CS315 HW1 Report

Efe Beydoğan

21901548

Section 1

Dart

1) What types are legal for subscripts?

The arrays in the Dart language are represented as List objects. The arrays only accept integers as subscript types. These integers start from 0 and extend to n – 1, where n is the size of an array. Square brackets [] are used to access array elements.

Code sample for Q1

```dart
int listSize = 5;
var list = new List(listSize); // creates a fixed size array
list[0] = 0; // initializing can be done this way for fixed size arrays
list[1] = 1;
list[2] = 2;
list[3] = 3;
list[4] = 4; // this would be the final legal statement for initializing elements
             // this is (n-1), so 4 is the last integer from 0 that will be accepted as a subscript for this array
// list[5] = 5; -> this statement will be illegal since list has a fixed size of 5
```

2) Are subscripting expressions in element references range checked?

Subscript expressions in Dart are range checked and going out of bounds is illegal. Only integers between 0 and n - 1 inclusive are accepted.

Code sample for Q2

```dart
var growableList = [1,2,3]; // creates a growable list
// print( growableList[3]); -> gives an exception "Index out of range: index should be less than 3"

var fixedList = new List(3);
fixedList[0] = 1;
fixedList[1] = 2;
fixedList[2] = 3;
// print( fixedList[5]); -> this also gives an exception "RangeError: index (5) must be in the range [0..3)"
```

Dart offers two types of arrays in terms of size, fixed and growable. For both of these type of arrays, subscripting expressions in element references are range checked and trying to access an element that is not in the range [0, n) throws an exception. The above code sample displays this property of Dart.

3) When are subscript ranges bound?

As discussed in the answer for Q2, Dart offers two types of arrays: fixed length and growable. The fixed length array is unable to grow or shrink at runtime. It maintains the length it was assigned initially, **so the subscripts for a fixed length array are bound to be [0, n) at compile time**.

```
var fixedList = new List(3); // subscripts are bound to be [0, 3) and can't be changed at runtime
fixedList[0] = 3; // legal
fixedList[2] = 12; // legal
print( fixedList[1]); // prints “null” but it is legal
// fixedList.add( 4); -> this statement throws an exception since the fixed length array cannot be grown: "Cannot add to a fixed-length list"
// fixedList.removeAt(2) -> illegal, can’t remove an element from fixed length array
```

The growable array however, can be extended or shrunk at runtime and the subscripts are bound according to the new size of the array, **so subscript binding takes place at runtime**. However, trying to access an index that is not in the interval [0, n) still results in an out of bounds error, although “n” is changeable.

```
var growableList = new List(); // creates a growable list
// growableList[0] = 5; -> throws an exception since the size of the array is 0, there are no elements in the array
growableList.add( 3); // a legal expression, since the list can be grown
growableList[0] = 2; // legal since the first element exists now
growableList.add(5);
growableList.add(10);
// print( growableList[5]); -> illegal, it is out of bounds
growableList.removeAt(1); // removes the element at index 1, which is 5. so the array can be shrunk
```

Dart also offers compile-time constant lists, whose subscript ranges are bound at compile-time, thus cannot be changed at runtime. Additionally, contrary to fixed length lists, constant lists’ elements cannot be changed either.

```
var constList = const [10,11,12];
// print( constList[6]); -> error
// constList[1] = 1; -> this will cause an error, the const list is immutable
```

4) When does allocation take place?

For fixed length lists, allocation takes place after declaration, at compile time. The size of the array is fixed in the beginning and the elements of the array can be reached with the subscript operator. However, if the element at a specified index hasn’t been initialized with a value, then it is initialized to null by default.

```
var fixedList = new List(3);
print( fixedList[0]); // prints "null"
```

For growable lists, the allocation takes place at runtime and if the list has been initialized with the new operator, then trying to access an index that doesn’t exist results in an error:

```
var growableList = new List();
// print( growableList[0]); -> error
growableList.add(9);
growableList[0] = 6; // now this can be done since an element has been added
print( growableList[0]); // after an element has been added, now the element at that
index can be printed
```

Similarly for compile time constant lists, the space for them is allocated after they are declared in compile time and all the elements are supposed to be specified during the declaration as well, so none of them are uninitialized.

```
var constList = const [1,2,3];
print( constList[2]); // prints "3"
```

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

Dart language offers support for both ragged and rectangular multidimensional arrays.

```
var rectangularList = [
        [31, 32, 33],
        [34, 35, 36],
        [37, 38, 39]
    ];
print( rectangularList); // prints [[31, 32, 33], [34, 35, 36], [37, 38, 39]]

var raggedList = [
        [31],
        [34, 35],
        [37, 38, 39]
    ];
print( raggedList); // prints [[31], [34, 35], [37, 38, 39]]
```

6) Can array objects be initialized?

For an integer array of fixed size, the elements can be initialized after the array has been declared, so array objects can't be initialized in a single assignment statement:

```
var fixedList = new List(3);
fixedList[0] = 3;
fixedList[1] = 9;
fixedList[2] = 13;
```

For a growable list, elements can be initialized when the array is declared and then specific elements may be changed later. Alternatively, the List.generate constructor can be used to create an array:

```
var growableList = [1,2,3];
growableList[0] = 3;
growableList[1] = 9;
growableList[2] = 13;

var generateList = new List<int>.generate(100, (i) => i + 1); // creates a list that
contains elements [1,100]
print( generateList);
```

7) Are any kind of slices supported?

