Informatique en Degead 1

—

Algorithmes en Maple (3/3)

Responsable : Denis Cornaz

denis.cornaz@dauphine.fr
01 44 05 41 83
P 409 bis

http://www.lamsade.dauphine.fr/∼cornaz/Enseignement/UV21-Degead1/

Université Paris-Dauphine

Planning (rappel)

- Du 24 Septembre au 28 Octobre puis du 5 au 11 Novembre :
  Semaines 1-6 (S1A)
  1h30 d'informatique par semaine en salle info :
  - Semaines 1-2 = prise en main Excel
  - Semaines 3-6 = Algorithmique en Maple
  - 3 amphis avec moi = 3, 10 et 24 Octobre
- Du 29 Octobre au 4 Novembre : Semaine de consolidation
- Du 12 Novembre au 23 Décembre : Semaines 7-12 (S1B)
  3h d'informatique par semaine =
  1h30 en salle info + 1h30 en salle de cours
  - Algorithmique en Maple
- 4 dernières semaines du semestre =
  2 de vacances + 1 de révision + 1 d'examen.

Évaluation (rappel)

- Note de Contrôle continu sur 20 : 3 interrogations en TD = 1 sur 8 (45 min en semaine 4) + 1 sur 12 (1h30 en semaine 8)
- Note d'examen sur 20 en Amphi (1h30 sans aucun document) = 5 points sur Excel + 15 points sur Maple
- Note finale = 0.5CC + 0.5E
  (ou zéro en cas de 6 absences justifiées ou non)

Mots clés en Maple (rappel)

Seuls sont autorisés :

- les séparateurs : `,` `[` `]` `(` `)` `:` `;`
- l'affectation : `:=`
- opérateurs arithmétiques : `+`,`-`,`*`,`/` et `<`, `>`, `<=`, `>=`, `=`, `<>`
- les mots clés : `proc`, `local`, `RETURN`, `end`, `if`, `then`, `fi`, `true`, `false`, `and`, `or`, `not`, `while`, `do`, `od`, `for`, `from`, `to`, `op`, `nops`, `NULL`.

Procédure (rappel)

Syntaxe

nom de la procédure :=proc(*paramètre d'entrée*)
    instructions
RETURN(*paramètre de sortie*)
end:

Ex :

```
> Plus4:=proc(x)
      RETURN(x+4):
    end:
> Plus4(6);
                                  10
> x:=3: x:=Plus4(x):
> x;
                                   7
```

Procédure appelant d'autres procédures (rappel)

Ex : Procédure `Plus6()` appelant la procédure `Plus4()`

```
> Plus6:=proc(x)
      RETURN(Plus4(x)+2):
    end:
> Plus6(6);
                                    12
```

Variables locales (rappel)

Syntaxe

nom de la procédure `:=proc(` *paramètre d'entrée*`)`
    déclaration des variables locales
    instructions
`RETURN(`*paramètre de sortie*`)`
`end:`

Ex : Redéfinir `Plus4()`

```
> Plus4:=proc(x)
      local y:                          #variable locale
      y:=x+4:
      RETURN(y):
   end:
```

Boucle `while` (rappel)

Syntaxe

`while` *condition* `do`
    instructions
`od:`

Ex :

```
> x:=3:
> while x <= 10 do x:=x+2; od:
> x;
                          11
```

Test `if` (rappel)

Syntaxe

`if` *expression booléenne* `then`
    instructions
`fi:`

Ex :

```
> x:=3:
> if x <= 10 then x:=x+2: fi:
> x;
                          5
> if x > 10 then x:=0: fi:
> x;
                          5
```

Test `if` (remarque)

Syntaxe

`if` *expression booléenne* `then`

    instructions

`fi:`

équivalent à

if avec while

b := true

`while` *expression booléenne* `and` *b* `do`

    instructions

    b :=false

`od:`

Boucle `for` (rappel)

Syntaxe

`for` *variable* `from` *valeur initiale* `to` *valeur finale* `do:`
    instructions
`od:`

Ex :

```
> x:=0:
> for i from 1 to 100 do: x:= x+i: od:
> x;
                              5050
```

Boucle `for` (remarque)

Syntaxe

`for` *variable* `from` *valeur initiale* `to` *valeur finale* `do:`
    instructions
`od:`

équivalent à

for avec while

i := valeur initiale
`while` i <= *valeur finale* `do`
    instructions
    i :=i+1
`od:`

Séquences (rappel)

Séquences : concaténation opérateur " ,"

```
> S1:=NULL:
> S2:=3,4,5,1,2:
> S1:=S1,6:
> S1;
                              6
> S:=S2,S1;
                     S:=3,4,5,1,2,6
```

`NULL` : séquence vide

`S1,S2` : séquence constituée des éléments de `S1` puis de `S2`

Listes (rappel)

Listes : accès direct `[]`, longueur `nops`

```
> L1:=[]:
> L2:=[S]:
> L2[3];
                              5
> L1:=[op(L2),4,5];
          L1:=[3,4,5,1,2,6,4,5]
> nops(L1);
                             8
```

`[]` : liste vide

`L[i]` : ième élément de la liste `L`

`nops(L)` : nombre d'éléments de la liste `L`

`op(L)` : séquence constituée des éléments de la liste `L`

`[S]` : liste constituée des éléments de la séquence `S`

Maple annale 2014

Étant donnés deux entiers a et b, tels que $a \geq b$, il existe deux uniques entiers q (le quotient) et r (le reste) tels que $a = qb + r$ avec $0 \leq r < b$.

