle *(ce que je n'aime pas mais que je peux tolérer car vous n'avez qu'une heure)* ou par une boucle `while`.
Votre script devra afficher alors l'indice `i` du candidat retenu et sa valeur `L[i]` que l'on notera `selec`. •2 pt.•

Pour vérifier la pertinence de l'algorithme, écrivez le script qui compte alors combien d'éléments de la liste `L` étaient supérieurs à cette valeur `selec`. La valeur de ce décompte sera appelée `compt`. •2 pt.•

Pour aller plus loin, créez un programme qui simule un millier de telles situations et calcule le nombre de fois où la procédure a sélectionné le dernier candidat, ainsi que la valeur moyenne de `compt`. •4 pt.•

MPSI 2/2014 **EXERCICE** 1-phô

On définit la procédure suivante :

```
def armacidegard(n) :
....if n%2 == 0 :
........return(armaciedegard(n//2))
....elif n == 1 :
........return(True)
....else :
........return(False)
```

Indiquez la réponse pour chacun des entiers suivants :

7	12	16	28	2014	2048	2057

•2 pt.•
Que calcule cette procédure ? •2 pt.•

2014-15 1-phô

Génération de la liste des valeurs.

On étudie le script proposé :

```
L = [].............. crée une liste vide que l'on va agrandir pas à pas
a = randrange(1000).. tire un nombre au hasard
for i in range(N) :.. avance pas à pas de N valeurs
....L.append(a)...... agrandit la liste au fur et à mesure
print(L)............. affiche la liste obtenue
```

Première erreur sans importance : avec `randrange(1000)`, on tire un nombre entre 0 et 999 inclus. Si on veut des entiers inférieurs ou égaux à 1000, il faut utiliser `randrange(1001)`.
Grossière erreur ensuite : on tire un nombre au hasard une seule fois, et c'est toujours le même qu'on colle au bout de la liste.

```
L = []
for i in range(N) :
....a = randrange(1001)
....L.append(a)
print(L)
```

Si besoin est, on en fait une procédure :

```
def genere_liste() :
....L = []
....for i in range(N) :
........L.append(randrange(1001))
....return(L)
```

2014-15 1-phô

Récherche du maximum.

On avance pas à pas dans la liste. Si un élément dépasse le maximum déjà atteint, c'est lui qui remplace l'ancien maximum.
Problème : on n' a pas initialisé `max`. De ce fait, le compilateur va vous indiquer que la variable n'a pas été définie. Il faut donc lui donner une valeur d'entrée, qui ne perturbe pas la recherche du maximum.

```
def ilémilitair(liste) :
....max = 0
....for i in range(len(liste)) :
........if liste[i] > max :
............max = liste[i]
....return(max)
```

La valeur 0 est ici un bon choix, puisque l'on est sûr que tous les éléments de la liste sont plus grands que 0 par construction. Mais si c'est une liste dont on ne connaît pas l'intervalle dans lequel elle pioche, il vaut mieux initialiser avec la valeur **liste[0]**.
Un informaticien prendra la précaution de vérifier d'abord que la liste contient au moins un terme d'indice 0.
Sinon, on note l'inutilité du recours à un indice. On préfèrera :

```
def ilémilitair(liste) :
....max = liste[0]
....for elt in liste :
........if elt > max :
............max = elt
....return(max)
```

dans lequel on parcourt la liste par le simple **for elt in liste :** .

Si l'on doit aussi compter combien de fois le maximum est atteint, on crée un compteur *(appelé ici Grimm)* qu'on incrémente à chaque fois que la valeur **max** est atteinte. Une solution est

```
def ilémilitair(liste) :
....max = liste[0]
....for elt in liste :
........if elt > max :
............max = elt
....Grim = 0
....for elt in liste :
........if elt == max :
............Grim +=1
....return(max, Grim)
```

On rappelle que l'instruction **Grim +=1** remplace l'instruction **Grim = Grim+1** qui incrémente **Grim** de une unité.
Mais il vaut mieux ne parcourir qu'une fois la liste :

```
def ilémilitair(liste) :
....max, Grim = liste[0], 0
....for elt in liste :
........if elt = max :
............Grim +=1
........elif elt > max :
............max = elt
............Grim = 1
....return(max)
```

Il ne faut en effet pas oublier de remettre le compteur à 1 quand on recroise un élément égal à **max**.

2014-15
Algorithme du tiers de liste. 1-phô

On commence par extraire le maximum du début de liste :

```
max = L[0]
for i in range(int(sqrt(N))) :
....if L[i] > max :
........max = L[i]
```

Il faudra penser à importer la fonction `sqrt` du module `math` (`from math import sqrt`).

On parcourt ensuite la liste jusqu'à ce qu'on croise un terme plus grand que `max` :

```
for i in range(int(sqrt(N)), N) :
....if L[i] > max :
........break
print(i, L[i])
```

On sort par le `break`, et si on n'a pas croisé de terme plus grand que max, on termine la boucle for avec `i = N` et on a donc retenu le dernier candidat.
On pourra estimer que cette procédure favorise le dernier candidat, mais il n'y a quand même qu'une très faible probabilité que l'on aille jusqu'au bout de la liste sans avoir trouvé d'élément plus grand que max.
Il y a quand même le cas où le vrai maximum était dans le début de la liste. Exemple avec une liste de seize termes :
`[100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]` ou `[99, 100, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 0]`
Dans le deuxième exemple, `max` vaut 100. Tous les candidats à 99 sont éliminés, et en désespoir de cause, on prend le dernier...

Si on est dans le corps du programme, on fera un affichage avec `print`, mais si on est dans une procédure, on sortira avec `return(i, L[i])` :

```
for i in range(int(sqrt(N)), N) :
....if L[i] > max :
```

```
........return(i, L[i])
return(N, L[N])
```

On pourra de manière plus propre utiliser une boucle `while` :

```
i = int(sqrt(N))
while (L[i] < max) and (i < N-1) :
....i += 1
```

`print(i, L[i])` ou `return(i, L[i])`
On avance par le `i+=1` dans la liste tant qu'on n'a pas dépassé `max`, et tant qu'on n'est pas au bout de la liste.

2014-15
Evaluation de la pertinence de la méthode. 1-phô

Une fois qu'on a sorti la valeur `L[i]` qu'on affecte à `selec`, on doit voir combien de candidats avaient une valeur plus élevée que le candidat retenu.
Il faut donc parcourir la liste avec un compteur qu'on incrémente à chaque fois qu'on dépasse `selec` :

```
compt = 0
for elt in L :
....if L[i] > selec :
.......compt += 1
print('il y avait '+str(compt)+' candidats de plus grande valeur')
```

La procédure peut ensuite être mise en boucle pour voir la pertinence de la méthode.

```
total, on_a_pris_le_dernier = 0, 0
for i in range(1000) :
....L = genere_liste()
....max = L[0]
....for i in range(int(sqrt(N))) :
........if L[i] > max :
............max = L[i]
....while (L[i] < max) and (i<N-1) :
........i += 1
....if i == N-1 :
........on_a_pris_le_dernier += 1
....compt = 0
....for elt in L :
........if elt > L[i]
............compt += 1
....total += compt
print('On a pris le dernier dans '+str(on_a_pris_le_dernier)+' cas.')
print(compt/1000)
```

Il s'avère que dans cinq pour cent des cas on finit par garder le dernier candidat, et qu'en moyenne six pour cent des candidats étaient meilleurs que le candidat retenu (cas $N = 200$).
Pour N égal à 1000, les deux taux d'erreur sont de l'ordre de deux pour cent.
Si on avait pris le premier candidat venu et directement éliminé les autres, c'est cinquante pour cent des candidats qui étaient meilleurs que celui retenu, ce qui semble naturel.

2014-15
Une procédure récursive sur les entiers. 1-phô

Tant que l'entier est pair *(test `n%2 == 0`)*, on recommence avec l'entier `n//2`.
On finit par arriver sur un entier impair *(`elif...`)*. Si il vaut 1 ; on sort avec la réponse True, sinon, on répond `False`.
En résumé, on divise l'entier tant qu'on peut par 2 *(exposant de 2 dans la décomposition de n en*

produit de facteurs premiers), si il reste autre chose que 1, on répond `False`. On ne répondra donc `True` que si n est une puissance de 2.

