BTI225 Assignment 2

# Submission Deadline:

Friday, February 2 @ 11:59pm

# Assessment Weight:

8% of your final course Grade

# Objective:

Practice Array Operations, Custom Objects & Working with Data.

# Specification:

Write a custom object called produceItemDB that will serve as an object "database" for a strictly formatted set of sample data. To begin, access the following file and copy/paste the contents into web console:

**Attached file: a2-test-data.js.txt**

Be sure to save this file as "assignment2.js".

This file includes two blocks of code (you will write your produceItemDB object between them):

* The block labeled **TEST DATA** at the bottom is currently commented out, but you must uncomment it to test your solution when you're ready. After running your code with the TEST Data uncommented, the output in the console or terminal should look like the sample on the file **A2-TEST-DATA-Output.pdf**.
* The second block of code at the top of the file (labeled **ALL DATA**) is an array of objects called "allData". All the objects in this array follow the same format:

{ **type**: "produceItem" */\* or "supermarket" or "address"\*/*, **data**: { */\* object with properties related to the "type" \*/* }

The "**type**" is the type of data with one of "**produceItem**", "**supermarket**", or "**address**", and the "**data**" is an object with properties that belong to the "produceItem" or "supermarket" or "address". The properties in the "**data**" object will always have the same names and follow the following convention for each of the following "types":

|  |  |
| --- | --- |
| **type: "produceItem"** | |
| **PLUCode** | Price look-up codes - the primary key used to uniquely identify each produce Item |
| **supermarketId** | A foreign key identifying "supermarket" that the produce item belongs to |
| **type** | The produce item's type, with one of "Fruits", "Herbs", "Nuts", "Vegetables", or "Dried Fruits" |
| **name** | The produce item's name |
| **unit** | The produce item's unit regarding its price |
| **price** | The produce item's sale price |
| **modified** | The date the produce item was added or modified to the system |

|  |  |
| --- | --- |
| **type: "supermarket"** | |
| **supermarketId** | A primary key that uniquely identifies the supermarket |
| **addressId** | A foreign key identifying the address of the supermarket |
| **name** | The name of the supermarket |
| **website** | The website of the supermarket |

|  |  |
| --- | --- |
| **type: "address"** | |
| **addressId** | A primary key used to uniquely identify each address. |
| **street** | The street of the address. |
| **city** | The city of the address. |
| **province** | The province (two-letter abbreviation) of the address. |
| **postalCode** | The postal code or ZIP code of the address (where applicable). |

## **Part A: ProduceItemDB Object**

In your assignment2.js file, underneath the "allData" array, declare an object called **produceItemDB** using "Object Literal Notation". This object will **contain** the following **properties** & **methods:**

## **Array Properties**

The following are internal arrays that will serve as the primary storage mechanism within the produceItemDB object for all of our sample data.

* **produceItems**This is an array that will contain all the sample data objects that are of **type "produceItem".** It is initialized as an empty array ( ie, [] ) and will be manipulated using the methods in the produceItemDB Object.
* **addresses**This is an array that will contain all the sample data objects that are of **type "address".** It is initialized as an empty array ( ie, [] ) and will be manipulated using the methods in the produceItemDB Object.
* **supermarkets**This is an array that will contain all the sample data objects that are of **type "supermarket".** It is initialized as an empty array ( ie, [] ) and will be manipulated using the methods in the produceItemDB Object.

## **Main "loadData" Function ( Method )**

The loadData Method is the first method that will be invoked on your produceItemDB object. It is this method that takes all of the sample data and loads it into the correct arrays (ie: "produceItems", "addresses", or "supermarkets"). It takes one parameter, the **allData** array from the top of your assignment2.js file and processes it one array element at a time using the following rules:

* if type is "supermarket", insert the "data" object into the "supermarkets" array (**HINT**: there is a function we will write called **addSupermarket(supermarketObj)** that will be perfect for this)
* if type is "produceItem", retrieve the "data" object and set it's "modified" property to the current date and add it into the "produceItem" array (**HINT**: there is a function we will write called **addProduceItem(produceItemObj)** that will be perfect for this)
* if type is "address", insert the "data" object into the "addresses" array (**HINT**: there is a function we will write called **addAddress(addressObj)** that will be perfect for this)

Once this method has run, your "produceItems" array should contain **10** "produceItem" data objects, your "addresses" array should contain **7** "address" data objects, and your "supermarkets" array should contain **6** "supermarket" data objects and your "database" is built.

