



CS315 PROGRAMMING LANGUAGES HW-1

Mahmut Furkan Gön

21902582

Section 3

Table of Content

1. Sample Code Segments and the Results	3
1.1. Dart	3
1.1.1. Initialize	3
1.1.2. Get the value for a given key	4
1.1.3. Add a new element	5
1.1.4. Remove an element	5
1.1.5. Modify the value of an existing element	5
1.1.6. Search for the existence of a key	6
1.1.7. Search for the existence of a value	6
1.1.8. Loop through an associative array, and apply a function called foo, which simply prints the key-value pair	7
1.2. Javascript	8
1.2.1. Initialize	8
1.2.2. Get the value for a given key	9
1.2.3. Add a new element	9
1.2.4. Remove an element	10
1.2.5. Modify the value of an existing element	11
1.2.6. Search for the existence of a key	12
1.2.7. Search for the existence of a value	12
1.2.8. Loop through an associative array, and apply a function called foo, which simply prints the key-value pair	13
1.3. Lua	14
1.3.1. Initialize	14
1.3.2. Get the value for a given key	15
1.3.3. Add a new element	15
1.3.4. Remove an element	15
1.3.5. Modify the value of an existing element	16
1.3.6. Search for the existence of a key	16
1.3.7. Search for the existence of a value	17
1.3.8. Loop through an associative array, and apply a function called foo, which simply prints the key-value pair	17
1.4. PHP	19
1.4.1. Initialize	19
1.4.2. Get the value for a given key	20
1.4.3. Add a new element	21
1.4.4. Remove an element	22
1.4.5. Modify the value of an existing element	23
1.4.6. Search for the existence of a key	24
1.4.7. Search for the existence of a value	24
1.4.8. Loop through an associative array, and apply a function called foo, which simply prints the key-value pair	24

1.5. Python	26
1.5.1. Initialize	26
1.5.2. Get the value for a given key	27
1.5.3. Add a new element	27
1.5.4. Remove an element	27
1.5.5. Modify the value of an existing element	28
1.5.6. Search for the existence of a key	28
1.5.7. Search for the existence of a value	29
1.5.8. Loop through an associative array, and apply a function called foo, which simply prints the key-value pair	29
1.6. Ruby	31
1.6.1. Initialize	31
1.6.2. Get the value for a given key	32
1.6.3. Add a new element	32
1.6.4. Remove an element	32
1.6.5. Modify the value of an existing element	33
1.6.6. Search for the existence of a key	33
1.6.7. Search for the existence of a value	33
1.6.8. Loop through an associative array, and apply a function called foo, which simply prints the key-value pair	34
1.7. Rust	35
1.7.1. Initialize	35
1.7.2. Get the value for a given key	36
1.7.3. Add a new element	36
1.7.4. Remove an element	36
1.7.5. Modify the value of an existing element	37
1.7.6. Search for the existence of a key	37
1.7.7. Search for the existence of a value	38
1.7.8. Loop through an associative array, and apply a function called foo, which simply prints the key-value pair	38
2. Evaluation of the Languages	39
3. Learning Strategy	40

1. Sample Code Segments and the Results

I was reading a book named Mesnevi-i Nuriye when this homework assignment was announced. That's why I will use this book in the code samples. I will be using the same variable and statements throughout homework for the sake of readability and readability..

1.1. Dart

1.1.1. Initialize

```
var mesneviiNuriye = {  
  "Mukaddime" : 3,  
  "Lem'alar" : 11,  
  "Reshalar" : 12,  
  "Lasiyyemalar" : 16,  
  "Katre" : 33,  
  "Hubab" : 34,  
  "Habbe" : 32,  
  "Zuhre" : 32,  
  "Zerre" : 11,  
  "Semme" : 13,  
  "Onuncu Risale" : 25,  
  "14. Resha" : 6,  
  "Sule" : 9,  
  "Nokta" : 14,  
};
```

Fig. 1: Initialization in Dart

The associated array (Map) object mesneviiNuriye is initialized in the code samples below and above. The associated array keeps the number of pages of each chapter in Mesnevi-i Nuriye. There is no warning in the first way of initialization; however, there is a warning in the second way, saying, "Use collection literals when possible." Although it warns us in the second way, both code segments work without error and initialize the associated array objects.

```

var mesneviiNuriye = Map<String, int>();
mesneviiNuriye["Mukaddime"] = 3;
mesneviiNuriye["Lem'alar"] = 11;
mesneviiNuriye["Reshalar"] = 12;
mesneviiNuriye["Lasiyyemalar"] = 16;
mesneviiNuriye["Katre"] = 33;
mesneviiNuriye["Hubab"] = 34;
mesneviiNuriye["Habbe"] = 32;
mesneviiNuriye["Zuhre"] = 32;
mesneviiNuriye["Zerre"] = 11;
mesneviiNuriye["Semme"] = 13;
mesneviiNuriye["Onuncu Risale"] = 25;
mesneviiNuriye["14. Resha"] = 6;
mesneviiNuriye["Sule"] = 9;
mesneviiNuriye["Nokta"] = 14;

```

Fig. 2: Second way of initialization in Dart

The initialized associated array looks like this when it is printed on the console:

```

{Mukaddime: 3, Lem'alar: 11, Reshalar: 12, Lasiyyemalar: 16, Katre: 33,
Hubab: 34, Habbe: 32, Zuhre: 32, Zerre: 11, Semme: 13, Onuncu Risale: 25,
14. Resha: 6, Sule: 9, Nokta: 14}

```

1.1.2. Get the value for a given key

<pre>print(mesneviiNuriye["Mukaddime"]);</pre>	3
<pre>print(mesneviiNuriye["Katre"]);</pre>	33
<pre>print(mesneviiNuriye["Hubab"]);</pre>	34
<pre>print(mesneviiNuriye["Nokta"]);</pre>	14

Fig. 3: Getting the value of a given key in Dart

In the code sample above, the syntax for getting the value of a given key is visible. When developers want to get the value of a key, they should put square brackets ([]) immediately after the name of the associated array object they are looking for and write the key inside square brackets. We see from above sample that Mukaddime is 3, Katre 33, Hubab 34, Nokta 14 pages long.

1.1.3. Add a new element

```
// Add a new item
mesneviiNuriye["Fihrist"] = 10;
mesneviiNuriye["Itizar"] = 1;
```

Fig. 4: Add a new element in Dart

To add a new element to an associated array in the Dart, developers must specify the key and value using subscript assignment operator ([]) as seen in the above example.

The resulting associated array looks like this when it is printed on the console:

```
{Mukaddime: 3, Lem'alar: 11, Reshalar: 12, Lasiyyemalar: 16, Katre: 33,
Hubab: 34, Habbe: 32, Zuhre: 32, Zerre: 11, Semme: 13, Onuncu Risale: 25,
14. Resha: 6, Sule: 9, Nokta: 14, Fihrist: 10, Itizar: 1}
```

1.1.4. Remove an element

```
// Remove an element
mesneviiNuriye.remove("Fihrist");
```

Fig. 5: Remove an element in Dart

Above code sample shows how to remove an element from an associated array object in Dart. General syntax is like `map_name.remove(key)`.

