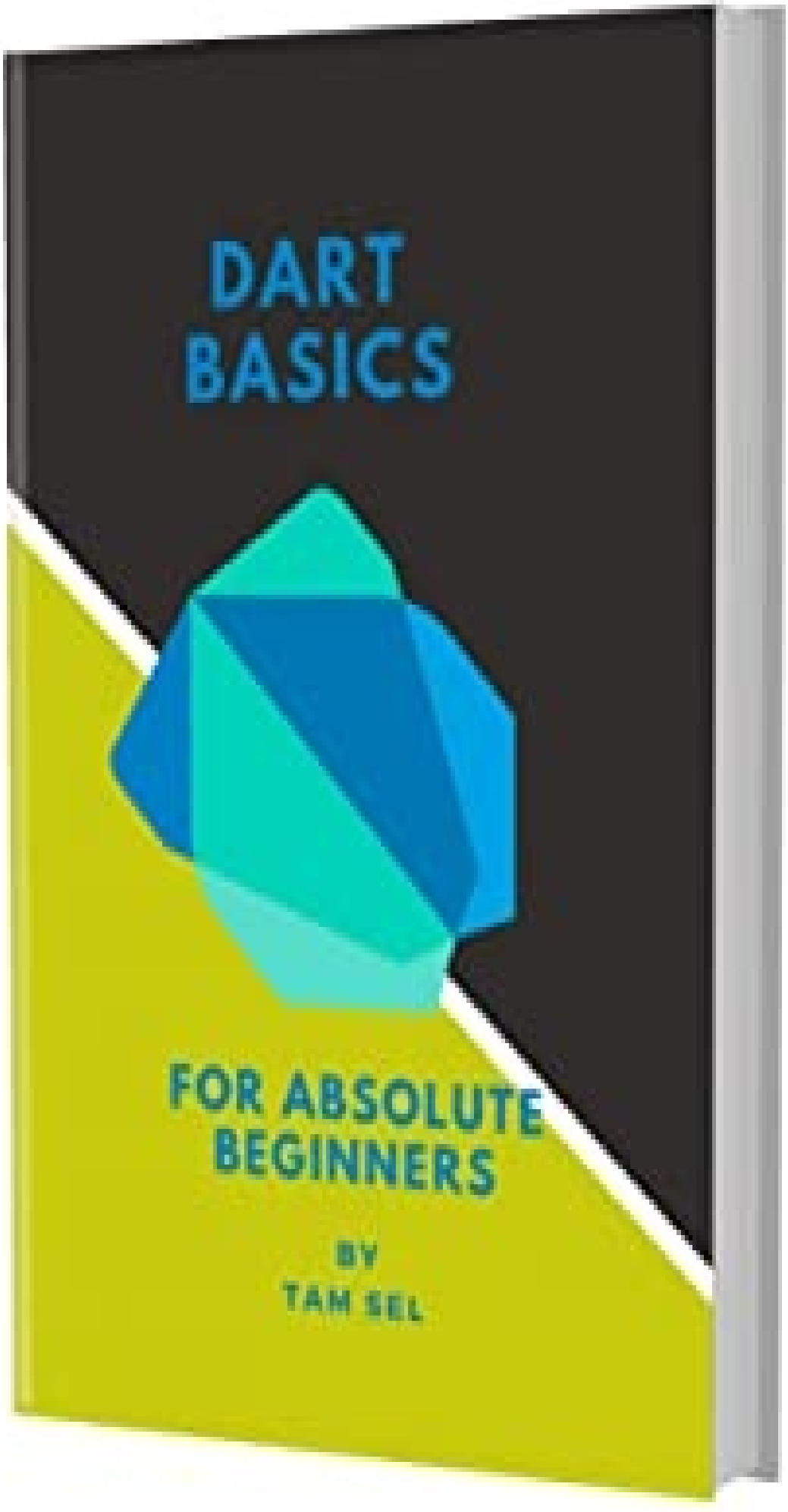

DART
BASICS
FOR ABSOLUTE
BEGINNERS
BY
TAM SEL

DART

PROGRAMMING

BASICS

FOR ABSOLUTE BEGINNERS

BY

TAM SEL

Dart Programming Language

Dart is a high-level, general-purpose modern programming language that was generated by Google.

It is a new programming language that first appeared in 2011, but only became stable in June 2017.

Dart was not very common at the time, but it gained popularity when the Flatter used it.

Dart is an object-oriented programming language that is dynamic, class-based, and has closure and lexical scope. It's very similar to Java, C, and JavaScript in terms of syntax.

If you're familiar with any of these programming languages, you'll find Dart to be a breeze to understand.

Dart is an open-source programming language that is commonly used in the Flatter system to build mobile apps, modern web apps, desktop apps, and the Internet of Things (IoT). Interfaces, mixins, abstract classes, refield generics, and type interface are among the advanced concepts endorsed. It's a compiled language that supports two different compilation methods.

AOT (Ahead of Time) - It uses the dar2js compiler to transform Dart code into optimized JavaScript code that runs on any modern web browser. It compiles the code during the build phase.

JOT (Just-In-Time) - It converts byte code to machine code (native code), but only the necessary code is converted.

IMPORTANT POINTS

In the following section, we describe Dart's characteristics.

Dart is a platform-agnostic language that runs on Windows, Mac OS X, Linux, and other operating systems.

It is an open-source language, which ensures that anybody can use it for free. It has a BSD license and is compliant with the ECMA standard.

It's an object-oriented programming language that supports all of oops' features, including inheritance, interfaces, and optional types.

Dart's reliability makes it ideal for developing real-time applications.

Dart includes the dar2js compiler, which converts Dart code into JavaScript code that can be run in any modern web browser.

These principles should be held in mind before learning the Dart. These ideas are outlined below.

Dart, like Python, treats everything as an entity, including numbers, Booleans, functions, and so on. The Object class is inherited by all objects.

When coding, Dart tools can report two types of issues: warnings and errors. Warnings indicate that the code can have a problem, but they do not prevent the code from running, while errors may prevent the code from running.

Dart programming language does not require any prior experience, and even complete beginners may learn it.

Dart's syntax is close to that of Java, C#, Java, JavaScript, and other programming languages.

If you are familiar with any of these programming languages, you can find learning to be much simpler and quicker.

