

## 1. Base du langage

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

- c. Créer un objet `Personne` à la volée avec un `let` qui contient un attribut `nom`, un prénom, une date de naissance et une fonction anonyme `nomComplet` qui retourne une string avec son nom complet
- d. Transformer votre objet à la volée avec une vraie classe.
- e. Créer une interface `Surface2D` avec de deux fonctions `arrow` `aire` et `périmètres`. Créer une classe `rectangle` qui implémente cette interface logiquement. Créer logiquement une classe `Carré` qui étend `rectangle`.
- f. Créer une interface `maStructureDeDonnées` avec un `number` `x`, `y` et un rayon optionnel. Créer deux variable `let` qui instancie cette interface.

g.

[REDACTED]

h.

[REDACTED]

- i. Exécuter le code suivant et analyser le :

```
var myUnionVar: string | number | boolean;
console.log('myUnionVar before setting a value = ' + myUnionVar);
myUnionVar = 5;
console.log('typeof myUnionVar = ' + typeof myUnionVar);
```

Créer une variable de type `Pipe` (ou `union`) qui peut être un `number` ou un `array`. Exécuter un `if` pour tester le vrai type de cette variable après lui avoir assigner en premier un `number` puis un `tableau`. Créer une variable de type `string` qui sera initialisé avec votre variable de type `Pipe` et `typeof`. Avec `switch`, afficher si la variable de `Pipe` est un `nombre` ou un `array`.

- j. Créer une fonction qui additionne deux nombres. Créer une fonction anonyme qui additionne deux nombres. Créer une fonction `arrow` qui additionne deux nombres.
- k. Créer une interface `Pet` qui a un `nom`, un `age` et un `poids`. Créer un `tableau` de `Pet`. Créer une fonction anonyme qui compare deux `Pet` du plus jeune au plus âgé (idem que `Comparable`) et appelle en paramètre de la méthode `sort` sur votre `tableau`.
- l. Analyser le code suivant:

```
var myUnionVar: string | number | boolean;
console.log('myUnionVar before setting a value = ' + myUnionVar);
myUnionVar = 5;
console.log('typeof myUnionVar = ' + typeof myUnionVar);
```

Créer une classe `displayUserId` avec une fonction arrow qui permet « this » de fonctionner normalement.

m. Analyser le code suivant :

```
(function () {
    // Let's get started!
    console.log("Let's get started!");
    // "this" works differently in different circumstances.
    // In this class "this" works in a way you might now expect.
    class employee {
        userId: string;
        displayUserId() {
            setTimeout(function () {
                console.log("this.UserId is: ${this.userId}");
            }, 1000);
        }
    }
    // Creating an object of type employee.
    var myEmployee = new employee();
    myEmployee.userId = 'abc123';
    // Calling the displayUserId method.
    // Notice "this.userId" returns "undefined".
    myEmployee.displayUserId();
})();
```

Créer une fonction simple (`console.log`) que vous pouvez utiliser comme un callback. Créer une fonction qui accepte cette fonction comme callback et qui l'appelle.

## 2. Tableaux

Compléter le fichier `exo2.ts` par les questions qui suivent.

Vous devez afficher proprement vos résultats dans une page web `index.html` qui ne contient qu'une div d'id `app`. Vous n'avez pas le droit d'utiliser un serveur.

Exemple de code pour manipuler le doc en TS :

```
const appDiv: HTMLElement = document.getElementById('app');
appDiv.innerHTML = `<h1>Exercice TypeScript</h1>`;
```

### Questions

Faire la somme des âges de tout les "users", afficher la valeur de `sumAge`.

Récupérer les noms des "users" dans le tableau `usersName`, afficher sa valeur.

Ajouter l'utilisateur "franckMonod" à la liste "users".

Récupérer les "users" avec le status "alternant" dans le tableau `usersAlternant` grâce à une boucle "for", afficher sa valeur.

Changer le status de l'utilisateur "Thomas" en "cofondateur".

Récupérer les "users" avec le status "cofondateur", afficher ces utilisateurs en 1 seule string.

