Summer Training Course

[FTR 330]

[Up Academy]

Submitted by:

Mahmoud Magdy Kamel

ID No: 42020342

Computer Science Dept Higher technological institute

[Mahmoud Magdy Kamel **, 42020342]**

Submitted to:

Dr.Youssria

computer science. Dept.

July.2023

Abstract

I choose flutter department because I want to Develop skills in flutter

I learn in this course dart programming ,flutter , how to to implement ui from figma , how to write dart programming, Difference between flutter and dart .

Acknowledgment

At first, Thanks to ALLAH the most merciful the most gracious, for this moment has come and this work has been accomplished.
Thanks to the Higher Technological Institute of 10th Ramdan for preparing me to be a successful Engineer and lifting me up to achieve this training in an environment that's full of encouragement and motivation.

My deepest gratitude is to be delivered to Dr. [Abou Zaid], my role model in engineering. He understood the nature of my thoughts and guided me step by step till this work was brought to light. Endless trust in my potential guided me till the end. Thank you.

Special thanks to [Eng. Sara] for his help and knowledge in the field of training. Their professional touches are sensed within every phase of this summer training.

I'd like to thank my father [Magdy] , who is my motivators, visionaries and great
supporter ever since my graduation. He always pushes me up and drives me to success.

Not to forget everyone who helped me, prayed for me, wished me luck or pushed me forwards and beard a lot to help this work come to life. Thanks to my colleagues, friends, labors, technicians and everyone else for everything they did.

Last but never forgotten, Thanks to my dear family, for being supportive and always by my side. No words can express my deepest and sincere gratitude towards the love and care you have granted me in my hardest times. May ALLAH fill your hearts with happiness when we share this success together

List of Contents

What is Dart:

Dart is an open source language developed in Google with the aim of allowing developers to use an object-oriented language with static type analysis. Since the first stable release in 2011, Dart has changed quite a bit, both in the language itself and in its primary goals. With version 2.0, Dart's type system went from optional to static, and since its arrival, Flutter (we'll explain later) has become the main target of the language.

What makes Dart special:

Unlike many languages, Dart was designed with the goal of making the development process as comfortable and fast as possible for developers. So it comes with a fairly extensive set of built-in tools such as its own package manager, various compilers / transpires, a parser and formatter. Also, the Dart virtual machine and Just-in-Time build make code changes immediately executable.
Once in production, the code can be compiled in native language, so no special environment is required to run. In case of web development, Dart is transpired into JavaScript.

As for the syntax, Dart's is very similar to languages like JavaScript, Java and C ++, so learning Dart by knowing one of these languages is a matter of hours.
In addition, Dart has great support for asynchrony, and working with generators and literals is extremely easy.

Where can I use Dart:

Dart is a general-purpose language, and you can use it for almost anything:

- In web applications, using the art library: html and the transpire to transform the Dart code into JavaScript, or using frameworks like Angular Dart.
- On servers, using the art: http and art: io libraries. There are also several frameworks that can be used, such as Aqueduct.
- In console applications.
- In mobile applications thanks to Flutter.

Data Type	Keyword	Description
Number	int, double, num, Biggins	Numbers in Dart are used to represent numeric literals
Strings	String	Strings represent a sequence of characters
Booleans	bool	It represents Boolean values true and false
Lists	List	It is an ordered group of objects
Maps	Map	It represents a set of values as key-value pairs

1. Number: The number in Dart Programming is the data type that is used to hold the numeric value. Dart numbers can be classified as:

- The int data type is used to represent whole numbers.
- The double data type is used to represent 64-bit floating-point numbers.
- The num type is an inherited data type of the int and double types.

```
void main() {

// declare an integer
int num1 = 2;

// declare a double value
double num2 = 1.5;

// print the values
print(num1);
print(num2);
var a1 = num.parse("1");
var b1 = num.parse("2.34");

var c1 = a1+b1;
print("Product = ${c1}");
}
```