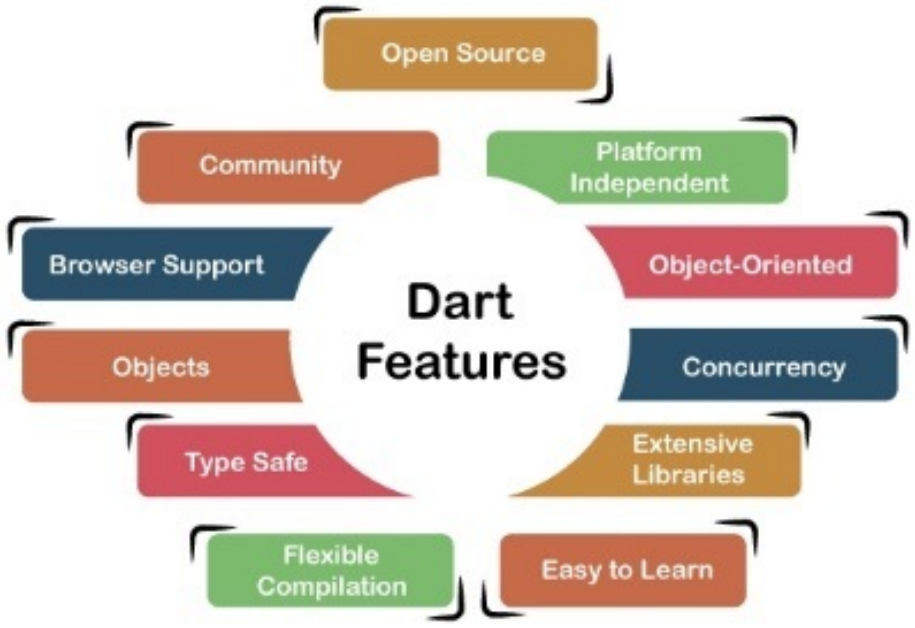
Dart Features

Dart is an open-source object-oriented programming language with a lot of useful features.

It is a brand-new programming language that provides a broad variety of programming features such as interfaces, lists, classes, dynamic and optional typing, and more.

It is designed for both the server and the browser.

The significant Dart features are described below.



Open Source

Dart is an open-source programming language, meaning it is available for free. It was created by Google, is compliant with the ECMA standard, and is licensed

under the BSD license.

Platform Independent

Dart operates for all major operating systems, including Windows, Linux, and Macintosh. Dart has its own Virtual Machine, dubbed Dart VM, which helps us to run Dart code on any operating system.

Object-Oriented

Dart is an object-oriented programming language that includes all of the oops concepts including classes, inheritance, interfaces, and optional typing. Advanced concepts such as mixin, abstract, classes, reified generic, and robust type framework are also supported.

Concurrency

Dart is an asynchronous programming language, which means it uses Isolates to help multithreading. Isolates are self-contained entities that are connected to threads but do not share memory and use message passing to create contact between processes. To ensure efficient contact, the message should be serialized.

Extensive Libraries

SDK (Software Development Kit), core, math, async, math, convert, javascript, IO, and other libraries are all included in Dart. It also allows you to organize your Dart code into libraries with appropriate namespacing. The import statement will reuse it.

Easy to learn

As we discussed in the previous section, learning Dart is not a Herculean task since its syntax is identical to Java, C#, JavaScript, Kotlin, and other

programming languages. If you know any of these languages, you should be able to pick up the Dart quickly.

Flexible Compilation

Dart helps you to compile the code in a number of ways, and it does so easily. It supports AOT (Ahead of Time) and JIT (Just in Time) compilation processes (Just-in-Time). The Dart code is sent in a different language that can be run on modern web-brewers.

Type Safe

Dart is a type-safe language, which means it uses both static type checking and runtime checks to ensure that a variable's value always matches its static type, often known as sound typing.

Kind annotations are optional due to type interference, even though types are necessary. This increases the readability of the code. Another benefit of using a type-safe language is that when we modify a portion of code, the framework alerts us about the previous update.

Objects

All is treated as an entity by the Dart. An object is the value that is assigned to the attribute. In Dart, functions, numbers, and strings are all objects.

Browser Support

Both modern web browsers are supported by the Dart. It includes the dart2js compiler, which transforms Dart code into optimized JavaScript code that works in any web browser.

Dart Installation

To learn the Dart programming language, we must first install the Dart programming environment on our local computer. The following instructions will show you how to install the Dart SDK (Software Development Kit) on a variety of operating systems. You can skip this section if you've already installed it.

Install the Dart SDK on your Windows device

To install Dark SDK on Windows, follow the steps below.

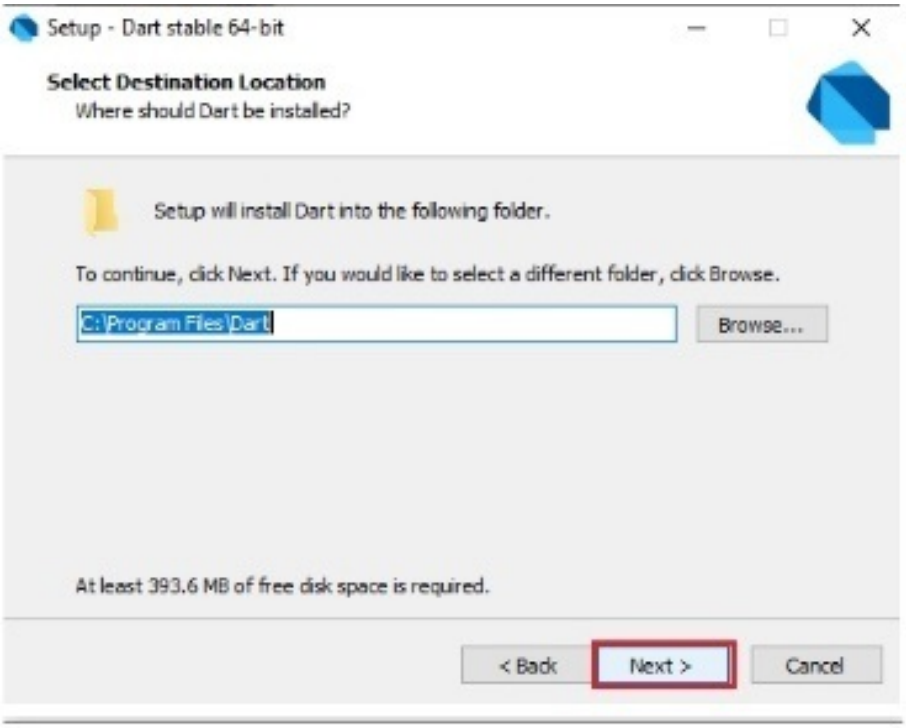
To download the SDK, go to your browser and type the following code.

