Bilkent University

CS 315 Programming Languages Homework #1

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

Barış Tan Ünal | 22003617

18.11.2022

Table of Contents

1.0 CODE SEGMENTS AND THE RESULTS OF EXECUTION

1.1 DART

1.1.1 CODE SEGMENT

```
void main() {

  //1. Initialize

  Map grades = {
    'CS202':   'B-',
    'CS224':   'B+',
    'PHYS102': 'C-',
    'HUM112':  'A-',
    'SOC101':  'A',
    'ENG401':  'F'
  };
  print("\nInitiliazed grades:");
  grades.forEach((course, grade) {
    print("$course: $grade");
  });

  //2. Get the value for a given key
  print( "The grade of CS202 is: " + grades["CS202"] );

  //3. Add a new element
  grades[ "PHYS252" ] = "D+";
  print("\nAdded the grade of PHYS252:");
  grades.forEach((course, grade) {
    print("$course: $grade");
  });

  //4. Remove an element
  grades.remove( "HUM112" );
  print("\nDeleted the grade of HUM112:");
  grades.forEach((course, grade) {
    print("$course: $grade");
  });

  //5. Modify the value of an existing element
  grades[ "CS202" ] = "C+";
  print("\nModified the grade of CS202:");
  grades.forEach((course, grade) {
    print("$course: $grade");
```

```
  });

  //6. Search for the existence of a key
  print( "\nAvailability of the grade of ENG401: " +
grades.containsKey( "ENG401" ).toString() );
  print( "Availability of the grade of MATH225: " +
grades.containsKey( "MATH225" ).toString() );

  //7. Search for the existence of a value
  print( "\nAvailability of a lesson with 'F' grade: " +
grades.containsValue( "F" ).toString() );
  print( "Availability of a lesson with 'FZ' grade: " +
grades.containsValue( "FZ" ).toString() );

  //8. Loop through an associative array, apply a function, called
foo, which simply prints the key-value pair
  foo( array ) {
    array.forEach((course, grade) {
      print("$course: $grade");
    });
  }
  print( "\nCalling foo..." );
  foo( grades );

}
```

1.1.2 EXPLANATION

In Dart programming language, associative arrays are called “Maps” and their initialization is done by using “Map” keyword and the variable name following it. I have initialized a Map for grades of a student. I have given the letter grades for six courses in the grades Map. After each time I did an operation to the Map, I have printed the key – value pairs by iterating through the elements one by one. Dart has a built-in function for iterating through an array called “foreach()”. This printing operation will not be mentioned from now on to avoid repetition.

Then I have retrieved the grade of CS202. In Dart, it is pretty easy to access a value through its key which is done by the structure “array[key]”. There is no need for iteration.

Adding a new element to a Map is also simple. Although there is no built-in function to do that, there is no need for it. The structure “array[key] = value” does the trick.

Removing an element from a Map is not a big deal neither. Dart has a built-in function for it which is called “remove()”. The structure is as follows: array.remove(key).

Modifying the value of an element is again straightforward. Just as adding a new element, I have used the structure “array[key] = value”. This operation overwrites the value of a given key if it already exists in the Map, which is the case for my example of CS202.

Dart has a built-in function for searching for a key in a Map called “containsKey()” which returns a boolean value. I have searched for courses “ENG401” and “MATH225” in the grades Map and printed them as string by using “toString()” function.

There is again a built-in function for searching for a value in a Map in Dart which is called “containsValue()”. This function returns a boolean value just as the previous one. I did the exact same operation as I did for part 6.

As I mentioned above and used each time after I executed an operation, Dart has a built-in function for iterating through a Map. I declared a function called “foo”. In Dart, functions are declared without the return type and a specific keyword such as “function”. “Foo” does the iteration inside for the given Map which is passed as a parameter and prints each element it iterates.

