```
> restart;
```

ÉCOLE POLYTECHNIQUE

CONCOURS D'ADMISSION 2009 FILIÈRES PSI et PT

ÉPREUVE D'INFORMATIQUE

Pour maple irem donne le reste dans la division euclidienne, et iquo le quotient exact; nous utiliserons ces notations.

La manière dont il fallait répondre aux questions n'est pas très claire et ce ne sont pas les rapports concernant l'épreuve qui peuvent nous y aider: le seul disponible est celui de l'année 2006.
Nous apprenons dans ce rapport qu'il faut écrire Maple et non Mapple et que
la syntaxe for avec un while de Maple est fortement déconseillée :
"bien que déjà fortement déconseillée les années précédentes"
alors qu'il n'y a pas eu de rapport les années précédentes, ou du moins, pas public.
Il est possible que cela veuille dire qu'il faut utiliser soit un "for do od" soit un "while () do od".

De manière générale il semble attendu que le candidat utilise les syntaxes les plus universelles possibles et s'interdise les particularités du langage utilisé. Par exemple le fait que dans l'exécution d'une procédure Maple renvoie le dernier résultat calculé ne doit pas être utilisé autrement qu'en rappelant le résultat juste avant la sortie de la procédure.

La démarche semble être : dans la première question on attend une réponse "simple" voire simpliste,
puis les questions suivantes révèlent au candidat des méthodes moins idiotes.
Il ne faut donc pas anticiper et il convient de lire plusieurs questions d'avance avant de commencer à écrire.

Il semble aussi que ce qui est apprécié c'est que les procédures écrites dans les questions précédentes soient réutilisées dans les questions suivantes, même si cela conduit à des lourdeurs en terme de temps de calcul.
De toutes manières cela peut permettre de gagner du temps dans la rédaction des réponses aux questions.
Il vaut mieux optimiser le temps de rédaction des procédures que leur temps d'exécution.

I. Nombres premiers

Question 1

```
> estPremier:=proc(n)
  local d, result;

  result:=1;
  for d from 2 to n-1 do
   if irem(n,d)=0 then result:=0 fi
  od;
  return result;
```

```
end:
```

On peut aussi arrêter aussitôt qu'on trouve un facteur

```
> estPremier:=proc(n)
  local d;

  d:=2;
  while (d<n) and not(irem(n,d)=0) do
   d:=d+1
  od;
  if d=n then return 1 else return 0 fi
  end:
```

Si n n'est pas premier, à la sortie de la boucle d est le plus petit diviseur >1,

et il est <n.
Si n est premier on sort avec d=n; d est encore le plus petit diviseur.

Question 2

```
> petitsPremiers:=proc(n)
  local q, i;
  global premier;

  i:=0;
  for q from 2 to n do
   if estPremier(q)=1 then
    i:=i+1;
    premier[i]:=q;
   fi;
  od;
  return i;
  end:
```

Le tableau premier est effectivement un tableau et non une liste pour Maple. On n'a pas à gérer sa taille.
La variable i est un compteur et en même temps l'indice dans le tableau "premier".

```
> nbr:=petitsPremiers(17);
  seq(premier[i],i=1..nbr);
```

$$nbr := 7$$

$$2, 3, 5, 7, 11, 13, 17$$

Question 3

```
> petitsPremiers2:=proc(n)
  local max, i, cont, p,q,estpremier, nbrpremier;
  global premier;

  premier[1]:=2;nbrpremier:=1;q:=3;
  while(q<=n) do
    max:=floor(sqrt(q));
    cont:=true;
    estpremier:=true;
    i:=2;
    while(premier[i]<=max) and cont do
      if irem(q,premier[i])=0 then estpremier:=false; cont:=false fi;
```

```
      i:=i+1
    od;
    if estpremier then
      nbrpremier:=nbrpremier+1;
      premier[nbrpremier]:=q;
    fi;
   q:=q+2
  od;
  return nbrpremier;
  end:
```