Slicing in Dart is done with the "sublist()" method.

When two parameters are supplied to the sublist(a, b) method, the method returns a new list containing the original array's elements starting from index "a" and ending at index "b-1". So the starting index supplied to the method is inclusive, while the ending index is exclusive:

```
List twoParam = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
List twoParamSlice = twoParam.sublist(5, 10);
print(twoParamSlice); // prints [6, 7, 8, 9, 10]
```

If no ending index is provided, then a new list starting with the element at the given starting index and ending with the last element in the array is returned:

```
List oneParam = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
List oneParamSlice = oneParam.sublist(3);
print( oneParamSlice); // prints [4, 5, 6, 7, 8, 9, 10]
```

If the ending index is smaller than the start index, or a negative index is provided, the sublist() method will throw an error. Negative indexing is not supported with the sublist() method in Dart.

8) Which operators are provided?

Dart provides the "+" operator for lists, which can be used to append two lists as follow:

```
List<int> list1 = [49];
List<int> list2 = [50, 51];
List<int> list3 = list1 + list2;
print( list3); // prints [49, 50, 51]
```

Also, the equality operator == is supported for lists but this operator only checks references, not contents:

```
print( list1 == list2); // prints false
```

The subscript operator is [], used for reaching elements in the array. Array elements can be altered with []= operator, as follows: arr[1] = 3 etc.

Resources used for Dart:

https://dart.dev/guides/language/language-tour#a-basic-dart-program

https://dartpad.dev/?null_safety=true

https://www.tutorialspoint.com/dart_programming/dart_programming_lists.htm

http://blog.sethladd.com/2011/12/lists-and-arrays-in-dart.html

https://medium.com/flutter-community/working-with-multi-dimensional-list-in-dart-78ff332430a https://medium.com/flutter-community/useful-list-methods-in-dart-6e173cac803d

https://discoverflutter.com/how-to-slice-a-list-in-dart/

https://api.dart.dev/stable/2.14.4/dart-core/List-class.html

JavaScript

1) What types are legal for subscripts?

The JavaScript language has 6 data types defined: Primitives (boolean, number, string, null, undefined, bigint) and objects (array is also an object). All of these types can be used as subscripts in a JavaScript array. The JavaScript arrays accept even other arrays as legal subscripts, since arrays are also objects. In the documentation, this is explained as subscripts being "coerced into a string by the JavaScript engine through an implicit toString conversion" (from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) . Subscript operator is [].

```javascript
let arr = ['a','b','c'];
console.log( arr[2]); // prints "c" on the console
console.log( arr['2']); // again prints "c" on the console, so a string is accepted as a subscript

arr['efe'] = 33; // legal
console.log( arr['efe']); // prints "33"

arr[true] = 7; // legal statement
console.log( arr); // prints [ 'a', 'b', 'c', true: 7 ]

console.log(arr['2']); // prints "c"
console.log(arr['02']); // prints "undefined" since '2' and '02' are different strings
arr[undefined] = 5; // legal
arr[null] = 9; // legal
let arr2 = [3,4];
arr[arr2] = 43; // legal
const person = {firstName:"Efe", lastName:"Beydogan", age:20, eyeColor:"brown"};
arr[person] = person; // legal
console.log( arr); /* prints [ 'a',
  'b',
  'c',
  efe: 33,
  true: 7,
  undefined: 5,
  null: 9,
  '3,4': 43,
  '[object Object]': { firstName: 'Efe', lastName: 'Beydogan', age: 20, eyeColor: 'brown' } ] */
```

2) Are subscripting expressions in element references range checked?

JavaScript doesn't throw exceptions if an index that is out of the bounds of the array is tried to be reached but returns "undefined" instead of a garbage value, so **there is range checking**. Similarly, since arrays accept all 6 data types as valid subscripts, trying to access an invalid index simply results in receiving the "undefined" type.

```javascript
let arrQ2 = [1, 2, 3];
console.log( arrQ2[1]); // prints "2"
console.log( arrQ2[7]); // prints "undefined"
console.log( arrQ2["CS315"]); // again prints "undefined", however it is not an error
```

3) When are subscript ranges bound?

JavaScript arrays are heap-dynamic, meaning subscript ranges can change any number of times and are **bound at runtime**. Also, any index that is an integer between 0 and 2^32-1 is a valid index and can be initialized, without an error. Integers exceeding this number or below it are regarded as property names, but they are also still legal subscripts and can be initialized, such as negative numbers. If a negative number is used as an index, then it simply becomes a key-value pair where the negative number is the key. JavaScript modifies the length accordingly when a valid integer index is initialized, however doesn't modify length if a property name is initialized. For example, in the below code sample the array initially has 3 elements, and when arrQ3[10] is initialized, then the array length becomes 11. If arrQ3['word'] is initialized too, the length still remains to be 11.

```javascript
let arrQ3 = [1, 2, 3];
console.log( arrQ3[1]); // prints "2"
console.log( arrQ3[7]); // prints "undefined"
arrQ3[10] = 5; // legal, even though the array has a length of 3 at the moment
arrQ3['word'] = 9;
console.log( "length: " + arrQ3.length); // prints "length: "11", there are 7 empty items and although arrQ3['word'] is also initialized, it doesn't alter the length
arrQ3[4294967294] = 91; // this is 2^32 - 2, and is a legal index
console.log( arrQ3); // prints [ 1, 2, 3, <7 empty items>, 5, <4294967283 empty items>, 91 ]
arrQ3[4294967295] = 101;
arrQ3[-3] = 101; // not a legal index but a property name
console.log( arrQ3); /* prints [ 1,
  2,
  3,
  <7 empty items>,
  5,
  <4294967283 empty items>,
  91,
  '4294967295': 101,
  '-3': 101 ] */
```

