Dart Operators:

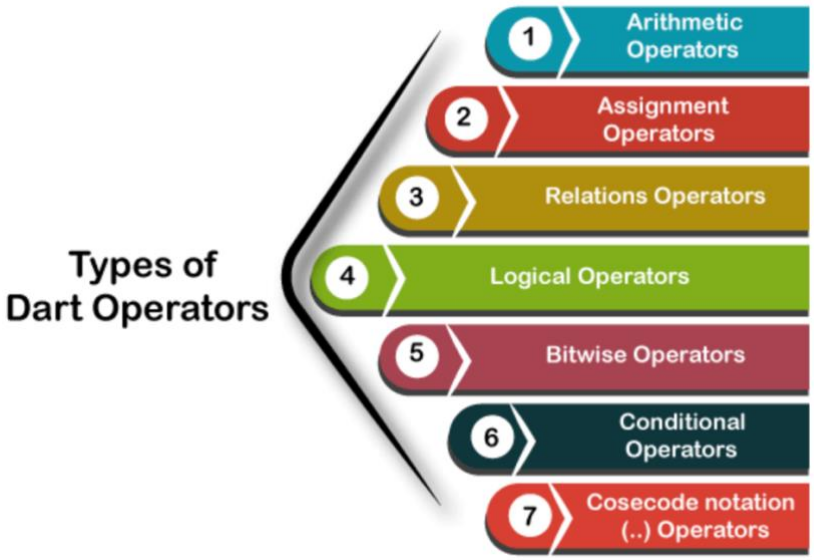
An operator is a symbol that is used to manipulating the values or performs operations on its operand. The given expression: 5+4, in this expression, 5 and 4 are operands and "+" is the operator.

Dart provides an extensive set of built-in operators to accomplish various types of operations. Operators can be unary or binary, which means unary take only on operand and binary take two operands with operators. There are several types of operators. Following is the list of Dart Operators.

Types of 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



Dart Arithmetic Operators

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

Dart supports the usual arithmetic operators, as shown in the following table.

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

Assignment Operator

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.

=, =+, -=, *=, /= etc.

Compound assignment operators such as `+=` combine an operation with an assignment.

`=`	`*=`	`%=`	`>>>=`
`+=`	`/=`	`<<=`	`&=`
`-=`	`~/=`	`>>=`	

Relational Operator

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 the following table.

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.

Dart Type Test Operators

The Type Test Operators are used to testing the types of expressions at runtime. Consider the following table.

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

Let's understand the following example.

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

```
(employee as Person).firstName = 'Bob';
```

If you aren’t sure that the object is of type T, then use is T to check the type before using the object.

```
if (employee is Person) {
  // Type check
  employee.firstName = 'Bob';
}
```

Dart Logical Operators

The Logical Operators are used to evaluate the expressions and make the decision. Dart supports the following logical operators.

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

Dart Bitwise Operators

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

Let's understand 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

Dart 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"**. The syntax is given below.

Syntax 1 -

```
condition ? exp1 : exp2
```

If the given condition is TRUE then it returns exp1 otherwise exp2.

Syntax 2 -

```
exp1 ?? expr2
```

If the exp1 is not-null, returns its value, otherwise returns the exp2's value.

Let's understand the following example.

Example - 1

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

Output:

```
20
```

Cascade notation

Cascades (.., ?..) allow you to make a sequence of operations on the same object. In addition to function calls, you can also access fields on that same object. This often saves you the step of creating a temporary variable and allows you to write more fluid code.

Consider the following code:

```
var paint = Paint()
  ..color = Colors.black
  ..strokeCap = StrokeCap.round
  ..strokeWidth = 5.0;
```

The constructor, Paint(), returns a Paint object. The code that follows the cascade notation operates on this object, ignoring any values that might be returned.

The previous example is equivalent to this code:

```
var paint = Paint();
paint.color = Colors.black;
paint.strokeCap = StrokeCap.round;
paint.strokeWidth = 5.0;
```