Context Lab

- Due No Due Date
- Points 1
- Submitting a website url

(https://github.com/learn-co-curriculum/phase-1-context-lab) (https://github.com/learn-co-curriculum/phase-1-context-lab/issues/new)

Learning Goals

- Update our time-card and payroll application to use the employee record as context rather than passing it as an argument.

Introduction

In this lab, we're going to build the time-card and payroll application using the record-oriented approach again. This lab will feature the same topic and area of work as the previous lab; *however*, *how* we call and use functions will change with our new knowledge. While the code will stay *mostly* the same, you're going to need to use `this` a lot more.

The tests guide you to implementing a time card system: when someone enters the company's state of the art technical office, the employee has to insert their card in a time-clock which will record the time they came in. When it's time to leave, the employee will "punch out."

For simplicity's sake, we'll make these assumptions:

1. Assume that employees always check in and check out
2. Assume employees always check in and out on the hour
3. The time is represented on a 24-hour clock (1300 is 1:00 pm); this keeps the math easier and is the standard in most of the world
4. When timestamps are needed, they will be provided as Strings in the form: "YYYY-MM-DD 800" or "YYYY-MM-DD 1800" e.g. "2018-01-01 2300"
5. Employees will never work across days i.e. in at 2200 and out at 0400 the next day.

The lab tests will guide you toward a solution. Keep in mind, the goal is to understand how to "grow" an application in "record-oriented" fashion in JavaScript, as well as pass the lab. Make sure you're learning about this app design while you pass the solutions.

Code your solution in `index.js` . To get everything passing, you will need to exercise the collection-processing strengths you developed earlier in this Phase.

As before, if you have extra time, use the guidance in the previous lab to make your application more robust.

While you will want to be guided by the tests, you will implement the following functions. To make the tests easier to read, we've provided the *signatures* of the functions. (A function *signature* is the function name, the arguments it expects, and what the function returns.) Be sure to refer back to this information as you work through the tests.

`createEmployeeRecord`

- **Argument(s)**
  - A 4-element Array of a `String` , `String` , `String` , and `Number` corresponding to a first name, family name, title, and pay rate per hour
- **Returns**
  - JavaScript `Object` with keys:
  - `firstName`
  - `familyName`
  - `title`
  - `payPerHour`
  - `timeInEvents`
  - `timeOutEvents`
- **Behavior**
  - Loads `Array` elements into corresponding `Object` properties. *Additionally*, initialize empty `Array` s on the properties `timeInEvents` and `timeOutEvents` .

`createEmployeeRecords`

- **Argument(s)**
  - `Array` of `Arrays`
- **Returns**
  - `Array` of `Object` s
- **Behavior**

- Converts each nested `Array` into an employee record using `createEmployeeRecord` and accumulates it to a new `Array`

`createTimeInEvent`

- **Argument(s)**
  - A date stamp (`"YYYY-MM-DD HHMM"`), where time is expressed in [24-hour standard](https://en.wikipedia.org/wiki/24-hour_clock) (https://en.wikipedia.org/wiki/24-hour_clock)
- **Returns**
  - The record that was just updated
- **Behavior**
  - Add an `Object` with keys:
  - `type` : Set to `"TimeIn"`
  - `hour` : Derived from the argument
  - `date` : Derived from the argument

`createTimeOutEvent`

- **Argument(s)**
  - A date stamp (`"YYYY-MM-DD HHMM"`), where time is expressed in [24-hour standard](https://en.wikipedia.org/wiki/24-hour_clock) (https://en.wikipedia.org/wiki/24-hour_clock)
- **Returns**
  - The record that was just updated
- **Behavior**
  - Add an `Object` with keys:
  - `type` : Set to `"TimeOut"`
  - `hour` : Derived from the argument
  - `date` : Derived from the argument

`hoursWorkedOnDate`

- **Argument(s)**
  - A date of the form `"YYYY-MM-DD"`
- **Returns**
  - Hours worked, an `Integer`

- **Behavior**
  - Given a date, find the number of hours elapsed between that date's timeInEvent and timeOutEvent

`wagesEarnedOnDate`

- **Argument(s)**
  - A date of the form `"YYYY-MM-DD"`
- **Returns**
  - Pay owed
- **Behavior**
  - Using `hoursWorkedOnDate` , multiply the hours by the record's payRate to determine amount owed. Amount should be returned as a number.

`allWagesFor`

- **Argument(s)**
  - *None*
- **Returns**
  - Sum of pay owed to **one** employee for all dates, as a number
- **Behavior**
  - Using `wagesEarnedOnDate` , accumulate the value of all dates worked by the employee in the record used as context. Amount should be returned as a number. **HINT**: You will need to find the available dates somehow....

`findEmployeeByFirstName`

- **Argument(s)**
  - `srcArray` : Array of employee records
  - `firstName` : String representing a first name held in an employee record
- **Returns**
  - Matching record or `undefined`
- **Behavior**
  - Test the `firstName` field for a match with the `firstName` argument

`calculatePayroll`

- **Argument(s)**
  - `Array` of employee records
- **Returns**
  - Sum of pay owed for **all** employees for all dates, as a number
- **Behavior**
  - Using `allWagesFor` for each of the employees, accumulate the value of all dates worked by the employee in the record used as context. Amount should be returned as a number.

A Mystery on the Horizon

You'll notice that in this lab we give you the implementation of `allWagesFor` . As part of writing this challenge, we ran right smack into one of the most famous bugs in JavaScript land: "the lost context bug." Because we haven't taught you to deal with it yet, we've "given" you this function.

If you have extra time, try researching this topic on your own. We'll tell you all about it in our next lesson, though.

Resources

- `bind` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind)
- `call` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/call)
- `apply` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/apply)