

# Dart – Basics and Data Types

Tushar B. Kute,  
<http://tusharkute.com>



# Getting Started

- Dart is easy to learn if you know any of Java, C++, JavaScript, etc.
- The simplest "Hello World" program gives the idea of the basic syntax of the programming language. It is the way of testing the system and working environment.
- There are several ways to run the first program, which is given below:
  - Using Command Line
  - Running on Browser
  - Using IDE

# Hello World

```
void main() {  
    print("Hello World!");  
}
```

# Identifiers

- Identifiers are the name which is used to define variables, methods, class, and function, etc.
- An Identifier is a sequence of the letters([A to Z],[a to z]), digits([0-9]) and underscore(\_), but remember that the first character should not be a numeric.

# Identifiers

- The first character should not be a digit.
- Special characters are not allowed except underscore ( `_` ) or a dollar sign ( `$` ).
- Two successive underscores ( `__` ) are not allowed.
- The first character must be alphabet (uppercase or lowercase) or underscore.
- Identifiers must be unique and cannot contain whitespace.
- They are case sensitive. The variable name **Tushar** and **tushar** will be treated differently.

# Printing and String Interpolation

- The `print()` function is used to print output on the console, and `$expression` is used for the string interpolation. Below is an example.

```
void main()
{
    var name = "Tushar";
    var marks = 78.56;
    print("My name is ${name} My marks are ${marks}");
}
```

# Semicolon

- The semicolon is used to terminate the statement that means, it indicates the statement is ended here. It is mandatory that each statement should be terminated with a semicolon(;).
- We can write multiple statements in a single line by using a semicolon as a delimiter. The compiler will generate an error if it is not use properly.
- Example -  

```
var msg1 = "Hello World!";  
var msg2 = "How are you?"
```

# Whitespace and Line Breaks

- The Dart compiler ignores whitespaces. It is used to specify space, tabs, and newline characters in our program.
- It separates one part of any statement from another part of the statement.
- We can also use space and tabs in our program to define indentation and provide the proper format for the program.
- It makes code easy to understand and readable.



# Block

- The block is the collection of the statement enclosed in the curly braces. In Dart, we use curly braces to group all of the statements in the block.
- Consider the following syntax.

- Syntax:

```
{ //start of the block  
    //block of statement(s)  
} // end of the block
```

# Comments

- Comments are the set of statements that are ignored by the Dart compiler during the program execution. It is used to enhance the readability of the source code.
- Generally, comments give a brief idea of code that what is happening in the code.
- We can describe the working of variables, functions, classes, or any statement that exists in the code.
- Programmers should use the comment for better practice.

# Comments

- Dart provides three kinds of comments
  - Single-line Comments
  - Multi-line Comments
  - Documentation Comments

# Single-line Comment

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

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

# Multi-line Comment

- 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.

- Example -

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

# Documentation Comment

- 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.

- Syntax

```
///This
```

```
///is
```

```
///a example of
```

```
/// multiline comment
```

# Keywords

|          |          |          |            |         |
|----------|----------|----------|------------|---------|
| 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  |          |            |         |

# Data Types

- The data types are the most important fundamental features of programming language.
- In Dart, the data type of the variable is defined by its value.
- The variables are used to store values and reserve the memory location.
- The data-type specifies what type of value will be stored by the variable. Each variable has its data-type.
- The Dart is a static type of language, which means that the variables cannot modify.



# Data Types

- Dart supports the following built-in Data types.
  - Number
  - Strings
  - Boolean
  - Lists
  - Maps
  - Runes
  - Symbols

# Numbers

- 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. Ex.  
`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;`

# 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 MITU";  
print("सुस्वागतम");
```

# 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;`

# List

- 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]
```

# Map

- 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': 'Rajesh', 'age':22, 'Branch':  
'Statistics'}
```

# 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}`.

# Example

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



# Symbol

- 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.

# 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.

# Variable Default Value

- While declaring the variable without initializing the value then the Dart compiler provides default value (Null) to the variable.
- Even the numeric type variables are initially assigned with the null value.
- Let's consider the following example.

