21801744
Uğur Erdem Seyfi
CS315 - 01

CS 315 Homework 01 Report

Javascript

1. What types are legal for subscripts?

It seems like all of the types can be put in subscripts in JavaScript. However the way that it works, especially for the objects seems confusing as it is shown in the example code below:

```
// What types are legal for subscripts?
// 1. What types are legal for subscripts?

var arr = []; // empty array

console.log( "1) What types are legal for subscripts? \n");

arr[3] = 5;
arr["h"] = "string";
arr[1.3] = "float number";
arr[true] = "boolean true";
arr[[2, 3]] = "array ";
arr[{ x: 5, y: 3 }] = "object ";
arr[{ x: 10, y: 15 }] = "object 2";
arr[TryClass] = "function ";
arr[new TryClass()] = "try class";

for (object in arr) {
       console.log(object + " : " + arr[object]);
}

// Try Class to initialize an object
function TryClass() {
       this.x = "x";
}
```

2. Are subscripting expressions in element references range checked?

In JavaScript, there is a type called ***undefined***, when you reference to an index that is not defined, the respective operation returns undefined as it is shown in the following program:

```
// 2. Are subscripting expressions in element references range checked?
```

```
console.log( "\n2. Are subscripting expressions in element references range checked? \n");

console.log( "Element at index 10: " + arr[10]); // Element at index 10: undefined
```

3. When are subscript ranges bound?

They are not bound. Since every undefined index returns ***undefined*** type and it is allowed to use all types in the subscript operator as we have already shown in question 1, there is no limitation.

```
// 3. When are subscript ranges bound?

console.log("\n3. When are subscript ranges bound? \n")
arr[-1] = 10;
arr[100] = 50;
console.log( arr[-1]); // 10
console.log( arr[100]); // 50
console.log( arr[500]); // undefined
```

4. When does allocation take place?

JavaScript arrays use Dynamic Allocation.

Since JavaScript is a scripting language, it allocates the needed memory simultaniously as it reads the code.

```
// 4. When does allocation take place?
console.log("\n4. When does allocation take place? \n")
first = [1,2,3,4,5,6]; // memory is allocated for the array
second = first; // second functions in a way that is similar to a pointer
second[2] = 1;
console.log( first); // [1,2, 1, 4, 5, 6]
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

As the following program shows, both ragged, rectangular (multidimensional) arrays are allowed in JavaScript:

```
// 5
console.log( "\n2.5. Are ragged or rectangular multidimensional arrays allowed, or both? \n");
var row0 = [0,1];
var row1 = [1,2,3];

var ragged_arr = [ row0, row1];
for(var i = 0; i < ragged_arr.length; i++){
       console.log( ragged_arr[i]);
}
console.log("---");

multidim_array = [[0,1,2],[3,4,5],[6,7,8]];
```

```
for(var i = 0; i < multidim_array.length; i++){
       console.log( multidim_array[i]);
}
```

6. What is the maximum number of subscripts?

As many of you want:

```
// 6. What is the maximum number of subscripts?
console.log( "\n6. What is the maximum number of subscripts? \n");

var inception = [[[[[["As many of you want!"]]]]]];
console.log( inception[0][0][0][0][0][0]); // As many of you want!
```

7. Can array objects be initialized?

Yes. They can be initialized with determined values when they are declared.

```
// 7. Can array objects be initialized?
init_arr = [1,2, "ugur"];
```

8. Are any kind of slices supported?

There is a built-in .slice() function for that in JavaScript array objects.

```
// 8. Are any kind of slices supported?
var slice_arr = [0,1,2,3,4];
var sliced = slice_arr.slice(1, 3);
console.log( sliced);
```

9. Which operators are provided?

You can put binary infix operators between arrays in JavaScript, but they probably won't act like you expected. For example you make some operations between arrays with * kind of operations you do not get an error but you get the result *NaN* which stands for "not a number". Similarly when you use '+' between two arrays it returns a string. In that sense maybe we can say there is no specifically designed infix operators for arrays in JS. However there are certainly built-in methods such as sort and slice that can be counted as unary operators (since operators are also functions in mathematics).

Example:

```
// 9. Which operators are provided?
console.log("\n9. Operators \n");

arr_1 = [3,2,1];
arr_2 = [1,2,3]
console.log( arr_1 * 5); // prints NaN
console.log( arr_1 * arr_1); // prints NaN
console.log( arr_1 + arr_2); // prints 3,2,11,2,3

arr_2 = arr_1; // obvious assign operator.
```

```
console.log( arr_1.sort() ); // prints "[1,2,3]". Note that arr_2 is also sorted.
console.log( arr_2); // [1,2,3]
```

10. Are associative arrays available?

Yes they are, arrays in JavaScript can take any type in their subscript so as a result of this we can also create associative arrays.

