CS315 - Programming Languages

Homework 1 Report

Associative Arrays in Various Languages

İrem Seven

21704269

Table of Contents

1 Associative Arrays in Different Languages

In this section, associative array examples are given for 7 different languages. To show the difference more clearly the example of each language is the same or very similar to each other. Code segments are explained and the related outputs are given right after each code segment.

1.1 Dart

In Dart, associative arrays are named as map which is a dynamic data structure.

Below it is given the foo function to be called later when we want to print a pair. It takes a String key value and a Map which is the associative array of Dart language.

```
foo(String k, Map m) {
  print("Key : $k, value : ${m[k]}");
}
```

A map can be created and initialized as below. The values on left of each : are keys and on right of each : are the corresponding values for the keys.

```
  //initialize
  var arr = {'Name':'John', 'City':'Rome', 'Age': 28, 'Height': 182};
```

We can access the map elements by using subscript operation just like ordinary arrays, without indexing.

```
  print(arr["Name"]);   //get the value for a given key
```

```
John
```

Dart language does not give an error when we want to access a nonexistent key value. In this case, it gives null value.

```
 print(arr["Salary"]);   //get the value for nonexistent key
```

```
null
```

We can print the initial version of the array as below. In Dart, we can just provide the map name in print function to print the map.

```
  print('**Initial Version**');
  print(arr);   //print initial version
```

```
**Initial Version**
{Name: John, City: Rome, Age: 28, Height: 182}
```

Addition can be done with the subscript operator.

```
  arr['Weight'] = 79;   //add new element
```

We can remove a key with its value by using remove(key); method. In Dart, if we try to delete a nonexistent key value we will not get any error, it will have no effect to the map structure.

```
  arr.remove('Height');   //remove an element
 arr.remove('Salary');    //remove nonexistent element
```

We can change an element's value using the subscript operator as below.

```
  arr['City'] = 'Seattle';   //change value of a key
```

Dart has a function containsKey(key); to learn whether a key exists in the map or not.

```
  print(arr.containsKey('Name'));   //search for existance of key
true
```

Similarly, to search for the existence of a value Dart provides containsValue(value); function.

```
  print(arr.containsValue('John'));   //search for existance of value
true
```

Below it is shown how to print pairs in the map by using a loop. Inside the loop the given foo function is called. The output is given below to see the effects of the operations we have done before.

```
print('\n**Modified Version**');
  //printing by using function foo in loop
  for (var k in arr.keys) {
      foo(k,arr);
  }
**Modified Version**
Key : Name, value : John
Key : City, value : Seattle
Key : Age, value : 28
Key : Weight, value : 79
```

In Dart, we can also print keys and values easily by using the .keys and .values methods.

```
  print(arr.keys);    //printing keys
  print(arr.values);   //printing values
```

```
(Name, City, Age, Weight)
(John, Seattle, 28, 79)
```

1.2 Javascript

In Javascript, associative arrays are simply dynamic objects. When we assign values to keys in an array variable it is automatically converted into an object. However, this condition causes the variable to lose the attributes and methods of arrays. Since the associative arrays are objects attributes are also keys. There are various ways to construct an associative array. Below it is given one of the simplest ways to create an associative array and perform operations.

This is the foo function to be called when we want to print a pair. Since in Javascript the variables outside functions are treated as global variables it only takes key as parameter. The function will be used in the last part of the example sample codes.

```
//function foo to print a pair
function foo(k) {
    document.write("Key: " + key + " Value: " + arr[key] + '<br>');
}
```

We can create and initialize an associative array as given below. The values on left of each : are keys and on right of each : are the corresponding values for the keys. After creating the array, it is shown a way to print the array elements. Unlike Dart and Python, Javascript does not provide a direct way to print array elements. This means that the print should be done by iterating over loop.

```
// initialize array
var arr = { "Name": "Harris", "City": "Los Angeles", "Age": 42, "Salary": "5420$"};
//print initial form
document.write("**Initial Associative Array**" + '<br>');
for(var key in arr)
{
    document.write("Key: " + key + " Value: " + arr[key] + '<br>');
}
```

```
**Initial Associative Array**
Key: Name Value: Harris
```

```
Key: City Value: Los Angeles
Key: Age Value: 42
Key: Salary Value: 5420$
```

