CS315 Homework 1

Cem Cebeci

April 2020

1 Introduction

This report is an investigation of array data structures in five different programming languages: Dart, Javascript, PHP, Python, and Rust. The same ten questions are answered about each of the programming languages' limitations on the usage of arrays. The questions are the following:

1. What types are legal for subscripts?
2. Are subscripting expressions in element references range checked?
3. When are subscript ranges bound?
4. When does allocation take place?
5. Are ragged or rectangular multidimensional arrays allowed, or both?
6. What is the maximum number of subscripts?
7. Can array objects be initialized?
8. Are any kind of slices supported?
9. Which operators are provided?
10. Are associative arrays available?

After the questions are answered for each language, a language is picked to the be the best for array operations. Finally, my learning strategy as a person who had little to no knowledge about the languages prior to this investigation will be explained.

2 Investigation of the Languages

2.1 Dart

A small note

In Dart, arrays are instances of the `List` class, therefore they are referred to as lists instead of arrays, much like the lists in Python.

Subscript types

Trying to compile the following code:

```
print( list["a"]);
```

Yields the error: `Error: A value of type 'String' can't be assigned to a variable of type 'int'.`
Therefore, it can be deduced that Dart expects an integer value as the subscript. Expressions evaluating to integer values are also accepted.

However, Dart supports associative arrays and those can be subscripted with any type. Here is an example:

```
var associative = {"Cem":"21703377", "Can": "21703376"};
print(associative["Cem"]);
```

Boundary checks

Subscripting expressions are range checked in Dart. Subscript should be less than the size and nonnegative.

Subscript range binding

Dart allows dynamic list lengths. For example:

```
var list = [1,2,3, "a", 5];
list.add(6);
print( list[5]);
```

Prints "6" without any errors. Lists can be extended. Therefore the range binding for subscripts should be at runtime since list lengths can not be determined at compile time.

Allocation time

Since the lengths are dynamic, as shown in the range binding section, allocation should be at runtime as well because the size of then list is not known at compile time.

Ragged arrays

The following code:

```
var twod_list = [ [1,2], [3,4]];
twod_list.add( [5,6,7]);
print( twod_list[1]);
print( twod_list[2]);
```

Runs and prints the expected values of [3,4] and [5,6,7]. Thus, ragged lists are allowed in Dart.

Maximum number of subscripts

Consider the following code:

```
List<List> list1 = [[1]];
List<List> list2 = [list1];
print(list2[0][0][0]);
```

Since a variable of type List<List> can be a list of <List> variables, inductively, a List<List> can be subscripted an arbitrary number of times.
In fact, the following code stops only because it runs out of memory, the code runs for smaller sizes.

```
List<List> l = [[1]];
for(int i = 0; i < 1000; i++) {
    l = [l];
}
print( l);
```

Initialization

List objects can be initialized in Dart. Here is an example of that:

```
List list = [1,2,3,4];
```

Slices

In Dart, slices are supported in the form of the `sublist()` function of the `List` class. Here is an example of its usage:

```
List to_be_sliced = ["a","b","c","d","e"];
List slice = to_be_sliced.sublist(1,3);
print( slice);
```

Operators

- **operator +** : Concatenates two lists.
- **operator ==** : Compares two list references, not an element-wise comparison.
- **operator []** : The subscript operator.

In addition to these operators, the `List` class has a number of methods to perform more complicated array operations such as shuffle and sort.

Associative arrays

Dart support associative arrays in the form of the `Map` class. The keys and values can be of any type. An example was given in the subscript types section.

2.2 JavaScript

Subscript types

JavaScript does not support associative arrays. Only subscripts of integer types are allowed for regular arrays. Example:

```
document.write(array[1.5]);
```

writes `undefined`.

Boundary checks

JavaScript does not have subscript boundary checks. The lines

```
var array = [1,2,3,4];
document.write(array[4] + "<br>");
document.write(array[-1] + "<br>");
```

write two `undefined`'s but they run without throwing errors.

Subscript range binding

Since there are no subscript boundary checks, there are no subscript ranges in JavaScript. If there were, they would need to be bound at runtime since the arrays have dynamic size.

Allocation time

The following statement appends 5 to the end of `array`:

```
array.push(5);
```

Since the array's size can change in runtime, it cannot be determined at compile time and thus the allocation should be dynamic.

