1) Code Explanations

Dart:

1)Initialize:

```
//1)Initialize
var assocArray = {
        'key1': 'val1',
        'key2': 'val2',
        'key3': 3
};
print("####################################");
print("1)Initialize:");
print(assocArray);
print("####################################\n");
```

In this part, I created an associative array with three pairs. Then, I print the whole associative array to see whether it is initialized with any fault. It prints the associative array as its pairs are separated by commas, in a linear fashion.

```
####################################
1)Initialize:
{key1: val1, key2: val2, key3: 3}
####################################
```

2)Get the value for a given key:

```
//2)Get the value for a given key
print("####################################");
print("2)Get the value for a given key");
print(assocArray['key1']);
print(assocArray['key2']);
print(assocArray['key3']);
print(assocArray['newKey']);
print("####################################\n");
```

In this part, I print the values using the key values I used in the initialization part. I also tried to print the value using a non-existing key to see how Dart would react. It prints values for valid keys and “null” for the invalid key.

```
####################################
2)Get the value for a given key
val1
val2
3
null
####################################
```

3)Add a new element:

```
//3)Add a new element
assocArray['newKey'] = 12;
print("####################################");
print("3)Array after adding a new element");
printAssocArray(assocArray);
print("####################################\n");
```

Here, I add a new pair, with the key “newKey” and value “12” in integer form. Then, I print the associative array to see what change has appeared. The output states that a new pair with the key “newKey” and the value “12” in integer form has been created.

```
####################################
3)Array after adding a new element
{key1: val1, key2: val2, key3: 3, newKey: 12}
####################################
```

4)Remove an element:

```
//4)Remove an element
assocArray.remove("newKey");
print("####################################");
print("4)Array after removing \"newKey\"");
print(assocArray);
print("####################################\n");
```

Here, I remove the pair I just created in part 3). I print the associative array to see the change. Indeed, the pair has been removed.

```
####################################
4)Array after removing "newKey"
{key1: val1, key2: val2, key3: 3}
####################################
```

5)Modify the value of an existing element:

```
//5)Modify the value of an existing element
assocArray['key1'] = 111;
print("####################################");
print("5)Array after modifying the value of key \"key1\"");
print(assocArray);
print("####################################\n");
```

Here, I modify the value of the pair with the key “key1” from “val1” to “111”. Then I print the associative array to see the change. Indeed, the value of that pair has changed.

```
####################################
5)Array after modifying the value of key "key1"
{key1: 111, key2: val2, key3: 3}
####################################
```

6)Search for the existence of a key:

```
//6)Search for the existence of a key
print("####################################");
print("6)Search for the existence of keys");
print("Looking for key \"key1\":" + assocArray.containsKey("key1").toString());
print("Looking for key \"key2\":" + assocArray.containsKey("key2").toString());
print("Looking for key \"key3\":" + assocArray.containsKey("key3").toString());
print("Looking for key \"newKey\":" + assocArray.containsKey("newKey").toString());
print("####################################\n");
```

Here, I look for the existence of existing and of a non-existing key. I used the built-in containsKey function that returns true if the parameter key exists and false if the parameter key does not exist in the associative array for my search. Since the return value of the function is bool(ean), and concatenating strings and bool(ean) values creates problems in Dart, I converted them to string using toString function. The output is as expected: prints true for existing keys and false for the non-existing key.

```
####################################
6)Search for the existence of keys
Looking for key "key1":true
Looking for key "key2":true
Looking for key "key3":true
Looking for key "newKey":false
####################################
```

7)Search for the existence of a value:

```
 //7)Search for the existence of a value
print("####################################");
 print("7)Search for the existence of values");
 print("Looking for value \"111\":" + assocArray.containsValue(111).toString());
 print("Looking for value \"val2\":" + assocArray.containsValue("val2").toString());
 print("Looking for value \"Val3\":" + assocArray.containsValue("Val3").toString());
 print("Looking for value \"aVal\":" + assocArray.containsValue("aVal").toString());
 print("####################################\n");
```

Here, I look for existing and non-existing values. I use containsValue function for this purpose. The way I use functions in this part is the same as the previous part. Output is again as expected.