The resulting associated array looks like this when it is printed on the console:

```
{Mukaddime: 3, Lem'alar: 11, Reshalar: 12, Lasiyyemalar: 16, Katre: 33,
Hubab: 34, Habbe: 32, Zuhre: 32, Zerre: 11, Semme: 13, Onuncu Risale: 25,
14. Resha: 6, Sule: 9, Nokta: 14, Itizar: 1}
```

1.1.5. Modify the value of an existing element

```
// Modify an existing element
mesneviiNuriye.update("Itizar",
                      (existingValue) => 3,
                      ifAbsent: () => 3,);
```

Fig 6: Modify the value of an existing element in Dart

The above code sample shows how to modify the value of an existing element in associated array objects in Dart language. Users should specify the key, then specify the new values if

there is a value corresponding to the given key or not. The code sample below depicts a more readable way of updating the value. This syntax looks the same as adding a new item to the map object in Dart. However, this time the value of the key is updated. This syntax might cause problems, like adding a new element to the map objects when typos occur.

```
mesneviiNuriye["Itizar"] = 3;
```

Fig. 7: Second way of modifying the value of an existing element in Dart

The resulting associated array looks like this when it is printed on the console:

```
{Mukaddime: 3, Lem'alar: 11, Reshalar: 12, Lasiyyemalar: 16, Katre: 33, Hubab: 34, Habbe: 32, Zuhre: 32, Zerre: 11, Semme: 13, Onuncu Risale: 25, 14. Resha: 6, Sule: 9, Nokta: 14, Itizar: 3}
```

Please note that the last element, Itizar's value, is updated from 1 to 3.

1.1.6. Search for the existence of a key

```
print(mesneviiNuriye.containsKey("Habbe"));  
print(mesneviiNuriye.containsKey("Sabah"));
```

Fig. 8: Searching for the existence of a key in Dart

As it can be seen in above example, people can search for existing of a key by using `map_name.containsKey(Key)` syntax. The result is a boolean value. For the sample above, first returns true and second returns false.

1.1.7. Search for the existence of a value

```
print(mesneviiNuriye.containsValue(12));  
print(mesneviiNuriye.containsValue(33));  
print(mesneviiNuriye.containsValue(38));
```

Fig. 9: Searching for the existence of a value in Dart

As it can be seen in the above example, people can search for existing of a value by using `map_name.containsValue(Value)` syntax. The result is a boolean value. For the sample above, first two returns true and the third returns false.

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

```
// Function foo()-----  
print("Function foo-----");  
  
mesneviiNuriye.forEach(foo);  
}  
  
void foo(var key, var value){  
    print("mesneviiNuriye[$key] = $value");  
}
```

Fig. 9: foo function and for loop printing pairs in Dart

As it can be seen in the above sample, foo function prints the entries of the map object and gets the keys and values of the entries. At the end it prints the key and value pairs of the map object, mesneviiNuriye, using built-in forEach loop. forEach calls foo function for each entry of mesneviiNuriye.

If we provoke the function with parameter mesneviiNuriye, the result will be like in Fig. 10.

```
Function foo-----  
mesneviiNuriye[Mukaddime] = 3  
mesneviiNuriye[Lem'alar] = 11  
mesneviiNuriye[Reshalar] = 12  
mesneviiNuriye[Lasiyyemalar] = 16  
mesneviiNuriye[Katre] = 33  
mesneviiNuriye[Hubab] = 34  
mesneviiNuriye[Habbe] = 32  
mesneviiNuriye[Zuhre] = 32  
mesneviiNuriye[Zerre] = 11  
mesneviiNuriye[Semme] = 13  
mesneviiNuriye[Onuncu Risale] = 25  
mesneviiNuriye[14. Resha] = 6  
mesneviiNuriye[Sule] = 9  
mesneviiNuriye[Nokta] = 14  
mesneviiNuriye[Itizar] = 3
```

Fig. 10: Result of the foo function

1.2. Javascript

In code submission, we buried our Javascript code into HTML tags. When I run the HTML file and inspect the code in the browser, my browser prints only the final version of the map object which has 15 elements. However, when I try my code on a different computer and browser, it runs perfectly, as Javascript code runs properly on an online compiler that I provided its link at the end. If my code doesn't run on your browser, please take this warning into consideration and try the Javascript code using the online compiler.

1.2.1. Initialize

```
const mesneviiNuriye = new Map();
mesneviiNuriye.set('Mukaddime', 3);
mesneviiNuriye.set('Lemalar', 11);
mesneviiNuriye.set('Reshalar', 12);
mesneviiNuriye.set('Lasiyyemalar', 16);
mesneviiNuriye.set('Katre', 33);
mesneviiNuriye.set('Hubab', 34);
mesneviiNuriye.set('Habbe', 32);
mesneviiNuriye.set('Zuhre', 32);
mesneviiNuriye.set('Zerre', 11);
mesneviiNuriye.set('Semme', 13);
mesneviiNuriye.set('Onuncu Risale', 25);
mesneviiNuriye.set('14. Resha', 6);
mesneviiNuriye.set('Sule', 9);
mesneviiNuriye.set('Nokta', 14);
```

Fig. 11: Initialization in Javascript

The associated array (Map) object mesneviiNuriye is initialized in the code samples above. The associated array keeps the number of pages of each chapter in Mesnevi-i Nuriye. If we print the associated array, we will see an output like the below:

```
Map {
  'Mukaddime' => 3,
  'Lemalar' => 11,
  'Reshalar' => 12,
  'Lasiyyemalar' => 16,
```

```
'Katre' => 33,
'Hubab' => 34,
'Habbe' => 32,
'Zuhre' => 32,
'Zerre' => 11,
'Semme' => 13,
'Onuncu Risale' => 25,
'14. Resha' => 6,
'Sule' => 9,
'Nokta' => 14 }
```

1.2.2. Get the value for a given key

```
console.log('Get the value of a key-----');
console.log(mesneviiNuriye.get('Mukaddime'));
console.log(mesneviiNuriye.get('Katre'));
console.log(mesneviiNuriye.get('Hubab'));
console.log(mesneviiNuriye.get('Nokta'));
Add a new item-----
```

Fig. 12: Getting the value of a given key in Javascript

In the code sample above, the syntax for getting the value of a given key is visible. When developers want to get the value of a key, they should use built-in function immediately after the name of the associated array object they are looking for and write the key as the parameter for the get function. We see from the above sample that Mukaddime is 3, Katre 33, Hubab 34, Nokta 14 pages long.

1.2.3. Add a new element

```
// Add a new item
console.log('Add a new item-----');

mesneviiNuriye.set('Fihrist', 10);
mesneviiNuriye.set('Itizar', 1);
```

Fig. 13: Add a new element in Javascript

To add a new element to an associated array in Javascript, developers must specify the key and value using set built-in function as seen in the above example.

