Exercice 1

Tracer la courbe d'équation paramétrique :

$$\Gamma \quad | \quad M(t) \begin{cases} x(t) = \cos(t) \\ y(t) = \sin(t)(1+\cos(t)). \end{cases}$$

Problème d'algorithmique : à propos des nombres premiers

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve. On rédigera les programmes en langage Maple.

Remarques :
- Le problème porte sur les entiers naturels, on utilisera les entiers fournis par le langage de programmation choisi (Maple donc) en supposant qu'ils sont les entiers naturels.
- Étant donnés deux entiers (naturels) a et b ($b > 0$), on supposera connues les procédures :
  - `iquo(a,b)` qui désigne le quotient de la division euclidienne de a par b (parfois appelée « division entière »).
  - `irem(a,b)` (avec $a \geqslant 0$ et $b > 0$) qui donne le reste de la division euclidienne de a par b.
- On supposera également connue une fonction `sqrt(a)` qui donne la partie entière de la racine carrée de a (notée $\lfloor\sqrt{a}\rfloor$).

1. Écrire la fonction `estPremier`(n) qui prend un entier n ($n \geq 2$) en argument, et qui renvoie 1 si n est premier et 0 sinon. On appliquera directement la définition des nombres premiers donnée ci-dessus.

   On veut maintenant calculer tous les nombres premiers inférieurs à un entier donné n. On va procéder selon trois méthodes différentes.

2. Écrire la fonction `petitsPremiers`(n) qui prend en argument un entier $n \geq 2$ et qui :
   1. Range (dans l'ordre croissant) les nombres premiers inférieurs ou égaux à n dans un tableau global (ou prédéclaré) `premier`, que l'on supposera assez grand.
   2. Renvoie le nombre d'entiers rangés dans `premier`.

   Par exemple pour $n = 17$, votre fonction devra renvoyer 7 et remplir le début de `premier` ainsi :

2	3	5	7	11	13	17	...

   La fonction `petitsPremiers` sera la plus simple possible, des techniques plus efficaces sont l'objet des deux questions suivantes.

   Il est assez facile de voir que, pour tester si un entier *impair* m ($3 \leq m$) est premier, il suffit de vérifier que m n'est divisible par aucun nombre *premier* compris entre 3 et $\lfloor\sqrt{m}\rfloor$.

3. Écrire une nouvelle fonction `petitsPremiers2`(n), qui agit comme `petitsPremiers`(n) mais est plus efficace. Cette nouvelle fonction appliquera la remarque ci-dessus, les nombres premiers entre 3 et $\lfloor\sqrt{m}\rfloor$ étant lus dans le tableau `premier` en cours de remplissage.

   Au IIIe siècle avant Jesus-Christ, Ératostène, mathématicien, astronome et philosophe invente une méthode efficace pour énumérer tous les nombres premiers inférieurs à un entier donné n. Cette méthode est connue sous le nom de crible d'Ératosthène, que nous allons décrire en tenant compte des moyens disponibles à l'époque.
   1. Sur le sable, avec un bâton, dessiner un tableau de $(n-1)$ cases, la première case représente 2, la suivante 3 etc. Ramasser un caillou, le poser à gauche du tableau.
   2. Avancer le caillou jusqu'à trouver une case non-rayée. Cette case représente l'entier i.
   3. Ramasser un second caillou et le faire avancer par sauts de i cases. Pour chacune des cases sur lesquelles le second caillou se pose :
      (a) Si la case est déjà rayée, ne rien faire.
      (b) Sinon, rayer la case avec le bâton.
   4. Poser le second caillou dès que l'on sort du tableau. Si l'étape (b) n'a pas été effectuée, c'est fini. Sinon recommencer en 2.

Les cases *non-rayées* représentent maintenant les nombres premiers. À titre d'exemple, voici l'état du tableau au début de l'étape 4 et pour un tableau de 10 cases, le premier caillou étant « • » :

•		X		X		X		X		$(i = 2)$
	•	X		X		X	X	X		$(i = 3)$
		X	•	X		X	X	X		$(i = 5)$

Les nombres premiers trouvés sont donc $2, 3, 5, 7, 11$.

4. Écrire la fonction `petitsPremiers3`(n), qui agit comme `petitsPremiers`(n) mais utilise le crible d'Érathostène. On s'attachera à respecter l'esprit de cette technique, et en particulier à éviter multiplications, divisions et racines carrées, difficiles à effectuer pour les mathématiciens grecs.

   On rappelle que tout entier n ($n > 2$) peut s'écrire de manière unique comme le produit $n = p_1 p_2 \dots p_k$, où $p_1 \leq p_2 \leq \dots \leq p_k$ sont des nombres premiers. On appelle cette écriture la décomposition en facteurs premiers de n. Cette définition suggère un algorithme simple pour décomposer n en facteurs premiers. Soit $\pi_1, \pi_2, \dots$ la suite des nombres premiers en ordre croissant.

   1. Poser $m = n$, poser $i = 1$.
   2. Si $m = 1$, alors terminer.
   3. Si π_i divise m, alors π_i entre dans la décomposition de n. Poser $m = m/\pi_i$ et aller en 3.
   4. Sinon, poser $i = i + 1$ et aller en 2.

   En résumé, l'algorithme, dit *essai des divisions* revient à diviser n par les nombres premiers.

5. Écrire la fonction `factoriser`(n) qui prend en argument un entier n ($n \geq 2$), et qui range (dans l'ordre croissant au sens large) les facteurs premiers de n dans un tableau global `facteur`, supposé grand. La fonction `factoriser` renvoie le nombre d'entiers rangés dans `facteur`. Par exemple pour $n = 90$, votre fonction devra renvoyer 4 et remplir le début de `facteur` ainsi :

2	3	3	5	...

   Il n'est pas trop difficile de voir que, dans l'algorithme d'essai des divisions, on peut remplacer la suite des nombres premiers par une suite plus simple à calculer, du moment que celle-ci commence par 2, soit croissante, et contienne les nombres premiers — par exemple, $q_1 = 2$, $q_{j+1} = 2j + 1$ ($j \geq 1$). En effet, un invariant de l'algorithme est que, à l'étape 2, m n'est divisible par aucun des entiers $q_1, \dots, q_{i-1}$, et donc par aucun nombre premier strictement inférieur à q_i. Par conséquent, si q_i divise m (étape 3), alors q_i est premier.

   Par ailleurs, on peut limiter la taille des facteurs essayés. La suite des q_i est strictement croissante, tandis que m décroit. On finira donc par avoir $q_i^2 > m$. Dès lors, m, s'il n'est pas égal à 1, est premier, puisque non-divisible par $q_1, q_2, \dots, q_{i-1}$, séquence qui contient tous les nombres premiers inférieurs ou égaux à $\lfloor \sqrt{m} \rfloor$. Dans ces conditions m est le dernier facteur de la décomposition de n.

6. Écrire une nouvelle fonction `factoriser2`(n) qui agit comme `factoriser`, mais ne calcule pas de table de nombres premiers et exploite la seconde remarque ci-dessus.