Bilkent University
Fall 2022-2023

CS315
Programming Languages

Homework 1
Associative Arrays

Section 2

Deniz Tuna Onguner
22001788

November 18, Friday

CONTENT

1.0 Python

In python, associative arrays are declared as dictionaries. A dictionary of python can take any primitive type as a key to the value of any type.

A dictionary is to define in the fallowing way: An identifier, assign operator, left curly bracket, items, right curly bracket. Items are simply key-value pairs that separated with a colon.

```
Age = {
      "Galadriel": 8372,
      "Elrond": 6518,
      "Arwen": 2901
}
Rings = {}
```

In the example above, a dictionary named Age is defined. Dictionary Age has 3 elements in this case, which can be found by len() function of default library. And also another dictionary is defined with name of Rings, which is empty.

```
item_count_Age = len(Age)
```

item_count_Age will be equal to the item count of Age dictionary, which is 3.

Like lists, an item of a dictionary is accessible by subscript operator but with its key instead of an index.

```
Age_Galadriel = Age["Galadriel"]
```

Variable Age_Galadriel will be equal to the value whose key is “Galadriel”.

```
Age_Elrond = Age['Elrond']
```

Single-quotes can also be used for string type keys.

```
Age_Arwen = Age.get(“Arwen")
```

Or, get() function can be used to get the desired value with the syntax above. The function takes a key as the parameter and return its value.

Subscript operator can also be used for adding new items to the dictionary. In the code segment above, new items are added to both dictionaries defined above. Since the passed keys are not available in the dictionary, they are automatically added.

```
Age['Legolas'] = 2931
Rings['Elves'] = 3
Rings['Sauron'] = 0
```

Similarly, subscript operator can also be used to modify values.

```
Rings['Sauron'] += 1
```

In python, different type of keys and values are allowed in a single dictionary.

```
Rings[1] = "The one ring"

Age['Gandalf'] = 'Elder than time'
```

In the code above, a string value with an integer key is added to Ring dictionary, which has string keys and integer values formerly. And another string is added to Age dictionary.

keys() function of dictionary class can be used to return a list which has only keys on a dictionary.

```
def find_in_keys(key, dictionary):
    for k in dictionary.keys():
        if key == k:
            return True
    return False
```

The function above takes two variables, a key and a dictionary, then search the keys of the dictionary by keys() function and returns the result according to the existence of the key.

```
Key = "Gandalf"
print("Key exits\n") if find_in_keys(Key, Age) \
else print("Key does not exist\n")
```

The code segment above will print "Key exits" since key "Gandalf" exist in the dictionary of Age.

```
def find_in_values(val, dictionary):
    for v in dictionary.values():
        if val == v:
            return True
    return False



print("Search for a value")
Val = 20
print("Value exits\n") if find_in_values(Val, Age) \
else print("Values does not exist\n")
```

Similarly, values() function will return only the values of a dictionary, so that the code segment above will print "Values does not exist" since such a value of 20 is not in the Age dictionary.

In python dictionaries, an item can also be removed with again subscript operator or a default function named pop().

```
del Age['Gandalf']

Age.pop("Arwen")
```

Keys of "Gandalf" and "Arwen" are going to be removed from the Age dictionary after the codes above executed.

copy() function can be used to make a copy of a dictionary

```
Age_Copy = Age.copy()
```

It is possible to remove all items of dictionary at once by clear() function

```
Rings.clear()

Age = {}
```

Setting a dictionary to an empty one would also make it clear.

foo(dict) function: Display all key-value pairs of a dictionary

```
def foo(dictionary):
    for key in dictionary.keys():
        print(f"{key}: {dictionary[key]}")



foo(Age)
```

The code above is going to print all key-value pairs one by one. But simply `print(f"{Age}")` can be used for the exact same behavior.

After the foo function executed, the output below is expected to appear.

```
Galadriel: 8372

Elrond: 6518

Legolas: 2931
```

Note: In python, nested dictionaries are allowed, which are simply dictionaries as values paired with keys in another dictionary. All the operators and functions above will also work for nested ones as well. (See the example program for details)

1.1 JavaScript

Associative arrays in JavaScript are similar to python dictionaries in numerous ways. They can be initialized in the following way.

```
let grades = {
      "Hasan": 95,
      "Deniz": 50,
      "Alper": 10,
      "Kerem": 40
}
let letterGrades = {}
```

An associative arrays named grades initialized in the code above. Just like python, the left side of colons are the keys and the right side are the values. In the given example keys are strings and values are integers, however they can be different types either.