The resulting associated array looks like this when it is printed on the console:

```
Map {
  'Mukaddime' => 3,
  'Lemalar' => 11,
  'Reshalar' => 12,
  'Lasiyyemalar' => 16,
  'Katre' => 33,
  'Hubab' => 34,
  'Habbe' => 32,
  'Zuhre' => 32,
  'Zerre' => 11,
  'Semme' => 13,
  'Onuncu Risale' => 25,
  '14. Resha' => 6,
  'Sule' => 9,
  'Nokta' => 14,
  'Fihrist' => 10,
  'Itizar' => 1 }
```

1.2.4. Remove an element

```
// Remove an element
console.log('Remove an item-----');

mesneviNuriye.delete('Fihrist');
```

Fig. 14: Remove an element in Javascript

Above code sample shows how to remove an element from an associated array object in Javascript. General syntax is like `map_name.delete(key)`.

The resulting associated array looks like this when it is printed on the console:

```
Map {
  'Mukaddime' => 3,
  'Lemalar' => 11,
  'Reshalar' => 12,
  'Lasiyyemalar' => 16,
  'Katre' => 33,
  'Hubab' => 34,
  'Habbe' => 32,
  'Zuhre' => 32,
```

```
'Zerre' => 11,  
'Semme' => 13,  
'Onuncu Risale' => 25,  
'14. Resha' => 6,  
'Sule' => 9,  
'Nokta' => 14,  
'Itizar' => 1 }
```

1.2.5. Modify the value of an existing element

```
// Modify an existing element  
console.log('Modify an existing item-----');  
mesneviiNuriye.set('Itizar', 3);  
  
console.log(mesneviiNuriye);
```

Fig 15: Modify the value of an existing element in Javascript

The above code sample shows how to modify the value of an existing element in associated array objects in Javascript. Users should specify the key, then specify the new value as parameters to the set built-in function.

The resulting associated array looks like this when it is printed on the console:

```
Map {  
  'Mukaddime' => 3,  
  'Lemalar' => 11,  
  'Reshalar' => 12,  
  'Lasiyyemalar' => 16,  
  'Katre' => 33,  
  'Hubab' => 34,  
  'Habbe' => 32,  
  'Zuhre' => 32,  
  'Zerre' => 11,  
  'Semme' => 13,  
  'Onuncu Risale' => 25,  
  '14. Resha' => 6,  
  'Sule' => 9,  
  'Nokta' => 14,  
  'Itizar' => 3 }
```

1.2.6. Search for the existence of a key

```
// Existence of a key
console.log('Contains Key-----|');

console.log(mesneviiNuriye.has('Habbe'));
console.log(mesneviiNuriye.has('Sabah'));
```

Fig. 16: Searching for the existence of a key in Javascript

As it can be seen in above example, people can search for existing of a key by using `map_name.has(Key)` syntax. The result is a boolean value. For the sample above, first returns true and second returns false.

1.2.7. Search for the existence of a value

```
// Existence of a vaule
console.log("Contains Value-----");

const values = [...mesneviiNuriye.values()];
console.log(values.includes(12));
console.log(values.includes(33));
console.log(values.includes(38));
```

Fig. 17: Searching for the existence of a value in Javascript

As it can be seen in the above example, people firstly should create an array using `[...map_name.values()]` syntax. Then, they can search for existing of a value by using `values.has(Value)` syntax. The result is a boolean value. For the sample above, first two returns true and the third returns false.

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

```
// Function foo()-----  
console.log('Function foo-----');  
  
function foo(value, key, map){  
    console.log(`mesneviiNuriye[${key}] = ${value}`);  
}  
mesneviiNuriye.forEach(foo);
```

Fig. 18: foo function and for loop printing pairs in Javascript

As it can be seen in the above sample, foo function prints the entries of the map object and gets the key, value and the map object itself. At the end it prints the key and value pairs of MesneviiNuriye using built-in forEach loop.

If we provoke the function with parameter mesneviiNuriye, the result will be like:

```
Function foo-----  
mesneviiNuriye[Mukaddime] = 3  
mesneviiNuriye[Lemalar] = 11  
mesneviiNuriye[Reshalar] = 12  
mesneviiNuriye[Lasiyyemalar] = 16  
mesneviiNuriye[Katre] = 33  
mesneviiNuriye[Hubab] = 34  
mesneviiNuriye[Habbe] = 32  
mesneviiNuriye[Zuhre] = 32  
mesneviiNuriye[Zerre] = 11  
mesneviiNuriye[Semme] = 13  
mesneviiNuriye[Onuncu Risale] = 25  
mesneviiNuriye[14. Resha] = 6  
mesneviiNuriye[Sule] = 9  
mesneviiNuriye[Nokta] = 14  
mesneviiNuriye[Itizar] = 3
```

Fig. 19: Result of the foo function

1.3. Lua

1.3.1. Initialize

```
mesneviiNuriye = {}

mesneviiNuriye["Mukaddime"] = 3
mesneviiNuriye["Lemalar"] = 11
mesneviiNuriye["Reshalar"] = 12
mesneviiNuriye["Lasiyyemalar"] = 16
mesneviiNuriye["Katre"] = 33
mesneviiNuriye["Hubab"] = 34
mesneviiNuriye["Habbe"] = 32
mesneviiNuriye["Zuhre"] = 32
mesneviiNuriye["Zerre"] = 11
mesneviiNuriye["Semme"] = 13
mesneviiNuriye["Onuncu Risale"] = 25
mesneviiNuriye["14. Resha"] = 6
mesneviiNuriye["Sule"] = 9
mesneviiNuriye["Nokta"] = 14
```

Fig. 20: Initialization in Lua

The associated array (Map) object `mesneviiNuriye` is initialized in the code samples above. The associated array keeps the number of pages of each chapter in Mesnevi-i Nuriye. The associated array object first declared by using braces, `{}`. Then, the entries are entered using brackets and assignment operator, `[] =`. Since the maps are implemented using tables in Lua, we will see a random numbers indicating the address like:

table: 0x559d1409e650

1.3.2. Get the value for a given key

```
print('Get the value of a key-----')
print(mesneviiNuriye["Mukaddime"])
print(mesneviiNuriye["Katre"])
print(mesneviiNuriye.Hubab)
print(mesneviiNuriye.Nokta)
```

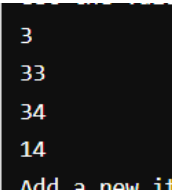


Fig. 21: Getting the value of a given key in Lua

In the code sample above, the syntax for getting the value of a given key is visible. When developers want to get the value of a key, they could use two methods. They can get the value using brackets and keys inside, or can get the values using directly `.keyName` syntax. Both approaches are visible in the above code sample.

1.3.3. Add a new element

```
print('Add a new item-----')

mesneviiNuriye["Fihrist"] = 10
mesneviiNuriye["Itizar"] = 1
```

Fig. 22: Add a new element in Lua

To add a new element to an associated array in Lua, developers must specify the key and value using brackets and assignment operators as seen in the above example.

1.3.4. Remove an element

```
mesneviiNuriye["Fihrist"] = nil
print(mesneviiNuriye["Fihrist"])
```

Fig. 23: Remove an element in Lua

Above code sample shows how to remove an element from an associated array object in Lua. General syntax is like `map_name[key] = nil`. `nil` value is a specific values like null value in other languages. Basically, we set the given part of the table `nil` and the compiler will ignore it when take the map object into consideration.