7	12	16	28	2014	2048	2057
False	False	True	False	False	True	False

Un exemple pour saisir : avec $n = 28$
28 est pair, on lance `armacidegard(14)`
14 est pair, on lance `armacidegard(7)`
7 est impair, on teste si il vaut 1
7 ne vaut pas 1, la réponse est `False`

M.P.S.I.2 2014 **Jeudi 4 novembre** 2015 Charlemagne — 1-phô Q

MPSI 2/2014 **Suite récurrente.** 1-phô Q

On se propose d'étudier suivant sa valeur initiale la suite définie par $u_{n+1} = F(u_n)$ où F est la fonction qui calcule la **distance entre un entier et son renversé**.

Pour saisir

a	543	34	14341	15432	8019	1089	8712
renversé de a	345	43	14314	23451	9108	9801	2178
$F(a)$	198	9	0	8019	1089	8712	6534

Pour bien saisir, prendre un entier à six chiffres et calculer les images successives *(utiliser Python en interactif pour calculer les différences $|a - renverse(a)|$)*.
Tiens, et pourquoi pas avec 1 013 ? Ou même 150 703 ?

Première programmation :

• Ecrire une procédure qui prend un entier a et détermine son renversé :

```
def renverse(a) :
....renv = 0
....while ....
........là c'est à vous de faire en utilisant //10 et %10
....return(renv)
```

• Ecrire une procédure `F` qui prend un entier a et calcule $|a - renverse(a)|$.

• Utiliser cette procédure pour calculer les images successives de 8 019.

• Calculer les images successives d'un entier à huit chiffres.

Etape mathématique :

• étudier la suite u pour les valeurs de u_0 entre 0 et 10.
• étudier la suite u pour les valeurs de u_0 entre 10 et 100.
• montrer que pour u_0 donné à N chiffres, les u_n ont tous au plus N chiffres et sont donc en nombre fini
• déduire que la suite u est périodique à partir d'un certain rang.
• étudier la question : "la suite u peut elle être constante à partir d'un certain rang *(autre que constante nulle)* ?".

Procédure :

• Créer une procédure qui pour a donné donne les termes de la liste jusqu'à ce qu'elle soit périodique

```
def suite(a) :
....L = []
....while not ...
........ a = F(a)
........
....return(L)
```

• Ajouter une information : la longueur du "vol" c'est à dire le nombre de termes de la liste avant qu'elle ne devienne périodique.
• Autre information : indiquer quelle sera la période de la suite une fois qu'elle sera devenue périodique.

Prolongements :

• Chercher des entiers a plus petits que 10^8 pour lesquels la durée du vol est la plus longue possible

• Calculer la longueur moyenne du vol pour les entiers de 0 à 10^6.

• Chercher des entiers a plus petits que 10^8 pour lesquels la longueur de la période est la plus longue possible

---

Exposition :

Transformer l'exercice en un exposé de mathématiques et informatique à traiter à plusieurs, en quarante minutes.

MPSI 2/2014 **Explorateur** rhum 1-phô

Je vous offre ici un programme, presque complet. Comprenez le avant que de le recopier.

```
# importation des modules et des fonctions utiles
from random import randrange
from Tkinter import *
from math import sin
from time import sleep
# et même d'une fonction inutile

# initialisation des constantes
Nlig, Ncol = 32, 22
taille, rayon = 22, 2
NbObstacles, affiche_obstacles = 100, True#False
prof = 0
di, dj = [-1,0,1,0],[0,-1,0,1,1]

# création d'un tableau
Tab = [[1]+[0]*(Ncol-2)+[1] for i in range(Nlig-2)]
Tab=[[1]*Ncol]+Tab+[[1]*Ncol]

# quelques procédures utiles
def dessin() :
....global Laby, fenetre
....fenetre=Tk()
....Laby=Canvas(fenetre,bg='cyan',height=taille*Ncol,width=taille*Nlig)
....Laby.pack(side=TOP,padx=10,pady=10)
# du pur langage du module Tkinter

# une procédure graphique
def avance(a,b) :
....Laby.create_line(a[0]*taille,a[1]*taille,b[0]*taille,b[1]*taille,fill='Black',width=4)
....Laby.create_oval(a[0]*taille-rayon,a[1]*taille-rayon,a[0]*taille+rayon,a[1]*taille+rayon, fill='Blue')
....Laby.create_oval(b[0]*taille-rayon,b[1]*taille-rayon,b[0]*taille+rayon,b[1]*taille+rayon, fill='Red')
....Tab[b[0]][b[1]]=1
....Laby.update()
....sleep(0.1)

# une procédure qui fait ce qu'indique son nom (case est un couple de deux entiers)
def visite_voisins(case) :
....NbVoisins, ListVoisins = 0, []
....for k in range(4) :
........ki,kj = (case[0]+di[k])%Nlig, (case[1]+dj[k])%Ncol
........if Tab[ki][kj]==0 :
............NbVoisins+=1
............ListVoisins.append([ki,kj])
....return(NbVoisins, ListVoisins)

# une procédure utilisant le hasard
def tire_case() :
```

```
....it, jt = randrange(Nlig-2)+1, randrange(Ncol-2)+1
....Tab[it][jt]=1
....if affiche_obstacles :
........Laby.create_oval(it*taille-rayon,jt*taille-rayon,it*taille+rayon,jt*taille+rayon, fill='Green')

# la procédure qui utilise une liste en tant que pile
def creuse() :
....global prof
....NouvelleCase = [1,1]
....pile, Tab[1][1] = [NouvelleCase], 1
....while (len(pile) != 0) :
........case=pile[-1]
........nv, lc = visite_voisins(case)
........if nv != 0 :
............NouvelleCase = lc[randrange(nv)]
............avance(case, NouvelleCase)
............pile.append(NouvelleCase)
........else :
............pile.pop()
............prof+=1
............# mine de rien, c'est une instruction capitale qui remonte

# le programme proprement dit, très court comme tout programme bien écrit
dessin()
for k in range(NbObstacles) :
....tire_case()
creuse()
fenetre.mainloop() # pour que la fenêtre de Tkinter reste affichée
```

Pourquoi est il judicieux d'avoir défini des constantes au début du programme ?

MPSI 2/2014 **Aides sur les fonctions.** `rhum` 1 - phô

Le module `time` est surtout utilisé pour sa fonction `sleep` qui prend en paramètre un temps en secondes (nombre à virgule autorisé) pendant lequel le programme doit s'arrêter, afin que l'utilisateur puisse y voir clair [4].

Le module `Tkinter` (appelé ***`Tk`*** ou ***`Tkinter`*** suivant les versions de Python) est là pour que les entrées et sorties soient conviviales. Il permet de créer des lignes de saisie, des boutons à cliquer, des menus déroulants, la souris... Son utilisation n'est pas au programme des Classes Préparatoires [5] mais on va l'utiliser quand même, pour son unité `Canvas`, qui permet de définir un canevas ou tableau sur lequel on va pouvoir tracer des lignes, des rectangles (et carrés), des ovales (et cercles), coller des images bitmap...
La syntaxe d'ouverture est ici

```
fenetre=Tk()
# crée une fenêtre dans laquelle on va insérer le canevas et des boutons si nécessaire
# on peut créer plusieurs fenêtres différentes sur l'écran, dont certaines pourront être à l'intérieur d'une autre
Laby=Canvas(fenetre,bg='cyan',height=taille*Ncol,width=taille*Nlig)
# crée le canevas dans la fenêtre fenetre, et précise sa couleur de fond bg, sa largeur width et sa hauteur height [6]
# à vous de comprendre le rôle de la variable taille initialisée au début du programme
Laby.pack(side=TOP,padx=10,pady=10)
```

[4] prévenez quand même les élèves de lycée qui passeraient dans le couloir que je ne suis pas un pervers quand je vous dis "*tu as mis un **`sleep`** trop long, tu aurais du utiliser un **`string`***"

[5] ce qui peut se comprendre tant il contient de fonctions à la syntaxe spécifique

[6] ou le contraire, je suis nul en anglais

 # si on oublie cette instruction, le canevas existe bien, mais il reste dans l'esprit de `fenetre`*, et il n'est pas affiché, il faut donc indiquer où le mettre dans fenetre, même si il est tout seul...*

Ensuite, entraînez vous à tracer des éléments dans un canevas *(appelé ici Laby)* par
`Laby.create_line(xd, yd, xf, yf, fill = '...')`
`Laby.create_rectangle(x, y, xx, yy, fill = '...')`
`Laby.create_oval(x, y, xx, yy, fill = '...')`
`fill` désigne bien sûr la couleur de remplissage, avec des guillemets sur un nom en anglais. Les quatre coordonnées à entrer sont celles des extrémités de la droite, ou des deux coins définissant le rectangle *(NordOuest et SudEst)*, ou du rectangle contenant notre ovale. A vous de faire quelques essais.
Où est le point (0, 0) ?
Expliquer le rôle d'une éventuelle instruction `width=2` ou même plus ou même `0` après le `fill='...'`. Regardez sur des exemples.