4) When does allocation take place?

For JavaScript arrays, allocation is dynamic and the memory allocated for an array can change any number of times throughout the lifetime of the array. **The allocation takes place at runtime**. Also, the array may contain empty, or undefined, elements and these empty locations are simply not allocated. For example, an array can contain elements only at the 0th and 100th indices, the indices in between don't take up space in memory and are allocated as they are initialized.

```javascript
let arrQ4 = new Array( 15, 16, 17); // creates an initial array
arrQ4[8] = 31; // legal statement, shows array length can be changed
arrQ4['HAG'] = "Halil Hoca"; // also legal, array can be grown arbitrarily
console.log( arrQ4); // prints "[ 15, 16, 17, <5 empty items>, 31, HAG: 'Halil Hoca' ]"
arrQ4.length = 1; // length can be shortened this way, doesn't affect property names
console.log( arrQ4); // prints [ 15, HAG: 'Halil Hoca' ]
console.log( arrQ4.shift()); // removes "15" from the array
```

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

JavaScript allows both ragged and rectangular arrays.

```javascript
let rectangularArray = [[1,2,3], [1,2,3], [1,2,3]];
console.log( rectangularArray); // prints [ [ 1, 2, 3 ], [ 1, 2, 3 ], [ 1, 2, 3 ] ]
let raggedArray = [[1,2,3,], [1,2], [1]];
console.log( raggedArray); // prints [ [ 1, 2, 3 ], [ 1, 2 ], [ 1 ] ]
```

6) Can array objects be initialized?

Array objects can be initialized. List comprehension is also supported, and can be used to create a new array based on an existing one. For example [for (i of [9, 10, 11]) i * i] creates the array [81, 100, 121].

```javascript
let arrQ6_2 = [33, 34, 35]; // creates an array of length 3 and initializes elements
let arrQ6 = Array(3); // creates an array of length 3
console.log( arrQ6); // prints [ <3 empty items> ]
arrQ6[0] = 1;
arrQ6[1] = 2;
arrQ6[2] = 3;
console.log( arrQ6); // prints [ 1, 2, 3 ]
```

7) Are any kind of slices supported?

JavaScript supports array slicing with the slice() method. If supplied with one parameter, the method returns a shallow copy of the array from the given starting index to the end of the original array. If two parameters are supplied, then a copy is returned with the starting index inclusive and ending index exclusive. Negative parameters are also supported.

```javascript
const names = ["Efe", "Emir", "Arda", "Eren"];
console.log( names.slice(2)); // prints [ 'Arda', 'Eren' ]
console.log( names.slice( 1,3)); // prints [ 'Emir', 'Arda' ]
console.log( names.slice( -2)); // prints [ 'Arda', 'Eren' ]
console.log( names.slice(0,7)); // prints [ 'Efe', 'Emir', 'Arda', 'Eren' ]
console.log( names.slice( 2, -1)); // prints [ 'Arda' ]
```

8) Which operators are provided?

The subscript operator is [], it is used to access array elements. The spread operator (...) is used to concatenate or clone arrays. Equality operators ===, !==, == and != are all supported for arrays as well. In addition to these, since JS arrays are objects, all the operators defined on objects (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) will work on arrays as well. For example, the + operator concats arrays as a string, rather than creating a new array.

```
const names1 = ["Efe", "Emir"];
const names2 = ["Arda", "Eren"];

const names3 = [...names1, ...names2]; // concats the two arrays
console.log( names3); // prints [ 'Efe', 'Emir', 'Arda', 'Eren' ]
const namesClone = [...names1]; // clones names1 array
console.log( namesClone) // prints [ 'Efe', 'Emir' ]
names5 = names1 + names2; // doesn't work, names5 becomes a string
console.log( typeof( names5)); // prints string
console.log( names1 === names2); // prints false
console.log( names1 != names2); // prints true
console.log( names1 == names2); // prints false
```

Resources used for JavaScript

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

https://262.ecma-international.org/5.1/#sec-11.2.1

https://stackoverflow.com/questions/27537677/is-a-javascript-array-index-a-string-or-an-integer

https://www.w3schools.com/js/js_arrays.asp

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Indexed_collections

https://www.digitalocean.com/community/tutorials/understanding-arrays-in-javascript

https://www.codingame.com/playgrounds/6181/javascript-arrays---tips-tricks-and-examples

http://cs.boisestate.edu/~alark/cs354/lectures/subscript_binding_multid_arrays.pdf

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

https://dmitripavlutin.com/operations-on-arrays-javascript/

https://www.programiz.com/javascript/online-compiler/

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

PHP

1) What types are legal for subscripts?