Output:

```
2
1.5
Product = 3.34
```

2. String: It used to represent a sequence of characters. It is a sequence of UTF-16 code units. The keyword string is used to represent string literals. String values are embedded in either single or double-quotes.

```
void main() {

    String string = 'Geeks''for''Geeks';
    String str = 'Coding is ';
    String str1 = 'Fun';
    print (string);
    print (str + str1);
}
```

Output:

GeeksforGeeks
Coding is Fun

3. Boolean: It represents Boolean values true and false. The keyword bool is used to represent a Boolean literal in DART.

```
void main() {
  String str = 'Coding is ';
  String str1 = 'Fun';

  bool val = (str==str1);
  print (val);
}
```

Output:

False

4. List: List data type is similar to arrays in other programming languages. A list is used to represent a collection of objects. It is an ordered group of objects.

```
void main()
{
    List gfg = new List(3);
    gfg[0] = 'Geeks';
    gfg[1] = 'For';
    gfg[2] = 'Geeks';

    print(gfg);
    print(gfg[0]);
}
```

Output:

[Geeks, For, Geeks]
Geeks

5. Map: The Map object is a key and value pair. Keys and values on a map may be of any type. It is a dynamic collection.

```
void main() {
  Map gfg = new Map();
  gfg['First'] = 'Geeks';
  gfg['Second'] = 'For';
  gfg['Third'] = 'Geeks';
  print(gfg);
}
```

Output:

{First: Geeks, Second: For, Third: Geeks}

Note: *If the type of a variable is not specified, the variable's type is* ***dynamic****. The dynamic keyword is used as a type annotation explicitly.*

Variables in Dart:

A variable name is the name assign to the memory location where the user stores the data and that data can be fetched when required with the help of the variable by calling its variable name. There are various types of variable which are used to store the data. The type which will be used to store data depends upon the type of data to be stored.

Syntax: *To declare a variable: type variable_name;*

Syntax: *To declare multiple variables of same type:*

type variable1_name, variable2_name, variable3_name, ...variableN_name;

Type of the variable can be among:

1. Integer

2. Double

3. String

4. Booleans

5. Lists

6. Maps

Conditions to write variable name or identifiers are as follows:

1. *Variable name or identifiers can't be the **keyword**.*
2. *Variable name or identifiers can contain alphabets and numbers.*
3. *Variable name or identifiers can't contain spaces and special characters, except the **underscore(_)** and the **dollar($)** sign.*
4. *Variable name or identifiers can't begin with number.*

***Note:** Dart supports **type-checking**, it means that it checks whether the data type and the data that variable holds are specific to that data or not.*

Example 1:

Printing default and assigned values in Dart of variables of different data types.

```
void main()
{
    // Declaring and initialising a variable
    int gfg1 = 10;

    // Declaring another variable
    double gfg2 = 0.2; // must declare double a value or it
                       // will throw error
    bool gfg3 = false; // must declare boolean a value or it
                       // will throw error

    // Declaring multiple variable
    String gfg4 = "0", gfg5 = "Geeks for Geeks";

    // Printing values of all the variables
    print(gfg1); // Print 10
    print(gfg2); // Print default double value
    print(gfg3); // Print default string value
    print(gfg4); // Print default bool value
    print(gfg5); // Print Geeks for Geeks
}
```

Output:

10

null

null

null

Geeks for Geeks

Keywords in Dart:

Keywords are the set of reserved words which can't be used as a variable name or identifier because they are standard identifiers whose function are predefined in Dart.

GeeksforGeeks
A computer science portal for geeks

abstract	continue	new	this	as
false	true	final	null	default
throw	finally	do	for	try
catch	get	dynamic	rethrow	typedef
if	else	return	var	break
enum	void	int	String	double
bool	list	map	implements	set
switch	case	while	static	import
export	in	external	this	super
with	class	extends	is	const
yield	factory			

Dynamic type variable in Dart:

This is a special variable initialised with keyword ***dynamic.*** *The variable declared with this data type can store implicitly any value during running the program. It is quite similar to* ***var*** *datatype in Dart, but the difference between them is the moment you assign the data to variable with var keyword it is replaced with the appropriate data type.*

Syntax: *dynamic variable_name;*

Example 2:

Showing how datatype are dynamically change using dynamic keyword.

```
void main()
{
    // Assigning value to geek variable
    dynamic geek = "Geeks For Geeks";

    // Printing variable geek
    print(geek);

    // Reassigning the data to variable and printing it
    geek = 3.14157;
    print(geek);
}
```

Output:

```
Geeks For Geeks
3.14157
```

Final And Const Keyword in Dart:

These keywords are used to define constant variable in Dart i.e. once a variable is defined using these keyword then its value can't be changed in the entire code. These keyword can be used with or without data type name.

