CS315

Homework Assignment 1

Fall 2022

Zeynep Doğa Dellal 22002572 Section 3

Instructor: Karani Kardaş

Teaching Assistants:

Dilruba Sultan Haliloğlu, Hamza İslam, Onur Yıldırım

1. **Dart**

Associative arrays are called maps in Dart programming language. There is no need to specify the associative array as Map, Dart understands the key-value match by syntax.

   a. **Initialize**

Code:

```
var assocArr = {
    'color': 'purple',
    'number': 2,
    'city': 'ankara',
    'class': 'CS-315'
  };
print(assocArr);
```

Output:

```
{color: purple, number: 2, city: ankara, class: CS-315}
```

   b. **Get the value for a given key**

I will use the Map above. Dart language uses the “$” symbol to refer to variables. Value of a key can be reached using name_of_the_map[key] syntax.

Code:

```
var colorInAssocArr = assocArr['color'];
print("\nGetting the value of color: $colorInAssocArr");
```

Output:

```
Getting the value of color: purple
```

   c. **Add a new element**

When adding a new element, Dart uses name_of_the_map[new_key] = new_value syntax. It adds to the end of the Map.

Code:

```
 assocArr['Language'] = 'dart';
```

Output:

{color: purple, number: 2, city: ankara, class: CS-315, Language: dart}

d. Remove an existing element

Dart has a built in function that removes a given key-value from Map.

Code:

assocArr.remove('class');

Output:

{color: purple, number: 2, city: ankara, Language: dart}

e. Modify the value of an existing element

By using name_of_the_map[existing_key] = new_value the value of a given key can be changed in Dart.

Code:

assocArr['number'] = '13';

Output:

{color: purple, number: 13, city: ankara, Language: dart}

f. Search for the existence of a key

Dart has a built-in function called containsKey(key) that returns boolean to search for the existence of a key.

Code:

bool containsName = assocArr.containsKey('name');

bool containsColor = assocArr.containsKey('color');

print("\nSearching if color key exists: $containsColor");

print("Searching if name key exists: $containsName");

Output:

Searching if color key exists: true

Searching if name key exists: false

g. Search for the existence of a value

Dart has a built-in function called containsValue(value) that returns boolean to search for the existence of a value.

Code:

```
bool containsAnkara = assocArr.containsValue('ankara');

bool containsIstanbul = assocArr.containsValue('istanbul');

print("\nSearching if ankara value exists: $containsAnkara");

print("Searching if istanbul value exists: $containsIstanbul");
```

Output:

```
Searching if ankara value exists: true

Searching if istanbul value exists: false
```

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

By using a simple for loop i called the foo function while iterating through the Map.

Code:

```
void foo(key, value) {

  print("Key : $key, value : $value");

}

for (var k in assocArr.keys) {

    foo(k, assocArr[k]);

  }
```

Output:

```
Key : color, value : purple

Key : number, value : 13

Key : city, value : ankara

Key : Language, value : dart
```

i. Evaluation of Dart language in terms of readability and writability of associative array data structure.

1. **Readability**

   Dart provides easy readability with easily understandable English function names like containsValue(), containsKey() and remove(). It requires characters such as “;”, “{}”

and “[]” which make the language more readable. Therefore Dart is easily can be read in terms of associative array data structure.

2. **Writability**

   Dart can be easily written as well because it separates keys and values using “{}” and “:”. One thing i found hard when using Dart was writing the character “$” while referring to variables. In terms of associative arrays Dart is a easily writable language.

1. **JavaScript**

Associative arrays are called Objects in Javascript programming language. There is no need to specify the associative array as Objects, Javascript understands the key-value match by syntax.

a. **Initialize**

Code:

```
var assocArr = {"color": "purple",
                "number": 2,
                "city": "ankara",
                "class":"CS-315" };
  console.log("Associative Array", assocArr);
```

Output:

Associative Array { color: 'purple', number: 2, city: 'ankara', class: 'CS-315' }

b. **Get the value for a given key**

I will use the the Object above. Value of a key can be reached using name_of_the_object[key] syntax.

Code:

```
 var colorInAssocArr = assocArr["color"];
console.log("\nGetting the value of color: "+ colorInAssocArr);
```

Output:

Getting the value of color: purple

c. **Add a new element**

When adding a new element, JavaScript uses name_of_the_map[new_key] = new_value syntax. It adds to the end of the Object.