Both integers and strings are legal for subscripts. Also, the subscript operator is either [] or {} in PHP, both can be used to access array elements. Additionally, boolean, float and null values can also be used as subcripts since they will be converted to integers or strings.

```
echo "CODE FOR QUESTION 1\n";
$arrQ1 = array( 1, 2, 3, 4, 5);
echo $arrQ1[1]; // prints "2"
echo "\n";
echo $arrQ1{"1"}; // prints "2" as well
echo "\n";
$arrQ1["string"] = "str"; // legal statement
echo $arrQ1["string"]; // prints "str"
echo "\n";
echo sizeof( $arrQ1); // prints 6, unlike JavaScript
echo "\n";
echo $arrQ1[false]; // prints "1", false is evaluated to "0"
echo "\n";
echo $arrQ1[1.3]; // prints "2", casts to integer "1"
echo "\n";
$arrQ1[""] = 69;
echo $arrQ1[null]; // prints 69 because null is cast to empty string ""
echo "\n";
```

2) Are subscripting expressions in element references range checked?

The lowest integer that is accepted as an index is 0 and the **subscripts are range checked**. If an out of bounds index is tried to be reached, PHP throws an error. Similarly, if a string is used to access an index that doesn’t exist, an error is thrown:

```
echo "CODE FOR QUESTION 2\n";
$arrQ2 = array( 3, 4, 5, 6, 7);
// echo $arrQ2[6]; -> gives error "Undefined offset: 6"
// echo $arrQ2['efe']; -> gives error "Undefined index: efe"
$arrQ2["efe"] = 21901548;
echo $arrQ2{2}; // prints "5"
echo "\n";
echo $arrQ2["efe"]; // now prints 21901548
$arrQ2[6] = 5; // this is legal, even though the 5th index doesn’t exist, 6th index
can be initialized but cannot be reached before this initialization
```

3) When are subscript ranges bound?

PHP arrays don’t have a fixed size, and are dynamic. **The subscript ranges are thus bound at runtime**.

```
$arrQ3 = array( 3, 4, 5, 6, 7);
echo $arrQ3[3]; // prints 6
echo "\n";
$arrQ3[] = 31; // adding an element at the end of the array
```

```
echo $arrQ3[5]; // prints 31
echo "\n";
echo sizeof( $arrQ3); // prints 6
echo "\n";
unset( $arrQ3[3]); // removes element at third index, which is 6
echo sizeof( $arrQ3); // prints 5
```

4) When does allocation take place?

An array in PHP doesn't have fixed size, hence **allocation for elements is done at runtime**. PHP has heap-dynamic arrays.

```
$arrQ4 = array();
$arrQ4[] = 1;
$arrQ4[] = 5;
$arrQ4[] = 19;
foreach ( $arrQ4 as $value) {
echo $value;
echo "\n";
} // prints every value that were added to the array
```

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

PHP supports both ragged and rectangular multidimensional arrays.

```
$rectangularArray = array (
  array(1,2,3),
  array(4,5,6),
  array(7,8,9)
  );

var_dump( $rectangularArray);

$raggedArray = array (
  array(1,2,3),
  array(4,5),
  array(7)
  );

var_dump( $raggedArray);
```

6) Can array objects be initialized?

Array objects can be initialized, and even indices that exceed the maximum available index in the array, meaning (size – 1), are able to be set. The array can be initialized in a single line, as displayed below.

```
$arrQ6 = array();
$arrQ6[3] = 5; // legal
var_dump( $arrQ6); /* prints array(1) {
                                [3]=>
                                int(5)
                              } */
```

```php
$arrQ6_2 = array(1,2,3); // initialization
$arrQ6_2[2] = 33; // legal
```

7) Are any kind of slices supported?

Slicing of arrays is supported in PHP with the “array_slice()” function. The first parameter of this function is the array to be sliced, then the starting index and the length of the subarray to be returned.

```php
$arrQ7 = array("efe", "emir", "eren", "arda", "kaan", "duru");
print_r( array_slice( $arrQ7, 2)); // prints from the second index until the last
print_r( array_slice( $arrQ7, 1, 3)); // prints from the 1st index and slices an array of length 3, so [“emir”, “eren”, “arda”]
print_r( array_slice( $arrQ7, -1, 1)); // prints "duru"
```

8) Which operators are provided?

PHP supports the union operator (+), equality operator ==, identity operator ===, inequality operator != or <> and the nonidentity operator !==, for arrays.

