2022-2023 Fall

CS 315 - Programming Languages

Homework 1 – Report

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

Berk Çakar - 22003021 - Section 1

Table of Contents

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

a) Dart

1) Initialize

```dart
// Initialize the associative array (HashMap)
HashMap hashMap = HashMap.of({
  'name': 'Berk Çakar',
  'university': 'Bilkent University',
  'department': 'Computer Engineering',
  'universityIdNo': 22003021,
  'isUndergraduate': true,
  'age': 21,
  'height': 1.82
});

print(hashMap);
print('Associative array has been initialized successfully');
```

In Dart programming language, after importing the “dart:collection” library, an associative array can be obtained using a map (Map<K, V>). However, the Map class is a base class for different related data structures. If the Map class is used on its own (Map map = {…}), it will default to an ordered map structure, which is LinkedHashMap. Since the LinkedHashMap data structure keeps its key-value pairs ordered by insertion order, and we are investigating unordered (as stated in the homework prompt) associative arrays, usage of the hash map data structure is a better approach. Hash map is one of the data structures that implement the Map class, but unlike the rest, it is unsorted, as we also desired. To initialize a hash map, one can use the format given above. As can be seen, the hash map is non-homogeneous right now, but the values can be limited to a certain type using HashMap<String, Type>. Lastly, the HashMap.of() function acts as the constructor for the HashMap class; it takes some key-value pairs and constructs the hash map. The same functionality could have also been achieved using HashMap.from() function. In our example above, we initialize a hash map with some values and then print it to ensure that it has been initialized successfully.

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 1: INITIALIZE ---------------------------------------------
{university: Bilkent University, department: Computer Engineering,
universityIdNo: 22003021, name: Berk Çakar, age: 21, height: 1.82,
isUndergraduate: true}
Associative array has been initialized successfully
```

2) Get the Value for a Given Key

```dart
// Get the value for the given key
print(hashMap['name']);
print(hashMap['university']);
print(hashMap['department']);
print(hashMap['universityIdNo']);
print(hashMap['isUndergraduate']);
print(hashMap['age']);
print(hashMap['height']);
```

First of all, all data structures derived from Map in Dart share identical functionalities. Therefore, the procedure for adding, updating, retrieving, and deleting a key-value pair is the same in Map, LinkedHashMap, HashMap, and more. To get a specific value for a given key in hash map, square bracket notation is used. If the key provided in the square bracket does not exist in the hash map, it returns null; otherwise it returns the value of the given key. In our example, we get and print all of the values available in our hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 2: GET THE VALUE FOR THE GIVEN KEY ---------------
Berk Çakar
Bilkent University
Computer Engineering
22003021
true
21
1.82
```

3) Add a New Element

```dart
// Add a new element
hashMap['weight'] = 70;
print('weight: ${hashMap['weight']}');
```

In order to enter a new element into a hash map, the square bracket notation can still be used. If the key in the square brackets already exists in the hash map, then its value will be updated, otherwise the given key-value pair will be added to the hash map. Alternatively, one can use the functions such as addAll(), addEntries(), and putIfAbsent(). In our example, we add a new key named “weight” with value of 70 to our hash map, then print it.

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 3: ADD A NEW ELEMENT ------------------------------
Adding 'weight' key with value 70
weight: 70
```

4) Remove an Element

```
print('Removing \'isUndergraduate\' key');
// Remove an element
hashMap.remove('isUndergraduate');
print('isUndergraduate: ${hashMap['isUndergraduate']}');
```

For removing an element remove() function is used. The function gets the key and removes the entry from the hash map. It actually returns the deleted value such that the deleted value can be assigned into variables (i.e., var x = hashMap.remove("age"); // x becomes 21). It returns null if the given key is not present in the hash map. In our example, we remove the "isUndergraduate" key from the hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 4: REMOVE AN ELEMENT ------------------------------
Removing 'isUndergraduate' key
isUndergraduate: null
```

5) Modify the Value of an Existing Element

```
print('Value of the \'age\' key before: ${hashMap['age']}');
print('Modifying the value of the \'age\' key');
// Modify the value of an existing element
hashMap['age'] = 22;
print('Value of the \'age\' key after: ${hashMap['age']}');
```

To modify an existing element, again, square bracket notation is used. If the given key exists in the hash map, the key's value will be updated with the given expression. Otherwise, a new key-value pair will be created. Alternatively, update() and updateAll() methods can also be used. In our example, we update the value of the "age" key to 22.

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 5: MODIFY THE VALUE OF AN EXISTING ELEMENT --------
Value of the 'age' key before: 21
Modifying the value of the 'age' key
Value of the 'age' key after: 22
```

6) Search for the Existence of a Key

```dart
// Search for the existence of a key
if (hashMap.containsKey('name')) {
  print('\'name\' key exists');
} else {
  print('\'name\' key does not exist');
}

if (hashMap.containsKey('isUndergraduate')) {
  print('\'isUndergraduate\' key exists');
} else {
  print('\'isUndergraduate\' key does not exist');
```

In order to check if the given key exists in the hash map or not, Dart provides the containsKey() function. It takes the desired key as a parameter and returns a boolean value depending on the existence of the key. In our example, we check if the “name” and “isUndergraduate” keys exist in the hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 6: SEARCH FOR THE EXISTENCE OF A KEY ----------
'name' key exists
'isUndergraduate' key does not exist
```

7) Search for the Existence of a Value

```dart
// Search for the existence of a value
if (hashMap.containsValue(22003021)) {
  print('22003021 value exists in one of the keys');
} else {
  print('22003021 value does not exist in one of the keys');
}

if (hashMap.containsValue('Basketball')) {
  print('Basketball value exists in one of the keys');
} else {
  print('Basketball value does not exist in one of the keys');
}
```

Similarly, we can check whether a given value exists in the hash map using the containsValue() function. It takes the desired value as a parameter and returns a boolean value depending on the existence of the key. It is a short-circuited function and stops searching as it encounters the desired value. In our example, we check if the “22003021” and “Basketball” values exist in the hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 7: SEARCH FOR THE EXISTENCE OF A VALUE ----------
22003021 value exists in one of the keys
Basketball value does not exist in one of the keys
```

8) Loop Through an Associative Array, Apply a Function, Called Foo, Which Simply Prints the Key-Value Pair

```dart
// A function to print the given key-value pair
void foo(key, value) {
    print('${key}: ${value}');
}

void main() {
  // ...

  // Loop through an associative array, apply a function, called foo,
  // which simply prints the key-value pair
  for (var key in hashMap.keys) {
    foo(key, hashMap[key]);
  }
}
```

Functions in Dart programming language is declared as it is in C/C++. Accordingly, a return type, a function name, parameters, and a function body, in the given order, forms a callable function. And for iterating over a hash map, keys of the hash map can be gathered using the “keys” property and looped through using the “in” keyword, similar to Python. In our example, we declare a function named “foo” with parameters “key” and “value”. Then, we iterate over the keys of the “hashMap” using the “in” keyword, and print out the key-value pairs using “foo” function. Since the HashMap data structure is unordered, it is expected to get key-value pairs in a random order (i.e., in a different order than what is given in the initialization phase).