Code:

```
assocArr['Language'] = 'dart';
```

Output:

Associative Array { color: 'purple', number: 2, city: 'ankara', class: 'CS-315', Language: 'dart'}

d. Remove an existing element

Object has a built in function that removes a given key-value from Object.

Code:

```
delete assocArr["class"];
```

Output:

Associative Array { color: 'purple', number: 2, city: 'ankara', Language: 'dart' }

e. Modify the value of an existing element

By using name_of_the_object[existing_key] = new_value the value of a given key can be changed in Object.

Code:

```
assocArr['number'] = '13';
```

Output:

Associative Array { color: 'purple', number: '13', city: 'ankara', Language: 'dart' }

f. Search for the existence of a key

Object doesn’t have a built in function but it has a in component that returns boolean to search for the existence of a key.

Code:

```
let containsColor= "color" in assocArr;

let containsName =  "name" in assocArr;

console.log("\nSearching if color key exists: "+containsColor);

console.log("Searching if name key exists: "+containsName);
```

Output:

Searching if color key exists: true

Searching if name key exists: false

g. Search for the existence of a value

JavaScript has a built-in function that is values(Object).includes(value) that returns if the associative array has the given value or not.

Code:

```
let containsAnkara = Object.values(assocArr).includes("ankara");

let containsIstanbul = Object.values(assocArr).includes('istanbul');

console.log("\nSearching if ankara value exists: "+containsAnkara);

console.log("Searching if istanbul value exists: "+containsIstanbul);
```

Output:

Searching if ankara value exists: true

Searching if istanbul value exists: false

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

By using a simple for loop i called the foo function while iterating through the Map.

Code:

```
function foo(key,value) {

    console.log("Key :"+ k + ", value : " + value);

 }

for (var k in assocArr) {

      foo(k, assocArr[k]);

  }
```

Output:

Key :color, value : purple

Key :number, value : 13

Key :city, value : ankara

Key :Language, value : dart

i. **Evaluation of JavaScript language in terms of readability and writability of associative array data structure.**

1. **Readability**

   Object provides easy readability with easily understandable English function names like delete. It requires characters such as ";", "{}" and "[]" which make the language more readable. Therefore JavaScript is easily can be read in terms of associative array data structure.

2. **Writability**

   Javascript can be easily written as well because it separates keys and values using "{}" and ":". In terms of associative arrays JavaScript is a easily writable language.

2. Lua

Associative arrays are called Tables in Lua programming language. There is no need to specify the associative array as Tables, Lua understands the key-value match by syntax.It requires "{}", "=" and "[]"

a. **Initialize**

Code:

```
local assocArr = {["color"] = "purple",
                  ["number"] = 2,
                  ["city"] = "ankara",
                  ["class"] = "CS-315"}
```

Output:

Key : class value :CS-315

Key : color value :purple

Key : city value :ankara

Key : number value :2

b. **Get the value for a given key**

I will use the the Table above. Value of a key can be reached using name_of_the_table[key] syntax.

Code:

```
local colorInAssocArr = assocArr['color'];

print("\nGetting the value of color:" , colorInAssocArr);
```

Output:

```
Getting the value of color: purple
```

c. Add a new element

When adding a new element, Lua uses name_of_the_table[new_key] = new_value syntax. It adds to the end of the Table.

Code:

```
assocArr['Language'] = 'lua';
```

Output:

```
Key :  number          value :2

Key :  color    value :purple

Key :  Language        value :lua

Key :  city       value :ankara

Key :  class    value :CS-315
```

d. Remove an existing element

When removing an element we equate the key with null and the key and value gets removed from the associative array.

Code:

```
assocArr["class"] = nil;
```

Output:

```
Key :  color    value :purple

Key :  number          value :2

Key :  Language        value :lua

Key :  city       value :ankara
```

e. **Modify the value of an existing element**

By using name_of_the_table[existing_key] = new_value the value of a given key can be changed in Table.

Code:

```
assocArr['number'] = '13';
```

Output:

```
Key :   Language        value :lua

Key :   city        value :ankara

Key :   color     value :purple

Key :   number          value :13
```

f. **Search for the existence of a key**

Table doesn't have a built in function but it has a in component that returns boolean to search for the existence of a key. If key value in the associative array is null, it returns false; otherwise it returns true.