From the PHP documentation: “The + operator returns the right-hand array appended to the left-hand array; for keys that exist in both arrays, the elements from the left-hand array will be used, and the matching elements from the right-hand array will be ignored.” (https://www.php.net/manual/en/language.operators.array.php)

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

$c = $a + $b; // union of $a and $b
var_dump( $c);

var_dump( $a == $b); // prints bool(false)
var_dump( $a === $b); // prints bool(false)
var_dump( $a != $b); // prints bool(true)
var_dump( $a <> $b); // prints bool(true), this is the inequality operator, same as !=
var_dump( $a !== $b); // prints bool(true)
```

Resources used for PHP

https://www.w3schools.com/php/phptryit.asp?filename=tryphp_compiler

https://www.upgrad.com/blog/php-array-length/

https://www.php.net/manual/en/language.operators.array.php

https://www.w3schools.com/php/phptryit.asp?filename=tryphp_syntax

https://stackoverflow.com/questions/5560103/in-php-how-do-i-set-the-size-of-an-array https://stackoverflow.com/questions/369602/deleting-an-element-from-an-array-in-php

https://www.php.net/manual/en/function.array-slice.php

https://onecompiler.com/php

Python

1) What types are legal for subscripts?

As indicated in the error message when an invalid subscript is written: “only integers, slices (:), ellipsis (...), numpy.newaxis (None) and integer or boolean arrays are valid indices (subscripts)”.

```python
arrQ1 = np.array( [0, 3, 4, 5]) # creates an array
print( arrQ1[1]) # prints 3
print( arrQ1[1:2:3]) # prints [3] (slice)
print( arrQ1[...]) # prints [0 3 4 5] (ellipsis)
print( arrQ1[None]) # prints [[0 3 4 5]] (None keyword)
arrQ1_2 = np.array( [1,2,3])
print( arrQ1[arrQ1_2]) #prints [3 4 5], the elements at indices 1, 2 and 3, passing an integer array
arrQ1_3 = np.array( [True, False, False, True])
print( arrQ1[arrQ1_3]) # prints [0 5], only elements at first and last indices, passing a boolean array
```

2) Are subscripting expressions in element references range checked?

The subscripting expressions are range checked, and Python throws an error if an out of bounds index is entered between brackets. However, Python supports negative indices, given that they aren’t out of bounds either. For zero and positive integers, the legal indices are in the range [0, n), where n is the length of the array. For negative indices, the legal range is [-n, -1].

```python
arrQ2 = np.array( [0, 3, 4, 5]) # creates an array
print( arrQ2[3]) # prints 5
# print( arrQ2[7]) -> gives error "IndexError: index 7 is out of bounds for axis 0 with size 4"
print( arrQ2[-4]) # prints 0
# arrQ2[9] = 5 -> also gives error, can't initialize the ninth element in an array of size 4
```

3) When are subscript ranges bound?

Numpy arrays are heap dynamic, and the **subscript ranges are bound in runtime**.

```python
arrQ3 = np.array( [31, 32, 33, 34]) # creates an array
print( arrQ3[3]) # prints 34
# print( arrQ3[6]) -> gives an error, can't access 6th index
```

4) When does allocation take place?

As stated in the answer of Q3, ndarrays are heap dynamic, meaning that binding of subscript ranges and storage allocation is dynamic and can change any number of times. **The allocation takes place at runtime**. Ndarrays can be resized with the resize() method (https://numpy.org/doc/stable/reference/generated/numpy.ndarray.resize.html#numpy.ndarray.resize).

```
arrQ4 = np.array( [31, 32, 33, 34]) # creates a heap-dynamic array
print( arrQ4[3]) # prints 34
# allocation is done during runtime
arrQ4[2] = 42 # this is legal
arrQ4.resize(2); # reduces the size of the array to two, in place
print( arrQ4) # prints [31 32]
```

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

Rectangular multidimensional arrays are allowed with ndarrays. When a ragged array is tried to be constructed, the compiler warns that this has been deprecated and a dtype="object" must be specified if a ragged array is to be created. By doing so, a ragged array can also be initiated, however this creates lists rather than arrays. Numpy doesn't support ragged arrays as it is.

```
rectangularArray = np.array( [[31, 32, 33], [40, 42, 43]])
print( rectangularArray) # [[31 32 33]
                         # [40 42 43]]

raggedArray = np.array( [[31, 32], [40], [43, 44, 45]], dtype="object")
print( raggedArray) # prints [list([31, 32]) list([40]) list([43, 44, 45])]
```

6) Can array objects be initialized?

Array objects can be initialized. Array objects can also be initialized with the comprehension syntax, displayed below.

```
arrQ6 = np.array( [1, 2, 3]) # creates and initializes an array
arrQ6_2 = np.array( np.zeros) # creates an array and initializes every element to be
zeroes, printing the array outputs “<built-in function zeros>”

arrQ6_3 = np.array( [x + 1 for x in range(100)]) # creates an array with elements
[1, 100]
print( arrQ6_3)
```

7) Are any kind of slices supported?

Slices are supported for ndarrays. The slicing operators are : and ellipses (...). If an array a is to be sliced, writing a[x:y:z] means slicing the array starting from the xth index up to (y-1)st index, with a step size of z.