Trouver le nom de l'entreprise de l'utilisateur ayant un id = 2.

Trouver le nom de la localisation de l'utilisateur ayant pour indice 2 dans le tableau "users".

Changer la localisation des entreprises ayant pour nom "PRISMO" en "Cran-Gevrier".

Trouver le nom de l'entreprise de l'utilisateur ayant un id = 2.

Trouver le nom de la localisation de l'utilisateur ayant pour indice 2 dans le tableau "users".

Changer la localisation des entreprises ayant pour nom "PRISMO" en "Cran-Gevrier".

L'utilisatrice Leïla n'a pas de compagnie associée. Elle souhaite ajouter une entreprise sur son profil. Pour ce faire, elle doit :

- Trouver l'entreprise "PRISMO", dans la liste des "companies";
  - Associer les valeurs de "PRISMO" à la propriété "company" de son compte
- Afficher son compte

Créer une interface correspondant à l'objet "user"

Créer une interface correspondant à l'objet "company"

3.



Considérons les deux classes **Personne** et **Adresse**. Les attributs de la classe **Adresse** sont :

- **rue** : un attribut privé de type chaîne de caractères.
- **ville** : un attribut privé de type chaîne de caractères.
- **codePostal** : un attribut privé de type chaîne de caractères.

Les attributs de la classe **Personne** sont :

- **nom** : un attribut privé de type chaîne de caractères.
- **sexe** : un attribut privé de type chaîne de caractères (cet attribut aura comme valeur soit 'M' soit 'F').
- **adresses** : un attribut privé de type tableau d'objet de la classe **Adresse**.

1. Créez les deux classes **Adresse** et **Personne** dans deux fichiers séparés. N'oubliez pas de définir les getters/setters et les constructeurs.
2. Créez une troisième classe **ListePersonnes** ayant un seul attribut **personnes** : un tableau d'objets **Personne**. Définissez les getters/setters et le constructeur de cette classe.
3. Écrivez la méthode **findByNom(s: string)** qui permet de chercher dans le tableau **personnes** si l'attribut **nom** d'un est égal à la valeur du paramètre **s**. Si c'est le cas, elle retourne le premier objet correspondant, sinon null.
4. Écrivez la méthode **findByCodePostal(cp: string)** qui permet de vérifier dans le tableau **personnes** si un objet possède au moins une adresse dont le code postal égal au paramètre **cp**. Si c'est le cas, elle retourne **true**, sinon **false**.
5. Écrivez la méthode **countPersonneVille(ville: string)** qui permet de calculer le nombre d'objets dans le tableau **personnes** ayant une adresse dans la ville passée en paramètre.
6. Écrivez la méthode **editPersonneNom(oldNom: string, newNom: string)** qui remplace les noms de personnes ayant un nom égal à la valeur **oldNom** par **newNom**
7. Écrivez la méthode **editPersonneVille(nom: string, newVille: string)** qui remplace les villes de personnes ayant un nom égal à la valeur du paramètre **nom** par **newVille**

#### 4. Promises

- a. Ecrire la fonction delay avec une promise (paramètre en ms).  
Ecrire la fonction timeout avec une promise (paramètre en ms et une promise).  
Tester les deux fonctions avec `timeout( 8000 , delay(5000))` :  

```
timeout(3000, delay(5000))  
  .then(function () {  
    console.log('5 seconds have passed!')  
  })  
  .catch(function (reason) {  
    console.error('Error or timeout', reason);});
```
- b. Ecrire une fonction testNum qui prend en param un nombre et qui retourne une Promise qui test si le param est plus grand que 10
- c. Créer et utiliser des Promises pour trier un tableau de string et l'afficher en majuscules. Si le tab ne contient pas que des string, on reject la Promise.
- d. Écrivez deux fonctions qui utilisent des promesses que vous pouvez enchaîner! La première fonction, makeAllCaps(), prendra un tableau de mots et les mettra en majuscule, puis la deuxième fonction, sortWords (), triera les mots par ordre alphabétique. Si le tableau contient autre chose que des chaînes, il doit générer une erreur.

