UNIT -2

Dart Programming

❖ Dart Programming

Dart is an open-source, general-purpose, object-oriented programming language with C-style syntax developed by Google in 2011. The purpose of Dart programming is to create a frontend user interfaces for the web and mobile apps. It is under active development, compiled to native machine code for building mobile apps, inspired by other programming languages such as Java, JavaScript, C#, and is Strongly Typed. Since Dart is a compiled language so you cannot execute your code directly; instead, the compiler parses it and transfer it into machine code.

❖ example simple Dart programming.

```
void main(){
  print('hello world');
  stdout.write('enter your name ::');
  var name=stdin.readLineSync();
  print("welcome,$name");
}
```

❖ Data Type

Dart is a Strongly Typed programming language. It means, each value you use in your programming language has a type either string or number and must be known when the code is compiled. Here, we are going to discuss the most common basic data types used in the Dart programming language.

Data Type	Example	Descriptions
String	String myName = 'javatpoint';	It holds text. In this, you can use single or double quotation marks. Once you decide the quotation marks, you should have to be consistent with your choice.

num, int, double	int age = 25; double price = 125.50;	The num data type stands for a number. Dart has two types of numbers: o Integer (It is a number without a decimal place.) Double (It is a number with a decimal place.)

Boolean	bool var_name = true;
	Or
	bool var_name = false;
		It uses the bool keyword to represents the Boolean value true and false.

Object	Person = Person()	Generally, everything in Dart is an object (e.g., Integer, String). But an object can also be more complex.

❖ Variables and Functions

- Variables are the namespace in memory that stores values. The name of a variable is called as **identifiers**. They are the data containers, which can store the value of any type

  - For example:

```
void main(){
 var name="rahu";
   name="rahul";
}
```

- **dynamic. Variables** can be also declared using the dynamic keyword in place of the var keyword.
  - For example:

```
void main(){
  dynamic section;
  section='rahul';
  section=7;
  print(section);
  }
```

- **Functions** are another core feature of any programming language. Functions are a set of statements that performs a specific task. They are organized into the logical blocks of code that are readable, maintainable, and reusable. The function declaration contains the function name, return type, and parameters. The following example explains the function used in Dart programming.

```
void main(){
 myfun();
}
void myfun(){
  print("this is my function");
}


void main(){
  print(add());
}
int add(){
  int a=10;
  int b=20;
  int sum=a+b;
  return sum;
}

void main(){
  print(add(2,3));
  print(add(40,10));  // fucutional argument
}

int add(int no1,int no2){  //Actual argument
  return no1+no2;

}
```

➢ Final and const Keyword

❖ Final keyword

- A variable with the final keyword will be initialized at runtime and can only be assigned for a single time.

```
void main(){
  final string name="rahul";
  name="rahul";
 }
```

❖ Const Keyword

- **"const"** variables must be assigned a value at the time of their declaration, and their value must be a compile-time constant**.**

```
void main(){
  const name="rahul";
  name="rahul"
 }
```

➢ Decision making and loops

- for loops
- while and do while loops

❖ for loops

- You can iterate with the standard for loop

```
void main(){
  for (int i = 0; i < 5; i++) {
  print('Hello World');
}
}
```

❖ While loops

- A while loop evaluates the condition before the loop

```
void main(){
  int i=0;
  while(i<5){
    print('Hello world');
    i++;
  }
}
```

❖ do while loops

- A do-while loop evaluates the condition after the loop

```
void main(){
  int i=0;
  do{
    print('Hello world');
    i++;
  }while(i<5);
}
```

➢ **Break** statement And **Continue** statement

❖ **Break** statement

➢ The break statement is used to take terminate the loop. Using break in a loop causes the program to exit the loop. Following is an example of the break statement.

```
void main(){
  int i;
  for(i=0; i<5; i++){
    if(i==3){
      break;
    }
    print(i);
  }
}
```

❖ **Continue** statement

➢ The continue statement skips the subsequent statements in the current iteration and takes the control back to the beginning of the loop.

```
void main(){
  int i;
  for(i=0; i<5; i++){
    if(i==3){
      continue;
    }
    print(i);
  }
}
```

❖ **Comments**

➢ Dart provides three kinds of comments

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

➢ **Single-line Comments**

▪ We can apply comments on a single line by using the // (double-slash). The single-line comments can be applied until a line break.

