Travel Planner

Callback functions

Exercise 1:

Create a function named `processTravelPlans`. This function should accept two parameters: an array of travel plan objects and a callback function.

The structure of a travel plan object in the array should be like this:

```
{
  id: 1,
  city: "City name",
  activities: [
    { name: "Activity name", duration: 2 }
  ]
}
```

Your `processTravelPlans` function should perform the following tasks:

1. Check if the first argument is an array. If not, throw an error with the message 'Expected an array of travel plans'.

2. For each object in the array, check if it contains the properties `id`, `city` and `activities`. If any object does not contain these properties, throw an error with the message `Travel plan with ID ${travelPlan.id} is not in the right structure` (the id property will always be).

3. If the objects structure is correct, clone the array to avoid mutating the original one: `const newTravelPlans = JSON.parse(JSON.stringify(travelPlans));`.

4. Loop over each object in the cloned array, and pass each object to the callback function.

5. Return the new array.

Exercise 2:

Create a function named `processActivities`. This function should accept two parameters: an array of activity objects and a callback function.

The structure of an activity object in the array should be like this:

```
{
  id: 1,
  name: "Activity name",
  type: "Activity type",
  duration: 2
}
```

Your `processActivities` function should perform the same tasks as in Exercise 1, but with activities. Adjust the error messages accordingly.

Exercise 3:

Create a function named `processPlan`. This function should accept two parameters: a travel plan object and a callback function.

The structure of the travel plan object should be like this:

```
{
  id: 1,
  city: "City name",
  activities: [
    { name: "Activity name", duration: 2 }
  ]
}
```

Your `processPlan` function should perform the same tasks as in the previous exercises, but with a single travel plan object. The structure verification should check for `id`, `city`, and `activities`. Adjust the error messages accordingly.

You could test the `processTravelPlans` function, for instance, with a callback that calculates the total duration of all activities in each travel plan like this:

```
let updatedTravelPlans = processTravelPlans(travelPlans, plan => {
  let totalDuration = plan.activities.reduce((sum, activity) => sum +
activity.duration, 0);
  plan.totalDuration = totalDuration;
  return plan;
});

console.log(updatedTravelPlans);
```

This will return a new array of travel plans, each with a `totalDuration` property that represents the total duration of all activities, without modifying the original `travelPlans` array.