We access a key value by using the subscript operator. In Javascript, if we try to access a nonexistent key value, we will not get any error. Instead the value will be shown as undefined.

```
//get value for key age
document.write('<br>' + "Value for key Age: " + arr["Age"] + '<br>');
//get value for nonexistent key
document.write('<br>' + "Value for key Car: " + arr["Car"] + '<br>');
```

```
Value for key Age: 42

Value for key Car: undefined
```

We can add a new item by using the subscript operator. Also, since the associative arrays are also objects we can add items by object notation. However, in this case the stored key value will no longer be a string.

```
arr["Country"] = "USA";          //add a new element
arr.height = 184;                //we can also add by using object notation
```

We can remove an element by using “delete” as below.

```
delete arr['Age'];               //remove an element
delete arr['Car'];          //remove nonexistent element
```

We can modify a value by using the subscript operator. Please note that if the key value does not exist, this operation will result in the addition of an element.

```
arr["City"] = "New York";        //modify a value
```

We can search a key value by using “in” notation. It will result in boolean. Below it is given an example usage.

```
//search existence of Key Salary
keybool = 'Salary' in arr;
document.write("Existence of Key Salary: " + keybool + '<br>');
```

```
Existence of Key Salary: true
```

Unlike keys, value search is more complicated. There is no built in method to search for a value on an object. We should iterate in each array and need to compare the values with the searched value. In the first case the 42 value exists and in the second case the Harris value does not exist.

```
//search existence of Value 42
```

```
valbool = false
for (var prop in arr) {
    if (arr.hasOwnProperty(prop) && arr[prop] === 42) {
        valbool = true
    }
}
document.write("Existence of Value 42: " + valbool + '<br>');
//search existence of Value Harris
for (var prop in arr) {
    if (arr.hasOwnProperty(prop) && arr[prop] === "Harris") {
        valbool = true
    }
}
document.write("Existence of Value Harris: " + valbool + '<br>');
```

```
Existence of Value 42: false
Existence of Value Harris: true
```

After the operations are performed, we can print the modified version of the array by using the foo function in a loop.

```
//print modified version in loop by foo
document.write('<br>' + "**Modified Associative Array**" + '<br>');
for(var key in arr)
{
    foo(key);
}
```

```
**Modified Associative Array**
Key: Name Value: Harris
Key: City Value: New York
Key: Salary Value: 5420$
Key: Country Value: USA
Key: height Value: 184
```

1.3 Lua

In Lua, associative array types are referred to table data structure.

The foo function that will be used later, it prints a single pair of the table.

```
--foo function to print a pair
function foo(k)
    print(k, mytable[k])
end
```

We can initialize a table as below. The values on left of each : are keys and on right of each : are the corresponding values for the keys.

```
--initialize
mytable = {["Name"] = "John", ["City"] = "Rome", ["Age"] = 28, ["Height"] =  182}
```

In Lua we cannot print the table contents by providing the table name to the print function. It will only show the type and unique id of the table object, not the contents.

```
print(mytable)  -- type and unique id of the object, not the contents
table: 0x55eec256c400
```

We can get a key value by just using the subscript operator. If we try to access a nonexistent key's value the result will be given as "nil" which is similar to null value.

```
print("\nName is ", mytable["Name"]) --get value for a key
print("Salary is ", mytable["Salary"])   --get value for nonexistent key
Name is         John
Salary is       nil
```

The initial state of the table is printed as below and the output is given. Unlike other languages when we want to perform print on table elements we cannot be sure in which order they will be printed, since Lua does not ensure the access order of the elements.

```
print("\n**Initial Table**")
for k, v in pairs(mytable) do --print initial by foo
    foo(k, v)
end
**Initial Table**
Height  182
Name    John
Age     28
City    Rome
```

We can add a new element by the subscript operator. Since the Weight does not included in the table below it is added to the table with its value 79.

```
mytable["Weight"] = 79 --add new element
```

We can remove an element in a very simple manner. All we have to do is to assign the related element to nil value. If we try to remove a nonexistent element we will not get any error.

```
mytable["Age"] = nil   --remove an element
mytable["Salary"] = nil    --remove nonexistent element
```