Again, like in python, subscript operator can be used for adding, modifying, and removing elements from the array.

```
letterGrades[99] = "A+"
letterGrades[90] = "A"
letterGrades[80] = "A-"
grades["Emma"] = 65
grades["Jack"] = 75
grades["Kerem"] = 90
delete grades.Jack
delete letterGrades[99]
console.log("\nEmma's grade is updated to", grades.Emma)
```

Above, subscript operator is used to add, delete, and modify elements of an associative array. However, subscript is not the only way to access elements. `AssociativeArray.Key` is also a valid syntax. For instance, Jack's grade is removed by that way above.

It is possible to search for either keys or values in an associative array in JavaScript.

```
function searchKeys(key, aa) {
      for (let k in aa)
            if (k === key)
                  return true
      return false
}
```

```
let key = "Emma"
if (searchKeys(key, grades))
      console.log("Key exists")
else
      console.log("Key does not exist")
```

The code segment above, is going to search for a desired key and if the key exist will print “Key exist” on the console, if not then print “”Key does not exist”. Since such a key “Emma” is available in the array, it expected to see “Key exist” on the console.

Similarly, values can be searched as well with such a function below.

```
function searchValues(val, aa) {
      for (let v in aa)
            if (aa[v] === val)
                  return true
      return false
}
```

In order to print all key-value pairs foo function below can de applied.

```
function foo(aa) {
      for (let key in aa)
            console.log(key, "=", aa[key])
}
```

The function is going to pass all keys one by one for the given array, and then by these keys, it will get the values corresponding to these keys one by one again, and prints them. But again, like python, classical print function can be applied for the exact same behavior rather than writing an additional function.

```
console.log("Grades =", grades)
```

For instance, the code above, without requiring any other function can simply print all key-value pairs of grades array. The expected output is given below.

```
Grades = { Hasan: 95, Deniz: 50, Alper: 10, Kerem: 40 }
```

In JavaScript, it is possible to clear an associative array by deleting each pair one by one in a loop like below.

```
for (let item in letterGrades)
      delete letterGrades[item]
```

Note: JavaScript also supports nested associative arrays like python. (See the example program for details)

1.2 Dart

In Dart, associative arrays are names as Map. However, they are to be initialized in the almost exact same way with python and JavaScript.

```
Map capitalsLatitudes = {
  "Paris": 48.864716,
  "Cairo": 30.033333,
  "Gitega": -3.42640,
  "Ankara": 39.925533,
  "Cape Town": -33.55
};
Map countOfMoons = {};
```

A map names capitalsLatitudes is declared above, again like former languages in this document, right side of the colons are keys and the left sides are values. Dart also accepts different types as keys and values as well. And another map countOfMoons is also declared, which is currently empty.

The element count of a map is simply accessible by length property of Map class.

```
var size = capitalsLatitudes.length;
print("Element count of the map = $size");
```

The code above is going to print the element count of the map, which is currently 5.

Again, just like python and JavaScript, subscript operator can used in many ways, including adding new element and modifying them.

```
countOfMoons["Earth"] = 1;
countOfMoons["Jupiter"] = 80;
countOfMoons["Jupiter"] -= 2;
print("Latitude of Ankara is ${capitalsLatitudes[“Ankara"]}");
print("Latitude of Cairo is ${capitalsLatitudes[“Cairo”]}\n");
```

However, to remove an element from the array remove method of the Map class should be applied.

```
  capitalsLatitudes.remove("Cape Town”);
```

Again, it is possible in dart to search for a specific key or a specific value in a map. In given code blocks below two functions are given to make these searches.

```
bool searchInKeys(Map map, var key) {
	for (var k in map.keys) {
		if (k == key) {
			return true;
		}
	}
	return false;
}
bool searchInValues(Map map, var val) {
	for (var v in map.values) {
		if (v == val) {
			return true;
		}
	}
	return false;
}
var key = "Paris";
if (searchInKeys(capitalsLatitudes, key)) {
	print("Key exists");
} else {
	print("Key does not exit");
}
var val = 10;
if (searchInValues(countOfMoons, val)) {
	print("Value exists");
} else {
	print("Value does not exit");
}
```

For the given code above, the expected output is “Key exist” since “Paris” is in the map; and “Value does not exist” since there is no such value 10. keys and values properties of the Map class can be used to access them separately.