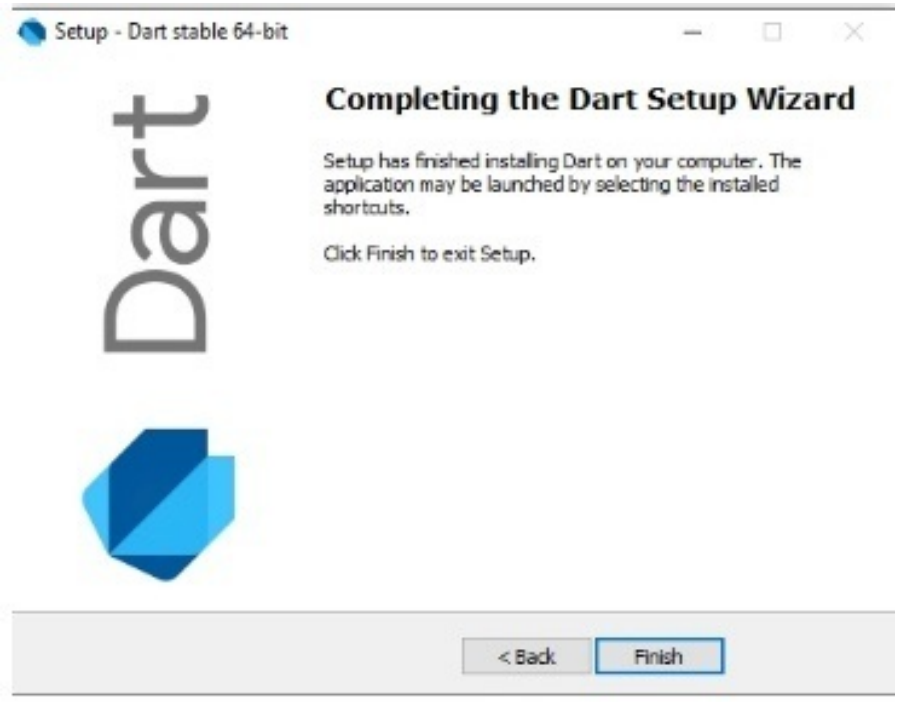
Dart Windows can be found at http://www.gekorm.com/dart-windows/ .

It will take you to the specified link. To get started, go to the following page.

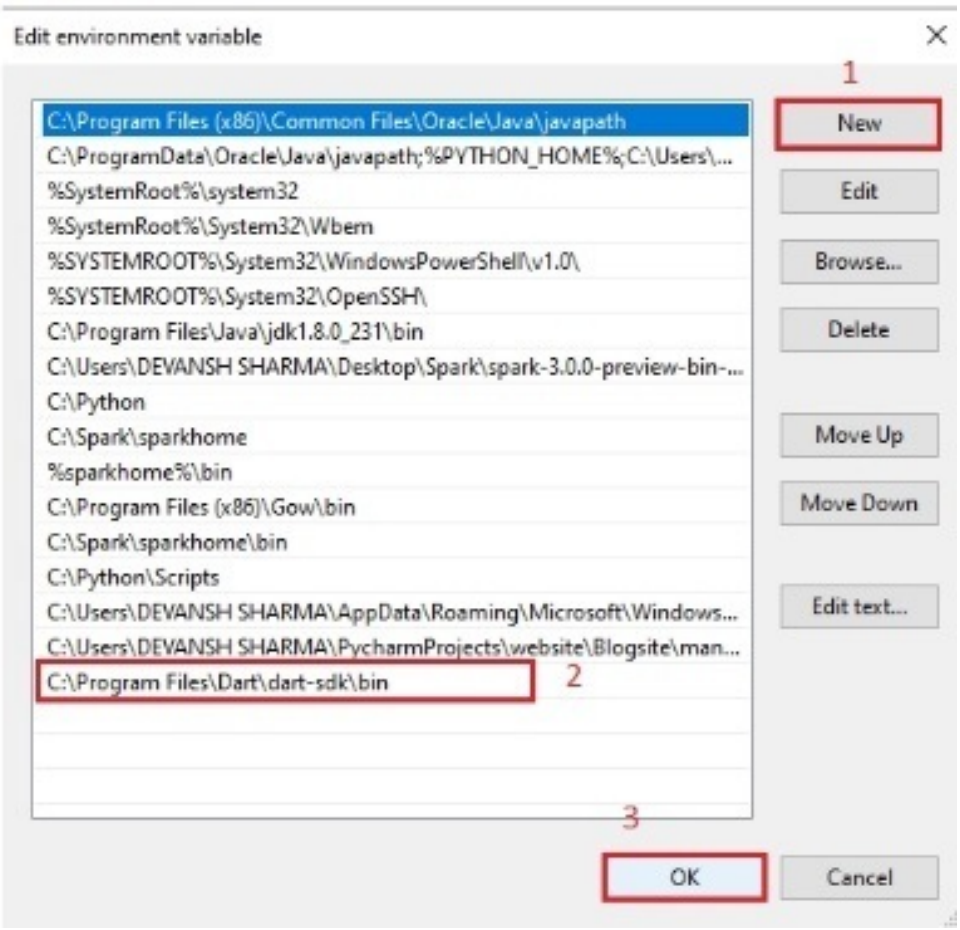
STEP 2 : Launch the Dart installer (it's the.exe file we downloaded in the previous step) and select Next.

Step 3 : It gives you the choice of choosing a Dart installation course. After you've chosen your course, press the Next button.





Step 4: In advance device properties, set the PATH environment variable to "C:Program FilesDartdart-sdkbin" after the download is complete.



Step 5 : Now open the terminal and type dart to check the Dart installation.

Online Dart Editor

We've talked about installing Dark on different operating systems so far, but if we don't want to install Dart, we can use an online Dark editor (called DartPad) to run Dark programs. DartPad is available online at https://dartpad.dev/. The DartPad will run dart scripts and view HTML as well as console output.