Syntax for Final:

// Without datatype

final variable_name

// With datatype

final data_type variable_name

Syntax for Const:

// Without datatype

const variable_name

// With datatype

const data_type variable_name

```
void main() {
  // Assigning value to geek1 variable without datatype
  final geek1 = "Geeks For Geeks";
  // Printing variable geek1
  print(geek1);

  // Assigning value to geek2 variable with datatype
  final String geek2 = "Geeks For Geeks Again!!";
  // Printing variable geek2
  print(geek2);
}
```

Operators in Dart:

The operators are special symbols that are used to carry out certain operations on the operands. The Dart has numerous built-in operators which can be used to carry out different functions, for example, '+' is used to add two operands. Operators are meant to carry operations on one or two operands

Different types of operators in Dart:

The following are the various types of operators in Dart:

1. *Arithmetic Operators*
2. *Relational Operators*
3. *Type Test Operators*
4. *Bitwise Operators*
5. *Assignment Operators*
6. *Logical Operators*
7. *Conditional Operator*
8. *Cascade Notation Operator*

1. Arithmetic Operators:

This class of operators contain those operators which are used to perform arithmetic operation on the operands. They are binary operators i.e they act on two operands. They go like this:

Operator Symbol	Operator Name	Operator Description
+	Addition	Use to add two operands
–	Subtraction	Use to subtract two operands
-expr	Unary Minus	It is Use to reverse the sign of the expression
*	Multiply	Use to multiply two operands
/	Division	Use to divide two operands
~/	Division	Use two divide two operands but give output in integer
%	Modulus	Use to give remainder of two operands

2. Relational Operators:

This class of operators contain those operators which are used to perform relational operation on the operands. It goes like this:

Operator Symbol	Operator Name	Operator Description
>	Greater than	Check which operand is bigger and give result as boolean expression.
<	Less than	Check which operand is smaller and give result as boolean expression.
>=	Greater than or equal to	Check which operand is greater or equal to each other and give result as boolean expression.
<=	less than equal to	Check which operand is less than or equal to each other and give result as boolean expression.
==	Equal to	Check whether the operand are equal to each other or not and give result as boolean expression.
!=	Not Equal to	Check whether the operand are not equal to each other or not and give result as boolean expression.

3. Type Test Operators:

This class of operators contain those operators which are used to perform comparison on the operands. It goes like this:

Operator Symbol	***Operator Name***	***Operator Description***
is	*is*	*Gives boolean value true as output if the object has specific type*
is!	*is not*	*Gives boolean value false as output if the object has specific type*

4. Bitwise Operators:

This class of operators contain those operators which are used to perform bitwise operation on the operands. It goes like this:

Operator Symbol	Operator Name	Operator Description
&	Bitwise AND	Performs bitwise and operation on two operands.
|	Bitwise OR	Performs bitwise or operation on two operands.
^	Bitwise XOR	Performs bitwise XOR operation on two operands.
~	Bitwise NOT	Performs bitwise NOT operation on two operands.
<<	Left Shift	Shifts a in binary representation to b bits to left and inserting 0 from right.
>>	Right Shift	Shifts a in binary representation to b bits to left and inserting 0 from left.

