Dart

Cross Platform Mobile Application Development

Introduction to Dart

- Dart is a client-optimized, object-oriented, modern programming language to build apps fast for many platforms like android, iOS, web, desktop, etc.
- Client optimized means optimized for crafting a beautiful user interface and high-quality experiences.
- Dart is the open-source programming language originally developed by Google.
- It is meant for both server side as well as the user side.
- The Dart SDK comes with its compiler – the Dart VM and a utility dart2js which is meant for generating Javascript equivalent of a Dart Script so that it can be run on those sites also which don’t support Dart.
- Dart is extensively use to create single-page websites and web-applications.
- **Best example of dart application is Gmail.**

Dart Features

- Free and open-source.
- Object-oriented programming language.
- Used to develop android, iOS, web, and desktop apps fast.
- Can compile to either native code or javascript.
- Offers modern programming features like null safety and asynchronous programming.
- Null safety is a feature in Dart that helps us distinguish between nullable variables and non-nullable variables. A nullable variable is one that can hold either a non-null value or a null value. On the other hand, a non-nullable variable is one that must always hold a non-null value
- The Dart Future is defined as getting a result sometime in the future. The Future object uses to facilitate asynchronous programming. Future objects are a tool to denote values returned by an expression whose execution will complete at a later point in time (In Future). In order to work with the future, we can use either async and await or the Future API.

- You can even use Dart for servers and backend.

Difference between Asynchronous & Synchronous

- Let's understand the difference between synchronous and asynchronous.
- In computer science, if we say a particular program is synchronous, that means it waits for an event to execute further. This approach is driven with a demerit, if a part of the code takes much time to execute, the succeeding blocks through an unrelated block will be blocked from executing.
- This is the main problem of the synchronous approach. A part of the program may require executing before the current part, but the synchronous approach doesn't allow it.
- This is not suitable for the webservers, where request must be independent of the others. It means, the webserver does not wait to finish the execution of the current request, it responds to the request from the other users.
- The web server should accept the request from the other user before executing the previous requests.
- This approach is called asynchronous programming. The asynchronous programming generally focuses on no waiting or non-blocking programming model. The dart: async is facilitated to implement the asynchronous programming block in a Dart script.

Difference Between Dart & Flutter

- **Dart** is a client optimized, object-oriented programming language. It is popular nowadays because of flutter.
- It is difficult to build complete apps only using Dart because you have to manage many things yourself.
- **Flutter** is a framework that uses dart programming language.
- With the help of flutter, you can build apps for android, iOS, web, desktop, etc. The framework contains ready-made tools to make apps faster.

Dart History

- Google developed Dart in 2011 as an alternative to javascript.
- Dart 1.0 was released on November 14, 2013.
- Dart 2.0 was released in August 2018.
- Dart 3.0 was released in May 2023.
- Dart gained popularity in recent days because of flutter.

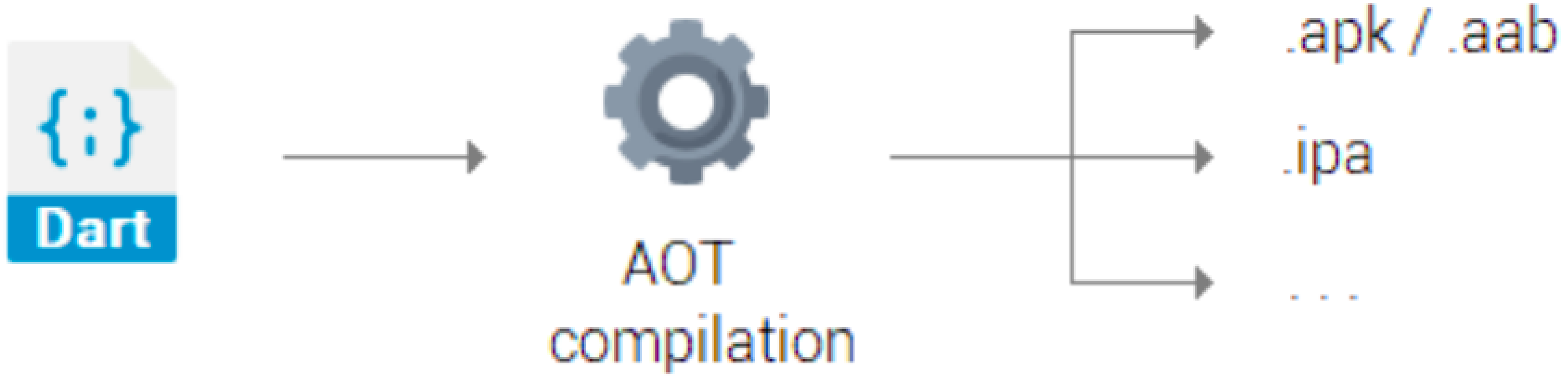
Dart : Supported Platform

- Dart can compile to native machine code for macOS, Windows, and Linux as command line tools. Dart can compile apps with user interfaces to the web, iOS, Android, macOS, Windows, and Linux using the Flutter framework.
- Dart’s compiler is used when executing a Dart program which translates the source program into a machine language equivalent.
- The Dart compiler architecture enables a variety of code execution methods on different platforms:

1. Native platform: Dart provides both a Dart virtual machine(VM) with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler for creating machine code for apps supporting both mobile and desktop platforms.
2. Web platform: Dart is converted to JavaScript using its web compiler.

Dart : Supported Platform

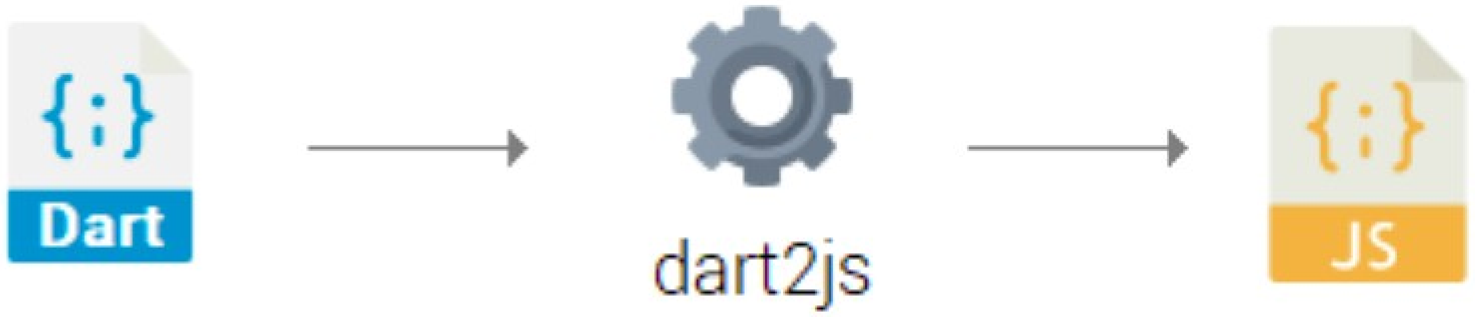
AOT compiled. The **A**head **O**f **T**ime compilation is the act of translating a high-level programming language, like Dart, into native machine code. Basically, starting from the Dart source code you can obtain a single binary file that can execute natively on a certain operating system. AOT is really what makes Flutter fast and portable.



With AOT there is **NO** need to have the DVM installed because at the end you get a single binary file (an *.apk* or *.aab* for Android, an *.ipa* for iOS, an *.exe* for Windows...) that can be executed.