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

```
//8)Loop through an associative array, apply a function, called foo, which prints the key-value pairs
print("####################################");
print("8)Loop through an associative array, apply a function, called foo, which prints the key-value pairs");
var keys = assocArray.keys;
for(var key in keys) {
        foo(assocArray, key);
}
```

```
void foo(var a, String key) {
        print("Key: " + key + ", Values: " + a[key].toString());
}
```

Here, I create a for loop to iterate over the associative array and pass both the associative array and its keys to a function “foo” as parameters, which prints keys and values in pairs. Output is as expected.

```
###################################
8)Loop through an associative array, apply a function, called foo, which prints the key-value pairs
Key: key1, Values: 111
Key: key2, Values: val2
Key: key3, Values: 3
```

Javascript:

In these parts, I print the values of the associative array after every significant change. Since outputs are printed in one-value-per-line, I didn’t include them in the report, except the part 8).

1)Initialize:

```
//1
//initializing
let associativeArray = { "Name1": "val1", "Name2": "val2", "Name3": "val3","Name4": "val4" };
```

Here, I initialized an associative array with 4 pairs.

2)Get the value for a given key:

```
//printing  each value one by one
let key1 = "Name1";
let key2 = "Name2";
let key3 = "Name3";
let key4 = "Name4";
console.log("Values for a given key:")
console.log(associativeArray[key1]);
console.log(associativeArray[key2]);
console.log(associativeArray[key3]);
console.log(associativeArray[key4] + "\n");
```

Here, I print all the values of the associative array.

3)Add a new element:

```
//2
//adding a new element
associativeArray["Name5"] = "val5";
```

Here, I add a new element to the associative array.

4)Remove an element:

```
//3
//deleting
delete associativeArray["Name5"];
```

Here, I deleted the element I created in the previous part.

5)Modify the value of an existing element:

```
//5
//modifying values of Name2 and Name3
associativeArray["Name2"] = "VAL2";
associativeArray["Name3"] = "VAL3";
```

I modify the values corresponding to keys “Name2” and “Name3”.

6)Search for the existence of a key:

```
//looking for existence of existing keys
console.log("Looking for existence of keys:");
console.log("Looking for \"Name1\":" + (key1 in associativeArray));
console.log("Looking for \"Name2\":" + (key2 in associativeArray));
console.log("Looking for \"Name3\":" + (key3 in associativeArray));
console.log("Looking for \"Name4\":" + (key4 in associativeArray));

//looking for existence of non-existing keys
console.log("Looking for \"NAME1\":" + ("NAME1" in associativeArray));
console.log("Looking for \"name2\":" + ("name2" in associativeArray));
console.log("Looking for \"NamE3\":" + ("NamE3" in associativeArray));
console.log("Looking for \" \":" + ( " " in associativeArray) + "\n");
```

I look for the existence of existing and non-existing keys. The output is as expected.

7)Search for the existence of a value:

```
let values = Object.values(associativeArray);
console.log("Searching values:");
console.log("Looking for \"val1\": " + (values.indexOf("val1") > -1));
console.log("Looking for \"val2\": " + (values.indexOf("val2") > -1));
console.log("Looking for \"VAL2\": " + (values.indexOf("VAL2") > -1));
console.log("Looking for \"val3\": " + (values.indexOf("val3") > -1));
console.log("Looking for \"VAL3\": " + (values.indexOf("VAL3") > -1));
console.log("Looking for \"val4\": " + (values.indexOf("val4") > -1));
console.log("Looking for \"val5\": " + (values.indexOf("val5") > -1) + "\n");
```

In this part, I get the values in the associative array and store them in the “values” variable using the values() function of the Object class. indexOf function returns a value >= 0, index of the value, if the value exists. I check it and print the result.