1.3.5. Modify the value of an existing element

```
mesneviNuriye["Itizar"] = 3
```

Fig 24: Modify the value of an existing element in Lua

The above code sample shows how to modify the value of an existing element in associated array objects in Lua. Users should specify the key, then specify the new value using brackets and assignment operators as seen above.

1.3.6. Search for the existence of a key

```
print(mesneviNuriye["Habbe"] ~= nil)
print(mesneviNuriye["Sabah"] ~= nil)
```

Fig. 25: Searching for the existence of a key in Lua

As it can be seen in above example, people can search for existing of a key by using `map_name.has[key] ~= nil` syntax. The result is a boolean value. For the sample above, first returns true and second returns false.

1.3.7. Search for the existence of a value

```
function contains(table, element)
    for _, value in pairs(table) do
        if value == element then
            return true
        end
    end
    return false
end

print(contains(mesneviiNuriye, 12))
print(contains(mesneviiNuriye, 33))
print(contains(mesneviiNuriye, 38))
```

Fig. 26: Searching for the existence of a value in Lua

As it can be seen in the above example, people firstly should create a function that returns if the map object contains the given value. Then, they can search for existing of a value by using this function. For the sample above, first two returns true and the third returns false.

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

```
function foo(key, value)
    print(key, "=>", value)
end

for k,v in pairs(mesneviiNuriye) do
    foo(k, v)
end
```

Fig. 27: foo function and for loop printing pairs in Lua

As it can be seen in the above sample, foo function prints the entries of the map object and gets the key, value and the map object itself. At the end it prints the key and value pairs of MesneviiNuriye using for loop.

If we provoke the function with parameter mesneviiNuriye, the result will be like:

```
Function foo-----  
Mukaddime => 3  
Hubab => 34  
Semme => 13  
Katre => 33  
Lasiyyemalar => 16  
Lemalar => 11  
Nokta => 14  
Onuncu Risale => 25  
Sule => 9  
Habbe => 32  
Zuhre => 32  
Reshalar => 12  
Itizar => 3  
14. Resha => 6  
Zerre => 11
```

Fig. 28: Result of foo function in Lua

1.4. PHP

1.4.1. Initialize

```
/*$mesneviiNuriye["Mukaddime"] = 3;
$mesneviiNuriye["Lem'alar"] = 11;
$mesneviiNuriye["Reshalar"] = 12;
$mesneviiNuriye["Lasiyyemalar"] = 16;
$mesneviiNuriye["Katre"] = 33;
$mesneviiNuriye["Hubab"] = 34;
$mesneviiNuriye["Habbe"] = 32;
$mesneviiNuriye["Zuhre"] = 32;
$mesneviiNuriye["Zerre"] = 11;
$mesneviiNuriye["Semme"] = 13;
$mesneviiNuriye["Onuncu Risale"] = 25;
$mesneviiNuriye["14. Resha"] = 6;
$mesneviiNuriye["Sule"] = 9;
$mesneviiNuriye["Nokta"] = 14;

*/

$mesneviiNuriye = array(
    "Mukaddime" => 3,
    "Lem'alar" => 11,
    "Reshalar" => 12,
    "Lasiyyemalar" => 16,
    "Katre" => 33,
    "Hubab" => 34,
    "Habbe" => 32,
    "Zuhre" => 32,
    "Zerre" => 11,
    "Semme" => 13,
    "Onuncu Risale" => 25,
    "14. Resha" => 6,
    "Sule" => 9,
    "Nokta" => 14,
);
```

Fig. 29: Two ways of initialization in PHP

The associated array (Map) object `mesneviiNuriye` is initialized in the code samples above. The associated array keeps the number of pages of each chapter in Mesnevi-i Nuriye. As sample shows there are two ways of initialization a map object. It can be initialized by brackets and

assignment operator, similar to Lua. The other approach is creating an array object and giving the key and values and arrows , =>, between them. If we print the map object just initialized, the output will be like:

```
array(14) {
    ["Mukaddime"]=>
    int(3)
    ["Lem'alar"]=>
    int(11)
    ["Reshalar"]=>
    int(12)
    ["Lasiyyemalar"]=>
    int(16)
    ["Katre"]=>
    int(33)
    ["Hubab"]=>
    int(34)
    ["Habbe"]=>
    int(32)
    ["Zuhre"]=>
    int(32)
    ["Zerre"]=>
    int(11)
    ["Semme"]=>
    int(13)
    ["Onuncu Risale"]=>
    int(25)
    ["14. Resha"]=>
    int(6)
    ["Sule"]=>
    int(9)
    ["Nokta"]=>
    int(14)
}
```

I use var_dump() function to print the map, so it prints the values and their type and value.

1.4.2. Get the value for a given key

```
print $mesneviNuriye["Mukaddime"] . "\n";
echo $mesneviNuriye["Katre"] . "\n";
echo $mesneviNuriye["Hubab"] . "\n";
echo $mesneviNuriye["Nokta"] . "\n";
```

Fig. 30: Getting the value of a given key in PHP

In the code sample above, the syntax for getting the value of a given key is visible. When developers want to get the value of a key, they should use `$map_name[key]` syntax. The output of the above sample is Mukaddime is 3, Katre 33, Hubab 34, Nokta 14 pages long.

1.4.3. Add a new element

```
$mesneviNuriye["Fihrist"] = 10;  
$mesneviNuriye["Itizar"] = 1;
```

Fig. 31: Add a new element in PHP

To add a new element to an associated array in PHP, developers must specify the key and value using `$map_name[key]` syntax as seen in the above example.

The resulting associated array looks like this when it is printed on the console:

```
array(16) {  
  ["Mukaddime"]=>  
    int(3)  
  ["Lem'alar"]=>  
    int(11)  
  ["Reshalar"]=>  
    int(12)  
  ["Lasiyyemalar"]=>  
    int(16)  
  ["Katre"]=>  
    int(33)  
  ["Hubab"]=>  
    int(34)  
  ["Habbe"]=>  
    int(32)  
  ["Zuhre"]=>  
    int(32)  
  ["Zerre"]=>  
    int(11)  
  ["Semme"]=>  
    int(13)  
  ["Onuncu Risale"]=>  
    int(25)  
  ["14. Resha"]=>  
    int(6)  
  ["Sule"]=>  
    int(9)  
  ["Nokta"]=>  
    int(14)  
  ["Fihrist"]=>
```

```

    int(10)
    ["Itizar"]=>
    int(1)
}

```

1.4.4. Remove an element

```

unset($mesneviNuriye["Fihrist"]);

```

Fig. 32: Remove an element in PHP

Above code sample shows how to remove an element from an associated array object in PHP. Developers should use the unset built-in function. General syntax is like unset(\$map_name[key]).

