Programing Languages Project

Group 11			
#	**Name**	**ID**	**Section**
1	Razan Arif Alamri (Leader)		IA
2	Dona Mamdouh Ali		
3	Wajd Bandar Yousef Alharbi		DAR
4	Shatha Khaled Binmahfouz		IAR
5	Joud Fayez Shaher Alahmadi		
6	Kawther Kamal Aldhaheri		IAR

Instructor: Dr. Rania Molla

Table of Contents

1. Introduction 4

2. Historical Overview of Dart 4

3. Implementation Methods 5

3.1 Compilation 5

3.2 Pure Interpretation 7

4. Domain of Dart 7

5. Using BNF or EBNF for Dart 8

5.1 Dart Identifier 8

5.2 Dart Variable Declaration 9

5.3 Dart Expressions 10

5.3.1 Arithmetic Operators 10

5.3.2 Relational Expressions 11

5.3.3 Logical Operators 12

5.3.4 Assignment Operators 13

5.3.5 Short-circuit Operators (&& and ||) 14

5.4 Dart Selection (2-way if else) 15

5.5 Dart Loop 16

6.The Evaluation of Dart 17

6.1 Readability 17

6.2 Writability 21

6.3 Reliability 22

7. Types of Variables 24

8. Array Types 27

9. Types of Scopes 27

10. Program (Code and Run) 29

10.1 Code in Dart 29

10.2 Sample Output 30

11. Conclusion 31

12. References 32

Illustrations

List of Figures:

Figure 1: Flow diagram of if else statement 15

Figure 2: Flow diagram of while statement.....16

Figure 3: Conditional spread operator in Dart.....18

Figure 4: Collection if in Dart.....18

Figure 5: Collection for in Dart. 19

Figure 6: Parameters Name in Dart.....19

Figure 7: Program code.....29

Figure 8: Screenshot of output 30

Table 1: Valid and invalid identifier.....8

Table 2: Arithmetic Operators 10

Table 3: Relational Expressions 11

Table 4: Logical Operators 12

Table 5: Assignment Operators.....13

Table 6: Dart exceptions.....23

1. Introduction:

Dart is an open source, modern, and easy-to-learn programming language. It is used in web, mobile and desktop application development (Answersje, 2021) .The Dart language can run on many operating systems such as Windows, MacOS, Linux, and others, which means that it is a cross-platform programming language. Also, its syntax is similar to that of the C programming language (Answersje, 2021). In Dart, the code is compiled and converted to JavaScript or to the native language (Walrath, 2012).

In this report, we will introduce a brief history of Dart, understand the language's implementation method and domains, present Dart's (BNF or EBNF) in detail while elaborating on the types of variables, arrays, and scopes that Dart uses. We will also evaluate the language Based on what we learned in CPCS-301 and by using supported facts.

2. Historical Overview of Dart:

Dart is a programming language created by Google and aimed at web developers. It was created by Lars Bak & Kasper Lund and was first revealed at the GOTO conference in Denmark on October 10, 2011 (Cleverism, 2016).

The goal of creating Dart was to solve problems in JavaScript such as the performance of programs, make them run faster, and protect against the danger of programming through the website (Walrath, 2012).

The first version of Dart 1.0 was released on November 14, 2013 (Bak, 2013). In the beginnings of Dart, opinions were divided between positive and negative due to Dart plans to integrate Dart Virtual Machine into Google Chrome and web fragmenting. On March 25, 2015, Dart announced the new version 1.9, which includes not integrating the Dart VM into Google Chrome and focusing on compiling Dart with JavaScript, to make it easier and faster for developers to focus on a single way to build their apps and increase productivity (Bak & Lund, Dart for the Entire Web, 2015).

Updates to the Dart language continued, and new versions were released, most notably:

- In August 2018, Dart 2.0 was released, which supports a new sound type system (Dart, 2022).
- Dart 2.1 supports converting int to double (Dart, 2022).
- Dart 2.2 supports the literal list (Dart, 2022).
- A new tool has been added to Dart 2.6 that allows compilation of Dart to native executables (dart2native) (Dart, 2022).
- Dart 2.7 supports extension methods (Dart, 2022).
- Dart 2.12 has added a feature that does not allow a variable's value to be null unless otherwise permitted (Dart, 2022).
- Dart 2.14 The triple-shift feature has been added, and some restrictions on type arguments have been removed (Dart, 2022).

