TP1 TypeScript L2DSI3 Gr1

I. Setup nouveau projet

1. Créez un nouveau dossier TP1
2. On doit créer un fichier nommé **package.json**. Pour ne pas le créer et saisir les données manuellement, il est recommandé d'utiliser le gestionnaire de paquets **npm**. Dans le terminal, introduisez la commande **npm init.** L'utilitaire va vous demander de fournir des informations sur le nouveau projet afin de remplir certains champs. Il faut tout simplement appuyer sur la touche **Enter** pour associer les valeurs par défaut.
3. On doit créer aussi un fichier de configuration pour TypeScript, **tsconfig.json.** La commande à exécuter dans le terminal est **tsc --init.**
4. A l'aide de l'interface graphique de Visual Studio Code, créez un nouveau fichier **index.ts**
5. Dans le Terminal, exécutez la commande **tsc index.ts**. Cela générera le **fichier .js** correspondant. Ce fichier ne doit pas être modifié, il sera utilisé seulement pour la compilation, car en reste on va travailler seulement avec les fichiers TypeScript
6. Pour exécuter le programme, saisissez la commande **node index.js**.

Pour certaines versions de système d'exploitation, il est très probable que vous rencontrez l'erreur suivante lorsque vous essayez de démarrer le programme : **tsc: cannot be loaded because running scripts is disabled on this system**. Pour résoudre ce problème, vous devez suivre les étapes suivantes :

Démarrez Windows PowerShell en tant qu'administrateur

1. Saisissez la commande suivante : **Set-ExecutionPolicy -ExecutionPolicy RemoteSigned**
2. Redémarrez **VSCode** et exécutez de nouveau le programme

II. Modules

Un module contient un ensemble de fonctions qui peuvent être incluses dans une application afin d'utiliser des fonctionnalités plus complexes.

Pour installer des modules, on utilise **npm install**. Le drapeau –save les ajoutera automatiquement au fichier package**.json** (pour une installation globale, on utilise -g):
Exemple:

npm install lib_name –save

Pour pouvoir utiliser tous les types de NodeJS, on devra exécuter dans le Terminal la commande npm install @types/node –save-dev.

Pour inclure un module installé dans l'application, avant de commencer a utiliser ce module il faut l'importer de la manière suivante :

```
import * as module_name from 'module_name';
```

Exemple : module fs

- **Lire le contenu d'un fichier**

Pour lire le contenu d'un fichier, on utilise le module fs, de la façon suivante:

```
import * as fs from 'fs';

let file_content: string;
try {
   file_content = fs.readFileSync('./test.txt', 'ascii');
   console.log(file_content);
} catch(error) {
   console.log(error);
}
```

- **Ecrire dans un fichier**

```
import * as fs from 'fs';

let to_write: string = "sample text";
try {
   fs.writeFileSync('./test.txt', to_write);
} catch(error) {
   console.log(error);
}
```