On met 2 en première place dans le tableau "premier"et ensuite on ne s'occupe que des entiers impairs q.
On donne à q toutes les valeurs impaires possibles, et pour q fixé on cherche s'il a des diviseurs premiers dans le tableau "premier".
En entrée de boucle tous les nombres premiers impairs < q sont dans le tableau premier (vrai pour q=3).
Si q n'est pas premier son plus petit diviseur premier p est tel que p<p^2<=q; ce nombre premier sera trouvé dans la liste des nombres premiers < q et <=max.
Si q est premier, le plus grand nombre premier p<q est tel que p^2>q (ce n'est pas évident, mais c'est vrai et suggéré par l'énoncé); donc on va trouver un nombre premier p<q tel que p>max.
On vérifie que dans les deux cas, après avoir ajouté 2 à q, les nombres premiers impairs <q sont encore tous dans le tableau "premier". Cette propriété est un invariant de boucle.
On sort de la boucle avec une valeur de q qui est >n, mais c'est n+1 si n est pair ou n+2 si n est impair.
Dans les deux cas il est clair que les nombres premiers impairs < q, sont bien les nombres premiers impairs qui sont <=n. Le tableau "premier" contient donc les nombres premiers <= n.

```
> st:=time():
  nbr:=petitsPremiers2(23654);
  st:=time()-st:
  st,s;
  seq(premier[i],i=1..nbr):
  premier[nbr];
```

$$nbr := 2631$$

$$27.541, s$$

$$23633$$

Variante sans utilisation de la racine carrée

```
> petitsPremiers2bis:=proc(n)
  local i, cont, p,q,estpremier, nbrpremier;
  global premier;

  premier[1]:=2;nbrpremier:=1;q:=3;
  while(q<=n) do
    cont:=true;
    estpremier:=true;
    i:=2;
    while (premier[i]^2<= q) and (cont) do
      if irem(q,premier[i])=0 then estpremier:=false;cont:=false fi;
      i:=i+1
    od;
    if estpremier then
      nbrpremier:=nbrpremier+1;
```

```
      premier[nbrpremier]:=q;
    fi;
   q:=q+2
  od;
  return nbrpremier;
  end:
```

```
> st:=time():
  nbr:=petitsPremiers2bis(23654);
  st:=time()-st:
  st,s;
  premier[nbr];
```

$$nbr := 2631$$

$$1.678, s$$

$$23633$$

Conclusion : au moins avec Maple l'essentiel du temps de calcul dans le premier algorithme est passé à calculer les racines carrées.
Le temps de calcul peut varier, mais un moyenne le temps de

calcul pour le premier est 20 plus grand que pour le deuxième. Si on veut utiliser une racine carrée, il faudrait utiliser un algorithme

vraiment très rapide pour la calculer.

Question 4

```
> petitsPremiers3:=proc(n)
  local cont,marque,c1,c2, nbrpremier;
  global premier;

  nbrpremier:=0;
  for c1 from 1 to n do marque[c1]:=false od;
  c1:=1;
  while(c1<n) do
    cont:=true;
    while(cont and (c1<n)) do
      c1:=c1+1;
      cont:=marque[c1]
    od;
    if not(cont) then
      nbrpremier:=nbrpremier+1;premier[nbrpremier]:=c1;
      c2:=c1+c1;
      while(c2<= n) do
        marque[c2]:=true;
        c2:=c2+c1;
      od;
    fi;
  od;
  return nbrpremier;
  end:
```

On initialise le sable avec une case 1 en plus qui est le bord gauche du tableau pour simplifier.
On pose le caillou 1 sur cette première case (c1:=1).

B) On avance le caillou1 jusqu'à trouver une case non marquée

si on n'en trouve pas c'est fini sinon
le caillou1 est sur une case correspondant à un nombre premier, on l'enregistre

puis on coche les multiples suivants de c1, on recommence à B)

F) sauf si le caillou 1 est arrivé à la dernière place.