The latest version of Dart, which is Dart 2.16 was released on February 3, 2022 (Thomsen, 2022).

3. Implementation Methods

The ways in which a computer program is executed are known as programming language implementation methods. There are three general methods of implementation:

- Compilation.
- Interpretation.
- Hybrid implementation systems.

And for our programming language we will explain the compilation and interpretation because it’s the most suitable methods for our program:

3.1. Compilation

Compilation is converting one programming language to machine. Traditionally, this refers to combining multiple input files into a single file that is runnable on the target system.

Also, the compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.

There are multiple phases to the compilation process:

1. **Lexical Analysis:**

   The scanner's first phase serves as a text scanner. This phase examines the source code as a stream of characters and turns them into lexemes that make sense. These lexemes are represented by tokens in the lexical analyzer.

2. **Syntax Analysis:**

   Syntax analysis follows the Lexical analysis phase in the Compiler Design process. Because the syntax analyzer transforms lexical units into parse trees, it's also known as the Parse Tree.
   Syntax analysis is the process of checking a given input string for confirmation of formal grammar rules and structure. It analyzes the syntactical structure and determines whether the given input conforms to the programming language's syntax. (Guru99, 2022)

3. **Semantics analysis:**

   The third phase of Compiler is Semantic Analysis. Semantic analysis checks the semantic integrity of program declarations and statements. It's a collection of processes that the parser calls as and when the grammar requires it. The given code is checked for consistency using both the previous phase's syntax tree and the symbol table. The compiler ensures that each operator has matching operands during type checking, which is a crucial aspect of semantic analysis. (sachiniyermalkapur, 2020)

4. **Code generation:**

   The machine code is generated.

3.2. Pure Interpretation

A pure interpreter reads a program's source text, analyzes it, and executes it in real time. The interpreter requires more spaces and spends a lot of time analyzing strings of characters to figure out what they mean, therefore this is usually very slow. Each time an expression in the source text is encountered, a pure interpreter must recognize and analyze it in determining what to do next.

4. Domain of Dart

Dart is used in conjunction with Flutter to create mobile apps. This is one of Dart's most popular applications nowadays. The fact that Dart and Flutter are cross-platform is a significant advantage. It implies that instead of creating two distinct apps for iPhone and Android, you can create an app with simply one code base.

Dart is generally used to create software for internet-connected devices (such as smartphones, tablets, and laptops) as well as servers. Until recently, it was impractical for novice programmers to attempt to create mobile or web-based applications on their own.

Furthermore, Dart programming's primary goal is to construct frontend user interfaces for online and mobile apps. It's under active development, compiled to native machine code for constructing mobile apps, and strongly typed. It's influenced by other programming languages like Java, JavaScript, and C#. Because Dart is a compiled language, you can't just run your code; rather, the compiler parses it and converts it to machine code.

5. Using BNF or EBNF for Dart

5.1 identifier

In every programming language program elements such as variables, methods, and classes must have an identifier. Identifiers are names given to each of these program elements to make access to these elements easy. It is recommended that the identifier is readable and meaningful. There are different rules to write the identifiers in every different language (Busbee & Braunschweig, n.d.). The rules that we should follow to define an identifies in dart are:

- Must start with alphabet (upper or lower) or underscore.
- Cannot begin with a digit.
- Cannot use a keyword as an identifier.
- Underscore (_) and dollar sign($) are the only allowed special characters.
- Two successive underscores are not allowed.
- Can not contain any whitespace.
- Identifiers are case sensitives.

The following table contains examples of the valid and invalid identifiers depending on the above rules. For example, the first row is an instance of the rule that says it is not allowed to write two successive underscores. The second row shows that whitespace is not allowed between identifier's characters. An example of the rule that says every identifier must start either alphabet or underscore and should not begin with a digit is '1resullt'. To fix this invalid identifier to a valid identifier move the '1' to the rightmost 'result1'.

The table also represents four examples that describe the rule that does not allow any special character except underscore and dollar sign, which is 'first-name' and '@var' which are invalid identifiers, 'First_name' and '$count' are considered valid identifiers.

