CS 315

Homework Assignment 1

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

Zeynep Selcen Öztunç

21902941

Table of Contents

1.Dart

In Dart, the associative arrays are called Maps. They are simply a collection of key/value pairs which can be of any type. They are dynamic so they can grow/shrink at runtime.

1.1 Initialize

The associative array is defined with the operator var, followed by the identifier for the array name and then the assignment operator "=" which is followed by curly braces. Inside the curly braces, the key-item pair values should be stated followed by commas.

```
void main() {
 //initializing the array
var newArr = {
 'a': 10,
 'b': 20,
 'c': 30,
 'd': 40,
 'e': 50,
 'f': 60,
 };
 print(newArr);
```

```
Console

 {a: 10, b: 20, c: 30, d: 40, e: 50, f: 60}
```

1.2 Get the value for a given key

In Dart, square brackets are used to access an item from the array. The print function is used to display the output on the console.

```

 //get the value for the given key
 print("The value for the key a is:");
 print(newArr['a']);

```

```
The value for the key a is:
10
```

1.3 Add a new element

To add a new element to the array, the new key to be added should be given inside square brackets. The new value to be added to the array is assigned by = operator.

```
14
15  // add a new element
16  newArr['g '] =70;
17  print(newArr);
18
```

```
{a: 10, b: 20, c: 30, d: 40, e: 50, f: 60, g : 70}
```

1.4 Remove an element

To remove an array element, remove() function can be used. The function takes the key of the key/value pair as a parameter.

```
18
19  //remove an element of the array
20  newArr.remove('f');
21  print(newArr);
22
```

```
{a: 10, b: 20, c: 30, d: 40, e: 50, g : 70}
```

1.5 Modify the value of an existing element

Modifying values of an existing element can be done in a similar manner with adding elements to the array. Again, square brackets are used to specify the key to be modified and the = operator is used to assign the value to that key.

```
23  //modify the value of an existing element
24  print("The initial value of e is : ");
25  print(newArr['e']);
26  newArr['e'] =100;
27  print("The value of e after modification is: ");
28  print(newArr['e']);
29
```

```
The initial value of e is :
50
The value of e after modification is:
100
```

1.6 Search for the existence of a key

To search whether a key exists, containsKey() function can be used, which takes a key as a parameter.

```
29
30  //search for the existence of a key
31  print(newArr.containsKey("a"));
32  print(newArr.containsKey("z"));
33
```

```
true
false
```

1.7 Search for the existence of a value

To search whether a key value, containsValue() function can be used, which takes a value as a parameter.

```
33
34  //search for the existence of a value
35  int value=20;
36  print("Does the value $value  exist in the array?");
37  print(newArr.containsValue(20));
38
```

```
Does the value 20  exist in the array?
true
```

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

The for loop iterates through every key in the array. For every pair, the foo function is called. The function signature includes the return type, which is void in this case, the function name, and the function parameters in parentheses. For the foo function, the function parameters are the array key and the array itself. Inside the function, a print function prints the key-value pairs.

```
38
39  //loop through the array
40    for (String key in newArr.keys){
41      foo(key,newArr);
42   }
43
44  //foo function
45  }
46  void foo(key,newArr){
47     print("The value for the key $key is: ");
48     print(newArr[key]);
49  }
```

```
The value for the key a is:
10
The value for the key b is:
20
The value for the key c is:
30
The value for the key d is:
40
The value for the key e is:
100
The value for the key g  is:
70
```

2.Javascript

In JavaScript, associative arrays are objects in which indexes are replaced with user-defined keys.

2.1 Initialize

The object is initialized using curly braces. In between the curly braces the keys are followed by the : operator, which is followed by the value for the key. The key-value pairs can be separated with a comma.

```
1  //initializing the array
2  var newArr = {
3  'a': 10,
4  'b': 20,
5  'c': 30,
6  'd': 40,
7  'e': 50,
8  'f': 60,
9  };
10 console.log(newArr);
11
```

```
[ a: 10, b: 20, c: 30, d: 40, e: 50, f: 60 ]
```

2.2 Get the value for a given key

The value for a given key can be accessed by using the [] operator and stating the key in between the brackets.

```
11  //get the value for the given key
12  console.log("The value for the key a is:")
13  console.log(newArr['a'])
14
```

```
The value for the key a is:
10
```