Dart First Program

Dart is simple to learn if you are familiar with Java, C++, JavaScript, and other programming languages. The simplest "Hello World" program demonstrates the programming language's basic syntax. It's a method of putting the framework and working environment to the test. We'll go over the basics of Dart syntax in this tutorial. The first program, which is mentioned below, can be run in several ways:

Use the Command Line

Running on Browser

Using the IDE

Before we run the first program, we need to make sure the Dart SDK is configured correctly. In our previous tutorial, we covered the entire installation process. Let's get started with our first program.

Using Command Line

1st step:

If the terminal prompts for dart runtime, Dart has been successfully installed.

2nd Step:

Build a file named "helloword.dart" in a text editor. The file extension should be.dart, which indicates that the file is a Dart program.

The main() function implies that we are at the start of our program. It's a crucial role that kicks off the program's execution.

print() - Displays the output on the console using this feature. It's comparable to C, JavaScript, or any other programming language. Curly brackets and semicolons are required for proper use.

3rd step:

Open a command prompt and compile the program. Type dart helloworld.dart to run the Dart software. The screen will show Hello World.

Running on Browser

Dart has an online editor called DartPad, which can be found at https://dartpad.dartlang.org/.

We can write code on the left side of the screen, and the output appears on the right. With Dart Code, we can add HTML and CSS. Dart also includes some learning resources, such as sample programs.

Using IDE

Dart is supported by a number of IDEs, including Visual Studio Code,

WebStorm, IntelliJ, and others. Download the dart extension for Visual Studio and run the code.

Dart Basic Syntax

Dart Identifiers

Identifiers are the names given to variables, methods, classes, and functions, among other things. An identifier is a string of letters ([A to Z],[a to z]), digits ([0-9]), and underscore (_), with the first character not being a number. A few guidelines for defining identifiers are listed below.

A digit should not be the first character.

Except for the underscore (_) and the dollar sign ($), special characters are not permitted.

Two underscores (__) in a row are not permitted.

Alphabet (uppercase or lowercase) or underscore must be the first letter.

Whitespace is not used in identifiers, and they must be unique.

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

Dart Printing and String Interpolation

The print() function is used to print output to the terminal, and the string interpolation is done with $expression. Here's an example:

```
void  main()
{
   var name =  "Peter" ;
   var roll_no =  24 ;
   print( "My name is ${name} My roll number is ${roll_no}" );
}
```

OUTPUT

```
My name is Peter My roll number is 24
```

Semicolon in Dart

The semicolon is used to end a sentence, indicating that the statement has come to an end.

A semicolon must be used at the end of each statement (;).

Using a semicolon as a delimiter, we can write several statements on a single line.

If the compiler is not used correctly, it will produce an error.

EXAMPLE

```
var msg1 =  "Hello World!" ;
var msg2 =  "How are you?"
```

Dart Whitespace and Line Breaks

Whitespaces are ignored by the Dart compiler.

In our software, it is used to define space, tabs, and newline characters.

It is used to differentiate one aspect of a sentence from another.

In our software, we can also use space and tabs to describe indentation and provide the proper format.

It makes code more readable and understandable.

Block in Dart

The curly braces enclose a series of statements that make up the block.

Curly braces are used in Dart to group all of the statements in a block.

Consider the syntax below.

SYNTAX

```
{  //start of the block

    //block of statement(s)

} // end of the block
```

Dart Command-Line Options

The Dart command-line options are used to modify Dart script execution. The standard command-line options are given below.

Sr.	Command-line Options	Descriptions
1.	-c or -c	It allows both assertion and type checks.
2.	--version	It shows VM version information.
3.	--package<path>	It indicates the path to the package resolution configuration file.
4.	-p <path>	It indicates where to find the libraries.
5.	-h or -help	It is used to seek help.

Enable Checked Mode

In general, the Dart program operates in one of two modes, as shown below.

- Checked Mode
- Production Mode

Checked Mode - In Dart code, the checked mode allows various checks, such as type-checking. During the creation of systems, it warns or throws errors. To use the checked mode, type -c or —checked before the name of the dart script-file in the command prompt. The Dart VM runs in tested mode by default.

Production Mode - In production mode, the Dart script plays. It ensures that performance is improved while the script is running. Consider the following illustration.

EXAMPLE

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

Dart Comments

The Dart compiler ignores the collection of statements known as comments during program execution.

It is used to make the source code more readable. In general, comments provide a quick overview of what is going on in the code.

Variables, functions, classes, and every other statement in the code can all be defined.

Programmers should make use of the statement to improve their skills.

In the Dart, there are three forms of remarks.

Types of Comments

- Single-line Comments
- Multi-line Comments
- Documentation Comments

Single-line Comment

We may use the / to add comments to a single line (double-slash). Single-line comments may be used before a line break is reached.

EXAMPLE

```
void  main(){
    // This will print the given statement on screen
   print( "Welcome to Codepoint" );
}
```

Output

Welcome to Codepoint

The Dart compiler completely ignored the / (double-slash) statement and retuned the output.

Multi-line Comment

When we need to add comments to several lines, we can use the /*.....*/ syntax. Anything written within the /*...*/ is ignored by the compiler, but it cannot be nested with multi-line comments. Consider the following situation.

EXAMPLE

```
void  main(){
    /* This is the example of multi-line comment
   This will print the given statement on screen */

   print( "Welcome to Codepoint" );
}
```

OUTPUT

Welcome to Codepoint

Dart Documentation Comment

The comments in a document are used to create documentation or a guide for a project or software package.

It can be a single-line or multi-line statement with the characters / or /* at the beginning. On consecutive lines, we can use ///, which is the same as the multiline statement.

Except for those written within the curly brackets, the Dart compiler ignores these lines.

Groups, functions, parameters, and variables can all be specified. Consider the following illustration.

SYNTAX

```
///This
///is
///a example of
/// multiline comment
```

EXAMPLE

```
void  main(){
    ///This is
    ///the example of
    ///multi-line comment
    ///This will print the given statement on screen.
   print( "Welcome to Codepoint" );
}
```

OUTPUT

```
Welcome to Codepoint
```

Dart Keywords

Dart Keywords are reserve terms for the compiler that have a special meaning.

It can't be used as the name of a variable, class, or function.

Keywords are case sensitive, so write them exactly as they are described.