1) Écrire une fonction `MyIrem:=proc(a,b)` prenant deux entiers a, b ($a \geq b$) en paramètre et qui renvoie la valeur du reste. Écrire aussi une fonction `MyIquo:=proc(a,b)` renvoyant la valeur du quotient.
Par-exemple :

```
> MyIrem(17,5);
                                    2

> MyIquo(17,5);
                                    3
```

Maple annale 2014 (correction)(rappel)

1)

```
> MyIquo:=proc(a,b)
      local q:
      q:=0:
      while q*b <= a do
         q:=q+1:
      od:
      RETURN(q-1):
   end:

> MyIrem:=proc(a,b)
      RETURN(a-MyIquo(a,b)*b):
   end:
```

Maple annale 2014 (énoncé)

2) Écrire une fonction `MyMember:=proc(L,x)` prenant une liste L d'entiers et un entier x en paramètre et retournant `true` si x est un élément de L, `false` sinon. Par-exemple :

```
> MyMember([1,5,86,4,2],3);
                                    false
> MyMember([1,5,86,4,2],4);
                                    true
```

Maple annale 2014 (correction)

2)

```
> MyMember:=proc(L,x)
        local i:
        for i from 1 to nops(L) do
              if L[i]=x then RETURN(true): fi:
        od:
        RETURN(false):
   end:
```

Maple annale 2014 (énoncé)

3) Écrire une fonction `Clean:=proc(L)` prenant une liste L d'entiers en paramètre et retournant la liste de tous les entiers distincts de L :
Par-exemple :

```
> Clean([4,6,4,1,78,4]);
                                [4,6,1,78]
```

Maple annale 2014 (correction)

3)

```
> Clean:=proc(L)
        local S:
        S:=NULL:
        for i from 1 to nops(L) do
          if not MyMember([S],L[i]) then S:=S,L[i]:  fi:
        od:
        RETURN([S]):
   end:
```

Maple annale 2014 (énoncé)

4) Écrire une fonction `AllEven:=proc(L)` prenant une liste L d'entiers en paramètre et retournant la liste de tous les entiers pairs distincts de L :
Par-exemple :

```
> AllEven([4,6,89,4,1,75,4,0,56]);
                                    [4,6,0,56]
```

Maple annale 2014 (correction)

4)

```
> AllEven:=proc(L)
        local S:
        S:=NULL:
        for i from 1 to nops(L) do
          if MyIrem(L[i],2)=0 then S:=S,L[i]:  fi:
        od:
        RETURN(Clean([S])):
   end:
```

Maple annale 2014 (énoncé)

5) Écrire une fonction `FizBuz:=proc(x)` prenant un entier x en paramètre et retournant `true` si x est un multiple de 5 ou de 7 exclusivement, et `false` sinon. Par-exemple :

```
> FizBuz(6);
                                                    false
> FizBuz(15);
                                                    true
> FizBuz(35);
                                                    false
```

Écrire différemment la même fonction, que l'on appellera `FizBuz2:=proc(x)`. La différence doit résider dans la façon d'écrire le test sur les variables booléennes.

Maple annale 2014 (correction)

5)

```
> FizBuz:=proc(x)
        RETURN(
                (MyIrem(x,5)=0 and MyIrem(x,7)>0)
            or (MyIrem(x,5)>0 and MyIrem(x,7)=0)
         ):
   end:


> FizBuz2:=proc(x)
        RETURN(
                (MyIrem(x,5)=0 or MyIrem(x,7)=0)
            and (MyIrem(x,5)>0 or MyIrem(x,7)>0)
         ):
   end:
```

Supprimer le *n*ème élément d'une liste (rappel)

```
> L:=[a,b,c,d,e]:
> MyRemove(L,3);
                                [a, b, d, e]
```

Solution :

```
> MyRemove:= proc(L,n)
          local i,S:
          S:=NULL:
          if nops(L) < n then RETURN(L): fi:
          for i from 1 to nops(L) do:
                    if i <> n then S:=S,L[i]: fi:
          od:
          RETURN([S]):
    end:
```

Remarque finale 1

```
> if a or b then ... fi:
```

équivalent à

```
> if not(not a and not b) then ... fi:
```

Remarque finale 2

```
> if a and b then ... fi:
```

équivalent à

```
> if not(not a or not b) then ... fi:
```