5. Assignment Operators:

This class of operators contain those operators which are used to assign value to the operands. It goes like this:

Operator Symbol	Operator Name	Operator Description
=	Equal to	Use to assign values to the expression or variable
??=	Assignment operator	Assign the value only if it is null.

```
void main()
{
    int a = 5;
    int b = 7;

    // Assigning value to variable c
    var c = a * b;
    print(c);

    // Assigning value to variable d
    var d;
    d ? ? = a + b; // Value is assign as it is null
    print(d);
    // Again trying to assign value to d
    d ? ? = a - b; // Value is not assign as it is not null
    print(d);
}
```

Output:

35

12

12

6. Logical Operators:

This class of operators contain those operators which are used to logically combine two or more conditions of the operands. It goes like this:

Operator Symbol	Operator Name	Operator Description
&&	And Operator	Use to add two conditions and if both are true than it will return true.
||	Or Operator	Use to add two conditions and if even one of them is true than it will return true.
!	Not Operator	It is use to reverse the result.

```
void main()
{
    int a = 5;
    int b = 7;

    // Using And Operator
    bool c = a > 10 && b < 10;
    print(c);

    // Using Or Operator
    bool d = a > 10 || b < 10;
    print(d);

    // Using Not Operator
    bool e = !(a > 10);
    print(e);
}
```

Output:

```
false

true

true
```

7. Conditional Operators:

This class of operators contain those operators which are used to perform comparison on the operands. It goes like this:

Operator Symbol	Operator Name	Operator Description
condition ? expersion1 : expersion2	Conditional Operator	It is a simple version of if-else statement. If the condition is true than expersion1 is executed else expersion2 is executed.
expersion1 ?? expersion2	Conditional Operator	If expersion1 is non-null returns its value else returns expression2 value.

```
void main()
{
    int a = 5;
    int b = 7;

    // Conditional Statement
    var c = (a < 10) ? "Statement is Correct, Geek" : "Statement is Wrong,
Geek";
    print(c);


    // Conditional statement
      int? n;
      // Warning: Operand of null-aware operation '??' has type 'int' which
excludes null.
      // For batter practice make both same type to avoid warning
      // var d = n ?? 10;
      var d = n ?? "n has Null value";
      print(d);

    // After assigning value to n
    n = 10;
      // we make it all ready null safe
```

```
    //d = n ? ? "n has Null value";
      d = n;
    print(d);
}
```

Output:

Statement is Correct, Geek

n has Null value

10

8. Cascade Notation Operators:

This class of operators allows you to perform a sequence of operation on the same element. It allows you to perform multiple methods on the same object. It goes like this:

Operator Symbol	***Operator Name***	***Operator Description***
..	*cascading Method*	*It is used to perform multiple methods on the same object.*

Switch Case:

switch-case statements are a simplified version of the nested if-else statements. Its approach is the same as that in Java.

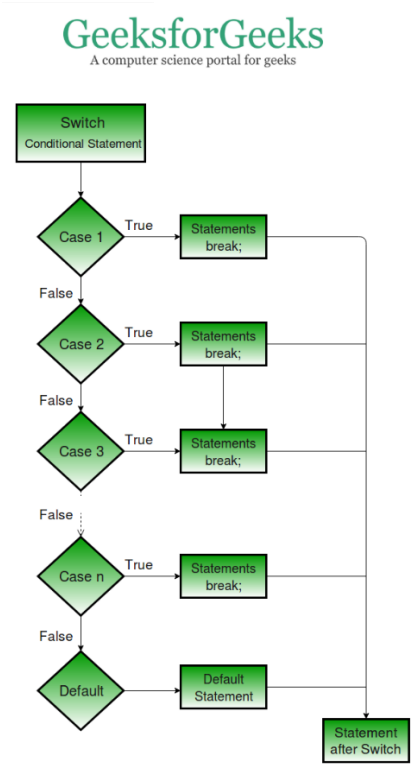
Syntax:

```
switch ( expression ) {
  case value1: {
    // Body of value1
  } break;
  case value2: {
    //Body of value2
  } break;
  .
  .
  .
  default: {
    //Body of default case
  } break;
}
```

The default case is the case whose body is executed if none of the above cases matches the condition.