```
arrQ7 = np.arange(10) # creates an array [0 1 2 3 4 5 6 7 8 9]
print( arrQ7) # prints [0 1 2 3 4 5 6 7 8 9]
print( arrQ7[3:7]) # prints [3 4 5 6]
print( arrQ7[1:9:2]) # prints [1 3 5 7]

arrQ7_2 = np.arange(24).reshape(2, 3, 4) # a multidimensional array
# [[[ 0  1  2  3]
#   [ 4  5  6  7]
#   [ 8  9 10 11]]
#
# [[12 13 14 15]
#   [16 17 18 19]
#   [20 21 22 23]]]

print( arrQ7_2[:, :, 0]) # this can be done to slice
# [[ 0  4  8]
#   [12 16 20]]

print( arrQ7_2[..., 0]) # gives the same outcome as the above slice code
```

8) Which operators are provided?

Every arithmetic (+, -, *, /, //, %, **, <>, &, ^, |, ~) and comparison (==, < , >, <=, >=, !=) operator is defined on ndarrays, elementwise. For example, when the + operator is called on two arrays, it sums every element at the same index in both arrays and outputs a new array with the summed values. For this, the arrays need to be the same size. If it is called with a number (e.g: arr + 2), then it adds the number to every element in the array and outputs it. Also included are [] and []= operators.

```
arrQ8 = np.arange(10) # creates an array [0 1 2 3 4 5 6 7 8 9]
arr2 = np.arange(10,20) # [10 11 12 13 14 15 16 17 18 19]
print( arrQ8 + arr2) # adds every element elementwise, prints [10 12 14 16 18 20 22
24 26 28]
print( arrQ8 / 2) # prints [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
print( arrQ8 < arr2) # prints [
True  True  True  True  True  True  True  True  True  True]
print( arr2 ** 2) # prints [100 121 144 169 196 225 256 289 324 361]
print( arrQ8 * arr2) # prints [  0  11  24  39  56  75  96 119 144 171]
```

Resources used for Python

https://numpy.org/doc/stable/user/basics.creation.html

https://numpy.org/doc/stable/reference/arrays.indexing.html#basic-slicing-and-indexing

http://cs.boisestate.edu/~alark/cs354/lectures/subscript_binding_multid_arrays.pdf

https://stackoverflow.com/questions/16346506/representing-a-ragged-array-in-numpy-by-padding

https://scipy-lectures.org/intro/numpy/operations.html

https://riptutorial.com/numpy/example/4255/array-operators

https://numpy.org/doc/stable/reference/generated/numpy.ndarray.resize.html#numpy.ndarray.resize

https://onecompiler.com/python/3vdnysc3a

Rust

1) What types are legal for subscripts?

In the Rust language, only unsigned integers are legal subscripts.

```rust
let arr = [1, 2, 3, 4, 5];
// arr["efe"] = 5; -> gives an error
println!( "{}", arr[4]); // prints "5"
```

2) Are subscripting expressions in element references range checked?

Subscripting expressions are range checked, and Rust gives an error if an index that is out of bounds is tried to be reached.

```rust
let arr_q2 = [1, 2, 3, 4, 5];
println!( "{}", arr_q2[3]) // legal
// println!( "{}", arr_q2[7]); -> "index out of bounds: the length is 5 but the
index is 7"
// println!( "{}", arr_q2[-3]); -> also erroneous, doesn't support negative indices
```

3) When are subscript ranges bound?

Arrays in rust are static. **Rust arrays' subscript ranges are bound at compile time**, they are also stack allocated. Also, all arrays are immutable if they don't have the "mut" keyword in front of them when being declared.

```rust
let arr_q3 = [1, 2, 3, 4, 5]; // subscripts are bound at compile time
println!( "{}", arr_q3[3]); // legal and prints 4
// println!( "{}", arr_q3[7]); -> "index out of bounds: the length is 5 but the
index is 7"

let arr3:[i32;3] = [0;3]; // the size of the array needs to be specified, subscripts
are bound
println!( "The array is: {:?}", arr3); // prints: "The array is: [0, 0, 0]"
```

4) When does allocation take place?

Rust arrays are stack allocated and the **allocation takes place at compile time**. The size of the arrays, or the non-negative compile time constant (https://doc.rust-lang.org/std/primitive.array.html), needs to be specified whenever creating an array so space can be allocated in the stack at compile time. Size cannot be changed at runtime.

```rust
let arr_q4 = [6, 7, 8]; // array without data type but fixed size and initialized
println!( "The array is: {:?}", arr_q4); // prints The array is: [6, 7, 8]

let arr_q4_2:[i32;3] = [9, 10, 11]; // array with data type and again, fixed size
println!( "The array is: {:?}", arr_q4_2); // prints The array is: [9, 10, 11]

```

```rust
let arr_q4_3:[i32;3] = [0;3]; // array with fixed size but initialized with default values
// the size of an array needs to be specified when creating it
println!( "The array is: {:?}", arr_q4_3); // prints The array is: [0, 0, 0]

let mut arr_q4_4:[i32;3] = [0;3]; // additionally, an array whose elements are mutable but still has fixed size
println!( "The array is: {:?}", arr_q4_4); // prints The array is: [0, 0, 0]
arr_q4_4[2] = 5; // legal
println!( "The array is: {:?}", arr_q4_4); // prints The array is: [0, 0, 5]
```

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

Rust allows rectangular multidimensional arrays, however ragged arrays aren't allowed.

```rust
let rectangular_array:[[i32;3];2] = [[31;3];2];
println!( "The array is: {:?}", rectangular_array); // prints: "The array is: [[31, 31, 31], [31, 31, 31]]"

// let ragged_array = [[1, 2, 3], [1]]; -> not allowed
```

6) Can array objects be initialized?

Array objects can be initialized when declared.