Dart : Supported Platform

Web. Thanks to the `dart2js` tool, your Dart project can be "transpiled" into fast and compact JavaScript code. By consequence Flutter can be run, for example, on Firefox or Chrome and the UI will be identical to the other platforms.



AngularDart [5] is a performant web app framework used by Google to build some famous websites, such as "AdSense" and "AdWords". Of course it's powered by Dart!

Dart : Supported Platform

Desktop/mobile. The **J**ust **I**n **T**ime (JIT) technique can be seen as a "real time translation" because the compilation happens while the program is executing. It's a sort of "dynamic compilation" which happens while the program is being used.

JIT compilation, combined with the DVM (JIT + VM in the picture), allows the dispatch of the code dynamically without considering the user's machine architecture. In this way it's possible to smoothly run and debug the code everywhere without having to mess up with the underlying architecture.

Web. The Dart development compiler, abbreviated with **dartdevc**, allows you to run and debug Dart web apps on Google Chrome. Note that dartdevc is for development only: for deployment, you should use **dart2js**. Using special tools like *webdev* [6] there's the possibility to edit Dart files, refreshing Chrome and visualizing changes almost immediately.

Package System

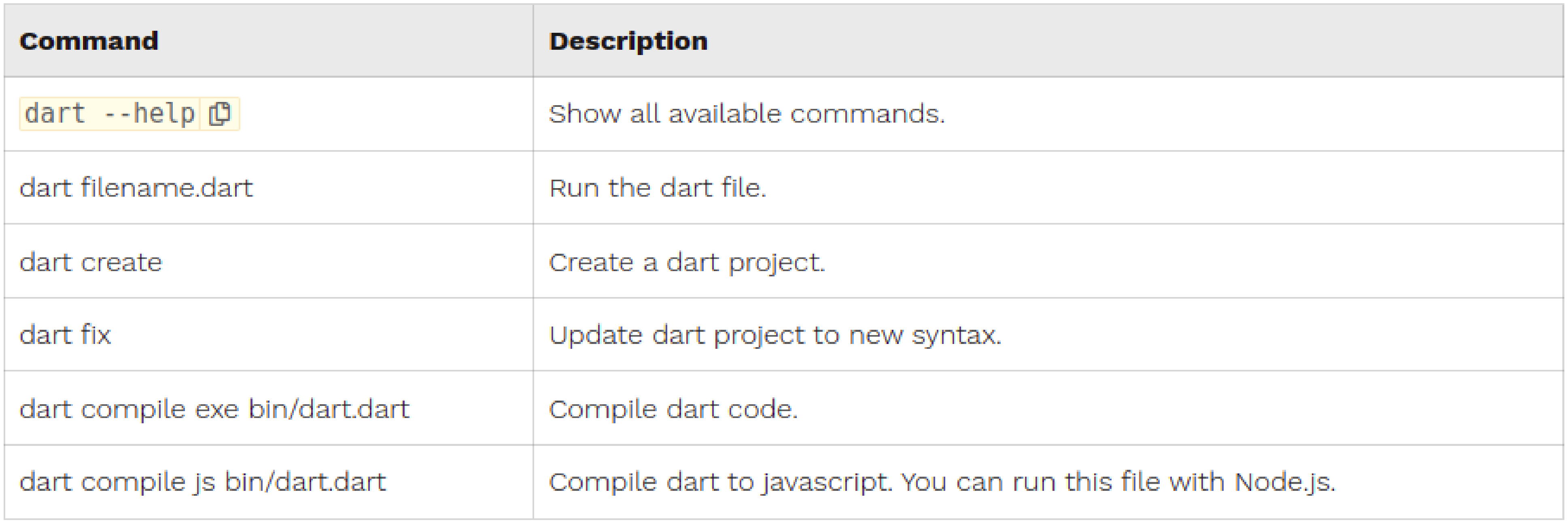
Dart's core API offers different packages, such as `dart:io` or `dart:collection`, that expose classes and methods for many purposes. In addition, there is an official online repository called *pub* containing packages created by the Dart team, the Flutter team or community users like you.

If you head to https://pub.dev you will find an endless number of packages for any purpose: I/O handling, XML serialization/de-serialization, localization, SQL/NoSQL database utilities and much more.

Install Dart

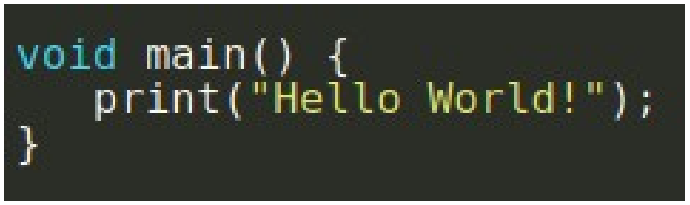
- Requirements :
  - Dart SDK,
  - VS code or other editors like Intellij [We will use VS Code here].
- Check Dart Installation :
  - dart –version
- Save and run the programe:
  - filename.dart
  - dart filename.dart(from terminal)

Usefull commands



Command	Description
`dart --help`	Show all available commands.
dart filename.dart	Run the dart file.
dart create	Create a dart project.
dart fix	Update dart project to new syntax.
dart compile exe bin/dart.dart	Compile dart code.
dart compile js bin/dart.dart	Compile dart to javascript. You can run this file with Node.js.

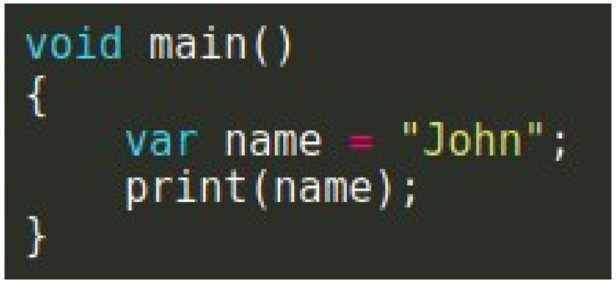
Basic Dart Program



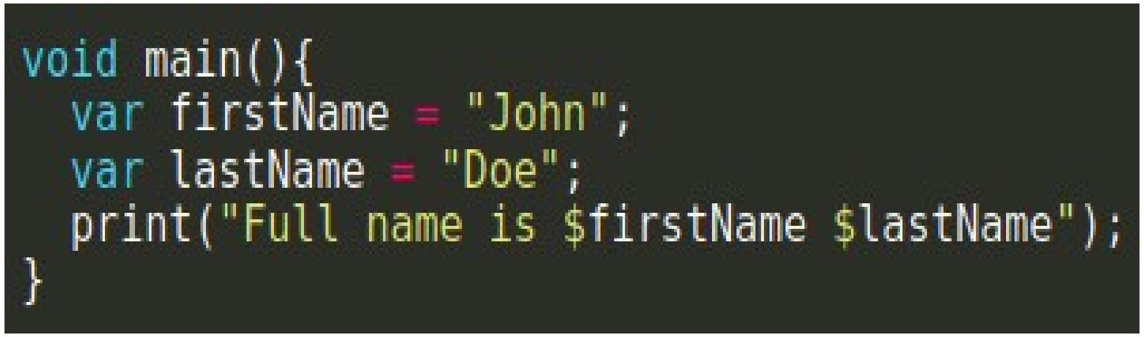
```
void main() {
   print("Hello World!");
}
```

- void main() is the starting point where the execution of your program begins.
- Every program starts with a main function.
- The curly braces {} represent the beginning and the ending of a block of code.
- print(“Hello World!”); prints Hello World! on screen.
- Each code statement must end with a semicolon.