The resulting associated array looks like this when it is printed on the console:

```

array(15) {
    ["Mukaddime"]=>
    int(3)
    ["Lem'alar"]=>
    int(11)
    ["Reshalar"]=>
    int(12)
    ["Lasiyyemalar"]=>
    int(16)
    ["Katre"]=>
    int(33)
    ["Hubab"]=>
    int(34)
    ["Habbe"]=>
    int(32)
    ["Zuhre"]=>
    int(32)
    ["Zerre"]=>
    int(11)
    ["Semme"]=>
    int(13)
    ["Onuncu Risale"]=>
    int(25)
    ["14. Resha"]=>
    int(6)
    ["Sule"]=>
    int(9)
    ["Nokta"]=>
}

```

```

    int(14)
    ["Itizar"]=>
    int(1)
}

```

1.4.5. Modify the value of an existing element

```

$mesneviNuriye["Itizar"] = 3;

```

Fig 33: Modify the value of an existing element in PHP

The above code sample shows how to modify the value of an existing element in associated array objects in PHP. Users should specify the key, then specify the new value using brackets and assignment operators as seen above.

The resulting associated array looks like this when it is printed on the console:

```

array(15) {
    ["Mukaddime"]=>
    int(3)
    ["Lem'alar"]=>
    int(11)
    ["Reshalar"]=>
    int(12)
    ["Lasiyyemalar"]=>
    int(16)
    ["Katre"]=>
    int(33)
    ["Hubab"]=>
    int(34)
    ["Habbe"]=>
    int(32)
    ["Zuhre"]=>
    int(32)
    ["Zerre"]=>
    int(11)
    ["Semme"]=>
    int(13)
    ["Onuncu Risale"]=>
    int(25)
    ["14. Resha"]=>
    int(6)
    ["Sule"]=>
    int(9)
    ["Nokta"]=>
    int(14)
    ["Itizar"]=>

```



```
int(3)
}
```

1.4.6. Search for the existence of a key

```
var_dump(array_key_exists("Habbe", $mesneviNuriye));
var_dump(array_key_exists("Sabah", $mesneviNuriye));
```

Fig. 34: Searching for the existence of a key in PHP

As it can be seen in above example, people can search for existing of a key by using `array_key_exists(Key, $map_name)` syntax. The result is a boolean value. For the sample above, first returns true and second returns false.

1.4.7. Search for the existence of a value

```
var_dump(in_array(12, $mesneviNuriye));
var_dump(in_array(33, $mesneviNuriye));
var_dump(in_array(38, $mesneviNuriye));
```

Fig. 35: Searching for the existence of a value in PHP

As it can be seen in the above example, people can search for the existence of a value using `in_array(value, $map_name)` syntax. The result is a boolean value. For the sample above, first two returns true and the third returns false.

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

```
function foo($key, $value) {
    echo "mesneviNuriye[" . $key . "] = " . $value . "\n";
}

// Loop keys
foreach($mesneviNuriye as $key => $value){
    foo($key, $value);
}
```

Fig. 36: foo function and for loop printing pairs in PHP

As it can be seen in the above sample, foo function gets the key and values as parameter. At the end it prints the key and value pairs of MesneviiNuriye using built-in forEach loop and calling foo function for each entry.

The result will be like:

```
Function foo-----  
mesneviiNuriye[Mukaddime] = 3  
mesneviiNuriye[Lem'alar] = 11  
mesneviiNuriye[Reshalar] = 12  
mesneviiNuriye[Lasiyyemalar] = 16  
mesneviiNuriye[Katre] = 33  
mesneviiNuriye[Hubab] = 34  
mesneviiNuriye[Habbe] = 32  
mesneviiNuriye[Zuhre] = 32  
mesneviiNuriye[Zerre] = 11  
mesneviiNuriye[Semme] = 13  
mesneviiNuriye[Onuncu Risale] = 25  
mesneviiNuriye[14. Resha] = 6  
mesneviiNuriye[Sule] = 9  
mesneviiNuriye[Nokta] = 14  
mesneviiNuriye[Itizar] = 3
```

Fig. 37: Result of the foo function

1.5. Python

1.5.1. Initialize

```
mesneviiNuriye = {  
    "Mukaddime" : 3,  
    "Lem'alar" : 11,  
    "Reshalar" : 12,  
    "Lasiyyemalar" : 16,  
    "Katre" : 33,  
    "Hubab" : 34,  
    "Habbe" : 32,  
    "Zuhre" : 32,  
    "Zerre" : 11,  
    "Semme" : 13,  
    "Onuncu Risale" : 25,  
    "14. Resha" : 6,  
    "Sule" : 9,  
    "Nokta" : 14,  
}
```

Fig. 38: Initialization in Python

The associated array (dictionary) object mesneviiNuriye is initialized in the code samples above. The associated array keeps the number of pages of each chapter in Mesnevi-i Nuriye. Developers must create an object using braces and give key and values and semicolons between those keys and values. If we print the associated array we will see an output like below:

```
{'Mukaddime': 3, 'Lem'alar': 11, 'Reshalar': 12, 'Lasiyyemalar': 16, 'Katre': 33, 'Hubab': 34,  
'Habbe': 32, 'Zuhre': 32, 'Zerre': 11, 'Semme': 13, 'Onuncu Risale': 25, '14. Resha': 6, 'Sule': 9,  
'Nokta': 14}
```

1.5.2. Get the value for a given key

```
print(mesneviiNuriye["Mukaddime"])
print(mesneviiNuriye["Katre"])
print(mesneviiNuriye["Hubab"])
print(mesneviiNuriye["Nokta"])
```

Fig. 39: Getting the value of a given key in Python

In the code sample above, the syntax for getting the value of a given key is visible. Developers must specify the key and value using `map_name[key]` syntax as seen in the above example. The output of the above sample says that Mukaddime is 3, Katre 33, Hubab 34, Nokta 14 pages long.

1.5.3. Add a new element

```
mesneviiNuriye["Fihrist"] = 10
mesneviiNuriye["Itizar"] = 1
```

Fig. 40: Add a new element in Python

To add a new element to an associated array in Python, developers must specify the key and value using brackets and equal sign as in the sample above.

The resulting associated array looks like this when it is printed on the console:

```
{'Mukaddime': 3, 'Lem'alar': 11, 'Reshalar': 12, 'Lasiyyemalar': 16, 'Katre': 33, 'Hubab': 34, 'Habbe': 32, 'Zuhre': 32, 'Zerre': 11, 'Semme': 13, 'Onuncu Risale': 25, '14. Resha': 6, 'Sule': 9, 'Nokta': 14, 'Fihrist': 10, 'Itizar': 1}
```

1.5.4. Remove an element

```
mesneviiNuriye.pop("Fihrist")
# the same can be achieved by
# mesneviiNuriye.pop("Fihrist", 10)
```

Fig. 41: Remove an element in Python

Above code sample shows how to remove an element from an associated array object in Python. General syntax is like `map_name.pop(key)` or `map_name.pop(key, value)`.

The resulting associated array looks like this when it is printed on the console:

```
{'Mukaddime': 3, 'Lem'alar': 11, 'Reshalar': 12, 'Lasiyyemalar': 16, 'Katre': 33, 'Hubab': 34, 'Habbe': 32, 'Zuhre': 32, 'Zerre': 11, 'Semme': 13, 'Onuncu Risale': 25, '14. Resha': 6, 'Sule': 9, 'Nokta': 14, 'Itizar': 1}
```

1.5.5. Modify the value of an existing element

```
mesneviNuriye["Itizar"] = 3
```

Fig 42: Modify the value of an existing element in Python

The above code sample shows how to modify the value of an existing element in associated array objects in Python. Users should specify the key, then specify the new value as parameters using `dict_name[key] = value` syntax.