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

```
//8
for(key in associativeArray) {
        foo(key, associativeArray[key]);
}
function foo(key, val) {
        console.log(key + ", " + val);
}
```

Here, I print the pairs in the associative array one by one in a loop by calling the function foo. The output is as expected.

```
Name1, val1
Name2, VAL2
Name3, VAL3
Name4, val4
```

Lua:

In this part, I used the documentation of Lua on Tables. You can access the documentation here: https://www.lua.org/pil/2.5.html

1)Initialize:

```
--[[ 1)Initialize  --]]
print("1)Initialize")
assocArray = { key1 = 1, key2 = 2, key3 = 3}
```

Here, I initialize an associative array with three key-value pairs.

2)Get the value for a given key:

```
--[[ 2)Get the value for a given key  --]]
print("################################")
print("2)Get the value for a given key")
print(assocArray.key1)
print(assocArray["key1"])
print(assocArray.key2)
print(assocArray["key2"])
print(assocArray.key3)
print(assocArray["key3"])
```

Here, I experimented with getting the values for a given key. In Lua, you can access the values using keys either by using the keys between closed parentheses or directly like accessing a property of an object. It should be noted that the latter method wouldn't work if I had variables named by any of the keys(key1, key2, or key3) with unrelated values. Documentation can be read to understand further.

The output is as expected.

```
2)Get the value for a given key
1
1
2
2
3
3
################################
```

3)Add a new element:

```
--[[ 3)Add a new element  --]]
print("################################")
print("3)Add a new elemet")
assocArray["key4"] = "val4"
print(assocArray["key4"])
```

Here, I add a new pair with key “key4” and value “val4”. Then, I print the value of the pair to see whether it is added or not.

```
3)Add a new elemet
val4
################################
```

4)Remove an element:

```
--[[ 4)Remove an element, let's say, key2  --]]
print("################################")
print("4)removing\"key2\":")
assocArray["key2"] = nil
print(assocArray["key2"])
```

Here, I remove the pair with hey “key2” and then try to print the value to see whether it is removed. Indeed, it is removed.

```
4)removing"key2":
nil
################################
```

5)Modify the value of an existing element:

```
--[[ 5)Modify the value of an existing element  --]]
print("################################")
print("5) Modifying the value of \"key1\" to \"val1\":")
assocArray.key1 = "val1"
print(assocArray.key1)
```

Here, I modify the value of the pair with key “key1”. Again, what I did here wouldn’t work due to the reason stated in part 2) if a variable key1 existed. Then I print the value of that pair.

```
5) Modifying the value of "key1" to "val1":
val1
################################
```

6)Search for the existence of a key:

```
--[[ 6)Search for the existence of a key  --]]
print("################################")
print("6)Seaching for the existence of \"key1\" and \"key2\":")
print(assocArray.key1 ~= nil)
print(assocArray.key2 ~= nil)
```

“key1” is still in the associative array, but “key2” was removed. I look whether they exist or not. If “keyx” doesn’t exist, assocArray.keyx would return nil, hence if it doesn’t return nil, the key “keyx” exists. The output is as expected.

```
6)Seaching for the existence of "key1" and "key2":
true
false
################################
```

7)Search for the existence of a value:

```
--[[ 7)Search for the existence of a value  --]]
print("################################")
print("7)Search for the existence of \"val1\" and \"val2\"")
doesVal1Exist = false;
doesVal2Exist = false;
for key,value in pairs(assocArray) do
        if(value == "val1") then
                doesVal1Exist = true;
        end
        if(value == "val2") then
                doesVal2Exist = true;
        end
end
print(doesVal1Exist)
print(doesVal2Exist)
```

“val1” is still in the associative array, but “val2” was removed. Here, traverse the associative array and look for values “val1” and “val2” for every pair. If they exist, their corresponding doesExist boolean will be true at the end of the loop. I print the boolean values, and they are as expected.

```
7)Search for the existence of "val1" and "val2"
true
false
################################
```