We can modify the value as below.

```
mytable["City"] = "Seattle" --modify value
```

We can search for the existence of a key as below.

```
--search for key Name
if mytable["Name"] ~= nil then print("\nkey Name exists") end
```

Searching for a value requires an iteration. We should check the existence manually as below. Two example cases are given for that purpose.

```
--search value Rome
for k, v in pairs(mytable) do
	if v == "Rome" then print("value Rome exists") end
end
--search value Seattle
for k, v in pairs(mytable) do
	if v == "Seattle" then print("value Seattle exists") end
end
```

```
key Name exists
value Seattle exists
```

Lastly, we can print the modified table by using the foo function in an iteration. The resultant output is given below.

```
print("\n**Table Changed**")
for k, v in pairs(mytable) do --print modified by foo
	foo(k)
end
print("\n")
```

```
**Table Changed**
Height	182
Name	John
City	Seattle
Weight	79
```

1.4 PHP

In PHP, associative arrays do not have a specific name.

The foo function to print a pair of an array. It will be used later.

```
<!DOCTYPE html>
<html>
<body>
```

```
<?php
//foo function to print pairs
function foo($k, $arr) {
  echo "Key=" . $k . ", Value=" . $arr[$k];
}
```

We can create and initialize an associative array as follows.

```
$arr = array("Name"=>"Maria", "Country"=>"Italy", "Age"=>14, "Height"=>154 );
```

The initial array elements can be printed as below. It requires special iteration on key values.

```
//printing initial array
echo "<br>**Initial Array**<br>";
foreach($arr as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
```

```
**Initial Array**
Key=Name, Value=Maria
Key=Country, Value=Italy
Key=Age, Value=14
Key=Height, Value=154
```

We can get the values for a given key by using the subscript operator. If we try to access a nonexistent key value we will not get any error, and we will not get any result for that operation.

```
//getting values for keys Name and Age
echo "<br>" . $arr['Name'] . " is " . $arr['Age'] . " years old.<br>";
//getting value for nonexistent key
echo "Salary is" . $arr['Salary'] . "<br>";
```

```
Maria is 14 years old.
Salary is
```

In PHP we can find the key from a given value. This functionality is not provided in most languages, so mentioning this functionality might be important.

```
//We can find key by value easily in php
echo array_search("Maria",$arr) . "<br>" ;
```

```
Name
```

We can add new elements by subscript operator.

```
//adding new element
$arr['Surname'] = "Brown";
```

To remove an element unset(array[key]); should be used. In the example given below the Height is removed from the array. Car key does not exist in the array, when we try to delete a nonexistent element the operation would not result in any way.

```
//remove an element
unset($arr["Height"]);
//remove nonexistent element
unset($arr["Car"]);
```

We can modify elements by subscript operator.

```
//modify element
$arr['Age'] = 16;
```

PHP provides the “array_key_exists” function to search for the existence of a key. It returns a boolean result. An example usage is as follows.

```
//search for Key Country
$keybool = array_key_exists("Country", $arr);
if ($keybool == 1) {
  echo "Key Country exists";
}
else{
        echo "Key Country does not exist";
}
echo "<br>";

//search for Key Height
$keybool = array_key_exists("Height", $arr);
if ($keybool == 1) {
  echo "Key Height exists";
}
else{
        echo "Key Height does not exist";
}
echo "<br>";
```

```
Key Country exists
Key Height does not exist
```

Similarly, PHP provides the “in_array” function to search for the existence of a value. It returns a boolean result. An example usage is as follows.

```
//search for Value Italy
$valbool = in_array('Italy', $arr);
if ($valbool == 1) {
  echo "Value Italy exists";
}
```

```
else
        echo "Value Italy does not exist";

echo "<br>";
```

```
Value Italy exists
```

We can print the resultant array of the previous operations by using foo function.

```
//printing modified array in loop by foo
echo "<br>**Modified Array**<br>";
foreach($arr as $x => $x_value) {
  foo($x, $arr);
  echo "<br>";
}
?>

</body>
</html>
```

```
**Modified Array**
Key=Name, Value=Maria
Key=Country, Value=Italy
Key=Age, Value=16
Key=Surname, Value=Brown
```

1.5 Python

Python provides dictionary data structure to simulate associative arrays.