Ragged arrays

Ragged arrays are allowed in JavaScript. The following code runs

```
var ragged = [[1,2],[3,4]];
ragged.push([5,6,7]);
document.write(ragged[1] + "<br>");
document.write(ragged[2] + "<br>");
```

and writes the two expected values.

Maximum number of subscripts

As with the lists in Dart, JavasScript arrays can be subscripted an arbitrary number of times. Consider the following example:

```
var max = 5;
for( var i = 0; i < 1000; i++) {
    max = [max];
}
document.write(max);
```

Again, 1000 could be incremented as high as the memory permits.

Initialization

Arrays can be initialized in JavaScript as in the example:

```
var array = [1,2,3,4];
```

Slices

Slices are supported in JavaScript in the form of the `slice()` method. Here is an example:

```
var array = [1,2,3,4,5];
document.write(array.slice(1,4) + "<br>");
```

writes "2,3,4".

Operators

- **operator +** : Exists, but does not concatenate as expected.
- **operator ==** : Compares two array references, not an element-wise comparison.
- **operator []** : The subscript operator.

In addition to these operators, array objects have a number of methods that can perform more complicated tasks. Below are some examples.

```
var array1 = [2,5,9,3,1,4];
array1.sort();
document.write(array1 + "<br>");
array1.reverse();
document.write(array1 + "<br>");
array1.shift();
document.write(array1 + "<br>");
array1.pop();
document.write(array1 + "<br>");
document.write(array1.concat([11,12,13]));
```

writes

```
1,2,3,4,5,9
9,5,4,3,2,1
5,4,3,2,1
5,4,3,2
5,4,3,2,11,12,13
```

Associative arrays

There are no associative arrays in JavaScript.

2.3 PHP

Subscript types

In PHP, associative arrays are supported. The regular indexed arrays can only be subscripted with integers but associative arrays can be subscripted with any primitive type. Following is an example:

```
$array = array(1,2,3,4,5);
echo $array[1] + "\n";

$assoc = array("Cem"=>"21703377", "Can"=> 21703376);
echo $assoc["Cem"] + "\n";
```

Boundary checks

PHP does perform boundary checks on subscript expressions and prints a notice if the program uses an invalid subscript but it does not stop executing. The following code

```
$array = array(1,2,3,4,5);
echo $array[-1] + "\n";
echo $array[5] + "\n";
```

Runs and prints two 0's, along with two notices saying that the offsets are undefined.

Subscript range binding

Consider the following code:

```
$array = array(1,2,3,4,5);
$array[5] = "6";
echo $array[5] + "\n";
```

The code runs and prints 6 because PHP has dynamic sized arrays. Note that PHP is an interpreted language, therefore the ranges could not be bound at compile time anyway. But the dynamic sizing indicates that the subscript ranges are bound again every time the size changes at runtime.

Allocation time

Since the array lengths are dynamic, allocation should be dynamic as well. New space needs to be allocated for newly added elements.

Ragged arrays

PHP does support ragged arrays. The code

```
$ragged = array();
$ragged[0] = array(1,2);
$ragged[1] = array(3,4,5);
foreach($ragged[0] as $val){
    echo $val;
} echo "\n";
foreach($ragged[1] as $val){
    echo $val;
} echo "\n";
```

prints 12 and 345, `$ragged` is clearly ragged.

Maximum number of subscripts

Similar to the arguments for Dart and JavaScript, we will make and inductive argument to state that an arbitrary number of subscripts are allowed in PHP. Consider the following code:

```
$max = array(1);
for($i = 0; $i < 1000; $i++){
    $max = array($max);
}
```

the iteration count 1000 can be incremented as high as the memory permits. Thus, any number of subscripts is allowed in PHP.

Initialization

Arrays can be initialized in PHP using the following syntax.

```
$array = array(1,2,3,4,5);
```

Slices

Array slices are supported in PHP in the form of the `array_slice()` static function. Here is an example:

```
$array = array(1,2,3,4,5,6);
print_r (array_slice($array, 2,3));
```

prints

```
Array
(
    [0] => 3
    [1] => 4
    [2] => 5
)
```

Note that the argument 3 indicates the length of the slice, as opposed to the end index in many programming languages.

Operators

- **operator +** : Union, adds the key-value pairs on the right side to the left if the key is not already in left side.
- **operator ==** : Compares key-value pairs.
- **operator ===** : Compares key-value pairs, their order and types.