An element can be removed with remove function of the Map class.

```
capitalsLatitudes.remove("Cape Town");
```

It is also possible to remove all elements at once. clear function of the Map class can be used or the variable holds the map can simply be seated to a new empty map.

```
  capitalsLatitudes.clear();

  countOfMoons = {};
```

Foo function can be used to print all key-value pairs one by one. But just like python and java, default print function can also be used for that purpose.

```
void foo(Map map) {

      map.forEach((key, val) => print("$key: $val"));

}

print("Map: Latitudes of capitals $capitalsLatitudes");

foo(capitalsLatitudes);
```

The expected output for foo function is given below, which is exactly same with the what print function does.

```
Paris: 48.864716

Cairo: 30.033333

Gitega: -3.4264

Ankara: 39.925533
```

Note: Dart allows nested maps. (See the example program for further details)

1.3 PHP

In PHP, associative arrays are slightly more different than former languages. They can be declared in the following way.

```
$populationInMillion = array(
    "Turkey" => 85,
    "Germany" => 83,
    "France" => 67.5,
    "Italy" => 59
);
```

Different from former languages, there should be =〉 between keys and values, which actually makes senses because that symbol looks like an arrow. But as the same before, keys and values from different types are allowed.

Again, subscript operator can used to access elements or add new ones.

```
$populationInMillion["Turkey"] += 5;
$populationInMillion["Belgium"] = 12.0;
$populationInMillion["Belgium"] = 11.5;
$popItaly = $populationInMillion[“Italy"];
$popGermany = $populationInMillion['Germany'];
```

unset function from the default library can be used to remove elements from an array.

```
unset ($populationInMillion["France"]);
```

arrays_keys method can bu used to access all keys. In the code below, it is used to search all keys.

```
function searchInKeys ($array, $key) {
    $keys = array_keys($array);
    for($i=0; $i < count($array); ++$i) {
        if ($key == $keys[$i]) {
            return true;
        }
    }
    return false;
}
```

There is no values method in PHP unlike former languages, but it is actually not necessary because keys can be used to access values. The function below, uses again array_keys method to search values of a given associative array.

```
function searchInValues ($array, $val) {
    $keys = array_keys($array);
    for($i=0; $i < count($array); ++$i) {
        if ($val == $array[$keys[$i]]) {
            return true;
        }
    }
    return false;
}
```

foo function is to print all key-value pairs one by one, just like in former languages.

```
function foo($array) {
    foreach ($array as $country => $population) {
        echo $country . " => " . $population . "\n";
    }
}
foo ($populationInMillion);
```

However, there two other ways to print all pairs of an associative array. Unlike former languages, the default way to print on console, echo, cannot be used for that purpose. But there are two other functions print_r and var_dump, that can print all pairs on console.

An associative array can be cleared in the following way.

```
unset ($populationInMillion);
$populationInMillion = array();
```

The pointer should be set to a new empty array before reused because unset deletes entire array, not just elements in it.

Note: PHP also allows nested associative arrays. (See the example code for further details)

1.4 Ruby

In Ruby, associative arrays are called hashes. They can be initialized in the following way.

```
discoveryYearsOfBosons = Hash[
      "Higgs" => 2012,
      "Gluon" => 199,
      "Photon" => 1926
]
```

The element count can be get by length property.

```
size = discoveryYearsOfBosons.length
```

Subscript operator, again, can be used to access elements of a hash, or simply can be used to add new elements.

```
puts "\nThe higgs boson was discovered in #{discoveryYearsOfBosons
["Higgs"]}"
# Modify an element
discoveryYearsOfBosons["Gluon"] = 1976
# Add a new element
discoveryYearsOfBosons["W"] = 1983
discoveryYearsOfBosons["Electron"] = 1897
```

values and keys properties can be used for getting only keys or only values. They can be used for search in a hash.

```
def searchInKeys(hash, key)
      for k in hash.keys;
            return true if k == key
      end
      return false
end
def searchInValues(hash, val)
      for v in hash.values;
            return true if v == val
      end
      return false
end
```

foo function can be used for printing all key-value pairs, which works in the following way.

```
# foo function
def foo(hash)
      hash.each { |key, val| puts "#{key} => #{val}\n" }
end


puts "\nfoo:\n"
foo(discoveryYearsOfBosons)
```

clear property can simply remove all elements from a hash

```
discoveryYearsOfBosons.clear
```

delete function can be used of removing a pair with a specific key.

```
discoveryYearsOfBosons.delete(“Electron")
```

Note: Like former languages on the document, Ruby also supports nested associative arrays – hash in that case. (Please see the example program for further details)