La méthode `canevas.update()` fait comme son nom l'indique une mise à jour du canevas. En effet, vous avez tracé des rectangles, importé des images, mais seulement en mémoire et il faut activer à l'écran les changements.

Que sait on faire avec une liste :
• la créer en donnant la liste de ses termes, et éventuellement par `L = [ ]` si il s'agit de créer une liste vide
• mesurer sa longueur `len(L)`
• empiler en lui ajouter un élément au bout : `liste.append(element)`
• lire un élément de la liste `liste[indice]`, sachant qu'on peut remonter en arrière et que `liste[-1]` donne donc le dernier élément de la liste
• dépiler la liste en extrayant le dernier élément `liste.pop()`
Pour saisir, essayez ce qui suit [7] :

```
L = [1, 2, 5, 6, 4, 3]
L.append(9)
print(L)
L.pop()
print(L)
a = L.pop()
print(L, a)
```

Il existe bien sûr d'autres méthodes qu'on peut appliquer aux listes *(couper, extraire des éléments, vérifier l'appartenance d'un élément, trouver l'indice de sa position)*.

On a crée une liste appelée `pile` qui est en fait même ce qu'on appelle une pile en programmation, sur laquelle on pose des éléments et les enlève, comme une pile d'assiettes.
Il s'agit ici d'une pile de type LIFO *(last in = first out)* comme pour un ascenceur.
Il existe aussi des piles FIFO *(first in = first out)* comme dans la file d'attente au guichet d'une administration. je me m'apesantirai pas ici là dessus *(programme de seconde année)*, mais je compte sur les élèves faisant l'exposé pour approfondir.
Elle sert ici à savoir comment remonter quand on aboutit par malchance sur un cul de sac.
Elle sert aussi à savoir quand il ne reste plus de chemins à explorer.

Pour la création d'une liste faite de termes tous égaux à a, on peut faire une boucle :

```
L = []
for k in range L :
....L.append(a)
```

Mais il y a plus simple : `L = [a]*n` qui crée n termes égaux à a dans une liste. On peut aussi opter pour `L = [a for k in range n]`.
Pour créer un tableau à double entrée *(une matrice)*, on peut faire une double boucle *(comme ci dessus)*. Mais on peut aussi essayer de recourir deux fois à l'astuce précédente : `M = [[0]*n]*p`.

[7] et pour l'exposé, présentez ces résultats au tableau

C'est tentant, mais il y a un problème. Tapez l'instruction précédente. Visualisez `M` *(*`print(M)`*)*. Ajoutez ensuite l'instruction `M[1][3]=1` qui doit modifier un terme de la matrice. Affichez la matrice. Que s'est il passé ?
Il s'avère que les p copies sont égales, au sens fort du terme. Il faut en fait que l'on crée des listes les unes après les autres et qu'on les colle ensuite : `M = [[0]*n for k in range(p)]`.

Que fait alors l'instruction
```
Tab = [[1]+[0]*(Ncol-2)+[1] for i in range(Nlig-2)]
Tab=[[1]*Ncol]+Tab+[[1]*Ncol]
```
A quoi servent les 1 sur le bord ?

MPSI 2/2014 **Pour aller plus loin.** `rhum`

• Pouvez vous tracer d'abord dans le canevas les murs extérieurs.

• Pouvez vous tirer au hasard la position de départ, sachant qu'il ne faut pas partir d'un obstacle *(répéter le tirage aléatoire tant que l'on tombe sur une case de valeur* 1*)*.

• Pouvez vous autoriser l'explorateur à se déplacer aussi en diagonale ?

• Pouvez vous visualiser les moments où l'explorateur remonte vers une case où il reste des voisins *(c'est dans le "dépilage")*.

• Pouvez vous créer un labyrinthe en dimension 3 *(et pourquoi pas plus ?)*, éventuellement en demandant que les déplacements dans la troisième direction soient moins nombreux que les déplacements dans les deux premières *(changement d'étage)*.

• Pouvez vous mettre une image de fond à ce labyrinthe.

• Pouvez vous ajouter une représentation graphique à l'aide du module `turtle`.

MPSI 2/2014 **Mélange parfait** 1-phô lower

Le point de départ : mon fils a mélangé un paquet de cartes numérotées, et il a constaté avec stupeur qu'il y avait une carte qui était restée finalement à sa place.
Y a-t-il de quoi être surpris ?

La question peut se reformuler :
Si on prend "au hasard" une permutation σ de la liste $[0, 1, \dots n]$, quelle est la probabilité qu'il y ait dans cette liste un point fixe ($\sigma(k) = k$) ?
Mieux encore : pour chaque permutation σ de la liste $[0, 1, \dots n]$, on compte le nombre de points fixes que l'on note $\phi(\sigma)$, et on calcule la proportion de permutations σ vérifiant $\phi(\sigma) \neq 0$.

Reformulation encore sous forme d'un classique des petits problèmes mathématiques :
n personnes arrivent au restaurant, et chacune dépose son chapeau au vestiaire ; quand ils sortent, une panne d'électricité fait que l'employé(e) du vestiaire leur rend à chacun un chapeau pris au hasard ; quelle est la probabilité qu'aucun n'ait récupéré son propre chapeau.
Le problème est parfois aussi raconté avec n lettres envoyées à n personnes au hasard ; on s'intéresse alors au nombre de *misadressed letters*.
Racontez la avec des couples dans une boite échangiste où la lumière vient de tomber en panne.
On parle aussi de permutations partout mélangeantes.

MPSI 2/2014 **PROGRAMMATION** 1-phô lower

On va se fixer N assez grand et générer listes/permutations de la liste $[1,\ 2, \dots N-1]$. On va ensuite pour chaque liste compter le nombre d'éléments bien placés.
On dressera ensuite un histogramme qui indique pour chaque valeur de k combien de permutations vérifient $\phi(L) = k$.
Au fait, combien vérifient $\phi(L) \geqslant N$?
Et combien vérifient $\phi(L) = N - 1$?

On a deux possibilités :
• engendrer toutes les permutations de la liste $[1,\ 2, \dots N-1]$ par un algorithme récursif *(dont je ne recommande la recherche qu'aux plus férus en informatique)*
• engendrer un nombre `Nb` assez grand de listes au hasard *(c'est ce que je recommande ici)*

Pour engendrer une permutation aléatoire de $[1,\ 2, \dots N-1]$, je vous propose le script suivant, à vous de l'expliquer :

```
def liste_alea(n) : #engendre une liste aleatoire de longueur n
....L = [k for k in range(n)]
....LL = []
....while len(L) > 0 :
........LL.append(L.pop(randrange(len(L))))
....return(LL)
```

La fonction `randrange` vous est connue et doit être importée.
La méthode `append` ajoute un élément au bout d'une liste
La méthode `pop` extrait un élément d'une liste et l'enlève de la liste en question.
Par défaut `a = L.pop()` met dans `a` le dernier élément de la liste `L` et racourcit en conséquence la liste `L` *(opération de "dépilage")*.
Plus précisément `a = L.pop(k)` met dans `a` l'élément d'indice `k` et l'enlève.
La seule méthode `L.pop(k)` efface l'élément d'indice `k` de la liste `L` et ne le met nulle part.
Expliquez pourquoi on ne peut pas utiliser la fonction `choice` qui choisit un élément au hasard

dans une liste. C'est ainsi que `choice(LL)` remplace `LL[randrange(len(LL))]`.

Il faut ensuite une fonction `phi` qui parcourt une liste donnée `L` et compte combien d'éléments `L[k]` sont en position `k`. On va donc créer un compteur qu'on incrémentera à chaque fois qu'on verra `L[k]==k`.

Il ne reste plus qu'à mettre en boucle une nombre `Nb` de fois la génération de listes aléatoires `L`, le décompte de termes bien placés $\phi(L)$. On compte ensuite combien de listes `L` vérifient $\phi(L) = 0$, combien vérifient $\phi(L) = 1$ et ainsi de suite.
A vous de voir ensuite si vous affichez un histogramme pour ce décompte, ou si vous calculez juste la proportion de permutations partout-mélangeantes $\frac{nombre\ de\ listes\ L\ verifiant\ \phi(L)=0}{Nb}$.

La proportion de permutations vérifiant $\phi(L) = 0$ doit, pour `N` et `Nb` assez grands se rapprocher de $36,8\%$.