## 5. Encore des promesses

Dans ce code, votre fonction reçoit une donnée de paramètre. Vous devez modifier le code ci-dessous en fonction des règles suivantes:

- Votre fonction doit toujours renvoyer une promesse
- Si les données ne sont pas un nombre, renvoyez une promesse rejetée instantanément et donnez les données "erreur" (dans une chaîne)
- Si les données sont un nombre impair, renvoyez une promesse résolue 1 seconde plus tard et donnez les données «impaires» (dans une chaîne)
- Si les données sont un nombre pair, renvoyez une promesse rejetée 2 secondes plus tard et donnez les données «paires» (dans une chaîne)

```
function job(data) {  
  return something;  
}  
  
module.exports = job;
```

## 6. Un petit QCM

Vous ne devez exécuter le code que pour vérifier vos réponses, par pour répondre à la question.

a. Quelle est la sortie du code suivant :

```
function job() {  
  return new Promise(function(resolve, reject) {  
    reject();  
  });  
}  
  
let promise = job();  
  
promise  
  
  .then(function() {  
    console.log('Success 1');  
  })  
  
  .then(function() {  
    console.log('Success 2');  
  })  
  
  .then(function() {  
    console.log('Success 3');  
  })  
  
  .catch(function() {  
    console.log('Error 1');  
  })  
  
  .then(function() {  
    console.log('Success 4');  
  });
```

- A. Error 1
- B. Success 1, Error 1
- C. Success 1, Success 2, Success 3, Success 4
- D. Error 1, Success 1, Success 3, Error 1, Success 4
- E. Error 1, Success 1, Success 2, Success 3, Success 4
- F. Error 1, Success 4

b. Quelle est la sortie du code suivant :

```
function job(state) {  
  return new Promise(function(resolve, reject) {  
    if (state) {  
      resolve('success');  
    } else {  
      reject('error');  
    }  
  });  
}  
  
let promise = job(true);  
  
promise  
  
  .then(function(data) {  
    console.log(data);  
  
    return job(false);  
  })  
  
  .catch(function(error) {  
    console.log(error);  
  
    return 'Error caught';  
  })  
  
  .then(function(data) {  
    console.log(data);  
  
    return job(true);  
  })  
  
  .catch(function(error) {  
    console.log(error);  
  });
```

- A. error, success, Error caught
- B. success, success
- C. success, error, success, error
- D. success, error, Error caught
- E. success, error, Error caught
- F. error, Error caught, success
- G. error, Error caught, success, error
- H. success, error, error
- I. success, success, success

```

function job(state) {
  return new Promise(function(resolve, reject) {
    if (state) {
      resolve('success');
    } else {
      reject('error');
    }
  });
}

let promise = job(true);

promise

.then(function(data) {
  console.log(data);

  return job(true);
})

.then(function(data) {
  if (data !== 'victory') {
    throw 'Defeat';
  }

  return job(true);
})

.then(function(data) {
  console.log(data);
})

.catch(function(error) {
  console.log(error);

  return job(false);
})

.then(function(data) {
  console.log(data);

  return job(true);
})

.catch(function(error) {
  console.log(error);

  return 'Error caught';
})

.then(function(data) {
  console.log(data);

  return new Error('test');
})

.then(function(data) {
  console.log('Success:', data.message);
})

.catch(function(data) {
  console.log('Error:', data.message);
});

```

c. Quelle est la sortie du code précédent :

- A. error, error, Error caught, Error: test
- B. success, success, Error caught, Success : Test
- C. success, Defeat, error, Error caught, Success : Test
- D. error, Error caught, Success : Test
- E. success, Defeat, error, Error caught, Error : Test
- F. success, error, Defeat, Success: test

## **7. Toujours des promesses..**

Faire le challenge à la page suivante :

<https://www.codingame.com/playgrounds/347/javascript-promises-mastering-the-asynchronous/the-last-challenge>