The foo function that prints pairs of a dictionary that will be used later.

```
#function foo to print a pair
def foo(k):
    print('Key is: ', k, 'Value is: ', arr[k])
```

We can create and initialize a dictionary as follows.

```
#initialize
arr = {'Name':'Pierre', 'City':'Paris', 'Age': 26, 'Height': 179}
```

We can access a key value by subscript operation. It will give the corresponding value of the Age key.

```
print("Value for Age is: ", arr["Age"])    #get the value for Age key
```

```
Value for Age is:  26
```

In Python, if we try to access a nonexistent element we will get an error as given in the output. Thus, we cannot implement the following line of code since key Salary does not exist in the “arr” dictionary.

```
print("Value for Age is: ", arr["Salary"])  #get value for nonexistent key
Traceback (most recent call last):
File "<string>", line 9, in <module>
KeyError: 'Salary'
```

We can print elements in a dictionary directly by the name as follows.

```
print("**Initial Dictionary**\n", arr)
**Initial Dictionary**
 {'Name': 'Pierre', 'City': 'Paris', 'Age': 26, 'Height': 179}
```

We can add elements by using the subscript operator.

```
arr['Weight'] = 68     #add new element
```

To remove an element we should use the “del” operation as follows.

```
del arr['Age']   #remove an element
```

In Python, as we cannot access elements of nonexistent keys, we also cannot implement the remove function on a nonexistent element. Since the following line of code will give an error.

```
del arr['Salary']   #remove nonexistent element
Traceback (most recent call last):
  File "<string>", line 14, in <module>
KeyError: 'Salary'
```

We can modify elements by the subscript operator.

```
arr['City'] = 'Istanbul' #modify a value
```

Python does not provide a function to search for the existence of key values. However, we can determine it by using simple notation “in” that is widely used in Python language.

```
#search for Age key existance
if 'Age' in arr:
    print('Key Age exist')
else:
    print('Key Age does not exist')
```

```
#search for City key existance
if 'City' in arr:
    print('Key City exist')
else:
    print('Key City does not exist')
```

```
Key Age does not exist
Key City exist
```

Similarly we can search for the existence of a value by using “in”. This time instead of directly using the name of the dictionary we should provide the values of it by using values() method.

```
 #search for Pierre value existance
if 'Pierre' in arr.values():
    print('Value Pierre exist')
else:
    print('Value Pierre does not exist')
```

```
Value Pierre exist
```

After the operations we can print the resultant table pairs by using foo function.

```
print("\n**Modified Dictionary**")
#print modified version in loop by calling foo
for k in arr:
    foo(k);
```

```
**Modified Dictionary**Key is:  Name Value is:  Pierre
Key is:  City Value is:  Istanbul
Key is:  Height Value is:  179
Key is:  Weight Value is:  68
```

1.6 Ruby

In Ruby, associative arrays are referred as hash data structure.

This foo function is for printing a single pair of a hash. It will be used in later examples.

```
# foo function to print pairs
def foo(k, temp)
    print "Key is: ", k, " Value is: ", temp[k]
    puts
  end
```

We can initialize a hash as follows. Unlike the languages mentioned before to initialize elements we use “=>” notation rather than “:” to assign values.

```
#initialize hash
arr = { "Name" => "Jane", "City" => "Amsterdam", "Age" => 32, "Weight" => 58}
```

The initial hash can be printed by providing a hash name directly to puts or print function.

```
puts "**Initial Hash**", arr   #printing first version
```

```
**Initial Hash**
{"Name"=>"Jane", "City"=>"Amsterdam", "Age"=>32, "Weight"=>58}
```

We can get the value for a given key by using subscript operation

```
puts "Value for key Name is ", arr["Name"]   #get the value for Name key
puts "Value for key Salary is ", arr["Salary"]  #get the value for nonexistent key
```

```
Value for key Name is
Jane
Value for key Salary is
```

We can add a new value by the subscript operator.

```
arr["Height"] = 162   #add new element
```

To remove an element function delete(key) should be used.

```
arr.delete("City")   #remove an element
arr.delete("Salary")   #remove nonexistent element
```

We can modify values as follows by subscript operator.

```
arr["Age"] = 33   #modify Age
```

