TP Informatique – le langage TypeScript

Exercice 1 : les fonctions : les nombres particuliers

➔ Tous les paramètres et valeurs de retour doivent être typés

- ✓ Ecrire la fonction « somDivPropre » qui calcule la somme des diviseurs propres d'un nombre entier n(les diviseurs sauf lui-même) : *attention le résultat n'est défini que pour n >=1*
- ✓ Ecrire la fonction « premier » qui indique si un nombre n est premier -> il n'a que deux diviseurs 1 et lui-même
- ✓ Vérifier que dans l'intervalle [1,1000] il y a 168 nombres premiers
- ✓ Ecrire la fonction « parfait » qui indique si un nombre n est parfait -> n est égal à la somme de tous ses diviseurs propres
- ✓ Vérifier que dans l'intervalle [2,1000] seuls 6, 28 et 496 sont parfaits
- ✓ Ecrire la fonction « chanceux » qui indique si un nombre n est chanceux d'Euler ? si pour tout nombre a tel que 0 <= a < (n-1) $n + a + a^2$ est premier.
  Attention le résultat n'est défini que n >= 2
- ✓ Vérifier que dans l'intervalle [2,1000] seul 2,3,5,7,11,17,41 sont chanceux
- ✓ Ecrire la fonction « amiable » qui indique si un nombre n est amiable c'est-à-dire si on peut lui trouver un ami différent de lui-même ; la fonction retournera cet ami s'il existe ; attention n'est défini que pour n > 0
  2 nombres n1 et n2 sont amiables => la somme des diviseurs propres de n1 est égal à n2 et la somme des diviseurs propres de n2 est égal à n
- ✓ Vérifier que dans l'intervalle [1,1000] seuls 220 et 284 sont amiables
- ✓ Ecrire une fonction qui calcule le nb et la somme des diviseurs d'un nombre (tous les diviseurs pas uniquement les diviseurs propres) ; attention défini que pour n > 0
- ✓ Ecrire la fonction « sublime » qui si un nombre n est sublime ? le nbre de diviseurs de n et la somme des diviseurs de n sont 2 nombres parfaits.
- ✓ Vérifier que dans l'intervalle [2,1000] seul 12 est sublime

Exercice 2 : les tableaux : les pays

- ✓ Déclarer ces 3 tableaux :
  let liste1 : string[] = ['France', 'Italie', 'Espagne', 'Portugal'];
  let liste2 : string[] = ['Suède', 'Danemark', 'Pays-Bas' ,'Belgique'];
  let tPays : string[] ;
- ✓ Initialiser tPays avec la liste1 puis afficher tPays et le nb d'éléments de tPays
- ✓ Ajouter la « Grèce » à la fin de tPays et afficher tPays et le nb d'éléments
- ✓ Supprimer le 1er pays et afficher tPays et le nb d'éléments
- ✓ Supprimer le dernier pays et afficher tPays et le nb d'éléments
- ✓ Ajouter la « Grèce » et la « France » en début de tableau ; afficher tPays et le nb d'éléments
- ✓ Ajouter les pays de liste2 ; afficher tPays et le nb d'éléments
- ✓ Recherche un pays dans tPays ; afficher un message adapté :
- ✓ Recherche et supprimer l' « Espagne » ; afficher tPays et le nb d'éléments
- ✓ Construire une chaine qui contient tous les pays séparés par des ;
- ✓ Trier la liste tPays dans l'ordre alphabétique
- ✓ Trier la liste tPays dans l'ordre alpha inverse en utilisant une closure
- ✓ Trier sur la longueur des noms des Pays en utilisant une closure

- ✓ Créer un nouveau tableau contenant les longueurs de chacun des pays en utilisant une closure
- ✓ Construire un nouveau tableau contenant uniquement les pays commençant par P ou F en utilisant une closure
- ✓ Calculer la somme des longueurs des chaines de la liste en utilisant une closure
- ✓ Mettre tous les pays en majuscules