Remarquons que nous n'avons utilisé que des additions et inégalités.

```
> nbr:=petitsPremiers3(54);
  seq(premier[i],i=1..nbr);
```

$$nbr := 16$$

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53$$

Question 5

```
> factoriser:=proc(n)
  local m,i,j,p,nbr;
  global premier,facteur;

  nbr:=petitPremiers3(n);
  m:=n;i:=0;j:=0;
  while(m>1) do
    i:=i+1;
    p:=premier[i];
    while(irem(m,p)=0) do
      j:=j+1;facteur[j]:=p;m:=iquo(m,p);
    od
  od;
  return j;
  end:
```

On calcule le tableau des nombres premiers <= n.
Au départ m = n et quand on a trouvé un facteur premier p on remplace m par m/p.

On ne passe pas au nombre premier suivant tant qu'on n'a pas épuisé ce facteur premier.
On arrête que le nombre m restant à factoriser vaut 1.

```
> nbr:=factoriser(2^4*7^2*4199);
  seq(facteur[i],i=1..nbr);
```

$$nbr := 9$$

$$2, 2, 2, 2, 7, 7, 13, 17, 19$$

On pourrait aussi ne calculer que les nombres premiers <= sqrt(n).
Avant d'appeler premier[i] il suffirait de tester si i >nbr et si c'est le cas c'est que n est premier.
On enregistrerait ce facteur n (d'exposant 1) et terminerait en posant m=1.

Question 6

```
> factoriser2:=proc(n)
  local m,j,q;
  global facteur;
```

```
  m:=n;j:=0;
  while(irem(m,2)=0) do
    j:=j+1;m:=iquo(m,2);facteur[j]:=2
  od;
  q:=1;
  while(m>1) do
    q:=q+2;
    if q^2>m then
      j:=j+1;facteur[j]:=m;m:=1
    else
    while(irem(m,q)=0) do
      j:=j+1;facteur[j]:=q;m:=iquo(m,q);
    od
    fi;
  od;
  return j;
  end:
```

On se débarrasse du facteur éventuel 2.
Le nombre restant à factoriser est m.
On essaye de diviser m par les nombres impairs q, dans l'ordre croissant à partir de 3, mais si q^2 devient strictement plus grand que le nombre m, cela signifie que m est premier;
m est alors le plus grand facteur de n et on termine en l'ajoutant à la liste et en posant m=1.

```
> st:=time():
  nbr:=factoriser2(559577854879808);
  st:=time()-st;
  st,s;
  seq(facteur[i],i=1..nbr);
```

$$nbr := 9$$

$$st := 3.548$$

$$3.548, s$$

$$2, 2, 2, 2, 2, 2, 23, 378523, 1004293$$

Variante : les carrés des nombres impairs sont calculables sans multiplication car (2k+1)^2-(2k-1)^2=8k qui est l'incrément.

```
> factoriser2bis:=proc(n)
  local m,j,q,increment,cq;
  global facteur;

  m:=n;j:=0;
  while(irem(m,2)=0) do
    j:=j+1;m:=iquo(m,2);facteur[j]:=2
  od;
  q:=1;cq:=1;increment:=0;
  while(m>1) do
    q:=q+2;increment:=increment+8;cq:=cq+increment;
    if cq>m then
      j:=j+1;facteur[j]:=m;m:=1
    else
    while(irem(m,q)=0) do
      j:=j+1;facteur[j]:=q;m:=iquo(m,q);
```

```
    od
    fi;
  od;
  return j;
  end:
```

```
> st:=time():
  nbr:=factoriser2bis(559577854879808);
  st:=time()-st;
  st,s;
  seq(facteur[i],i=1..nbr);
```

$$nbr := 9$$

$$st := 3.045$$

$$3.045, s$$

$$2, 2, 2, 2, 2, 2, 23, 378523, 1004293$$

Conclusion : joli, mais pas franchement décisif.
On peut essayer la méthode suggérée par l'énoncé, en calculant la racine carrée de m
On change la valeur de max uniquement quand m change, c'est-à-dire quand on trouve un facteur premier (impair)

```
> factoriser2ter:=proc(n)
  local m,j,q,max;
  global facteur;

  m:=n;j:=0;
  while(irem(m,2)=0) do
    j:=j+1;m:=iquo(m,2);facteur[j]:=2
  od;
  q:=1;
  max:=floor(sqrt(m));
  while(m>1) do
    q:=q+2;
    if q>max then
      j:=j+1;facteur[j]:=m;m:=1
    else
    while(irem(m,q)=0) do
      j:=j+1;facteur[j]:=q;m:=iquo(m,q);max:=floor(sqrt(m));
    od
    fi;
  od;
  return j;
  end:
```

```
> st:=time():
  nbr:=factoriser2ter(559577854879808);
  st:=time()-st;
  st,s;
  seq(facteur[i],i=1..nbr);
```