1-phô

MPSI 2/2014 **MATHEMATIQUES** `lower`

Pour tout entier naturel n, on note p_n le nombre de permutations σ de $[0,\ 1, \dots n-1]$ vérifiant $\forall k,\ \sigma(k) \neq k$.
Montrez : $p_1 = 0$, $p_2 = 1$ et $p_3 = 2$.
Quelle valeur donneriez vous à p_0 ?
Montrez par un argument de dénombrement [8] : $p_{n+1} = n.(p_{n-1} + p_n)$
(indication : on ajoute l'élément n à la liste $[0,\ 1, \dots n-1]$, on sait que son image $\sigma(n)$ est un élément k de $[0,\ 1, \dots n-1]$ puisque $\sigma(n) \neq n$, on regarde alors suivant que $\sigma(k)$ est égal à n ou pas).
Complétez le tableau :

n	0	1	2	3	4	5	6	7
p_n		0	1	2				

Quel sens donner à p_0 ? Retrouvez p_4 par un argument de dénombrement.
Donnez la forme des éléments p_5 permutations pour n égal à 5.

Vérifiez qu'on a quand même toujours $p_n < n!$.

On note alors $a_n = \frac{p_n}{n!}$. Exprimez a_{n+1} à l'aide de a_n et a_{n-1}.
Montrez par récurrence sur n : $a_n = \sum_{k=0}^{n} \frac{(-1)^k}{k!}$.
Montrez alors par récurrence et intégration par parties : $e^{-1} = a_n + \frac{(-1)^{n+1}}{n!}.\int_0^1 (1-t)^n.e^{-t}.dt$.
Montrez : $\frac{1}{n!}.\int_0^1 (1-t)^n.e^{-t}.dt \leqslant \frac{1}{n!}$.
Déduisez alors que la proportion de permutations partout mélangeantes a_n converge vers une valeur qui vaut bien $36,8\%$ à 10^{-3} près *(et plus précisément encore ?)*.

Avez vous une conjecture pour le pourcentage de listes vérifiant $\phi(L) = 1$? *(moi je n'en ai pas, mais...)*

Supplément gratuit : maintenant qu'on a une formule explicite pour a_n, il serait bon de créer une procédure qui calcule a_n ou p_n pour n donné.
Plusieurs procédures pourront être envisagées : calcul approché en somme de floats, calcul exact par procédure récursive.

[8] et surtout pas par une récurrence, mort aux réflexes idiots de Terminable ! vive l'intelligence, à bas l'endoctrinement

MPSI 2/2014

Nombres seconds

1-phô

blond

On appelle nombre second tout nombre qui s'écrit comme produit de deux nombres premiers *(distincts ou non)*. L'appellation n'est pas homologuée, mais je la trouve bien.
Voici les premiers seconds *(si je puis dire)* :

4	6	9	10	14	15	21	22	25	26

Notre but est d'engendrer la liste de ces nombres, de savoir reconnaître si un entier donné est un second, compter combien il y en a entre 2 et n.

On va avoir besoin de travailler sur les nombres premiers.O n va donc mettre en place deux algorithmes, sans chercher à les optimiser.
L'algorithme basique qui teste si un nombre donné est premier est classique :
on cherche déjà si il est pair, et sinon, on cherche si il est divisible par un nombre impair plus petit que sa racine carrée. Pourquoi chercher un diviseur plus petit que $\sqrt{N}$? Simplement parce que pour tout diviseur d de N qui serait plus grand que $\sqrt{N}$, on aurait eu le diviseur N/d, plus petit que $\sqrt{n}$.

```
def TestPrem(N) :
....if n%2 == 0 and n > 3 :
........return(False)
....d = 3
....while (N%d != 0) and d*d <= N :
........d += 2
....return(d*d > N)
```

Il faut s'habituer à une syntaxe un peu plus évoluée que les bidouillages de calculette.
L'instruction `while (p%d != 0) and d*d <= p : d=d+2` nous permet d'avancer de 2 en 2 *(donc de ne regarder que des nombres impairs)*. On sort de la boucle quand un certain d divise N *(test `(N%d != 0)`)* ou quand on a atteint $\sqrt{N}$ *(test `(d*d<=N)`)*. Dans le premier cas, on a un nombre composé *(non premier)* dans le second, on a un nombre premier. Il suffit donc de savoir par laquelle de ces deux portes on est sorti. C'est le `return(d*d>N)`. Ce que la procédure va retourner est donc un booléen qui vaut `True` ou `False`.

Ensuite, il y a une procédure fort efficace pour engendrer la liste des `Nb` premiers nombres premiers. On met `2` dans la liste, puis on ne regarde que des nombres impairs. Chaque fois qu'on regarde un nouveau nombre `p`, afin de savoir si il est premier, il suffit de regarder si il est divisible par les nombres premiers qui l'ont précédé, et non par tous les nombres impairs.

```
#script pour engendrer une liste de nombres premiers
Lprem = [2]
p = 1
while len(Lprem) < Nb :
....p += 2
....k = 0
....while not((p%Lprem[k])==0) and k < len(Lprem)-1 :
........k+=1
....if k == len(Lprem)-1 :
........Lprem.append(p)
print(Lprem)
```

Expliquez ce programme, et vérifiez le.
Modifiez le pour qu'il engendre la liste des nombres premiers plus petits qu'un certain `Pmax` donné à l'avance.

Une fois qu'on a fait tourner ce petit script, il est plus rapide de tester si un nombre `N` *(plus petit que* ***Pmax****)* est premier : on regarde si il est dans cette liste *(****N in Lprem****)*.
Optimisez le test : si `N` est plus petit que `Pmax`, on regarde si il est dans la liste, sinon, on lance `TestPrem`.

Passons à présent aux nombres seconds.
On veut engendrer une liste de nombres seconds, c'est à dire de nombres de la forme `Lprem[i]*Lprem[j]`.
Expliquez pourquoi le script suivant n'est pas optimal :

```
Lsec = []
for i in range(...) :
....for j in range(...) :
........Lsec.append(Lprem[i]*Lprem[j])
print(Lsec)
```

Pouvez vous compacter votre instruction en une seule ligne ?

Ce script va créer une liste assez désordonnée, et contiendra certains éléments sans en contenir d'autres.

	2	3	5	7	11	13
2	4	6	10	14	22	26
3	6	9	15	21	33	39
5	10	15	25	35	55	65
7	14	21	35	49	77	91
11	22	33	55	77	121	143
13	26	39	65	91	143	169

On a ainsi crée le nombre 169 mais on n'a pas encore croisé 34.
Arrangez votre script pour n'engendrer que les nombres seconds plus petits que `Smax`.

Il faut ensuite trier.
Je vous rappelle l'algorithme qui pousse en fin de liste `L` le plus grand de ses élements :

```
for k in range(len(L)-1) :
....if L[k] > L[k+1] :
........L[k], L[k+1] = L[k+1], L[k]
```

Mettez le en boucle pour faire remonter les éléments dans l'ordre.
La liste doit alors être triée *(algorithme du tri-bulle)*.
Donnez la liste des nombres seconds plus petits que 1000.

On vous demande ensuite de créer une procédure qui teste si un nombre `N` est second, sans utiliser la liste `Lsec`. Je vous propose de chercher un diviseur premier `p` de `N` *(utiliser* ***Lprem****)* et de voir si `N/p` est alors dans la liste `Lprem`. Ecrivez ce script.

Exploitez alors ce script pour engendrer les `Nbs` premiers nombres seconds.

Ecrivez un script qui calcule combien il y a de nombres seconds plus petits que `Nmax`.

MPSI 2/2014

RACINES

1-phô

L'objectif est la résolution d'une équation $f(x) = 0$ d'inconnue x, avec f continue sur un segment $[a,\ b]$, avec évidemment $f(a)$ et $f(b)$ de signes opposés. On va mettre en place divers algorithmes.
On suppose que f est donnée par une procédure telle que

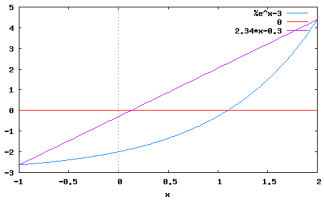
```
def f(x) :
....return(...)
```

pouvant faire appel à une boucle pour une somme, des tests...
Corrigez le premier algorithme que voici : •1 pt.•

```
precision = float(input(...))
aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)
while abs(bb-aa) > precision :
....c = (a+b)/2
....fc = f(c)
....if fa*fc > 0 :
........bb, fb = c, fc
....else :
........aa, fa = c, fc
print(aa, '< racine <', bb)
```

Ajoutez un morceau de script qui demande la saisie de `a` et `b` et vérifie $f(a) * f(b) < 0$ *(en boucle jusqu'à avoir effectivement* $f(a).f(b) < 0$*)*. •1 pt.•
Expliquez pourquoi on a mis `aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)`. •1 pt.•
Par quoi pourrait on remplacer `if fa*fc < 0 :` pour que le programme "tourne" plus vite ? •1 pt.•
Pourquoi a-t-on proposé `aa, fa = c, fc` plutôt que `aa, fa = c, f(c)`. •1 pt.•

On se propose d'utiliser la méthode dite "des sécantes". On pose $A(aa,\ f(aa))$ et $B(bb,\ f(bb))$ et on note s l'abscisse du point d'intersection de la droite $(A\ B)$ avec l'axe des abscisses.

