Taking advantage of Functional Programming in TypeScript (Part 2)

We'll see another example about functional programming and its concepts. We'll use Pure Functions and High Order Functions.

The example

Suppose that we have the Plan concept, that is a set of operations that can be executed. Suppose that we execute a plan every day of a week. We can represent that information with a matrix, where every row represents a week, and every column is a day of that week. For example:

```
Row = Week
Column = Week's days

1|1|3|2|3|2|1
-+-+-+-+-+-+-
1|3|2|1|3|2|1
-+-+-+-+-+-+-
3|2|3|2|2|3|1
```

The plan #1, in week #1 will be executed on monday, thursday and sunday.

We want to represent that data and want to be able to print as you can see in previous example.

Data types and helper functions

We'll represent a plan as a number:

```
type Plan = number;
```

to generate random plans we'll use this function:

```
const randomNumber = (max: number) => Math.floor(Math.random() * max) + 1;
```

We'll need two new types: PlanTable that is a Plan matrix, and PlanTablePosition, that is a tuple of two numbers:

```
type PlanTable = Plan[][];

type PlanTablePosition = [number, number];
```

We'll write some pure functions to represent our plan table. We'll combine a set of simple functions to get the correct representation.

```
//allows us to concat to numbers with a separator. For example: concat('|')(1,2)
will return 1|2
const concat = (separator: string) => (left: number, right: number) => left +
separator + right;

//allows us to get a week representation like a '|' char and numbers combination.
const showWeek = (week: Plan[]) => week.reduce(concat('|'));
```

As you can see, we use the concat function to get the week representation. We are combining data and a function to get a final result.

Following that concept, we can get the full PlanTable representation:

```
function showPlanTable(table: PlanTable) {
  const weekSeparator = "\n" + table[0].map(() => '-').reduce((concat('+'))) +
'\n';
  return table.map(showWeek).reduce(concat(weekSeparator));
}
```

We are using pure functions as you can see, we aren't modifying any input parameter.

Finally, we need three more functions: createPlanTable to generate a random table, getPositionsFromArray is a generic function to get the array's indices of all occurrences of a value; and getPlanTablePositions return an array of PlanTablePosition of all occurrences of one concrete plan:

```
function createPlanTable(numberOfWeeks: number) {
  const planTable: PlanTable = [];
  for (let i = 0; i < numberOfWeeks; i++){
    planTable[i] = [];

    for (let j = 0; j < 7; j++){
      planTable[i][j] = randomNumber(4);
    }
  }
  return planTable;
}
function getPositionsFromArray<T>(array: T[], value: T) {
  return array.reduce((acc, current, index) => value === current ?
acc.concat(index) : acc, []);
}

function getPlanTablePositions(table: PlanTable, plan: Plan) {
   return table.reduce((acc, currentWeek, index) =>
acc.concat(getPositionsFromArray(currentWeek, plan).map(plan => [index, plan])),
[]);
}
```

Now we can execute an example:

```
const table = createPlanTable(3);
console.log(showPlanTable(table));
console.log(getPlanTablePositions(table, 2));
```

We are creating that example PlanTable:

```
3|3|3|4|1|4|4
-+-+-+-+-+-+-
1|2|4|3|4|2|3
-+-+-+-+-+-+-
1|4|1|4|2|2|3
```

and gets all positions of plan #2:

```
[[1, 1], [1, 5], [2, 4], [2, 5]]
```

You can get that example here