$$nbr := 9$$

$$st := 2.403$$

$$2.403, s$$

$$2, 2, 2, 2, 2, 2, 23, 378523, 1004293$$

Conclusion : la situation est tout à fait différente de celle de la question 3 (petitsPremier2).
Ici on calcule la racine carrée un nombre de fois égal au nombre de facteurs premiers impairs de n.
Alors qu'on devrait calculer le carré de q un nombre de fois de l'ordre de sqrt(n)/2 si n est premier.
Bien entendu le test " if q > floor(sqrt(m)) " réduirait à néant l'avantage, jusqu'à rendre le calcul pratiquement impossible.

```
> factoriser2quat:=proc(n)
  local m,j,q;
  global facteur;

  m:=n;j:=0;
  while(irem(m,2)=0) do
    j:=j+1;m:=iquo(m,2);facteur[j]:=2
  od;
  q:=1;
  while(m>1) do
    q:=q+2;
    if q>floor(sqrt(m)) then
      j:=j+1;facteur[j]:=m;m:=1
    else
    while(irem(m,q)=0) do
      j:=j+1;facteur[j]:=q;m:=iquo(m,q);
    od
    fi;
  od;
  return j;
  end:
```

```
> st:=time():
  nbr:=factoriser2quat(559577854879808);
  st:=time()-st;
  st,s;
  seq(facteur[i],i=1..nbr);
```

```
Warning,  computation interrupted
```

$$st := 89.134$$

$$89.134, s$$

$$2, 2, 2, 2, 2, 2, 23, facteur_8, facteur_9, facteur_{10}, facteur_{11}, facteur_{12}, facteur_{13}, facteur_{14}, facteur_{15}, facteur$$

II. Reconnaître les puissances

Question 7

```
> calculerAlpha:=proc(n)
  local m,k,q,s;
  global alpha;

  m:=n;
  s:=0;k:=0;
  while(irem(m,2)=0) do
    s:=s+1;m:=iquo(m,2)
  od;
```

```
  if s>0 then k:=k+1;alpha[k]:=s fi;
  q:=1;
  while(m>1) do
    q:=q+2;
    if q^2>m then
      k:=k+1;alpha[k]:=1;m:=1
    else
      s:=0;
      while(irem(m,q)=0) do
        s:=s+1;m:=iquo(m,q);
      od;
      if s>0 then k:=k+1;alpha[k]:=s fi;
    fi;
  od;
  return k;
  end:
```

L'algorithme est proche de celui utilisé pour factoriser, mais on calcule les multiplicités et on ne garde que ça.
A remarquer que si q^2 devient > m, c'est que m est premier et c'est un facteur de multiplicité 1.

```
> nbr:=calculerAlpha(45678*2);
  seq(alpha[i],i=1..nbr);
```

$$nbr := 4$$

$$2, 1, 1, 1$$

Question 8

```
> estPuissance:=proc(n,b)
  local nbr,i,result;
  global alpha;

  nbr:=calculerAlpha(n);
  result:=1;
  i:=1;
  while (i<=nbr) and (irem(alpha[i],b)=0) do
    i:=i+1
  od;
  if i>nbr then return 1 else return 0 fi;
  end:
```

On arrête aussitôt qu'on a trouvé une multiplicité qui ne convient pas.
Evidemment le calcul du tableau de toutes les multiplicités n'est pas indispensable.