Valid Identifiers	Invalid Identifiers
firstname	__firstname
firstName	first name
var1	V5ar
$count	first-name
_firstname	1result
First_name	@var

Table 1: valid and invalid identifiers

5.2 Variable Declaration

Variables are names that refer to a memory location that stores data. So, access to the stored data is done by the variable name (Dart Variables, 2021).In Dart before a variable is used, it must be declared (dart variable, n.d.). We can declare a variable in dart in any of these different ways:

1. **Start with a data type:**

   Syntax:

   < type><var_name>= <value>;

   OR

   < type><var_name> = <expression>;

   OR

   < type><var_name>;

   Example:

   int num = 25;

   int is a data type, num is a variable name and 25 is a value.

2. **Use var keyword instead of data type:**

   Syntsx:

   var <var_name>;

   OR

   var <var_name> = <expression>;

   Example:

   var number;

   var name = "Dart";

3. **Dart also allows to declare more than one variable with the same data type in one single line but they should separated by commas:**

   Syntax :

   <type><var1,var2....varN>;

   Example:

   int i,j,k;

5.3 Expressions:

An expression is a part of Dart code that can be evaluated at run time. Every expression is a combination of operators and operations. In this section we will discuss many expressions available in Dart and their BNF (Tutorialspoint, 2015).

5.3.1 Arithmetic Operators

Operators	Meaning
+	Add
–	Subtract
-expr	Unary minus, also known as negation (reverse the sign of the expression)
*	Multiply
/	Divide
~/	Divide, returning an integer result
%	Get the remainder of an integer division (modulo)
++	Increment
--	Decrement

Table 2: Arithmetic Operators

The BNF for arithmetic expressions:

<expression> : : = <term><arithmetic_operator><term> | <arithmetic_operator><term> | <term><arithmetic_operator

<term> : : = <identifier>| <constant> | <expression>

<arithmetic_operator> : : = + | - | * | / | ~/ | % | ++ | --

5.3.2 Relational Expressions

Operator	Description
>	Greater than
<	Lesser than
>=	Greater than or equal to
<=	Lesser than or equal to
==	Equality
!=	Not equal

Table 3: Relational Expressions

The BNF for Relational expressions:

<expression> : : = <term>< Relational_operator><term>

<term> : : = <identifier>|<constant>|<expression>

<Relational _operator> : : = > | < | >= | <= | == | !=

5.3.3 Logical Operators

Operator	Description
&&	**And** − The operator returns true if and only if all the expressions specified return true
||	**OR** − The operator returns true if at least one of the expressions specified return true
!	**NOT** − The operator returns the inverse of the expression's result.

Table 4: Logical Operators

The BNF for Logical expressions:

<expression> : : = <term>< Logical _operator><term> | < Logical _operator><term>

<term> : : = <identifier>|<constant>|<expression>

<Logical _operator> : : = && | || | !

5.3.4 Assignment Operators

Operator	Description
=	**Simple Assignment** Assigns values from the right-side operand to the left side operand
??=	Assign the value only if the variable is null
+=	**Add and Assignment** It adds the right operand to the left operand and assigns the result to the left operand.
—=	**Subtract and Assignment** It subtracts the right operand from the left operand and assigns the result to the left operand.
*=	**Multiply and Assignment** It multiplies the right operand with the left operand and assigns the result to the left operand.
/=	**Divide and Assignment** It divides the left operand with the right operand and assigns the result to the left operand.

Table 5: Assignment Operators

The BNF for Assignment Operators:

<expression> : : = <term>< Assignment _operator><term>

<term> : : = <identifier>|<constant>|<expression>

<Assignment _operator> : := **??=** | = | **+=** | **—=** | ***=** | **/=**

5.3.5 Short-circuit Operators (&& and ||):

The **&&** and || operators are used to combine expressions.

The && operator returns true only when both the conditions return true. So, if the first expression returns false, then the operator skips the second expression and returns false.

The || operator returns true if one of the expressions returns true. So, if the first expression returns true, then the || operator skips the subsequent expression and returns true.

5.4 Selection (2-way if statement):

The Selection statements must have Boolean expression to specify certain conditions and the statement is executed when the condition is true (Tutorialspoint, 2015).

