Intro to app dev using Flutter

Dart Basics

Table of Contents

1) The Hello World Program

```dart
void main() {
  //My first Dart Program
  print("Hello World");
}
```

```
Console

  Hello World
```

- ➢ **main() function**:- The Starting point for program execution. This is where the program starts.
- ➢ **print() function**:- This prints the given object in the console.
- ➢ **Literals**:- String and Integer
  - ○ String Literal- “Hello World”
  - ○ Integer Literal- 3;
- ➢ **void**:- void means the function returns nothing.
- ➢ The program code should only be between these curly braces else it’ll thrown an error.
- ➢ **Comments**:- These are like notes added to a program to provide explanatory information.
  - ○ Single Line Comments:- //
  - ○ Multi Line Comments:- /* */
  - ○ These comments don’t affect the execution of the program in any way.
- ➢ After every command you need to put a ‘;’ (semi-colon) unlike python. This indicates the end of a command. If you forget to include it then throws an error.

2) Data Types

a) The Basic Data Types

```
void main() {
  int a = 56; //numbers without decimal points
  double b = 45.67; //basically integers with
decimal points
  print("int variable a= $a"); //$varname prints
the value of that variable
  print("double variable b= $b");
  num c = 35; //num can either be a integer or a
double
  print("num variable c= $c");
  c = 23.8; //doubles can also be assigned to num
  print("new value of num variable is c= $c");
  String str = "Hello World,My name is Aditya";
  print(str);
  bool d = (5 * 5 == 25); //can either have true or
false
  print("5x5 equals 25 is $d");
  d = (5 * 5 == 50);
  print("5x5 equals 50 is $d");
  var e = 23;
  print("value of the var e= $e");
  /*
  Once a var is initialized it's type of data
can't be changed
  var e=23;
  e=88; //this statement is valid as it's changed
only to a int
  e=34.5 //this statement would throw an error as
it's being changed to a double
  e="Hello"// this statement would throw an error
too
  */
  dynamic f = 56; //similar to var but it's type
can be changed
```

```
  print("value of dynamic f= $f");
  f = "Changed to a String now";
  print("new value of f= $f");
}
```

```
Console

  int variable a= 56
  double variable b= 45.67
  num variable c= 35
  new value of num variable is c= 23.8
  Hello World,My name is Aditya
  5x5 equals 25 is true
  5x5 equals 50 is false
  value of the var e= 23
  value of dynamic f= 56
  new value of f= Changed to a String now
```

- **Numbers**: ‘num’ keyword includes both integers and doubles
  - Integers:- ‘int’ – these are ordinary numbers without a decimal point.
  - Doubles:- ‘double’ – these are basically integers with a decimal point.
- **Strings**:- ‘String’ – note that String starts with a capital ‘S’. Dart is a case sensitive language.
  - “Hello World, My name is Aditya.”
- **bool**:- They could have only two values, either ‘true’ or ‘false’
  - (5*5==25) results in true
  - (5*5==50) results in false
- **var**:- ‘var’ stands for variable, let’s say for example you are fetching some data from the server, and you have no idea what type of data you are going to get (either some text or some numbers) then we use var.
  - Once a var is initialized it’s data type cannot be changed.
- **dynamic**:- ‘dynamic’ is basically the same as ‘var’ but it’s data type can be changed even after it is already initialized.

b) Collection Data Types

i) Lists

```
void main() {
  List marks = [56, 77, 37, 41, 91];
  /*
  When no type is specified List is List<dynamic>
  So Technically even Strings could be added to
the above List
  */
  marks.add("Ram"); //built-in method to add
elements to a List
  print(marks);
  List<int> marksList = [56, 77, 37, 41, 91];
  /*
  The type is specified here
  <int>
  marksList.add("Ram"); //this Would throw an
error
  */
  print(marksList);
  print("The first element is ${marksList[0]}");
  print("The second element is ${marksList[1]}");
  //use ${varname[arrindex]} when involving arrays
or classes
  List<String> names = ["Aditya", "Sneha",
"Nithin", "Vignesh"];
  print(names[2]);
}
```

```
Console

  [56, 77, 37, 41, 91, Ram]
  [56, 77, 37, 41, 91]
  The first element is 56
  The second element is 77
  Nithin
```