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

```
--[[ 8)Loop through an associative array, apply a function, called foo, which simply prints the key-value pair  --]]
print("################################")
print("8)Loop through an associative array, apply a function, called foo, which simply prints the key-value pair")
function foo(key, value)
        print("Key: " .. key .. ", Value: " .. value)
end

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

Here, I traverse the array in a for loop and print pairs calling a function foo. The output is as expected.

PHP:

1)Initialize:

```php
$assoc_array = array("key1" => "val1", "key2" => "val2", "key3" => 3);
echo  "1)Initialize\n";
printAssocArray($assoc_array);
echo "#####################################\n\n";
```

Here, an associative array with three pairs is created. Then I print the associative array using my function in part 8). The output is as expected.

```
1)Initialize
Key: key1, Val: val1
Key: key2, Val: val2
Key: key3, Val: 3
#####################################
```

2)Get the value for a given key:

```php
echo "2)Get the value for a given key\n";
echo $assoc_array["key1"] . "\n";
echo $assoc_array["key2"] . "\n";
echo $assoc_array["key3"] . "\n";
```

Here, I get the values for initially created keys and print them. The output is as expected.

```
2)Get the value for a given key
val1
val2
3
```

3)Add a new element:

```php
echo "#####################################\n\n";
echo "3)Add a new element\n";
$assoc_array["key4"] = "val4";
printAssocArray($assoc_array);
echo "\n";
```

I add a new pair here with key “key4” and value “val4”. Then I print the array using the function in part 8). The output is as expected

```
#####################################

3)Add a new element
Key: key1, Val: val1
Key: key2, Val: val2
Key: key3, Val: 3
Key: key4, Val: val4
```

4)Remove an element:

```
echo "#########################################\n\n";
echo "4)Remove an element\n";
unset($assoc_array['key1']);
printAssocArray($assoc_array);
echo "\n";
```

I remove the pair with key “key1“ and then print the associative array. The output is as expected.

```
#########################################

4)Remove an element
Key: key2, Val: val2
Key: key3, Val: 3
Key: key4, Val: val4
```

5)Modify the value of an existing element:

```
echo "#########################################\n\n";
echo "5)Modify the value of an existing element\n";
$assoc_array['key2'] = "VAL2";
printAssocArray($assoc_array);
echo "\n";
```

I modify the value of the pair with key “key2” from “val2” to “VAL2”. Then I print the associative array. The output is as expected.

```
#########################################

5)Modify the value of an existing element
Key: key2, Val: VAL2
Key: key3, Val: 3
Key: key4, Val: val4
```

6)Search for the existence of a key:

```
echo "#########################################\n\n";
echo "6)Search for the existence of a key\n";
echo "Looking for \"key1\": " . array_key_exists("key1", $assoc_array) . "\n";
echo "Looking for \"key2\": " . array_key_exists("key2", $assoc_array) . "\n";
echo "Looking for \"key3\": " . array_key_exists("key3", $assoc_array) . "\n";
echo "Looking for \"key4\": " . array_key_exists("key4", $assoc_array) . "\n";
echo "Looking for \"key5\": " . array_key_exists("key5", $assoc_array) . "\n";
```

At the moment, pairs with keys “key2”, “key3” and “key4” exist in the associative array. I used the built-in function of PHP assoc_key_exists that returns 1 if the parameter key exists in the parameter associative array. Indeed, it returns 1 for existing keys.

```
6)Search for the existence of a key
Looking for "key1":
Looking for "key2": 1
Looking for "key3": 1
Looking for "key4": 1
Looking for "key5":
```

7)Search for the existence of a value:

```php
echo "#####################################\n\n";
echo "7)Search for the existence of a value\n";
$found;

$val = "val1";
$found = in_array($val, $assoc_array);
echo "Looking for \"val1\":" . $found . "\n";


$val = "val2";
$found = in_array($val, $assoc_array);
echo "Looking for \"val2\":" . $found . "\n";

$val = "VAL2";
$found = in_array($val, $assoc_array);
echo "Looking for \"VAL2\":" . $found . "\n";

$val = 3;
$found = in_array($val, $assoc_array);
echo "Looking for \"3\":" . $found . "\n";
```

Currently, “VAL2” and “3” values exist in the associative array. I use the built-in function in_array for existing and non-existing values. The output is as expected.

