Checkpoint !

Petits exercices pour reprendre les notions vues jusqu'à maintenant !

1

Vous devez écrire une fonction qui prend en paramètre un tableau contenant des valeurs de type inconnu.

```
import assert from 'assert'

function categorize(values){
    // ???
}

assert.deepStrictEqual(
    categorize([1, 'hello', sayHi, 'world', true, 0n, 1000]),
    {
        number: [1, 1000],
        string: ['hello', 'world'],
        function: [sayHi],
        boolean: [true],
        bigint: [0n]
    }
)
```

Vous devez renvoyer un objet où les clés sont des noms de type JS et les valeurs sont celles qui correspondent au type dans le tableau passée à la fonction.

Par exemple, pour l'entreé suivante:

```
[1, 'hello', function sayHi(){ console.log('hi') }, 'world', true, 0n,
1000]
```

On devrait avoir la sortie suivante:

```
{
    'string': ['hello', 'world'],
    'function': [function sayHi(){ console.log('hi') }],
    'boolean': [true],
    'number': [1, 1000],
    'bigint': [0n]
}
```

Il est interdit d'utiliser des boucles `for`/`while`/`do while`.

2

Vous devez faire en sorte qu'une méthode `dedup()` soit disponible sur tous les tableaux JS du programme.

Cette méthode doit renvoyer un nouveau tableau, se basant sur le tableau courant, mais qui ne contient aucun doublon.

Pour comparer des doublons, on utilisera `===`.

Interdit d'utiliser des objets `Set`.

```
import assert from 'assert'

// ???

assert.deepStrictEqual(
    [1, 1, 6, 2, 3, 2, 2, 4, 6, 6, 1].dedup(),
    [1, 6, 2, 3, 4]
)
```

3

Ecrire une fonction qui filtre certaines clés d'un objet passé en paramètre.

```
function filterObject(/* ??? */){
    // ???
}

assert.deepStrictEqual(
    filterObject(
        {
            foo: 1,
            bar: 'hello',
            baz: true
        },
        (key, value) => key === 'foo' || value === 'hello'
    ),
    {
        foo: 1,
        bar: 'hello'
    }
)
```

La fonction doit renvoyer **un nouvel objet**, basé sur celui passé en paramètre mais qui ne contient pas les clés/valeurs filtrées.

Le filtrage se fait via un *prédicat* (fonction qui renvoie `true` ou `false`) passé en paramètre. La clé et la valeur sont fournies en tant que paramètres de ce prédicat.

4

Transformer ce code en utilisant le style `async`/`await`.

Vous n'avez le droit d'appeler la méthode `.then()` ou `.catch()` qu'une seule et unique fois.

```
// ne pas toucher
const asyncJob = (n) => Math.random() > 0.5 ? Promise.resolve(n + 1) :
Promise.reject(Error('boom'))

// a transformer
asyncJob(0)
    .then(i => asyncJob(i))
    .then(i => Promise.all([
        asyncJob(i),
        asyncJob(i),
        asyncJob(i)
    ]))
    .then(([x, y, z]) => asyncJob(x + y + z))
    .then(total => console.log(total))
    .catch(err => console.error(`gestion erreur globale: ${err.message}`))
```

5

Implémenter `Promise.race`.

```
import assert from 'assert'
const pSetTimeout = ms => new Promise(resolve => setTimeout(resolve, ms))

function race(promises){
    // ???
}

assert.doesNotReject(
    race([
        pSetTimeout(4000).then(() => 1),
        pSetTimeout(3000).then(() => 2),
        pSetTimeout(2000).then(() => 3),
        pSetTimeout(1000).then(() => 4)
    ]).then(winner => {
        assert(winner === 4)
    })
)
```

6

Implémenter `Promise.all`.

```
import assert from 'assert'

function all(promises){
    // ???
}

assert.doesNotReject(
    all([
        Promise.resolve(1),
        Promise.resolve(2),
        Promise.resolve(3),
    ])
    .then(([x, y, z]) => {
        assert(x === 1)
        assert(y === 2)
        assert(z === 3)
    })
)
```

Rendu

Quand vous aurez terminé, uploadez vos résultats sur un pastebin ou un dépôt/gist github, et envoyez moi l'URL via Teams ou à `r.nzaou@sncf.fr`.

Les 4 premiers exercices sont à réaliser pendant la séance.

Les 2 derniers (#5 et #6) sont à réaliser pour la séance suivante.