We can search for the existence of a key value by using the “has_key” function.

```
#search for existance of a key
puts "City key existance: ", arr.key?("City")
```

```
City key existance:
false
```

We can search for the existence of a key value by using the “has_value” function.

```
#search for existence of a value
puts "Value Amsterdam existence: ", arr.has_value?(162)
```

```
Value Amsterdam existence:
true
```

Finally, we can print the hash element pairs by calling the foo function in an iteration over the hash. The resultant output can be seen below.

```
#print modified version in loop by foo
puts "\n**Modified Hash**"
arr.each do |key, value|
  foo(key, arr)
end
```

```
**Modified Hash**
Key is: Name Value is: Jane
Key is: Age Value is: 33
Key is: Weight Value is: 58
Key is: Height Value is: 162
```

1.7 Rust

In Rust, associative array structure can be described by Hashmap structure.

To use the HashMap functionality we should first import the corresponding collection.

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

This foo function is for printing a single pair of a hash. It will be used in later examples.

```
fn foo(k: &str, v: &str) {
    println!("Key: {}, Value: {}", k, v);
}
```

Rust language differs from the other languages in terms of creation and initialization of the array. There are various ways to perform creation and initialization, below it is given the easiest way to do this task. First we should create a HashMap and then we should insert the elements one by one.

```
    //create
    let mut mymap = HashMap::new();
    //initialize
    mymap.insert("Name", "Kate");
    mymap.insert("City", "Berlin");
    mymap.insert("Age", "27");
    mymap.insert("Height", "159");
    mymap.insert("Weight", "46");
```

The initial map can be printed by iteration as follows. The output is also given below.

```
    println!("**Initial Map**");
    for (mapkey, &mapval) in mymap.iter() {
        println!("Key: {}, Value: {}", mapkey, mapval);
    }
```

```
**Initial Map**
Key: Name, Value: Kate
Key: Height, Value: 159
Key: Age, Value: 27
Key: City, Value: Berlin
Key: Weight, Value: 46
```

We can get the value for a given key by using a subscript operator. Since Name exists as a key, we can access it.

```
 println!("\nValue of key Name: {}\n", mymap["Name"]);   //get the value for given key
```

```
Value of key Name: Kate
```

When we want to access a nonexistent key's value we will get an error as given below. Thus, we should avoid accessing nonexistent elements.

```
 println!("\nValue of key Salary: {}\n", mymap["Salary"]);   //get the value for given
nonexistent key
```

```
thread 'main' panicked at 'no entry found for key',
src/main.rs:22:45
```

We can insert new elements by using the "insert" function. The first value is the ley and the second is the corresponding value.

```
    mymap.insert(&"Country", "Germany");  //adding new element
```

We can remove elements by using the “remove” function. Unlike accessing, we can try to remove a nonexistent element, it will not give an error. The operation simply will not perform anything.

```
    mymap.remove(&"City");   //removing element
    mymap.remove(&"Salary");   //removing nonexistent element
```

One way to modify an element is given as below.

```
    *mymap.get_mut("Name").unwrap() = "Alice";   //modify value of Key Name
```

Rust provides the “contains_key” function to check existence of a key.

```
    //checking key existance
    println!("Key City existance: {}", mymap.contains_key(&"City"));
    println!("Key Country existance: {}", mymap.contains_key(&"Country"));
```

```
Key City existance: false
Key Country existance: true
```

Unlike keys, Rust does not provide a function to check value existance. It can be done as follows.

```
    //cheking value existance
    let bool_exist = mymap.values()
        .any(|&val| *val == *"46");
    println!("Value 46 Existance: {}", bool_exist);
```

```
Value 46 Existance: true
```

Finally, we can print the pairs of the hashmap by calling the foo function.

```
    println!("\n**Modified Map**");
    //printing contents by foo
    for (keyval, &val) in mymap.iter() {
        foo(keyval, val);
    }
```

```
**Modified Map**
Key: Name, Value: Alice
Key: Country, Value: Germany
Key: Height, Value: 159
Key: Age, Value: 27
Key: Weight, Value: 46
```

2 Evaluation of Languages

Readability and writability levels differ from language to language for associative array structure. There are similar conventions among most of the languages such as usage of the subscript operator to access and add elements. Rust differs from the other languages in terms of its array of conventions which makes it harder to learn and write. For instance, performing operations mostly require passing the hashmap addresses or accessing the elements by indirect access. Creation and initialization of hashmap are harder than compared to others. Also, the functions provided may be hard to understand since the naming is a bit complicated. For instance, we need to use **mymap.get_mut("Name").unwrap()* to modify a key-value name.