```
// 10. Associative arrays
console.log("\n10. Associative arrays \n");

var associative = [];
associative["one"] = 1;
associative["two"] = 2;
associative["three"] = 3;

for(key in associative){
       console.log( key + " : " + associative[ key]);
}
```

Python

1. What types are legal for subscripts?

Only integers or slices, for example the following code results with the output: `TypeError: list indices must be integers or slices, not str`

```
arr = []
arr["deneme"] = 1
arr[true] = 2
```

2. Are subscripting expressions in element references range checked?

Yes, it is. If you have an array with length 3, you can use integers from -3 to 2 in subscripts. However if you exceed this numbers you get “IndexError: list assignment index out of range”

Example program:

```
# 2

arr = [1,2,3]
arr[-3] = 5;
arr[2] = 1;

print( arr) # [5,2,1]
```

3. When are subscript ranges bound?

If you have an array with length 3, you can use integers from -3 to 2 in subscripts. However if you exceed this numbers you get an error.

4. When does allocation take place?

Python arrays use Dynamic Allocation.

Like JavaScript, Python is also a scripting language, it creates allocates a memory as it sees an array created simultaniously.

```
# 4. When does allocation take place?
print("-4-")

# referencing
arr = [1, 2, 3, [5,5]] # allocates memory space
arr_2 = arr # does not allocate memory space
arr_2[1] = 77
print( arr) # [1,77,3, [5,5]]

# cloning
arr_3 = []
for i in range(0, len(arr)):
       arr_3.append( arr[i]) # arr_3 might allocate a new memory place as it is appended like
ArrayLists in Java

print("before change\n arr is: " + str(arr) )
print(" arr_3 is: " + str(arr_3) )

arr[2] = 11
arr[3][1] = 1
print("after change\n arr is: " + str(arr) + "\n arr_3 is: " + str(arr_3))
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

Yes.

Example program:

```
print( "-5-")


print("ragged")
ragged_arr = [ [1,2,3], [1,2], [1]]
for i in range(0, len(ragged_arr)):
print( ragged_arr[i])

print("rectangular")
rectangular_arr = [ [1,2,3], [4,5,6] ]
for i in range(0, len(rectangular_arr) ):
       print( rectangular_arr[i] )
```

6. What is the maximum number of subscripts?

There is no such limit.

```
# 6. What is the maximum number of subscripts?

arr = [[[[[0]]]]]
print( arr[0][0][0][0][0])
```

7. Can array objects be initialized?

Yes. We can initialize array objects at the time we create the array.

```
# 7. Can array objects be initialized?

arr = ["mehmet", 132, 7.3]
```

8. Are any kind of slices supported?

Of course! Python is a very rich language on this aspect. You can even slice the array with respect to certain index incrementations, with that feature, we can, for example get the reverse sorted array.

```
# 8. Are any kind of slices supported?

arr = [0,1,2,3,4,5,6,7]
print( arr[1:4] ) # [1, 2, 3]
print( arr[::-1]) # [7, 6, 5, 4, 3, 2, 1, 0]
print( arr[::-2]) # [7, 5, 3, 1]
```

9. Which operators are provided?

= and + operators are allowed for arrays, + is used for the concetination. We cannot multiply two arrays together with * but we can multiply an array with an integer and it extends the array with same elements (similar the way string * int works in python).

```
# 9. Which operators are provided
print("-9-")
arr_1 = [0,1,2,3,4]
arr_2 = [0,1,3]

# print( arr_1 * arr_2) -> error
print( arr_1 + arr_2) # [0, 1, 2, 3, 4, 0, 1, 3]
print( arr_1 * 2)
# print( arr_1 + "asd") -> error
# print( arr_1 + 5) -> error
```

10. Are associative arrays available?

You cannot do associative operations (using an array like a hashmap) on arraylists but one can use the dictionaries for that purpose.