Exercice 3 : les classes : liste de choses à faire

- ✓ Créer une classe correspondant à une chose à faire caractérisée par un texte et un indicateur permettant de savoir si la chose est à faire ou si elle a déjà été faite.
- ✓ Le constructeur doit permettre d'initialiser le texte et l'indicateur ; si celui-ci n'est pas fourni il sera initialiser à faux par défaut
- ✓ La classe doit comporter un getter pour le texte et un setter pour le texte et l'indicateur
- ✓ La classe doit comporter une méthode pour afficher la chose à faire sous la forme

  ==> chose (à faire) ou ==> chose (faite)

  →utiliser l'interpolation de chaines et l'opérateur ternaire
- ✓ Créer une classe permettant de gérer une liste de choses à faire

  Le constructeur doit permettre d'initialiser la liste à partir de plusieurs chaines de caractères (les textes des choses à faire).

  Il doit y avoir des méthodes :

  - ajouter une chose à partir de son texte
  - afficher la liste des choses à faire
  - supprimer une chose à faire à partir de son texte
  - faire une chose
- ✓ Exemple de pgm de test :

```
let maliste = new Liste("menage","courses");
maliste.afficher();
maliste.ajouterChose("velo");
maliste.afficher();
maliste.faire("menage");
maliste.afficher();
maliste.supprimerChose("courses");
maliste.afficher();
```

Exercice 4 : les classes : jeu de cartes

L'objectif est de pouvoir simuler un jeu de cartes.

- ✓ Créer une énumération contenant les 4 couleurs (cœur, carre, pique, trefle)
- ✓ Créer une classe « Carte » avec un numéro et une couleur ; pour simplifier les traitements, les figures seront manipulés par des valeurs (par exemple pour un jeu de 52 cartes : Valet -> 11, Dame -> 12, …)

- ✓ Cette carte doit comporter un constructeur et une méthode pour la carte sous la forme d'une chaine pour l'affichage ; par exemple : « 2 de cœur », « 4 de trèfle », « As de pique », « Dame de carreau », …
- ✓ Créer une classe « Jeu » permettant de gérer un jeu de cartes (52 cartes par exemple) avec un constructeur permettant d'initialiser correctement le jeu en fonction du nombre de cartes (32, 52, …). Une méthode permettra aussi de tirer une carte au hasard.
- ✓ Créer une classe « Main » permettant de gérer une main composée d'un nombre de cartes tirées au hasard dans un jeu de cartes
- ✓ Compléter cette classe avec des méthodes permettant de savoir si :
  - o La main est une couleur ? [toutes les cartes sont de la même couleur]
  - o La main est une quinte ? [toutes les cartes forment une suite]
  - o La main est une quinte flush ? [quinte et couleur]
  - o La main est un carré ?
  - o La main est une paire ?
  - o La main est un brelan ?
  - o La main est un full ? [brelan + paire]

Exemple :

```
la main est composée de 5 cartes :
    ==> As de coeur  - 9 de coeur  - 9 de carreau  - 10 de pique  - 10 de trefle
    ==> ce n'est pas une couleur
    ==> ce n'est pas une quinte
    ==> ce n'est pas un carré
    ==> ce n'est pas un brelan
    ==> c'est une paire
```

Exercice 5 : les interfaces et les modules : formes géométriques

- ✓ Créer le module Couleur.ts contenant la définition d'une énumération :

```
enum Couleur { rouge, vert, bleu, jaune, gris, orange }
export { Couleur }
```

- ✓ Créer le module Forme.ts contenant la classe « Forme » ayant ces propriétés :

Nb de droite : la forme est constituée de X segments de droites
Nb de courbes : la forme est constituée de X segments courbes
Figure fermée : la forme est fermée ou ouverte (booléen)

et ces méthodes :