There are three forms of Selection Statements in Dart:

1. If statement (1-way if)
2. If - else Statement (2-way if)
3. If - else if – else Statement

The syntax of if-else statement:

```
if(boolean_expression) {
   // the block will execute if the Boolean expression is true.
} else {
   // the block will execute if the Boolean expression is false.
}
```

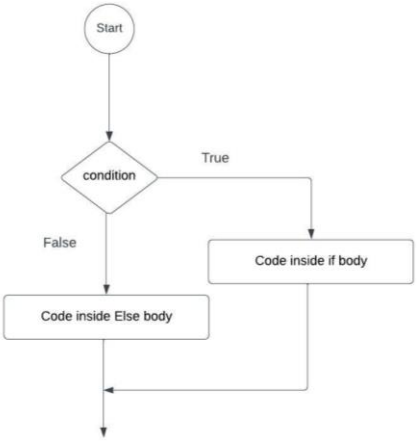

Figure 1: Flow diagram of if else statement

If the Boolean expression evaluates to be true, then the block of the if statement will be executed. If Boolean expression evaluates to be false, then the block of the else statement will be executed (Bracha, 2016).

The BNF:

<if-statement>::= if <expression> then <statement> else <statement>

5.5 Loop:

There are some instructions need repeated execution. Loops are the perfect choice. A loop represents a set of instructions that must be repeated, and it will execute if the condition true (Tutorials point, 2015).

There are three forms of Loop in Dart:

1. For loop
2. While loop
3. Do while loop

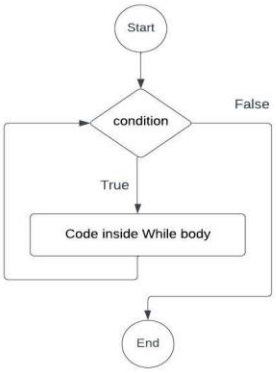

Figure 2: Flow diagram of while

The syntax of while loop:

```
while (expression) {
    // the block will execute if expression is true
}
```

The **while** loop will be executed each time the condition is true. If the condition is false, it will end the execution of while loop (Bracha, 2016).

The BNF:

<while loop> ::= while (<condition>) <statement

6. The Evaluation Of Dart:

Dart is built for developer efficiency and is one of the most popular Over programming languages according to the Stack Flow poll. It is a hybrid of Java and JavaScript.

Dart's evolution explained by the three criteria:

1. Readability
2. Writability
3. Reliability

6.1 Readability:

- **Simplicity:**

Dart is a basic, object-oriented programming language with more structure than JavaScript, on which it is heavily based. It provides strong typing and can be compiled into JavaScript, allowing it to operate anywhere JavaScript can, which includes the web, mobile devices, and servers.

Dart provides a lot of conveniences and syntax for developers to use as a modern programming language. As a result, programmers can frequently use operator overloading to provide easily legible code.

However, nothing can be overloaded in dart. Because operator and function overloading are not supported in Dart. This means you can't have two functions with the same name but distinct parameters in the same class.

- **Orthogonality:**

There are 3 main Dart Language Constructs:

1- Conditional Spread Operator:

This is undoubtedly my favorite and most elegant, despite the fact that it is now only accessible in Dart 2.90. When you want to spread an iterable that could be undefined, you use the conditional spread operator. It's essentially chaining for an iterable that you want to distribute. Consider the following two examples, each with [1, 2, 5] and [1, 2, 5, 1, 7] as outputs:

```dart
void main() {
  print(concatLists([1, 2, 5], null));
  print(concatLists([1, 2, 5], [1, 7]));
}

List<int> concatLists(List<int> arr, List<int>? arr2) {
   return [...arr, ...?arr2];
}
```

Figure 3: Conditional spread operator in Dart.

2- Collection if and Collection for:

After so many years of programming in Java and JavaScript, collection if and collection for seem strange at first. However, they are quite readable and make a lot of sense. Let's look at an example of how an array might be used for navigation:

```dart
void main() {
  const promoActive = true;
  var nav = ['Home',
             'Furniture',
             'Plants', if (promoActive) 'Outlet'
            ];
}
```

Figure 4: Collection if in Dart

.You may use the collection for to insert values into a list:

```dart
void main() {
  var listOfInts = [1, 2, 3];
  var listOfStrings = ['#0',
                       for (var i in listOfInts) '#$i',
                       '#4'];
}
```

Figure 5: Collection for in Dart

3- Named Parameters

Have you ever examined a piece of code, particularly a function call, and wondered what the parameters being given in do? This can be difficult, especially when using magic numbers or strings in an older codebase, and often necessitates hovering over the function to see the signature in order to understand the passed parameters.

