```
> #Exercice 1 : Ecrire une procédure "longueurliste" qui prend une
  liste d'entiers (sauf le dernier élément), qui est soit vide,
  soit dont le dernier est F, en argument et renvoie la longueur
  de cette liste, et cela de trois façons différentes, dont une en
  utilisant une boucle "for" et une autre avec un "while". Par
  définition, dans notre cas, la longueur d'une liste non vide est
  égale à la longueur de la liste moins 1 (on ne compte pas le
  dernier élément F) et la longueur d'une liste vide est zéro.
>
> #1ère façon :
> longueurliste1:=proc(L)
  if (nops(L)=0) then return 0;
  else return nops(L)-1;
  end if;
  end proc;
```

longueurliste1 :=

proc(*L*) **if** nops(*L*) = 0 **then return** 0 **else return** nops(*L*) − 1 **end if end proc**

```
> longueurliste1([]),longueurliste1([F]),longueurliste1([1,2,7,F])
  ;
```

0, 0, 3

```
> #2ième façon : Avec "for"
> longueurliste2:=proc(L)
  local long;
  if (L=[]) then return 0;
  else
  for long from 1 to infinity do
  if L[long]=F then return long-1;
  end if;
  end do;
  end if;
  end proc;
```

longueurliste2 := **proc**(*L*)

local *long*;

if *L* = [ ] **then return** 0

else for *long* **to** ∞ **do if** *L*[*long*] = *F* **then return** *long* − 1 **end if end do**

end if

end proc

```
> longueurliste2([]);
```

0

```
> longueurliste2([F]);
```

0

```
> longueurliste2([1,2,7,F]);
```

3

```
> #3ième façon : avec un "while"
```

```
> longueurliste3:=proc(L)
  local long;
  long:=1;
  if (L=[]) then return 0;
  else
  while (L[long]<>F) do
  long:=long+1;
  end do;
  return long-1;
  end if;
  end proc;
```

longueurliste3 := **proc**(*L*)

local *long*;

long := 1;

if *L* = [ ] **then return** 0

else while *L*[*long*] ≠ *F* **do** *long* := *long* + 1 **end do**; **return** *long* – 1

end if

end proc

```
> longueurliste3([]);
```

0

```
> longueurliste3([F]);
```

0

```
> longueurliste3([1,2,7,F]);
```

3

```
> ##############################################################
  ############
> #Exerice 2 : Ecrire une procédure OpListe qui prend en argument
  une liste de nombres ou de polynômes, ainsi que soit `+` soit
  `*` et renvoie la somme (ou le produit) des éléments de cette
  liste. Faire cela de trois façons différentes (dont une avec
  for, et une autre avec while).

> OpListe1:=proc(L,s)
  if L=[] then return 0;end if;
  if(s=`+` or s=`*`) then return convert(L,s);
  end if;
  end proc;


>
```

OpListe1 := **proc**(*L*, *s*)

if *L* = [ ] **then return** 0 **end if**; **if** *s* = `+` **or** *s* = `*` **then return** convert(*L*, *s*) **end if**

end proc

```
> OpListe1([2,2,3],`+`);
```

```
                                  7
> OpListe1([],s);
                                  0
> OpListe1([],`+`);
                                  0
> OpListe1([x,2*x,3*x],`+`);
                                6 x
> OpListe1([x,2*x,3*x],`*`);
                              6 x^3
> OpListe2:=proc(L,s)
  local l,i,res;
  l:=nops(L);
  if s=`+` then
  res:=0;
  if (l=0) then return 0;
  else
  for i from 1 to l do
  res:=res+L[i];
  end do;
  end if;
  else
  res:=1;
  if (l=0) then return 1;
  else
  for i from 1 to l do
  res:=res*L[i];
  end do;
  end if;
  end if;
  end proc;
```