The Dart contains 61 keywords.

Some of them are popular, and you may be familiar with a few of them, while others are special.

The following is a list of the Dart keywords.

abstract [2]	else	import [2]	super
as [2]	enum	in	switch
assert	export [2]	interface [2]	sync [1]
async [1]	extends	is	this
await [3]	extension [2]	library [2]	throw
break	external [2]	mixin [2]	true
case	factory	new	try
catch	false	null	typedef [2]
class	final	on1	var
const	finally	operator [2]	void
continue	for	part [2]	while
covarient [2]	Function [2]	rethrow	with
default	get [2]	return	yield [3]

deffered [2]	hide [1]	set [2]	
do	if	show [1]	
dynamic [2]	implements [2]	static [2]	

There are a few keywords labelled with a superscript in the above list of keywords (1,2 and 3). Following that, we'll explain why superscript is used.

Subscript 1 - These keywords are referred to as contextual keywords. They have a specific purpose and are found in specific locations.

Subscript 2 - These keywords are referred to as built-in identifiers. These keywords are used to port JavaScript code to Dart; they are considered valid identifiers, but they cannot be used in class names, feature names, or import prefixes.

Subscript 3 - These are newly added asynchrony-related keywords.

Dart Data Types

The most significant fundamental features of a programming language are the data types.

The data form of a variable in Dart is determined by its value.

Variables are used to store values and to reserve memory space.

The data-type defines the type of value that the variable will store.

Every variable has a distinct data form.

The variables in the Dart language cannot change because it is a static language.

The built-in Data types supported by Dart are as follows.

- Number
- Strings
- Boolean
- Lists
- Maps
- Runes
- Symbols

Dart Number

The numeric values are stored in the Darts Number. There are two types of numbers: integer and double.

Integer - An integer represents a whole number or a value that is not fractional.

e 64-bit non-decimal numbers between -263 and 263 are represented by the integer data form.

An unsigned or signed integer value may be stored in a variable. The following is an example:

```
int  marks =  80 ;
```

Double - A double value for a floating number or a number with several decimal points represents 64 bits of information (double precision). The double form variable is declared using the double keyword.

```
doubl e   pi =    3.1 4  ;
```

Dart Strings

A character's series is represented by a string. If we store information such as a person's name, address, or a special character, for example.

Single quotes or double quotes are used to indicate it. A Dart string is made up of UTF-16 code units in a particular order.

```
var msg =  "Welcome to Codepoint" ;
```

Dart Boolean

The true and false values are represented by the Boolean form. Boolean Form is denoted by the bool keyword. The true or false value cannot be expressed by the numeric values 1 and 0.

```
bool isValid =    tru e  ;
```

Dart Lists

The list in Dart is a set of organized items (value). A list is equivalent to an array

in definition.

An array is a set of multiple elements stored in a single variable.

The comma enclosed in the square bracket[] separates the elements in the set.

The following is a sample list.

```
var list = [ 1 , 2 , 3 ]
```

Dart Maps

The maps form is used to store key-value pairs of values. Each key has a value associated with it.

Any type of key and value can be used.

The key in Map must be special, but a value can be repeated.

Curly braces () are used to describe the Map, and a comma separates each pair.

Dart Runes

The strings, as we know, are a series of Unicode UTF-16 code units.

Unicode is a method of describing a distinct numerical value for each digit, letter, and symbol.

Since Dart Runes are a unique string of Unicode UTF-32 characters.

It's a symbol for the special syntax.

The special heart character, for example, is similar to Unicode code u2665, where u stands for Unicode and the numbers are hexadecimal integers.

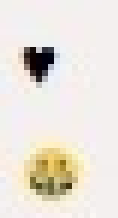
If the hex value is less than or equal to four digits, a curly bracket () is used.

An emoji, for example, is represented as u1f600.

The following is an example.

```
void  main(){
   var heart_symbol =  '\u2665' ;
   var laugh_symbol =  '\u{1f600}' ;
   print(heart_symbol);
   print(laugh_symbol);
}
```

OUTPUT



Dart Symbols

Dart Symbols are artifacts that are used to refer to a declared operator or identifier in a Dart program.

Since identifier names can change but identifier symbols cannot, it is common in APIs to refer to identifiers by name.

Dart Variable

A variable is a data structure that stores a value and corresponds to a memory location in a machine.

The Dart compiler allocates memory space when we construct a variable.

The type of variable determines the size of the memory block of memory. To build a variable, we must adhere to certain guidelines.

Here's an example of a variable being generated and a value being allocated to it.

```
var name =    'TAMSEL
```

The name variable retains the value of the string 'TAMSEL'. Variables in Dart are used to store references. The above variable holds a reference to a TAMSEL-valued String.

Rule to Create Variable

Special characters, such as whitespace, mathematical symbols, runes, Unicode characters, and keywords, are not included in the variable.

The variable's first character should be an alphabet ([A to Z],[a to z]). As the first character, digits are not permitted.

Case matters when it comes to variables. Variable age and AGE, for example, are handled differently.

Except for the underscore(_) and the dollar sign($), special characters such as #, @, &, not allowed.

The variable name should be retable and readable by the program.

Declare Variable in Dart

Before we can use a variable in a program, we must first declare it.

The var keyword is used to declare variables in Dart.

Since Dart is an infer type language, the compiler automatically decides the type of data based on the value assigned to the variable.

The syntax is as follows.

```
var <variable_name>  = <value>;
```

or

```
var <variable_name>;
```

EXAMPLE

```
var name =  'Andrew'
```

In the preceding case, the variable name has allocated memory space.

The semicolon(;) is needed because it distinguishes one program statement from the next.

Type Annotations

The Dart is an infer language, but it also has a form annotation, as we mentioned earlier.

The type of value that the variable will store is suggested when the variable is declared.

We add the data type as a prefix before the variable's name in the type annotation to ensure that the variable may store a particular data type.

The syntax is as follows.

<type> <variable_name>;

or

<type> <name> = <expression>;

Example -

```
int  age;
String msg =  "Welcome to Codepoint" ;
```

In the preceding example, we declared a variable called age to hold the integer data. The string type data was stored in the msg variable.

Dart Operators

An operator is a symbol that manipulates the values of its operand or performs operations on it.

The defined expression is 5+4. The operands are 5 and 4, and the operator is "+."