III. Travail à faire

1. Faites un nouveau projet en Node JS qui s'appelle **TP1_Votre Nom_Prenom**.
   a. Créez un nouveau directoire Tp1_ **Votre Nom_Prenom**.
   b. Dans le nouveau directoire, créez un fichier index.ts
   c. Entrez dans le directoire et utilisez la commande **npm init -y**, ensuite la commande **tsc --init**. Qu'est-ce qui se passe ?
2. Ecrivez un programme qui affiche sur l'écran votre nom et prénom, en utilisant l'opération de concaténation.
3. Installez le module **chalk**, en utilisant **npm**. Le module doit être ajouté automatiquement en **package.json.**
4. Affichez votre nom et prénom avec des couleurs différentes, en utilisant le module chalk.
5. Faites un nouveau projet qui s'appelle **systeminfo**. Inspectez la documentation du module os et utilisez-le pour afficher les données suivantes. N'oubliez pas d'installer les types de node avec **npm install --save-dev @types/node.):** voir **https://nodejs.org/api/os.html**
   a. Le répertoire de base pour l'utilisateur actuel (home directory)
   b. Le nom d'hôte du système d'exploitation
   c. La mémoire disponible en KB
   d. La quantité totale de mémoire système en GB
6. Déclarez une variable pour la mémoire en kB, une pour la mémoire en MB (nombres entièrs, 1KB = 1024B, 1MB = 1024KB), une pour le répertoire de base et une avec l'endianité du processeur ***. Ajoutez ces variables dans un string, avec des espaces entre eux. Affichez le string sur l'écran.
7. Ecrivez dans le fichier os.info toutes les informations de l'exercice précédent. (**fs.writeFileSync**)
8. Faites un projet qui s'appelle **boucles.** En utilisant la boucle **for/ while,** affichez :
   a. Tous les nombres pairs dans l'intervalle 1000-1020 (utilisez boucle **for of**)
   b. Les mêmes nombres en ordre inverse
   c. Tous les nombres divisibles par 3 (utilisez boucle **while**)
   d. Tous les nombres divisibles par 5 et 7
   e. Tous les nombres divisibles par 11 ou 6 (utilisez boucle **for of**)
9. Stockez un nombre **N** quelconque dans une variable. Utilisez la boucle **while** pour compter seulement ses diviseurs pairs. Affichez sur l'écran :
   a. "Le nombre N à y diviseurs", où y le numéro de ses diviseurs
   b. Si le nombre est premier.

Partie2 :

1. Créez un programme et exécutez-le avec **5 paramètres de type number**. Sauvegardez la liste des paramètres dans une variable. Affichez :
   a. La liste des paramètres avec **console.log**
   b. Le nombre de paramètres
   c. Les paramètres avec un **for** (à votre choix)
2. Faites un programme qui contient une fonction **power** qui reçoit deux nombres comme paramètres, qui calcule la puissance et qui
   a. Affiche le résultat
   b. Retourne le résultat
   c. Stocke le résultat dans une variable et l'affiche
3. A l'aide des interfaces, créez un nouveau type de données nommé Employe, qui contienne les champs suivants : **nom, prénom, département, expérience** (numéro)
   a. Créez un objet de type Employe
   b. Affichez chaque attribut de l'objet
   c. Créez un tableau avec 3 objets de type Employe
   d. Affichez l'étudiant avec la plus grande expérience
   e. Affichez la personne avec le prénom le plus longue
   f. Affichez tous les objets en ordre alphabétique selon le nom de famille
4. Créez une classe Employe ayant les mêmes propriétés définies pour l'interface de l'exercice précèdent
   a. Créez une sous-classe Manager, dérivée de la classe Employe, qui contient un champ supplémentaire **noSubordonnés** (number)
   b. Créez un objet de type Manager
   c. Créez une fonction qui reçoit comme paramètre un objet de type Manager et qui vérifie si objet.**noSubordonnés** est supérieur à 15 ou non

Annexe

1. Opérateurs

Operateur	Description	Example
+	Plus	x + y
-	Moins	x - y
*	Multiplication	x * y
/	Division	x / y
%	Reste de la division	x % y
==	Valeur d'égalité	x == y
===	Égalité de la valeur et du type	x === y
!=	Valeur différente	x != y
!==	Différence de la valeur ou du type	x !== y
<	Moins grand	x < y
<=	Moins grand ou égal	x <= y
>	Plus grand	x > y
>=	Plus grand ou égal	x >= y
!	Pas	!x
&&	Et	x && y
||	Ou	x || y
&	AND	x & y
|	OR	x | y
~	NOT	x ~ y
^	XOR	x ^ y
»	Right Shift	x » y
«	Left Shift	x « y