OpListe2 := **proc**(*L*, *s*)
local *l*, *i*, *res*;
l := nops(*L*);
if *s* = `+` **then**
res := 0; **if** *l* = 0 **then return** 0 **else for** *i* **to** *l* **do** *res* := *res* + *L*[*i*] **end do end if**
else *res* := 1; **if** *l* = 0 **then return** 1 **else for** *i* **to** *l* **do** *res* := *res*∗*L*[*i*] **end do end if**
end if
end proc

```
> OpListe2([],`+`), OpListe2([],`*`);
                               0, 1
> OpListe2([x,2*x,3*x],`+`);
                                6 x
> OpListe2([x,2*x,3*x],`*`);
                              6 x^3
```

```
> OpListe3:=proc(L,s)
  local l,i,res;
  l:=nops(L);
  if (s=`+`) then
  if (l=0) then return 0; else
  i:=1;res:=0;
  while(i<=l) do
  res:=res+L[i];
  i:=i+1;
  end do;
  end if;
  end if;
  if (s=`*`) then
  if (l=0) then return 1; else
  i:=1;res:=1;
  while(i<=l) do
  res:=res*L[i];
  i:=i+1;
  end do;
  end if;
  end if;
  return res;
  end proc;
>
```

OpListe3 := **proc**(*L*, *s*)
local *l*, *i*, *res*;
l := nops(*L*);
if *s* = `+` **then**
if *l* = 0 **then return** 0
else *i* := 1; *res* := 0; **while** *i* ≤ *l* **do** *res* := *res* + *L*[*i*]; *i* := *i* + 1 **end do**
end if
end if;
if *s* = `*` **then**
if *l* = 0 **then return** 1
else *i* := 1; *res* := 1; **while** *i* ≤ *l* **do** *res* := *res*∗*L*[*i*]; *i* := *i* + 1 **end do**
end if
end if;
return *res*
end proc

```
> OpListe3([],`+`), OpListe3([],`*`);
```

$$0, 1$$

```
> OpListe3([x,2*x,3*x],`+`),OpListe3([x,2*x,3*x],`*`);
```

$$6\,x, 6\,x^3$$

```
> # Exo 3 : Ecrire une procédure qui prend deux entiers entiers a,
  b en arguments, avec 0<=a<b et qui calcule l'inverse de a modulo
  b lorsque c'est possible et renvoie l'infini sinon.
> inverse:=proc(a,b)
  local i;
  i:=1;
  if (0<=a and a<b) then
  if (igcd(a,b)<>1) then return infinity;
  else
  for i from 1 to b-1 do
  if irem(a*i,b)=1 then return i;end if;
  end do;
  end if;
  else return infinity;
  end if;
  end proc;
```

inverse := **proc**(*a*, *b*)

local *i*;

i := 1;

if 0 ≤ *a* **and** *a* < *b* **then**

if igcd(*a*, *b*) ≠ 1 **then return** ∞

else for *i* **to** *b* − 1 **do if** irem(*a***i*, *b*) = 1 **then return** *i* **end if end do**

end if

else return ∞

end if

end proc

```
> inverse(4,5);
```

4

```
> irem(4*4,5);
```

1

```
> inverse(22,73);
```

10

```
> irem(22*10,73);
```

1

```
> inverse(13,26);
```

∞

```
> #Exo 4 : Ecrire une procédure qui prend une liste de réels et
  renvoie la somme des éléments négatifs de cette liste. Même
  chose en remplaçant la somme par le produit. Vous écrirez les
  procédures sous deux formats l'un avec for et l'autre avec
  while.
> somneg:=proc(L)
  local l,i,s;
  l:=nops(L);
```

```
  s:=0;
  if (L=[]) then return 0; else
  i:=1;
  while(i<=l) do
  if (L[i]<0) then
  s:=s+L[i];
  end if;
  i:=i+1;
  end do;
  end if;
  return s;
  end proc;
```

somneg := **proc**(*L*)
local *l*, *i*, *s*;
l := nops(*L*);
s := 0;
if *L* = [ ] **then return** 0
else *i* := 1; **while** *i* ≤ *l* **do if** *L*[*i*] < 0 **then** *s* := *s* + *L*[*i*] **end if**; *i* := *i* + 1 **end do**
end if;
return *s*
end proc

```
> somneg([-1,0,-2,3,-1,4]);
```