The advantages extend beyond that. You can have optional named parameters at the beginning of your signature, for example, and so won't be constrained to placing your function's optional arguments at the conclusion.

Named arguments are not available in JavaScript or TypeScript. Below is an example that demonstrates the ability to make code more understandable. Dart makes extensive use of named parameters (you have definitely seen many examples if you have used Flutter). They make your widget constructors seem nice and easy to read. Without knowing the signature of the functions or constructors, you'll know exactly what the numbers you're sending to the widgets signify.

```dart
void main() {
  final box = new SizedBox(100, 100, 20);
  final boxNamed = new SizedBoxNamed(width: 100, height: 100, padding: 20);
}

class SizedBox {
  SizedBox(int width, int height, int padding);
}

class SizedBoxNamed {
  SizedBoxNamed({int width = 50, int height = 50, int padding = 0});
}
```

Figure 6: Parameters Name in Dart

- **Data Type:**

The following kinds are supported by the Dart language:

- Numbers
- Strings
- Booleans
- Lists
- Maps

1- Numbers:

Numeric literals in Dart are represented by numbers. There are two types of Number Darts:

- Integer − numeric numbers without a decimal point, are represented as integer values. For instance, the number "ten" is an integer. The int keyword is used to represent integer literals.
- Double − values with decimal points, are likewise supported by Double Dart. A 64-bit (double-precision) floating-point number is represented by the Double data type in Dart. Consider the value "10.10." Floating point literals are represented by the keyword double.

2- Strings

A string is a collection of characters. If you want to store data like a name or an address, for example, the string data type should be utilized. A UTF-16 code unit is used to create a Dart string. A sequence of UTF-32 code units is represented by runes.

String literals are represented by the keyword String. Single or double quotes are used to enclose string values.

3- Boolean

True and false are represented by the Boolean data type. The bool keyword in Dart is used to represent a Boolean value.

4- List and Map

The data types list and map are used to represent a collection of objects. In several programming languages, the concept of an array is equivalent to the List data type in Dart. A set of values is represented by the Map data type as key-value pairs. The predefined List and Map classes in the dart core library allow for the generation and management of these collections.

5- The Dynamic Type

Dart is a language with optional typing. The type of a variable is dynamic if the type is not explicitly specified. The dynamic keyword can also be explicitly used as a type annotation.

6.2 Writability:

Abstract Dart Classes:

Dart classes with one or more abstract methods are known as abstract classes. Abstraction is a type of data encapsulation in which the users are not aware of the function's internal workings. They only interface with external software. The abstract class can be declared using the abstract keyword. It's possible that an abstract class doesn't contain any abstract methods.

Methods that are declared but not implemented are known as abstract methods. With implementation, the concrete or normal methods are declared. Both sorts of methods can be found in an abstract class, but abstract methods cannot be found in a regular class.

An abstract class cannot be instantiated since its instance cannot be created. It can only be expanded by a subclass, and the subclass must be given access to the abstract methods that exist in the current class. Then you must declare an abstract subclass.

6.3 Reliability:

Activating Checked Mode:

- Checked Mode: Dart programs execute in two modes:
- Mode of Production (Default)

During development and testing, it is advised to run the Dart VM in checked mode, as this includes warnings and errors to aid the development and debugging process. The checked mode imposes several checks, such as type-checking and so on. While running the script, add the -c or –-checked option before the script-file name to enable checked mode. However, it is advised that the script be executed in production mode to get the best performance.

Consider the following Test.dart script file :

```
void main() {
   int n = "hello";
   print(n);
}
```

Run the script by entering :

```
dart Test.dart
```

Despite the type mismatch, the script runs correctly because the checked mode is disabled. The following is the output of the script:

```
hello
```

Try running the script now with the "- - checked" or the "-c" option :

```
dart -c Test.dart
```

Or,

```
dart - - checked Test.dart
```

There is a type mismatch, and the Dart VM will give an error.

```
Unhandled exception:
type 'String' is not a subtype of type 'int' of 'n' where
  String is from dart:core
  int is from dart:core
#0  main (file:///C:/Users/Administrator/Desktop/test.dart:3:9)
#1  _startIsolate.<anonymous closure> (dart:isolate-patch/isolate_patch.dart :261)
#2  _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:148)
```

Exception:

In Dart, every exception is a subclass of the predefined Exception class. To avoid the program closing unexpectedly, exceptions must be handled.