1.2 JAVASCRIPT

1.2.1 CODE SEGMENT

```
// 1. Initialize.
var jsArr = {
  "key1": 'firstKey',
```

```
  "key2": 2,
  "key3": null,
  "key4": 44.4,
  "key5": true,
  "key6": [ 'subArr1', 'subArr2', 'subArr3' ]
};

// 2. Get the value for a given key.
console.log(jsArr['key2']);

// 3. Add a new element.
jsArr["key7"] = 'javaScript';
console.log("\nAdded key7:");
console.log(jsArr);

// 4. Remove an element.
delete jsArr["key2"];
console.log("\nRemoved key2:");
console.log( jsArr);

// 5. Modify the value of an existing element.
jsArr["key3"] = 'thirdkey';
console.log("\nModified key3:");
console.log(jsArr);

// 6. Search for the existence of a key
console.log("\nChecking for the existence of key1: ");
console.log("key1" in jsArr);
console.log("\nChecking for the existence of key2: ");
console.log("key2" in jsArr);

// 7. Search for the existence of a value
console.log("\nChecking for the existence of value 'null': ");
console.log( Object.values(jsArr).includes( null ) );
console.log("\nChecking for the existence of value '44.4': ");
console.log( Object.values(jsArr).includes( 44.4 ) );

// 8. Loop through an associative array, apply a function, called foo,
which simply prints the key-value pair
function foo( arr ) {
  for( const key in arr ) {
    console.log( key, arr[key] );
  }
}
console.log("\nCalling foo: ");
foo( jsArr );
```

1.2.2 EXPLANATION

In JavaScript programming language, one can initialize an array with key-value pairs straightaway. The initialization is very similar to Dart. The only difference is that there is no specific type for associative arrays such as “Map”.

Retrieving a value for a given key, adding a new key-value pair and modifying the value of an existing key are exactly the same with Dart. I have printed the array by just giving the associative array as a parameter to “print” function. JavaScript has a default printing structure for arrays therefore does not need iteration through its elements like Dart.

Removing an element is a bit different than Dart. The “delete” keyword is used for that. However, this operation does not remove the key – value pair, instead it sets the value of the given key to “undefined”.

Searching for the existence of a key is simple in JavaScript. Although there are several ways to do that, I have used “in” keyword whose structure is “key in array”. This operation returns a boolean value which states whether that key exists in the given array or not.

Searching for the existence of a value in an associative array is a bit trickier but there is a built-in function for that, too. The structure is “Object.values(array).includes(value)”, which first parses the values of the associative array and then checks if the array includes the given value. “Includes” function returns boolean value. I have printed those returned boolean values to indicate the values’ existence.

The “foo” function simply has a for loop which iterates through the keys of the associative array and prints the keys that it iterates and their related values which were accessed by “array[key]” structure. Calling a function in JavaScript is exactly the same as it is in Dart.

1.3 LUA

1.3.1 CODE SEGMENT

```
-- 1. Initialize
print "\n";
movieRatings = {};
movieRatings[ "Pulp Fiction" ] = 8;
movieRatings[ "The Green Mile" ] = 2;
movieRatings[ "Top Gun" ] = 7;
for movie, rating in pairs( movieRatings ) do
  print( movie, ": ", rating );
end

-- 2. Get the value for a given key
print "\n";
print( "- Rating of Top Gun: ", movieRatings[ "Top Gun" ] );

-- 3. Add a new element
print "\n";
movieRatings[ "Rear Window" ] = 10;
for movie, rating in pairs( movieRatings ) do
  print( movie, ": ", rating );
end

-- 4. Remove an element
print "\n";
movieRatings[ "Pulp Fiction" ] = nil;
for movie, rating in pairs( movieRatings ) do
  print( movie, ": ", rating );
end

-- 5. Modify the value of an existing element
print "\n";
movieRatings[ "The Green Mile" ] = 1;
for movie, rating in pairs( movieRatings ) do
  print( movie, ": ", rating );
end

-- 6. Search for the existence of a key
print "\n";
function keyExists( table, key )
  for movie, rating in pairs( table ) do
      if movie == key then return true; end
    end
```

```
  return false
end
print ( "Existence of Pulp Fiction: ", keyExists( movieRatings, "Pulp
Fiction" ) );
print ( "Existence of Rear Window: ",keyExists( movieRatings, "Rear
Window" ) );

-- 7. Search for the existence of a value
print "\n";
function valueExists( table, value )
  for movie, rating in pairs( table ) do
      if rating == value then return true; end
    end
  return false
end
print ( "Existence of a movie with 10/10 rating: ", valueExists(
movieRatings, 10 ) );
print ( "Existence of a movie with 5/10 rating: ", valueExists(
movieRatings, 5 ) );

-- 8. Loop through an associative array, apply a function, called foo,
which simply prints the key-value pair
print "\n";
function foo( table )
    for movie, rating in pairs( table ) do
      print( movie, ": ", rating );
    end
end
foo( movieRatings );
```