An example on the union operator:

```
$a = array(1,2,3);
$b = array(3,4,5,6);
$c = $a + $b;
print_r( $c);
```

prints 1,2,3,6. Since the keys for 3,4,5 in `$b` (0,1,2) exist in `$a`, they are not added to the union. PHP also provides lots of built-in functions to perform more complicated array operations.

Associative arrays

Associative arrays are supported in PHP. Here is an example:

```
$assoc = array("Cem"=>"21703377", "Can"=> 21703376);
echo $assoc["Cem"] + "\n";
```

prints 21703377.

2.4 Python

Subscript types

For regular arrays, only integer subscripts are allowed in Python but associative arrays are also supported and they can be subscripted with other primitive types. Here are examples of both:

```
array = [1,2,3,4,5]
print(array[1])

assoc = {"Cem" : 21703377, "Can" : 21703376}
print (assoc["Can"])
```

Boundary checks

Subscript expressions are boundary checked in Python and an exception is thrown for invalid subscripts. For example, the following code crashes.

```
array = [1,2,3,4,5]
print(array[5])
```

One thing to note is that negative subscripts are valid in Python, the subscript -i for a positive int i means "the i'th element from the end." Here is an example:

```
array = [1,2,3,4,5]
print(array[-2])
```

prints 4.

Subscript range binding

Python is an interpreted language, therefore range binding can not be done at "compile time". Furthermore, Python enables dynamic lengths for its lists, therefore the range binding changes even after the variable is allocated.

Allocation time

Array allocation in Python is done when the variable is first assigned a value and every time the allocated space can not accommodate the newly added elements.

Ragged arrays

Ragged arrays are allowed in Python, here is an example:

```
ragged = [[1,2],[3,4,5]]
print(ragged[0])
print(ragged[1])
```

The code prints [1,2] and [3,4,5] as expected.

Maximum number of subscripts

As with the three programming languages discussed before, Python supports any number of subscripts. Here is a piece of code to investigate this:

```
max = [1]
for i in range (1000) :
    max = [max]
print (max)
```

The iteration number 1000 can be incremented as high as the system memory permits.
One thing to note is that the same is not true for numpy arrays, which are part of a library that is commonly used in python. They have a limitation of 32 dimensions.

Initialization

Arrays can be initialized in Python with the following syntax.

```
array = [1,2,3,4,5]
```

Slices

Python has an operator for array slices, the syntax is `array[start,end,step]`. The result of the expression is an array that has the elements of `array` that are at indices start, start + step , start + 2 * step,..., end. Here is an example:

```
to_be_sliced = range(30)
slice = to_be_sliced[0:30:6]
print(slice)
```

prints [0, 6, 12, 18, 24].

Operators

- **operator +** : Concatenates two lists.
- **operator *** : Repetition, concatenates the list with itself the specified number of times.
- **operator [:]** : Slicing, explained in the section above.
- **operator ==** : Element-wise comparison.
- **operator in** : Membership, true if the element is in the list.

In addition to these, the list class has lots of methods to implement more complex array operations.

Associative arrays

Python allows associative arrays, they are calls `dict` in Python. They have a slightly different syntax:

```
assoc = {"Cem" : 21703377, "Can" : 21703376}
print(assoc["Can"])
```

prints 21703376.

2.5 Rust

Subscript types

Only integer subscripts are allowed in Rust. Rust does not support associative arrays in its standard library.

Boundary checks

Boundary checks do exist in Rust. Running the following code:

```
let array:[i32;5] = [1,2,3,4,5];
let index = 5;
println!("{}",array[index]);
```

yields a runtime error saying "index out of bounds: the len is 5 but the index is 5".

Subscript range binding

Rust, in contrast to the other languages investigated here, does not support dynamic sized arrays. Array lengths are either integer literals or constants. Therefore, the array lengths are known at compile time. The range binding is also at compile time.

Allocation time

The space allocation takes place when the array is first declared. Because its length does not change over time, no other allocations need to happen.

Ragged arrays

Ragged arrays are not allowed in Rust's standard library. Every element's size should be the same in a multi-dimensional array. Here is an example of a multi-dimensional type declaration:

```
let mut multd:[[i32;5];7];
```

It's impossible to declare a non-rectangular type like this.

Maximum number of subscripts

Since the types [a;b] and [[a;b];b] are different in Rust, an argument like the previous ones can not be made here. Therefore, we have no way of proving infinite number of subscripts can be made. That being said, using Python's numpy library's limit of 32 as a reference, I've tried and confirmed that 33 subscripts are legal in Rust.