Code:

```
local containsName = assocArr['name'] ~= nil;

local containsColor = assocArr['color'] ~= nil;

print("\nSearching if color key exists: ", containsColor);

print("Searching if name key exists: ", containsName);
```

Output:

```
Searching if color key exists:  true

Searching if name key exists: false
```

g. **Search for the existence of a value**

Lua requires to iterate through the array to find if a value exists.

Code:

```
 local containsAnkara = false;

local containsIstanbul = false;

for k, v in pairs(assocArr) do
```

```
    if v == "ankara" then
        containsAnkara = true;
    end
    if v == "istanbul" then
        containsIstanbul = true;
    end
end
print("\nSearching if ankara value exists: ", containsAnkara);
print("Searching if istanbul value exists: ", containsIstanbul);
```

Output:

```
Searching if ankara value exists: true
Searching if istanbul value exists: false
```

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

By using a simple for loop i called the foo function while iterating through the Table.

Code:

```
function foo(key, value)
    print("Key :", key, " value :" , value);
end
for k, v in pairs(assocArr) do
    foo(k,v);
end
```

Output:

```
Key :color, value : purple
Key :number, value : 13
Key :city, value : ankara
Key :Language, value : dart
```

i. **Evaluation of Lua in terms of readability and writability of associative array data structure.**

1. **Readability**

   Lua provides easy readability with easily understandable English function names like delete. It requires characters such as “=”, “{}” and “[]” which make the language more readable.

2. **Writability**

   Lua can be easily written as well because it separates keys and values using “{}”, “=” and “[]”. It does not have built-in functions to search the existing key or value though. In terms of associative arrays Lua is a easily writable language but it needs some improvement.

3. **PHP**

   a. **Initialize**

Initializing can be done with “=>” and “()” using array keyword in PHP.

Code:

```
$assocArr = array("color" => "purple", "number" => 2, "city" => "ankara", "class" => "CS-315");
```

Output:

```
Key :color, value : purple

Key :number, value : 2

Key :city, value : ankara

Key :class, value : CS-315
```

   b. **Get the value for a given key**

I will use the the array above. Value of a key can be reached using $name_of_the_array[key] syntax.

Code:

```
$colorInAssocArr = $assocArr["color"];

echo "\nGetting the value of color: " . $colorInAssocArr . "\n";
```

Output:

```
Getting the value of color: purple
```

c. **Add a new element**

When adding a new element, PHP uses the “”+” symbol, syntax is such as: name_of_the_array += [new_key => new_value]. It adds to the end of the Array.

Code:

```
$assocArr += ["language" => "php"];
```

Output:

Key :color, value : purple

Key :number, value : 2

Key :city, value : ankara

Key :class, value : CS-315

Key :language, value : php

d. **Remove an existing element**

PHP has a built in function that removes a given key-value from Array. The keyword is “unset”.

Code:

```
unset($assocArr["class"]);
```

Output:

Key :color, value : purple

Key :number, value : 2

Key :city, value : ankara

Key :language, value : php

e. **Modify the value of an existing element**

By using $name_of_the_object[existing_key] = new_value the value of a given key can be changed in PHP.

Code:

```
$assocArr["number"] = 13;
```

Output:

Key : Language value :dart

Key : city value :ankara

Key : color value :purple

Key : number value : 13

f. Search for the existence of a key

PHP has a built in function called array_key_exists(key, array) that searches for the existence of a key but it returns either 1 or 0. So I converted them to true or false.

Code:

```
echo "Searching if color key exists: " ,  array_key_exists("color", $assocArr) ? "true\n" : "false\n";

echo "\nSearching if name key exists: " , array_key_exists("name", $assocArr) ? "true\n": "false\n";
```

Output:

Searching if color key exists: true

Searching if name key exists: false

g. Search for the existence of a value

PHP has a built in function called in_array(value, array) that searches for the existence of a value but it returns either 1 or 0. So I converted them to true or false.

Code:

```
echo "\nSearching if ankara value exists: " , in_array("ankara", $assocArr) ? "true\n": "false\n" . "\n";

echo "Searching if istanbul value exists: " , in_array("istanbul", $assocArr) ? "true\n": "false\n" . "\n";
```

Output:

Searching if ankara value exists: true

Searching if istanbul value exists: false

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

By using a simple for loop i called the foo function while iterating through the Map.