1.3.2 EXPLANATION

In Lua, associative arrays are a type of “Table” which can be either traditional array, associative array, or a record. Curly braces are used for initialization. I have added several key – value pairs which are movies and their personal ratings out of 10 after initialization in order to access them in part 2. Also, I have printed the array each time I performed an operation for tracing purposes. Printing is done by iterating the pairs with “for key, value in pairs(array)” structure.

Accessing an existing value by its key, adding a new element, and modifying an element are exactly the same with Dart and JavaScript which was mentioned above.

Removing an element is similar to JavaScript. Again, we cannot remove the key – value pair completely but instead we set the value of that key to “nil”.

Lua does not have a built-in function for inspecting the existence of a key nor value, so I have implemented two functions for that use. Both of them has the structure and passes an associative array and a key or a value as parameters. Then, they iterate through the key – value pairs and checks whether the current value or key equals to the value or key that we are searching for. It returns true if they are the same right away and it returns false when the iteration is completed but there is no match.

Lastly, functions are declared by using the “function” keyword followed by the function identifier and parameter list. The “foo” function does the same iterations as the “keyExists” and “valueExists” functions but only prints the key – value pairs in its body.

1.4 PHP

1.4.1 CODE SEGMENT

```
<?php

  echo "Hello, World!";

  // 1. Initialize
  $capitals = array(
    "Brazil"=>"Brasilia",
    "Russia"=>"Moscow",
    "Thailand"=>"Bangkok",
    "Cuba"=>"Havana",
    "India"=>"New Delhi",
    "Malaysia"=>"Putrajaya"
  );
  print_r( "\nInitialized 'capitals':\n" );
  print_r( $capitals );

  // 2. Get the value for a given key
  print_r( "\nCapital of Cuba is: " );
```

```
print_r( $capitals[ "Cuba" ] );

// 3. Add a new element
$capitals += array("Argentina" => "Buenos Aires");
print_r( "\n\nAdded Argentina:\n" );
print_r( $capitals );

// 4. Remove an element
unset( $capitals["Brazil"] );
print_r( "\nRemoved Brazil:\n" );
print_r( $capitals );

// 5. Modify the value of an existing element
$capitals[ 'Malaysia' ] = "Kuala Lumpur";
print_r( "\nUpdated Malaysia:\n" );
print_r( $capitals );

// 6. Search for the existence of a key
if( array_key_exists( "Russia", $capitals ) ) {
  print_r( "\nRussia exists." );
}
else {
  print_r( "\nRussia does not exist." );
}

if( array_key_exists( "USA", $capitals ) ) {
  print_r( "\USA exists." );
}
else {
  print_r( "\nUSA does not exist." );
}

// 7. Search for the existence of a value
if( array_search( "Kuala Lumpur", $capitals ) ) {
  print_r( "\nKuala Lumpur exists." );
}
else {
  print_r( "\nKuala Lumpur does not exist." );
}

if( array_search( "Budapest", $capitals ) ) {
  print_r( "\nBudapest exists." );
}
else {
  print_r( "\nBudapest does not exist." );
}
```