Dart

1. What types are legal for subscripts?

According to the compiler error that I get, the only types that are legal for subscripts are integers in Dart.

```
// 1. What types are legal for subscripts?
var arr = [1,2,3,4];
print( arr);

/*
arr[10] = 5;
arr[ true] = 10;
arr["deneme"] = 3;

Error: The argument type 'String' can't be assigned to the parameter type 'int'.
*/
```

2. Are subscripting expressions in element references range checked?

Yes. Dart expects for programmers to use the exact numbers from 0 (inclusive) to length (exclusive).

```
// 2. Are subscripting expressions in element references range checked?
print("-2-");
/*
print( arr[5] );

Unhandled exception:
RangeError (index): Invalid value: Not in range 0..3, inclusive: 5
/*
```

3. When are subscript ranges bound?

You are bound by the indexes from 0 (inclusive) to length (exclusive), as we have already shown in the previous example.

4. When does allocation take place?

Dart arrays use Dynamic Allocation.

Dart appears to be first compiled into JavaScript code and then executed in JavaScript code. So the answer here is the same as for JavaScript. It makes the allocation when it encounters with array operations as in JS.

5. Are ragged or rectangular multidimensional arrays allowed, or both?

Yes.

```
// 5. Are ragged or rectangular multidimensional arrays allowed, or both?
print("-5-");

var jagged_arr = [ [1], [1,2], [1,2,3] ];
var square_arr = [ [1,2,3], [4,5,6], [7,8,9] ];

print( jagged_arr);
print( square_arr);
```

6. What is the maximum number of subscripts?

It seems like there is no limit.

```
// 6. What is the maximum number of subscripts?

print("-6-");
var subs_arr = [[[[[0]]]]];
print( subs_arr[0][0][0][0][0] );
```

7. Can array objects be initialized?

Yes.

```
// 7. Can array objects be initialized?
print("-7-");
var initialized = [1.5, "mahmut", 'A'];
print( initialized);
```

8. Are any kind of slices supported?

There is built-in sublist method that allows slicing in Dart.

```
// 8. Are any kind of slices supported?
print("-8-");
var slice_arr = [0,1,2,3,4];
print( slice_arr.sublist(1,3) );
```

9. Which operators are provided?

As in Python, in Dart + also function as a merger for arrays. However unlike Python, operations such as * is not provided and not allowed in the language.

```
// 9. Which operators are provided?
print("-9-");
var arr_1 = [1,2];
var arr_2 = [1,2];

print(arr_1 + arr_2);
```

```
// print(arr_1 * arr_2); -> Error
// print(arr_1 * 2); -> Error
```

10. Are associative arrays available?

It is not allowed to directly use strings in subscripts of arrays but there are data structures called maps in Dart that functions as associative arrays.

```
// 10. Are associative arrays available?
print("-10-");
 var map = {
'isim': 'a',
'yas': 20
};

print( map['yas'] );
```

Rust

1. What types are legal for subscripts?

When I tried to put a string as a subscript in an array, the compiler gave me the error: “slice indices are of type `usize` or ranges of `usize`”. Which means array indexes can be only the type usize. I searched the type usize and appearantly it is just an unsigned integer.

```
println!("-1-");
let arr = [1,2,3,4,5];

println!("{:?}", arr);
println!("{}", arr[1]);
/*
arr["str"] = 3;
slice indices are of type `usize` or ranges of `usize
*/
```

2. Are subscripting expressions in element references range checked?

Yes. If you exceed length the compiler gives error in rust.

```
println!("-2-");

/*
arr[8] = 10; // index out of bounds: the len is 5 but the index is 8
arr[-1] = 3; // the trait bound `usize: std::ops::Neg` is not satisfied
*/
```

3. When are subscript ranges bound?

As it is shown above, they are upperbounded by length and lowerbounded by 0.

4. When does allocation take place?

Rust arrays use Static (compile-time) Allocation.

In Rust, the memory is allocated at the compile-time, this is also understandable since the arrays in Rust are fixed sized and immutable by default.