Code:

```
function foo($key, $value) {
    echo "Key :" . $key . ", value : " . $value;
}
foreach ($assocArr as $key => $value) {
    foo($key,$value);
    echo "\n";
  }
```

Output:

```
Key :color, value : purple
Key :number, value : 13
Key :city, value : ankara
Key :Language, value : php
```

i. **Evaluation of PHP language in terms of readability and writability of associative array data structure.**

1. **Readability**

   In terms of associative array data structure, PHP is very well readable with English built-in function names. But dollar sign in front of the variables and some of the functions like searching for the key and value returning 1's and 0's makes PHP harder to read.

2. **Writability**

   In terms of associative array data structure, PHP is sufficient with components like English built-in function names and clear syntax while declaration. But dollar sign in front of the variables and some of the functions like searching for the key and value returning 1's and 0's makes PHP harder to write.

4. **Python**

Associative arrays are called Dictionaries in Python programming language. There is no need to specify the associative array as Dictionaries, Python understands the key-value match by syntax.

a. **Initialize**

Code:

```
assocArr = {'color': 'purple',
        'number': 2,
        'city': 'ankara',
        'class': 'CS-315'}
print(assocArr)
```

Output:

```
{'color': 'purple', 'number': 2, 'city': 'ankara', 'class': 'CS-315'}
```

b. Get the value for a given key

I will use the the Dictionary above. Value of a key can be reached using name_of_the_dictionary[key] syntax.

Code:

```
colorInAssocArr = assocArr['color']
print("\nGetting the value of color: " + colorInAssocArr)
```

Output:

```
Getting the value of color: purple
```

c. Add a new element

When adding a new element, Python uses name_of_the_dictionary[new_key] = new_value syntax. It adds to the end of the dictionary.

Code:

```
assocArr['Language'] = 'python'
```

Output:

```
{'color': 'purple', 'number': 2, 'city': 'ankara', 'class': 'CS-315', 'Language': 'python'}
```

d. Remove an existing element

Python has a built in function that removes a given key-value from dictionary.

Code:

```
assocArr.pop('class')
```

Output:

Key : color value :purple

Key : number value :2

Key : Language value :dart

Key : city value :ankara

e. **Modify the value of an existing element**

By using name_of_the_dictionary[existing_key] = new_value the value of a given key can be changed in Python.

Code:

```
assocArr['number'] = '13';
```

Output:

{'color': 'purple', 'number': 2, 'city': 'ankara', 'Language': 'dart'}

f. **Search for the existence of a key**

Python has a in component that returns boolean to search for the existence of a key. While searching for the key, “in” keyword is used.

Code:

```
containsName = False

containsColor = False

if 'name' in assocArr.keys():

    containsName = True

if 'color' in assocArr.keys():

    containsColor = True

print("\nSearching if color key exists: " + str(containsColor))

print("Searching if name key exists: " + str(containsName))
```

Output:

Searching if color key exists: True

Searching if name key exists: False

g. **Search for the existence of a value**

Python has a in component that returns boolean to search for the existence of a key. While searching for the key, “in” keyword is used. To reach the values in Python array_name.values can be used.

Code:

```
containsAnkara = False

containsIstanbul = False

if 'ankara' in assocArr.values():

    containsAnkara = True

if 'istanbul' in assocArr.values():

    containsIstanbul = True

print("\nSearching if ankara value exists: " + str(containsAnkara))

print("Searching if istanbul value exists: " + str(containsIstanbul) + "\n")
```

Output:

```
Searching if ankara value exists: True

Searching if istanbul value exists: False
```

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

By using a simple for loop i called the foo function while iterating through the Dictionary.

Code:

```
for k in assocArr:

    foo(k, assocArr[k])

def foo(key, value):

    print("Key : " + key + "  value : " + value)
```

Output:

```
Key : color  value : purple

Key : number  value : 13

Key : city  value : ankara

Key : Language  value : python
```

i. **Evaluation of Python in terms of readability and writability of associative array data structure.**

1. **Readability**

   Dictionary provides easy readability with easily understandable English function names like in. But the name pop to delete a key-value was not as very clear. It requires characters such as ":" and "{}" while declaring the array which make the language more readable. Therefore Python is easily can be read in terms of associative array data structure.

2. **Writability**

   Python can be easily written as well because it separates keys and values using "{}" and ":". In terms of associative arrays Python is a easily writable language.

