|  |  |
| --- | --- |
| Sprint 15, Assignment 4.5 Please also update the doc name with correct numbers. | |
| Assignment type: JS Interactive | |
| Assignment name  Can remain the same as the assignment topic name, or…  can be phrased in the active/imperative voice(depending on the task), e.g. “Create your own X” | "Create Original Errors" |
| Description  The essential tasks a student must perform in order to complete the assignment | The client would like your user database script to include some custom errors: 1) when an incorrect operation verb is used (so, currently, a verb other than 'add' or 'delete') and 2) when an incorrect ID # is used during 'delete' operation (when someone attempts to delete an entry that doesn't exist) |
| Link to full code file (Answer)  Push a full code file to our [GitHub repo](https://github.com/bitdegree-foundation/academy-assignments-code-files) for this sprint’s assignment and paste a link to that file here. Please write the file in a concise and clear format, according to the “[example-js-code-file](https://github.com/bitdegree-foundation/academy-assignments-code-files/blob/master/example-js-code-file.js)” found in our repo. |  |

|  |  |  |
| --- | --- | --- |
| # | Step  Write each small step of the task. These assignments should be doable by students on their own (without any BitDegree instructor help). It can be any number of steps, but keep them pretty clear and separate (don’t combine 2 steps into 1 step). | 1x hint  Write brief text or partial code that would help the student figure out how to move forward or accomplish the given step. |
| 1 | (In the index.js file of your "Create a user database script with Node.js" assignment) Add two parameters to your 'run' function called "operationIncoming" and "dataIncoming". Change run()'s declared constants 'operation' and 'incomingData' to store these parameters, respectively (instead of process.argv[2] or .argv[3]) |  |
| 2 | Immediately-before your 'run()' invocation, declare a function named "checks" and give it two parameters (named "input1" and "input2") . Within your 'checks' function, declare two constants named "operation" and "incomingData"; store input1 and input2 in them, respectively. Declare a third, constant named currentOperationsList, and store in it your currently available operations ("add" and "delete") as an array of strings. Invoke your 'check's function immediately-after itself. "Turn off" your final line's invocation of 'run()'. |  |
| 3 | (Within your 'checks' function) Declare a function named "verifyOperationInput" and give it two parameters: "operationInput" and "currentOperations". Using these two parameters, log the index of the operationInput in the list of currentOperations. In order to test your function, invoke it immediately-after itself and pass in two arguments: "add" and your currentOperationsList (you should see "0" returned) | Hint: use curentOperations.indexOf() |
| 4 | Add a conditional to your verifyOperationInput function that assesses whether the indexOf the current operationInput is greater than -1 and logs a message like "That doesn't exist as an operation" if it is not. Test your function by passing in a verb that is not in your currentOperationsList (like "move", "elevate", or "jump-around"). | Note: the .indexOf method will return "-1" if it cannot find the given element/property  Hint: remember that vanilla conditionals ask whether the supplied statement has a boolean value of true  Hint: use "!" to ask if (it is true that) the operationInput is \*not\* greater than "-1" |
| 5 | Turn-off/delete the log statement of your conditional, and instead have it do three things: 1) Declare a constant named "err" that stores a new Error (with a message like "that isn't an available operation"), 2) Sets 'err's "code" parameter to something like "ERR\_CAN\_ONLY\_BE\_A\_VALID\_OPERATION", and 3) throw[s] err. Test your function to see if your new error message appears. | Hint: use "new Error()"  Hint: use "err.code"  Hint: use "throw" |
| 6 | Alter your invocation to pass the variable 'operation' into verifyOperationInput. Invoke your "run" function afterwards (as a second invocation), and pass in two parameters: 'operation' and 'incomingData'. After/outside-of your 'checks' function, turn off your 'checks' invocation and declare an empty try...catch statement |  |
| 7 | "Turn off" or delete your 'checks' invocation. In your try...catch, make your 'try' 1) log a message like `Try was executed`, 2) invoke the 'checks' function twice. For the first invocation, pass in correct values (like "add" and "Denzel Washington"); for the second, pass in an \*incorrect\* verb (like "spin"). Test your that your 'try' is working by "turning-off" the first (leaving the second "on"), and then turning the second "off" (and making the first "on"). |  |
| 8 | Pass "error" into your catch. Give your catch a conditional that will do three things: first, if the error.code is ERR\_CAN\_ONLY\_BE\_A\_VALID\_OPERATION, it will log a message like `That is not an available operation`. Second, if the code is "ERR\_MUST\_BE\_A\_CURRENT\_USER" (which we will use later), have the conditional log a message like `A user with that ID # does not exist`. Finally, in all other cases, have the conditional simply log the error. | Hint: use if ... else if ... else |
| 9 | (In your 'run' function) Find the conditional that checks whether the operation is "delete" and insert into-it two instructions (prior to 'data' = deleteUser): 1) log a message like "DELETE was used", and 2) declare a second, nested conditional. You will use this nested conditional to eventually check whether the user-supplied ID number matches any entries in the users.json database. For now, make it simply check whether the parsed-integer version of incomingData is (equal to) 5, and log a message like "The fifth user entry is about to be deleted" if true. To test this, add a third 'checks' invocation to your 'try' statement that will pass "delete" and "5" in. |  |
| 10 | (Back in your 'run's "delete" conditional) Change your nested-conditional's parameters to, using the .find method, do two things: 1) return any element in 'data's array that matches incomingData's parsed-number, and 2) evaluate whether what's returned is undefined (instead of 5) |  |
| 11 | Make your nested-conditional do three things (if the ID number can't be found): 1) Declare a constant named "err" that stores a new Error (with a message like "that isn't an available operation"), 2) Sets 'err's "code" parameter to something like "ERR\_CAN\_ONLY\_BE\_A\_VALID\_OPERATION", and 3) throw[s] err. | Hint: use "new Error()"  Hint: use "err.code"  Hint: use "throw" |
| 12 | In your try statement, add a fourth 'checks' invocation with the parameters of "delete" and "10" (or any number that wont match the current ID's in your users.json file). Test your script to see if your ID error message appears. |  |
| 13 |  |  |
| ... |  |  |