Dart Data Types:

- The data types are the most important fundamental features of programing 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.

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;
```

Both int and double are subtypes of num. You can also declare a variable as a num. If you do this, the variable can have both integer and double values.

```
num x=12 ;
x +=2.5 ;
```

Integer literals are automatically converted to doubles when necessary:

```
double z = 1; // Equivalent to double z = 1.0.
```

Here’s how you turn a string into a number, or vice versa:

```
// String -> int
var one = int.parse('1');
assert(one == 1);

// String -> double
var onePointOne = double.parse('1.1');
assert(onePointOne == 1.1);

// int -> String
String oneAsString = 1.toString();
assert(oneAsString == '1');
```

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 softwarica";
```

You can put the value of an expression inside a string by using ${*expression*}. If the expression is an identifier, you can skip the {}. To get the string corresponding to an object, Dart calls the object's toString() method.

```
var s = 'string interpolation';

assert('Dart has $s, which is very handy.' ==
   'Dart has string interpolation, '
      'which is very handy.');
assert('That deserves all caps. '
      '${s.toUpperCase()} is very handy!' ==
   'That deserves all caps. '
      'STRING INTERPOLATION is very handy!');
```

You can concatenate strings using adjacent string literals or the + operator:

```
var s1 = 'String '
   'concatenation'
   " works even over line breaks.";
assert(s1 ==
   'String concatenation works even over '
      'line breaks.');

var s2 = 'The + operator ' + 'works, as well.';
assert(s2 == 'The + operator works, as well.');
```

Another way to create a multi-line string: use a triple quote with either single or double quotation marks:

```
var s1 = '''
You can create
multi-line strings like this one.
''';

var s2 = """This is also a
multi-line string.""";
```

You can create a "raw" string by prefixing it with r:

```
var s = r'In a raw string, not even \n gets special treatment.';
```

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]
```

Lists use zero-based indexing, where 0 is the index of the first value and list.length - 1 is the index of the last value. You can get a list's length and refer to list values just as you would in JavaScript:

```
var list = [1, 2, 3];
assert(list.length == 3);
assert(list[1] == 2);

list[1] = 1;
assert(list[1] == 1);
```

Sets

A set in Dart is an unordered collection of unique items. Dart support for sets is provided by set literals and the Set type.

Here is a simple Dart set, created using a set literal:

```
var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};
```

To create an empty set, use {} preceded by a type argument, or assign {} to a variable of type Set:

```
var names = <String>{};
// Set<String> names = {}; // This works, too.
// var names = {}; // Creates a map, not a set.
```

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'}
```

Here are a couple of simple Dart maps, created using map literals:

```
var gifts = {
  // Key:    Value
  'first': 'partridge',
  'second': 'turtledoves',
  'fifth': 'golden rings'
};

var nobleGases = {
  2: 'helium',
  10: 'neon',
  18: 'argon',
};
```

You can create the same objects using a Map constructor:

```
var gifts = Map<String, String>();
gifts['first'] = 'partridge';
gifts['second'] = 'turtledoves';
gifts['fifth'] = 'golden rings';

var nobleGases = Map<int, String>();
nobleGases[2] = 'helium';
```

```
nobleGases[10] = 'neon';
nobleGases[18] = 'argon';
```

Add a new key-value pair to an existing map just as you would in JavaScript:

```
var gifts = {'first': 'partridge'};
gifts['fourth'] = 'calling birds'; // Add a key-value pair
```

Retrieve a value from a map the same way you would in JavaScript:

```
var gifts = {'first': 'partridge'};
assert(gifts['first'] == 'partridge');
```

If you look for a key that isn't in a map, you get a null in return:

```
var gifts = {'first': 'partridge'};
assert(gifts['fifth'] == null);
```

Use `.length` to get the number of key-value pairs in the map:

```
var gifts = {'first': 'partridge'};
gifts['fourth'] = 'calling birds';
assert(gifts.length == 2);
```

To create a map that's a compile-time constant, add `const` before the map literal:

```
final constantMap = const {
  2: 'helium',
  10: 'neon',
  18: 'argon',
};

// constantMap[2] = 'Helium'; // This line will cause an error.
```

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.

Example -

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

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.

- dynamic value=12;