2.3 Add a new element

To add a new element, the [] and = operators are used. The new key value is expressed between the brackets and then the = operator is used to assign the new value to the new key.

```
15  // add a new element
16  newArr['g '] =70;
17  console.log(newArr);
```

```
[ a: 10, b: 20, c: 30, d: 40, e: 50, f: 60, 'g ': 70 ]
```

2.4 Remove an element

To remove an element, the delete operator and the [] operators are used.

```
//removing an element from the array
delete newArr['f'];
console.log(newArr);
```

```
[ a: 10, b: 20, c: 30, d: 40, e: 50, 'g ': 70 ]
```

2.5 Modify the value of an existing element

To modify the value of an existing element, the [] and = operators are used.

```
//modify the value of an existing element
console.log("The initial value of e: "+ newArr['e']);
newArr['e'] =100;
console.log("The value of e after modification is "+
    newArr['e']);
console.log(newArr);
```

```
The initial value of e: 50
The value of e after modification is 100
[ a: 10, b: 20, c: 30, d: 40, e: 100, 'g ': 70 ]
```

2.6 Search for the existence of a key

To search for an existence of a key, the operator in can be used. The result of the expression inside the parentheses is true if the key is in the array, false otherwise.

```
//search for the existence of a key
console.log("z" in newArr);
console.log("a" in newArr);
```

```
false
true
```

2.7 Search for the existence of a value

To search for the existence of a value, Object.values() and Array.includes() methods can be used. Object.values() takes a single parameter object, in this case it is the associative array newArr, and it returns an array of the object's enumerable property values. Array.includes method also has a single parameter which is the value to be searched for. It returns true when the array includes that value and false otherwise. I also implemented an alternative way of searching for a value which simply iterates through the array elements by a for loop and checks whether the value is in the array.

```
//search for the existence of a value
let value=20;
console.log("Does the value "+ value + " exist in the array?");
console.log(Object.values(newArr).includes(20));
```

```
37
38  //alternative way
39  let exist=0;
40  for (var key in newArr){
41      var name = newArr[key];
42      if (name == value) {
43          exist = 1;
44          break;
45      }
46  }
47
48  if(exist){
49      console.log("Exists");
50  }
51  else{
52      console.log("Does not exist");
53  }
54
```

```
Does the value 20 exist in the array?
true
Exists
```

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

Each key in the array is iterated by a for loop, using the **in** operator. In each iteration, a user defined function foo() is called. The foo function takes a single parameter i1 which corresponds to an array key. It prints the key-value pairs in the array using console.log()

```
54
55  //loop through the array
56  for (var key in newArr){
57      foo(key);
58  }
59
60  //foo function
61  function foo(i1){
62        console.log("The value for the key "+ i1+" is " +
              newArr[i1]);
63  }
```

```
The value for the key a is 10
The value for the key b is 20
The value for the key c is 30
The value for the key d is 40
The value for the key e is 100
The value for the key g  is 70
```

3.Lua

In Lua, the Table type implements associative arrays. Any number of items can be added to tables since they have no fixed size.

3.1 Initialize

Table elements can be initialized using “{}” , “[]” and “=” operators. The table key-value pairs are stated in {} operator and are separated by commas. The key of the table is given inside the [] operator and the value of the key is assigned by the = operator.

```
3  --initializing the array
4  local newArr = { ["a"] = 10,
5      ["b"] = 20,
6      ["c"] = 30,
7      ["d"] = 40
8  }
9   for key,value in pairs(newArr) do
10     print (key .. " - " .. value)
11  end
```

```
c - 30
b - 20
a - 10
d - 40
```

3.2 Get the value for a given key

To get the value for a given key, square brackets [] are used.

```
--get the value for the given key
  print("The value for the given key a is: " .. newArr["a"])
```

```
The value for the given key a is: 10
```

3.3 Add a new element

To add a new element to the array, [] and = operators are used. The new key to be added is stated inside the [] operator and the new value to be paired with it is assigned by the = operator.

```
--add a new element
print("Adding a new element with the key e")
 newArr["e"] = 50
for key,value in pairs(newArr) do
        print (key .. " - " .. value)
end
```

```
The value for the given key a is: 10
Adding a new element with the key e
e - 50
d - 40
c - 30
b - 20
a - 10
```