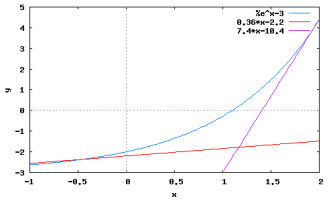


Mettez vous dans la peau de chacun des élèves suivants pour calculer s et détaillez leurs calculs : •3 pt.•

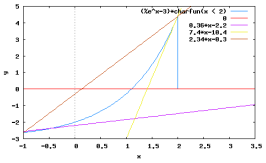
élève	méthode	réponse
Accinée	je calcule l'équation de la sécante, puis...	
Endual-Ennemi	j'écris une condition d'alignement par un déterminant	
Espa-Sasfabric-Encorh	j'utilise le théorème de Thalès	

Vérifiez que s est entre aa et bb. •1 pt.•
Réécrivez alors l'algorithme. •2 pt.•
Expliquez pourquoi cette méthode est plus judicieuse. •1 pt.•

On suppose qu'on a aussi la procédure qui calcule $f'(x)$ pour x donné. On utilise alors la méthode de Newton : on détermine `Ta` *(et `Tb`)*, intersection(s) de l'axe des abscisses avec la tangente en `A` *(et avec la tangente en `B`)*. Donnez la formule qui calcule `Ta` et `Tb`. •3 pt.•



On peut démontrer mathématiquement qu'avec `c` ou `s`, la convergence est linéaire *(la précision double quand on double le nombre d'étapes)*, alors qu'avec la méthode de Newton, une fois qu'on est "assez proche de la racine", la convergence est quadratique.
Adaptez le script pour qu'à chaque itération on retienne l'intervalle dont les extrémités sont à choisir parmi `a`, `b`, `c`, `s`, `Ta` et `Tb`, le plus court possible sur lequel il y a un changement de signe de f. •5 pt.•



MPSI 2/2014 — ANNIVERSAIRES — 1-phô

Objectif : on donne une liste `L` composée d'entiers *(des références produits ou clients dans une commande, la liste des 9 nombres d'une ligne ou colonne de Su-Do-Ku, des dates d'anniversaire...)*, il faut savoir si il y a un élément en double dans la liste.
On va donc écrire une procédure introduite sous la forme `def test(L) :` qui donnera par `return` la valeur `True` si tous les éléments de la liste sont différentes, et `False` si il y a un élément en double *(en option, le programme pourra indiquer la valeur trouvée en double)*.
On rappelle les fonctions et méthodes sur une liste `L` :

fonctions et affectations :

`len(L)`	longueur	entier
`L[k] = a`	remplace l'élément d'indice `k`	
`L[k]`	lit l'élément d'indice `k`	
`a in L`	test d'appartenance	`True` ou `False`

méthodes :

`L.append(x)`	ajoute `x` en fin de liste `L`	allonge la liste
`L.insert()`	insère x en position k dans L	allonge la liste
`L.pop(k)`	sort l'élément d'indice `k` [9] de `L`	raccourcit la liste et donne un élément
`L.sort()`	trie la liste	modifie la liste en un temps n.log(n)

Ecrivez un algorithme en Python. Expliquez en quelques mots l'algorithme que vous avez conçu. •3 pt.•
Indiquez le nombre maximum d'étapes en fonction des données *(ici la longueur de la liste)*. •1 pt.•
Attention, si vous risquez de modifier la liste par votre procédure, pensez à travailler sur une copie.

On va exploiter cette fonction pour un problème classique : quelle est la probabilité que deux élèves de la classe aient la même date d'anniversaire ?
On connaît la résolution rigoureuse de cette question *(dans un modèle simplifié où on ne tient pas compte des années bissextiles et où on estime que les dates de naissance sont équiprobables* [10]*)*. Mais on va ici utiliser l'ordinateur et son module `random`.
Je vous demande de tirer des listes de 48 nombres aléatoires compris entre 0 et 364 et de compter le pourcentage de listes dans lesquelles toutes les dates sont différentes. •4 pt.•
Variante : on suppose qu'il y a un mois de l'année où la probabilité de naissances double. Revoyez l'algorithme. •2 pt.•

[10] *dans les villages de mon enfance, on notait un pic de naissances trois mois avant la fête du village, ou plutôt neuf mois après*

2014-15

Algorithme par dichotomie. 1-phô

C'est l'algorithme classique qui cherche à chaque fois le milieu de l'intervalle et prend comme nouvel intervalle celui sur lequel il y a un changement de signe.

`precision = float(input(...))`

On saisit la précision ε avec laquelle on veut encadrer la racine. `input` ne suffirait pas, il faut s'assurer qu'on a bien un flottant.

`aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)`

On initialise avec le premier segment $[a, b]$ et on regarde le signe à chaque extrémité *(en espérant que ce soit deux signes opposés).*

`while abs(bb-aa) > precision :`

On met en boucle tant que la longueur de l'intervalle $[aa, bb]$ est plus grande que ε *(il faut en effet s'assurer qu'on n'entre pas dans boucle folle).*

`....c = (aa+bb)/2`

et non pas `c = (a+b)/2`

`....fc = f(c)`

On cherche le milieu du segment et on calcule la valeur de f au milieu du segment.

```
....if fa*fc > 0 :
........bb, fb = c, fc
```

C'est là qu'est l'erreur. Si il n'y a pas eu un changement de signe entre `a` et `c`, alors il est entre `c` et `b`. Le nouvel intervalle doit être `[c, b]`. Comme `bb` ne change pas, c'est `aa` qu'on doit remplacer par `c`, et dans le même temps `fa` par `fc` *(déjà calculé).*

```
....else :
........aa, fa = c, fc
```

Dans l'autre cas, c'est l'autre moitié d'intervalle qu'on garde.

`print(aa, '< racine <', bb)`

Une fois qu'on a obtenu `abs(bb-aa)<precision`, on sort de la boucle et on affiche que la racine est entre `aa` et `bb`.

Ce qu'on aurait dû proposer :

```
....if fa*fc < 0 : #c'est le bon test
........bb, fb = c, fc
....else :
........aa, fa = c, fc
```

Ce qu'on pouvait mettre au début pour saisir `a` et `b` :

```
boucle = True
while boucle :
....a = float(input('Saisissez la premiere abscisse a :'))
....b = float(input('Saisissez la seconde abscisse b :'))
....fa, fb = f(a), f(b)
....if fa*fb > 0 :
........print('Mauvais intervalle, recommencez...')
....else :
........boucle = False
```

La boucle est effectuée tant que le booléen `boucle` est à la valeur `True` *(qu'on lui a donnée initialement).* Quand on a enfin $f(a).f(b) < 0$, on met `boucle` à `False` et on sort de la boucle.

`aa, fa, bb, fb = float(a), f(aa), float(b), f(bb)` c'est pour s'assurer que aa et bb sont bien des float et non pas des int (sinon, avec des int, la division par 2 risque de nous faire tomber brutalement sur 0, à tort).

D'autre part, on travaille donc sur une copie de a et b, et on garde les valeurs saisies de côté.
On stocke aussi f(a) et f(b) dans deux variables fa et fb pour ne pas les recalculer inutilement par la suite.

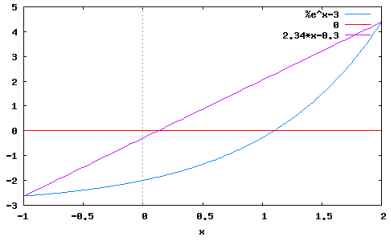
Le test `if fa*fc < 0 :` est inutilement lourd. Il calcule le produit de deux réels, alors que seul nous importe le signe. On perd donc un temps fou à calculer des décimales inutiles.
On va plutôt extraire le signe de chacune et calculer le produit des signes :
`if (fa<0 and fc>0) or (fa>0 and fc<0) :`

`aa, fa = c, fc` est préférable `aa, fa = c, f(c)` car dans ce dernier, on recalcule $f(c)$ qui a déjà été calculé auparavant. L'ordinateur est bête et obéissant. Il recalcule. Il fait donc deux fois un même calcul. Or, si le calcul de $f(x)$ est "long", c'est idiot. Or, si f n'est pas définie par une simple formule mais par une somme infinie que l'on tronque à un certain rang, ou comme une intégrale *(si si, ça existe)*, c'est un peu idiot.