5. Are ragged or rectangular multidimensional arrays allowed, or both?

Ragged arrays are not directly built-in for arrays in the language Rust. When you try to initialize an array with arrays that have different sizes, it gives a compiler error. On the other hand rectangular and multidimensional arrays are allowed in the language.

```
println!("-5-");

let mut rectangular_arr: [[ i32; 5]; 6] = [[0; 5]; 6];
for i in 0..rectangular_arr.len(){
       for j in 0..rectangular_arr[i].len(){
              rectangular_arr[i][j] = (i * rectangular_arr[i].len() + j) as i32;
       }
}
/*
let ragged_arr = [ [1], [1,2], [1,2,3] ];
^^^^^ expected an array with a fixed size of 1 elements, found one with 2 elements
*/

println!("{:?}", rectangular_arr);
```

6. What is the maximum number of subscripts?

As many as you want.

```
println!("-6-");

let subs_arr = [[[[[0]]]]];
println!("{:?}", subs_arr[0][0][0][0][0]);
```

7. Can array objects be initialized?

Yes. However the Rust compiler expects arrays have the same type of elements.

```
println!("-6-");

let arr: [[i32;2]; 2] = [[1,2],[2,3]];
/*
let arr_diff = [ "hi", 123];
= note: expected type `&str`
found type `{integer}`
*/
println!( "{:?}", arr);
```

8. Are any kind of slices supported?

Yes. We can slice using &arr_name[x..y] syntax.

```
println!("-8-");

let slice_arr = [0,1,2,3,4,5,6];
println!("{:?}", &slice_arr[2..5]);
```

9. Which operators are provided?

Unlike previous languages we examined, Rust do not support + operator for arrays. However this is not something unexpected, since Rust is known with its reliability it is not surprising for it to not allow use + for arrays. It seems like the only operator that is allowed for arrays in the Rust is = and == kind of operators.

```
println!("-9-");

let arr_1 = [0,1,2];
let arr_2 = [3,4,5];

/*
println!("{:?}", arr_1 + arr_2);
println!("{:?}", arr_1 * arr_2);
*/

if arr_1 == arr_2{
       println!( "arr_1 is equal to arr_2"); // this call is executed
}
```

10. Are associative arrays available?

In Rust, there is a type called HashMap which funtions as associative arrays, however, you cannot directly use arrays as associative arrays in Rust.

PHP

1. What types are legal for subscripts?

Seems like PHP works as the JavaScript works. It allows users the enter string, boolean and numbers as indices. However unlike JavaScript objects such as arrays are not accepted as index in PHP.

```
echo " \n -1- \n";

$arr = array(1, "ugur", true, false);
$arr["a"] = "a";
$arr[true] = "5";
$arr[false] = 2;
$arr[-1] = 3;
```

```
$ind_arr = array(1,2);
# $arr[ $ind_arr ] = "array"; -> Illegal

foreach ($arr as $key => $value) {
	echo "{$key} => {$value} \n ";
}
```

2. Are subscripting expressions in element references range checked?

It is not checked before hand but if you try to access and undefined index it notices it and gives the warning “PHP Notice: Undefined offset: 99 in /home/runner/CS315PHPCLI/main.php on line 18”

```
echo "\n -2- \n";

echo $arr[99];
```

3. When are subscript ranges bound?

They are not bound. However if a given index is undefined the respective subscript operation results in an error as we have showed in the above example.

4. When does allocation take place?

PHP arrays use Dynamic Allocation.

When an array is initialized and when = op is used between two arrays. Unlike other languages above, in PHP = operator clones the context of one array to another so it allocates another (new) memory space for the new variable.

```
$first = array(1,2,3,4);
$second = $first;
$second[0] = 2;
print_r($first); # [1,2,3,4]
print_r($second); # [2,2,3,4]
```

5. Are ragged or rectangular multidimensional arrays allowed, or both?

They are both allowed.

```
echo "\n -5- \n";

echo "\n rectangular \n";

$rect_arr = array( array(1,2,3), array(4,5,6), array(7,8,9));
foreach ($rect_arr as $row) {
	foreach($row as $value){
		echo "{$value} ";
	}
	echo " \n";
}

echo "\n ragged \n";
```

```
$ragged_row1 = array(1,2);
$ragged_row2 = array(4,5,6);

$ragged_arr = array( $ragged_row1, $ragged_row2);

foreach ($ragged_arr as $row) {
        foreach($row as $value){
                echo "{$value} ";
        }
        echo " \n";
}
```

6. What is the maximum number of subscripts?

There is no limit.

```
echo "\n -6- \n";

$subs_arr = array(array(array(array(0))));
echo $subs_arr[0][0][0][0];
```

7. Can array objects be initialized?

Yes. Arrays can be initialized at the declaration process, also like JavaScript and Python, PHP allows programmers to add different types of elements into an array.

```
$init_arr = array("deneme", 123, 5 => 3, 1, true, true, array(1,2));
```

8. Are any kind of slices supported?

There is an array_slice() method that is built for that in the PHP language.

```
echo "¨\n -8- \n";

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

9. Which operators are provided?

+ operator between arrays sort of functions as union operator in mathematics. However, when there is a two values with same indices, operator choose to keep the left operands value.