The resulting associated array looks like this when it is printed on the console:

```
{'Mukaddime': 3, 'Lem'alar': 11, 'Reshalar': 12, 'Lasiyyemalar': 16, 'Katre': 33, 'Hubab': 34, 'Habbe': 32, 'Zuhre': 32, 'Zerre': 11, 'Semme': 13, 'Onuncu Risale': 25, '14. Resha': 6, 'Sule': 9, 'Nokta': 14, 'Itizar': 3}
```

1.5.6. Search for the existence of a key

```
print("Habbe" in mesneviNuriye)
print("Sabah" in mesneviNuriye)
```

Fig. 43: Searching for the existence of a key in Python

As it can be seen in above example, people can search for existing of a key by using `(key in map_name)` syntax. The result is a boolean value. For the sample above, first returns true and second returns false.

1.5.7. Search for the existence of a value

```
print(12 in mesneviiNuriye.values())
print(33 in mesneviiNuriye.values())
print(38 in mesneviiNuriye.values())
```

Fig. 44: Searching for the existence of a value in Python

As it can be seen in the above example, people can search for the existence of a value using (value in map_name.values()) syntax. People actually are searching the values in the values array containing all the values of the dictionary object. The result is a boolean value. For the sample above, first two returns true and the third returns false.

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

```
def foo (key, value):
    print("mesneviiNuriye[",key,"] = ", value)

print("Function foo-----")

for k in mesneviiNuriye:
    foo(k, mesneviiNuriye[k])
```

Fig. 45: foo function and for loop printing pairs in Python

As it can be seen in the above sample, foo function prints the entries of the map object and gets the key, value. At the end it prints the key and value pairs of MesneviiNuriye using built-in for loop and calling foo function for each key-value pair.

If we provoke the function by iterating over mesneviiNuriye, the result will be like:

```
mesneviiNuriye[ Mukaddime ] = 3
mesneviiNuriye[ Lem'alar ] = 11
mesneviiNuriye[ Reshalar ] = 12
mesneviiNuriye[ Lasiyyemalar ] = 16
mesneviiNuriye[ Katre ] = 33
mesneviiNuriye[ Hubab ] = 34
mesneviiNuriye[ Habbe ] = 32
mesneviiNuriye[ Zuhre ] = 32
mesneviiNuriye[ Zerre ] = 11
mesneviiNuriye[ Semme ] = 13
mesneviiNuriye[ Onuncu Risale ] = 25
mesneviiNuriye[ 14. Resha ] = 6
mesneviiNuriye[ Sule ] = 9
mesneviiNuriye[ Nokta ] = 14
mesneviiNuriye[ Itizar ] = 3
```

Fig. 46: Result of the foo function

1.6. Ruby

1.6.1. Initialize

```
mesneviNuriye = {  
  "Mukaddime" => 3,  
  "Lem'alar" => 11,  
  "Reshalar" => 12,  
  "Lasiyyemalar" => 16,  
  "Katre" => 33,  
  "Hubab" => 34,  
  "Habbe" => 32,  
  "Zuhre" => 32,  
  "Zerre" => 11,  
  "Semme" => 13,  
  "Onuncu Risale" => 25,  
  "14. Resha" => 6,  
  "Sule" => 9,  
  "Nokta" => 14,  
}
```

Fig. 47: Initialization in Ruby

The associated array (Map) object `mesneviNuriye` is initialized in the code samples above. The associated array keeps the number of pages of each chapter in Mesnevi-i Nuriye. If we print the associated array we will see an output like below:

```
{"Mukaddime"=>3, "Lem'alar"=>11, "Reshalar"=>12, "Lasiyyemalar"=>16, "Katre"=>33,  
"Hubab"=>34, "Habbe"=>32, "Zuhre"=>32, "Zerre"=>11, "Semme"=>13, "Onuncu Risale"=>25,  
"14. Resha"=>6, "Sule"=>9, "Nokta"=>14}
```


1.6.2. Get the value for a given key

```
puts(mesneviiNuriye["Mukaddime"])
puts(mesneviiNuriye["Katre"])
puts(mesneviiNuriye["Hubab"])
puts(mesneviiNuriye["Nokta"])
```

```
Kta"=>14}
Get the value of a k
3
33
34
14
Add a new item-----
```

Fig. 48: Getting the value of a given key in Ruby

In the code sample above, the syntax for getting the value of a given key is visible. When developers want to get the value of a key, they should use `map_name[key]` syntax. We see from above sample that Mukaddime is 3, Katre 33, Hubab 34, Nokta 14 pages long.

1.6.3. Add a new element

```
mesneviiNuriye["Fihrist"] = 10
mesneviiNuriye["Itizar"] = 1
```

Fig. 49: Add a new element in Ruby

To add a new element to an associated array in Ruby, developers must specify the key and value using `map_name[new_key] = new_value` syntax.

The resulting associated array looks like this when it is printed on the console:

```
{"Mukaddime"=>3, "Lem'alar"=>11, "Reshalar"=>12, "Lasiyyemalar"=>16, "Katre"=>33,
"Hubab"=>34, "Habbe"=>32, "Zuhre"=>32, "Zerre"=>11, "Semme"=>13, "Onuncu Risale"=>25,
"14. Resha"=>6, "Sule"=>9, "Nokta"=>14, "Fihrist"=>10, "Itizar"=>1}
```

1.6.4. Remove an element

```
mesneviiNuriye.delete("Fihrist")
```

Fig. 50: Remove an element in Ruby

Above code sample shows how to remove an element from an associated array object in Ruby. General syntax is like `map_name.delete(key)`.

The resulting associated array looks like this when it is printed on the console:

```
{"Mukaddime"=>3, "Lem'alar"=>11, "Reshalar"=>12, "Lasiyyemalar"=>16, "Katre"=>33, "Hubab"=>34, "Habbe"=>32, "Zuhre"=>32, "Zerre"=>11, "Semme"=>13, "Onuncu Risale"=>25, "14. Resha"=>6, "Sule"=>9, "Nokta"=>14, "Itizar"=>1}
```

1.6.5. Modify the value of an existing element

```
mesneviNuriye["Itizar"] = 3
```

Fig 51: Modify the value of an existing element in Ruby

The above code sample shows how to modify the value of an existing element in associated array objects in Ruby. Users should specify the key, then specify the new value as parameters similar to the adding new elements.

The resulting associated array looks like this when it is printed on the console:

```
{"Mukaddime"=>3, "Lem'alar"=>11, "Reshalar"=>12, "Lasiyyemalar"=>16, "Katre"=>33, "Hubab"=>34, "Habbe"=>32, "Zuhre"=>32, "Zerre"=>11, "Semme"=>13, "Onuncu Risale"=>25, "14. Resha"=>6, "Sule"=>9, "Nokta"=>14, "Itizar"=>3}
```

1.6.6. Search for the existence of a key

```
puts(mesneviNuriye.include?("Habbe"))  
puts(mesneviNuriye.include?("Sabah"))
```

Fig. 52: Searching for the existence of a key in Ruby

As it can be seen in above example, people can search for existing of a key by using `map_name.include?(Key)` syntax. The result is a boolean value. For the sample above, first returns true and the second returns false.