Sr.No	Exceptions & Description
1	**DeferredLoadException** Thrown when a deferred library fails to load.
2	**FormatException** Exception thrown when a string or some other data does not have an expected format and cannot be parsed or processed.
3	**IntegerDivisionByZeroException** Thrown when a number is divided by zero.
4	**IOException** Base class for all Inupt-Output related exceptions.
5	**IsolateSpawnException** Thrown when an isolate cannot be created.
6	**Timeout** Thrown when a scheduled timeout happens while waiting for an async result.

Table 6: Dart exceptions

7. Types of variables:

A variable is a named space in memory where values are stored. In other words, it serves as a holder for program values. Identifiers are the names for variable names. The guidelines for naming an identifier are:

• It is not allowed for Identifiers to be Keywords.

• Alphabets and numerals can both be used in identifiers.

• Except for the underscore (_) and the dollar ($) sign, identifiers cannot contain spaces or special characters.

• A number can’t be the first character in a variable name.

• A single statement can define several variables. However, commas should be used to separate them.

• To assign values to variables, use the assignment operator (=).

Type Syntax:

Before a variable can be utilized, it must first be declared. in Dart this is done using the ‘var’ keyword. Rather than storing the value, all variables in Dart save a reference to it. The initial value of all uninitialized variables is null, because Dart treats all values as objects.

• The dynamic keyword:

Variables that don't have a static type are assumed to be dynamic. In addition to the var keyword, the dynamic keyword can be used to declare variables.

• Final and Const:

1. Constants are declared using the final and const keywords. Dart makes it impossible to change the values of variables defined using the final or const keywords. These keywords can be used in place of or in addition to the var keyword, depending on the data type of the variable.

2. A compile-time constant is represented by the const keyword. Variables specified with the const keyword are considered final by default.

Here are the Data Types which are built-in Dart:

• Numbers (Integer and Double):

Integer: When whole integers must be held in a variable, an integer data type is needed. It may hold either unsigned or signed values (only positive numbers) (both

positive and negative numbers). The Integer data type has a value range of -263 to 263 that it can carry. In Dart, the keyword int is used to declare integers.

Double: Double precision (numbers with many decimal points) numbers, often known as floating point numbers, are stored in this format.

Using The keyword “double” declares double values.

• Boolean:

The Boolean Data Type is used to represent and store boolean values (0 and 1). True indicates 1 and False implies 0. In decision-making statements, the Boolean Data type is commonly employed. The keyword bool is used to declare Boolean values.

• String:

A string is a collection of characters, numbers, and special characters. In Dart, it can be represented with either single or double quotations. The string data type can store character sequences, special characters, and numbers. The string keyword in Dart can be used to specify a data type that is a string.

• Maps:

A map is a data structure that stores a set of values as key or value pairs of any type. However, each key can only appear once, but its value can be used several times. [] can be used to declare maps, and [] can be used to retrieve the values.

• Lists:

Arrays are represented in Dart as list objects. A list is a data type for storing an ordered group of elements. The [] brackets can be used to declare a list, and the values must be separated by commas,.

• Symbols:

In a Dart program, symbols are used to refer to an identifier declared. In APIs that refer to identifiers by name, symbols are frequently utilized.

and the identifier name can be used to declare a symbol.

8. Array Types:

An array is a highly popular collection in programming. Arrays are represented in Dart as Lists. A list is essentially a collection of objects that are arranged in a certain order; an orderly manner. The index number of each list element, which always begins at zero, can be used to access it. The List class in the dart:core library allows you to create and manipulate lists.

Lists can be categorized into:

- Fixed Length List: A fixed length list constructs a list of the specified length that cannot be expanded or contracted at runtime. An exception will be thrown if you try to resize the list.

- Growable List: the list's length can be modified at run-time. No exception will be thrown if you try to resize the list.

9. Types of Scopes:

The scope of a variable or function is the area in which it can be accessed. One variable can access another if they are both in the same scope, however, a n error will be thrown if they are in separate scopes. While inheriting from the scope in which it was declared, each set of curly braces gets its own new scope. Dart is a language with a lexical scope. The most newly declared variable of the same name will be accessed by subsequent scopes using lexical scoping. The innermost scope is first searched, followed by a search of the surrounding scopes.

