CS 315 – Homework 1

Associative Arrays in Dart, JavaScript, Lua, PHP, Python, Ruby and Rust

Maryam Shahid

21801344

Section 01

November 25, 2020

Dart

1. **Initialize**

   Dart uses Map objects as associative arrays, which is a collection of key/value pairs. The elements can be initialized in the declaration of the array as the following:

   ```
   var myArray =  {'age': 20, 'cat': 'Jem', 'kitten': 'Beans', 'fruit': 'Peach'};
   ```

   In dart, var is used to declare a variable without a specified type. myArray is an associative array with the keys: age, cat, kitten, fruit, and the values: 20, Jem, Beans, Peach respectively.

2. **Get the value for a given key**

   As in most languages, dart uses square brackets as legal subscript and uses them to obtain values of the associated key in the array.

   ```
   myArray['cat'];
   ```

   This code segment will return the value of the key ‘cat’ – which is “Jem”. It can be printed to the console using the print statement as follows:

   ```
   print(myArray['cat']);

   Output: Jem
   ```

3. **Add a new element**

   The following statement will add the key ‘color’ with the value ‘Black’ to the associative array (in case the key is already present, its value will be modified).

   ```
   myArray['color'] = 'Black';
   ```

4. **Remove an element**

   Dart provides the method “remove” to remove a key and its associated value from the array (map). The following statement removes the key-value pair “age: 20”.

   ```
   myArray.remove('age');
   ```

5. **Modify the value of an existing element**

An existing element's value is modified the same way a new element is entered. The following statement changes the value of the key "color" from "Black" to "Yellow".

```
myArray['color'] = 'Yellow';
```

6. **Search for the existence of a key**

Dart provides the method containsKey() which returns true if the array (map) contains the given key. Since 'cat' is a key in the array, it will return true. Whereas 'dog' will return false.

```
myArray.containsKey('cat');
myArray.containsKey('dog');
```

The result can be output using the print statement.

```
print(myArray.containsKey('cat');
print(myArray.containsKey('dog');

Output: true
        false
```

7. **Search for the existence of a value**

Similarly, the method containsValue() returns true if the array (map) contains the given value. The result can be output using the print statement.

```
myArray.containsValue('Jem');
myArray.containsValue('Orange');

Output: true
        false
```

8. **Loop through an associative array, apply a function, called foo, which simply prints the key-value pair**

The following creates a function "foo" which simply prints the value held by its parameters – key and value. Dollar sign $ followed by a name makes up a variable name in Dart.

```
foo (key, value) {
  print('$key, $value');
}
```

Dart provides a method forEach which applies the f function to each element of the array and iterates through a loop. The loop then calls the function foo on each key-value pair.

```
myArray.forEach((key, value) => foo(key, value));

Output: cat, Jem
        kitten, Beans
        fruit, Peach
        color, Yellow
```

JavaScript

1. **Initialize**

   Javascript does not support associative arrays. However, Javascript has dynamic objects which can be used for associative arrays. The Map object in JS holds key-value pairs in the order of insertion. They are initialized by using the keyword “let”, and creates an object of Map using the “new” keyword. The following statement creates an object called myArray which is used as an associative array and has the key-values pairs as name: Maryam, cat: Jem, kitten: Beans.

   ```
   let myArray = new Map([["name", "Maryam"], ["cat", "Jem"],
   ["kitten", "Beans"]])
   ```

2. **Get the value for a given key**

   Javascript provides the method get() to retrieve the value for a given key. The following code gets the value for the key ‘cat’. The result can be output using the document.write() method.

   ```
   document.write(myArray.get('cat'))

   Output: Jem
   ```

3. **Add a new element**

   JS provides a method set(“key”, “value”) which adds a new element to the array. The following statement will add the key ‘color’ with the value ‘Blue’ to the associative array (in case the key is already present, its value will be modified).