3.4 Remove an element

To remove an element from the array, the key of the key value pair is first stated in between square brackets[]. Then, it is assigned to **nil** by the = operator.

```
--remove an element of an array
print("Removing the element with the key b ")
newArr["b"] = nil
for key,value in pairs(newArr) do
        print (key .. " - " .. value)
end
```

```
Removing the element with the key b
e - 50
d - 40
c - 30
a - 10
```

3.5 Modify the value of an existing element

Modifying the value of an existing element can be done in a similar way as adding a new element using the [] and = operators, except this time an existing key is given inside [].

```lua
--modify the value of an existing element
print("The value of e is modified to 100")
 newArr["e"] = 100
for key,value in pairs(newArr) do
        print (key .. " - " .. value)
end

--search for the existence of a key
```

```
The value of e is modified to 100
a - 10
d - 40
c - 30
e - 100
```

3.6 Search for the existence of a key

To learn whether a key exists in Lua, the value of a key can be compared with **nil**, using the ~= operator. It will return true if the key exists in the array and false otherwise.

```lua
--search for the existence of a key
local k1="a"
local k2="w"
print("Search for the existence of key: "..k1)
print(newArr[k1] ~= nil)
print("Search for the existence of key: "..k2)
print(newArr[k2] ~= nil)
```

```
Search for the existence of key: a
true
Search for the existence of key: w
false
```

3.7 Search for the existence of a value

In Lua, there is no function to check whether a value exists, so I have written a for loop to do this. Inside the for loop, an if statement compares the values in the array to the value that is being searched. Based on whether the value is found or not, a message is printed accordingly.

```lua
--search for the existence of a value
local val=30
local isFound=0
print("Search for the existence of value: "..val)
for key in pairs(newArr) do
    if(newArr[key]==val)
        then
        isFound =1
     end
end
if isFound==1
    then
        print ("The value exists")
    else
        print ("The value does not exist")
end
```

```
Search for the existence of value: 30
The value exists
```

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

The array elements are iterated by the for loop and for each iteration the foo function is called. The function takes the key of the array as a parameter and prints the key-value pair.

```lua

--foo function
function foo(key)
   print("The value for the key ".. key.. " is: "..newArr[key])
end

--loop through the array
for key in pairs(newArr) do
   foo(key)
end
```

```
The value for the key e is: 100
The value for the key d is: 40
The value for the key c is: 30
The value for the key a is: 10
```

4.PHP

PHP's arrays are both normal arrays and associative arrays can be treated as either.

4.1 Initialize

In PHP associative arrays are declared inside parentheses() and the key-value pairs are separated by commas. The value of the keys are assigned by the => operator.

```php
<?php
//php 7.2.24
//initialize
$newArr = array("a"=>"20", "b"=>"40", "c"=>"60");
print_r($newArr);

```

```
Array
(
    [a] => 20
    [b] => 40
    [c] => 60
)
```

4.2 Get the value for a given key

To get the value for a given key, the key should be stated inside [] brackets. The $ sign is used to print the value of the key.

```php
//get the value for a given key
echo "The value for the key a is: ${newArr['a']} ";

```

```
The value for the key a is: 20
```

4.3 Add a new element

For adding a new element the $ is used before the array name, which is followed by [] operator. Inside [] the key is given and after that = operator is used to assign the new value.

```php
// add a new element
$newArr["d"]="80";
print_r($newArr);

```

```
Array
(
    [a] => 20
    [b] => 40
    [c] => 60
    [d] => 80
)
```

4.4 Remove an element

To remove an element from the array, the unset function can be used.

```php

//remove an element
unset($newArr['b']);
print_r($newArr);

```

```
Array
(
    [a] => 20
    [c] => 60
    [d] => 80
)
```

4.5 Modify the value of an existing element

Modifying the element is done in a similar manner with adding a new element, the only difference is that instead of giving a new key, an existing key which is wanted to be modified is used between []. Again the $ sign, and the [] and = operators are used.

```
18 //modify the value of an existing array
19 $newArr["d"]="160";
20 print_r($newArr);
```

```
Array
(
    [a] => 20
    [c] => 60
    [d] => 160
)
```

4.6 Search for the existence of a key

To search whether a key exists in PHP, the function array_key_exists() is used. It takes two parameters, the given key and the array to be looked in. It returns true if the key exists and false otherwise.

```
20 //search for the existence of a key
21 print_r("Does the key c exist?");
22 if (array_key_exists("c",$newArr))
23 {
24     echo "Key exists \n";
25 }
26 else{
27     echo "Key does not exist \n";
28 }
```

```
)
Does the key c exist?Key exists
```