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

  function foo( $array ) {
    foreach( $array as $key => $value ) {
      echo $key . ": " . $value . "\n";
    }
  }
  print_r( "\n\nCalling foo...\n" );
  foo( $capitals );

?>
```

1.4.2 EXPLANATION

PHP's arrays can be treated as either a traditional array or an associative array by using "Array" keyword. Variable names are stated with a dollar sign followed by alphanumeric characters. In this code segment, I have implemented an array called "capitals" where the keys are country names, and their associated values are the capitals of those countries. While stating the key – value pair during initialization and addition, an arrow (=>) is used in between them.

Accessing a value through its key is pretty straightforward and the same with Dart and JavaScript. Printing the array is also the same with JavaScript where we only pass the identifier of the array to the print function without explicitly iterating the array.

Adding a new element to an associative array is a bit different than the others since it introduces a distinct operator, "+=". That operator concatenates two arrays. In the code segment, I have declared a new array with only one element and concatenated it with the existing "capitals" array and assigned it to "capitals" associative array.

PHP has a built-in function for removing an element from an associative array called "unset()". I have passed one of the existing keys to that function to remove it from the associative array.

Modifying the value of a key has the same logic and structure with JavaScript where it overwrites the value of a key since that key already exists in the array.

PHP has built-in functions for both searching for the existence of a key and searching for the existence of a value in an associative array called “array_key_exists” and “array_search”. They both return boolean values according to the result. I have used those boolean values in an if – else block to print that that country or capital exists or not.

Finally, in the “foo” function I have used the “function” keyword to declare it. Inside the function, I have used the “foreach” loop structure to explicitly iterate the array where I declare key – value pairs and print them as it iterates the keys of the associative array. Calling “foo” is the same with JavaScript and others by specifying the identifier of the function and passing the associative array as the only parameter.

1.5 PYTHON

1.4.1 CODE SEGMENT

```
# 1. Initialize
print( "\n\n", "1. Initialize" );
books = {
  "Frankenstein": { "author": "Mary Shelley", "availability": True },
  "Utopia": { "author": "Thomas More", "availability": False },
  "The Illiad": { "author": "Homer", "availability": True },
  "The Tempest": { "author": "Shakespeare", "availability": False }
}
for x in books:
    print( "\n", x )
    for y in books[x]:
        print( y, ":" ,books[x][y] )

# 2. Get the value for a given key
print( "\n\n", "2. Get the value for a given key" );
print( books[ "Utopia" ] );

# 3. Add a new element
```

```
print( "\n\n", "3. Add a new element" );
books[ "The Strange Case of Dr. Jekyll and Mr. Hyde" ] = { "author":
"R. L. Stevenson", "availability": False  }
for x in books:
    print( "\n", x )
    for y in books[x]:
        print( y, ":" ,books[x][y] )

# 4. Remove an element
print( "\n\n", "4. Remove an element" );
del books[ "The Tempest" ];
for x in books:
    print( "\n", x )
    for y in books[x]:
        print( y, ":" ,books[x][y] )

# 5. Modify the value of an existing element
print( "\n\n", "5. Modify the value of an existing element" );
books[ "Frankenstein" ] = { "author": "Mary Shelley", "availability":
False };
for x in books:
    print( "\n", x )
    for y in books[x]:
        print( y, ":" ,books[x][y] )

# 6. Search for the existence of a key
print( "\n\n", "6. Search for the existence of a key" );
def checkKey( dict, key ):
    if key in dict.keys():
        print( key, " exists.")
    else:
        print( key, " does not exist.")
checkKey( books, "1984" )
checkKey( books, "Frankenstein" )