```
friberk@Berk-MacBook-Pro  hw1 ❯ dart "cakar_berk.dart"
PART 8: PRINT THE ASSOCIATIVE ARRAY VIA THE FOO FUNCTION
university: Bilkent University
department: Computer Engineering
universityIdNo: 22003021
name: Berk Çakar
age: 22
height: 1.82
weight: 70
```

b) Javascript

1) Initialize

```
// Initialize the associative array old way, it is an object actually
var oldMap = {
    'name': 'Berk Çakar',
    'university': 'Bilkent University',
    'department': 'Computer Engineering',
    'universityIdNo': 22003021,
    'isUndergraduate': true,
    'age': 21,
    'height': 1.82
}

// Map data structure initialization
var map = new Map([
    ['name', 'Berk Çakar'],
    ['university', 'Bilkent University'],
    ['department', 'Computer Engineering'],
    ['universityIdNo', 22003021],
    ['isUndergraduate', true],
    ['age', 21],
    ['height', 1.82]
]);

console.log(oldMap);
console.log(map);
console.log('Associative array has been initialized successfully');
```

Although there is no Javascript equivalent of an unordered associative array, plain objects and the “Map” data structure can achieve similar functionality. However, Map is introduced in the recent versions of Javascript, and it remembers the original insertion order of its keys. Map objects allow the use of any type for keys, while in plain objects only the string values can be used as keys. Also, it provides alternative ways for adding, updating, retrieving, and deleting a key-value pair to the structure. On the other hand, plain objects’ keys are ordered as well, in the recent versions of Javascript. However, before ES6 (2015), the order of the keys of a plain object was not guaranteed. Therefore, I will evaluate map data structure instead of plain objects for this part of the homework. Since we do not have an unordered alternative, it will be more convenient to use the more optimized and feature-loaded data structure, which is map. In the source code above, it is possible the see the initialization format for both plain objects and maps. Then we print it to ensure that the map has been initialized successfully.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 1: INITIALIZE ------------------------------------
{
  name: 'Berk Çakar',
  university: 'Bilkent University',
  department: 'Computer Engineering',
  universityIdNo: 22003021,
  isUndergraduate: true,
  age: 21,
  height: 1.82
}
Map(7) {
  'name' => 'Berk Çakar',
  'university' => 'Bilkent University',
  'department' => 'Computer Engineering',
  'universityIdNo' => 22003021,
  'isUndergraduate' => true,
  'age' => 21,
  'height' => 1.82
}
Associative array has been initialized successfully
```

Note: For the plain objects, operations in parts 2, 3, 4, and 5 are done using the square bracket notation, similar to Dart. Since I will explore the newer Map type, operations related to plain object representation of associative arrays will not be included in the rest of this section.

2) Get the Value for a Given Key

```
// Get the value for the given key
console.log(map.get('name'));
console.log(map.get('university'));
console.log(map.get('department'));
console.log(map.get('universityIdNo'));
console.log(map.get('isUndergraduate'));
console.log(map.get('age'));
```

In order to retrieve a specific value for a given key, get() method is used. The method takes the desired key as a parameter and returns the value which corresponds to the given key. It returns “undefined” if the given key does not exist in the map. For any operation, square bracket operator does not work with maps (updating, retrieving, deleting…). In the example above, we retrieve and print every available value in our map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 2: GET THE VALUE FOR THE GIVEN KEY ------------
Berk Çakar
Bilkent University
Computer Engineering
22003021
true
21
```

3) Add a New Element

```
console.log('Adding \'weight\' key with value 70');
// Add a new element
map.set('weight', 70);
console.log('weight: ' + map.get('weight'));
```

For adding a new element to a map, the set() function is used. It takes the key-value pair to be added in two parameters. If the given key already exists in the map, its value will be overwritten. In our example, we add a new key named “weight” with value of 70 to our map, then print it.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 3: ADD A NEW ELEMENT --------------------------
Adding 'weight' key with value 70
weight: 70
```

4) Remove an Element

```
console.log('Removing \'isUndergraduate\' key');
// Remove an element
map.delete('isUndergraduate');
console.log('isUndergraduate: ' + map.get('isUndergraduate'));
```

In order to remove an element from a map, delete() function is used. It takes the key to be deleted as a parameter, and returns true or false depending on the success of the deletion. If the given key does not exist in the map, it returns false. In our example, we remove the “isUndergraduate” key from the map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 4: REMOVE AN ELEMENT --------------------------
Removing 'isUndergraduate' key
isUndergraduate: undefined
```

5) Modify the Value of an Existing Element

```
console.log('Value of the \'age\' key before: ' + map.get('age'));
console.log('Modifying the value of the \'age\' key');
// Modify the value of an existing element
map.set('age', 22);
console.log('Value of the \'age\' key after: ' + map.get('age'));
```

For modifying a value in a map, set() function is used. It takes the key-value pair to be modified in two parameters. If the given key does not exist in the map, it will be created with the given value. In our example, we update the value of the "age" key to 22.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 5: MODIFY THE VALUE OF AN EXISTING ELEMENT -----
Value of the 'age' key before: 21
Modifying the value of the 'age' key
Value of the 'age' key after: 22
```

6) Search for the Existence of a Key

```
// Search for the existence of a key
if (map.has('name')) {
    console.log('\'name\' key exists');
} else {
    console.log('\'name\' key does not exist');
}

if (map.has('isUndergraduate')) {
    console.log('\'isUndergraduate\' key exists');
} else {
    console.log('\'isUndergraduate\' key does not exist');
}
```

In order to check if the given key exists in the map or not, Javascript provides the has() function. It takes the desired key as a parameter and returns a boolean value depending on the existence of the key. In our example, we check if the "name" and "isUndergraduate" keys exist in the map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 6: SEARCH FOR THE EXISTENCE OF A KEY -----------
'name' key exists
'isUndergraduate' key does not exist
```

7) Search for the Existence of a Value

```
// Search for the existence of a value
if ([...map.values()].includes(22003021)) {
    console.log('22003021 value exists in one of the keys');
} else {
    console.log('22003021 value does not exist in one of the keys');
}

// Search for the existence of a value
if ([...map.values()].includes("Basketball")) {
    console.log('Basketball value exists in one of the keys');
} else {
    console.log('Basketball value does not exist in one of the keys');
}
```

For searching whether a value exists in the map, Javascript does not provide any built-in functions. However, it is still possible to produce an efficient solution. Map provides an iterator for its values which can be gathered using the values() function. The values which are pointed by this iterator can be unpacked to an array using the "[…(iterator)]" syntax. Finally, since we obtained an array, we can use the built-in includes() function to check whether the desired value exists in the map's values. Includes function is short-circuited function and returns true as it encounters the desired value. In our example, we check if the "22003021" and "Basketball" values exist in the map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 7: SEARCH FOR THE EXISTENCE OF A VALUE ---------
22003021 value exists in one of the keys
Basketball value does not exist in one of the keys
```