4.7 Search for the existence of a value

To check whether a value exists in PHP, in_array() function can be used. The function has two parameters: the first one is the value to be searched for and the second one is the array to be looked in. It returns true if the key exists and false otherwise.

```
29 //search for an existence of a value
30 if( in_array( "60" ,$newArr ) )
31 {
32     echo "The value exists\n";
33 }
34 else{
35     echo "The value does not exist \n ";
36 }
37
```

```
The value exists
```

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

A foreach loop looks for every key-item pair and calls the foo function in each iteration. The foo function takes two parameters; the key and the array and it prints the key-value pairs on the console.

```php
//loop through the array
foreach ($newArr as $key => $value) {
 //echo $key;
    foo($key,$newArr);
}

//prints the key-value pair
function foo($x,$newArr) {
    echo "The value for the key ". $x." is ". $newArr[$x]."\n";
}
?>
```

```
The value for the key a is 20
The value for the key c is 60
The value for the key d is 160
```

5.Python

In Python, associative arrays are implemented by dictionaries. Each key in a dictionary should be unique and a Dictionary can only store one item for each key value.

5.1 Initialize

A dictionary can be initialized using {} and : operators as shown in the sample code:

```python
#initialize
newArr = {'a': 10, 'b': 20, 'c':30,'d':40}
print(newArr)

#get the value for a given key
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

5.2 Get the value for a given key

To get the value for a given key, the key should be stated inside [] brackets.

```python
#get the value for a given key
print(newArr["a"])
```

```
10
```

5.3 Add a new element

To add a new element to the dictionary, the key should be stated inside [] brackets, and then the = operator should be used to assign the value to the key.

```
# add a new element
newArr["e"] = "50"
print(newArr)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': '50'}
```

5.4 Remove an element

In Python, a dictionary element can be deleted using **del** keyword. The array name followed by the key inside the [] operator should be followed by it.

```
#remove an element
del newArr["e"]
print(newArr)
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

5.5 Modify the value of an existing element

Modifying an element is done in a similar manner with adding, the [] and the = operators are used.

```
#modify the value of an existing element
newArr["a"] = "11"
print(newArr)
```

```
{'a': '11', 'b': 20, 'c': 30, 'd': 40}
```

5.6 Search for the existence of a key

To search for an existence of a key, the **in** operator can be used in an if statement. If the key is in the dictionary it returns true, otherwise it returns false.

```
#search for the existence of a key
if "b" in newArr:
 print ("The key exists")
else:
  print ("Key does not exist")
```

```
The key exists
```

5.7 Search for the existence of a value

To search if a value exists in a dictionary, the **in** operator and the values() method can be used. The values method returns a view object that contains the value of the dictionary as a list.

```
#search for an existence of a value
if 30 in newArr.values():
 print ("The value exists")
else:
  print ("The value does not exist")

```

```
The value exists
```

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

The for loop iterates through the dictionary keys, and in each iteration the user defined foo method is called. The foo method takes the dictionary key as a parameter and it prints the key-value pair using that key.

```
#prints the key-value pair
def foo(key):
     print(f"The value for the key ", key ," is ", newArr[key])

#loop through the array
for val in newArr:
    foo(val)

```

```
The value for the key  a  is  11
The value for the key  b  is  20
The value for the key  c  is  30
The value for the key  d  is  40
```

6. Ruby

In Ruby, associative arrays are called Hash. A Hash maps each of its keys to a unique value.

6.1 Initialize

The hash is initialized using the {} operator. The key-value pairs are written in between the {} operator and are separated by commas. The keys are assigned to values using the => operator.

```
#initialize
newArr = {'a'=> 10, 'b'=> 20, 'c'=>30,'d'=>40}
puts newArr

```

```
{"a"=>10, "b"=>20, "c"=>30, "d"=>40}
```

6.2 Get the value for a given key

To get the value for a key, the key should be written inside [] operator.

```
#get the value for a given key
puts newArr['a']
```

```
10
```

6.3 Add a new element

To add a new element, the new key should be written inside the [] operator which then should be followed by the = operator, and the value to be added.

```
# add a new element
newArr['e']=50
puts newArr
```

```
{"a"=>10, "b"=>20, "c"=>30, "d"=>40, "e"=>50}
```