Rules to follow in switch case:

1. There can be any number of cases. But values should not be repeated.
2. The case statements can include only constants. It should not be a variable or an expression.
3. There should be a flow control i.e break within cases. If it is omitted than it will show error.
4. The default case is optional.
5. Nested switch is also there thus you can have switch inside switch.



Example 1: Normal switch-case statement

```
void main()
{
   int gfg = 1;
   switch (gfg) {
   case 1: {
      print("GeeksforGeeks number 1");
   } break;
   case 2: {
      print("GeeksforGeeks number 2");
   } break;
   case 3: {
      print("GeeksforGeeks number 3");
   } break;
   default: {
      print("This is default case");
   } break;
   }
}
```

Output:

GeeksforGeeks number 1

Loops:

A looping statement in Dart or any other programming language is used to repeat a particular set of commands until certain conditions are not completed. There are different ways to do so. They are:

- for loop
- for… in loop
- for each loop
- while loop
- do-while loop

1 for loop

For loop in Dart is similar to that in Java and also the flow of execution is the same as that in Java.

Syntax:

```
for(initialization; condition; text expression){
   // Body of the loop
}
```

Control flow:

Control flow goes as:

1. initialization
2. Condition
3. Body of loop
4. Test expression

The first is executed only once i.e in the beginning while the other three are executed until the condition turns out to be false.

Example:

```
// Printing GeeksForGeeks 5 times
void main()
{
    for (int i = 0; i < 5; i++) {
        print('GeeksForGeeks');
    }
}
```

Output:

GeeksForGeeks

GeeksForGeeks

GeeksForGeeks

GeeksForGeeks

GeeksForGeeks

2 for...in loop

For...in loop in Dart takes an expression or object as an iterator. It is similar to that in Java and its execution flow is also the same as that in Java.

Syntax:

```
 for (var in expression) {
   // Body of loop
}
```

Example:

```
void main()
{
    var GeeksForGeeks = [ 1, 2, 3, 4, 5 ];
    for (int i in GeeksForGeeks) {
        print(i);  }
}
```

Output: :

```
1
2
3
4
5
```

3 for each … loop

The for-each loop iterates over all elements in some container/collectible and passes the elements to some specific function.

Syntax:

collection.foreach(void f(value))

Parameters:

- *f(value): It is used to make a call to the f function for each element in the collection.*

```
void main() {
  var GeeksForGeeks = [1,2,3,4,5];
  GeeksForGeeks.forEach((var num)=>
print(num));
}
```

Output:

```
1
2
3
4
5
```

4 while loop:

The body of the loop will run until and unless the condition is true.

Syntax:

```
while(condition){
   text expression;
   // Body of loop
}
```

Syntax:

```
while(condition){
   text expression;
   // Body of loop
}
```

Example:

```
void main()
{
    var GeeksForGeeks = 4;
    int i = 1;
    while (i <= GeeksForGeeks) {
        print('Hello Geek');
        i++;
    }
}
```

Class in Dart:

Class is the blueprint of objects and class is the collection of data members and data function means which include these fields, getter and setter, and constructor and functions.

Declaring class in Dart

Syntax:

```
class class_name {
   // Body of class
}
```

In the above syntax:

- **Class** is the keyword use to initialize the class.
- **class_name** is the name of the class.
- Body of class consists of fields, constructors, getter and setter methods, etc.

objects in Dart:

Objects are the instance of the class and they are declared by using new keyword followed by the class name.

Syntax:

var object_name = new class_name([arguments]);

In the above syntax:

- ***new** is the keyword use to declare the instance of the class*
- ***object_name** is the name of the object and its naming is similar to the variable name in dart.*
- ***class_name** is the name of the class whose instance variable is been created.*
- ***arguments** are the input which are needed to be pass if we are willing to call a constructor.*

Abstract Classes in Dart:

An *Abstract class* in Dart is defined as those classes which contain one or more than one abstract method (methods without implementation) in them. Whereas, to declare an abstract class we make use of the *abstract* keyword. So, it must be noted that a class declared abstract may or may not include abstract methods but if it includes an abstract method then it must be an abstract class.