5. Ruby

Associative arrays are called Hash in Ruby programming language. There is no need to specify the associative array as Hashes, Ruby understands the key-value match by syntax.

a. **Initialize**

Code:

```
assocArr = {"color" => "purple", "number" => 2, "city" => "ankara", "class" => "CS-315"};

print(assocArr);
```

Output:

```
{"color"=>"purple", "number"=>2, "city"=>"ankara", "class"=>"CS-315"}
```

b. **Get the value for a given key**

I will use the the Hash above. Value of a key can be reached using name_of_the_hash[key] syntax.

Code:

```
colorInAssocArr = assocArr["color"];

print("\nGetting the value of color: " + colorInAssocArr + "\n");
```

Output:

```
Getting the value of color: purple
```

c. **Add a new element**

When adding a new element, Ruby uses name_of_the_hash[new_key] = new_value syntax. It adds to the end of the Hash.

Code:

```
assocArr["language"] = "ruby";
```

Output:

```
{"color"=>"purple", "number"=>2, "city"=>"ankara", "class"=>"CS-315", "language"=>"ruby"}
```

d. Remove an existing element

Ruby has a built in function that removes a given key-value from Hash.

Code:

```
assocArr.delete("class");
```

Output:

```
{"color"=>"purple", "number"=>2, "city"=>"ankara", "language"=>"ruby"}
```

e. Modify the value of an existing element

By using name_of_the_hash[existing_key] = new_value the value of a given key can be changed in Hash.

Code:

```
assocArr['number'] = '13';
```

Output:

```
{"color"=>"purple", "number"=>2, "city"=>"ankara", "language"=>"ruby"}
```

f. Search for the existence of a key

Ruby does not have a built in function that returns boolean to search for the existence of a key but it can be obtained by doing name_of_the_hash.key?(name_of_the_key).

Code:

```
print "Searching if color key exists: " , assocArr.key?("color") , "\n";

print "Searching if name key exists: " , assocArr.key?("name"), "\n\n";
```

Output:

```
Searching if color key exists: true

Searching if name key exists: false
```

g. **Search for the existence of a value**

Ruby does not have a built in function that returns boolean to search for the existence of a value but it can be obtained by doing name_of_the_hash.has_value?(name_of_the_the).

Code:

```
print "Searching if ankara value exists: " , assocArr.has_value?("ankara"), "\n";

print "Searching if istanbul value exists: " , assocArr.has_value?("istanbul"), "\n\n";
```

Output:

Searching if ankara value exists: true

Searching if istanbul value exists: false

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

By using a simple for loop i called the foo function while iterating through the Hash. In Ruby, component we know as for is used with a different syntax. It uses the keyword “do“.

Code:

```
def foo(key, value)
   print "Key :" , key , ", value : " , value , "\n";
 end
assocArr.each do |key,value|
   foo(key,value);
end
```

Output:

Key :color, value : purple

Key :number, value : 13

Key :city, value : ankara

Key :language, value : ruby

i. **Evaluation of Rubylanguage in terms of readability and writability of associative array data structure.**

1. **Readability**

Object provides easy readability with easily understandable English function names like delete. It requires characters such as “=>” and “{}” which make the language more readable. Therefore Ruby is easily can be read in terms of associative array data structure. Rubly lacks clarity only during searching for keys and values.

2. **Writability**

Ruby can be easily written as well because it separates keys and values using “{}” and “=>”. In terms of associative arrays Ruby is a easily writable language. Except for searching if keys and values exist.

6. Rust

Associative arrays are called HashMaps in Rust programming language.

a. Initialize

Ruby requires specification while initializing the Hashmap. If the HashMap needs to be muted, mut keyword is being put before the variable name. Ruby requires “[]”, “[]” and “,” characters to declare a Hashmap. It does not allow heterogenous Hashmaps, all keys and values have to be of the same type. It also requires .iter().cloned().collect() built-in functions to declare a HashMap.