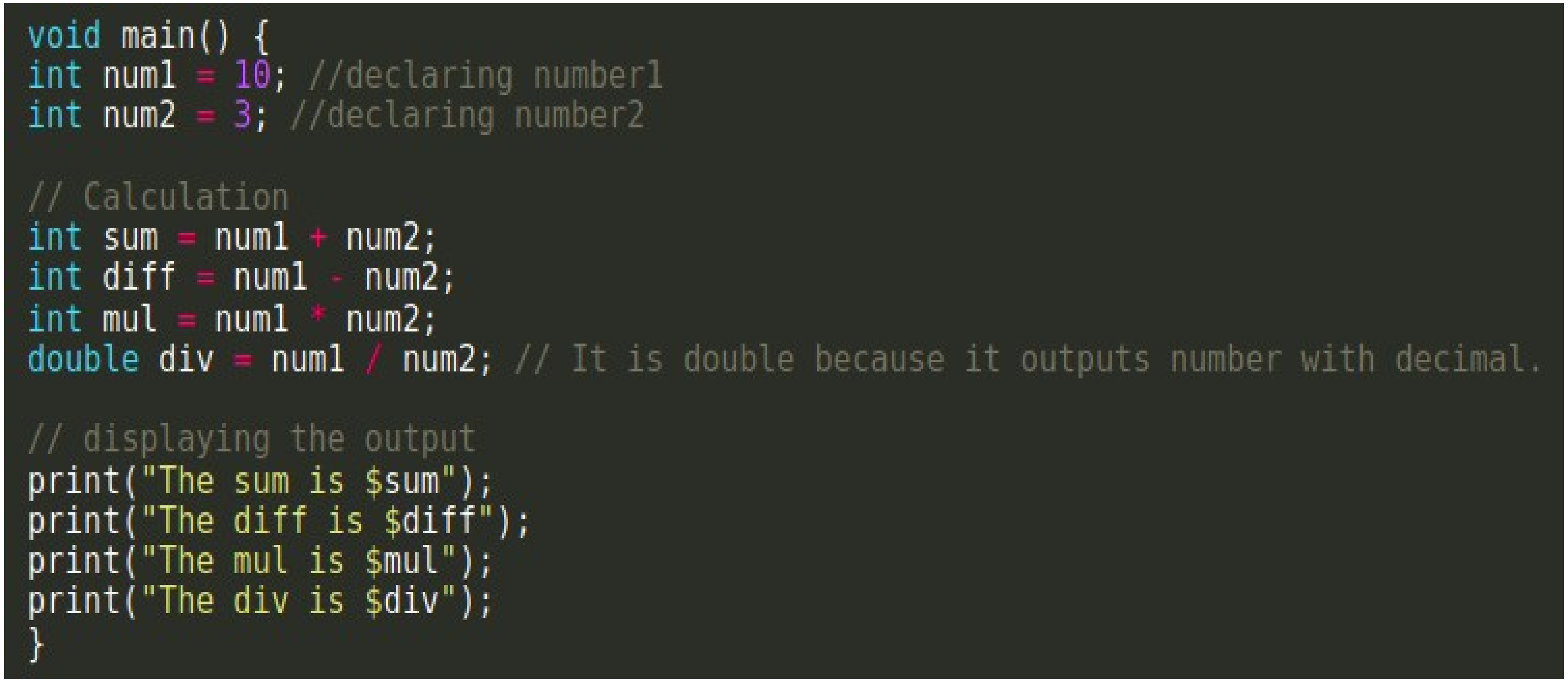
Few examples of Dart



```
void main()
{
     var name = "John";
     print(name);
}
```



```
void main(){
  var firstName = "John";
  var lastName = "Doe";
  print("Full name is $firstName $lastName");
}
```



```
void main() {
int num1 = 10; //declaring number1
int num2 = 3; //declaring number2

// Calculation
int sum = num1 + num2;
int diff = num1 - num2;
int mul = num1 * num2;
double div = num1 / num2; // It is double because it outputs number with decimal.

// displaying the output
print("The sum is $sum");
print("The diff is $diff");
print("The mul is $mul");
print("The div is $div");
}
```