8) Loop Through an Associative Array, Apply a Function, Called Foo, Which Simply Prints the Key-Value Pair

```
// A function to print the given key-value pair
function foo(key, value) {
    console.log(key + ": " + value);
}

// Loop through an associative array, apply a function, called foo, which
simply prints the key-value pair
for (let [key, value] of map) {
    foo(key, value);
}
```

In Javascript, functions are defined using the “function” keyword followed by the function name, parameters, and function body. And for iterating over a map, the “of” keyword can be used. In every iteration, the iterator points to an element of the map, and we assign the terms which form that element to “key” and “value” variables. In our example, we declare a function named “foo” with parameters “key” and “value”. Then, we iterate over the “map”, and call the “foo” function for corresponding key-value pairs. Since the map data structure is ordered, it is expected to get key-value pairs in the insertion order.

```
friberk@Berk-MacBook-Pro  hw1 ❯ node "cakar_berk.js"
PART 8: PRINT THE ASSOCIATIVE ARRAY VIA THE FOO FUNCTION
name: Berk Çakar
university: Bilkent University
department: Computer Engineering
universityIdNo: 22003021
age: 22
height: 1.82
weight: 70
```

c) Lua

1) Initialize

```lua
-- Initialize the associative array, it is called as a table in Lua
table = {
    name = 'Berk Çakar',
    university = 'Bilkent University',
    department = 'Computer Engineering',
    universityIdNo = 22003021,
    isUndergraduate = true,
    age = 21,
    height = 1.82
}

print(table)
print('Associative array has been initialized successfully')
```

An unordered associative array can be achieved in Lua programming language using the “table” data type. In fact, “tables” are the only data structure available in Lua. They are used in representing ordinary arrays as well. In the case that keys are not specified explicitly, keys will default to consecutive integers, and the table will become an ordinary array. Moreover, it is also possible to implement sets, queues, stacks, records, and other data structures using tables. To initialize a table as an associative array, the syntax given above can be used. Tables are objects in Lua, and “{}” acts as the

constructor for tables. In our example above, we initialize a table with some values, and then print it to ensure that it has been initialized successfully (the print() function will output the table's address in the memory).

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 1: INITIALIZE ----------------------------------
table: 0x7fd4b0508100
Associative array has been initialized successfully
```

2) Get the Value for a Given Key

```lua
-- Get the value for the given key
print(table['name']);
print(table['university']);
print(table['department']);
print(table['universityIdNo']);
print(table['isUndergraduate']);
print(table['age']);
print(table['height']);
```

To get a value for a given key, square bracket notation can be used in Lua. On the other hand, it is also possible to use dot operator followed by the key's name (i.e., print(table.university)) to get the value. If the given key is not available in the table, "nil" will be returned. In the example above, we retrieve and print every available value in our table.

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 2: GET THE VALUE FOR THE GIVEN KEY -------------
Berk Çakar
Bilkent University
Computer Engineering
22003021
true
21
1.82
```

3) Add a New Element

```lua
print('Adding \'weight\' key with value 70')
-- Add a new element
table['weight'] = 70
print('weight: ' .. table['weight'])
```

For adding a new element to a table, the square bracket notation can be used. Similarly, it is possible to add a new field using the dot operator (i.e., table.weight = 70). For convenience, in our example above, we use the square bracket notation and add a new

key named “weight” with a value of 70, then print it. If the given key already exists in the table, its value will be overwritten.

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 3: ADD A NEW ELEMENT ----------------------------
Adding 'weight' key with value 70
weight: 70
```

4) Remove an Element

```
print('Removing \'isUndergraduate\' key')
-- Remove an element
table['isUndergraduate'] = nil
print('isUndergraduate: ', table['isUndergraduate'])
```

In order to remove an element from a table, it is enough to set the key’s value as nil. For that both square bracket notation and dot operator can be used. In our example, we remove the “isUndergraduate key from the table.

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 4: REMOVE AN ELEMENT ----------------------------
Removing 'isUndergraduate' key
isUndergraduate:        nil
```

5) Modify the Value of an Existing Element

```
print('Value of the \'age\' key before: ' .. table['age'])
print('Modifying the value of the \'age\' key')
-- Modify the value of an existing element
table['age'] = 22
print('Value of the \'age\' key after: ' .. table['age'])
```

To modify an existing element's value, we can use the square bracket notation or the dot operator. If the given key does not exist in the table, it will be created with the given value. In our example, we modify the value of ‘age’ to 22.

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 5: MODIFY THE VALUE OF AN EXISTING ELEMENT ------
Value of the 'age' key before: 21
Modifying the value of the 'age' key
Value of the 'age' key after: 22
```

6) Search for the Existence of a Key

```lua
-- Search for the existence of a key
if table.name ~= nil then -- or, table['name'] ~= nil
    print('\'name\' key exists')
else
    print('\'name\' key does not exist')
end

if table.isUndergraduate ~= nil then -- or, table['isUndergraduate'] ~=
nil
    print('\'isUndergraduate\' key exists')
else
    print('\'isUndergraduate\' key does not exist')
end
```

To check if a key exists in the table, we can check if its value is nil or not. If its value is not nil, then we ensure that the given key exists in the table. In our example, we check if the table contains the “name” and “isUndergraduate” keys.

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 6: SEARCH FOR THE EXISTENCE OF A KEY -----------
'name' key exists
'isUndergraduate' key does not exist
```

7) Search for the Existence of a Value

```lua
-- Search for the existence of a value
found1, found2 = false, false
for key, value in pairs(table) do
    if value == 22003021 then
        print('22003021 value exists in key: ' .. key)
        found1 = true
        break
    end
end

if found1 == false then
    print('22003021 value does not exist in one of the keys')
end

for key, value in pairs(table) do
    if value == 'Basketball' then
        print('Basketball value exists in key: ' .. key)
        found2 = true
        break
    end
end

if found2 == false then
    print('Basketball value does not exist in one of the keys')
end
```

Unfortunately, Lua does not offer a built-in solution to perform such a task. Therefore, it is required the iterate over the table and look for the desired value. For that, we use the “in” keyword to iterate over the “pairs” of the table, and in each iteration, we assign elements of the pair to “key” and “value” variables. Then, if the “value” is the same as the desired value, we break the loop. In our example, we check if the “22003021” and “Basketball” values exist in our table.

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 7: SEARCH FOR THE EXISTENCE OF A VALUE ----------
22003021 value exists in key: universityIdNo
Basketball value does not exist in one of the keys
```

8) Loop Through an Associative Array, Apply a Function, Called Foo, Which Simply Prints the Key-Value Pair

```lua
-- A function to print the given key-value pair
function foo(key, value)
    print(key .. " = " .. value)
end

-- ...

-- Loop through an associative array, apply a function, called foo, which
-- simply prints the key-value pair
for key, value in pairs(table) do
    foo(key, value)
end
```

In Lua programming language, function definitions are similar to Javascript. They start with the “function” keyword, followed by the function name, parameters, and function body. Function definitions end with the “end” keyword. In our example, we traverse over the table, just as we did in the previous part, and for each iteration, we call the “foo” function, which takes “key” and “value” variables as a parameter and then prints them. As expected, we get the key-value pairs as unordered.