Dart comes with a large number of built-in operators that can be used to perform a variety of tasks.

Operators may be unary or binary, with unary operators taking just one operand and binary operators taking two operands.

Operators come in a variety of shapes and sizes.

The Dart Operators are listed below.

Types of Operators

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Type test Operators
- Logical Operators
- Bitwise Operator
- Conditional Operators
- Casecade notation(..) Operators

Dart Arithmetic Operators

The most popular operators for performing addition, subtraction, multiplication,

and division are arithmetic operators.

Let's assume variable a has a value of 20 and variable b has a value of 10, then –

Sr.	Operator Name	Description	Example
1.	Addition(+)	It adds the left operand to the right operand.	a+b will return 30
2.	Subtraction(-)	It subtracts the right operand from the left operand.	a-b will return 10
3	Divide(/)	It divides the first operand by the second operand and returns quotient.	a/b will return 2.0
4.	Multiplication(*)	It multiplies the one operand to another operand.	a*b will return 200
5.	Modulus(%)	It returns a reminder after dividing one operand to another.	a%b will return 0
6.	Division(~/)	It divides the first operand by the second operand and returns integer quotient.	a/b will return 2
7.	Unary Minus(-expr)	It is used with a single operand changes the sign of it.	-(a-b) will return -10

Example

```
void main(){
```

```
  print( "Example of Assignment operators" );
  var n1 =  10 ;
  var n2 =  5 ;

  print( "n1+n2 = ${n1+n2}" );
  print( "n1-n2 = ${n1-n2}" );
  print( "n1*n2 = ${n1*n2}" );
  print( "n1/=n2 = ${n1/n2}" );
  print( "n1%n2 = ${n1%n2}" );
}
```

Output:

Example of Arithmetic operators

n1+n2 = 15

n1-n2 = 5

n1*n2 = 50

n1/=n2 = 2

n1%n2 = 0

Dart Unary Operators

The ++ and — operators in Java are known as increment and decrement operators, and they are also known as unary operators.

Unary operators run on a single operand, with ++ adding 1 to the operands and — subtracting 1 from the operands.

There are two ways to use the unary operators: postfix and prefix. When ++ is

used as a postfix (for example, x++), it first returns the value of the operand before incrementing the value of x.

When the prefix — is used as a prefix (for example, ++x), the value of x is increased.

Sr.	Operator Name	Description	Example
1.	++(Prefix)	It increment the value of operand.	++x
2.	++(Postfix)	It returns the actual value of operand before increment.	x++
3.	--(Prefix)	It decrement the value of the operand.	--x
4.	--(Postfix)	It returns the actual value of operand before decrement.	x--

Example

```
void  main() {
  var x =  30 ;
  print(x++);                //The postfix value

var y =  25 ;
print(++y);                //The prefix value,

var z =  10 ;
print(--z);                //The prefix value
```

```
var u =  12 ;
  print(u--);    }          //The postfix value
```

Output:

30

26

9

12

Assignment Operator

The assignment operators are used to give variables a value.

It can also be used in conjunction with arithmetic operators.

The following is a list of assignment operators.

Assume **a** has a value of 20 and **b** has a value of 10.

Operators Name	Description
= (Assignment Operator)	It assigns the right expression to the left operand.
+=(Add and Assign)	It adds right operand value to the left operand and resultant assign back to the left operand. For example - a+=b → a = a+b → 30
-=(Subtract and Assign)	It subtracts right operand value from left operand and resultant assign back to the left operand. For example - a-=b → a = a-b → 10
*=(Multiply and Assign)	It multiplies the operands and resultant assign back to the left operand. For example - a*=b → a = a*b → 200
/=(Divide and Assign)	It divides the left operand value by the right operand and resultant assign back to the left operand. For example - a%=b → a = a%b → 2.0

~/=(Divide and Assign)	It divides the left operand value by the right operand and integer remainder quotient back to the left operand. For example - a%=b → a = a%b → 2
%=(Mod and Assign)	It divides the left operand value by the right operand and remainder assign back to the left operand. For example - a%=b → a = a%b → 0
<<=(Left shift AND assign)	The expression a<<=3 is equal to a = a<<3
>>=(Right shift AND assign)	The expression a>>=3 is equal to a = a>>3
&=(Bitwise AND assign)	The expression a&=3 is equal to a = a&3
^=(Bitwise exclusive OR and assign)	The expression a^=3 is equal to a = a^3
|=(Bitwise inclusive OR and assign)	The expression a|=3 is equal to a = a|3

Example

```
void  main(){
 print( "Example of Assignment operators" );

  var n1 =  10 ;
  var n2 =  5 ;
```

```
  n1+=n2;
  print( "n1+=n2 = ${n1}" );

  n1-=n2;
  print( "n1-=n2 = ${n1}" );

  n1*=n2;
  print( "n1*=n2 = ${n1}" );

  n1~/=n2;
  print( "n1~/=n2 = ${n1}" );
  n1%=n2;
  print( "n1%=n2 = ${n1}" );
}
```

Output:

Example of Assignment operators

n1+=n2 = 15

n1-=n2 = 10

n1*=n2 = 50

n1~/=n2 = 10

n1%=n2 = 0

Relational Operator

The comparison operators, also known as relational operators or comparison operators, are used to compare two expressions and operands.

The Boolean truth and false are returned when two expressions are compared.

Find the following table if a holds 20 and b holds 10.

Sr.	Operator	Description
1.	>(greater than)	a>b will return TRUE.
2.	<(less than)	a<b will return FALSE.
3.	>=(greater than or equal to)	a>=b will return TRUE.
4.	<=(less than or equal to)	a<=b will return FALSE.
5.	==(is equal to)	a==b will return FALSE.
6.	!=(not equal to)	a!=b will return TRUE.

Example -

```
1.  void  main() {
2.  var a =  30 ;
3.  var b =  20 ;
4.
5.  print( "The example of Relational Operator" );
6.
7.  var res = a>b;
8.  print( "a is greater than b: " +res. toString());
    // We will learn the toString in next tutorial
9.
10. var res0 = a<b;
11. print( "a is less than b: " +res0. toString());
12.
13. var res1 = a>=b;
14. print( "a is greater than or equal to b: " +res1. toString());
```

```
15.
16.  var res2 = a<=b;
17.  print( "a is less than and equal to b: " +res2. toString());
18.
19.  var res3 = a!= b;
20.  print( "a is not equal to  b: " +res3. toString());
21.
22.  var res4 = a==b;
23.  print( "a is  equal to  b: " +res4. toString());
24.  }
```

Output:

The example of Relational Operator

a is greater than b: true

a is less than b: false

a is greater than or equal to b: true

a is less than and equal to b: false

a is not equal to b: true

a is equal to b: false

Dart Type Test Operators

At runtime, the Form Test Operators are used to test the types of expressions.