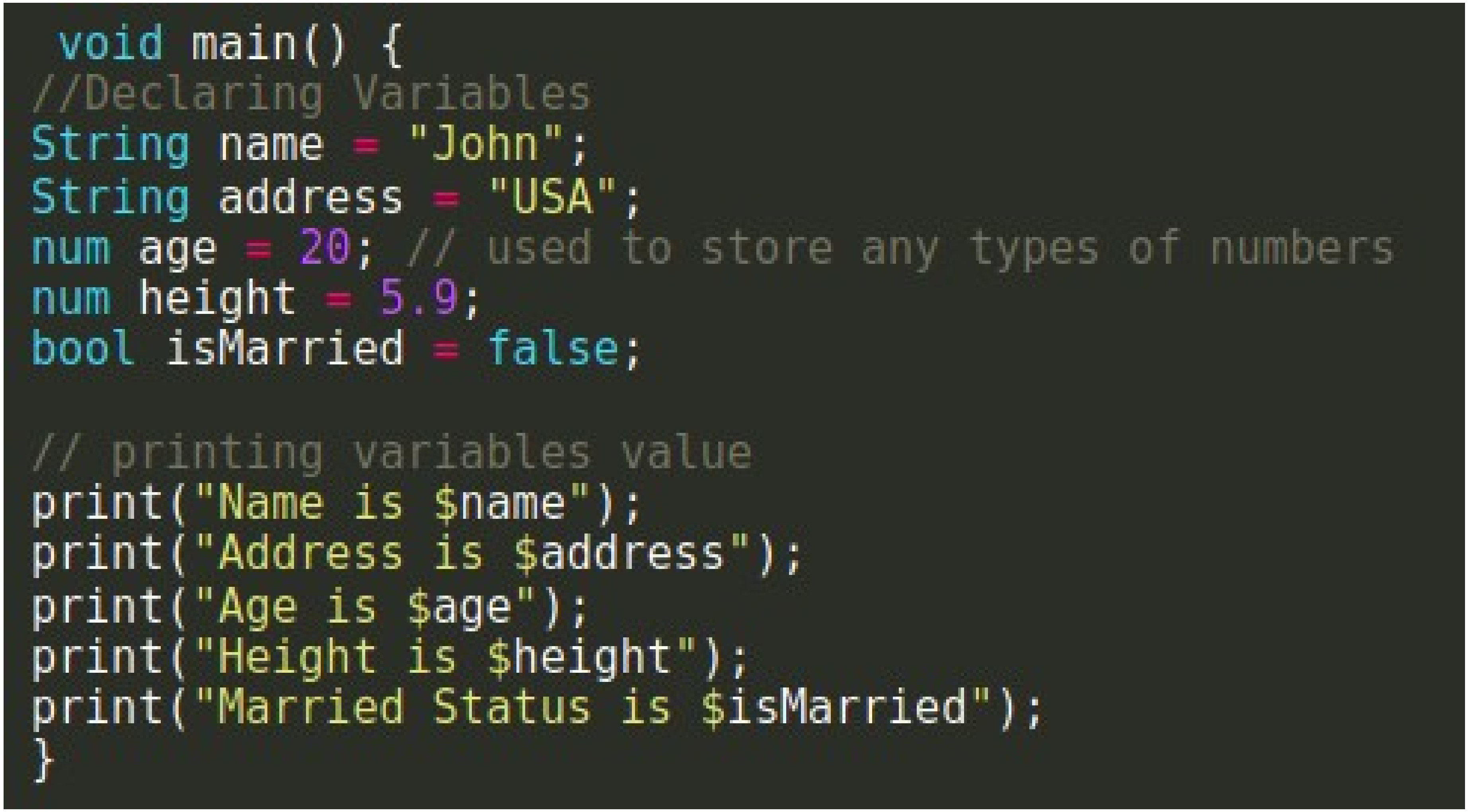
Variable in dart

- Variables are containers used to store value in the program. There are different types of variables where you can keep different kinds of values. Here is an example of creating a variable and initializing it.Ex: var name = "Dart";
- String: For storing text value. E.g. “John” [Must be in quotes]
- int: For storing integer value. E.g. 10, -10, 8555 [Decimal is not included]
- double: For storing floating point values. E.g. 10.0, -10.2, 85.698 [Decimal is included]
- num: For storing any type of number. E.g. 10, 20.2, -20 [both int and double]
- bool: For storing true or false. E.g. true, false [Only stores true or false values]
- var: For storing any value. E.g. ‘Bimal’, 12, ‘z’, true

Conditions to write variable name or identifiers are as follows:

- Variable name or identifiers can’t be the keyword.
- Variable name or identifiers can contain alphabets and numbers.
- Variable name or identifiers can’t contain spaces and special characters, except the underscore(_) and the dollar($) sign.
- Variable name or identifiers can’t begin with number.

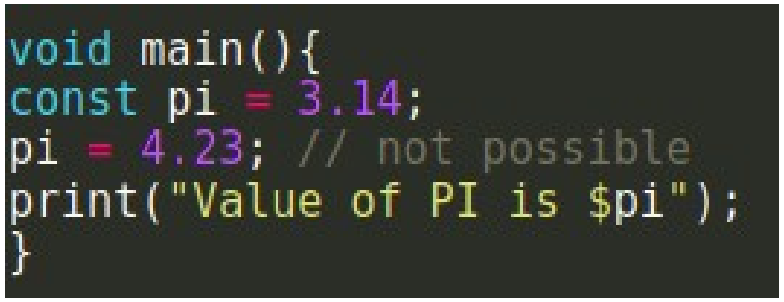
Example : Varibales



```
 void main() {
//Declaring Variables
String name = "John";
String address = "USA";
num age = 20; // used to store any types of numbers
num height = 5.9;
bool isMarried = false;

// printing variables value
print("Name is $name");
print("Address is $address");
print("Age is $age");
print("Height is $height");
print("Married Status is $isMarried");
}
```

Dart Constant

- Constant is the type of variable whose value never changes.
- In programming, changeable values are mutable and unchangeable values are immutable.
- Sometimes, you don’t need to change the value once declared. Like the value of PI=3.14, it never changes. To create a constant in Dart, you can use the const keyword.



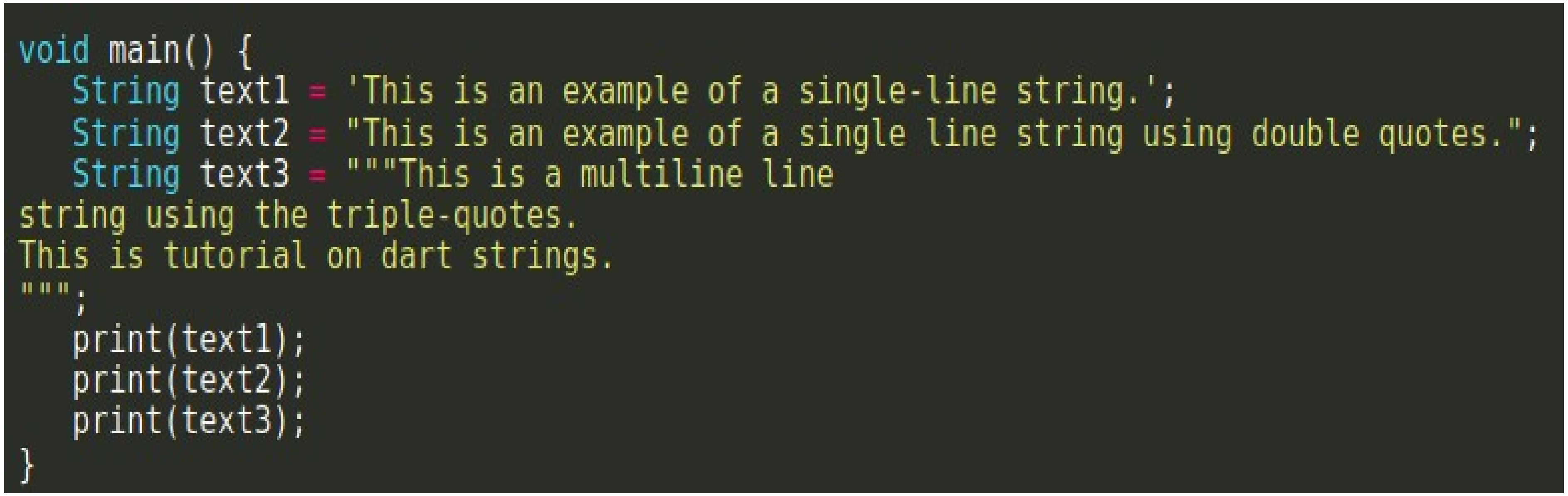
```
void main(){
const pi = 3.14;
pi = 4.23; // not possible
print("Value of PI is $pi");
}
```

Data Types

- Data types help you to categorize all the different types of data you use in your code.
- For e.g. numbers, texts, symbols, etc. The data type specifies what type of value will be stored by the variable. Each variable has its data type. Dart supports the following built-in data types :
- Numbers
- Strings
- Booleans
- Lists
- Maps
- Sets
- Runes
- Null

String datatype

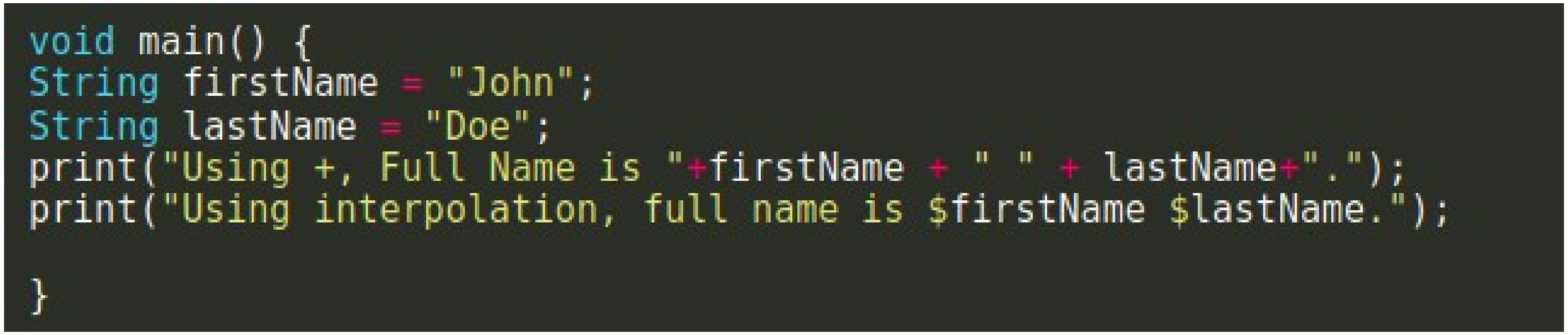
- String helps you to store text based data. In String, you can represent your name, address, or complete book.
- It holds a series or sequence of characters – letters, numbers, and special characters. You can use single or double, or triple quotes to represent String.



```
void main() {
   String text1 = 'This is an example of a single-line string.';
   String text2 = "This is an example of a single line string using double quotes.";
   String text3 = """This is a multiline line
string using the triple-quotes.
This is tutorial on dart strings.
""";
   print(text1);
   print(text2);
   print(text3);
}
```