   ```
   myArray.set('color', "Blue")
   ```

4. **Remove an element**

   JS provides the method delete() to remove a key and its associated value from the array. The following statement deletes the key-value pair “name: Maryam”.

```
myArray.delete('name')
```

5. **Modify the value of an existing element**

   An existing element's value is modified the same way a new element is entered – by using the method set(). The following statement changes the value of the key "color" from "Blue" to "Yellow".

```
myArray.set('color', 'Yellow')
```

6. **Search for the existence of a key**

   Javascript provides a method has() which returns true if the associative array (Map) has the element being called upon.

```
myArray.has('cat')
```

   The above statement will return true since 'cat' is indeed a key in myArray. The result can be displayed using the document.write() method.

```
document.write(myArray.has('cat'))

Output: true
```

7. **Search for the existence of a value**

   Javascript has not provide a method to search for the existence of a value but it can be done using a for loop which allows an iterative traversal of the array (Map). In the following code, I created a function namely checkValue which takes the value to search as its parameter and loops through the associative array. It returns true if the value exists in the array.

```
function checkValue(valueToSearch){
    for (let [key, value] of myArray){
        if( value == valueToSearch){
            return true
        }
    }
    return false;
}
```

   In the following statement, I have called checkValue() upon "Jem". It will return true since Jem is indeed a value in the array. The boolean result can be displayed using document.write().

```
document.write(checkValue("Jem")) //returns true

Output: true
```

8. **Loop through an associative array, apply a function, called foo, which simply prints the key-value pair**

   The following code creates a function foo with two parameters – key and value. Foo simply displays the values of its parameters using the document.write() function.

```
function foo(key, value){
    document.write(key + ': ' + value + '<br>');
}
```

   To loop through an associative array, the forEach() function is used. Its parameter holds the values that are passed to the function called in the loop. The loop then calls the function foo on each key, value pair.

```
myArray.forEach(function(value, key) {
  foo(key, value)
})

Output: cat: Jem
        kitten: Beans
        color: Yellow
```

Lua

1. **Initialize**

   Tables are the only data structure available in Lua, and are used to create arrays and dictionaries. They can be initialized in one of the following ways:

```
myArray = {}

myArray = {["name"] = "Maryam", ["cat"] =  "Jem", ["kitten"] =
"Beans", ["fruit"] = "Peach"}
```

   The first statement creates an empty associative array - myArray, and the second statement (used in my code) creates an associative array with the following key-value pairs respectively, name: Maryam, cat: Jem, kitten: Beans, fruit: Peach.

2. **Get the value for a given key**

Lua uses square brackets as its legal subscript and uses them to obtain the value associated to the given key.

```
myArray["name"]
```

The above statement will return the value of the key ‘name’ – which is ‘Maryam’. It can be printed to the console using the print statement as follows:

```
print(myArray["name"])
```

```
Output: Maryam
```

3. **Add a new element**

   The following statement will add the key ‘color’ with the value ‘Black’ to the associative array (in case the key is already present, its value will be modified).

   ```
   myArray["color"] = "Black"
   ```

4. **Remove an element**

   To remove an element in Lua, its value is set to ‘nil’ which is a global variable in the language, assigned to variables in order to delete them. The following code removes the key-value pair “fruit: Peach”.

   ```
   myArray["fruit"] = nil
   ```

5. **Modify the value of an existing element**

   An existing element’s value is modified the same way a new element is entered. The following statement changes the value of the key “color” from “Black” to “Purple”.

   ```
   myArray["color"] = "Purple"
   ```

6. **Search for the existence of a key**

   To search for the existence of key, Lua checks the element against nil (which means empty), and returns true if the element is not equal to nil using the ~= operator.

   ```
   myArray["color"] ~= nil
   ```

   The above statement will return true since ‘color’ is a key in the array. The output can be printed using the print statement as follows:

```
print(myArray["color"] ~= nil)

Output: true
```

7. **Search for the existence of a value**

   Since Lua does not provide a method to search for the existence of a value, it can be done using a for loop which allows an iterative traversal of the array. In the following code, I created a function namely checkValue which takes the value to search as its parameter and loops through the associative array. It prints "exists" if the value exists in the array.

```
function checkValue(valueToCheck)
  for key, value in pairs(myArray) do
    if value == valueToCheck then
      print(valueToCheck, "exists")
    end
  end
end

checkValue("Purple")
```

   In the following statement, I have called checkValue() upon "Pruple". It will return true since Jem is indeed a value in the array.