6.4 Remove an element

To remove an element from the hash, the delete method is used. It only takes a single parameter, which is the key of the element to be deleted.

```
#remove an element
newArr.delete('b')
puts newArr
```

```
{"a"=>10, "c"=>30, "d"=>40, "e"=>50}
```

6.5 Modify the value of an existing element

To modify the value of an element, the key should be written inside the [] operator which then should be followed by the = operator, and the value to be modified.

```
#modify the value of an existing
newArr['e']=100
puts newArr
```

```
{"a"=>10, "c"=>30, "d"=>40, "e"=>100}
```

6.6 Search for the existence of a key

To search for the existence of a key, the assoc() method is used. The assoc() takes a single parameter which is the key to be searched for. It returns true if the key exists, false otherwise.

```ruby
#search for an existence of a key
if newArr.assoc('c')
    puts "The key exists"
else
    puts "The key does not exist"
end
```

```
The key exists
The value for the key "a" is 10
```

6.7 Search for the existence of a value

To search for an existence of a value, the flatten() and the include?() methods are used. The flatten method returns a one dimensional flattening of the hash and the include? method checks whether the value is in the hash by taking the value as a parameter. It returns true if the value exists, false otherwise.

```ruby
#search for the existence of a value
 puts newArr.flatten.include?(10)
```

```
true
The value for the key "a" is 10
```

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

By using a for loop, the foo method is called for the number of keys in the hash. The user-defined foo method takes the key and the hash as parameters and prints the key-value pairs inside.

```ruby
#prints the key-value pair
def foo(key,newArr)
    puts "The value for the key #{key.inspect} is #{newArr[key].inspect}"
end

#Loop through the array
for key in newArr.keys.sort
    foo(key,newArr)
end
```

```
The value for the key "a" is 10
The value for the key "c" is 30
The value for the key "d" is 40
The value for the key "e" is 100
```

7.Rust

In Rust, the associative arrays are implemented through HashMap.

7.1 Initialize

The key and its corresponding value are written inside the () operator separated by commas. The key-value pairs are also separated by commas inside the [] operator. The keyword **mut** means that the map is mutable. The sample code for initializing a hashmap is as follows:

```
//rustc 1.39.0

fn main() {

//initializing the array
    use std::collections::HashMap;
    //let mut newArr = HashMap::new();
    let mut newArr: HashMap<&str, i32> =
    [("a", 4),
    ("b", 3),
    ("c", 4),
    ("d", 2)].iter().cloned().collect();
    println!("{:?}", newArr);
```

```
{"d": 40, "a": 10, "c": 30, "b": 20}
```

7.2 Get the value for a given key

To get the value for a given key, the key should be written inside the [] operator. HashMap.get() method can also be used for this purpose

```

//get the value for the given key
    println!("{}",newArr["b"]);

```

```
20
```

7.3 Add a new element

To add a new element to the hashmap, HashMap.insert() method can be used which takes the key of the element to be added and the value of the key as parameters.

```
// add a new element
    println!("Adding a new element with key e");
    newArr.insert(String::from("e"), 50);
    println!("{:?}", newArr);
```

```
Adding a new element with key e
{"d": 40, "a": 10, "c": 30, "e": 50, "b": 20}
```

7.4 Remove an element

To remove an element, HashMap.remove() method can be used. It takes only a single parameter, which is the key of the element to be deleted

```
//remove an element of an array
     println!("Remove the element with key a");
     newArr.remove("a");
     println!("{:?}", newArr);
```

```
Remove the element with key a
{"d": 40, "c": 30, "e": 50, "b": 20}
```

7.5 Modify the value of an existing element

To modify the value of an existing element, the insert method can be used to overwrite the existing value of the key.

```
//modify the value of an existing element
    println!("Modify the element wih key e");
    newArr.insert(String::from("e"), 100);
    println!("{:?}", newArr);
```

```
Modify the element wih key e
d: 40 c: 30 e: 100 b: 20
```

7.6 Search for the existence of a key

To search whether a key exists in the hash, containsKey() method can be used which takes a single parameter which is the key of the hash.

```
//search for the existence of a key
println!("Does the key b exist?");
println!("{}", newArr.contains_key("b"));
```

```
Does the key b exist?
true
```