# 7. Search for the existence of a value
print( "\n\n", "7. Search for the existence of a value" )
def checkValue( dict, value ):
    if value in dict.values():
        print( value, " exists.")
    else:
        print( value, " does not exist.")
checkValue( books, { "author": "Mary Shelley", "availability": True }
)
checkValue( books, { "author": "Homer", "availability": True }, )
```

```
# 8. Loop through an associative array, apply a function, called foo,
which simply prints the key-value pair
print( "\n\n", "8. Loop through an associative array, apply a
function, called foo, which simply prints the key-value pair" )
def foo( dict ):
  for i in dict:
    print( "\n", i )
    for j in dict[i]:
        print( j, ":" ,dict[i][j] )

foo( books )
```

1.5.2 EXPLANATION

In Python, associative arrays are called dictionaries. The values are all references to objects in dictionaries. Dictionary initialization has the exact same structure to JavaScript's associative arrays. In my code segment, I have initialized the dictionary's keys as books which have values of arrays consisting of author and availability properties. Therefore, the values are all references to arrays with size of two in this example. Printing a dictionary is pretty simple. I have used nested for loops which first accesses the keys and then the values inside them. It was the best option since I had two properties for values instead of one.

Getting a value for a given key, adding a new element, and modifying an existing element are all the same in Python with JavaScript and others. Modifying an element has the overwriting an existing key logic. If given key does not exist in the dictionary, it is added to it as a new element.

Removing an element is also straightforward with the keyword "del" followed by the dictionary identifier and the key to be deleted in brackets.

For checking if a key or a value exists in a dictionary, I have declared two separate functions which are called "checkKey" and "checkValue". The first function uses the "in" operator alongside "keys()" function which returns the keys of the dictionary in an array. The "in" operator returns a boolean value depending on the existence of the given key in that array. The second function does exactly the same except that it uses "values()" function which returns the values of the

dictionary. With a simple if – else block, this example prints whether a book exists in the dictionary or not.

Declaration of a function in Python is done with “def” keyword followed by the function identifier and arguments list. I have implemented a simple nested loop as I did while printing the dictionary each time, I performed an operation in the dictionary. Calling the function “foo” is again the same with the other languages.

1.6 RUBY

1.6.1 CODE SEGMENT

```
# 1. Initialize
fastestLaps = Array.new;

fastestLaps = { 'Deniz' => 46.5, 'Bengi' => 43.9, 'Arda' => 47.1,
'Enes' => 48.6 };
puts fastestLaps;

# 2. Get the value for a given key
puts "Bengi's fastest lap is " + fastestLaps['Bengi'].to_s();

# 3. Add a new element
fastestLaps['Almila'] = 39.5;

# 4. Remove an element
fastestLaps.delete( 'Deniz' );
puts fastestLaps;

# 5. Modify the value of an existing element
fastestLaps['Arda'] = 45.2;
puts fastestLaps;

# 6. Search for the existence of a key
puts fastestLaps.include?( 'Almila' );
puts fastestLaps.include?( 'Bora' );

# 7. Search for the existence of a value
puts fastestLaps.has_value?( 45.2 );
```

```
puts fastestLaps.has_value?( 30.7 );

# 8. Loop through an associative array, apply a function, called foo,
which simply prints the key-value pair
def foo( laps )
  laps.each do |x|
    puts x;
  end
end
  foo (fastestLaps);
```

1.6.2 EXPLANATION

In Ruby, I have used the “Array.new” statement for declaration of my associative array which consists of fastest laps from a race of competitors. To initialize it, I have stated the keys and their values with arrows (=>) in between where elements are separated by commas just as in PHP. Passing the array identifier as the only parameter to the “puts” function is enough to print the entire associative array in Ruby.

Getting a value of a given key, adding a new element, and modifying an existing value of a key have the same logic and the structure as JavaScript and others where it overwrites the value if given key exists but creates a new key – value pair if it does not exist. I have used “to_s()” function to make the types match inside the “puts” function.

Removing a value is simple with the built-in function “delete()”. I have only passed the key that I want to remove as the parameter.

In Ruby, checking for a key and a value is also pretty easy with its built-in functions “include?()” for keys and “has_value?()” for values. I have passed the key and the value that I want to check the existence of to those function. Those two functions both return a boolean value.

While declaring the function “foo”, I have used the keyword “def” followed by the function identifier and arguments list just like Python. Inside its body, a simple combination of “.each” and “|x|” operators worked for iteration through the keys of the associative array. I have printed

each “x” value inside the loop body. Calling the function is again the same with the other programming languages.

1.7 RUST

1.7.1 CODE SEGMENT

