Corrigé de la feuille d'exercice n°1

Exercice n°1 : Fonction **Somme** la somme des entiers de 1 à n.

```
> Somme := proc(n::integer)
  local s,i;
  s:=0;
  for i from 1 to n do s := s + i od;
  RETURN(s);
  end;
```

$$\begin{aligned} &\mathit{Somme} := \mathbf{proc}(n{::}\mathit{integer}) \\ &\mathbf{local}\, s,\, i; \\ &\quad s := 0\,;\, \mathbf{for}\, i\, \mathbf{to}\, n\, \mathbf{do}\, s := s + i\, \mathbf{end\ do}\,;\, \mathrm{RETURN}(s) \\ &\mathbf{end\ proc} \end{aligned}$$

```
> Somme(10);
```

55

Deuxième version en utilisant les possibilités de Maple

```
> Somme := proc(n::integer)
  sum(‘i‘,‘i‘=1..n);
  end;
```

$$\mathit{Somme} := \mathbf{proc}(n{::}\mathit{integer})\, \mathrm{sum}(i,\, i = 1..n)\, \mathbf{end\ proc}$$

```
> Somme(8);
```

36

Exercice n°2 : fonction **somprem** qui calcule la somme de n premiers nombres premiers

```
> somprem := proc(n :: posint)
  sum(ithprime(‘i‘),‘i‘=1..n);
  end;
```

$$\mathit{somprem} := \mathbf{proc}(n{::}\mathit{posint})\, \mathrm{sum}(\mathrm{ithprime}(i),\, i = 1..n)\, \mathbf{end\ proc}$$

```
> somprem(4);
```

17

Exercice n°3 : fonction **parfait** qui retourne la liste des nombres parfaits compris entre 1 et n.

```
> est_parfait := proc(n :: posint)
  2*n = convert(numtheory[divisors](n),‘+‘);
  end;
```

$$\mathit{est\_parfait} := \mathbf{proc}(n{::}\mathit{posint})\, 2 * n = \mathrm{convert}(\mathit{numtheory}_{\mathit{divisors}}(n),\, ‘+‘)\, \mathbf{end\ proc}$$

```
> parfait := proc(n :: posint)
  local l, i;
  l := [];
  for i from 1 to n do
     if est_parfait(i) then l := [op(l),i] fi
  od;
  RETURN(l);
  end;
```

parfait := **proc**(*n*::*posint*)
local *l*, *i*;
l := [] ;
for *i* **to** *n* **do if** est_parfait(*i*) **then** *l* := [op(*l*), *i*] **end if end do** ;
RETURN(*l*)
end proc

```
> parfait(1000);
```

[6, 28, 496]

Exercice n°4 : fonction **plouton** qui retourne la liste des n premiers nombres ploutons

```
> plouton := proc(n::integer)
  local l, ndiv, compteur, i;
  # compteur : pour stocker le nombre de ploutons trouvés
  # ndiv : pour stocker le nombre de diviseurs du dernier plouton trouvé
  l := []; compteur := 0; ndiv := 0;
  for i from 1 while compteur < n do
     if numtheory[tau](i) > ndiv
       then
         l := [op(l),i]; ndiv := numtheory[tau](i);
         compteur := compteur + 1;
       fi;
  od;
  RETURN(l);
  end;
```

plouton := **proc**(*n*::*integer*)
local *l*, *ndiv*, *compteur*, *i*;
l := [] ;
compteur := 0 ;
ndiv := 0 ;
for *i* **while** *compteur* < *n* **do**
if *ndiv* < $numtheory_{\tau}(i)$ **then**
l := [op(*l*), *i*] ; *ndiv* := $numtheory_{\tau}(i)$; *compteur* := *compteur* + 1
end if
end do;
RETURN(*l*)
end proc

```
> plouton(20);
```

[1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, 1680, 2520, 5040, 7560]

Exercice n°5 : fonction **amiable**

```
> amiable := proc(n::integer)
  local l, a, b;
  l := {};
  for a from 1 to n do
     b := numtheory[sigma](a)-a; # on calcule le seul b qui puisse
  convenir
     if (b <= n) and (a <> b) and (a = numtheory[sigma](b)-b)
       then l := l union {[a,b]};
     fi;
  od;
  RETURN(l);
  end;
```

amiable := **proc**(*n*::*integer*)
local *l*, *a*, *b*;
l := {} ;
for *a* **to** *n* **do**
b := $numtheory_{\sigma}(a) - a$;
if $b \leqslant n$ **and** $a \neq b$ **and** $a = numtheory_{\sigma}(b) - b$ **then** $l := l$ union $\{[a, b]\}$ **end if**
end do;
RETURN(*l*)
end proc

```
> amiable(3000);
```

{[1210, 1184], [220, 284], [1184, 1210], [284, 220], [2620, 2924], [2924, 2620]}

Exercice n°6 : fonction **palindrome** qui retourne `true` si n est un nombre ègal à son symétrique et `false` sinon.

```
> retourne := proc(n::nonnegint)
  local m, p;  # retourne calcule le "symétrique" d'un nombre
  m := n; p := 0;
  while m <> 0 do
     p := p*10 + irem(m,10,'q');
     m := q;
     od;
  RETURN(p);
  end;
```

retourne := **proc**(*n*::*nonnegint*)
local *m*, *p*;
m := *n* ;
p := 0 ;
while *m* ≠ 0 **do** *p* := 10 ∗ *p* + irem(*m*, 10, '*q*') ; *m* := *q* **end do** ;
RETURN(*p*)
end proc

```
> retourne(123456);
```

654321

```
> palindrome := proc(n :: nonnegint)
     RETURN( evalb(n = retourne(n)));
  end;
```

palindrome := **proc**(*n*::*nonnegint*) RETURN(evalb(*n* = retourne(*n*))) **end proc**

```
> palindrome(12345698789654321);
```

true

```
> palindrome(123654078987654321);
```

false

Exercice n°7 : fonction d'affichage des tables logiques.

```
> printf(‘%6s | %6s | %6s | %6s | %6s | %6s\n‘,‘a  ‘,‘b  ‘,‘a ou
  b‘,‘a et b‘,‘a => b‘,‘a<=>b‘);
  for a in [true,false] do
     for b in [true,false] do
       printf( ‘%6s | %6s | %6s | %6s | %6s | %6s\n‘,
         a, b, a or b, a and b, not a or b , (a and b) or (not
  a and not b));
     od;
  od;
```

```
   a   |    b   | a ou b | a et b | a => b |  a<=>b
  true |   true |   true |   true |   true |   true
  true |  false |   true |  false |  false |  false
 false |   true |   true |  false |   true |  false
 false |  false |  false |  false |   true |   true
```