```
int count;
```

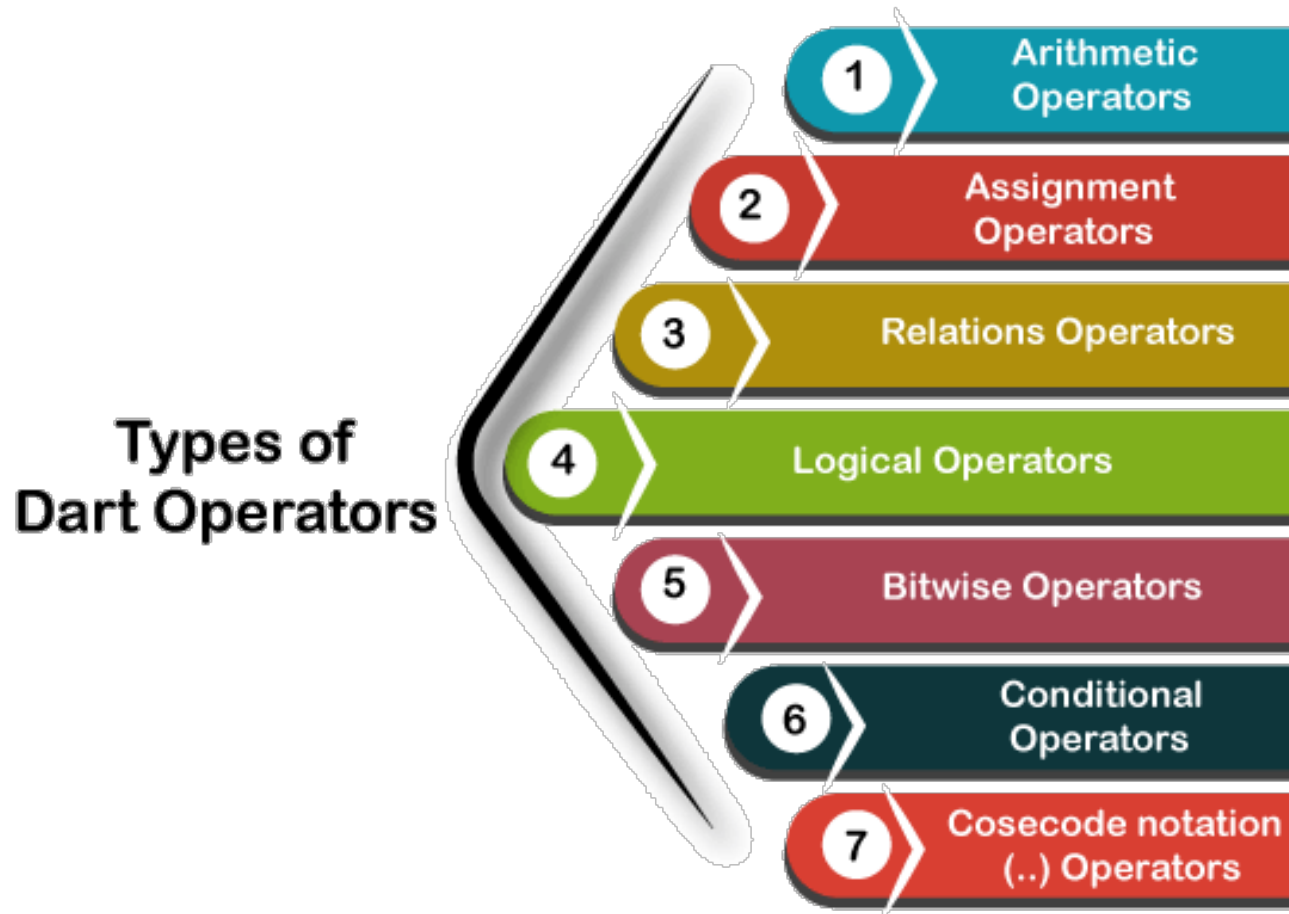
# Final and const

- 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.
- Example -  
final name = 'Rashmi';  
// final variable without type annotation.  
final String msg = 'Hi?';  
// final variable with type annotation.

# Final and const

- 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`

# Operators



# Arithmetic Operators

- +
- -
- \*
- /
- %
- Unary -

# Arithmetic Operators

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



# Increment and Decrement

- ++ and -- operators are known as increment and decrement operators and also known as unary operators, respectively.
- Unary operators, operate on single operand where ++ adds 1 to operands and -- subtract 1 to operand respectively.
- The unary operators can be used in two ways - postfix and prefix.
- If ++ is used as a postfix (like `x++`), it returns the value of operand first then increments the value of `x`. If -- is used as a prefix (like `--x`), it increases the value of `x`.

# Assignment Operators

- $=$
- $+=$
- $-=$
- $*=$
- $\sim/=$
- $\%=$

# Relational Operators

- $==$
- $!=$
- $<$
- $>$
- $<=$
- $>=$

# Bitwise Operators

- AND &
- OR |
- EX-OR ^
- >>
- <<
- ~

# Type Test Operators

- `as`
  - It is used for typecast.
- `is`
  - It returns TRUE if the object has specified type.
- `is!`
  - It returns TRUE if the object has not specified type.

# Type Test Operators

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

# Logical Operators

- `&&`
- `||`
- `!`

# 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.



# Conditional Operators

- Syntax 2 -

`exp1 ?? expr2`

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

# Conditional Operators

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

# Conditional Operators