Code:

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

 let mut assoc_arr: HashMap<&str, &str> =

    [("color", "purple"),

            ("number", "2"),

            ("city", "ankara"),

            ("class", "CS-315")].iter().cloned().collect();
```

Output:

```
{"color": "purple", "city": "ankara", "class": "CS-315", "number": "2"}
```

b. Get the value for a given key

I will use the the HasMap above. Value of a key can be reached using name_of_the_hashmap.get(key).unwrap(); syntax.

Code:

```
println!("\nGetting the value of color: {:?}", assoc_arr.get("color").unwrap());
```

Output:

Getting the value of color: purple

c. Add a new element

When adding a new element, Rust uses name_of_the_map.insert(new_key, new_value) syntax. It adds to the start of the Hashmap.

Code:

assoc_arr.insert("language","rust");

Output:

Key : language, value : rust

Key : number, value : 2

Key : city, value : ankara

Key : class, value : CS-315

Key : color, value : purple

d. Remove an existing element

Rust has a built in function that removes a given key-value from HashMap.

Code:

assoc_arr.remove("class");

Output:

Key : city, value : ankara

Key : language, value : rust

Key : color, value : purple

Key : number, value : 2

e. Modify the value of an existing element

By using *name_of_the_hashmap.get_mut(key).unwrap = new_value syntax the value of a given key can be changed in Hashmap.

Code:

*assoc_arr.get_mut("number").unwrap() = "13";

Output:

Getting the value of color: "purple"

Key : language, value : rust

Key : color, value : purple

Key : number, value : 13

Key : city, value : ankara

f. Search for the existence of a key

Rust has a built in function that returns boolean to search for the existence of a key.

Code:

```
 println!("Searching if color key exists: {:?}", assoc_arr.contains_key("color"));

println!("Searching if name key exists: {:?}", assoc_arr.contains_key("name"));
```

Output:

Searching if color key exists: true

Searching if name key exists: false

g. Search for the existence of a value

Rust does not have a built in function that returns boolean to search for the existence of a value. But basically it can be found using name_of_the_hashmap.values and “==” components.

Code:

```
println!("Searching if ankara value exists: {:?}", assoc_arr.values().any(|&value| value == "ankara"));

   println!("Searching if ankara value exists: {:?}", assoc_arr.values().any(|&value| value == "istanbul" ));
```

Output:

Searching if ankara value exists: true

Searching if istanbul value exists: false

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

By using a simple for loop i called the foo function while iterating through the HashMap.

Code:

```
fn foo(key:&str,value:&str){

    println!("Key : {}, value : {}" , key , value);

}

 for (k, v) in &assoc_arr {

        foo(k,v);

    }
```

Output:

Key : number, value : 13

Key : color, value : purple

Key : city, value : ankara

Key : language, value : rust

i. **Evaluation of Rust language in terms of readability and writability of associative array data structure.**

1. **Readability**

   Rust is not a readable language. It requires extra symbols, built-in functions and syntactic rules to do basic operations compared to other languages used above. Heterogenous arrays are not allowed. Rust is the hardest language to read thus far.

2. **Writability**

   As well as Rust requiring extra symbols, built-in functions and syntactic rules to do basic operations compared to other languages used above, writing even printing outputs requires extra work that makes it confusing to write Rust. I would not recommend Rust in terms of associative arrays as it is not writeable.

7. Learning Strategy

While writing all of the programs, I decided to use VSCode for all languages except for Rust to be more efficient and fast when running the code. I used official documents of the programming languages and websites such as StackOverflow and W3schools. I first researched what associative arrays are called in that language while I was writing. Then i searched how to declare it and its built-in functions. I also experimented whether that languages supports different types of variables in the associative array. Here are the resources I used:

Dart official documentation: https://dart.dev/guides

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

Lua official documentation: https://www.lua.org/docs.html

PHP official documentation: https://www.php.net/docs.php

Python official documentation:https://docs.python.org/3/

Ruby official documentation: https://www.ruby-lang.org/en/documentation/

Rust official documentation: https://www.rust-lang.org/learn

Additional Websites:

Rust compiler website: https://play.rust-lang.org/

W3schools: https://www.w3schools.com/

StackOverflow: https://stackoverflow.com/

WordPress StackExchange: https://wordpress.stackexchange.com/

8. Conclusion

From my observations, the best programming language for associative arrays is Dart. It has a built in function for every operation, adding removing, searching, printing etc. Only criticizable part of Dart is that it requires a dollar sign before variable names. Worst programming language for associative arras is Rust. Rust was the hardest to learn and write. Rust requires extra symbols, built-in functions and syntactic rules to do basic operations compared to other languages used.

9. Additional Notes

I did not specify how I printed the arrays after doing every operation. I used the print function if the language has one for associative arrays or I used the function I wrote myself.