```
friberk@Berk-MacBook-Pro  hw1 ❯ lua "cakar_berk.lua"
PART 8: PRINT THE ASSOCIATIVE ARRAY VIA THE FOO FUNCTION
weight = 70
name = Berk Çakar
department = Computer Engineering
height = 1.82
age = 22
universityIdNo = 22003021
university = Bilkent University
```

d) PHP

1) Initialize

```php
// Initialize the associative array
$assoc_array = [
    'name' => 'Berk Çakar',
    'university' => 'Bilkent University',
    'department' => 'Computer Engineering',
    'universityIdNo' => 22003021,
    'isUndergraduate' => true,
    'age' => 21,
    'height' => 1.82
];

print_r($assoc_array);
echo 'Associative array has been initialized successfully', PHP_EOL;
```

Similar to tables in Lua, PHP's only native data structure is the arrays (libraries add more data structures anyway). Again, similar to Lua, PHP does not distinguish between associative and indexed arrays. However, there is not any unordered alternative for associative arrays in PHP. Arrays are ordered by default. To initialize arrays in PHP, the "array()" constructor or "[]" syntax can be used. In our example above, we use square bracket syntax to indicate that the given expression is an array and put specified key-value pairs inside the square brackets to initialize an associative array. Lastly, we print the array to ensure that it has been initialized correctly.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"
PART 1: INITIALIZE ----------------------------------
Array
(
    [name] => Berk Çakar
    [university] => Bilkent University
    [department] => Computer Engineering
    [universityIdNo] => 22003021
    [isUndergraduate] => 1
    [age] => 21
    [height] => 1.82
)
Associative array has been initialized successfully
```

2) Get the Value for a Given Key

```php
// Get the value for the given key
echo $assoc_array['name'], PHP_EOL;
echo $assoc_array['university'], PHP_EOL;
echo $assoc_array['department'], PHP_EOL;
echo $assoc_array['universityIdNo'], PHP_EOL;
echo $assoc_array['isUndergraduate'], PHP_EOL;
echo $assoc_array['age'], PHP_EOL;
echo $assoc_array['height'], PHP_EOL;
```

In order to get a value for a given key, square bracket notation is used. If the given key does not exist in the associative array PHP just gives a warning and continues to execution (i.e., echo $assoc_array[‘test’], will not print out anything but the execution will continue). In the example above, we retrieve and print every available value in our associative array.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"
PART 2: GET THE VALUE FOR THE GIVEN KEY ------------
Berk Çakar
Bilkent University
Computer Engineering
22003021
1
21
1.82
```

3) Add a New Element

```php
echo 'Adding \'weight\' key with value 70', PHP_EOL;
// Add a new element
$assoc_array['weight'] = 70;
echo 'weight: ' . $assoc_array['weight'], PHP_EOL;
```

When adding new elements to the associative arrays, square bracket notation is used. If the given key already exists in the associative array, it will be overwritten. In our example, we add a new key named “weight” with the value of 70 to our associative array, then print it.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"
PART 3: ADD A NEW ELEMENT ---------------------------
Adding 'weight' key with value 70
weight: 70
```

4) Remove an Element

```php
echo 'Removing \'isUndergraduate\' key', PHP_EOL;
// Remove an element
unset($assoc_array['isUndergraduate']);
echo 'isUndergraduate: ' . $assoc_array['isUndergraduate'], PHP_EOL;
```

For removing an element from an associative array, unset() function is used. This function is actually designed for destroying specified variables but overloaded for array elements anyway. For arrays, unset() takes a list of array elements, then it removes the specified elements from the array. In our example, we remove the "isUndergraduate" key from our associative array.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"

PART 4: REMOVE AN ELEMENT ----------------------------
Removing 'isUndergraduate' key

Warning: Undefined array key "isUndergraduate"
isUndergraduate:
```

5) Modify the Value of an Existing Element

```php
echo 'Value of the \'age\' key before: ' . $assoc_array['age'], PHP_EOL;
echo 'Modifying the value of the \'age\' key', PHP_EOL;
// Modify the value of an existing element
$assoc_array['age'] = 22;
echo 'Value of the \'age\' key after: ' . $assoc_array['age'], PHP_EOL;
```

To modify the value of an existing element in an associative array, square bracket notation is again used. If the given key does not exist in the associative array, it will be created with the given value. In our example, we update the value of the "age" key to 22.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"
PART 5: MODIFY THE VALUE OF AN EXISTING ELEMENT ------
Value of the 'age' key before: 21
Modifying the value of the 'age' key
Value of the 'age' key after: 22
```

6) Search for the Existence of a Key

```php
// Search for the existence of a key
if (array_key_exists('name', $assoc_array)) {
    echo '\'name\' key exists', PHP_EOL;
} else {
    echo '\'name\' key does not exist', PHP_EOL;
}

if (array_key_exists('isUndergraduate', $assoc_array)) {
    echo '\'isUndergraduate\' key exists', PHP_EOL;
} else {
    echo '\'isUndergraduate\' key does not exist', PHP_EOL;
}
```

In order to check if the given key exists in the associative array or not, PHP provides the array_key_exists() function. It takes the desired key and the array as parameters and returns a boolean value depending on the existence of the key. In our example, we check if the “name” and “isUndergraduate” keys exist in our associative array. It is also possible to use the isset() function to check for the existence of a key, but isset() does not return “true” for array keys that correspond to a “null” value, while array_key_exists() does. Therefore array_key_exists() is the recommended way to achieve such a task.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"
PART 6: SEARCH FOR THE EXISTENCE OF A KEY ----------
'name' key exists
'isUndergraduate' key does not exist
```

7) Search for the Existence of a Value

```php
if (in_array(22003021, $assoc_array)) {
    echo '22003021 value exists in one of the keys', PHP_EOL;
} else {
    echo '22003021 value does not exist in one of the keys', PHP_EOL;
}

if (in_array('Basketball', $assoc_array)) {
    echo 'Basketball value exists in one of the keys', PHP_EOL;
} else {
    echo 'Basketball value does not exist in one of the keys', PHP_EOL;
}
```

To search for the existence of a value in an associative or indexed array, PHP offers a function called in_array(). It takes two parameters. The first one is the value which is going to be searched in the array, and the second one is the array itself. Depending

on the existence of the given value, in_array() returns a boolean value. In our example, we check if the “22003021” and “Basketball” values exist in our associative array.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"
PART 7: SEARCH FOR THE EXISTENCE OF A VALUE ---------
22003021 value exists in one of the keys
Basketball value does not exist in one of the keys
```

8) Loop Through an Associative Array, Apply a Function, Called Foo, Which Simply Prints the Key-Value Pair

```php
// A function to print the given key-value pair
function foo($key, $value) {
    echo "$key => $value", PHP_EOL;
}

// ...

// Loop through an associative array, apply a function, called foo, which
simply prints the key-value pair
foreach($assoc_array as $key => $value) {
    foo($key, $value);
}
```