Features of Abstract Class:

- A class containing an abstract method must be declared abstract whereas the class declared abstract may or may not have abstract methods i.e. it can have either abstract or concrete methods
- A class can be declared abstract by using **abstract** keyword only.
- A class declared as abstract can’t be initialized.
- An abstract class can be extended, but if you inherit an abstract class then you have to make sure that all the abstract methods in it are provided with implementation.

Flutter:

Flutter is a framework made bygoogle fordeveloping mobileapps, web ,desktop apps.

Flutter Scaffold

The Scaffold is a widget in Flutter used to implements the basic material **design visual layout structure**. It is quick enough to create a general-purpose mobile application and contains almost everything we need to create a functional and responsive Flutter apps. This widget is able to occupy the whole device screen. In other words, we can say that it is mainly responsible for creating a base to the app screen on which the child widgets hold on and render on the screen. It provides many widgets or APIs for showing Drawer, SnackBar, BottomNavigationBar, AppBar, FloatingActionButton, and many more.

The Scaffold class is a shortcut to set up the look and design of our app that allows us not to build the individual visual elements manually. It saves our time to write more code for the look and feel of the app. The following are the **constructor and properties** of the Scaffold widget class.

Flutter Container

The container in Flutter is a **parent widget that can contain multiple child widgets** and manage them efficiently through width, height, padding, background color, etc. It is a widget that combines common painting, positioning, and sizing of the child widgets. It is also a class to store one or more widgets and position them on the screen according to our needs. Generally, it is similar to a box for storing contents. It allows many attributes to the user for decorating its child widgets, such as using **margin**, which separates the container with other contents.

A container widget is same as ** ** tag in html. If this widget does not contain any child widget, it will fill the whole area on the screen automatically. Otherwise, it will wrap the child widget according to the specified height & width. It is to **note that** this widget cannot render directly without any parent widget. We can use Scaffold widget, Center widget, Padding widget, Row widget, or Column widget as its parent widget.

Why we need a container widget in Flutter?

If we have a widget that needs some background styling may be a color, shape, or size constraints, we may try to **wrap it in a container widget**. This widget helps us to compose, decorate, and position its child widgets. If we wrap our widgets in a container, then without using any parameters, we would not notice any difference in its appearance. But if we add any properties such as color, margin, padding, etc. in a container, we can style our widgets on the screen according to our needs.

Flutter Buttons

Buttons are the graphical control element that **provides a user to trigger an event** such as taking actions, making choices, searching things, and many more. They can be placed anywhere in our UI like dialogs, forms, cards, toolbars, etc.

Buttons are the Flutter widgets, which is a part of the material design library. Flutter provides several types of buttons that have different shapes, styles, and features.

Features of Buttons

The standard features of a button in Flutter are given below:

1. We can easily apply themes on buttons, shapes, color, animation, and behavior.
2. We can also theme icons and text inside the button.
3. Buttons can be composed of different child widgets for different characteristics.

Types of Flutter Buttons

Following are the different types of button available in Flutter:

- Flat Button
- Raised Button
- Floating Button
- Drop Down Button
- Icon Button
- Inkwell Button
- PopupMenu Button
- Outline Button

Flutter Icons

An icon is a **graphic image** representing an application or any specific entity containing meaning for the user. It can be selectable and non-selectable. **For example**, the company's logo is non-selectable. Sometimes it also contains a **hyperlink** to go to another page. It also acts as a sign in place of a detailed explanation of the actual entity.

Flutter provides an **Icon Widget** to create icons in our applications. We can create icons in Flutter, either using inbuilt icons or with the custom icons. Flutter provides the list of all icons in the **Icons class**. In this article, we are going to learn how to use Flutter icons in the application.

Icon Widget Properties

Flutter icons widget has different properties for customizing the icons. These properties are explained below:

Property	Descriptions
icon	It is used to specify the icon name to display in the application. Generally, Flutter uses material common actions and items.
color	It is used to specify the color of the icon.
size	It is used to specify the size of the icon in pixels. Usually, icons have equal height and width.
text Direction	It is used to specify to which direction the icon will be rendered.