Correction du TP n°5

Exercice 1

Une façon de faire :

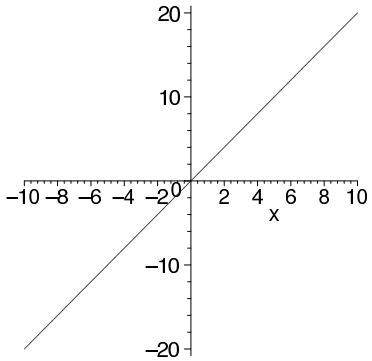
```
> restart;
  grapheder:=proc(f)
  RETURN(plot(diff(f(x),x),x));
  end:
```

Attention à bien donner en entrée une fonction ! (et non pas une expression en x). Par exemple, la commande suivante ne fonctionne pas :

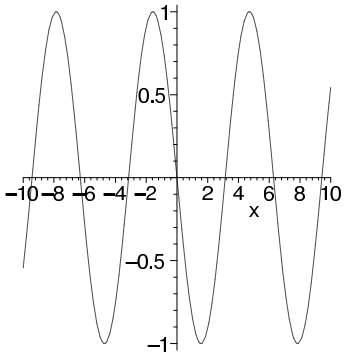
```
> grapheder(x^2);
Warning, unable to evaluate the function to numeric values in the region; see
the plotting command's help page to ensure the calling sequence is correct

Plotting error, empty plot
```

Mais celle-ci fonctionne :

```
> grapheder(x->x^2);
```




```
> grapheder(x->cos(x));
```



Exercice 2

ithprime(k) donne le k-ième nombre premier.

Avec la commande *seq* :

```
> restart;
  seq(ithprime(k),k=1..20);
```

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71

Avec une boucle *for* :

```
> s:=NULL:  # on initialise la séquence s en y mettant la
  séquence vide
  for k from 1 to 20 do
    s:=s,ithprime(k);  #à chaque étape, on met ithprime(k) à droit
  de la séquence
  od:
  s;
```

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71

Exercice 3

La distance entre les points (x,y) et (3,4) est donnée par : $\sqrt{(x-3)^2+(y-4)^2}$.

```
> restart;
  distance:=proc(x,y)
  RETURN(sqrt((x-3)^2+(y-4)^2));
  end:


> distance(7,-5);
```

$$\sqrt{97}$$

Exercice 4

Pour le triangle équilatéral : en entrée, on donne les longueurs a,b, et c des trois cotés.
Le triangle est équilatéral si et seulement si a=b=c.

```
> restart;
  testéquilatéral:=proc(a,b,c)
  if a=b and b=c then RETURN('Le triangle est équilatéral')
  else RETURN('Le triangle n'est pas équilatéral') fi;
  end:
```

Remarque : pour définir une chaine de caractères, on a utilisé le symbole ' (à distinguer du guillemet simple ")

```
> testéquilatéral(2,2,3);
```

Le triangle n'est pas équilatéral

```
> testéquilatéral(8,8,8);
```

Le triangle est équilatéral

Pour le triangle rectangle : en entrée, on donne encore les longueurs a,b, et c des trois cotés. Une façon de faire : le triangle est équilatéral si et seulement s'il vérifie $a^2+b^2=c^2$ ou bien $a^2+c^2=b^2$ ou bien $b^2+c^2=a^2$ (c'est la réciproque du théorème de Pythagore).

```
> testrectangle:=proc(a,b,c)
  local aa,bb,cc;
  aa:=a^2; bb:=b^2; cc:=c^2;
  if aa+bb=cc or aa+cc=bb or bb+cc=aa then RETURN('Le triangle
  est rectangle')
  else RETURN('Le triangle n'est pas rectangle') fi;
  end:
```

```
> testrectangle(3,4,5);
```
Le triangle est rectangle
```
> testrectangle(5,3,4);
```
Le triangle est rectangle
```
> testrectangle(1,2,3);
```
Le triangle n'est pas rectangle

Exercice 5: nombres de Mersenne

On commence par définir une fonction M qui à n associe le nombre de Mersenne M(n).

```
> restart;
  M:=n->2^n-1;
```
$$M := n \to 2^n - 1$$

1) *isprime(k)* dit si le nombre k est un nombre premier (répond *true* ou *false*).
nextprime(k) donne le nombre premier qui suit *k*.
Une façon de calculer le plus petit nombre premier p tel que M(p) ne soit pas premier est la suivante :
- on démarre à p=2 (le premier nombre premier)
- tant que M(p) est premier, on remplace p par le nombre premier qui suit p

La boucle s'arrete la premiere fois que M(p) n'est pas premier. On demande à la fin d'afficher p.

```
> p:=2:
  while isprime(M(p)) do p:=nextprime(p) od:
  p;
```
11

La réponse est p=11.

2) M(11) est le plus petit nombre de Mersenne non premier. Il vaut :

```
> M(11);
```
2047

La commande *ifactor* donne sa factorisation en produit de nombres premiers :

```
> ifactor(M(11));
```
(23) (89)

Exercice 6: suite de Fibonacci

1) On peut définir la suite en utilisant la récursivité. C'est une récurrence d'ordre 2, il faut donc donner deux conditions d'initialisation pour que la suite soit bien définie.

```
> restart;
  u:=proc(n)
  if n=0 then RETURN(0)
  elif n=1 then RETURN(1)
  else RETURN(u(n-1)+u(n-2));
  fi;
  end:
> u(0); u(1); u(2); u(3);
```
0
1
1
2

2)

```
> u(10);
```
55

```
> u(25);
```
75025

3)

```
> # u(50);
```
Le calcul de u(50) est excessivement long...
L'explication est la suivante : quand Maple calcule u(50), il calcule séparément u(49) et u(48). Pour calculer u(49), Maple calcule à nouveau u(48) et u(47)... et ainsi de suite. Comme Maple ne stocke pas les calculs intermédiaires, il passe son temps à recalculer des données qu'il a déjà calculées avant... cela lui prend beaucoup de temps.
Une façon de remédier à cela : modifier la procédure en mettant "option remember" sur la première ligne (remember = souviens-toi !).

```
> restart;
  u:=proc(n) option remember;
  if n=0 then RETURN(0)
  elif n=1 then RETURN(1)
  else RETURN(u(n-1)+u(n-2));
  fi;
  end:
```
Maple crée alors une table interne dans laquelle il stocke les valeurs qu'il a déjà calculées. A chaque appel de la procédure, Maple cherche d'abord si le résultat se trouve dans la table (si elle ne s'y trouve pas, il fait le calcul et le stocke).

```
> u(50);
```
12586269025

Le calcul de u(50) est alors immédiat.