On aurait pu réécrire "calculerAlpha" sans enregistrer les multiplicités et en sortant

aussitôt trouvé une multiplicité qui ne convienne pas.
Si toutes conviennent, n est bien une puissance b ième.

```
> estPuissance(3^5*4^10,5);
  estPuissance(3^5*4^10,3);
```

$$1$$

$$0$$

III. Nombres de Carmichael

Question 9

Il suffit de tester si le nombre est le carré de la partie entière de sa racine carrée.
Voici un algorithme de calcul de la partie entière de la racine carré de n entier >=0

```
> EntRac:=proc(n) local result;
  result:=n;
  while(result^2>n) do
    result:=iquo(result+iquo(n,result),2)
  od;
  return result;
  end:
```

```
> N:=67678965696079857678078097807567858348768078070870970878 5;
  st:=time():
  R:=EntRac(N);
  time()-st,s;
```

$$N := 676789656960798576780780978075678583487680780708709708785$$

$$R := 260151812786457164979429828 95$$

$$0.004, s$$

Pour réaliser la même chose avec la fonction sqrt de Maple il faut augmenter "suffisamment" le nombre de décimales,

ce qui ralenti évidemment le calcul.

```
> st:=time():
  R2:=floor(evalf(sqrt(N)));
  time()-st,s;
  st:=time():
  Digits:=floor(ln(N)/ln(10)/2)+5;
  R2:=floor(evalf(sqrt(N)));
  time()-st,s;
  Digits:=10:
```

$$R2 := 260151812800000000000000000 00$$

$$0.001, s$$

$$Digits := 33$$

$$R2 := 260151812786457164979429828 95$$

$$0.007, s$$

Question 10

```
> estCarmichael:=proc(c)
  local result, i, nbr,s,m,q;

  result:=1;m:=c;
  if irem(m,2)=0 then result:=0 fi;
  q:=1;
  while(m>1) and (result=1) do
    q:=q+2;
```

```
    if q^2>m then
      if m=c then
        result:=0
      else
        if not(irem(c-1,m-1)=0) then result:=0 fi;
        m:=1
      fi;
    elif irem(m,q)=0 then
        m:=iquo(m,q);
        if irem(m,q)=0 or not(irem(c-1,q-1)=0) then
          result:=0
        fi;
    fi;
  od;
  return result;
  end:
```

On commence par éliminer le cas où c est divisible par 2.

On va ensuite passer en revue les nombres impairs en utilisant le canevas de factoriser2.
Quand on trouve un facteur qui ne convient pas,

on pose result=0 ce qui fait sortir immédiatement de la boucle; sinon on divise par ce facteur ce qui donne m.
Si le nombre q essayé a son carré >m , c'est que m est le plus grand des facteurs de c et qu'il est simple.
Il reste alors à vérifier que m n'est pas c, sinon c est premier, et que m-1 divise c-1.

Sinon, et que q divise c, c n'est pas premier et il faut vérifier que ce facteur q est simple et que q-1 divise c-1.

```
> calculerCarmichael2:=proc(n)
  local i,k;
  global carmichael;

  k:=0;
  for i from 3 to n do
    if estCarmichael(i)=1 then k:=k+1;carmichael[k]:=i fi;
  od;
  return k;
  end:

  st:=time():
  nbr:=calculerCarmichael2(10000);
  time()-st,s;
  seq(carmichael[i],i=1..nbr);
```

$$nbr := 7$$

$$1.328, s$$

$$561, 1105, 1729, 2465, 2821, 6601, 8911$$

Le premier nombre de Carmichael est 561, ce qui est bien connu.

Question 11

```
> calculerCarmichael3:=proc(n)
  local nbr,i,j,k,h,p,q,r,c;
```

```
  global premier,carmichael;

  nbr:=petitsPremiers3(n);
  h:=0;
  for i from 2 to nbr-2 do
    p:=premier[i];
    for j from i+1 to nbr-1 do
      q:=premier[j];
      for k from j+1 to nbr do
       r:=premier[k];
       c:=p*q*r;
       if (c<=n) and (irem(c-1,p-1)=0) and (irem(c-1,q-1)=0) and
(irem(c-1,r-1)=0)  then
         h:=h+1; carmichael[h]:=c
       fi;
      od
    od
  od;
  return h
  end:
```