```
7)Search for the existence of a value
Looking for "val1":
Looking for "val2":
Looking for "VAL2":1
Looking for "3":1
```

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

```php
echo "8)Loop\n";

printAssocArray($assoc_array);

function printAssocArray($assoc_array) {
        foreach($assoc_array as $key => $val) {
                foo($key, $val);
        }
}
function foo($key, $val){
        echo "Key: $key, Val: $val \n";
}
```

Here, I turned the loop asked in the assignment into a function I named printAssocArray to use in previous parts. the associative array is traversed and a foo function is called to print key and value pairs. The output is as expected.

```
8)Loop
Key: key2, Val: VAL2
Key: key3, Val: 3
Key: key4, Val: val4
```

Python:

1)Initialize:

```
print("#########################################################")
print("1)Initialize:")
assoc_array = {"key1": "val1", "key2": "val2", "key3": "val3"}
print(assoc_array)
```

I initialized an associative array with three pairs. Then, I print the array. The output is as expected.

```
#########################################################
1)Initialize:
{'key1': 'val1', 'key2': 'val2', 'key3': 'val3'}
```

2)Get the value for a given key:

```
print("#########################################################")
print("2)Get the value for a given key:")
print(assoc_array["key1"])
print(assoc_array["key2"])
print(assoc_array["key3"])
```

I get the values for each key existing in the associative array and print them. The output is as expected.

```
#########################################################
2)Get the value for a given key:
val1
val2
val3
```

3)Add a new element:

```
print("#########################################################")
print("3)Add a new element:")
assoc_array["key4"] = "val4"
print(assoc_array)
```

I add a new pair in the same syntax as modifying a value. Then I print the array. The output is as expected.

```
#########################################################
3)Add a new element:
{'key1': 'val1', 'key2': 'val2', 'key3': 'val3', 'key4': 'val4'}
```

4)Remove an element:

```
print("4)Remove an element:")
del assoc_array["key1"]
#pop method can also be used. popItem can be used for the last item
print(assoc_array)
```

Here I used the built-in del method to delete a pair. For the last pair, the pop function could've been used. It seems like Python provides stack functions to associative arrays. After the removal of the pair, I print the array. The output is as expected.

```
############################################################
4)Remove an element:
{'key2': 'val2', 'key3': 'val3', 'key4': 'val4'}
```

5)Modify the value of an existing element:

```
print("############################################################")
print("5)Modify the value of an existing element")
assoc_array["key2"] = "VAL2"
print(assoc_array)
```

I modify the pair's value with key "key2" from "val2" to "VAL2" and then print the array. The output is as expected.

```
############################################################
5)Modify the value of an existing element
{'key2': 'VAL2', 'key3': 'val3', 'key4': 'val4'}
```

6)Search for the existence of a key:

```
print("#########################################################")
print("6)Search for the existence of the keys \"key2\" and \"key1\"")
#has_key() method could've been used for python2. It is removed in python3.
#get() can also be used. I used if conditions, as it seems to work for every version of python
key = "key2"
if key in assoc_array:
    print("\"key2\" Found")
else:
    print("\"key2\" Not Found")
key = "key1"
if key in assoc_array:
    print("\"key1\" Found")
else:
    print("\"key1\" Not Found")
```

Here, I look for the existence of keys "key1" and "key2" using if conditions and print the results. Since I was using python3 in the assignment, I couldn't use the built-in has_ket function of python2 as it is removed in python3. I didn't look in detail at the built-in get function, but it seems it could've been used too. The output is as expected.