```
use std::collections::HashMap;

fn main() {
    println!("Euroleage Players Associative Array");

    // 1. Initialize
    let mut euroleague_players = HashMap::new();

    euroleague_players.insert(
        "Marko Guduric", "Fenerbahce"
    );
    euroleague_players.insert(
        "Mike James", "Monaco"
    );
    euroleague_players.insert(
        "Walter Tavares", "Real Madrid"
    );

    // 2. Get the value for a given key
    println!("Marko Guduric plays for: {}", euroleague_players["Marko
Guduric"]);

    // 3. Add a new element
    euroleague_players.insert( "Will Clyburn", "CSKA Moscow" );

    // 4. Remove an element
    euroleague_players.remove( "Walter Tavares" );

     for (key, value) in &euroleague_players {
        println!( "{}: {}", key, value );
    }

    // 5. Modify the value of an existing element
    euroleague_players.insert( "Will Clyburn", "Anadolu Efes" );
```

```
    for (key, value) in &euroleague_players {
        println!( "{}: {}", key, value );
    }

    // 6. Search for the existence of a key
    if euroleague_players.contains_key("Mike James") {
        println!("Mike James plays for a team in Euroleague.");
    }

    // 7. Search for the existence of a value
    let team_to_find = "Fenerbahce";
    for (_player, team) in &euroleague_players {
        if *team == team_to_find {
            println!("There is a player that plays for Fenerbahce.");
        }
    }


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

    fn foo ( arr: &mut HashMap<&str, &str> ) {
        for ( player, team ) in arr {
            println!( "{}: {}", player, team );
        }
    }

    foo ( &mut euroleague_players);