2014-15

Méthode des sécantes. 1-phô

On dispose de deux points $A(aa,\ fa)$ et $B(bb,\ f(bb))$ ou même $A(aa,\ fa)$ et $B(bb,\ fb)$ avec nos notations.



Le premier élève a été pollué par le programme idiot du secondaire qui transforme tout en équations *(la dictature du signe "égale")*. Il cherche l'équation de la droite sous la forme la plus simple qui soit *(j'espère que c'est celle des programmes officiels)* :
$y = coeff.(x - a) + fa$ qui tient compte du coefficient directeur et se débrouille pour que l'on ait $y = fa$ pour $x = a$: $\boxed{y = \dfrac{fb - fa}{bb - aa}.(x - a) + fa}$

J'avoue que je renierai totalement l'élève (mais en est ce encore un ?) *qui passera par : l'équation est de la forme* $y = \alpha.X + \beta$ *et résoudra* $\alpha.aa + \beta = fa$ *et* $\alpha.bb + \beta = fb$. *Je crois que pour celui là, il n'y a plus rien à faire, à part une école de chimie* (avec le seul espoir qu'il n'ait ensuite aucun emploi, car je tiens à préserver les générations futures d'un tel danger).

Bref, ayant l'équation $y = \dfrac{fb - fa}{bb - aa}.(x - aa) + fa$, on résout $\dfrac{fb - fa}{bb - aa}.(\underline{x} - aa) + fa = 0$ et on trouve $s = aa - fa.\dfrac{bb - aa}{fb - fa}$ sachant que $fb - fa$ est non nul *(fa et fb sont de signes opposés).*
Un géomètre reconnaît qu'on part de l'abscisse aa et qu'on avance avec un certain coefficient directeur pour compenser l'ordonnée fa.

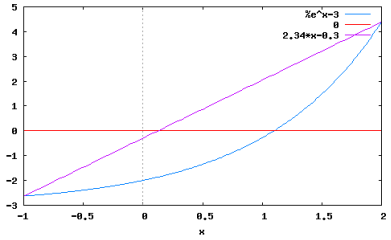
Le second élève a une approche d'algébriste. Il prend trois points $A(aa,\ fa)$, $B(bb,\ fb)$ et $S(s,\ 0)$. Il détermine deux vecteurs : $\overrightarrow{SA} = \begin{pmatrix} aa - s \\ fa \end{pmatrix}$ et $\overrightarrow{AB} = \begin{pmatrix} bb - aa \\ fb - fa \end{pmatrix}$ et il leur demande d'être colinéaires : $\begin{vmatrix} aa - s & bb - aa \\ fa & fb - fa \end{vmatrix} = 0$. On retrouve très vite le même calcul :

$$\boxed{s = aa - fa.\frac{bb - aa}{fb - fa} = \frac{fb.aa - fa.bb}{fb - fa} = \frac{f(b).a - f(a).b}{f(b) - f(a)}}$$

Sous cette forme plus compacte, on reconnaît d'ailleurs une moyenne de aa et b pondérée de coefficients fb et $-fa$ (tous deux positifs).

Le dernier élève a une approche de géomètre, avec des mesures d'angles ou ce qu'on appelle des

triangles semblables pour faire peur.
Il se place en S et mesure la tangente des deux côtes : $\frac{oppose}{adjacent} = \frac{fb}{bb-s} = \frac{fa}{aa-s}$. Il retrouve $aa.fb - bb.fa = s.(fb - fa)$.



Enfin, l'élève qui "voit les mathématiques" dit que s est bien une moyenne de aa et bb avec les coefficients correspondant eux ordonnées. Comme les plateaux d'une balance.

Disposant de cette formule, on modifie l'algorithme :

```
while abs(bb-aa) > precision :
....s = (aa*fb-bb*fa)/(fb-fa)
....fc = f(c)
....if fa*fc < 0 :
........bb, fb = c, fc
....else :
........aa, fa = c, fc
print(aa, '< racine <', bb)
```

On évitera bien sûr la formule `s = (aa*f(bb)-bb*f(aa))/(f(bb)-f(aa))` qui sollicite bien trop la fonction f pour des calculs répétitifs.

Cette méthode va nous rapporcher de la solution plus vite que la dichotomie. Elle va en effet couper l'intervalle en deux parts inégales et c'est dans le plus court des deux segments que se trouvera la racine *(sauf fonction particulière, changeant de concavité)*.
En effet, si $|f(aa)|$ est plus petit que $|f(bb)|$, l'abscisse s est plus proche de aa que de bb.

2014-15

Méthode de Newton

1-phô

Cette fois, on suppose f dérivable. On peut donc considérer la tangente en B et lui faire intersecter Ox (sauf si elle lui est parallèle).
Cette tangente a pour équation $y = f'(b).(x - b) + f(b)$. C'est direct : coefficient directeur, et exigence que y vaille $f(b)$ pour $x = b$.
On intersecte avec Ox : $f'(b).(x - b) + f(b) = 0$ et on trouve $Tb = b - \frac{f(b)}{f'(b)}$
La formule est homogène : $x - \frac{y}{dy/dx}$. Pour $f(b)$ et $f'(b)$ positifs, elle place Tb avant b.
On peut aussi effectivement raisonner sur les triangles et angles : $coefficient\ directeur = \frac{oppose}{adjacent} = \frac{fb}{b - Tb}$.
On a de même $Ta = a - \frac{f(a)}{f'(a)}$.

On isère de nouvelles lignes dans le programme :

```
fprimea, fprimeb = f'(a), f'(b)
if fprimea != 0 :
....Ta = a - fa/fprimea
else :
....Ta = a
if fprimeb != 0 :
....Tb = b - fb/fprimeb
```

```
else :
....Tb = b
```
On évite de devoir par un hasard énorme à diviser par 0.

L'optimisation pour chercher l'intervalle le plus court avec changement de signe sera plus lourde.
On met a, b, c, s, Ta et Tb dans une liste `Lx` (de longueur 6) et on met les images dans une autre.
```
Lx = [aa, bb, c, s, Ta, Tb]
Ly = [fa, fb, fc, fs, f(Ta), f(Tb)]
```
On parcourt cette liste sans prendre deux fois le même élément :
```
for i in range(1, 6) :
....for j in range(i) :
```
On regarde si il y a un changement de signe, et si oui, on mesure la longueur du segment :
```
........if Ly[i]*Ly[j] < 0
........long = abs(Lx[i]-Lx[j])
```
On regarde si on a trouvé une longueur optimale :
```
........if long < min :
```
Et on mémorise alors ce segment :
```
............bas, haut, min = min(Lx[i], Lx[j]), max(Lx[i], Lx[j]), long
```
A la fin, on a mis dans bas et haut les deux extrémités de l'intervalle (dans le bon sens) et on est sûr d'avoir pris le plus court.

```
Lx = [aa, bb, c, s, Ta, Tb]
Ly = [fa, fb, fc, fs, f(Ta), f(Tb)]
min = abs(bb-aa)
for i in range(1, 6) :
....for j in range(i) :
........if Ly[i]*Ly[j] < 0
........long = abs(Lx[i]-Lx[j])
........if long < min :
............bas, haut, min = min(Lx[i], Lx[j]), max(Lx[i], Lx[j]), long
```

2014-15

Savoir si il y a un élément en double dans une liste. 1-phô

Comme on va sûrement modifier la liste, on va travailler sur une copie appelée `double` :
```
def test(L) :
....double = L[ : ]
```
Une solution va constister à prendre un par un les éléments de la liste et à voir si on les retrouve dans le reste de la liste.
On notera qu'une fois qu'un élément a été testé, il n'est plus utile de le garder dans la liste, puisqu'on le sait unique.
Pour tester si le dernier élément de la liste `L` est en double :
```
dernier = L.pop()
if dernier in L : #puisque L ne contient plus dernier
....return(False) #et même éventuellement return(False, dernier)
```

Pour mettre ce processus en boucle :
```
while len(double) > 0 :
....dernier = double.pop()
....if dernier in L :
........return(False)
return(True) #si on est sorti de la boucle while c'est qu'aucun élément n'était en double
```
Attention, ne pas mettre un
```
....if dernier in L :
........return(False)
....else :
........return(True)
```

qui aurait pour effet de nous faire sortir de la procédure dès le premier test, même si il est favorable.