Variable scopes: The lifespan of a variable in which a specific block of code is run is referred to as variable scope. The type of variables can be used to determine the scope of variables:

1. Local Variable: A local variable is a variable that is declared within programming blocks or functions (for example: for loop, if else, switch, and many more).

- It should only be used in the block or function code where it was declared or defined.
- The scope of a variable is finished, or to say, the local variable is eliminated, once that particular code of block or functions is run.

2. Global Variable: Global variables are variables that are defined outside of the function and can be accessed from any location. The scope of a global variables is confined starting from the creation of the program to the ending of it.

Ultimately, in comparison, local variables are used only when they need to be manipulated within a single block of code, whereas global variables are used whenever they need to be accessed from anywhere in the program. Local variables are erased as soon as a block of code is performed, whereas global variables retain memory until the program is completed. As a result, each sort of variable has a distinct purpose. Therefore, whenever possible, opt to utilize a local variable to save memory. Because a single global variable uses more memory than a local variable, it is more expensive.

10. Program (Code and Run):

10.1. Code in Dart:

```dart
/*
CPCS-301 Project - Group-11-
Program Idea:
Write a program that reads "X" grades from the user. The grades should be stored in an array of X
sizes. The program should compute the "Average", "Minimum", and "Maximum" grades

*/
import 'dart:math';
import 'dart:convert';
import 'dart:io';

void main() {
  // read number of greads from the user
  print('Enter Number of grades: ');
  int Number_of_grades=int.parse(stdin.readLineSync());
  // creat array with size (Number_of_grades) to save grades
  var array_grades=new List(Number_of_grades);
  // variable to calculate the total grades
  var total=0;
  // variable to save grade
  int grade=0;
  // variable to count number of grades
  int count=1;
  // loop to save greades in array
  for(int i=0;i<array_grades.length;i++){
    // read grade from user
    print('Grade $count: ');
    grade=int.parse(stdin.readLineSync());
    array_grades[i]=grade;
    total+=grade;
    count++;
  }
  // calculate average of grade
  var average=total/Number_of_grades;
  print('\nAvreage: $average');
  // find Maximum and Minimum grade by sorting array
  array_grades.sort();
  var maximum = array_grades.last;
  var minimum = array_grades.first;
  print('Maximum Grade: $maximum');
  print('Minimum Grade: $minimum');
}
```

Figure 7: Program code

10.2. Sample Output:

```
> run-project
Enter Number of grades:
4
Grade 1:
24
Grade 2:
12
Grade 3:
33
Grade 4:
17

Avreage: 21.5
Maximum Grade: 33
Minimum Grade: 12
>
```

Figure 8: Screenshot of output

11. Conclusion:

In conclusion, Dart is an easy-to-learn programming language offered by Google aimed at web developers. It was announced in late 2011 and its first version appeared in late 2013. It appeared to solve problems in the JavaScript language. Dart has many versions, and its last release was in the current year 2022 which is Dart 2.16.

Dart is a lexical scope language. dart has no arrays, instead it uses lists. dart types of variables include dynamic, final and const. data types include boolean, sting, numbers, maps, lists, and symbols.

There are many expressions in Dart such as Arithmetic Operators, Relational Expressions, Logical Operators, Assignment Operators and Short-circuit Operators (&& and ||). All expressions are constructed from operands and operators. Some operators may be used in different contexts.

Dart language is one of the languages that facilitate work in applications and websites. It also supports work in Android and Apple, and this is not available in most other languages. Dart combines design and code and is not a restricted language in one domain.