Consider the table below.

Operator	Description
as	It is used for typecast.
is	It returns TRUE if the object has specified type.
is!	It returns TRUE if the object has not specified type.

EXAMPLE

```
void  main()
{
  var num =  10 ;
  var name =  "Codepoint" ;
  print(num is  int );
  print(name is! String );
}
```

Output:

true

false

Dart Logical Operators

The expressions are analyzed and a decision is taken using the Logical Operators.

The following logical operators are provided by Dart.

Operator	Description
&&(Logical AND)	It returns if all expressions are true.
||(Logical OR)	It returns TRUE if any expression is true.
!(Logical NOT)	It returns the complement of expression.

EXAMPLE

```
void  main(){
  bool bool_val1 =  true , bool_val2 =  false ;
  print( "Example of the logical operators" );

  var val1 = bool_val1 && bool_val2;
  print(val1);

  var val2 = bool_val1 || bool_val2;
  print(val2);

  var val3 = !(bool_val1 || bool_val2);
  print(val3);
}
```

Output:

Example of the logical operators

false

true

false

Dart Bitwise Operators

The Bitwise operators work on the value of the two operands bit by bit.

The table of bitwise operators is shown below.

Let's take a look at the following example.

If a = 7
b = 6
then binary(a) = 0111
binary(b) = 0011
Hence a & b = 0011 , a|b = 0111 and a^b = 0100

Operators	Description
&(Binary AND)	It returns 1 if both bits are 1.
|(Binary OR)	It returns 1 if any of bit is 1.
^(Binary XOR)	It returns 1 if both bits are different.
~(Ones Compliment)	It returns the reverse of the bit. If bit is 0 then the compliment will be 1.
<<(Shift left)	The value of left operand moves left by the number of bits present in the right operand.
>>(Shift right)	The value of right operand moves left by the number of bits present in the left operand.

Example

```
1.  void  main(){
2.    print( "Example of Bitwise operators" );
3.
4.    var a  =  25 ;
5.    var b =  20 ;
6.    var c =  0 ;
7.
8.     // Bitwise AND Operator
9.    print( "a & b = ${a&b}" );
10.
11.    // Bitwise OR Operator
12.   print( "a | b = ${a|b}" );
13.
14.    // Bitwise XOR
15.   print( "a ^ b = ${a^b}" );
16.
17.    // Complement Operator
18.   print( "~a = ${(~a)}" );
19.
20.    // Binary left shift Operator
21.   c = a << 2 ;
22.   print( "c<<1= ${c}" );
23.
24.    // Binary right shift Operator
25.   c = a >> 2 ;
26.   print( "c>>1= ${c}" );
27.  }
```

Output:

Example of Bitwise operators

a & b = 16

a | b = 29

a ^ b = 13

~a = 4294967270

c<<1= 100

c>>1= 6

Dart Conditional Operators

The Conditional Operator is similar to an if-else statement in that it performs the same functions.

It's the second sort of if-else condition.

It's also known as the "Ternary Operator."

The syntax is as follows.

SYNTAX 1

condition ? exp1 : exp2

SYNTAX 2

exp1 ?? expr2

Example - 1

```
void  main() {
  var x =  null ;
  var y =  20 ;
  var val = x ?? y;
  print(val);
}
```

Output:

20

Dart Cascade notation Operators

The Operators (..) in the Cascade notation are used to evaluate a series of operations on the same object. It's the same as method chaining in that it skips a few steps and eliminates the need to save data in temporary variables.

Dart Constants

A Dart Constant is an immutable entity, which means it can't be changed or updated while the program is running.

The value of the constant variable cannot be reassigned after it has been initialized.

Defining/Initializing Constant in Dart

The Dart constant can be expressed in two different ways.

- Using the final keyword
- Using the const keyword

It's useful when we want to keep the value constant in the program. To make a constant variable, use the keywords final and const.

The data-type is used in combination with the keywords final and const. If we attempt to change the constant variable, Dart will throw an exception.

The compile-time constant is represented by the const keyword, and the final variable can only be set once.

Define Constant Using final Keyword

Using the final keyword, we can describe the constant.

The syntax is as follows.

Syntax:

```
final  const_name;
```

or

```
final  data_type const_name
```

Example

```
void  main () {
  final  a =  10 ;
  final  b =  20 ;

 print(a);
 print(b);
}
```

Output:

10

20

Define Constants Using const Keyword

The const keyword is used to describe constants. The syntax is as follows.

Syntax -

```
const  const_name
```

Or

```
const  data_type const_name
```

EXAMPLE

```
void  main() {
   const  name=  "Peter" ;
  print(name);
```

```
}
```

Output:

```
Peter
```

Dart Number

The numeric value is represented by the Number data form.

It can be one of two styles of Dart:

Dart integer

Whole numbers that can be written without a fractional part are known as integers.

For instance, - 20, 30, -3215, 0, and so on. There are two types of integer numbers: signed and unsigned.

The integer value is represented by non-decimal numbers ranging from -263 to 263 in length.

In Dart, the int keyword is used to declare an integer value.

```
int  id =  501 ;
```

Dart Double

The numbers that can be written with floating-point numbers or numbers with greater decimal points are known as double numbers.

In Dart, the double keyword is used to declare a Double value.

```
double  root =  1.41234 ;
or
double  rupees  =  100000 ;
```

Rules for the integer value

- A digit must be present in an integer value.
- In an integer number, the decimal points should not be used.

- Numbers that aren't signed are still positive. Numbers can be both positive and negative.
- The size of the integer value is determined by the platform, but it should not exceed 64 bits.

Example

```
void  main(){
  int  r =  5 ;
  double  pi =  3.14 ;
  double  res =  4 *pi*r*r;
 print( "The area of sphere = ${(res)}" );
}
```

Output:

The area of sphere 314

Dart String

A Dart String is a character or UTF-16 code unit sequence.