```
void main() {  
    var a = 30;  
    var output = a > 38 ? "value greater than 10":"value lesser  
than equal to 30";  
    print(output);  
}
```

# The parse()

- The parse() function converts the numeric string to the number. Consider the following example -

```
void main(){  
    var a = num.parse("20.56");  
    var b = num.parse("15.63");  
    var c = a+b;  
    print("The sum is = ${c}");  
}
```

# The number properties

|            |   |
|------------|---|
| hashCode   | It returns the hash code of the given number.                       |
| isFinite   | If the given number is finite, then it returns true.                |
| isInfinite | If the number infinite it returns true.                             |
| isNan      | If the number is non-negative then it returns true.                 |
| isNegative | If the number is negative then it returns true.                     |
| sign       | It returns -1, 0, or 1 depending upon the sign of the given number. |
| isEven     | If the given number is an even then it returns true.                |
| isOdd      | If the given number is odd then it returns true.                    |

# The number methods

|             |  |
|-------------|--|
| abs()       | It gives the absolute value of the given number.                 |
| ceil()      | It gives the ceiling value of the given number.                  |
| floor()     | It gives the floor value of the given number.                    |
| compareTo() | It compares the value with other number.                         |
| remainder() | It gives the truncated remainder after dividing the two numbers. |
| round()     | It returns the round of the number.                              |
| toDouble()  | It gives the double equivalent representation of the number.     |
| toInt()     | Returns the integer equivalent representation of the number.     |
| toString()  | Returns the String equivalent representation of the number       |
| truncate()  | Returns the integer after discarding fraction digits.            |

# Strings

- String is a sequence of the character or UTF-16 code units. It is used to store the text value. The string can be created using single quotes or double-quotes.
- The multiline string can be created using the triple-quotes. Strings are immutable; it means you cannot modify it after creation.
- In Dart, The String keyword can be used to declare the string.

# Strings

- `String msg = 'Welcome to MITU';`  
or
- `String msg1 = "This is double-quoted string example.";`  
or
- `String msg2 = ''' line1  
line2  
line3'''`



# Strings

- The + or += operator is used to merge the two string.
- String Interpolation
  - The string interpolation is a technique to manipulate the string and create the new string by adding another value.
  - It can be used to evaluate the string including placeholders, variables, and interpolated expression.
  - The `${expression}` is used for string interpolation. The expressions are replaced with their corresponding values.

# Strings

- Properties and Methods

# Useful web resources

- [www.mitu.co.in](http://www.mitu.co.in)
- [www.pythonprogramminglanguage.com](http://www.pythonprogramminglanguage.com)
- [www.scikit-learn.org](http://www.scikit-learn.org)
- [www.towardsdatascience.com](http://www.towardsdatascience.com)
- [www.medium.com](http://www.medium.com)
- [www.analyticsvidhya.com](http://www.analyticsvidhya.com)
- [www.kaggle.com](http://www.kaggle.com)
- [www.stephacking.com](http://www.stephacking.com)
- [www.github.com](http://www.github.com)

# Thank you

*This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License*



@mitu\_skillologies



/MITuSkillologies



@mitu\_group



/company/mitu-  
skillologies



MITUSkillologies

## Web Resources

<https://mitu.co.in>

<http://tusharkute.com>

[contact@mitu.co.in](mailto:contact@mitu.co.in)

[tushar@tusharkute.com](mailto:tushar@tusharkute.com)