Informatique
UV21
Correction de l'examen de juin 2008

Exercice :

Question 1.

a) f1 supprime les éléments de L qui sont égaux à x. j est incrémenté de 1 à chaque fois que L[i]=x et permet donc de compter le nombre d'éléments de L qui sont égaux à x.

b)
Initialement : L2 vaut [4,1,12,3,12,12,4,12], v vaut 0, n vaut -1.

Après le 1er passage dans la boucle : t vaut L2[1] donc 4, L2 vaut [1,12,3,12,12,12] (les éléments égaux à 4 sont supprimés), v vaut 2 (le nombre de fois qu'il y avait l'élément 4 dans la liste) et n vaut 4.

Après le 2eme passage dans la boucle : t vaut 1, L2 vaut [12,3,12,12,12], v et n ne changent pas (1 était présent une seule fois dans L2, donc le test est faux).

Après le 3eme passage dans la boucle : t vaut 12, L2 vaut [3], v vaut 4 et n vaut 12.

Après le 4eme passage dans la boucle : t vaut 3, L2 vaut [], v et n ne changent pas (3 était présent une seule fois dans L2, donc le test est faux).

L est vide, donc la boucle est terminée. Après l'appel de f2, v vaut 4 et n vaut 12. n égale l'élément le plus fréquent dans L, et v le nombre de fois que cet élément apparaît dans L.

Question 2.

La fonction f1 n'était pas demandée.
Les paramètres de sortie (et d'entrée-sortie) sont de type name. Attention à l'utilisation d'eval pour ces paramètres (lorsqu'on veut leur valeur).

```
f1:=proc(x::integer,L::name,j::name)
 local i:
 j:=0:
 for i from nops(eval(L)) to 1 by -1 do
  if L[i]=x then
   j:=eval(j)+1:
   L:=subsop(i=NULL,eval(L)):
  fi:
 od:
end:
```

```
f2:=proc(L::list,v::name,n::name)
 local L2::list,t,t2::integer:
 L2:=L:
 v:=0:
 n:=-1:
 while nops(L2)>0 do
  t:=L2[1]:
  f1(t,'L2','t2'):
  if t2>eval(v) then
   v:=t2:
   n:=t:
  fi:
 od:
end:
```

Problème :

Partie 1

1.1.
Il suffit de retrancher l'heure de départ à l'heure d'arrivée (car tout se passe sur une seule journée).

```
Duree:=proc(T::list)
 return(T[4]-T[2]):
end:
```

1.2.
Si l'heure de départ est supérieure ou égale à H, le voyageur peut prendre le train, sinon il ne peut pas.

```
Possible:=proc(T::list,H::integer)
 return(evalb(H<=T[2])):
end:
```

1.3.
Il s'agit de faire une boucle pour parcourir tous les trains. On peut se servir de la fonction *Duree* de la question 1.1. Il faut faire attention de raisonner sur la position (variable *p* ci-dessous) du train dans PB, et non pas directement sur la durée. En effet si la question porte sur la recherche de la durée minimale, la fonction doit retourner la liste correspondant à ce minimum. La variable *p* permet d'enregistrer la valeur de la position de ce minimum.

```
Rapide:=proc(PB::list)
 local i,p::integer:
 p:=1:
 for i from 2 to nops(PB) do
  if Duree(PB[i])<Duree(PB[p]) then p:=i:
  fi:
 od:
 return(PB[p]):
end:
```

1.4.
Les trains étant classes par heure de départ croissante dans PB, il suffit de déterminer le premier train dans la liste que le voyageur peut prendre.

```
Prochain:=proc(PB::list,H::integer)
 local i::integer:
 i:=1:
 while not(Possible(PB[i],H)) do
  i:=i+1:
 od:
 return(PB[i][2]):
end:
```

1.5.
C'est une procédure, qui modifie la liste de trains placée en paramètre. Le paramètre est donc de type *name* (attention à l'évaluer (fonction *eval*) quand on veut utiliser la liste).
Dans la solution proposée, on construit d'abord la liste des trains possibles, puis on remplace PB par cette liste.

```
TrainsOK:=proc(PB::name,H::integer,N::integer)
 local s::exprseq,i::integer:
 s:=NULL:
 for i from 1 to nops(eval(PB)) do
  if Possible(eval(PB)[i],H) and eval(PB)[i][5]<=N then
   s:=s,eval(PB)[i]:
  fi:
 od:
 PB:=[s]:
end:
```

Remarque: comme il est plus simple de construire une séquence qu'une liste, s est de type séquence puis est transformée en liste pour devenir la valeur de PB.

Partie 2

2.1.
Il s'agit de voir si l'heure de départ du train T2 est au moins 2 heures après l'heure d'arrivée du train T1.

```
AllerRetour:=proc(T1::list,T2::list)
 RETURN(evalb(T2[2]>=T1[4]+2)):
end:
```

2.2.
Pour cette question, les couples de trains (les allers-retours) sont obtenus en faisant deux boucles imbriquées. Il faut vérifier si c'est un aller-retour réalisable (grâce à la fonction *AllerRetour*), puis faire les tests pour obtenir à la fin l'aller-retour réalisable le moins cher.

Dans la correction, *p* correspond au prix de l'aller-retour. Conformément à l'énoncé, l'initialisation est faite sur l'aller-retour constitué du premier train aller et du dernier train retour (on ne sait pas si les autres sont réalisables).

```
Economique:=proc(PB::list,BP::list)
 local i,j,p::integer:
 p:=PB[a][5]+BP[r][5]:
 for i from 1 to nops(PB) do
  for j from 1 to nops(BP) do
   if AllerRetour(PB[i],BP[j]) and PB[i][5]+BP[j][5]<p then
    p:=PB[i][5]+BP[j][5]:
   fi:
  od:
 od:
 RETURN(p):
end:
```

Remarque: si on avait voulu d'autres renseignements sur l'aller-retour le moins cher (pas seulement son prix), on aurait utilisé deux variables correspond respectivement à la position dans PB du train de l'aller et à celle dans BP du train du retour pour l'aller-retour le moins cher.

Partie 3

3.1.
La formule à mettre en D7 est: `=B7-A7`
La formule à mettre en E7 est : `=SI(ET(A7>=$D$3;C7<=$B$3);"oui";"non")`
Pour obtenir la plage de D7 à E12, il suffit d'étendre les formules précédentes grâce à la poignée de recopie. Attention aux $ dans la formule, qui permettent de fixer les cellules B3 et D3.

3.2.
La formule à mettre en B17 est: `=MIN(D7:D12)`

3.3.
On peut d'abord utiliser la colonne F pour sélectionner en quelque sorte les trains possibles. Pour les autres, on peut mettre une durée très grande, comme cela ils n'interviendront pas quand on prendra le minimum des durées.
On peut utiliser la colonne F pour mettre une durée grande (>24) si le trajet est impossible, et la durée du trajet si le trajet est possible. Ainsi, les trajets impossibles n'interviendront pas quand on prendre le minimum des durées.
On peut par exemple mettre en F7: `=SI(E7="oui";D7;10000)`
Ensuite on étend la formule jusqu'en F12. Enfin, en B18 on met la formule: `=MIN(F7:F12)`