String Concatenation

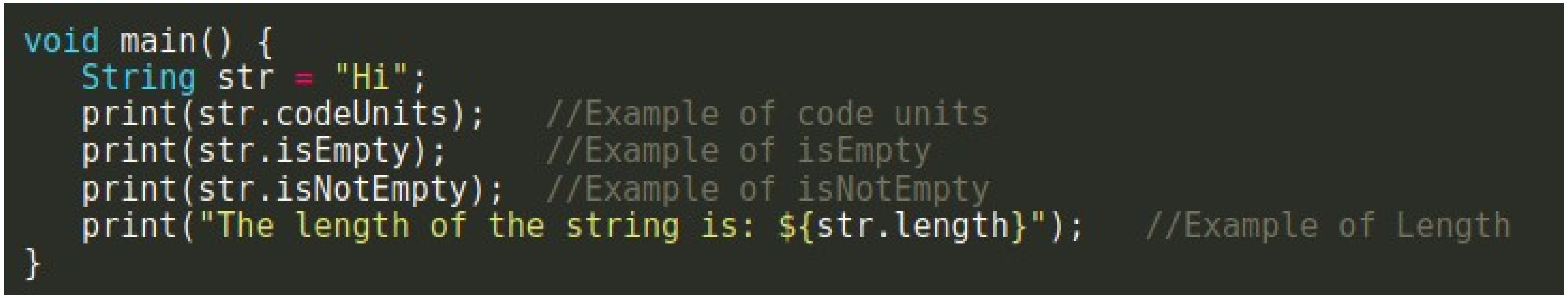
- You can combine one String with another string. This is called concatenation.
- In Dart, you can use the + operator or use interpolation to concatenate the String. Interpolation makes it easy to read and understand the code.



```dart
void main() {
String firstName = "John";
String lastName = "Doe";
print("Using +, Full Name is "+firstName + " " + lastName+".");
print("Using interpolation, full name is $firstName $lastName.");

}
```

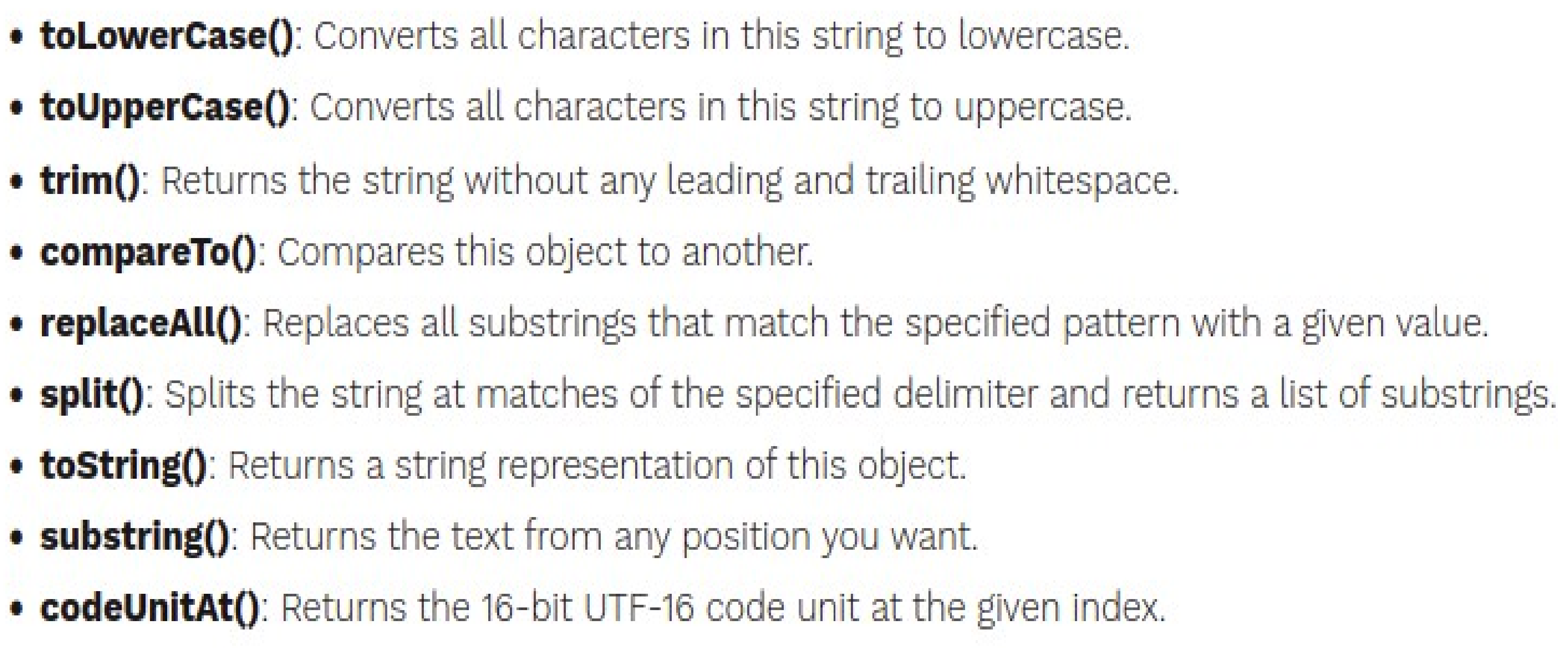
Properties Of String

- codeUnits: Returns an unmodifiable list of the UTF-16 code units of this string.
- isEmpty: Returns true if this string is empty.
- isNotEmpty: Returns false if this string is empty.
- length: Returns the length of the string including space, tab, and newline characters.



```
void main() {
   String str = "Hi";
   print(str.codeUnits);   //Example of code units
   print(str.isEmpty);     //Example of isEmpty
   print(str.isNotEmpty);  //Example of isNotEmpty
   print("The length of the string is: ${str.length}");   //Example of Length
}
```