PHP’s syntax for defining functions is very similar to Javascript and Lua. Function definitions start with the “function” keyword and are followed by a function name, parameters, and a function body. And for iterating over an associative array, “foreach” is used. In each iteration, we assign the elements of a pair to “$key” and “$value” variables using the syntax given above. In our example, we traverse over the associative array. For each iteration, we call the “foo” function, which takes “$key” and “$value” variables as a parameter and then prints them. As expected, key-value pairs get printed according to the insertion order.

```
friberk@Berk-MacBook-Pro  hw1 ❯ php "cakar_berk.php"
PART 8: PRINT THE ASSOCIATIVE ARRAY VIA THE FOO FUNCTION
name => Berk Çakar
university => Bilkent University
department => Computer Engineering
universityIdNo => 22003021
age => 22
height => 1.82
weight => 70
```

e) Python

1) Initialize

```python
# Initialize the associative array, it is called as dictionary in Python
dict = {
    'name': 'Berk Çakar',
    'university': 'Bilkent University',
    'department': 'Computer Engineering',
    'universityIdNo': 22003021,
    'isUndergraduate': True,
    'age': 21,
    'height': 1.82
}

print(dict)
print('Associative array has been initialized successfully')
```

In Python, the closest equivalent of an associative array is called “dictionaries”. Dictionaries were unordered data structures until Python 3.7. But, as of now, we do not have an alternative to obtain unordered associative array-like data structures. Therefore, we will be using the dictionary data structure. To initialize a dictionary in Python, “{}” is used. In our example above, we use curly braces syntax to indicate that the given expression is a dictionary. Then, inside the curly braces, we put our desired key-value pairs to be found in our dictionary. Lastly, we print the contents of the dictionary to ensure that it has been initialized successfully.

```
friberk@Berk-MacBook-Pro  hw1 ❯ python3 "cakar_berk.py"
PART 1: INITIALIZE ------------------------------------------------------
{'name': 'Berk Çakar', 'university': 'Bilkent University', 'department':
 'Computer Engineering', 'universityIdNo': 22003021, 'isUndergraduate':
True, 'age': 21, 'height': 1.82}
Associative array has been initialized successfully
```

2) Get the Value for a Given Key

```python
# Get the value for the given key
print(dict['name'])
print(dict['university'])
print(dict['department'])
print(dict['universityIdNo'])
print(dict['isUndergraduate'])
print(dict['age'])
print(dict['height'])
```

To get the value for a given key, Python offers two alternatives. The first of them is to use the regular square bracket notation. In that case, if the given key does not exist in the dictionary, it will raise a “KeyError”, and the execution will stop if the error is not handled. The second alternative is to use the get() function. This function takes the name of the key as a parameter and returns its value if the key exists in the dictionary. If the key does not exist in the dictionary, it does not raise an error but simply returns a “None” type object (i.e., dict.get(‘test’) will return None, in our case). In our example above, we retrieve and print every available value in our dictionary.

```
friberk@Berk-MacBook-Pro  hw1 ❯ python3 "cakar_berk.py"
PART 2: GET THE VALUE FOR THE GIVEN KEY ----------------
Berk Çakar
Bilkent University
Computer Engineering
22003021
True
21
1.82
```

3) Add a New Element

```
print('Adding \'weight\' key with value 70');
# Add a new element
dict['weight'] = 70
print('weight: {}'.format(dict['weight']))
```

Unlike lists and sets, append(), add(), or insert() functions do not work with dictionaries. Again, Python offers two ways to add a new element to a dictionary. The first way is to use the already familiar square bracket notation. And the second way is to use the update() function. We can add single or multiple key-value pairs in a single line with the update() function. For example, dict.update({‘weight’: 70, ‘test’: true}) will add both the ‘weight’ and ‘test’ keys at the same time. However, in both two ways, if the given key(s) exists in the dictionary, their values will be overwritten. Otherwise, the key-value pair(s) will be added to the dictionary. In our example, we add a new key named “weight” with a value of 70 to our associative dictionary using the square bracket syntax, then print it.

```
friberk@Berk-MacBook-Pro  hw1 ❯ python3 "cakar_berk.py"
PART 3: ADD A NEW ELEMENT ------------------------------
Adding 'weight' key with value 70
weight: 70
```

4) Remove an Element

```
print('Removing \'isUndergraduate\' key')
# Remove an element
del dict['isUndergraduate']
print('isUndergraduate: {}'.format(dict.get('isUndergraduate')))
```

To remove an element from a dictionary, Python again offers two solutions. The first solution is to use the generic “del” keyword. The second solution is to use the pop() function. They are similar in a sense because they both throw an error if the given key does not exist in the dictionary. However, pop() will return the value of the deleted key, whereas using “del” keyword does not (i.e., x = dict.pop(‘weight’) # x becomes 70). In our example, we remove the “isUndergraduate” key from our dictionary using the “del” keyword.

```
friberk@Berk-MacBook-Pro  hw1 ❯ python3 "cakar_berk.py"
PART 4: REMOVE AN ELEMENT ------------------------------
Removing 'isUndergraduate' key
isUndergraduate: None
```

5) Modify the Value of an Existing Element

```
print('Value of the \'age\' key before: {}'.format(dict['age']))
print('Modifying the value of the \'age\' key')
# Modify the value of an existing element
dict['age'] = 22;
print('Value of the \'age\' key after: {}'.format(dict['age']));
```

Similar to adding a new element, we can modify the value of an existing element in a dictionary using both square bracket syntax and the update() function. In both methods, if the given key does not exist in the dictionary, it will be created with the given value. In our example, we update the value of the “age” key to 22 using square bracket syntax.

```
friberk@Berk-MacBook-Pro  hw1 ❯ python3 "cakar_berk.py"
PART 5: MODIFY THE VALUE OF AN EXISTING ELEMENT --------
Value of the 'age' key before: 21
Modifying the value of the 'age' key
Value of the 'age' key after: 22
```

6) Search for the Existence of a Key

```
# Search for the existence of a key
if 'name' in dict:
    print('\'name\' key exists')
else:
    print('\'name\' key does not exist')

if 'isUndergraduate' in dict:
    print('\'isUndergraduate\' key exists')
else:
    print('\'isUndergraduate\' key does not exist')
```

To check if the given key exists in the dictionary or not, Python's "in" operator can be used. Actually, in Python 2, there was a has_key() function, but its functionality for dictionaries was replaced with the "in" operator in Python 3. Depending on the existence of the key, the "in" operator returns a boolean value. In our example, we check if the "name" and "isUndergraduate" keys exist in our dictionary.

```
friberk@Berk-MacBook-Pro  hw1 ❯ python3 "cakar_berk.py"
PART 6: SEARCH FOR THE EXISTENCE OF A KEY ---------------
'name' key exists
'isUndergraduate' key does not exist
```

7) Search for the Existence of a Value

```
# Search for the existence of a value
if 22003021 in dict.values():
    print('22003021 value exists in one of the keys');
else:
    print('22003021 value does not exist in one of the keys');

if 'Basketball' in dict.values():
    print('Basketball value exists in one of the keys')
else:
    print('Basketball value does not exist in one of the keys')
```

Similar to previous part, "in" operator is still used to check for an existence. However, this time we need to use "in" operator to search in the dictionary's values. For that, we can use the values() function to gather the values of a dictionary. Then, depending on the existence of the desired value in dict.values() "in" operator returns true or false. In our example, we check if the "22003021" and "Basketball" values exist in our dictionary.

