Etablissement : ISET Charguia	**Département :** Technologies de l'Informatique
Matière : Atelier Programmation côté serveur	**Classe :** DSI3
Année Universitaire : 2020 - 2021	

TP n° 0 : Node.JS – Les fonctions callback

Objectifs du TP :
- Programmation asynchrone.
-

.

N.B : L'ensemble des exercices doivent être placés dans un répertoire nommé « TP1 » placé sous un répertoire portant votre nom, prénom et classe.

1. **Exercice 1 :** Ecrivez un script à base d'une fonction asynchrone pour avoir l'affichage suivant :

La question est posée et après 5 sec vous avez la réponse.

2. **Exercice 2 : Mettre en place un compte à rebours : setTimeout()**

Rappel sur les deux timers JS :

1. setTimeout() : permet d'attendre un certains temps avant d'exécuter une fonction
2. setInterval() : exécute une fonction suivant un intervalle de temps régulier

Exercice 3 : Utiliser un timer pour créer une petite horloge en Javascript.

Exercice 4 : Coder un feu de circulation.
- le feu reste 5 secondes au vert.
- le feu reste 2 secondes à l'orange.

- le feu reste 3 secondes au rouge puis on relance un cycle Indice

Les Callbacks

Exercice 1 : Réalisez un script contenant une fonction **output** qui accepte une fonction callback **result** qui permet de faire l'addition de deux entier. La fonction **output** affiche : « **le résultat est …. »**

Exercice 2 : Réalisez maintenant une fonction **compute (x1,x2, mycallback)**, qui accepte une fonction callback qui réalise le produit de deux entiers.

Exercice 3 : soit le code source ci-dessous, exécutez-le et dites s'il sagit d'un code bloquant ou non bloquant :

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("Program Ended");
```

Transformez ce code en utilisant une fonction de type callback pourque ça devient non bloquant. Vérifiez-le.

Javascript Promises

Rappel : la forme d'une promesse en JavaScript est la suivante :

```
a(() => {
  b(() => {
    c(() => {
      d(() => {
        // and so on ...
      });
    });
  });
});
```

qui devient comme suit:

```
Promise.resolve()
  .then(a)
  .then(b)
  .then(c)
  .then(d)
  .catch(console.error);
```

Syntaxe

```
asyncFunc()
.then(value => { /* success */ })
.catch(error => { /* failure */ });
```