8. **Loop through an associative array, apply a function, called foo, which simply prints the key-value pair**

   The following code segment represents a function foo which simply prints its parameters – a key-value pair.

```
function foo(key, value)
  print (key, value)
end
```

   The generic for loop allows us to traverse all values returned by an iterator function. The pairs function in Lua is used to iterate over all key-value pairs in an associative array (table). The following code loops through the array and calls the function foo on each element. The function must be defined prior to its use.

```
for key, value in pairs(myArray) do
  foo(key, value)
end
```

```
Output: name    Maryam
        cat     Jem
        kitten  Beans
        color   Purple
```

PHP

1. Initialize

Variables in PHP are represented by a dollar sign, and arrays are initialized using the array() function. Associated arrays define elements in a “key” => “value” format. The following statement creates an associated array with the respective key-value pairs, name: Maryam, cat: Jem, kitten: Beans, fruit: Peach.

```
$myArray = array("name" => "Maryam", "cat" => "Jem", "kitten" =>
"Beans", "fruit" => "Peach");
```

2. Get the value for a given key

PHP uses square and curly brackets interchangeably to access array elements. For clarity, I have used square brackets throughout my code.

```
$myArray['name']
```

The above statement returns the value of the key “name” which is “Maryam.” It can be output using the echo statement as follows:

```
echo $myArray['name'];

Output: Maryam
```

3. Add a new element

The following statement will add the key “color” with the value “Purple” to the associative array (in case the key is already present, its value will be modified).

```
$myArray['colour'] = "Purple";
```

4. Remove an element

PHP provides the unset() function which deletes the specified element from an associative array. The following statement removes the key-value pair “name: Maryam” from the array.

```
unset($myArray['name']);
```

5. **Modify the value of an existing element**

   An existing element's value is modified the same way a new element is entered. The following statement changes the value of the key "color" from "Purple" to "Black".

   ```
   $myArray['color'] = "Black";
   ```

6. **Search for the existence of a key**

   PHP provides a function array_key_exists() which returns true if the given key is set in the array.

   ```
   array_key_exists('cat', $myArray)
   ```

   The above statement will return true since the key 'cat' is present in the array. The result can be output using the var_dump() function which displays information about a variable.

   ```
   var_dump(array_key_exists('cat', $myArray));
   ```

   ```
   Output: bool(true)
   ```

7. **Search for the existence of a value**

   PHP provides an in_array() function which returns true if a value exists in an array, and can be used for associative arrays as well.

   ```
   in_array("Jem", $myArray)
   ```

   The above statement will return true since the value 'Jem' is present in the array. Here again, the result can be output using the var_dump() function which displays information about a variable.

   ```
   var_dump(in_array("Jem", $myArray));
   ```

   ```
   Output: bool(true)
   ```

8. **Loop through an associative array, apply a function, called foo, which simply prints the key-value pair**

The following code segment shows a function foo which takes two variables – key and value (represented with a dollar sign) as parameters and displays their value using echo. echo
 is used to give a line break.

```
function foo($key, $value) {
  echo $key . ": " . $value;
  echo "<br>";
}
```

PHP provides a foreach() function which iterates over arrays. In the following code, it loops through the associative array and calls the foo function on each key => value pairs.

```
foreach($myArray as $key => $value) {
  echo foo($key, $value)
}
```

```
Output: cat Jem
        kitten Beans
        fruit Peach
        color Black
```

Python

1. Initialize

Associative arrays are called dictionaries in Python. The key-value pair is written as 'key': 'value'. The following statement creates an associative array (myArray) with key-value pairs as - age: 20, cat: Jem, kitten: Beans, fruit: Peach

```
myArray = {'age': 20, 'cat':  'Jem', 'kitten': 'Beans', 'fruit':
'Peach'}
```

2. Get the value for a given key

Since Python uses square brackets as legal subscript, they are used to access the value for a given key.

```
myArray['cat']
```