}
```

1.7.2 EXPLANATION

In Rust, unlike the other programming languages, I needed to import a “HashMap” library since associative arrays are called hash maps in Rust. I have initialized the hash map of Euroleague Basketball players with “let” and “mut” keywords along with “HashMap::new()” statement. This statement creates a new HashMap object in the heap. Only after then I have inserted the first elements to my HashMap by “insert()” function with key – value pairs inside it as parameters.

Accessing an element is exactly the same as JavaScript and others which uses the bracket operator next to the associative array identifier with a key as a parameter.

Adding a new element is simple but slightly different since there is a built-in function called “insert()” which takes the key – value pairs separated by a comma as the parameters. There is also another built-in function for removing an element which is called “remove()” with a key as parameter. I have printed the entire associative array after these operations by a simple for loop which is used with the “in” operator which specifies the array that we will iterate.

Rust has a built-in function for checking for a key in an associative array called “contains_key()” which returns a boolean value. I have checked for the key “Mike James” in my array and used its returned value in an if block whose body prints "Mike James plays for a team in Euroleague.", if the condition is true.

However, searching for a value is not that simple in Rust since we do not have a built-in function. I have implemented a for loop that iterates through the “euroleague_players” array and parses the key – value pairs. In its body, I have a conditional statement which checks whether the current value (team) is the value that we are searching for. If that condition holds, I indicate that there is player that plays for the team we are searching for with a “println!” statement. I have put a underscore just before the player (key) in the for loop in order to let the compiler know that I intentionally ignored the key in the body.

Function declarations in Rust use the keyword “fn” followed by function identifier and arguments list. I have stated that I will pass a HashMap which is mutable and consists of a key and a value which are both strings. Then, inside the “foo” function I have a simple for loop as I had previously which only has one print statement in its body. Calling a function is again very similar to other programming languages but additionally I only stated that “euroleague_players” array is mutable.

2.0 EVALUATION

Although creating an associative array in Dart, JavaScript, Lua, PHP, Python, Ruby and Rust is possible, they have different names and structures. Some of them supports associative arrays directly whereas some of them supports them with standard class libraries. One has no chance but to implement a function themselves to do some operations from given eight fundamental parts, while several of the languages have built-in functions to perform those operations.

The “Map” object in Dart makes everything smoother than it can ever be with built-in functions for removing elements, checking the existence of keys and values and array iterations. I had no need to implement a function by myself while implementing an associative array in Dart. I can easily conclude that Dart is both writable and readable in terms of associative array implementation thanks to its built-in functionalities.

Associative arrays in JavaScript are more straightforward with no specific object type. Adding and modifying elements are in the standard form and can be implemented with bracket operators. There is a reserved word for deletion which is both readable and writable. Although there are several ways to check the existence of a key in an array which have varying efficiencies, I have chosen the most readable and writable one. But this “in” operator can give us a wrong boolean value when there is a key with a value undefined intentionally. Unlike checking for a key, checking for a value in an associative array requires multiple built-in functions to implement. This slightly decreases the readability of the program. Iterations through associative arrays are pretty simple if not the simplest in JavaScript which clearly states the key - value pairs. To conclude, JavaScript is a bit less readable compared to Dart but not much different in terms of writability.

Lua has no built-in functions for associative arrays at all. Tables in Lua could be powerful, but it is reasonably more difficult to check for existence of keys and values. I have implemented two separate functions for those operations. Additionally, for loops are a bit less readable compared to the others but they are powerful when one learns how to use them. Nonetheless, Lua programming language is simple and orthogonal when it comes to working with associative

arrays. Lack of built-in functions decreases writability, but it is no difficult to predict that it a design choice in order to increase its reliability since Lua is a C-based language.

PHP is very similar to Dart in terms of its readability and writability due to its built-in functions, however one could get confused with new element addition unlike Dart. I could not find a way to use “array_push()” method to add new elements to an associative array, therefore I have solved the problem by concatenating two associative arrays. PHP allows indexed access but there is no use of that in my example. Also, PHP can be said to be orthogonal in terms of associative arrays since an array can be both treated as either traditional array or associative array. Overall, PHP’s associative arrays are readable and writable, but Dart has a slight edge over PHP with its ease of new element addition.

In Python, associative arrays are named “dictionaries” and it is directly supported. Keys can be any object rather than strings except arrays and hashes since they do not make good keys because of their continuous alterations. Dictionaries have their own built-in functions for checking for existence of keys and values. Iterating the dictionary is also straightforward and Python’s for loops are both readable and writable. Generally speaking, Python is equally readable and writable with Dart which are the best performers so far.