## **Methods to work with "produceItem" data**

The following are all methods that deal primarily with the "produceItems" array. Any output for these functions is meant for the web console, so whenever the term "print" or "output" is used, you may assume that we're outputting to the console with console.log(). HINT: to refer to the "produceItems" array from within these methods, use the "this" keyword, ie: "this.produceItems".

* **addProduceItem ( produceItemObj )**

This method takes an object of type "produceItem", sets it's "modified" property to the current date and adds it to the "produceItems" array.

* **printProduceItemByCode( PLUCode)**

This method takes a string representing a **PLUCode** and outputs all of the produce item object data for the corresponding **PLUCode** from the "produceItems" array:

Produce Item: **PLUCode**: ***name, type, price*** (per ***unit*** )  
Date added: ***modified***

For Example, **produceItemDB.printProduceItemByCode("3000")**:

Produce Item 3000: Apples fresh & generic, Fruits, $2.49 (per lb)

Date modified: Sun, 14 Jan 2024 19:32:18 GMT

* **printAllProduceItems( )**

This method takes no parameters and simply outputs all produceItems in the **same format as above**, including a header at the top of the output stating "All Produce Items (count: [put the number of ProduceItem objects in the array here])", ie:

All Produce Items (count: 10)

Produce Item 3000: Apples fresh & generic, Fruits, $2.49 (per lb)

Date modified: Sun, 14 Jan 2024 19:32:18 GMT

Produce Item 3062: Bay Leaves Culinary, Herbs, $4.95 (per 100g)

Date modified: Sun, 14 Jan 2024 19:32:18 GMT

*…etc…*

* **printProduceItemsBySupermarket ( supermarketId )**

This method takes a number representing a supermarketId and outputs all of the produce Item data for the corresponding supermarketId from the "produceItems" array in the **same format as above**, including a header at the top of the output in the format: Produce Items in Supermarket ***name*** (count: *[put the number of produce items with the given* supermarketId *in the array here])*

For Example, **produceItemDB.printProduceItemsBySupermarket(222);**

Produce Items in Supermarket: Loblaws Bayview Village (count: 3 )

Produce Item 3000: Apples fresh & generic, Fruits, $2.49 (per lb)

Date modified: Sun, 14 Jan 2024 19:55:35 GMT

Produce Item 3079: Asparagus Green, Vegetables, $3.5 (per 200g)

Date modified: Sun, 14 Jan 2024 19:55:35 GMT

Produce Item 4070: Celery Bunch, Vegetables, $3.99 (per bunch)

Date modified: Sun, 14 Jan 2024 19:55:35 GMT

**HINT**: there is a function we will write called **getSupermarketById(supermarketId)** that can help to get the name of the supermarket with id (for example, "222").

* **printProduceItemsByType( produceType )**

This method takes a string representing a produce ***type*** and outputs all of the produce Item data for the corresponding produce type from the "produceItems" array in the **same format as above**, including a header at the top of the output in the format: Produce Items by product type ***type*** (count: *[put the number of produce items with the given* produceType *in the array here])*

For Example, **produceItemDB.printProduceItemsByType("Fruits");**

Produce Items by produce type: 'Fruits' (count: 2 )

Produce Item 3000: Apples fresh & generic, Fruits, $2.49 (per lb)

Date modified: Sun, 14 Jan 2024 20:41:52 GMT

Produce Item 3016: Pears General, Fruits, $1.99 (per lb)

Date modified: Sun, 14 Jan 2024 20:41:52 GMT

* **removeProduceItemByCode ( PLUCode)**

This method takes a string representing a **PLUCode** and searches through the produceItems array and removes the produceItem with the matching "**PLUCode**" property from the array.

**HINT:** to remove elements from the middle of an array, you can either build a new array one element at a time, making sure to NOT include the element you don't want, or check out the [Array.prototype.splice()](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/splice) method.

## **Methods to work with "address" data**

The following are all methods that deal primarily with the "addresses" array. Any output for these functions is meant for the web console, so whenever the term "print" or "output" is used, you may assume that we're outputting to the console with console.log(). HINT: to refer to the "addresses" array from within these methods, use the "this" keyword, ie: "this.addresses".