Le constructeur init()
isQuadrilataire()
isCercle()
isTriangle()

surface() -> calcule la surface, 0 si pas pertinent
perimetre() -> calcule le périmètre, 0 si pas pertinent
getForme() -> affiche toutes les propriétés de la forme

remarque 1 : une forme est d'une seule couleur
remarque 2 : les formes seront de couleur verte

Tester en créant des formes comme par exemple :

```
==================================================
couleur vert nbDroites : 3 nbCournes : 0
la surface est fermée
surface 0  Perimètre 0
==================================================
```

- ✓ Créer le module « Formes.ts » contenant les classes « Carre », « Rectangle », « Cercle » en redéfinissant les méthodes « surface » et « périmètre »
  - Un carré est caractérisé par la longueur d'un de ses cotés (propriété cote)
  - Un rectangle est caractérisé par sa longueur et sa largeur
  - Un cercle est caractérisé pas son rayon

Remarque : les carrés seront orange, les rectangles gris, les cercles rouges

Tester :

```
Carre de cote 3
==================================================
couleur orange nbDroites : 4 nbCournes : 0
la surface est fermée
surface 9  Perimètre 12
==================================================


Rectangle de largeur 2 et longueur 3
==================================================
couleur gris nbDroites : 4 nbCournes : 0
la surface est fermée
surface 6  Perimètre 10
==================================================


Cercle de rayon 4
==================================================
couleur rouge nbDroites : 0 nbCournes : 1
la surface est fermée
surface 50.26548245743669  Perimètre 25.132741228718345
==================================================
```

- ✓ Créer le module « interfaceSommets.ts » contenant la définition de l'interface Sommets défini par :
  - Une propriété « lesSommets » qui contiendra un tableau de chaines de caractères
  - Une méthode « listeSommets() » qui renvoie une chaine de caractères contenant la liste des noms des sommets
- ✓ Modifier les classes « Carre », « Rectangle » pour qu'elles implémentent l'interface précédente.
- ✓ Tester :

```
======== figure A ===============

        ==================================================
        couleur vert nbDroites : 3 nbCournes : 0
        la surface est fermée
        surface 0  Perimètre 0
        ==================================================


======== figure B ===============

        ==================================================
        couleur vert nbDroites : 0 nbCournes : 1
        la surface est fermée
        surface 0  Perimètre 0
        ==================================================


======== figure ABCD ===============

        Carre de cote 3
        ==================================================
        couleur orange nbDroites : 4 nbCournes : 0
        la surface est fermée
        surface 9  Perimètre 12
        ==================================================
        Les sommets A,B,C,D

======== figure EFGH ===============

        Carre de cote 4
        ==================================================
        couleur orange nbDroites : 4 nbCournes : 0
        la surface est fermée
        surface 16  Perimètre 16
        ==================================================
        Les sommets E,F,G,H
```

```
======== figure MNOP ===============

        Rectangle de largeur 2 et longueur 3
        ==================================================
        couleur gris nbDroites : 4 nbCournes : 0
        la surface est fermée
        surface 6  Perimètre 10
        ==================================================
        Les sommets M,N,O,P

======== figure QRST ===============

        Rectangle de largeur 2 et longueur 3
        ==================================================
        couleur gris nbDroites : 4 nbCournes : 0
        la surface est fermée
        surface 6  Perimètre 10
        ==================================================
        Les sommets Q,R,S,T

======== figure C ===============

        Cercle de rayon 4
        ==================================================
        couleur rouge nbDroites : 0 nbCournes : 1
        la surface est fermée
        surface 50.26548245743669  Perimètre 25.132741228718345
        ==================================================
```

- ✓ Stocker toutes les formes dans un tableau puis compléter le pgm pour calculer la surface et le nb de carrés et aussi la surface et le nb de rectangles

- En utilisant des boucles for (rappel instanceof permet de savoir de quelle classe l'objet est une instance)
- En utilisant les fonctions de type map/reduce (dans ce cas on doit pouvoir par exemple calculer le nb de carrés en une seule instruction, idem pour la surface)