12. Reference:

1. *Bracha, G. (2016). Statements. In The Dart programming language. essay, Addison-Wesley.*
2. *Tutorialspoint. (2015). Dart Programming Operators, Decision Making and loops.Dart programming tutorial. Retrieved from: https://www.tutorialspoint.com/dart_programming/index.htm*
3. *Flutter: What is Dart Programming – Javatpoint: https://www.javatpoint.com/flutter-dart-programming*
4. *An Introduction to Scheme and its Implementation - Interpretation and Compilation: https://www.cs.utexas.edu/ftp/garbage/cs345/schintro-v14/schintro_112.html*
5. *Syntax Analysis: Compiler Top Down & Bottom Up Parsing Types: https://www.guru99.com/syntax-analysis-parsing-*
6. *Semantic Analysis in Compiler Design – GeeksforGeeks: https://www.geeksforgeeks.org/semantic-analysis-in-compiler-design*
7. *Walrath, K. (2012, March 21). What is Dart? Retrieved from O'Reilly Radar: http://radar.oreilly.com/2012/03/what-is-dart.html*
8. *Thomsen, M. (2022, February 4). Dart 2.16: Improved tooling and platform handling. Retrieved from Medium: https://medium.com/dartlang/dart-2-16-improved-tooling-and-platform-handling-dd87abd6bad1*
9. *Dart. (2022, April 18). Dart language evolution. Retrieved from Dart: https://dart.dev/guides/language/evolution*
10. *Bak, L. (2013, November 14). Dart 1.0: A stable SDK for structured web apps. Retrieved from Dart News & Updates: https://news.dartlang.org/2013/11/dart-10-stable-sdk-for-structured-web.html*
11. *Bak , L., & Lund, K. (2015, March 5). Dart for the Entire Web. Retrieved from Dart News & Updates: https://news.dartlang.org/2015/03/dart-for-entire-web.html*
12. *Answersje. (2021, July 3). Dart Programming language: history, features, applications, why learn? Retrieved from Answersjet:*

https://www.answersjet.com/2021/06/dart-programming-language-history-features-applications-why-should-learn.html

13. *Dart Programming - Variables . (sd). Dart Programming. Retrieved May 7, 2022, from https://www.tutorialspoint.com/dart_programming/dart_programming_variables.htm*

14. *Durg, S. S. (2020, March 16). Data types and variables in Dart. OpenGenus IQ: Computing Expertise & Legacy. Retrieved May 7, 2022, from https://iq.opengenus.org/data-types-and-variables-in-dart/*

15. *Jay Tillu. (2019, September 16). List in dart. Medium. Retrieved May 7, 2022, from https://medium.com/jay-tillu/list-in-dart-53b20246ca7b*

16. *Codevscolor. (n.d.). Dart Scope and lexical scoping. CodeVsColor. Retrieved May 7, 2022, from https://www.codevscolor.com/dart-lexical-scoping*

17. *A tour of the Dart language. (n.d.). Dart.dev. https://dart.dev/guides/language/language-tour*

18. *Dart Programming - Exceptions. (n.d.). Www.tutorialspoint.com. Retrieved May 7, 2022, from https://www.tutorialspoint.com/dart_programming/dart_programming_exceptions.htm*

19. *Nasserzadeh, A. (2020, July 24). 3 Language Constructs From Dart That You Are Missing Out on in JavaScript. Medium. https://betterprogramming.pub/3-language-constructs-from-dart-that-you-are-missing-out-on-in-javascript-23c53cb767a*

20. *javinpaul. (2022, March 21). 7 Best and Free Dart Programming Courses for Beginners to Learn in 2022. Javarevisited. https://medium.com/javarevisited/6-best-dart-programming-courses-for-*

beginners-to-learn-in-2021-2-are-free-24dc56f5ac14#:~:text=If%20you%20don%E2%80%99t%20know%2C%20Dart%20is%20a%20simple%2C

21. *Dart Abstract Classes - Javatpoint. (n.d.). Www.javatpoint.com. https://www.javatpoint.com/dart-abstract-classes*

22. *Dart API Readability Rubric. (n.d.). Fuchsia. https://fuchsia.dev/fuchsia-src/development/api/dart*

23. *Dart Variables. (2021, Feb 12). Retrieved from https://www.geeksforgeeks.org/variables-and-keywords-in-dart/*

24. *dart basic syntax. (n.d.). Retrieved from javatpoint: https://www.javatpoint.com/dart-basic-syntax*

25. *Dart Basic Syntax. (n.d.). Retrieved from w3adda: https://www.w3adda.com/dart-tutorial/dart-basic-syntax*

26. *dart variable. (n.d.). Retrieved from javatpoint: https://www.javatpoint.com/dart-variable*

27. *Busbee , K. L., & Braunschweig, D. (n.d.). identifier names. Retrieved from rebus community: https://press.rebus.community/programmingfundamentals/chapter/identifier-names/*