```rust
let arr_q6 = [1, 2, 3]; // this is legal, and one of the correct ways of initializing an array
println!( "The array is: {:?}", arr_q6); // prints [1, 2, 3]

let arr_q6_2:[i32;3] = [1,2,3]; // another legal way of initializing
println!( "The array is: {:?}", arr_q6_2); // prints [1, 2, 3] again

let arr_q6_3 = [0;3]; // initializes every value to 0, no type specified
println!( "The array is: {:?}", arr_q6_3); // prints [0, 0, 0]

let arr_q6_4:[i32;3] = [0;3]; // initializes every value to 0, type is 32 bit integer
println!( "The array is: {:?}", arr_q6_4); // prints [0, 0, 0]
```

7) Are any kind of slices supported?

Rust offers slices, and the documentation states that "Slices let you reference a contiguous sequence of elements in a collection rather than the whole collection" (https://doc.rust-lang.org/book/ch04-03-slices.html).

```rust
let arr_q7 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
println!( "The array is: {:?}", arr_q7); // prints: "The array is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]"

let slice1 = &arr_q7[2..5]; // first index is inclusive, ending is exclusive
println!( "The slice is: {:?}", slice1); // prints: "The slice is: [3, 4, 5]"

```

```
let slice2 = &arr_q7[..3]; // ending index is exclusive, starts from the beginning of the array
println!( "The slice is: {:?}", slice2); // prints: The slice is: [1, 2, 3]

let slice3 = &arr_q7[4..]; // start index is inclusive, slices until the end
println!( "The slice is: {:?}", slice3); // prints: The slice is: [5, 6, 7, 8, 9, 10]
```

8) Which operators are provided?

The subscript operator in Rust is [], this can be used to access array elements. Other operators supported by Rust are the [A;B] operator, which is used to specify the type of an array and the size of the array or the default values to fill the array with and the size, the []= operator to change an element of the array, and the &[] operator to create a slice. In addition, Rust supports the == and != for array comparisons, as well as >, >=, < and <= operators.

```
let mut arr_q8_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
arr_q8_1[5] = 39; // []= operator
let arr_q8_2:[i32;3] = [33;3]; // [A;B] operator
println!( "The array is: {:?}", arr_q8_2); // prints: The array is: [33, 33, 33]
let slice = &arr_q8_1[3..7]; // &[] operator for slicing
println!( "The slice is: {:?}", slice); // prints: The slice is: [4, 5, 39, 7]