```
############################################################
7)Search vor the existence of values "val2" and "VAL2"
Does "val2" exists: False
Does "VAL2" exists: True
```

7)Search for the existence of a value:

```
print("#########################################################")
print("7)Search vor the existence of values \"val2\" and \"VAL2\"")
val = "val2"
print("Does \"val2\" exists: " + str(val in assoc_array.values()))
val = "VAL2"
print("Does \"VAL2\" exists: " + str(val in assoc_array.values()))
```

I used the built-in values function to get the values in the associative array and then print whether “val2” and “VAL2” exist in the array or not. The output is as expected.

```
#########################################################
7)Search vor the existence of values "val2" and "VAL2"
Does "val2" exists: False
Does "VAL2" exists: True
```

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

```
print("#####################################################")
print("8)Loop through an associative array, apply a function, called foo, which simply print
s the key-value pair")

def foo(key, val):
    print("Key: " + key + ", Value: " + val)

for key in assoc_array:
    foo(key, assoc_array[key])
```

Here, I traverse the associative array with a for loop and call function foo to print key and values as pairs. The output is as expected.

```
#####################################################
8)Loop through an associative array, apply a function, called foo, which simply prints the key-value pair
Key: key2, Value: VAL2
Key: key3, Value: val3
Key: key4, Value: val4
```

Ruby:

1)Initialize:

```
puts "1)Initialize"
assoc_array = {key1: "val1", key2: "val2", key3: "val3"}
puts assoc_array
puts "#############################################\n"
```

I initialize an associative array and the print it. The output is as expected.

```
1)Initialize
{:key1=>"val1", :key2=>"val2", :key3=>"val3"}
#############################################
```

2)Get the value for a given key:

```
puts "2)Get the value for a given key"
puts assoc_array[:key1]
puts assoc_array[:key2]
puts assoc_array[:key3]
```

Here I get the value for evert key existing and then print. The output is as expected.

```
2)Get the value for a given key
val1
val2
val3
##############################################
```

3)Add a new element:

```
puts "3)Add a new element "
assoc_array[:key4] = "val4"
puts assoc_array
puts "##############################################\n"
```

Here I add a new element with key “key4” and value “val4” and then print the array. the output is as expected.

```
3)Add a new element
{:key1=>"val1", :key2=>"val2", :key3=>"val3", :key4=>"val4"}
##############################################
```

4)Remove an element:

```
puts "4)Remove an element "
assoc_array.delete(:key1)
puts assoc_array
puts "###############################################\n"
```

I remove the element with key “key1” and then print the array. The output is as expected.

```
4)Remove an element
{:key2=>"val2", :key3=>"val3", :key4=>"val4"}
###############################################
```

5)Modify the value of an existing element:

```
puts "5)Modify the value of an existing element "
assoc_array[:key2] = "VAL2"
puts assoc_array
puts "###############################################\n"
```

I modify the element's value with key “key2” and then print the array. The output is as expected.

```
5)Modify the value of an existing element
{:key2=>"VAL2", :key3=>"val3", :key4=>"val4"}
###############################################
```

6)Search for the existence of a key:

```
puts "6)Search for the existence of keys \"key1\" and \"key2\" "
puts assoc_array.has_key?(:key1)
puts assoc_array.has_key?(:key2)
puts "##############################################\n"
```

I used the built-in has_key? function for an existing key “key2” and for a non-existing one “key1”. The output is as expected.

```
6)Search for the existence of keys "key1" and "key2"
false
true
##########################################
```

7)Search for the existence of a value:

```
puts "7)Search for the existence of a value "
puts assoc_array.has_value?("val1")
puts assoc_array.has_value?("val2")
puts assoc_array.has_value?("VAL2")
puts "##########################################\n"
```

Here, I used the built-in has_value? function on existing and non-existing values. The output is as expected.

```
7)Search for the existence of a value
false
false
true
##########################################
```

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

```
puts "8)Loop through an associative array, apply a function, called foo, which simply prints the key-value pair"
def foo(pair)
  puts "#{pair}"
end
for pair in assoc_array do
  foo(pair)
end
puts "########################################\n"
```

I use a for loop to iterate over the associative array and call a function foo to print the pairs. The output is as expected.