2. Fonctions prédéfinies pour les tableaux

- **list.length** - Retourne le numéro d'éléments de la liste
- **list.indexOf (search, index)** - Retourne le premier index auquel l'élément donné peut être trouvé dans la liste , ou -1 s'il n'est pas présent dans la liste.
- **list.lastIndexOf (search, index)** - Retourne le dernier index auquel l'élément donné peut être trouvé dans la liste, ou -1 s'il n'est pas présent dans la liste
- **list.push (element, ...)** - Ajoute un nouvel élément à la fin de la liste
- **list. unshift (element, ...)** - Ajoute un nouvel élément au début de la liste
- **list.pop ()** - supprimer et retourner la valeur du dernier élément de la liste
- **list.splice (start, deleteCount, pushElement, ...)** - changer le contenu de la liste, en supprimant ou en remplaçant certains éléments
- **list.map (functionToRun) -** créer un nouveau tableau contenant des éléments générés par la fonction appelée
- **list.findIndex (functionToSearch)** - retourne l'indice du premier élément qui satisfait la fonction appelée
- **list.filter (functionToFilter)** - nouveau tableau avec les éléments qui répondent à la condition incluse dans la fonction-paramètre

3. Instructions conditionnelles

En TypeScript, nous avons les instructions conditionnelles suivantes:

- **if** - pour spécifier un bloc de code à exécuter, si la condition est vraie
- **else** - pour spécifier un bloc de code à exécuter, si la même condition qua est fausse
- **else if** - pour spécifier une nouvelle condition à tester, si la première condition est fausse
- **switch** - pour spécifier plusieurs blocs de code alternatives à exécuter

- **If**

```
let x: number = 1;
if (x === 1) {
  console.log(x) ; //l'instruction s'exécute
}
```

- **If Else**

```
let x: number = 5;

if(x < 3) {
  console.log(x + 3);
}
else {
  console.log(x + 1); //seulement cette instruction s'exécute
}
```

- **If Else if**

```
let x: number = 2;

if(x < 3) {
  console.log(x + 3); //seulement cette instruction s'exécute
}
else if (x > 4){
  console.log(x + 1);
}
```

- **Switch**

```
let x: number = 2;

switch(x) {
  case 1:
    console.log(x + 1);
    break;
  case 2:
    console.log(x + 2); //seulement cette instruction s'exécute
    break;
  case 3:
    console.log(x + 3);
    break;
  default:
    console.log(x); //instruction qui s'execute si x n'est pas egal a 1, 2 ou 3
  }
```

4. Boucles

Les boucles peuvent exécuter un bloc de code plusieurs fois.

- **For**

La boucle for parcourt un bloc de code un certain nombre de fois.

```
let x: number= 10;
let sum: number = 0;

for(let i = 1; i <= x; i++) {
  sum += i;
}

console.log(sum); //55
```

- **For in**

L'instruction *for in* itère sur les propriétés énumérables d'un objet. Pour chaque propriété distincte, les instructions peuvent être exécutées.

```
let array: number[] = [5, 7, 2, 9, 3]

for(let i in array) {
  console.log("Value " + array[i] + " at index " + i);
```

```
}

/* Output:
Value 5 at index 0
Value 7 at index 1
Value 2 at index 2
...
*/
```

- **For of**

L'instruction *for of* permet de créer une boucle qui parcourt un objet itérable et qui permet d'exécuter une ou plusieurs instructions pour la valeur de chaque propriété.

```
let array: number[] = [5, 7, 2, 9, 3]

for(let i of array) {
  console.log("Value " + i);
}

/* Output:
Value 5
Value 7
Value 2
...
*/
```

- **While**

La boucle while parcourt un bloc de code tant qu'une condition spécifiée est vraie.

```
let x:number = 1;
let sum: number = 0;

while(x <= 10) {
  sum += x;
  x++;
}

console.log(sum); //55
```

*** endianness : l'ordre dans lequel ces octets sont placés. Il existe deux conventions opposées l'orientation BE qui démarre avec les octets de poids forts, et l'orientation inverse BL