arr_q8_3 = [1, 2, 3, 5, 5, 6, 7, 8, 9, 10]; // only difference between this array and arr_q8_1 is the "5" at the 3rd index
// the relational operators compare two arrays depending on the first different character in both arrays
// since 5 > 4, arr_q8_3 > arr_q8_1 and arr_q8_3 >= arr_q8_1 both evaluate to "true" down below
println!( "{}", arr_q8_1 == arr_q8_3); // == operator, prints false
println!( "{}", arr_q8_1 != arr_q8_3); // != operator, prints true
println!( "{}", arr_q8_1 < arr_q8_3); // prints true
println!( "{}", arr_q8_1 <= arr_q8_3); // prints true
println!( "{}", arr_q8_1 > arr_q8_3); // prints false
println!( "{}", arr_q8_1 >= arr_q8_3); // prints false
```

Resources used for Rust

https://play.rust-lang.org/

https://www.educative.io/edpresso/arrays-in-rust

https://www.rust-lang.org/

https://stackoverflow.com/questions/30253422/how-to-print-structs-and-arrays

https://turreta.com/2019/09/08/rust-how-to-create-two-dimensional-array-example/

https://docs.rs/jagged_array/0.2.4/jagged_array/

https://subscription.packtpub.com/book/application-development/9781788390637/1/ch01lvl1sec24/arrays

https://doc.rust-lang.org/rust-by-example/primitives/array.html

Best Language for Array Operations

After having experimented with all of the 5 languages, I believe Dart is the best language for array operations. I think this is so, because first of all Dart offers only integer subscripts for arrays, and this makes the language more reliable and readable compared to JavaScript or PHP, where arrays can have different subscripts as well, such as strings, booleans and even objects. Following this, Dart supports range checking for subscripting expressions in element references, which makes it much more reliable than JavaScript where no error is thrown if an index that is out of bounds is tried to be reached. Also, unlike Rust where arrays have fixed sizes, Dart offers both fixed and growable arrays, which may make it less readable due to the lack of apparent distinction between these two array types, but also makes it better writable because of the variety of arrays it offers compared to Rust, which in my opinion is more important. Dart has compile time constant arrays, fixed arrays, whose sizes are fixed but elements are alterable, and growable arrays, whose sizes and elements can be changed at runtime. The availability of all of these different constructs make Dart more preferable for me amongst all 5 languages. Unlike Python's ndarray, Dart supports ragged arrays as well as multidimensional arrays and has better functionality. Additionally, Python's lack of type names makes it less readable and reliable, while Dart offers some types to differentiate between objects and is more reliable compared to Python in this regard. All of these languages offer slicing and initializing of array objects, so Dart isn't very different in these areas. The operators for arrays that Dart provide are also simple and useful enough, making it a better choice than Python for example, where there are a lot of array operators and thus more confusion. All in all, after taking into consideration every trade-off and functionality, Dart prevails to be the best language for array operations.

Learning Strategy

For all of the languages, I first conducted a search on the internet to find relevant sources for the array constructs of those languages. After identifying necessary sources, I found an online compiler for every language to try and run my code while learning how to construct and operate on arrays. I have answered every question for a specific language first, then moved on to the next language, rather than answer every single question for all of the languages and proceed to the next question.

Dart:

I used the online compiler "https://dartpad.dev/?null_safety=true" for writing Dart code. I first had to understand arrays in Dart before starting the task, and utilized this source: "https://www.tutorialspoint.com/dart_programming/dart_programming_lists.htm" which thoroughly explains the usage of arrays in this language. For answering the first question I experimented myself first, and then searched on the internet to understand how subscripts work in Dart and which types are legal. I again experimented with the code to see if the language is range checked or if multidimensional/ragged arrays are supported. Consulting this blog post "https://medium.com/flutter-community/working-with-multi-dimensional-list-in-dart-78ff332430a" allowed me to confirm my experiments about multidimensional lists in Dart. For slicing, "https://discoverflutter.com/how-to-slice-a-list-in-dart/" the guide that can be found at this link helped me understand how to use the relevant method to slice an array. Finally, I read the official documentation (https://dart.dev/guides) all the way through to find the most accurate information about Dart language constructs.

JavaScript:

For the JS part of the homework, I both did research on the internet and used my prior knowledge of this language. I used the compiler at this link: https://www.programiz.com/javascript/online-compiler/. For the first question, I already knew that JS arrays accepted every data type available in the language as subscripts, however confirmed this knowledge with the research I have done. I also learned through the official documentation (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) that subscripts are coerced into strings by the JavaScript engine. Also by trial I realized that subscripts are range checked, however JS doesn't throw an error when an out of bounds index is reached. From this StackOverflow discussion "https://stackoverflow.com/questions/27537677/is-a-javascript-array-index-a-string-or-an-integer" I learnt that the highest valid subscript integer for JS arrays is 2^32-1. From my own experiments I could see that both multidimensional and ragged arrays are supported, and also array objects can be initialized. I learnt about slicing from this link https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice and array operators from this one: https://dmitripavlutin.com/operations-on-arrays-javascript/.

PHP:

I used the compiler at this link https://onecompiler.com/php for writing and testing my PHP code. The official documentation https://www.php.net/manual/en/ let me realize that PHP arrays accept both integers and strings as subscript types. I experimented with the code to see if subscripts are range checked or not. I researched on the internet to see if PHP arrays have fixed sizes, and found out that they can grow and shrink, hence are dynamic (https://stackoverflow.com/questions/369602/deleting-an-element-from-an-array-in-php). My own experiments showed me that both ragged and multidimensional arrays are supported and that array objects can be initialized. I read about slicing at https://www.php.net/manual/en/function.array-slice.php and operators at https://www.php.net/manual/en/language.operators.array.php from the official documentation.

Python:

I used the compiler at https://onecompiler.com/python/3vdnysc3a to compile Python code by using the numpy library for ndarrays. When experimenting with the code, I received the error message "only integers, slices (:), ellipsis (...), numpy.newaxis (None) and integer or boolean arrays are valid indices", which showed me the legal subscript types for ndarrays. My experiments and research yielded the result that ndarrays are range checked, however negative indices are accepted. I found out through this link https://numpy.org/doc/stable/reference/generated/numpy.ndarray.resize.html#numpy.ndarray.resize that ndarrays are resizable, thus are dynamic and can grow. From this link https://stackoverflow.com/questions/16346506/representing-a-ragged-array-in-numpy-by-padding and my experiments, I saw that although ndarrays support multidimensional arrays, support for ragged arrays has been deprecated. Throughout all my research, I saw that array elements can be initialized at declaration. I learned about slicing and available operators on the internet through these links: https://riptutorial.com/numpy/example/4255/array-operators and https://numpy.org/doc/stable/reference/arrays.indexing.html#basic-slicing-and-indexing.

Rust:

I used this https://play.rust-lang.org/ compiler for running my Rust code. I first experimented with the code to see if different types would be accepted as subscripts, however the results of my experiments and this link https://www.educative.io/edpresso/arrays-in-rust have shown me that only integers are valid subscripts in Rust. Again, my own experiments showed me that subscripts are range checked. The official documentation at https://doc.rust-lang.org/std/primitive.array.html helped me understand that Rust arrays are static and have defined types. I learned from https://turreta.com/2019/09/08/rust-how-to-create-two-dimensional-array-example/ this link how to create rectangular multidimensional arrays and from this one https://docs.rs/jagged_array/0.2.4/jagged_array/ that primitive arrays in Rust don't support jagged arrays. From the official documentation I could observe that an array is supposed to be initialized when it is declared, and my own experiments also corroborated this, since the compiler didn't let me initialize the elements of an array after first declaring it. From this link https://doc.rust-lang.org/rust-by-example/primitives/array.html I learned about slicing, and from the official documentation I learned about available array operators.