1.5 Lua

In Lua, associative arrays are called Tables. They can be initialized in the following way.

```
BirthYearsOfPLs = {}
BirthYearsOfPLs["Java"] = 1895
BirthYearsOfPLs["Lua"] = 1993
BirthYearsOfPLs["C++"] = 1983
BirthYearsOfPLs["Dart"] = 2011
```

Subscript operator can be used to add new elements, modify them, or to get them.

```
-- Modify an element
BirthYearsOfPLs["Java"] = 1995
-- Remove an element
-- Setting the key that is desired to,
-- remove to nil is the way to remove it
BirthYearsOfPLs["Lua"] = nil
```

pairs function can be used for accessing all key-value pairs respectively. And just like in former languages that function can used for searching of keys and values.

```
-- Search in keys
function SearchInKeys(table, key)
	for k, _ in pairs(table) do
		if k == key then
			return true
		end
	end
	return false
end
```

```
-- Search in values
function SearchInVals(table, val)
	for _, v in pairs(table) do
		if v == val then
			return true
		end
	end
	return false
end
```

For once more foo function prints all key-value pairs.

```
function Foo(table)
	print ("Table: {")
	for key, val in pairs(table) do
		print (key," => ",val)
	end
	print ("}")
end


Foo(BirthYearsOfPLs)
```

And finally, a table can be cleared in the following way.

```
-- Clear table
for key, _ in pairs(BirthYearsOfPLs) do
	BirthYearsOfPLs[key] = nil
end
```

1.6 Rust

Compared to all other language covered in this paper, associative arrays are really different in Rust. First of all, they are names as HashMaps in Rust, and can be initialized in the following way.

```
let mut movies_imdb = HashMap::new();
movies_imdb.insert("Avatar", 7.8);
movies_imdb.insert("Inception", 8.8);
movies_imdb.insert("Interstellar", 6.6);
movies_imdb.insert("Titanic", 9.4);
movies_imdb.insert("Friends", 8.9);
```

insert method can be used to add new elements to the HashMap.

```
let mut size = movies_imdb.len();
```

len function can be used to get the element count of a HashMap

```
let imdb_of_titanic = movies_imdb[“Titanic"];
```

To modify an element, unwrap method is required because.

```
*movies_imdb.get_mut("Interstellar").unwrap() = 8.6;
```

remove method can be applied to remove an element from a HashMap

```
movies_imdb.remove(“Friends");
```

For loops can be used to get either keys or values

```
fn search_in_keys(map: HashMap<&str, f32>, key: String) -> bool {
      for (k, _) in &map {
            if k == &key {
                  return true
            }
      } return false
}
fn search_in_vals(map: HashMap<&str, f32>, val: f32) -> bool {
      for (_, v) in &map {
            if v == &val {
                  return true
            }
      } return false
}
```

Once again, foo function can print all key-value pairs

```
// foo
fn foo(map: HashMap<&str, f32>) {
      for (k, v) in &map {
            println!("{k}: {v}");
      }
}
foo(movies_imdb.clone());
```

But, HashMap should pass to the function by clone function which returns as an exact copy of the HashMap.

Note: Different from all former ones, Rust's HashMaps are not allow different type keys or values. In the examples above and the example program, HashMap of movies_imdb accepts only strings as keys, and only 32-bit floats as values. The syntax HashMap<&str, f32> corresponds to this behavior.

2.0 Comparison of programming languages in terms of associative arrays

In between all these programming languages, Python is the best choice to use associate arrays because of many reasons. Firstly, dictionaries of python provides several embedded functions for modification. However, these functions are not actually required, only subscript operator can be used to do all modifications. Which makes Python the most writable language between all. Moreover, Python allows nested dictionaries for which all default functions and again subscript operator behaves in the exact same way of single dictionaries. That makes Python easy to read as well because not additional functions or behaviors are possible. So that, for associative arrays, Python is clearly the best choice respectively.

3.0 Individual learning strategy

Tools that are used for each programming language:

Python: PyCharm IDE

JavaScript: WebStrom IDE

Dart: DartPad Online Compiler: https://dartpad.dev/?

PHP: Replit Online Compiler: https://replit.com/~

Ruby: Replit Online Compiler: https://replit.com/~

Lua: Replit Online Compiler: https://replit.com/~

Rust: Replit Online Compiler: https://replit.com/~

Online sources applied:

Stackoverflow: https://stackoverflow.com/

Geeksforgeeks: https://www.geeksforgeeks.org/

Tutorialspoint: https://www.tutorialspoint.com/index.htm

I simply applied to Google and online sources in order to accomplish this particular homework. I mostly got benefit the websites and tools above. Further details about the websites can be found on the very beginning of the example programs.