- **Lists**: These are technically arrays
  - List marks=[56, 77, 37, 41, 91];
  - By default the type of List is dynamic (List <dynamic>)
  - List<int> marks=[56, 77, 37, 41, 91]; ensures that Strings cannot be added to these lists. (A good programming practice generally).
  - List<String> names=[“Aditya”, ”Sneha”, ”Nithin”, ”Vignesh”];
  - names[2] would point to “Nithin” .
  - Elements are zero based which means the first element is names[0], second is names[1], third is names[2] and so on.
  - List uses square brackets. []

ii) Sets

```dart
void main() {
  List<String> namesList = ["Aditya", "Sneha",
"Nithin", "Vignesh", "Aditya"];
  print(namesList);
  Set<String> namesSet = {"Aditya", "Sneha",
"Nithin", "Vignesh", "Aditya"};
  //elemets are unique in a set
  print(namesSet);
  print(namesSet.elementAt(3));
  //the element of a set can be accessed by using
a built-in method .elementAt(index)
}
```

```
Console

  [Aditya, Sneha, Nithin, Vignesh, Aditya]
  {Aditya, Sneha, Nithin, Vignesh}
  Vignesh
```

- **Sets**:- Sets are basically Lists but the data is unique and the elements cannot be duplicated.
  - Sets use curly brackets {}
  - Set names={ “Aditya”, ”Sneha”, ”Nithin”, ”Vignesh”, “Aditya”};
  - The element “Aditya” occurs only once and does not get repeated whereas in a List it would get repeated.
  - The built-in method .elementAt(int index) is used to access an element from a set given it’s position.

iii) Maps

```dart
void main() {
  Map<String, int> students = {"Aditya": 55,
"Dhiganth": 99};
  print(students);
  print(students["Aditya"]);
  //here "Aditya" is the key and it's used to
access it's value
  print(students["Dhiganth"]);
  /*
  You cannot print the key given a value
  print(students[99]); //this would print null
because that key does not exist
  This is Similar to a Dictionary in Python
  */
  Map<int, int> squares = {2: 4, 3: 9, 4: 16, 5:
25};
  print(squares);
  print(squares[5]);
}
```

```
Console

{Aditya: 55, Dhiganth: 99}
55
99
{2: 4, 3: 9, 4: 16, 5: 25}
25
```

➢ Maps:- Maps are a collection of key-value pairs.
- Map<String,int> students={“Aditya”:55,”Dhiganth”:99};
  print(students);
  print(students[“Aditya”]);
  print(students[“Dhiganth”]);
- students[99] won’t work. The element on the left is the key and the element on the right of ‘:’ is the value.
- Map<int,int> squares={2:4,3:9,4:16,5:25};
  print(squares);
  print(squares[5]);

3) Naming a variable

Variables are later used in the program to access their data. Hence there are certain things to be kept in mind before naming them.

i) You cannot use any of the **reserved keywords** like “int”, “String” etc.
Eg:- int int=88; - throws an error because a reserved keyword was used in place of the name of a variable
Some Reserved Keywords are:-

abstract [2]	else	import [2]	show [1]
as [2]	enum	in	static [2]
assert	export [2]	interface [2]	super
async [1]	extends	is	switch
await [3]	extension [2]	late [2]	sync [1]
break	external [2]	library [2]	this
case	factory [2]	mixin [2]	throw
catch	false	new	true
class	final	null	try
const	finally	on [1]	typedef [2]
continue	for	operator [2]	var
covariant [2]	Function [2]	part [2]	void
default	get [2]	required [2]	while
deferred [2]	hide [1]	rethrow	with
do	if	return	yield [3]
dynamic [2]	implements [2]	set [2]	

ii) Cannot use **Variable names** which are already used.

For eg:- int a=56;

String a=”hi”;

This throws an error because ‘a’ has already been used.

iii) Unlike other programming languages like C++, C , JAVA etc. Dart doesn’t have private, public, protected identifiers.

If you want a **variable to be private** or hidden from other files then you have to add an ‘_’(underscore) before the name of the variable. So that the data of this variable cannot be accessed outside this file.

4) Operators

These are symbols which perform specific operations on variables.

a) Arithmetic

```dart
void main() {
  int a = 16;
  int b = 5;
  print("$a + $b = ${a + b}");
  print("$a - $b = ${a - b}");
  print("$a * $b = ${a * b}");
  print("$a / $b = ${a / b}");
  print("$a % $b = ${a % b}");
  /*
  whenever using an expression in a string use curly
braces around them
  ${var * var2}
  */
  int result = (a / b).toInt();//printing the quotient
without decimal places
  //This is how you typecast in dart
  print(result);

}
```