The resultant key can be output using the print statement as follows:

```
print(myArray['cat'])
```

```
Output: Jem
```

3. **Add a new element**

   The following statement will add the key “color” with the value “Purple” to the associative array (in case the key is already present, its value will be modified).

```
myArray['color'] = 'Purple'
```

4. **Remove an element**

   Python provides the del keyword to delete objects. Since everything is an object in python, it can be used to delete elements from an associative array. The following statement deletes the “age: 20” key-value pair.

```
del myArray['age']
```

5. **Modify the value of an existing element**

   An existing element’s value is modified the same way a new element is entered. The following statement changes the value of the key “color” from “Purple” to “Yellow”.

```
myArray['color'] = 'Yellow'
```

6. **Search for the existence of a key**
   Python provides the in keyword to check if a certain element is present in the list. This can be used to search for the existence of a key. It also has a keys() method which is specifically for keys in an associative array, and returns true if the key exists.

```
'cat' in myArray
'cat' in myArray.keys()
```

   The above two statements check if the key ‘cat’ exists in myArray. The result can be output using the print() method as follows:

```
print('cat' in myArray.keys())
```

```
Output: True
```

7. **Search for the existence of a value**

Similar to the previous method, python provides a values() method which returns true if a specific value exists in the array. The following statement tests if 'Jem' is a value in the array. Again, the result can be output using the print() method.

```
print('Jem' in myArray.values())

Output: True
```

8. **Loop through an associative array, apply a function, called foo, which simply prints the key-value pair**

Functions are declared using the def keyword in Python. The following statement creates a function foo which takes two values as its parameters – k, v. And simply prints their values.

```
def foo(k, v):
  print(k, v)
```

The following code shows a for loop being used to iterate over an associative array. The items() method allows access to the array's key-value pairs. The loop iteratively calls the function foo on each key-value pair.

```
for key, value in myArray.items():
  foo(key,value)

Output: cat Jem
        kitten Beans
        fruit Peach
        color Yellow
```

Ruby

1. **Initialize**

Ruby uses hashes for associative arrays, and the key-value pairs are defined as "key" => "value". The following statement creates an associative array with the key-value pairs as – name: Maryam, cat: Jem, kitten: Beans, fruit: Peach.

```
myArray = {'name' => 'Maryam', 'cat' => 'Jem', 'kitten' => 'Beans',
'fruit' => 'Peach'}
```

2. **Get the value for a given key**

Python uses square brackets as legal subscript which allows access to the array's elements. The following statement retrieves the value of the given key – name. The result can be output using the keyword puts which prints the value of the key. In this case, Maryam.

```
puts myArray['name']
```

3. **Add a new element**

The following statement will add the key "color" with the value "Blue" to the associative array (in case the key is already present, its value will be modified).

```
myArray['color'] = 'Blue'
```

4. **Remove an element**

Ruby has an inbuilt method delete() which deletes the given object and returns it. It can be used to remove an element from as associative array. The following statement removes the key-pair "name: Maryam" from the array.

```
myArray.delete('name')
```

5. **Modify the value of an existing element**

An existing element's value is modified the same way a new element is entered. The following statement changes the value of the key "color" from "Blue" to "Yellow".

```
myArray['colour'] = 'Yellow'
```

6. **Search for the existence of a key**

Ruby consists of a method key?() which returns true if the given key is present in the associative array. In the following code, the method will return true since 'cat' is indeed a key in the array. The result can be printed using the keyword puts.

```
puts myArray.key?('cat')
```

```
Output: true
```

7. **Search for the existence of a value**

Similarly, Ruby has a method value?() which returns true if the given value is present in the associative array. In the following code, the method will return true since 'Jem' is a value to the key 'cat' in the array. The result can be printed using the keyword puts.

```
puts myArray.value?('Jem')

Output: true
```

8. **Loop through an associative array, apply a function, called foo, which simply prints the key-value pair**

The following code segment creates a function foo using the def keyword. Foo has two parameters – key and value, and simply prints their values when it is called.

```
def foo(key, value)
  puts "#{key}: #{value}"
end
```