8) Loop Through an Associative Array, Apply a Function, Called Foo, Which Simply Prints the Key-Value Pair

```python
# A function to print the given key-value pair
def foo(key, value):
    print('{}: {}'.format(key, value))

# ...

# Loop through an associative array, apply a function, called foo, which
simply prints the key-value pair
for key in dict:
    foo(key, dict[key])
```

In Python, function definitions start with the “def” keyword. Then, “def” is followed by the function name and parameters. The function body is specified after a semicolon and an indented block. And for iterating over a dictionary, again, the “in” keyword is used. In our example, we traverse over our dictionary. For each iteration, we call the “foo” function, which takes “key” and “value” variables as a parameter and then prints them. As expected, we print the key-value pairs in insertion order.

```
friberk@Berk-MacBook-Pro  hw1 ❯ python3 "cakar_berk.py"
PART 8: PRINT THE ASSOCIATIVE ARRAY VIA THE FOO FUNCTION
name: Berk Çakar
university: Bilkent University
department: Computer Engineering
universityIdNo: 22003021
age: 22
height: 1.82
weight: 70
```

f) Ruby

1) Initialize

```ruby
# Initialize the associative array, it is called as a hash in Ruby
hash = {
    name: 'Berk Çakar',
    university: 'Bilkent University',
    department: 'Computer Engineering',
    universityIdNo: 22003021,
    isUndergraduate: true,
    age: 21,
    height: 1.82
}
```

```
puts hash
puts "Associative array has been initialized successfully"
```

In Ruby, an associative array is called a hash. In recent versions of Ruby, hashes are ordered by the insertion order. However, there are not any unordered alternatives for hashes. Therefore, we will be evaluating the hash data structure. To initialize a hash in Ruby, "{}" is used. In our example above, we use curly braces to indicate that the given expression is a hash. Then, inside the curly braces, we put our desired key-value pairs to be found in our hash. Lastly, we print the hash's contents to ensure that it has been initialized successfully.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 1: INITIALIZE ----------------------------------------------------
{:name=>"Berk Çakar", :university=>"Bilkent University", :department=
>"Computer Engineering", :universityIdNo=>22003021, :isUndergraduate=
>true, :age=>21, :height=>1.82}
Associative array has been initialized successfully
```

2) Get the Value for a Given Key

```
# Get the value for the given key
puts hash[:name]
puts hash[:university]
puts hash[:department]
puts hash[:universityIdNo]
puts hash[:isUndergraduate]
puts hash[:age]
puts hash[:height]
```

To get a value for a given key in a Ruby hash, square bracket syntax can be used. Inside the square brackets, the key can be specified either starting with a ":" followed by the name of the key (like in the example above) or using a string literal of the key's name (i.e., hash['name']). If the given key does not exist in the array, using this syntax will result in continuing the execution. It will return a "nil" value without giving any error. However, another alternative is to use the fetch() function. This function takes the name of the key as a parameter and returns its value (i.e., hash.fetch('name')). If the given key does not exist in the array, fetch() will raise an error. In our example above, we retrieve and print every available value in our hash.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 2: GET THE VALUE FOR THE GIVEN KEY ----------
Berk Çakar
Bilkent University
Computer Engineering
22003021
true
21
1.82
```

3) Add a New Element

```ruby
puts "Adding 'weight' key with value 70"
# Add a new element
hash[:weight] = 70
puts "weight: #{hash[:weight]}"
```

In Ruby, most common way to add a new element into a hash is to use the square bracket notation. If the given key already exists in the hash, its value will be overwritten. Otherwise, the new key will be added into our hash with the given value. This task can also be achieved by using functions such as merge() and store() (i.e., hash.store("weight", 70), hash.merge("weight" => 70)). In our example, we add a new key named "weight" with a value of 70 to our hash using the square bracket syntax, then print it.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 3: ADD A NEW ELEMENT ------------------------
Adding 'weight' key with value 70
weight: 70
```

4) Remove an Element

```ruby
puts "Removing 'isUndergraduate' key"
# Remove an element
hash.delete(:isUndergraduate)
puts "isUndergraduate: #{hash[:isUndergraduate]}"
```

In order to remove an element from a hash, delete() function is used. It takes the key to be deleted as a parameter, and returns the value of the deleted key. If the given key does not exist in the hash, it returns a "nil" value. In our example, we remove the "isUndergraduate" key from the hash.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 4: REMOVE AN ELEMENT ----------------------------
Removing 'isUndergraduate' key
isUndergraduate:
```

5) Modify the Value of an Existing Element

```ruby
puts "Value of the 'age' key before: #{hash[:age]}"
puts "Modifying the value of the 'age' key"
# Modify the value of an existing element
hash[:age] = 22
puts "Value of the 'age' key after: #{hash[:age]}"
```

Similar to adding a new element, in Ruby, we can modify the value of an existing element in a hash using both square bracket syntax as well as the update() and merge() functions. In both methods, if the given key does not exist in the hash, it will be created with the given value. In our example, we update the value of the "age" key to 22 using square bracket syntax.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 5: MODIFY THE VALUE OF AN EXISTING ELEMENT -----
Value of the 'age' key before: 21
Modifying the value of the 'age' key
Value of the 'age' key after: 22
```

6) Search for the Existence of a Key

```ruby
# Search for the existence of a key
if hash.has_key?(:name)
    puts "'name' key exists"
else
    puts "'name' key does not exist"
end

if hash.has_key?(:isUndergraduate)
    puts "'isUndergraduate' key exists"
else
    puts "'isUndergraduate' key does not exist"
end
```

In order to check if the given key exists in the associative array or not, Ruby provides the has_key?() function. It takes the desired key as a parameter and returns a boolean value depending on the existence of the key. In our example, we check if the "name" and "isUndergraduate" keys exist in our hash.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 6: SEARCH FOR THE EXISTENCE OF A KEY ----------
'name' key exists
'isUndergraduate' key does not exist
```

7) Search for the Existence of a Value

```ruby
# Search for the existence of a value
if hash.has_value?(22003021)
    puts "22003021 value exists in one of the keys"
else
    puts "22003021 value does not exist in one of the keys"
end

if hash.has_value?("Basketball")
    puts "Basketball value exists in one of the keys"
else
    puts "Basketball value does not exist in one of the keys"
end
```

To search for the existence of a value in hash, Ruby offers a function called has_value?(). It work similar to the has_key?() function. It takes the value which is going to be searched in the has as a parameter. Depending on the existence of the given value, has_value?() returns a boolean value. In our example, we check if the "22003021" and "Basketball" values exist in our hash.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 7: SEARCH FOR THE EXISTENCE OF A VALUE ----------
22003021 value exists in one of the keys
Basketball value does not exist in one of the keys
```

8) Loop Through an Associative Array, Apply a Function, Called Foo, Which Simply Prints the Key-Value Pair

```ruby
# A function to print the given key-value pair
def foo(key, value)
    puts "#{key}: #{value}"
end

# ...

# Loop through an associative array, apply a function, called foo, which
simply prints the key-value pair
hash.each do |key, value|
    foo(key, value)
end
```

In the Ruby programming language, defining a function is a mixture between Python and Lua. Like Python, they start with the “def” keyword, and, like Lua, they end with the “end” keyword. However, unlike Python, the indentation of the function body is not required. And for iterating over a hash, hash.each iterator is used. Then, in each iteration, the components of each hash pair are assigned to “key” and “value” variables. In our example, we traverse over our hash. For each iteration, we call the “foo” function, which takes “key” and “value” variables as a parameter and then prints them. As expected, we print the key-value pairs in insertion order.