Initialization

Initialization is possible for multi-dimensional arrays in Rust as well as linear arrays. Here are examples of both:

```
let array = [1,2,3,4,5];
println!("{}",array[1]);

let twod = [[0;5];5];
println!("{}",array[1][3]);
```

prints 2 and 0.

Slices

Array slices are supported in Rust and there exists an operator for slicing. Here is an example of its use:

```
let array = [1,2,3,4,5]
let slice = &array[1..3];
println!("{:?}",slice);
```

prints `2, 3` as expected.

Operators

- **operator [:]** : Slicing, explained in the section above.
- **operator ==** : Element-wise comparison.

Associative arrays

Associative arrays are not supported in Rust's standard library. The class `std::collections::HashMap;` can be used for associative arrays but they are not in the standard library.

3 Language Preference

In my opinion, Python is the best programming language to work with arrays out of the 5 options discussed in this report. Arrays in Python have dynamic size, they can be initialized, multi-dimensional arrays can be ragged, they can have any number of dimensions; all of those are important contributors to the language's writability. On top of that, the subscript expressions are boundary checked in Python, increasing the language's reliability. What really makes the difference between Python and the other investigated languages is that Python provides more operators than any of those languages, including the slice operator `[:]`, the repetition operator `*` and the membership operator `in`, which add to the language's writability. In addition, another feature of Python that is not discussed in this report is list comprehensions. List comprehensions allow the programmer to work with arrays easily by providing shortcut expressions for the most used array operations. Another feature that increases the writability is that negative integers can be used to count from the end of the array, though this feature has a reliability trade-off, it increases the convenience of the language. Last but not lease, Python also supports associative arrays in the form of `dict`'s. To sum up, Python provides a number of features to work with arrays without missing out on the features the other languages have and thus, I would prefer Python to work with arrays.

4 Learning Strategy

In this section, I will explain how I answered the questions about the five languages I knew very little about.

4.1 Dart

First, I installed Dart on my computer, which has an Ubuntu operating system. Then, I took a look at Dart's official website and found a quick language tour that can help me get started quickly. Here is the link to the webpage: https://dart.dev/guides/language/language-tour. I read the beginning of the language tour and wrote a simple "Hello, World!" program on my computer. Then I read the section about arrays. I also read about Dart arrays from TutorialsPoint, which can be found at here:https://www.tutorialspoint.com/dart_programming/dart_programming_lists.htm. I used those two resources and experimented on my computer for Dart.

4.2 JavaScript

I knew that JavaScript had something to do with web development but I did not really know what JavaScript is. Thus, I started by reading an introduction to JavaScript article at https://javascript.info/intro. Then, I read the JavaScript quickstart tutorial on W3Schools, found at https://javascript.info/intro. I installed a plugin to the IDE Visual Studio Code in my computer to help me with the syntax and wrote a "Hello, World!" program. After that, I experimented on my computer, consulting W3Schools and TutorialsPoint where i needed to.

4.3 PHP

Similar to JavaScript, i knew that PHP was a part of web development but I did not know how it was different from JavaScript. I started by reading the W3Schools introduction at https://www.w3schools.com/php/php_intro.asp. Then, I decided to install PHP on my computer so I would not have to use my browser for every experiment and I could use my terminal. I installed PHP and the corresponding VSCode plugin. I also read the array-related sections on W3Schools, which allowed me to experiment since i learned all the syntax. I also had to refer to the manual at https://www.php.net/manual/en/index.php a couple of times.

4.4 Python

Before starting to write this report, I already had some experience with Python because I had to use Python during my first internship. Because of that, I already knew the answers to most of the questions and had Python installed on my computer already. I started experimenting with Python right away and searched the forums on stackoverflow whenever I needed to double check my

knowledge or when I was confused. The class discussion we had about arrays in Python also helped me with syntax and some concepts like associative arrays.

4.5 Rust

I had no experience with Rust prior to this report, nor had I heard of it. Therefore I started by taking a look at the official Rust website and found a very useful series of examples called "Rust by example",which can be found at https://doc.rust-lang.org/stable/rust-by-example/. I also installed Rust on my computer and followed the examples until I felt confident about experimenting on my own. Rust has very complicated concepts such as ownership and pointers. I tried not to think about those features as my time was limited and those were not really the focus of my studies. I searched stackoverflow at points that confused me.