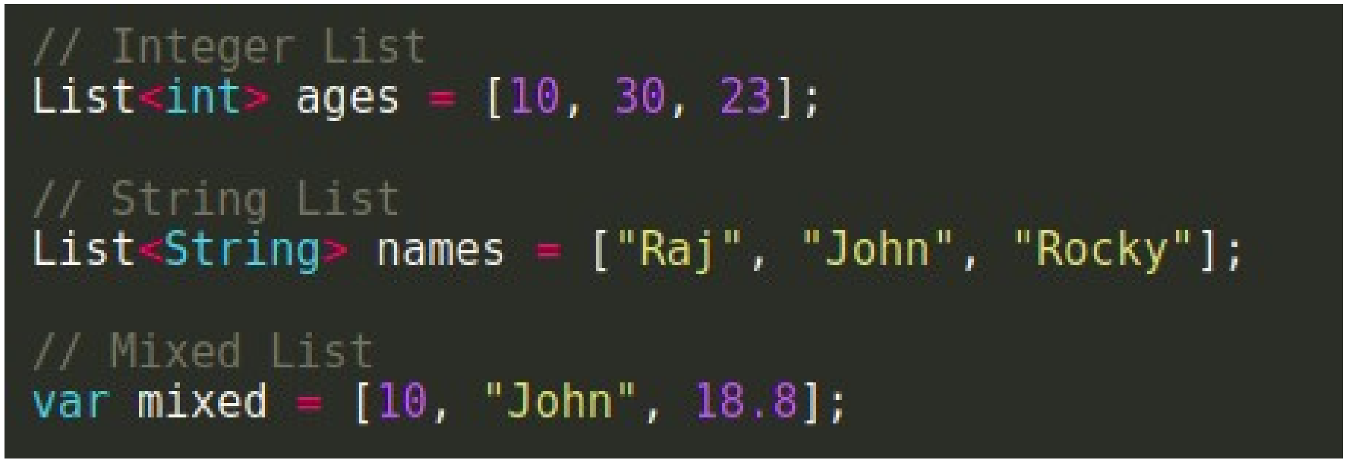
Methods Of String



- **toLowerCase()**: Converts all characters in this string to lowercase.
- **toUpperCase()**: Converts all characters in this string to uppercase.
- **trim()**: Returns the string without any leading and trailing whitespace.
- **compareTo()**: Compares this object to another.
- **replaceAll()**: Replaces all substrings that match the specified pattern with a given value.
- **split()**: Splits the string at matches of the specified delimiter and returns a list of substrings.
- **toString()**: Returns a string representation of this object.
- **substring()**: Returns the text from any position you want.
- **codeUnitAt()**: Returns the 16-bit UTF-16 code unit at the given index.

List

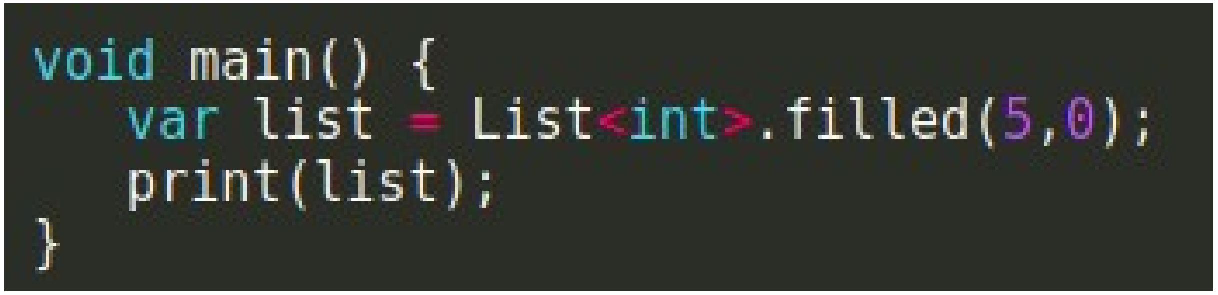
- If you want to store multiple values in the same variable, you can use List.
- List in dart is similar to Arrays in other programming languages. E.g. to store the names of multiple students, you can use a List. The List is represented by Square Braces[].



```
// Integer List
List<int> ages = [10, 30, 23];

// String List
List<String> names = ["Raj", "John", "Rocky"];

// Mixed List
var mixed = [10, "John", 18.8];
```

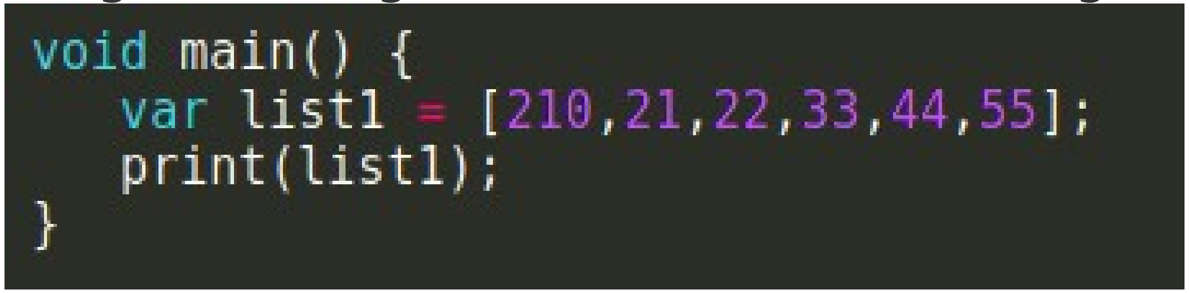
Types Of Lists

- Fixed Length List
- Growable List [Mostly Used]
- The fixed-length Lists are defined with the specified length. You cannot change the size at runtime. This will create List of 5 integers with the value 0.



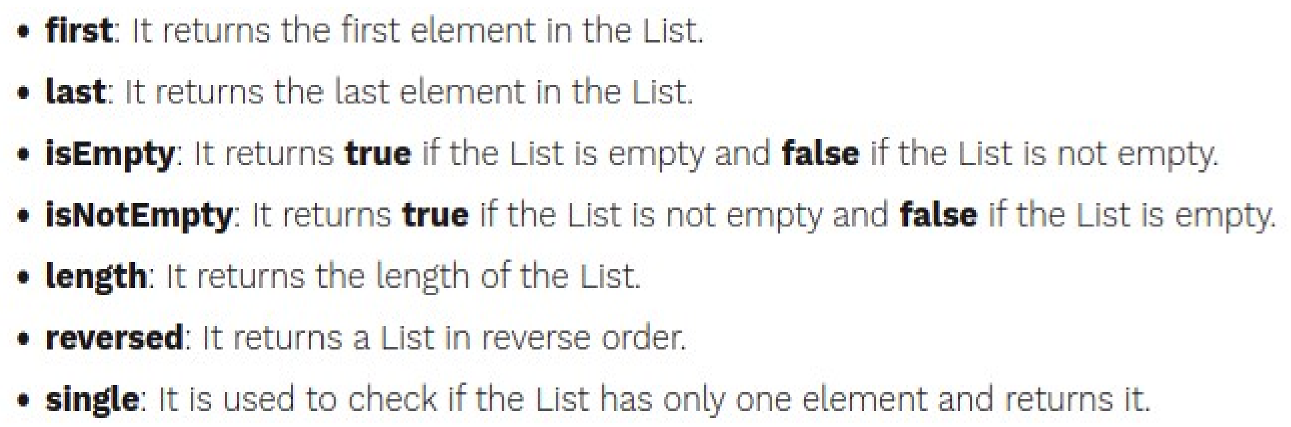
```
void main() {
   var list = List<int>.filled(5,0);
   print(list);
}
```

- A List defined without a specified length is called Growable List. The length of the growable List can be changed in runtime.