```
friberk@Berk-MacBook-Pro  hw1 ❯ ruby "cakar_berk.rb"
PART 8: PRINT THE ASSOCIATIVE ARRAY VIA THE FOO FUNCTION
name: Berk Çakar
university: Bilkent University
department: Computer Engineering
universityIdNo: 22003021
age: 22
height: 1.82
weight: 70
```

g) Rust

1) Initialize

```rust
// Initialize the associative array, it is called as a HashMap
// in Rust
let mut hashMap = HashMap::from([
    ("name", "Berk Çakar"),
    ("university", "Bilkent University"),
    ("department", "Computer Engineering"),
    ("universityIdNo", "22003021"),
    ("isUndergraduate", "true"),
    ("age", "21"),
    ("height", "1.82")
]);

println!("{:?}", hashMap);
println!("Associative array has been initialized successfully");
```

In Rust, the equivalent of an unordered associated array is a “HashMap”. This data structure is available after importing the “std::collections::HashMap” library, and it is unordered in parallel to our expectations. However, it does not support non-homogenous values without going through complex workarounds, such as deriving a new union type using enumerators. To initialize a hash map in Rust, the syntax

given above can be used. “from()” function acts as the constructor for the HashMap class, and the way key-value pairs are specified is also different from the other languages we covered. The constructor takes an array of tuples, where the tuples contain key-value pairs. Lastly, we print the contents of the hash map to ensure that it has been initialized successfully.

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 1: INITIALIZE ------------------------------------------------------------
{"university": "Bilkent University", "universityIdNo": "22003021", "isUndergraduate"
: "true", "age": "21", "name": "Berk Çakar", "department": "Computer Engineering", "
height": "1.82"}
Associative array has been initialized successfully
```

2) Get the Value for a Given Key

```
    // Get the value for the given key
    println!("{}", hashMap["name"]);
    println!("{}", hashMap["university"]);
    println!("{}", hashMap["department"]);
    println!("{}", hashMap["universityIdNo"]);
    println!("{}", hashMap["isUndergraduate"]);
    println!("{}", hashMap["age"]);
    println!("{}", hashMap["height"]);
```

To get the value for a given key, Rust offers two alternatives. The first of them is to use the regular square bracket notation. In that case, if the given key does not exist in the hash map, it will give an error and the execution will stop no matter what. The second alternative is to use the get() function. This function takes the name of the key as a parameter and returns its value if the key exists in the hash map. If the key does not exist in the hash map, get() function returns a “None” value (i.e., hashMap.get(“test”) will return None, in our case). In our example above, we retrieve and print every available value in our hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 2: GET THE VALUE FOR THE GIVEN KEY ---------------------------------
Berk Çakar
Bilkent University
Computer Engineering
22003021
true
21
1.82
```

3) Add a New Element

```
println!("Adding 'weight' key with value 70");
// Add a new element
hashMap.insert("weight", "70");
println!("weight: {}", hashMap["weight"]);
```

For adding a new element to a hash map, the insert() function is used. It takes the key-value pair to be added in two parameters. If the given key already exists in the hash map, its value will be overwritten. In our example, we add a new key named "weight" with value of 70 to our hash map, then print it.

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 3: ADD A NEW ELEMENT ------------------------------------------
Adding 'weight' key with value 70
weight: 70
```

4) Remove an Element

```
println!("Removing 'isUndergraduate' key");
// Remove an element
hashMap.remove("isUndergraduate");
println!("isUndergraduate: {:?}", hashMap.get("isUndergraduate"));
```

In order to remove an element from a hash map, remove() function is used. It takes the key to be deleted as a parameter, and returns the value of the deleted key. If the given key does not exist in the hash map, it returns a "None" value. In our example, we remove the "isUndergraduate" key from the hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 4: REMOVE AN ELEMENT ------------------------------------------
Removing 'isUndergraduate' key
isUndergraduate: None
```

5) Modify the Value of an Existing Element

```
println!("Value of the 'age' key before: {}", hashMap["age"]);
println!("Modifying the value of the 'age' key");
// Modify the value of an existing element
hashMap.insert("age", "22");
println!("Value of the 'age' key after: {}", hashMap["age"]);
```

For modifying a value in a hash map, insert() function is used. It takes the key-value pair to be modified in two parameters. If the given key does not exist in the hash map, it will be created with the given value. In our example, we update the value of the “age” key to 22.

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 5: MODIFY THE VALUE OF AN EXISTING ELEMENT ----------------------
Value of the 'age' key before: 21
Modifying the value of the 'age' key
Value of the 'age' key after: 22
```

6) Search for the Existence of a Key

```
// Search for the existence of a key
if hashMap.contains_key("name") {
    println!("'name' key exists");
} else {
    println!("'name' key does not exist");
}

if hashMap.contains_key("isUndergraduate") {
    println!("'isUndergraduate' key exists");
} else {
    println!("'isUndergraduate' key does not exist");
}
```

In order to check if the given key exists in the hash map or not, Rust provides the contains_key() function. It takes the desired key as a parameter and returns a boolean value depending on the existence of the key. In our example, we check if the “name” and “isUndergraduate” keys exist in our hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 6: SEARCH FOR THE EXISTENCE OF A KEY ----------------------------
'name' key exists
'isUndergraduate' key does not exist
```

7) Search for the Existence of a Value

```
if hashMap.values().any(|&value| value == "22003021") {
    println!("22003021 value exists in one of the keys");
} else {
    println!("22003021 value does not exist in one of the keys");
}

if hashMap.values().any(|&value| value == "Basketball") {
```

```
        println!("Basketball value exists in one of the keys");
    } else {
        println!("Basketball value does not exist in one of the keys");
    }
```

Although Rust does not offer a built-in solution to perform such a task, we can still implement an elegant solution. First of all, we can retrieve an iterator for the values of our hash map using values() function. Then, using any() function, we can traverse over the hash map's values in just one line. The any() function takes a parameter indicating which condition traversal will be ended. In our first search, we want any of the values to be "22003021". Therefore, we pass "value == "22003021" to any() function. Accordingly, any() function will traverse over the values of our hash map, and whenever it encounters "22003021", it will return true and end the traversal. Otherwise, if it does not encounter any value that matches the given parameter, it will return false. Overall, in our example, we check if the "22003021" and "Basketball" values exist in our hash map.

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 7: SEARCH FOR THE EXISTENCE OF A VALUE ------------------------
22003021 value exists in one of the keys
Basketball value does not exist in one of the keys
```