```
8)Loop through an associative array, apply a function, called foo, which simply prints the key-value pair
[:key2, "VAL2"]
[:key3, "val3"]
[:key4, "val4"]
########################################
```

Rust

1)Initialize:

```
println!("##################################################");
println!("1)Initialize:");
let mut assoc_array = HashMap::new();
assoc_array.insert("key1", "val1");
assoc_array.insert("key2", "val2");
assoc_array.insert("key3", "val3");
println!("{:?}", assoc_array);
println!("##################################\n");
```

Here I initialized an empty associative array. I couldn't find a way to create an associative array with initial values in Rust, so I inserted values after creating it. Then, I printed the associative array. The output is as expected.

```
##################################
1)Initialize:
{"key3": "val3", "key2": "val2", "key1": "val1"}
##################################
```

2)Get the value for a given key:

```
println!("#####################################");
println!("2)Get the value for a given key:");
println!("{}", assoc_array["key1"]);
println!("{}", assoc_array["key2"]);
println!("{}", assoc_array["key3"]);
println!("#####################################\n");
```

I get the values for existing keys and print them. The output is as expected.

```
#####################################
2)Get the value for a given key:
val1
val2
val3
#####################################
```

3)Add a new element:

```
println!("#####################################");
println!("3)Add a new element:");
assoc_array.insert("key4", "val4");
println!("{:?}", assoc_array);
println!("#####################################\n");
```

I add a new element using the built-in insert function that I used in part 1) and print the array. The output is as expected.

```
##################################
3)Add a new element:
{"key3": "val3", "key4": "val4", "key2": "val2", "key1": "val1"}
##################################
```

4)Remove an element:

```
println!("#####################################");
println!("4)Remove an element:");
assoc_array.remove("key1");
println!("{:?}", assoc_array);
println!("#####################################\n");
```

I removed an element using the built-in remove function and then I print the array. The output is as expected.

```
###################################
4)Remove an element:
{"key3": "val3", "key4": "val4", "key2": "val2"}
###################################
```

5)Modify the value of an existing element:

```
println!("####################################");
println!("5)Modify the value of an existing element:");
assoc_array.insert("key2","VAL2");
println!("{:?}", assoc_array);
println!("####################################\n");
```

The way I modified an element is the same as I added a new element. There can't exist two elements with the same key, so the previous element gets deleted, and a new one is created. Then I print the associative array. The output is as expected.

```
###################################
5)Modify the value of an existing element:
{"key3": "val3", "key4": "val4", "key2": "VAL2"}
###################################
```

6)Search for the existence of a key:

```
println!("####################################");
println!("6)Search for the existence of a key:");
println!("{}", assoc_array.contains_key("key1"));
println!("{}", assoc_array.contains_key("key2"));
println!("####################################\n");
```

I used the built-in contains_key function to check the existence of keys and then print the results. The output is as expected.

```
###################################
6)Search for the existence of a key:
false
true
###################################
```

7)Search for the existence of a value:

```
println!("####################################");
print!("7)Search for the existence of the value \"VAL2\": ");
let values = assoc_array.values();
let value_to_search_1 = "VAL2";
let value_to_search_2 = "val2";
let mut does_exist_1 = false;
let mut does_exist_2 = false;
for value in values {
    if value == &value_to_search_1 {
        does_exist_1 = true;
    }
    if value == &value_to_search_2 {
        does_exist_2 = true;
    }
}
println!("{}", does_exist_1);
println!("Looking for the value \"val2\": {}", does_exist_2);
println!("####################################\n");
```

First, I copied the values in the associative array using the built-in values function to a variable. then I created variables for values I wanted to search and mutable boolean values for search purposes. Then I print the boolean values. The output is as expected.

```
####################################
7)Search for the existence of the value "VAL2": true
Looking for the value "val2": false
####################################
```

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

```
println!("#################################");
println!("8)Loop through an associative array, apply a function, called foo, which simply prints the key-value pair ");
for pair in assoc_array {
    foo(pair);
}

fn foo(pair: (&str, &str)) {
    println!("{:?}", pair);
}
println!("#################################\n");
```

I traversed the associative array in pairs and called a foo function to print them. The output is as expected.