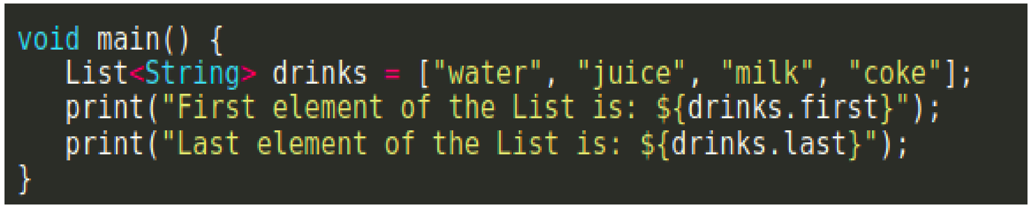


```
void main() {
   var list1 = [210,21,22,33,44,55];
   print(list1);
}
```

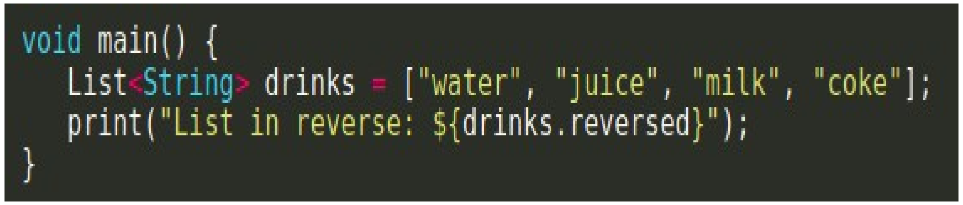
List Properties In Dart



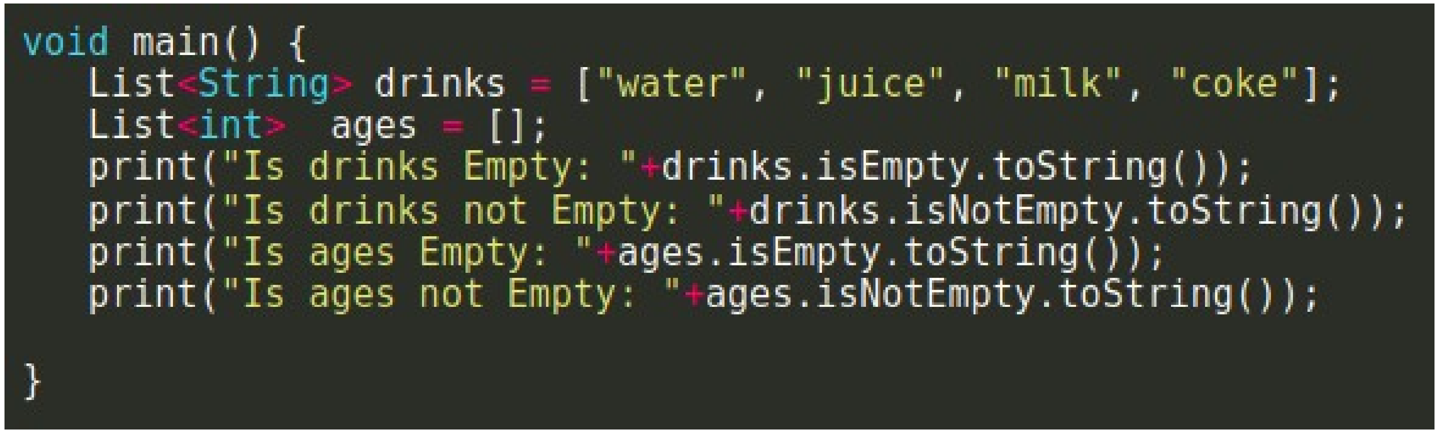
- **first**: It returns the first element in the List.
- **last**: It returns the last element in the List.
- **isEmpty**: It returns **true** if the List is empty and **false** if the List is not empty.
- **isNotEmpty**: It returns **true** if the List is not empty and **false** if the List is empty.
- **length**: It returns the length of the List.
- **reversed**: It returns a List in reverse order.
- **single**: It is used to check if the List has only one element and returns it.



```dart
void main() {
   List<String> drinks = ["water", "juice", "milk", "coke"];
   print("First element of the List is: ${drinks.first}");
   print("Last element of the List is: ${drinks.last}");
}
```

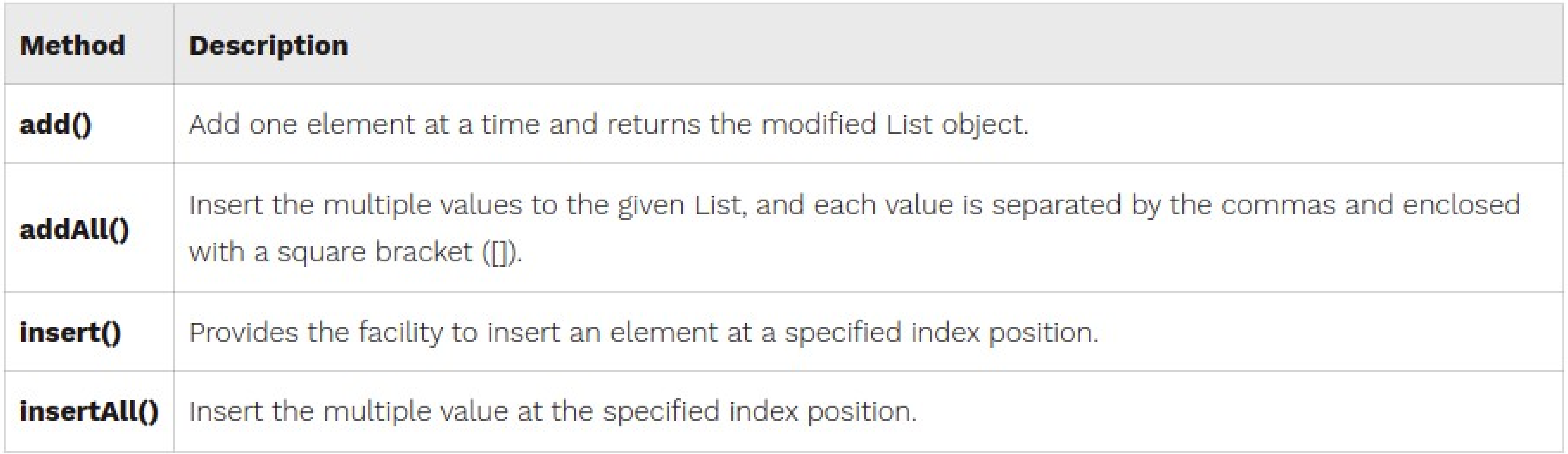


```dart
void main() {
   List<String> drinks = ["water", "juice", "milk", "coke"];
   print("List in reverse: ${drinks.reversed}");
}
```



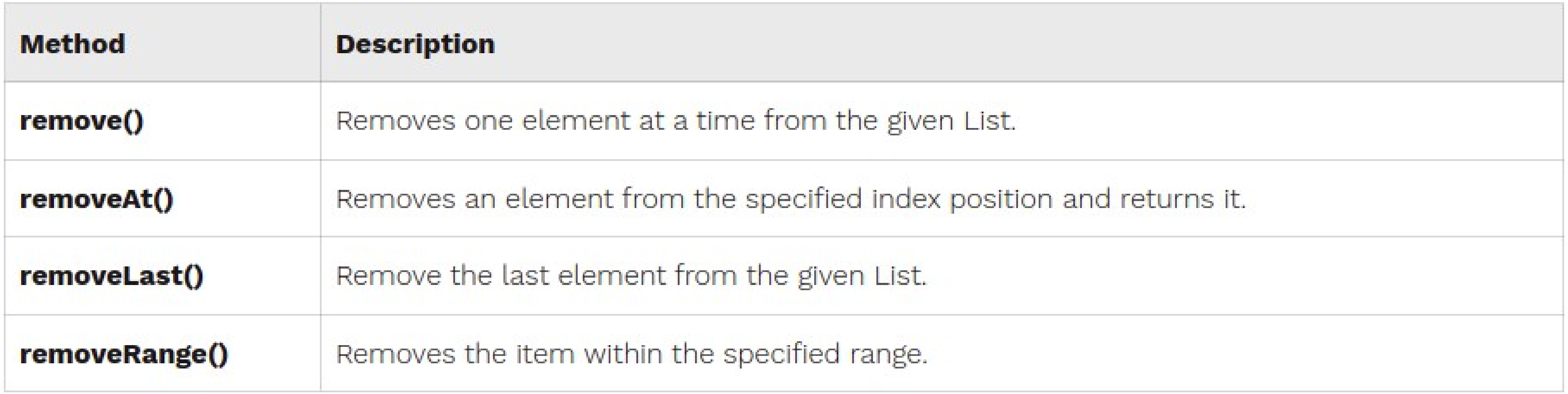
```dart
void main() {
   List<String> drinks = ["water", "juice", "milk", "coke"];
   List<int>  ages = [];
   print("Is drinks Empty: "+drinks.isEmpty.toString());
   print("Is drinks not Empty: "+drinks.isNotEmpty.toString());
   print("Is ages Empty: "+ages.isEmpty.toString());
   print("Is ages not Empty: "+ages.isNotEmpty.toString());

}
```

