Dart Basic Syntax

Dart Printing and String Interpolation

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

// Output Will Be --> My name is Peter My roll number is 24
```

Dart Data Types

Dart supports the following built-in Data types.

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

Dart Number

The Darts Number is used to store the numeric values. The number can be two types - integer and double.

- **Integer** - Integer values represent the whole number or non-fractional values. An integer data type represents the 64-bit non-decimal numbers between -2^63 to 2^63. A variable can store an unsigned or signed integer value. The example is given below -

```
int marks = 80;
```

- **Double** - Double value represents the 64-bit of information (double-precision) for floating number or number with the large decimal points. The double keyword is used to declare the double type variable.

```
double pi = 3.14;
```

Dart Strings

A string is the sequence of the character. If we store the data like - name, address, special character, etc. It is signified by using either single quotes or double quotes. A Dart string is a sequence of UTF-16 code units.

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

Dart Boolean

The Boolean type represents the two values - true and false. The bool keyword uses to denote Boolean Type. The numeric values 1 and 0 cannot be used to represent the true or false value.

```
bool isValid = true;
```

Dart Lists

In Dart, The list is a collection of the ordered objects (value). The concept of list is similar to an array. An array is defined as a collection of the multiple elements in a single variable. The elements in the list are separated by the comma enclosed in the square bracket[]. The sample list is given below.

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

Dart Maps

The maps type is used to store values in key-value pairs. Each key is associated with its value. The key and value can be any type. In Map, the key must be unique, but a value can occur multiple times. The Map is defined by using curly braces ({}), and comma separates each pair.

```
var student = {'name': 'Joseph',  'age':25, 'Branch': 'Computer Science'}
```

Dart Runes

As we know that, the strings are the sequence of Unicode UTF-16 code units. Unicode is a technique which is used to describe a unique numeric value for each digit, letter, and symbol. Since Dart Runes are the special string of Unicode UTF-32 units. It is used to represent the special syntax.

For example - The special heart character ♥ is equivalent to Unicode code \u2665, where \u means Unicode, and the numbers are hexadecimal integer. If the hex value is less or greater than 4 digits, it places in a curly bracket ({}). For example - An emoji 😀 is represented as \u{1f600}. The example is given below.

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

OutPut

```
♥
😀
```

Dart Symbols

The Dart Symbols are the objects which are used to refer an operator or identifier that declare in a Dart program. It is commonly used in APIs that refers to identifiers by name because an identifier name can changes but not identifier symbols.

Dart Dynamic Type

Dart is an optionally typed language. If the variable type is not specified explicitly, then the variable type is dynamic. The dynamic keyword is used for type annotation explicitly.

Final and const

When we do not want to change a variable in the future then we use final and const. It can be used in place of var or in addition to a type. A final variable can be set only one time where the variable is a compile-time constant. The example of creating a final variable is given below.

```
final name = 'Ricky';                          // final variable without type annotation.
final String msg = 'How are you?';      // final variable with type annotation.
```

If we try to change these values then it will throw an error.

```
name = 'Roger';                                 // Error: Final variable can't be changed.
```

The **const** is used to create compile-time constants. We can declare a value to compile-time constant such as number, string literal, a const variable, etc.

```
const a = 1000;
```

The const keyword is also used to create a constant value that cannot be changed after its creation.

```
var f = const[];
```

If we try to change it, then it will throw an error.

```
f = [12];    //Error, The const variable cannot be change
```

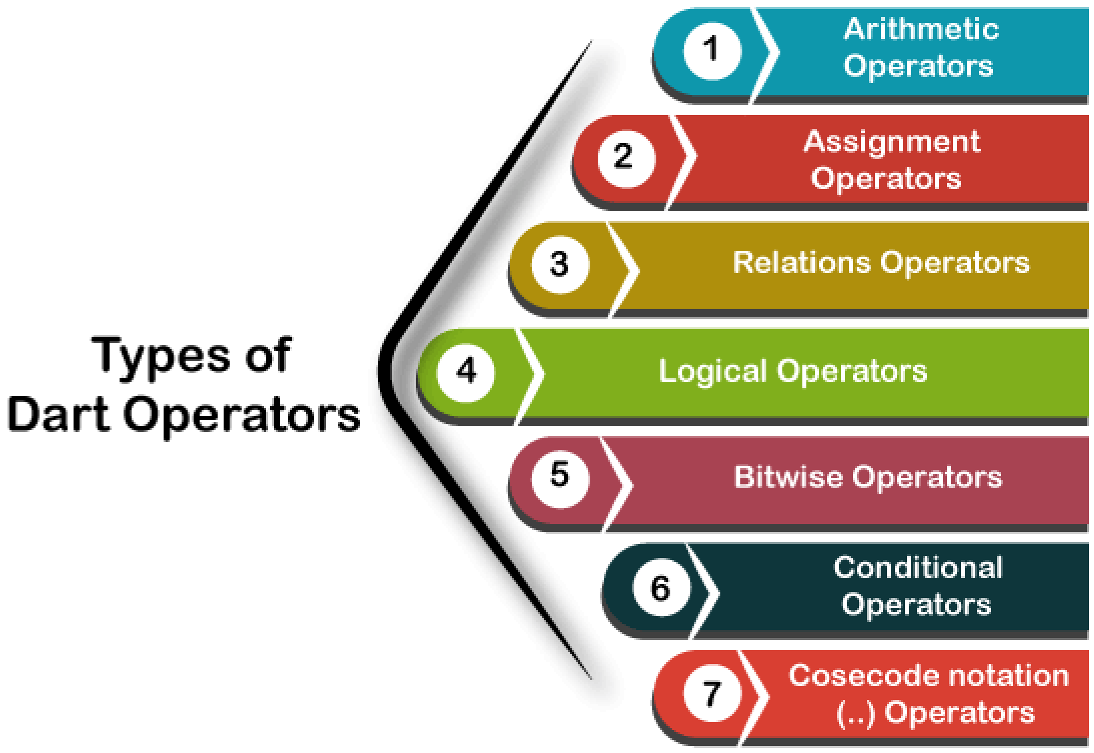
We will learn more about const in upcoming tutorials.

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



```
```

```
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
```

```
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
```

```
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. toString());  // We will learn the toString in
next tutorial

var res0 = a<b;
print("a is less than b: "+res0. toString());

var res1 = a>=b;
print("a is greater than or equal to b: "+res1. toString());

var res2 = a<=b;
print("a is less than and equal to b: "+res2. toString());

var res3 = a!= b;
print("a is not equal to  b: "+res3. toString());

var res4 = a==b;
print("a is  equal to  b: "+res4. toString());
}
```

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

Sr.	Operator	Description
1.	as	It is used for typecast.
2.	is	It returns TRUE if the object has specified type.

Sr.	Operator	Description
3.	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 );
}
```

Output

```
true
false
```

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.

```
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 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
```

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

```
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}");
  }
```

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 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);
}
// Output Will Be --> 20
```

```
void main() {
   var a = 30;
   var output = a > 42 ? "value greater than 10":"value lesser than equal to 30";
   print(output);
}

// Output Will Be --> value lesser than or equal to 30
```

Dart Cascade 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.