-4

```
> somneg([0,1,2,3]);
```

0

```
> somneg2:=proc(L)
  local l,i,s;
  l:=nops(L);
  s:=0;
  if (L=[]) then return 0; else
  for i from 1 to l do
  if (L[i]<0) then
  s:=s+L[i];
  end if;
  end do;
  end if;
  return s;
  end proc;
```

somneg2 := **proc**(*L*)
local *l*, *i*, *s*;
l := nops(*L*);
s := 0;
if *L* = [ ] **then return** 0

```
    else for i to l do if L[i] < 0 then s := s + L[i] end if end do
    end if;
    return s
end proc
> somneg2([-1,0,-2,3,-1,4]);
> somneg2([0,1,2,3]);
                                    -4
                                    0
> prodneg:=proc(L)
  local l,i,p;
  l:=nops(L);
  p:=1;
  if (L=[]) then return 1; else
  i:=1;
  while(i<=l) do
  if (L[i]<0) then
  p:=p*L[i];
  end if;
  i:=i+1;
  end do;
  end if;
  return p;
  end proc;
prodneg := proc(L)
local l, i, p;
    l := nops(L);
    p := 1;
    if L = [ ] then return 1
    else i := 1; while i ≤ l do if L[i] < 0 then p := p*L[i] end if; i := i + 1 end do
    end if;
    return p
end proc
> prodneg([-1,0,-2,3,1,4]);
> prodneg([0,1,2,3]);
                                    2
                                    1
> prodneg2:=proc(L)
  local l,i,p;
  l:=nops(L);
  p:=1;
  if (L=[]) then return 1; else
  for i from 1 to l do
  if (L[i]<0) then
```

```
  p:=p*L[i];
  end if;
  end do;
  end if;
  return p;
  end proc;
> prodneg2([-1,0,-2,3,-1,4]);
> prodneg2([0,1,2,3]);
```

prodneg2 := **proc**(*L*)

local *l*, *i*, *p*;

l := nops(*L*);

p := 1;

if *L* = [ ] **then return** 1

else for *i* **to** *l* **do if** *L*[*i*] < 0 **then** *p* := *p***L*[*i*] **end if end do**

end if;

return *p*

end proc

-2

1

```
> #Exo 5 : Ecrire une procédure "sfi" ("sfi" pour "square-free
  integers") qui prend un entier strictement positif en argument
  et renvoie true si cet entier n'admet pas de facteurs carrés
  dans sa décomposition en nombres premiers, et false dans le cas
  contraire. Puis écrire une procédure "multsfi" qui prend deux
  entiers strictement positifs, et renvoie le produit des deux à
  condition que le produit obtenu n'est pas de facteurs premiers
  au carré, et renvoie 0 sinon.
```

$(2)^2$ (17)

```
> sfi:=proc(m)
  local i,res;
  i:=2;
  res:=true;
  while (i<m) do
  if (isprime(i)=true and irem(m,i^2)=0) then
  return false;
  else
  i:=i+1;
  end if;
  end do;
  return res;
  end proc;

  multsfi:=proc(m,n)
  local i,res;
```

```
  res:=m*n;
  if (sfi(m)=false or sfi(n)=false or sfi(m*n)=false) then
  return 0;
  else return m*n;
  end if;
  end proc;
```

```
sfi := proc(m)
local i, res;
    i := 2;
    res := true;
    while i < m do
        if isprime( i ) = true and irem( m, i^2 ) = 0 then return false else i := i + 1 end if
    end do;
    return res
end proc
multsfi := proc(m, n)
local i, res;
    res := m*n;
    if sfi( m ) = false or sfi( n ) = false or sfi( m*n ) = false then return 0
    else return m*n
    end if
end proc
```

```
> sfi(26), sfi(12);
```

$$true, false$$

```
> multsfi(26,12);
```

$$0$$

```
> multsfi(7,13);
```

$$91$$

```
> multsfi(14,21);
```

$$0$$

```
> ifactor(14*21);
```

$$(2)\ (3)\ (7)^2$$

```
>
```