1.6.7. Search for the existence of a value

```
puts(mesneviNuriye.value?(12))  
puts(mesneviNuriye.value?(33))  
puts(mesneviNuriye.value?(38))
```

Fig. 53: Searching for the existence of a value in Ruby

As it can be seen in the above example, people can search for existing of a value by using `map_name.value?(Value)` syntax. The result is a boolean value. For the sample above, first two returns true and the third returns false.

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

```
def foo (key, value)
  puts("mesneviiNuriye[ #{key} ] = #{value} ")
end

mesneviiNuriye.each{|k, v| foo(k, v)}
```

Fig. 54: foo function and for loop printing pairs in Ruby

As it can be seen in the above sample, `foo` function prints the entries of the map object and gets the key, value. At the end it prints the key and value pairs of `MesneviiNuriye` using built-in `each` loop.

If we provoke the function, the result will be like:

```
mesneviiNuriye[ Mukaddime ] = 3
mesneviiNuriye[ Lem'alar ] = 11
mesneviiNuriye[ Reshalar ] = 12
mesneviiNuriye[ Lasiyyemalar ] = 16
mesneviiNuriye[ Katre ] = 33
mesneviiNuriye[ Hubab ] = 34
mesneviiNuriye[ Habbe ] = 32
mesneviiNuriye[ Zuhre ] = 32
mesneviiNuriye[ Zerre ] = 11
mesneviiNuriye[ Semme ] = 13
mesneviiNuriye[ Onuncu Risale ] = 25
mesneviiNuriye[ 14. Resha ] = 6
mesneviiNuriye[ Sule ] = 9
mesneviiNuriye[ Nokta ] = 14
mesneviiNuriye[ Itizar ] = 3
```

Fig. 55: Result of the `foo` function

1.7. Rust

1.7.1. Initialize

```
let mut mesnevii_nuriye = HashMap::new();

mesnevii_nuriye.insert(String::from("Mukaddime"), 3 );
mesnevii_nuriye.insert(String::from("Lem'alar"), 11 );
mesnevii_nuriye.insert(String::from("Reshalar"), 12 );
mesnevii_nuriye.insert(String::from("Lasiyyemalar"), 16 );
mesnevii_nuriye.insert(String::from("Katre"), 33 );
mesnevii_nuriye.insert(String::from("Hubab"), 34 );
mesnevii_nuriye.insert(String::from("Habbe"), 32 );
mesnevii_nuriye.insert(String::from("Zuhre"), 32 );
mesnevii_nuriye.insert(String::from("Zerre"), 11 );
mesnevii_nuriye.insert(String::from("Semme"), 13 );
mesnevii_nuriye.insert(String::from("Onuncu Risale"), 25 );
mesnevii_nuriye.insert(String::from("14. Resha"), 6 );
mesnevii_nuriye.insert(String::from("Sule"), 9 );
mesnevii_nuriye.insert(String::from("Nokta"), 14 );
```

Fig. 56: Initialization in Rust

The associated array (Map) object `mesnevii_nuriye` is initialized in the code samples above. Developers should use the `insert` built-in function to initialize the map object in Rust language. The associated array keeps the number of pages of each chapter in Mesnevi-i Nuriye. If we print the associated array, we will see an output like the below:

```
{"Zerre": 11, "Hubab": 34, "Zuhre": 32, "Onuncu Risale": 25, "Lasiyyemalar": 16, "Nokta": 14,
"Reshalar": 12, "Semme": 13, "Lem'alar": 11, "Katre": 33, "Habbe": 32, "Sule": 9, "14. Resha": 6,
"Mukaddime": 3}
```

Please be aware that the output is not in the order we initialized the map object.

1.7.2. Get the value for a given key

```
println!("{}", mesnevii_nuriye["Mukaddime"]);
println!("{}", mesnevii_nuriye["Katre"]);
println!("{}", mesnevii_nuriye["Hubab"]);
println!("{}", mesnevii_nuriye["Nokta"]);
```

Get the
3
33
34
14

Fig. 57: Getting the value of a given key in Rust

In the code sample above, the syntax for getting the value of a given key is visible. When developers want to get the value of a key, they should use `map_name[key]` syntax. We see from the above sample that Mukaddime is 3, Katre 33, Hubab 34, and Nokta 14 pages long.

1.7.3. Add a new element

```
mesnevii_nuriye.insert(String::from("Fihrist") , 10);
mesnevii_nuriye.insert(String::from("Itizar") , 1);
```

Fig. 58: Add a new element in Rust

To add a new element to an associated array in Rust, developers must specify the key and value using the `insert` built-in function as seen in the above example.

The resulting associated array looks like this when it is printed on the console:

```
{"Zerre": 11, "Hubab": 34, "Zuhre": 32, "Onuncu Risale": 25, "Lasiyyemalar": 16, "Nokta": 14,
"Fihrist": 10, "Reshalar": 12, "Semme": 13, "Lem'alar": 11, "Katre": 33, "Habbe": 32, "Sule": 9,
"14. Resha": 6, "Mukaddime": 3, "Itizar": 1}
```

1.7.4. Remove an element

```
mesnevii_nuriye.remove("Fihrist");
```

Fig. 59: Remove an element in Rust

Above code sample shows how to remove an element from an associated array object in Rust. The general syntax is like `map_name.remove(key)`.

The resulting associated array looks like this when it is printed on the console:

```
{"Zerre": 11, "Hubab": 34, "Zuhre": 32, "Onuncu Risale": 25, "Lasiyyemalar": 16, "Nokta": 14, "Reshalar": 12, "Semme": 13, "Lem'alar": 11, "Katre": 33, "Habbe": 32, "Sule": 9, "14. Resha": 6, "Mukaddime": 3, "Itizar": 1}
```

1.7.5. Modify the value of an existing element

```
mesnevii_nuriye.insert(String::from("Itizar") , 3);
```

Fig 60: Modify the value of an existing element in Rust

The above code sample shows how to modify the value of an existing element in associated array objects in Rust. Users should specify the key, then specify the new value as parameters to the insert built-in function.

The resulting associated array looks like this when it is printed on the console:

```
{"Zerre": 11, "Hubab": 34, "Zuhre": 32, "Onuncu Risale": 25, "Lasiyyemalar": 16, "Nokta": 14, "Reshalar": 12, "Semme": 13, "Lem'alar": 11, "Katre": 33, "Habbe": 32, "Sule": 9, "14. Resha": 6, "Mukaddime": 3, "Itizar": 3}
```

1.7.6. Search for the existence of a key

```
println!("{}", mesnevii_nuriye.contains_key("Habbe"));
println!("{}", mesnevii_nuriye.contains_key("Sabah"));
```

Fig. 61: Searching for the existence of a key in Rust

As it can be seen in the above example, people can search for existing of a key by using `map_name.contains_key(Key)` syntax. The result is a boolean value. For the sample above, the first returns true and the second returns false.