The following statement consists of a loop each{} which iterates through the array elements and calls the function foo for each key-value pair and passes their values as parameters.

```
myArray.each{|key, value| foo(key, value)}

Output: cat: Jem
        kitten: Beans
        fruit: Peach
        color: Yellow
```

Rust

1. **Initialize**

Rust does not support associative arrays in its standard library. However, the class `std::collections::HashMap` can be used to create associative arrays (maps). The keyword `let` allows us to introduce new variables and the keyword `mut` lets us declare a mutable variable whose type does not need to be declared.

```
let mut my_array = HashMap::new();
```

Unlike other languages, elements must be entered separately in the associative array using the insert("key", "value") method as follows:

```
my_array.insert("cat", "Jem");
```

```
my_array.insert("kitten", "Beans");
my_array.insert("name", "Maryam");
```

2. **Get the value for a given key**

   Since Rust uses square brackets as legal subscript, they can be used to obtain the value for a given key. The value can be printed using the println!{} function which uses the following format:

   ```
   println!("{}", my_array["name"]);
   ```

3. **Add a new element**

   Rust provides the method insert("key", "value") which adds a new element to the associative array (in case the key is already present, its value will be modified). The following statement adds the key "color" with the value "yellow" to the array.

   ```
   my_array.insert("color", "Yellow");
   ```

4. **Remove an element**

   Rust has the method remove() which deletes the given element from the associative array. The following statement removes the key-value pair "name: Maryam".

   ```
   my_array.remove("name");
   ```

5. **Modify the value of an existing element**

   An existing element's value is modified the same way a new element is entered. The following statement changes the value of the key "color" from "Yellow" to "Black".

   ```
   my_array.insert("color", "Black");
   ```

6. **Search for the existence of a key**

   Rust provides the function contains_key() which returns true if the given key exists in the array. In order to print the output, the println!() function can be used. The following statement outputs true since the key "color" exists in the array.

   ```
   println!("{}", my_array.contains_key("color"));

   Output: true
   ```

7. **Search for the existence of a value**

   Rust allows us to create a predefined variable does_contain which checks through all values present in an associative array using the values() method and tries to match it against the given value ("Jem" in the following code). The boolean result of does_contain can be printed using the println!{} statement as follows:

```
let does_contain = my_array.values().any(|&val| val == "Jem");
println!("{}", does_contain);

Output: true
```

8. **Loop through an associative array, apply a function, called foo, which simply prints the key-value pair**

   The following code segment consists of a for loop that traverses through my_array and calls the function foo (defined later) on each of its key-value pairs.

```
for (key, value) in &my_array {
  foo(key, value);
}
```

   Rust is versatile when it comes to functions as it allows us to declare the function after (or before) it is called. The following function foo takes two string parameters (represented by &str) – key and value, and prints their value using the println!{} function in the following way:

```
fn foo(key: &str, value: &str) {
  println!("{}: \"{}\"", key, value);
}

Output: kitten: "Beans"
        cat: "Jem"
        color: "Black"
```

Evaluation of Languages

1. Dart

Dart is rather easy to read and write for a first-time user like myself. It allows initialization of the associative array in its declaration which makes it easier to write the code and thus enhances its readability. It also follows a generic pattern for the initialization of the key-value pairs such as “key:value”, this favors both readability and writability of the code. However, it requires a var keyword which infers the type of the variable. Some may say this reduces readability since multiple different types with similar names can be initialized with var, and the type will be unknown to the reader without further reading. Retrieving, adding and modifying an element all make use of square brackets, which allows an easy access to the array’s elements. This paves way for a more readable code. Removing an element, searching for the existence of a key or value and traversing through an array using a loop, all make use of the built-in methods of the language. This greatly enhances the writability of the code. The semi-colons at the end of each statement clearly mark the ending of a statement which makes the code more readable. Dart requires a main() function for each code which allows the code to be more compact and easy to read and follow.