7.7 Search for the existence of a value

To check whether a value exists in a hashmap, “values().any(|&x| x== value)” can be used. **Any** method returns true when the value is found, false otherwise.

```
//search for the existence of a value
println!("Does the value 3 exist?");
let value_exists = newArr.values().any(|&x| x == 3);
    if value_exists{
            println!("The value exists");
    }
    else{
        println!("The value does not exist");
    }
```

```
Does the value 3 exist?
The value exists
```

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

The hashmap elements are iterated through a for loop writing the key and value between () and using **in.** With each iteration, the foo function is called. The foo function takes the key and value of the element as parameters and prints a string containing each key-value pair.

```
46
47 //Loop through the array
48  for (key, value) in &newArr {
49        foo(key.to_string(),*value);
50   }
51 }
52 //foo function
53 fn foo(key:String,value:i32){
54      print!("{}: {} ", key, value);
55 }
```

```
d: 40 c: 30 e: 100 b: 20
```

8.Evaluation of the Languages

8.1 Dart

I believe the Dart language is close to the English language with its specifiers and keywords, so it was easy to read. The loops and conditionals are straightforward and the use of curly brackets for the scopes also make it readable. I think Dart was also easy to write. Names of the data types and the built in functions made it writable and readable.

8.2. JavaScript

I think the JavaScript was easy to read since the language is close to the English language with its specifiers and keywords and function names. The loops and conditionals are straightforward and the use of curly brackets for the scopes also make it readable. I think JavaScript was also easy to write as well. Use of the square brackets [] to get, add and modify elements made it readable and writable in my opinion.

8.3 Lua

I think Lua was readable and writable, array operations such as deletion, addition and multiplication were usually straightforward. However, the use of Tables made the code writing a bit trickier. The **nil** type was something I have never seen before and was unusual, but it was easy to learn and implement.

8.4 PHP

I think PHP was hard to read and write. The array declarations, adding, modifying and deleting an item from the array were not straightforward and the function names were not simple. The use of $ sign made writing even more trickier, so I believe PHP is not readable or writable.

8.5 Python

I think the readability and the writability of Python were high. The array initialization, adding, deleting and modifying elements were pretty simple. The loops and the function declarations were easy to read and write as well. This might be due to the reason that Python is similar to the English language.

8.6 Ruby

I think Ruby was partly readable, the array initialization, modifying a new element and the array addition were straightforward but the methods used in searching for a key made it hard to read. The excessive amount of methods also did not make it easy to write. Even the concatenation of the strings were tricky, and I had to look up the documentations for many things.

8.7 Rust

In my opinion, Rust was not readable or writable. There are so many operators and functions used which makes it hard to keep up with what the code is trying to do. The code was also hard to write, I had to check for documentation and other sources multiple times just for a single operation. For writing, many declaration assignment statements and functions were not straightforward and the language had a lot of operators and keywords.

8.8 Conclusion

I believe out of all these languages, Rust was the hardest in terms of the readability and writability of associative arrays because of all of its expressions, functions and operators. I also think that the easiest language to implement the associative arrays was Python. It was very close to the English language, and even though it was my first time using Python, I had no difficulties writing it.

9.Learning Strategy

To learn the associative arrays in all of these languages, I first checked the official documentations of the languages. For the simple array operations like addition and deletion, I first tried implementing these myself, and when I had an error, I checked the internet for the actual expressions/methods/declarations that should be used. When I faced a specific error, I

solved it by looking at stackoverflow and other forums. For the methods/functions that I have used in the languages, if they returned a boolean value, I tried getting both true and false as a result so that I would know the function/ method worked. After each array operation that I did, such as addition, deletion and insertion, I printed the array afterwards so that I knew each of the operations were done correctly. Even though I was not familiar with the languages except for JavaScript, I eventually figured out how to implement associative arrays in these languages in a short amount of time.

URLs of the compilers that I have used:

For Dart: https://dartpad.dev/?

For JavaScript: https://www.programiz.com/javascript/online-compiler/

Other sources that I have used are as follows:

https://dartpad.dev/?

https://ruby-doc.org/core-3.1.2/Hash.html

https://apidock.com/ruby/Array/flatten

https://developer.mozilla.org/en-US/docs/Web/JavaScript

https://flexiple.com/javascript/associative-array-javascript/

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

https://docs.rs/assoc/latest/assoc/

https://python-course.eu/python-tutorial/dictionaries.php

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