```
echo "\n -9- \n";

$arr_1 = array(1,2);
$arr_2 = array(1,5, 4);

$sum = $arr_1 + $arr_2;
print_r( $sum); # 0 => 1, 1 => 2, 2 => 4
```

```
# $subs = $arr_1 - $arr_2; -> error
# print_r( $arr_1 * $arr_2); -> error
```

10. Are associative arrays available?

Yes. We can even implicitly declare associative arrays as in the following example:

```
echo "\n -10- \n";

$associative = array( "erdem" => 2, "ugur" => 1, "seyfi" => 3);
foreach($associative as $key => $value){
echo "{$key} => {$value} \n";
}
```

Best Language for Array Operations?

From the languages above, the ones that particularly interested me was Python, PHP and Rust.

The reason why I like Python is it's easing the process of writing a code, it really provides nice features such as making easier slicing in several ways with [::] syntax, you don't need to lose time for writing types and you can even iterate your array backwards by using negative indexes. JavaScript can also be considered as a good language in terms of writability, however in my opinion it is way worse then Python in terms of reliability (python at least checks range bounds).

The reason why I like Rust is its reliability. Rust feels like a language that really cares reliability, it has type-checking features, it forces the programmer to put the same kind of elements into an array, it even has a "mut" keyword to be sure the user realizes that he is dealing with a mutable/immutable kind of an array. In my opinion, it is also the best language from the ones above in terms of readability.

The particular reason that I like PHP is its way of dealing with associative arrays. You can explicitly initiliaze an array while defining the associations in the same expression. The fact that this language clones an array into another array when = operator is used might be considered as a more intuitive introduction for beginners. On the otherahand it might also be considered as a poor way of managing memory.

If I were to be mastering a language from the ones above, I would personally choose Rust since I give more importance to reliability and readability more than writability.

My Learning Strategy

Before setting up a local environment on my PC for starting on a particular language to study on, I decided to get an advantage of an online IDE and compiler in order to make the process of trying new things easier for me. I thought this kind of an approach would speed up my learning progress. I asked one of my friends whether or not he knows such online IDE and he suggested me repl.it. After that I decided to starting with the languages that I am mostly familiar with (which were Python and JavaScript), this way after doing the stuff in the languages that I am familiar, I could compare my code in those languages with the other ones and see the differences in a better way. For each particular language I looked except Python and JavaScript, I felt a need for a rewise. Before jumping to each one of these language, I looked example programs that are written in these languages to recognize what kind of expressions are allowed and what kind of syntax they have.

For each question, I first tried to write non-trivial experiments about them, this way if there is a problem I would see the relevant error message in the console during the execution/compilation process beforehand. Since some of the questions could be answered by just looking at these kind of compiler errors, I believe this kind of an approach also speeded up my progress since several questions are related in this way.

In this homework, I realized that the process of learning a new programming language is mostly about knowing or being familiar the concepts underneath of that language (and in other programming languages generally) beforehand. The particular syntax that a programming language follows do not matter that much, as much as you have the fundamental understanding of the motivation of a language (and the build process of programming languages), it really becomes easier to adapt to it. My experiences led me to believe that being good on a programming language is something that is effected alot by being able to understand how the developers of that language think.

I also think being comfortable using documentations and searching through the internet for things like syntax specific things is one of the key proponents of speeding the process of learning a programming language. In this assignment, there were several sources that I looked through that helped me, those sources are listed below:

- https://dart.dev/guides
- https://www.educative.io/edpresso/how-to-create-an-array-in-dart
- https://www.oreilly.com/library/view/dart-1-for/9781680500479/f_0023.html
- https://www.educative.io/edpresso/arrays-in-rust
- https://www.tutorialspoint.com/rust/rust_slices.htm
-https://www.geeksforgeeks.org/php-arrays/