On pouvait aussi travailler en mode impératif :

```
for i in range(len(L)-1) : #on n'a pas de teste à faire sur le dernier élément de L
....for j in range(i+1, len(L)) : #on ne teste que les éléments qui suivent
........if L[i] == L[j] :
............return(False)
return(True) #si on a fait a double boucle sans retourner False, c'est que la liste est "injective"
```

On peut aussi être lourd :

```
L.sort() #on trie la liste
for i in range(len(L)-1) :
....if L[i] == L[i+1] :
........return(False)
```

Que ce soit par la boucle `for j in range()` que par le test `dernier in double`, on parcourt la liste dans son entier. Mais à chaque étape, la liste a diminué en taille.

L	dernier	test	nombre d'étapes	total
`[0, 2, 5, 7, 4, 8]`	8	`8 in [0, 2, 5, 7, 4]`	5	5
`[0, 2, 5, 7, 4]`	4	`4 in [0, 2, 5, 7]`	4	5+4
`[0, 2, 5, 7]`	7	`7 in [0, 2, 5]`	3	5+4+3
`[0, 2, 5]`	5	`5 in [0, 2]`	2	5+4+3+2
`[0, 2]`	2	`2 in [0]`	1	5+4+3+2+1
				n.(n+1)/2

En notant classiquement `N=len(L)`, le nombre maximum d'étapes est $\boxed{\frac{N.(N+1)}{2}}$ dans le cas de la liste sans doublon, ou dans la cas d'une liste comme `[5, 5, 2, 5, 7, 4, 8]`.

On passe au problème des dates de naissances. On importe la fonction `randrange` du module `random`.
On crée une liste de dates par `[randrange(365) for eleve in range(48)]`
On fait le test sur chacune de ces listes et quand il est favorable, on incrémente un compteur *(qu'on a initialisé à 0)*. On affiche au final le pourcentage :

```
from random import randrange
for k in range(1000) : #on va créer 1000 classes
....classe = [randrange(365) for eleve in range(48)]
....if test(classe) : #et pas if test(classe) == True qui est idiot
........compt += 1 #on augmente le compteur de 1
print(compt/1000.) #et pas compt/1000 qui donne une division entière, donc 0
```

L'application numérique donne **quatre vingt quinze pour cent** [11] des cas où deux élèves ont la même date d'anniversaire.
La résolution passant par "probabilité que les dates soient toutes distinctes" indique qu'à partir de vingt quatre élève, la probabilité dépasse 1/2. Ce que beaucoup de personnes interprètent en "à partir de vingt quatre élèves, il y en a deux qui ont le même annversaire".
La formule est $\boxed{1-\frac{365.364.363\ldots(365-N+1)}{365.365.365\ldots365}}$ à vous de la justifier.

Si on suppose maintenant que la répartition des dates n'est pas uniforme, il faut modifier le ranrange. On va suppose que le mois de décembre est renforcé *(par symétrie des rôles)*. Comme l'énoncé propose simplement d'en doubler la probabilité, on va travailler avec une année de 365+31 jours, et on va identifier les derniers, ceux de décembre bis avec ceux de décembre.

On remplace le tirage `ranrange(365)` par

```
jour = randrange(365+31)
if jour > 365 :
```

[11] allez voir la chanson de Georges Bassens intitulée "quatre quinze fois sur cent", même si elle n'a rien à voir

`....jour = jour-31`
Tous les jours au delà de 365 sont ramenés en jours du mois de décembre qui héritent donc d'une probabilité doublée.
On notera que cela ne change guère les valeurs mentionnées plus haut.

MPSI 2/2014 **JUNIPER GREEN** 1-phô

Je vous donne les règles du jeu de Juniper Green qui se joue normalement avec un jeu de cartes. Vous devrez en faire une version conviviale pour ordinateur, dans un premier temps avec deux joueurs qui s'affrontent. Ensuite, vous ferez la version où un joueur peut jouer contre l'ordinateur.

MPSI 2/2014 **Règle du jeu.** 1-phô

L'origine du nom du jeu m'est inconnue *(cherchez sur Internet)*. Il a été présenté à plusieurs reprises dans les rubriques mathématiques et ludiques de revues telles que Pour La Science, notamment sous la plume de Ian Stewart et Martin Gardner.

Matériel : jeu de cent cartes numérotées simplement de 1 à 100 *(et pas de 0 à 99 comme avec le Python suprème, vous allez comprendre pourquoi)*.

Nombre de joueurs : 2.
Le premier joueur choisit une carte portant un numéro pair, et la retire du jeu.
Le second joueur doit alors retirer à son tour une carte dont la valeur doit être un diviseur ou un multiple du nombre choisi par le premier.
A tour de rôle, chacun retire une carte dont la valeur doit être un diviseur ou un multiple de la carte du joueur précédent.
Le jeu s'arrête quand un joueur ne peut plus retirer de carte obéissant à cette règle. Il a alors perdu.
En général, ceci arrive avant que le jeu tout entier soit épuisé *(d'ailleurs, ceci pourrait il arriver ?)*.

Exemple de partie *(fort mal jouée)* :

joueur	0	1	0	1	0	1
coups possibles	pairs	1,2,13,52	1,2,4,13	1,39,65,78,91	tout sauf...	rien
coup joué	26	52	13	1	17	perdu !

Quel rôle particulier jouent les nombres premiers *(surtout ceux plus grands que 50)* ?
Pourquoi interdit on que le premier coup joué soit un nombre impair ?

MPSI 2/2014 **Cahier des charges.** 1-phô

L'écran d'ordinateur sert à visualiser les cent cartes sous forme de cases umérotées *(utiliser un `Canvas` de `Tkinter`)*.
Les joueurs jouent à tour de rôle *(l'ordinateur indiquera qui de 0 ou 1 doit jouer)*, avec la souris, en cliquant sur une case licite.
Si le joueur ne clique pas sur une case licite *(multiple ou diviseur du coup précédent)*, soit il ne se passe rien, soit un message l'invite à respecter la règle et à recliquer.
Si il clique sur une case licite, cette case "s'efface", la valeur cliquée est indiquée pour le joueur suivant, ou alors la mention "perdu" s'affiche si l'adversaire n'a plus de case licite.
On pourra disposer d'une option pour surligner ou non à tout instant les cases licites en en changeant la couleur.

MPSI 2/2014 **Méthodes Tkinter à connaître.** 1-phô

Création d'une fenêtre :

```
fen = Tk()
piste = Canvas(fen, width=..., height=..., background='...')
bouton = Button(fen, text='...', command=...)
```

Vous pouvez créer plusieurs `Canvas` ayant chacun son nom, plusieurs boutons ayant chacun une fonction différente. La commande est déclarée lors de la création du bouton sans parenthèse *(exemple : `BouFin=Button(fen, 'Fin de partie', command=fen.destroy)`)*. Vous devrez avoir défini la procédure *(si ce n'est pas une méthode déjà connue de `Tkinter`)* avant la création du bouton.
Attention, si vous créez des `Canvas` et des boutons mais que vous ne les placez pas dans la fenêtre, vous ne les verrez pas. Il faut donc penser à
`piste.pack()` et `bouton.pack()`
Vous pourrez préférer `piste.grid(column=..., row=...)` et `bouton.grid(column=..., row=...)` qui vous permet de posotionner les widgets dans la fenêtre aux emplacements que vous voulez.