Ruby, on the other hand, is also not as good in terms of readability and writability as Dart, Lua, PHP, Javascript, and Python. For instance, to initialize an array we should use “=>” rather than “:” such as *arr = { "Name" => "Jane", "City" => "Amsterdam", "Age" => 32, "Weight" => 58}.* Writing “:” is easier than writing “=>” and since “:” is used widely it makes it easier for someone who has previous programming experience. Also, to search for existence we should use a function that does not have the simplest naming form such as *arr.key?("City")* and *arr.has_value?(162).*

Considering Lua, it is easier to read and write than Rust and Ruby, but it has some disadvantages. To initialize an array, a table, we should use extra subscript operator for the key values such as *mytable = {["Name"] = "John", ["City"] = "Rome", ["Age"] = 28, ["Height"] = 182}.* This might be complicated since we access elements as usual with a single “[ ]”, but in creation, it is added for keys. One of the biggest disadvantages is that Lua does not perform printing in an ordered manner, which means that every time we start the program the order of the elements of the table changes. This might make testing, so writing, and reading harder. Also, in Lua, we cannot print the table elements directly in a print function since when we do that it will only give the type and unique id of the object, not the contents.

Considering PHP and Javascript even though associative arrays might be implemented easily they should not be used for only that purpose since the languages are specifically designed for website scripting, so the nature of the languages requires different syntax structures than usual.

Dart can be considered as good in terms of readability and writability, and since it is similar to widely used languages such as Java and C++ it is easy to learn.

Considering all seven languages, in my opinion, Python is the best language for associative array structure in terms of readability and writability. Reading a Python code is easier than the other languages since it is simple and the naming of the language is understandable. It is much easier to learn how to implement an associative array, a dictionary, since the syntax is minimal. It is also minimal in terms of functions, but at the same time powerful. It does not include unnecessary functions and naming but it provides a sufficient amount of them to perform programming. For instance, in other languages, there are mostly built-in functions to search for the existence of a key and a value. Although this may seem to make writing code easier it makes it harder to write code since the programmer should learn and check the method names constantly when the function names are complicated and when there are lots of functions to be used. For this case, Python simply requires using “in” which is widely used in Python programming all the time. Iteration over dictionary elements is simple and easy compared to other languages. Also, accessing keys and values is much easier because of functions as keys() and values(), and because of the function names, they are easy to learn. These functions make calculations and access much easier. Also, in Python printing array elements are easy, since we are only required to provide the dictionary name in the print function. This condition makes testing a lot easier. Considering testing, Python is the only language that gives an error for both when we try to access a nonexistent element and when we try to remove a nonexistent element. This case makes writability much easier since it is much easier to be aware of the errors when we perform programming.

3 Learning Strategy

In a short time, I had to learn how to use associative arrays and have to provide examples for seven different languages. For this purpose, I used internet sources as much as I could. I have read manuals from the official websites of the languages. Apart from official manuals, I have used blogs such as *GeeksforGeeks* for examples of code snippets to discover the syntax and working principles of the languages. Since my task is on associative arrays, I have focused specifically on learning associative array structures of these languages rather than the whole language implementation purposes. Then I tested the examples using online compilers. The examples generally were focused on one point, such as array creation and printing only. For other features, I had to do research on the internet separately. Finally, I started creating programs myself by combining the information I have gained. During this process, there were many problems I encountered and I was able to solve them by searching on the internet. *Stack Overflow*, for example, has been very useful for this purpose.

4 Compilers/Interpreters

The online interpreters and compilers used for testing are given below.

Dart:	https://dartpad.dev/
Javascript:	https://js.do/
Lua:	https://www.tutorialspoint.com/execute_lua_online.php
PHP:	https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler
Python:	https://www.programiz.com/python-programming/online-compiler/
Ruby:	https://www.tutorialspoint.com/execute_ruby_online.php
Rust:	https://play.rust-lang.org/