2. JavaScript

Javascript was one of the complex languages in the assignment. Although it allowed initialization of the associative array within its declaration, it had to create an object of Map. This is because associative arrays are not supported by Javascript directly. This may hinder its writability. Although Javascript has square brackets as its legal subscripts, the Map class does not use them since it uses a set of primitive functions. This further affects the readability of the language. The map class consists of method that are used to access, add, modify, delete and search for the existence of the associative array’s elements. This consistency adds to the readability of the language. The loops of the language, mainly for, are rather unique since they require functions as its conditions, this reduces the readability of the code. The language does not require the type of function to be declared which adds to the writability of the code. It also used {} brackets to mark the beginning and ending of a function which further strengthens its readability.

3. Lua

Lua is a fairly easy to follow language. It is quite consistent with the general syntax of languages in terms of associative arrays. It allows initialization of the array within its declaration thus making the code easier to read and write. It follows a concise format of defining a key-value pair of “[key] = value”. This makes it easier to distinguish between the

key and value element of the array thus increasing readability. However, using a pair of square brackets for initialization of keys may impact the writability and readability of the code as it can be confused with the legal subscript for accessing an array's elements. Likewise, retrieving a value for a given key, adding a new element, removing an element or modifying an existing element, all make use of square brackets to access the element and perform the respective function, and do not require any additional methods. This greatly increases the readability of the code as it makes it easier for the programmer to follow. Moreover, even searching for the existence keys and values in the array does not require methods. It is done simply by relational operations. Although this increases the writability, it may lead to confusions amongst the reader between accessing array elements and checking for their existence. Iterating through an array is done simply by a for loop and a function is created by the function keyword. Both of these increase the readability and writability of the code.

4. **PHP**

PHP is relatively different from the other languages. This is mainly due to the fact that it is used in web development. It allows initialization of the array within the declaration which adds to its readability and writability. It follows an unconventional format of initializing key-value pairs as "key => value", the => marks a clear distinction between the key and value elements thus adding to its readability. It makes a distinction between variables and other values by using $ to initialize variables. Retrieving, adding and modifying an element all make use of square brackets to access the elements which adds to the readability of the code. Deleting, searching for the existence of key or value and traversing the array utilizes the primitive functions of the language which enhances its writability.

5. **Python**

Python is one of the most straightforward languages. It allows initialization of associative arrays within its declaration and follows the general format for initialization of key-value pairs as "key: value", this makes it easier to write and read associative arrays. It uses square brackets to access array elements for addition and modification which adds to its readability. Python uses general keywords such as "del" for deletion, "in" for checking if the given key/ value is present in the array, and also for traversing. These common words greatly increase the language's readability and make the program clear for even a first-time user. The language also does not declare the function type in its definition which makes it more writable for the programmer. Iterating through an array is done simply by a for loop and a function is created by the function keyword. Both of these increase the readability and writability of the code.

6. **Ruby**

Ruby follows the general pattern as most languages. It allows initialization within declaration which adds to its readability and writability. It follows an unconventional format of initializing key-value pairs as “key => value”, the => marks a clear distinction between the key and value elements thus adding to its readability. It uses square brackets to access array elements for addition and modification which further adds to its readability and writability. It uses primitive functions such as key?, value? for searching the existence of a key and value. The self-explanatory function names greatly enhance its readability. However, the format for loops was somewhat complex and not generic, this can be said to decrease the code’s writability. It also does not use {} brackets to mark the beginning and end of a function which can further diminishes its readability.

7. **Rust**

I found Rust to be the most complex language of all. Since it does not support associative arrays in its standard library, they need to be used via different collections and tree types. It only allows variables and functions to use snake case names which feels restricting to the user. Rust does not allow initialization within declaration and the two must be done separately. This is because first it needs to specify the collection being used (HashMap in my case), and then insert the elements using the built-in function. This makes the code more complex, thus decreases writability and readability of the code. It does not allow access to the array elements via square brackets in most cases thus insertion, deletion and modification must be done using primitive functions provided by the language. There are also similar functions for searching for the existence of keys and values. This consistency adds to the readability of the language. Loops follow the general pattern in Rust. However, unlike most declarations, functions can be declared even after they have been called. This negatively impacts the readability of the code.