Pour dessiner sur un `Canvas` *(appelé ici `piste`, sachant que 0,0 est en haut à gauche)* :
`piste.create_line(xd, yd, xf, yf, width=..., fill=...)`
`xd` et `yd` sont les coordonnées d'une extrémité du segment, l'autre extrémité a our coordonnées `xf` et `yf`. Vous pouvez préciser la largeur du trait *(par défaut, 1 pixel)* et la couleur *(par défaut : `'Black'`)*.
`piste.create_rectangle(xno, yno, xse, yse, fill='...')`
`xno` et `yno` sont les deux coordonnées du point Nord-Ouest du rectangle tandis que `xse` et `yse` sont les coordonnées de l'extrémité opposée *(Sud-Est)*. `fill` désigne la couleur de remplissage. Vous pouvez aussi préciser la largeur du trait qui entourre le rectangle (par défaut : 1 pixel).
`piste.create_circle(xno, yno, xse, yse, fill='...')`
Mêmes commentaires.
`piste.create_text(x, y, text='...', fill = '...', font='...')`
Vous deves définir où vous le placez, quel texte vous écrivez, la couleur, ainsi que la fonte *(police et taille, par exemple `'Times 10'`)*.
`piste.create_image(x,y, nom de fichier)`

Ces méthodes définissent des objets qui ont leurs attributs. Ces attributs sont modifiables à tout instant, de manière simple, et le changement est immédiatement visible à l'écran.
C'est pourquoi je vous recommande de leur donner un nom et d'utiliser alors la méthode itemconfig pour votre `Canvas`.
Pour comprendre la puissance de ce mode de programmation :

```
can = Canvas(fen, height = 200, width = 200, bg='Blue')
texte = can.create_text(100,100, text=' ', fill='Red', font='Times 10')
for k in range(100) :
....can.itemconfig(texte, text=str(k))
....sleep(0.01)
for k in range(100) :
....can.itemconfig(texte, fill=choice(['Blue', 'Green', 'Yellow', 'Black', 'Brown'])
....sleep(0.1)
```

après avoir bien sûr importé `sleep` du module `time` et `choice` du module `random`.
Si écessaire, insérez un `fen.update()` au bon endroit si le rafrachissement de fenêtre ne se fait pas.
Je vous laisse alors imaginer comment vous pouvez effacer une case, la mettre en valeur en changeant (momentanément) sa couleur...

Pour utiliser la souris *(qui est un objet du type event)* :

```
def clic_gauche(event) :
....ix = event.x
....iy = event.y
```

ix et iy vont récupérer les coordonnées sur le `Canvas` de l'endroit où vous aurez cliqué avec la souris. A vous de ne pas oublier le facteur d'échelle avec lequel vous aurez tracé vos cases.

```
piste.bind("<Button-1>", clic_gache)
piste.bind("<Button-3>", clic_droit)
```

Vous pouvez associer une méthode souris à chaque `Canvas` de votre fenètre, et dissocier les clics droit et gauche.

MPSI 2/2014

BASE -10

1-phô

On rappelle le principe de la base -10 : l'entier qui s'écrit avec les chiffres a_0 jusqu'à a_d (c'est dire $\overline{a_d \dots a_0}$ vaut $\sum_{k=0}^{d} a_k.(-10)^k$). Exemple : $\overline{2543}$ est $2.(-1000) + 5.100 + 2.(-10) + 3$.

Ecrivez une procédure qui reçoit une liste `A` de chiffres et calcule la valeur du nombre.

Ecrivez une procédure qui reçoit une liste `A` et indique si l'entier est positif.

Ecrivez une procédure qui, recevant deux listes calcule la somme *(elle aussi en base -10)*. Exemple : listes `[3,4,5,2,0,1]` et `[4,6,2,1,3,2]`, nombres $\overline{102543}$ et $\overline{231264}$, somme $\overline{333607}$. Vous pourrez supposer dans un premier temps que les deux listes ont la même longueur.

Attention au cas de la retenue sur les chiffres de tête, comme $\overline{432} + \overline{751} = \overline{19183}$.

MPSI 2/2014 Bases, système IPT

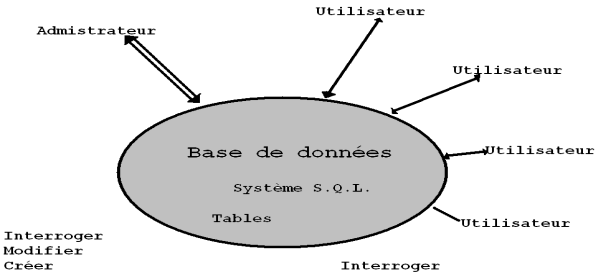
Une base de donnée *(B.D.D. ou B.D.D.R. si relationnelle)* est formée de tables qu'on peut interroger séparément mais aussi par jointure $\bowtie$.

Pour créer, gérer, interroger une base de données, on dispose d'un Système de Gestion de Bases de Données *(S.G.B.D., tel que MySQL, SQlite, Oracle, Dbase, FoxBase...)* qui dispose d'nterfaces plus ou moins conviviales et qu'on peut aussi piloter/émuler avec `Python` par exemple.

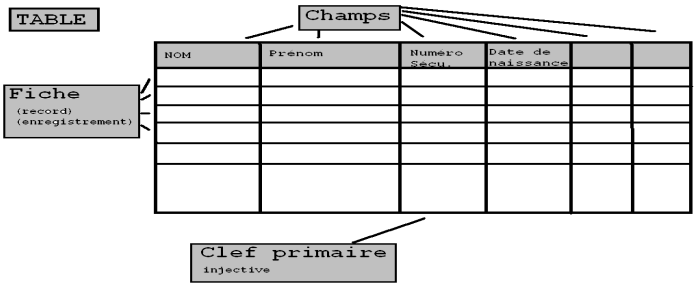
L'administrateur peut créer, effacer, modifier des tables, insérer des enrgistrements.

Suivant son statut *(défini par l'administrateur)* un utilisateur peut interroger la base de données, et éventuellement insérer des enregistrements ou en détruire.

Plusieurs utilisateurs peuvent théoriquement interroger simultanément la base de données sans qu'il n'y ait de problème.



MPSI 2/2014 Tables IPT



Une table est formée d'enregistrements *(en très grand nombre assez souvent)*.

Ces enregistrements sont faits de champs de type numérique/alphabétique/date, de format prédéfini, complétés ou non.

Parmi les différents champs, certains sont des clefs *(il n'y a pas deux enregistrements ayant la même valeur de clef)*. L'une d'entre elles sera la clef primaire et servira à relier des tables entre elles.

MPSI 2/2014 Requêtes. IPT

Ce sera la principale compétence attendue de vous aux concours.

• On peut lire tout le contenu d'une table

`SELECT *` | `FROM (nom_table)` [12].

• On peut ne lire que certains champs

`SELECT (nom_champs)` | `FROM (nom_table)`.

• On peut ne consulter que certaines fiches

`SELECT (nom_champs)` | `FROM (nom_table)` | `WHERE (condition)`.

• On peut effectuer des calculs et statistiques sur certaines champs de certaines fiches

`SELECT COUNT/MAX/MIN` | `FROM (nom_table)` | `WHERE (condition)`.

• On peut lire des informations issues de plusieurs tables ensemble :

[12] la distinction majuscules/minuscules n'est qu'une convention sans importance mais qui permet de s'y retrouver à la lecture

SELECT (nom_champs.table)	FROM (nom_table) JOIN (autre_table) ON (condition)	WHERE (condition)

.
la jointure est une opération qui crée une énorme table "produit cartésien des deux tables en jeu", n'en garde qu'une partie grâce à une condition introduite par `ON`.

Les mots et syntaxes à connaître/maîtriser :

SELECT		FROM		WHERE		SORT BY	
	nom des champs séparés par des virgules		nom d'une table		condition booléenne		critère
	COUNT(*) / MAX(champ) / MIN(champ) / AVG(champ) / SUM(champ)		ou jointure de plusieurs tables avec condition de jointure introduite par ON et une formule booléenne portant sur les deux tables		tests =, <, > LIKE "%_%" NOT, AND, OR BETWEEN		
DELETE INSERT UPDATE		INTO				GROUP BY	

Quand deux tables interviennent par jointure *(ou réunion)*, il faut préciser de quelle table est issu le champ *(structure table.champ ; exemples : clients.nom, consultations.date, ...)*.
Si le nom d'une table devient rébarbatif à écrire, on peut l'alléger par *alias* :
`SELECT cl.nom, cons.date FROM clients AS cl, consultations AS cs WHERE ...`

MPSI 2/2014 — Administrateur — IPT

Il s'agit ici des prérogatives de l'administrateur :

CREATE DATABASE	créer une base relationnelle
CREATE TABLE	créer une table, en définissant ses champs, leur type, diverses conditions [a], en spécifiant qui sera clef primaire...
ALTER TABLE	supprime ou ajoute des champs dans une table
DESCRIBE	
UPDATE	met à jour le contenu d'une table
DROP	efface une table
GRANT	définit les droits d'accès des différents utilisateurs

[a] on peut ainsi demander que le contenu d'un champ ne puisse faire partie que d'une liste préfdéfinie

On peut mettre à jour une base de donnée en important directement un fichier "texte" contenant déjà toutes les données utiles. Avec `Python`, on utilisera `load()`, `read()`, `readln()` et `close()`.
Il est peu probable que les concours vous posent ce type de questions.

M.P.S.I.2 2014 — 1073741907 points — 2015 Charlemagne — p — IPT — q