It's used to keep track of the text meaning. Single or double quotes may be used to build the string. The triple-quotes can be used to make a multiline series.

Strings are permanent, which means they can't be modified after they've been formed.

The String keyword in Dart can be used to declare a string. The string declaration's syntax is shown below.

Syntax:

```
String msg =    'Welcome to JavaTpoint '  ;
or
String msg1 =    "This is double-quoted string example. "  ;
or
String msg2 =    ' '   ' line1
line2
line 3  ' ' '
```

Printing String

The string is printed on the screen using the **print()** feature. Any letter, speech, or other object can be formatted as a string.

Dart has a function called **${expression}** that can be used to place a value within a string.

Take a look at the following scenario.

Example

```
void  main() {
   String str1 =  'this is an example of a single-line string' ;
   String str2 =  "this is an example of a double-quotes multiline line string" ;
   String str3 =  "" " this  is a multiline line
string using the triple-quotes "" ";

   var  a =  10 ;
   var b =  20 ;

   print(str1);
   print(str2);
   print(str3);

  // We can add expression using the ${expression}.
   print( "The sum is  = ${a+b}" );
}
```

Output:

```
this is an example of a single-line string

this is an example of a double-quotes multiline line string

this is a multiline line

   string using the triple-quotes

The sum is  = 30
```

String Concatenation

To combine the two strings, use the **+ or += operator** . The following is an example.

```
void  main() {
  String str1 =  'Welcome To ' ;
  String str2 =  "Codepoint" ;
  String str3 = str1+str2;

  print(str3);
}
```

Output:

Welcome To Codepoint

String Interpolation

String interpolation is a method of manipulating a string and creating a new string by adding another value to it.

It can be used to test a string that contains placeholders, variables, or an interpolated expression.

String interpolation is achieved with the $expression variable.

The corresponding values are substituted for the expressions.

Let's look at an example to help you understand.

EXAMPLE

```
void  main() {
  String str1 =  'Hello ' ;
  String str2 =  "World!" ;
  String str3 = str1+str2;
```

```
 print(str3);

   var x =  26 ;
   var y =  10 ;

   print( "The result is  = ${x%y}" );

   var name =  "Peter" ;
   var roll_nu =  101 ;

   print( "My name is ${name}, my roll number is ${roll_nu}" );
}
```

Output:

Hello World!

The result is = 6

My name is Peter, my roll number is 101

Explanation

We declared two strings variables, generated a new string after concatenation, and printed the result in the above code.

We generated two variables that hold integer values, ran the mod command, and then printed the result using string interpolation.

As seen in the example above, we can use string interpolation as a placeholder.

String Properties

Property	Description
codeUnits	It returns an unmodified list of the UTF-16 code units of this string.
isEmpty	If the string is empty, it returns true.
Length	It returns the length of the string including whitespace.

String Methods

Methods	Descriptions
toLowerCase()	It converts all characters of the given string in lowercase.
toUpperCase()	It converts all characters of the given string in uppercase.
trim()	It eliminates all whitespace from the given string.
compareTo()	It compares one string from another.
replaceAll()	It replaces all substring that matches the specified pattern with a given string.
split()	It splits the string at matches of the specified delimiter and returns the list of the substring.
substring()	It returns the substring from start index, inclusive to end index.
toString()	It returns the string representation of the given object.
codeUnitAt()	It returns the 16-bits code unit at the given index.

Dart Lists

An array, which is an ordered set of items, is similar to a Dart List.

In any other programming language, the array is the most common and widely used set.

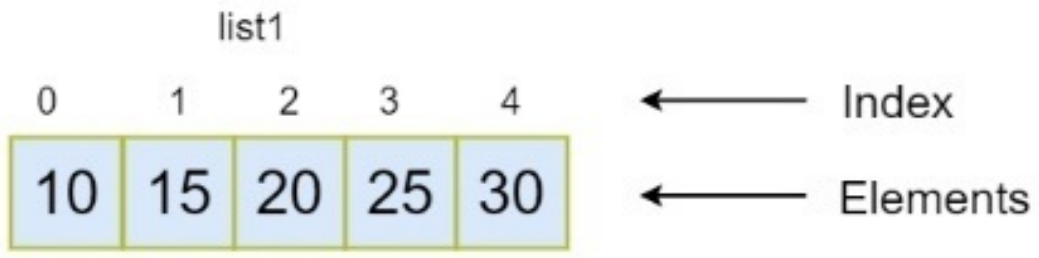
The Dart list tends to be the same as the JavaScript array literals.

The list's declaration syntax is shown below.

```
var list1 = [ 10 ,  15 ,  20 , 25 , 25 ]
```

All elements in the Dart list are stored within the square bracket ([]) and separated by commas (,).

Let's look at the list's graphical representation –



list 1 - This is the list variable for the list object.

Inde x Each element has an index number that indicates where it belongs in the list.

The index number, such as list name[index], is used to access a specific element from the list.

The indexing of a list begins at 0 and ends at length-1, where length denotes the number of elements in the list. For instance, the above list is four items long.

Element s - The actual value or dart object contained in the specified list is referred to as the List Elements.

Types of Lists

- Fixed Length List
- Growable List

Fixed Length List

The length of the fixed-length lists may be determined.

At runtime, we are unable to adjust the scale. The syntax is as follows.

SYNTAX - Make a list of fixed-size products.

```
var list_name =  new  List(size)
```

The above syntax is used to generate a fixed-size chart. At runtime, we are unable to add or remove components. If someone attempts to adjust its size, it will throw an exception.

The initialization syntax for the fixed-size list feature is shown below.

Syntax - Set the fixed-size list variable to its default value.

```
list_name[index] = value;
```

EXAMPLE

```
void  main() {
  var list1 =  new  List( 5 );
  list1[ 0 ] =  10 ;
```

```
  list1[ 1 ] =  11 ;
  list1[ 2 ] =  12 ;
  list1[ 3 ] =  13 ;
  list1[ 4 ] =  14 ;
  print(list1);
}
```

Output:

[10, 11, 12, 13, 14]

EXPLANATION

In the preceding example, we created a variable list1 that refers to a fixed-size list.

The list is five items long, and we inserted the elements in the order of their index position, with the 0th index holding 10, the 1st index holding 12, and so on.