My Choice:

Although all languages had a fairly similar syntax for associative arrays (especially Python, Ruby and Dart as they followed similar commands), I found Python to be the best. As stated above, it follows a straightforward patter to initialize and manipulate associative arrays. Its keywords for checking and traversing through an array consists of words from our everyday language which brings this high-level language very close to the language we use to communicate with each other and adds to its readability and writability. My preference for Python may also be the result of my bias towards the language as it was the only language that I had used extensively prior to the assignment. Along with the extensive readability and writability, it provides a degree of flexibility that I did not observe in the other languages that I investigated thus making it my go-to language.

Learning Strategy

Since most of the programming languages in the assignment were new to me, my first strategy was to view their official documentation. In order to familiarize myself with the language's basic syntax, I used online compilers for each and played around with them. Once I was comfortable with the languages, I began working on the task at hand. I went through the associative arrays' section in the official documentation of each language, and referred to Tutorials-point and StackOverFlow in case I were unable to find enough information. Below I have documented my research and strategy for each of the languages.

1. **Dart**

   To get started with the language, I went through Dart's official website and found a valuable tour of the language at https://dart.dev/guides/language/language-tour. I used the online compiler provided by Dart's official website to run my programs which can be found at https://dartpad.dev. Since I had difficulty finding some methods for arrays in the language, Tutorials-point proved to be a good resource: https://www.tutorialspoint.com/dart_programming/index.htm.

2. **JavaScript**

   Prior to the assignment, I knew that Javascript was one of the languages in web development. However, I did not know more than that. I began studying the language through the documentation provided at Mozilla's website and focused on associative arrays (Maps) (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map). I also found the following tutorial at w3schools to be quite useful during my research: https://www.w3schools.com/js/js_arrays.asp. I used the online compiler provided at js.do (https://js.do) to test my codes.

3. **Lua**

   Since I had never heard of Lua prior to the assignment, it took me a while to familiarize myself with the language. However, the manual provided by Lua's official website proved to be of help https://www.lua.org/pil/. To get information about associative arrays, or tables as they are known in Lua, I used the following link, https://www.tutorialspoint.com/lua/lua_tables.htm. Finally. I used the online compiler provided by repl.it to run my programs: https://repl.it/languages/lua.

4. **PHP**

   I knew PHP was a part of languages for web development, however I lacked knowledge beyond that. To get started with the language, I used the manual provided by PHP's official website to understand the language (https://www.php.net/manual/en/). I found a useful tutorial about associative arrays in PHP at w3schoolswhich further helped me understand the task, https://www.w3schools.com/php/php_arrays_associative.asp. Finally, I used the online compiler at https://www.writephponline.com to test my codes.

5. **Python**

   Since I had used Python before, it was relatively easier for me to attempt the task. I also already had Python installed on my computer. However, I lacked knowledge about associative arrays in the language. I went through the official documentation of the language and focused on the dictionaries' data structure: https://docs.python.org/3/tutorial/datastructures.html#dictionaries. I also found the following python course which helped me further strengthen my knowledge about associative arrays in the language: https://www.python-course.eu/python3_dictionaries.php.

6. **Ruby**

   I had little knowledge about Ruby prior to the assignment and consequently struggled in it. I used a website called rubylearning to understand the languages where I learnt that associative arrays are equivalent to hashes in the language and further researched on them at: http://rubylearning.com/satishtalim/ruby_hashes.html. I also referred to tutorialspoint for other complications in the language at: https://www.tutorialspoint.com/ruby/ruby_loops.htm. I used the online compiler provided by repl.it to run my codes at: https://repl.it/languages/ruby.

7. **Rust**

   Out of all the languages, I found Rust the hardest to deal with. Since associative arrays were not a part of its standard library, I had to research its collections. I used Rust's official documentation to understand the language. There were a series of examples "Rust by example" which helped me get used to the language. They can be found at https://doc.rust-lang.org/stable/rust-by-example/. I used the information provided about the HashMap collection to learn its implementation: https://doc.rust-lang.org/std/collections/struct.HashMap.html. I used the compiler provided by Rust's official website to test my codes: https://play.rust-lang.org.