* **addAddress ( addressObj )**

This method takes an object of type "address", and adds it to the "addresses" array.

* **getAddressById ( addressId )**

This method takes a number representing an addressId and searches through the "addresses" array looking for an address object that has a matching "addressId ". This method will return the corresponding address object, for example: **getAddressById(1453)** will return the object with address:

" 2877 Bayview Ave , Toronto, ON M2K 2S3".

* **printAllAddresses( )**

This method takes no parameters and simply outputs all addresses in the following format, including a header at the top of the output stating "All Addresses (count: [put the number of Address objects in the array here])": ie:

All Addresses (count: 6)

Address ***addressId*** : ***street,* *city***, ***province***. ***postalCode***

For Example, **printAllAddresses();**

All Addresses (count: 7)

Address 1453: Bayview Ave #2877, Toronto, ON M2K 2S3

Address 1680: 6201 Bathurst St., North York, ON M2R 2A5

Address 1870: 3139 Sheridan Dr, Amherst, NY 14226

*…etc…*

* **removeAddressById ( addressId )**

This method takes a number representing an addressId and searches through the addresses array and removes the address with the matching "addressId " property **only if** the "addressId " is not referenced by any "supermarket" objects in the produceItem array.

For example, if we try to remove the address object with addressId 2024, we **cannot remove** it because the supermarket "T&T Supermarket Weldrick Store" is still uses that address.

**HINT:** to remove elements from the middle of an array, you can either build a new array one element at a time, making sure to NOT include the element you don't want, or check out the [Array.prototype.splice()](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/splice) method.

## **Methods to work with "supermarket" data**

The following are all methods that deal primarily with the "supermarkets" array. Any output for these functions is meant for the web console, so whenever the term "print" or "output" is used, you may assume that we're outputting to the console with console.log(). HINT: to refer to the "supermarkets" array from within these methods, use the "this" keyword, ie: "this.supermarkets".

* **addSupermarket ( supermarketObj )**

This method takes an object of type "supermarket", and adds it to the "supermarkets" array.

* **getSupermarketById ( supermarketId )**

This method takes a number representing a supermarketId and searches through the "supermarkets" array looking for a supermarket object that has a matching "supermarketId". This method will return the corresponding supermarket object, for example: **produceItemDB.getSupermarketById(225);** will return the object with name "Metro Bathurst & Steeles".

* **printAllSupermarkets( )**

This method takes no parameters and simply outputs all supermarkets (including their address from the "addresses" array - **HINT**: the function **getAddressAddressById(addressId )** is perfect for this), in the following format, including a header at the top of the output stating "All Supermarkets (count: [put the number of Supermarket objects in the array here])": ie:

All Supermarkets (count: 6)

Supermarket ***supermarketId***: ***name***  
Location (***addressId)*** : ***street,* *city***, ***province***. ***postalCode***

For Example, **printAllSupermarkets ();**

Supermarket 222: Loblaws Bayview Village

Location (1453): 2877 Bayview Ave, Toronto, ON M2K 2S3

Supermarket 225: Metro Bathurst & Steeles

Location (1680): 6201 Bathurst St., North York, ON M2R 2A5

Supermarket 325: Whole Foods Market

Location (1870): 3139 Sheridan Dr, Amherst, NY 14226

*…etc…*

## TEST DATA Output

Once you're ready to test your solution, uncomment the "TEST DATA" block of code and run your code. If your console output looks like the sample on the file **A2-TEST-DATA-Output.pdf** provided, everything should be working properly.

Please Note, your "Date added: " dates will be different, as you will run your program at a different time.

## Assignment Submission:

* Make sure that the "TEST DATA" block of code is uncommented and that your solution gives the correct output (see **A2-TEST-DATA-Output.pdf**) in the web console when run.
* Add the following declaration at the top of your code:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* BTI225 – Assignment 02  
\* I declare that this assignment is my own work in accordance with Seneca Academic Policy.

\* No part of this assignment has been copied manually or electronically from any other source  
\* (including web sites) or distributed to other students.  
\*   
\* Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* Submit your **assignment2.js** file to My.Seneca under **Assignments** -> **Assignment** 2

## Important Note:

* **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
* After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.