Ruby supports associative arrays directly too. I have used the array structure instead of the hash structure but both of them are possible. There is a “delete()” function for removing elements, therefore it makes the operation more readable than setting an array value to “null”. Ruby has built-in functions for checking for the existence of keys and values which increases its readability and writability. For loops are a bit different than traditional ones. The “end” keyword at the end of function declarations and “do” statements increase the readability of Ruby. There is a “.each” keyword for iterating each element in the array. It is writable once learned but one might be unfamiliar with the |x| operator. On the whole, Ruby’s readability and writability is decent but Dart and Python is more preferrable with their simplicity of for loops in “foo” functions.

Rust does not directly support associative arrays, so I needed to import the “HashMap” standard library. It has built-in functions for addition and deletion operations and accessing an element is straightforward with bracket operators. Although there is another built-in function for checking for the existence of a key in the HashMap, there is none for searching for the existence of values. Therefore, I have implemented a for loop which was kind of difficult compared to other due to its documentation. The “&” and “_” operators might confuse someone who reads the code segment for the first time. Function declarations have their own authentic way which again decreases the readability of the language. The need for stating the types of the arguments and whether the argument will be mutable or not might increase its reliability, however it definitely contributes to writability and readability of the language negatively. Overall, I would not prefer Rust to implement associative arrays since it is less readable and writable compared to Dart and Python.

In my opinion, Python and Dart is the best choices for associative array operations. They both have built-in functions for the parts 4 (removing an element), 6 (checking for existence of a key) and 7 (checking for existence of a value). They also have simple for loop structures which gives them a lead against PHP, Lua, Ruby, and Rust. When it comes to comparing Python and Dart, Dart might have a slight edge over Python if we only evaluate them based on readability and writability because of the built-in “foreach()” function of Dart. Iterations are also pretty straightforward in Python but “foreach()” function is more writable since it parses key and value implicitly. All eight operations including initialization is quite readable and writable in Dart, therefore it would be the best choice for working with associative arrays.

3.0 LEARNING STRATEGY

Each time I have started to implement associative arrays in a new programming language for the report, first of all I have looked up for documentation of comments since I have used comments in all of my code segments to separate sections. Then, I have learned how the associative arrays are called in the language and look up for their documentation, too. Some of the programming languages that I have used supports associative array directly such as Python and JavaScript whereas some of them requires to import a library for associative arrays such as Ruby's "HashMaps". The next task has always been to find out how to initialize an associative array. I have used the official websites of programming languages if I was able to find them for the documentation of associative arrays. PHP and Lua had such websites that I have used. Then, I have searched if those languages have built-in functions for checking whether a key and a value exist in the associative array. If not, I have implemented functions for that purpose.

I have printed the arrays each time I performed an operation so that I can trace the alterations in the array part by part. Although in most of the languages "print" or equivalent functions did let me pass the array directly as a parameter, some did not. In a few languages I have implemented a function to print the key – value pairs either because I was not able to pass the array as parameter to the print function or for visual arrangement purposes.

If I had any unexpected errors that I cannot solve, I have generally checked "www.stackoverflow.com" website to find the similar errors and their solutions. I have used "www.w3schools.com" and "www.tutorialspoint.com" website for further documentation information with examples in some cases.

I have used "www.replit.com" as my online compiler which supports every single programming language in this report.

I have had a little personal communication with some of my friends on the evaluation of the languages on readability and writability to make sure that I have not missed a data structure that

is more efficient and usable for any of the programming languages. Below, one can access all of the online sources that I have made use of.

RUST:

https://doc.rust-lang.org/std/collections/struct.HashMap.html

https://turreta.com/2019/09/07/rust-how-to-compare-strings/

RUBY:

https://programmingresources.fandom.com/wiki/Ruby-_Remove_Elements_From_An_Array

https://linuxhint.com/check-array-contains-values-ruby/

https://www.tutorialspoint.com/ruby/ruby_methods.htm

LUA:

https://www.lua.org/pil/2.5.html

PYTHON:

https://www.w3schools.com/python/python_dictionaries.asp

DART:

https://dart.dev/guides/language/language-tour

https://dart.dev/guides/language/effective-dart/documentation

https://www.oreilly.com/library/view/dart-1-for/9781680500479/f_0023.html

https://www.tutorialspoint.com/dart_programming/dart_programming_map_function_remove.htm

https://www.tutorialspoint.com/dart_programming/dart_programming_functions.htm

PHP:

https://flexiple.com/php/php-foreach/

https://www.w3schools.com/php/php_functions.asp

https://www.w3schools.com/php/php_arrays_associative.asp

https://stackoverflow.com/questions/5384847/adding-an-item-to-an-associative-array

ONLINE COMPILERS:

https://replit.com/languages/lua

https://replit.com/languages/python3

https://replit.com/languages/nodejs

https://replit.com/languages/dart

https://replit.com/languages/php_cli

https://replit.com/languages/ruby

https://replit.com/languages/rust