8) Loop Through an Associative Array, Apply a Function, Called Foo, Which Simply Prints the Key-Value Pair

```
// A function to print the given key-value pair
fn foo(key: &str, value: &str) {
    println!("{}: {}", key, value);
}
fn main () {
    // ...

    // Loop through an associative array, apply a function, called foo,
    // which simply prints the key-value pair
    for (key, value) in hashMap {
        foo(key, value);
    }
}
```

Rust's syntax for defining functions is very similar to Javascript. The main differences Rust function definitions start with the "fn" keyword instead of "function", and we can specify data types for parameters in Rust, whereas Javascript

uses dynamic typing. And for iterating over a hash map, the “in” keyword can be used. In each iteration, we assign the elements of a pair to “key” and “value” variables using the syntax given above. In our example, we traverse over our hash map. For each iteration, we call the “foo” function, which takes “key” and “value” variables as a parameter and then prints them. Since the hash map data structure is unordered in Rust, it is expected to get key-value pairs in a random order (i.e., in a different order than what is given in the initialization phase).

```
friberk@Berk-MacBook-Pro  hw1 ❯ rustc "cakar_berk.rs" && ./cakar_berk
PART 8: PRINT THE ASSOCIATIVE ARRAY VIA THE FOO FUNCTION ------------
height: 1.82
university: Bilkent University
universityIdNo: 22003021
name: Berk Çakar
age: 22
department: Computer Engineering
weight: 70
```

Evaluation

After comprehending the associative array concepts in seven different programming languages, I think Dart is the best option for associative arrays. This is because Dart is the only language that offers both ordered and unordered associative array types. Also, the language is feature-loaded, and its syntax is very similar to C/C++/Java (C-family programming languages); therefore, it has a good level of readability and writability. Moreover, for map operations such as adding a new element, modifying an existing element, and searching for an element, Dart offers practical functions like addAll() and update() with intuitive names to increase readability and writability. Lastly, for associative arrays, square bracket notation (subscript syntax) is still optimized and valid, improving readability and writability for those already familiar with C-family programming languages' arrays. Regarding reliability, I think Dart is also a great alternative since, depending on the constructors used, the language can perform both compile time and runtime type checks for the contents of the associative arrays. For example, if we limit the hash map to <String, Int> and use a boolean value for a value, an error will be raised. The problem in Lua is that although the readability and writability of the language are not bad, the "pairs(table)" syntax is used when traversing associative arrays, and there is no easy method to read and write when searching for a value in associative arrays. Although PHP is a good alternative to Dart in terms of readability and writability, it is less reliable for associative arrays

because it does not have unordered data structures and only supports dynamic typing. I think Python and Ruby are second in terms of readability and writability after Dart. They provide all the necessary functions to use in the associative array operations, and the overall syntax is easy to follow. Still, the main problems with these are the support for only dynamic typing and the lack of variance for ordered/unordered versions for an associative array. Lastly, for me, the worst language to use for associative arrays is Rust. The language can be efficient for low-level programming, but it has a very complex syntax which reduces the overall readability and writability. And in Rust, creating a non-homogenous associative array is complicated, which also minimizes the writability of the language.

Learning Strategy

In general, I referred to the official documentations of the given seven languages while doing the assignment. I went through the data structures sections of these documentations to see if an official equivalent exists for unordered associated arrays. Whenever I did not encounter an equivalent, or if the information needed to be more useful, I performed detailed searches over the internet. Then, I filtered the results of these searches to gather reliable sources. In other words, in addition to official documentations, I also referred to credible tech blogs and Q&A sites such as StackOverflow. During the experiments with associative arrays in seven different programming languages, I did not use any online compilers/interpreters since I had all the required compilers/interpreters installed on my computer. However, whenever possible, I ran my codes on the Dijkstra machine as well to ensure that there were no platform-related issues. The resources I used for each language are as follows:

Dart:

- https://api.dart.dev/stable/2.18.4/dart-core/Map-class.html
- https://api.dart.dev/stable/2.18.4/dart-collection/HashMap-class.html
- https://www.bezkoder.com/dart-map/
- https://www.tutorialspoint.com/dart_programming/dart_programming_map.htm
- https://stackoverflow.com/questions/14930950/what-are-the-differences-between-the-different-map-implementations-in-dart
- https://stackoverflow.com/questions/66952993/dart-hashmap-initial-value

Javascript:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map
- https://flexiple.com/javascript/associative-array-javascript/
- https://stackoverflow.com/questions/37515959/how-to-create-an-associative-array-in-javascript-literal-notation
- https://stackoverflow.com/questions/38339531/creating-some-associative-array-in-javascript
- https://stackoverflow.com/questions/5525795/does-javascript-guarantee-object-property-order

Lua:

- https://www.lua.org/pil/2.5.html
- http://lua-users.org/wiki/TablesTutorial
- https://www.tutorialspoint.com/lua/lua_tables.htm
- https://stackoverflow.com/questions/30970034/lua-in-pairs-with-same-order-as-its-written

PHP:

- https://www.php.net/manual/en/language.types.array.php
- https://www.geeksforgeeks.org/associative-arrays-in-php/
- https://www.w3schools.com/php/php_arrays_associative.asp
- https://stackoverflow.com/questions/10914730/are-php-associative-arrays-ordered
- https://stackoverflow.com/questions/11487374/does-php-conserve-order-in-associative-array

Python:

- https://docs.python.org/3/tutorial/datastructures.html#dictionaries
- https://www.geeksforgeeks.org/python-dictionary/
- https://snakify.org/en/lessons/dictionaries_dicts/
- https://www.w3schools.com/python/python_dictionaries.asp
- https://realpython.com/python-dicts/
- https://python-course.eu/python-tutorial/dictionaries.php

- https://stackoverflow.com/questions/39980323/are-dictionaries-ordered-in-python-3-6
- https://medium.com/junior-dev/python-dictionaries-are-ordered-now-but-how-and-why-5d5a40ee327f

Ruby:

- https://ruby-doc.org/core-3.1.2/Hash.html
- https://www.tutorialspoint.com/ruby/ruby_hashes.htm
- https://launchschool.com/books/ruby/read/hashes
- https://www.markhneedham.com/blog/2010/09/07/ruby-hash-ordering/
- https://stackoverflow.com/questions/31418673/is-order-of-a-ruby-hash-literal-guaranteed

Rust:

- https://doc.rust-lang.org/std/collections/index.html
- https://doc.rust-lang.org/std/collections/hash_map/struct.HashMap.html
- https://web.mit.edu/rust-lang_v1.25/arch/amd64_ubuntu1404/share/doc/rust/html/book/second-edition/ch08-03-hash-maps.html
- https://www.geeksforgeeks.org/rust-hashmaps/
- https://practice.rs/collections/hashmap.html
- https://simonewebdesign.it/rust-hashmap-insert-values-multiple-types/
- https://users.rust-lang.org/t/associative-array-in-rust/70625
- https://stackoverflow.com/questions/26498704/creating-an-associative-array-in-rust
- https://stackoverflow.com/questions/26498704/creating-an-associative-array-in-rust
- https://stackoverflow.com/questions/30243100/how-do-i-sort-a-map-by-order-of-insertion