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

➢ Multi-line Comments

- Sometimes we need to apply comments on multiple lines; then, it can be done by using /*…..*/. The compiler ignores anything that written inside the /*…*/, but it cannot be nested with the multi-line comments. Let's see the following example.

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

➢ Documentation Comments

- The document comments are used to generate documentation or reference for a project/software package. It can be a single-line or multi-line comment that starts with /// or /*. We can use /// on consecutive lines, which is the same as the multiline comment. These lines ignore by the Dart compiler expect those which are written inside the curly brackets. We can define classes, functions, parameters, and variables. Consider the following example.

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

➢ Operators

- Dart supports the following types of operators.

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

➢ Arithmetic Operators

- Arithmetic Operators are the most common operators that are used to perform addition, subtraction, multiplication, divide, etc.

```
void main(){
  print("Example of Arithmetic 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}");
}
```

➢ Assignment Operators

- Assignment operators are used to assigning value to the variables. We can also use it combined with the arithmetic operators. The list of assignment operators is given below. Suppose a holds value 20 and b holds 10.

```
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}");
}
```

➢ Relational Operators

- Relational operators or Comparison operators are used to making a comparison between two expressions and operands. The comparison of two expressions returns the Boolean true and false. Suppose a holds 20 and b hold 10 then consider

```
void main() {
var a = 30;
var b = 20;
print("The example of Relational Operator");
var res = a>b;
print("a is greater than b:$res");
var res0 = a<b;
print("a is less than b:$res ");
var res1 = a>=b;
print("a is greater than or equal to b:$res ");
var res2 = a<=b;
print("a is less than and equal to b:$res ");
var res3 = a!= b;
print("a is not equal to  b:$res ");
var res4 = a==b;
print("a is  equal to  b:$res ");
}
```

➢ Type test Operators

- The Type Test Operators are used to testing the types of expressions at runtime.

Sr.	Operator	Description
1	Is	It returns TRUE if the object has specified type.
2	is!	It returns TRUE if the object has not specified type.

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

- **Logical Operators**
- The Logical Operators are used to evaluate the expressions and make the decision.

```
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);
}
```

- **Bitwise Operator**
- The Bitwise operators perform operation bit by bit on the value of the two operands. Following is the table of bitwise operators.

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

  var a  = 25;
  var b = 20;
  var c = 0;
  // Bitwise AND Operator
  print("a & b = ${a&b}");
  // Bitwise OR Operator
  print("a | b = ${a|b}");
  // Bitwise XOR
  print("a ^ b = ${a^b}");
  // Complement Operator
  print("~a = ${(~a)}");
  // Binary left shift Operator
```

```
  c = a <<2;
  print("c<<1= ${c}");

  // Binary right shift Operator
  c = a >>2;
  print("c>>1= ${c}");
}
```

➢ Conditional Operators

- The Conditional Operator is same as if-else statement and provides similar functionality as conditional statement. It is the second form of **if-else statement**. It is also identified as **"Ternary Operator"**.

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

➢ Casecade notation(..) Operators

- The Cascade notation Operators (..) is used to evaluate a series of operation on the same object. It is an identical as the method chaining that avoids several of steps, and we don't need store results in temporary variables**.**

```
class Sample{
   int a=0;
   String name="";
}
void main(){
  Sample obj=Sample()
  ..a=10
  ..name="rahul";
  print("value ${obj.a} and name is ${obj.name}");

}
```