Console

```
16 + 5 = 21
16 - 5 = 11
16 * 5 = 80
16 / 5 = 3.2
16 % 5 = 1
3
```

➢ **Arithmetic:- +, - , *, /, %.**
- This is used to perform mathematical operations between two variables and always returns a number(int or double) back
- * is the multiplication symbol
- / returns the quotient between two variables
- % returns the remainder between two variables

b) Relational

```
void main() {
  int a = 16;
  int b = 5;
  print("$a is less than $b is ${a < b}");
  print("$a is more than $b is ${a > b}");
  print("$a is equal to $b is ${a == b}");
  print("$a is not equal to $b is ${a != b}");
  print("$a is less than or equal to $b is ${a <= b}");
  print("$a is more than or equal to $b is ${a >= b}");
}
```

Console

```
16 is less than 5 is false
16 is more than 5 is true
16 is equal to 5 is false
16 is not equal to 5 is true
16 is less than or equal to 5 is false
16 is more than or equal to 5 is true
```

➢ **Relational Operators :- < , >, ==, !=, >=, <=**
- These always return Boolean values and are used mostly in conditional statements
- a < b:- “a less than b”
- a > b:- “a more than b”
- a == b:- “a equal to b”
- a != b:- “a not equal to b”

- a <= b:- “a less than or equal to b”
- a >= b:- “a more than or equal to b”

c) Unary

```dart
void main() {
  int a = 5;
  print("Initial Value of a is $a");
  print("Pre-Increment of a is ${++a}");
  print("Post-Increment of a is ${a++}");
  print("Value of a after Post-Increment is $a");
  print("Pre-Decrement of a is ${--a}");
  print("Post-Decrement of a is ${a--}");
  print("Value of a after Post-Decrement is $a");
}
```

```
Console

  Initial Value of a is 5
  Pre-Increment of a is 6
  Post-Increment of a is 6
  Value of a after Post-Increment is 7
  Pre-Decrement of a is 6
  Post-Decrement of a is 6
  Value of a after Post-Decrement is 5
```

➢ **Unary Operators:- ++, –**

- There are two types:- Prefix(Before the variable) and Postfix(After the variable).
- int a=5;
- print(a++);still prints 5 but the value is changed to 6 after printing.
- print(++a); value is first changed to 6 and then it prints 6.
- ++a; is equivalent to a=a+1;

d) Assignment

```
void main() {
  int a = 5;//here integer literal 5 is assigned to
variable a
  print("Initial value of a is $a");
  a += 2;
  print("Value of a after += assignment operator is $a");
  a *= 5;
  print("Value of a after *= assignment operator is $a");
  a -= 10;
  print("Value of a after -= assignment operator is $a");
  a ~/= 5;// ~/ is the same as / but instead of double it
returns int
  print("Value of a after ~/= assignment operator is $a");
}
```

```
Console

  Initial value of a is 5
  Value of a after += assignment operator is 7
  Value of a after *= assignment operator is 35
  Value of a after -= assignment operator is 25
  Value of a after ~/= assignment operator is 5
```

- **Assignment Operators:- =, +=, *=, /=, %=**
  - int a=5; The equal-to(=) symbol here denotes that the literal 5 is assigned to the integer variable ‘a’.
  - a+=5; is equivalent to a=a+5; This is called shorthand notation and any other operator could be used in the same way.

e) Logical

```dart
void main() {
  print("Not Operator");
  print("Not true is ${!true}");
  print("Not false is ${!false}");
  print("\nOr Operator");
  print("false or false is ${false || false}");
  print("false or true is ${false || true}");
  print("true or false is ${true || false}");
  print("true or true is ${true || true}");
  print("\nAnd Operator");
  print("false and false is ${false && false}");
  print("false and true is ${false && true}");
  print("true and false is ${true && false}");
  print("true and true is ${true && true}");
}
```

Console

```
Not Operator
Not true is false
Not false is true

Or Operator
false or false is false
false or true is true
true or false is true
true or true is true

And Operator
false and false is false
false and true is false
true and false is false
true and true is true
```

- **Logical:- &&, ||, !**
  - These are generally used in conditional statements between two relational operations (two Boolean expressions) .
  - && stands for and
  - || stands for or
  - ! stands for not and it inverts the boolean value

5) Control Flow

Using Conditional Statements in Dart:

Program to Check if a person is eligible to Vote:-

```
void main() {
  int age = 19;
  if (age >= 18) {
    print("Person is eligible to vote");
  } else {
    print("Person is not eligible to vote");
  }
}
```

```
Console

  Person is eligible to vote
```

Try Changing the value of age to see the different outputs possible.

Program to find the greatest of three numbers (uses else if)

```
void main() {
  int a = 5, b = 6, c = 7;
  if (a > b && a > c) {
    print("$a is the greatest number");
  } else if (b > c) {
```

```
    print("$b is the greatest number");
  } else {
    print("$c is the greatest number");
  }
}
```

Console

```
7 is the greatest number
```

Try changing the values of a, b and c to notice the difference.