La procédure est facile à comprendre, on essaye tous les "produits possibles" de 3 nombres premiers deux à deux distincts.
Mais on pourrait assez facilement restreindre le nombre de ces "produits possibles"; par exemple on a p<q<r et pqr<=n donc

p^3 < r et pour p fixé q^2 < n/p.

```
> st=time():
  nbr:=calculerCarmichael3(2000);
  time()-st,s;
  seq(carmichael[i],i=1..nbr);
```

$$nbr := 3$$

$$54.482, s$$

$$561, 1105, 1729$$

Concrètement énumérer les nombres de Carmichael par la méthode "évidente" n'est pas si coûteux que ça.

```
> calculerCarmichael4:=proc(n)
  local nbr,i,j,k,h,p,q,r,c,pr;
  global premier,carmichael;

  nbr:=petitsPremiers3(n);
  h:=0;
  for i from 2 to nbr-2 while(premier[i]^3 < n)do
    p:=premier[i];
    for j from i+1 to nbr-1 while(p* premier[j]^2<n) do
      q:=premier[j];pr:=p*q;
      for k from j+1 to nbr while(pr*premier[k]<=n)do
       r:=premier[k];
       c:=p*q*r;
       if (irem(c-1,p-1)=0) and (irem(c-1,q-1)=0) and (irem(c-1,r-1)=0)
```

```
  then
        h:=h+1; carmichael[h]:=c
      fi;
    od
  od
od;
return h
end:
```

```
> st=time():
  nbr:=calculerCarmichael4(2000);
  time()-st,s;
  seq(carmichael[i],i=1..nbr);
```

$$nbr := 3$$

$$1.438, s$$

$$561, 1105, 1729$$

Question 12

```
> calculerCarmichael:=proc(n)
  local i,k,nbr,incr,p,ctable;
  global premier,carmichael;

  nbr:=petitsPremiers2(floor(sqrt(n))-1);
  for i from 1 to n do
    ctable[i]:=i
  od;
  for k from 2 to nbr do
    p:=premier[k];
    incr:=p*(p-1);
    i:=p+incr;
    while(i<=n) do
      ctable[i]:=iquo(ctable[i],p);
      i:=i+incr;
    od;
  od;
  k:=0;
  for i from 2 to n do
    if ctable[i]=1 then
      k:=k+1;carmichael[k]:=i;
    fi;
  od;
  return k;
  end:
```

La table "t" est ici notée ctable.
La valeur maximale de l'indice est n puisque la seule condition est que les nombres de Carmichael <= n soient des indices valides.
Les cases dont la valeur peut être modifiée sont les cases dont l'indice est >=3^2=9.
Les précédentes ne servent donc à rien; en particulier la case de numéro 0.
On se demande donc pourquoi l'énoncé précise que "t" contient n+1 cases.
Pour simplifier les écritures on prend une table dont l'indice varie de 1 à n.

On initialise la table comme demandé, on calcule la liste des nombres premiers à utiliser (on laisse 2 qui est

le premier).
Pour chacun des nombres premiers p possibles, à partir de i= p^2 et tant que i reste <= n, on fait t[i]=t[i]/p et on incrémente i de p(p-1).

On passe en revue ctable (à partir de l'indice 2, mais on pourrait partir de 9) et quand on trouve la valeur 1 on ajoute le nombre à la liste des nombres de Carmichael.

```
> st:=time():
  nbr:=calculerCarmichael(50000);
  time()-st,s;
  seq(carmichael[i],i=1..nbr);
```

$$nbr := 12$$

$$0.895, s$$

$$561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657$$

C'est effectivement plus rapide, mais plus encombrant.

```
> map(factoriser2,[seq(carmichael[i],i=1..nbr)]);
```

$$[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 3]$$

Un nombre de Carmichael ne peut pas avoir 2 facteurs premiers, sinon comme pq -1 = p(q-1)+p-1, q-1 devrait diviser p-1 et réciproquement par symétrie donc p=q.
Le premier nombre de Carmichael qui ait 4 facteurs est 41041.

```
>
```