```
#################################
8)Loop through an associative array, apply a function, called foo, which simply prints the key-value pair
("key3", "val3")
("key4", "val4")
("key2", "VAL2")
#################################
```

2) Writability&Readability of Languages:

Dart:

I think Dart is good both in terms of readability and writability for associative arrays. The syntax for initializing associative arrays is not absurd, which is helpful both for readability and writability. When writing in Dart, I tried to do whatever came to my mind without looking at the documentation, and it mostly worked. This indicated good writability. There are built-in functions with sensible for searching keys and values, which is helpful both for writability and readability.

Javascript:

I found Javascript good both in terms of writability and readability. It is better than Dart when it comes to associative arrays, but that might be because I am familiar with Javascript more than Dart. It is readable, as the syntax is quite simple and you don't gel lost in it. It is also possible to write various codes to success a purpose. For example, Javascript has a built-in function called hasOwnProperty which checks whether a key exists in the associative array or not. I didn't need to use it as trying the first thing that came to my mind (keyx in associativeArray) was already working. I think this is a sign of good writability.

Lua:

I think the designers of Lua sacrificed readability for writability, which doesn't bother me, but still a questionable decision. For example, there are multiple ways to get a value for a given key, you can either use the array notation or the property notation if no such variable has been created with the property name. This is good for writability but makes it hard to read the code. When one gets used to the syntax of it, one wouldn't have much problem using associative arrays in Lua, but it is not that readable at first for beginners.

PHP:

I think PHP is good in terms of writability but not so much in readability for associative arrays. There are built-in functions, such as array_key_exists and in_array, that helps to search for keys and values. Initialization syntax is a bit strange with equal signs and arrows, but other than that, writability is no problem. Readability is an issue as it is hard to distinguish variables from each other.

Python:

Python is my favourite in terms of readability and writability. There are multiple ways to search keys and values, and one way doesn't axe the others. The only problem I had was the differences between Python versions, but still, there are ways to handle things that work in every version of Python.

Ruby:
To be fair, Ruby is quite like Python. The only difference between them, which makes me put Ruby below Python, is the length of built-in function names. They are too long for my taste, and there are also some syntax differences which I dislike, such as the usage of “:”.

Rust:
Rust has strange ways to do things. I think it doesn’t have much problems when it comes to writability, and it can be the best when it comes to it, but readability is a real issue, particularly with print statements, in my opinion. This indirectly axes the readability of associative arrays.. It is not simple to understand what is going on with one look. Besides that, Rust is well designed both for readability and writability with its built-in functions, such as contains_key, insert, and remove.

My Favourite, Python:
I think Python is the best programming language for associative arrays due to its built-in functions for getting both values and keys. What I wrote simply worked. There were no complications in syntax, which is good for writability. Reading the code with associative arrays in Python is quite easy. Additionally, Python’s popularity also makes it a good choice for implementing associative arrays.

3) Learning Strategy
The main learning strategy I used was reading the documentation. For example, Lua and Rust’s documentation are top-notch, and they provide everything you would need. I also read a lot of error messages when I was dealing with the syntax of some programming languages, such as Rust.

Even though Rust’s documentation is top-notch, it was problematic to me. The problem I had with Rust’s documentation was that it requires the reader to be familiar with its syntax, which is natural. Since the syntax of Rust was unusual to me, I had quite a lot of trouble reading it. Thankfully, Rust’s compile-time errors and suggestions led me in the right direction.

Watching YouTube tutorials on associative arrays and programming languages in assignments was also helpful, as it led me to see how associative arrays were used in practice by other people.

It is necessary to note that I did **not** use any online compilers. I am using an up-to-date Ubuntu-based Linux distribution(Pop_OS 22.4 LTS, to be specific) at the moment, and I set up all the programming languages on my machine. Therefore, I didn’t need to use online compilers.