6) Loops

Program to print the first 10 natural numbers

Using For Loop

```
void main(){
  for(int i=1;i<=10;i++){
    print(i);
  }
}
```

Using while loop

```
void main(){
  int i=1;
  while(i<11){
    print(i);
    i++;
  }
}
```

```
Console

1
2
3
4
5
6
7
8
9
10
```

A Demonstration of Break statement:-

```dart
void main(){
  print("Loop starts");
  for(int i=1;i<=10;i++){
    print(i);
    if(i==6){
      print("6 is found and the loop is exiting");
      break;
    }
  }
  print("Outside the loop");
}
```

```
Console

  Loop starts
  1
  2
  3
  4
  5
  6
  6 is found and the loop is exiting
  Outside the loop
```

A Program demonstrating switch case:-

```dart
void main() {
  String grade = "A";
  switch (grade) {
    case "A":
      print("Excellent");
      break;
    case "B":
      print("Good");
      break;
    case "C":
      print("Average");
      break;
    case "D":
      print("Below Average");
      break;
    case "F":
      print("Fail");
      break;
    default:
      print("Invalid Grade");
  }
}
```

```
Console

  Excellent
```

Try changing the value of grade variable to see different possible outputs.

7) Functions

Function without parameters

```dart
void main() {
  print("Calling a function without parameters");
  exampleFunction();
  print("Function has finished Executing");
}

void exampleFunction() {
  print("Printing From the Function");
}
```

```
Console

  Calling a function without parameters
  Printing From the Function
  Function has finished Executing
```

Function with parameters

```dart
void main() {
  print("Calling a function with parameters");
  checkeligibility(17);
  checkeligibility(24);
}
```

```dart
void checkeligibility(int age) {
  if (age >= 18) {
    print("Person is eligible to vote");
  } else {
    print("Person is not eligible to vote");
  }
}
```

Console

```
Calling a function with parameters
Person is not eligible to vote
Person is eligible to vote
```

Optional Parameters

These Parameters are optional which means they need not be initialized. They generally have a default value or are nullable.

There are two types of Optional Parameters:-

i) Named Parameter

They are used in between curly braces {}

```dart
void main() {
  print("Calling a function with named parameters");
  checkeligibility(21,name: "Aditya");
  print("Calling the function without providing the named
     parameter");
  checkeligibility(16);
}

void checkeligibility(int age, {String name = "Person"})
     {
  if (age >= 18) {
    print("$name is eligible to vote");
  } else {
```

```
    print("$name is not eligible to vote");
  }
}
```

```
Console

  Calling a function with named parameters
  Aditya is eligible to vote
  Calling the function without providing the named parameter
  Person is not eligible to vote
```

ii) Positional Parameter

They are used in between square brackets []

```
void main() {
  print("Calling a function with positional parameters");
  checkeligibility(21, "Aditya");
  print("Calling the function without providing the
     positional parameter");
  checkeligibility(16);
}

void checkeligibility(int age, [String name = "Person"]) {
  if (age >= 18) {
    print("$name is eligible to vote");
  } else {
    print("$name is not eligible to vote");
  }
}
```

```
Console

  Calling a function with positional parameters
  Aditya is eligible to vote
  Calling the function without providing the positional parameter
  Person is not eligible to vote
```

Functions with a return type

```dart
void main() {
  print("Calling a function with a return type");
  int result = cube(3) + cube(5);
  print("Sum of Cubes of 3 and 5 is $result");
}

int cube(int num) {
  int value = num * num * num;
  return value;
}
```

```
Console

  Calling a function with a return type
  Sum of Cubes of 3 and 5 is 152
```

8) Classes and Objects

Class is a way to create a user defined data type (like int, String)

Formal definition:- Blueprint for creating an object, it has a state(member variables) and behavior (functions and methods).

```dart
void main() {
  Student a = Student("Aditya", 21, 195252316, 63,84, 72);
  Student b = Student("Dhiganth", 22, 180050203, 71, 86, 99);
```

```
  b.displaydetails();
  a.displaydetails();
  int totalofa = a.total();
  print("${a.name} got a total of $totalofa");
}

class Student {
  String name;
  int age;
  int rollno;
  int maths, physics, chem;
  Student(
      this.name, this.age, this.rollno, this.maths, this.chem,
this.physics);

  void displaydetails() {
    print(
        "$name($rollno) is $age years old and their total mark is
${total()}");
  }

  int total() {
    return maths + physics + chem;
  }
}
```

Console

```
Dhiganth(180050203) is 22 years old and their total mark is 256
Aditya(195252316) is 21 years old and their total mark is 219
Aditya got a total of 219
```