1.7.7. Search for the existence of a value

```
{
    let v = 12;
    let does_contain = mesnevii_nuriye.values().any(|&val| val == v);
    println!("{:?}", does_contain);
}

{
    let v = 33;
    let does_contain = mesnevii_nuriye.values().any(|&val| val == v);
    println!("{:?}", does_contain);
}

{
    let v = 38;
    let does_contain = mesnevii_nuriye.values().any(|&val| val == v);
    println!("{:?}", does_contain);
}
```

Fig. 62: Searching for the existence of a value in Rust

As seen in the above example, people first should create a variable `v` to store the desired value. Then, they can search for the existing of value by using `map_name.values().any(|&val| val == v)` syntax. The result is a boolean value. For the sample above, first two returns true and the third returns false.

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

```
fn foo(key: &String, value: &i32) {
    println!("mesneviiNuriye[{}] = {}", key, value);
}

for (key, value) in &mesnevii_nuriye {
    foo(key, value);
}
```

Fig. 63: foo function and for loop printing pairs in Rust

As it can be seen in the above sample, foo function prints the entries of the map object and gets the key, value. In the end, it prints the key and value pairs of Mesnevii_nuriye using for loop and calling foo function with the key, value pairs of each entry.

If we provoke the code above, the result will be like:

```
mesneviiNuriye[Itizar] = 3
mesneviiNuriye[Lasiyyemalar] = 16
mesneviiNuriye[Zerre] = 11
mesneviiNuriye[Semme] = 13
mesneviiNuriye[Sule] = 9
mesneviiNuriye[Mukaddime] = 3
mesneviiNuriye[Lem'alar] = 11
mesneviiNuriye[14. Resha] = 6
mesneviiNuriye[Hubab] = 34
mesneviiNuriye[Onuncu Risale] = 25
mesneviiNuriye[Katre] = 33
mesneviiNuriye[Reshalar] = 12
mesneviiNuriye[Nokta] = 14
mesneviiNuriye[Zuhre] = 32
mesneviiNuriye[Habbe] = 32
```

Fig. 64: Result of the foo function

2. Evaluation of the Languages

The evaluation of the languages in this homework will be done according to their readability, and writeability because others are implementation choices and I believe those design choices shouldn't be evaluated.

Based on their readability and writeability I can say that Python and Ruby are the best languages in these seven languages. They both have easy initialization, and easy built-in functions and attributes of the associated array objects are easily reachable.

Again based on the same criteria, Rust is the worst language. Its functions are not intuitive. Even there is a shortage of the functions. A developer should call functions of functions or so in some cases.

The other languages are similar in terms of readability and writeability. They mostly have intuitive approaches like brackets, or useful built-in approaches, like set, get in Javascript.

Finding a proper tutorial, documentation, or a community to ask questions is another factor to assess a programming language. From my experience with associative arrays in these seven

languages, the order is the same in terms of resources for the language is somewhat similar to the above order. Python is the best, and Rust is the worst in my ordering. Python's syntax is intuitive, and in case I stuck, I could use the built-in help function to get an insight about the dictionary objects.

However, writing Rust was suffering. I haven't any experience; hence, I need tutorials and documentation a lot. However, there are not enough materials to get the required knowledge about the language. When I was stuck and couldn't find a solution on the web, I tried some possible solutions and some make-up solutions based on the error messages and my intuition. That's why I put Rust as the worst on my list.

The others are somewhat similar to each other, and finding a solution on the web was straightforward. They most have sufficient documentation, either official or not.

So, if I have some work with associated arrays, I will prefer Python for dealing with that job, given that performance and space requirements aren't considered since they require a lot of work to assess those languages.

3. Learning Strategy

From my internship experience, I learned that using official documentation is the best way to learn a language. So, in my learning process, I tried to make use of the official documentation. In cases where I couldn't find documentation or thought the documentation insufficient, I took the advantage of some well-known programming web site and forums such as StackOverflow, GeeksForGeeks, etc.

I used online compilers except for Python since I am already using it on my local machine. There are many online compilers, but after all the ones that I used seem fair enough for this assignment.

Throughout the process, I stored the useful links as well as the knowledge. The list of the links for each language, its documentation, online compiler, and additional links I used when I stuck, is given in below.

Dart

Documentation: <https://dart.dev/guides/language/language-tour#maps>

Online Compiler: <https://dartpad.dev/?id=57b5bb0b8e8eea41d662f2aad36893d7>

Additional: <https://www.educative.io/answers/what-is-the-mapremove-method-in-dart>
<https://stackoverflow.com/questions/57037101/map-update-method-with-ifabsent-in-dart>
<https://www.folkstalk.com/2022/09/get-value-from-map-with-key-flutter-with-code-examples.html>

Javascript

Documentation:

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

Online Compiler: <https://www.programiz.com/javascript/online-compiler/>

Additional: <https://bobbyhadz.com/blog/javascript-update-value-in-map>
<https://www.geeksforgeeks.org/javascript-map-has-method/>

Lua

Documentation:

<https://www.lua.org/pil/2.5.html#:~:text=An%20associative%20array%20is%20an,want%20to%20a%20table%20dynamically>.

Online Compiler: https://www.tutorialspoint.com/execute_lua_online.php

Additional:

<https://stackoverflow.com/questions/1758991/how-to-remove-a-lua-table-entry-by-its-key>

<https://programming-idioms.org/idiom/51/check-if-map-contains-key/1681/lua>

<https://stackoverflow.com/questions/2282444/how-to-check-if-a-table-contains-an-element-in-lua>

<https://www.codegrepper.com/code-examples/lua/lua+how+to+print+a+table>

PHP

Documentation: <https://www.php.net/manual/en/language.types.array.php>

Online Compiler: <https://paiza.io/projects/VwgjBAq-hmMsuv3N6w08ow>

Additional: <https://programming-idioms.org/idiom/52/check-if-map-contains-value>

<https://stackoverflow.com/questions/55560722/how-to-print-array-element-with-keys-values-in-php>

https://www.w3schools.com/php/php_functions.asp

Python

Documentation: I used the built-in help function to get information when I stuck.

Online Compiler: I used my local interpreter to run python code

Additional:

<https://stackoverflow.com/questions/1602934/check-if-a-given-key-already-exists-in-a-dictionary>

<https://stackoverflow.com/questions/26660654/how-do-i-print-the-key-value-pairs-of-a-dictionary-in-python>

Ruby

Documentation: I can't find official documentation for Ruby.

Online Compiler: <https://replit.com/languages/ruby>

Additional: <https://www.geeksforgeeks.org/ruby-assoc-function/>

<https://www.rubyguides.com/2018/10/ruby-map-method/>

<https://apidock.com/ruby/Hash/delete>

<https://programming-idioms.org/idiom/51/check-if-map-contains-key/851/ruby>

<https://programming-idioms.org/idiom/52/check-if-map-contains-value/850/ruby>

https://www.tutorialspoint.com/ruby/ruby_methods.htm

Rust

Documentation: <https://docs.rs/dict/latest/dict/>

Online Compiler: <https://replit.com/languages/rust>

Additional: <https://www.geeksforgeeks.org/rust-hashmaps/>

<https://programming-idioms.org/idiom/52/check-if-map-contains-value/455/rust>
<https://play.rust-lang.org/?version=stable&mode=debug&edition=2015&gist=e51781a1c9400c7e17423a3d06c64b27>