Adding Item To List



Method	Description
add()	Add one element at a time and returns the modified List object.
addAll()	Insert the multiple values to the given List, and each value is separated by the commas and enclosed with a square bracket ([]).
insert()	Provides the facility to insert an element at a specified index position.
insertAll()	Insert the multiple value at the specified index position.

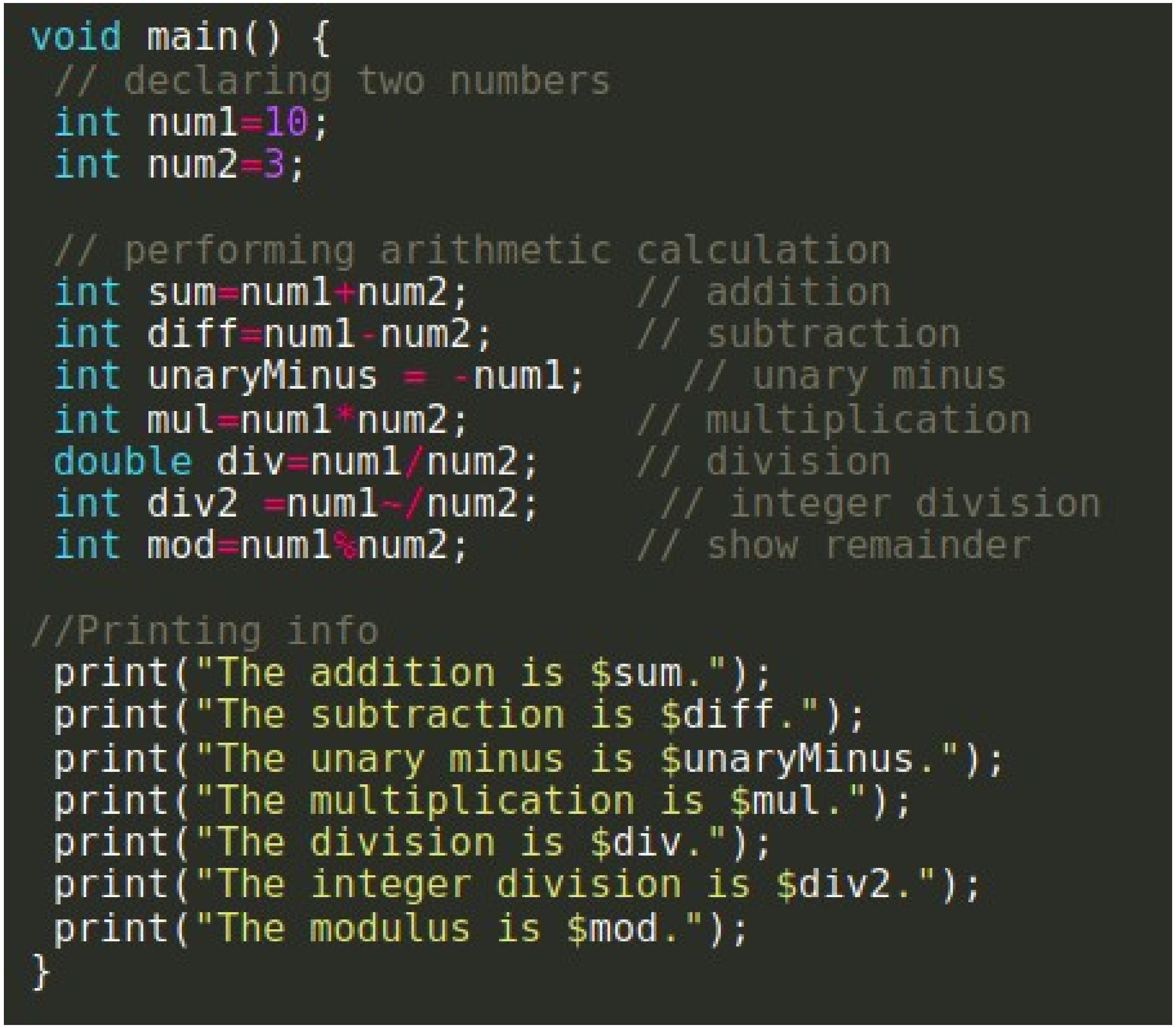
Removing List Elements



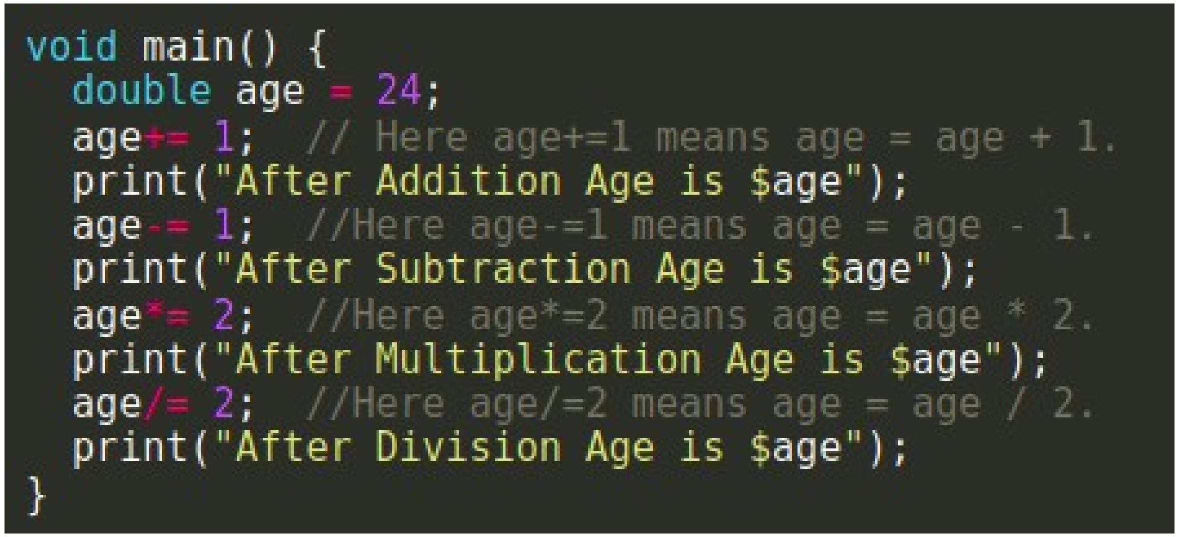
Method	Description
remove()	Removes one element at a time from the given List.
removeAt()	Removes an element from the specified index position and returns it.
removeLast()	Remove the last element from the given List.
removeRange()	Removes the item within the specified range.

Operators In Dart

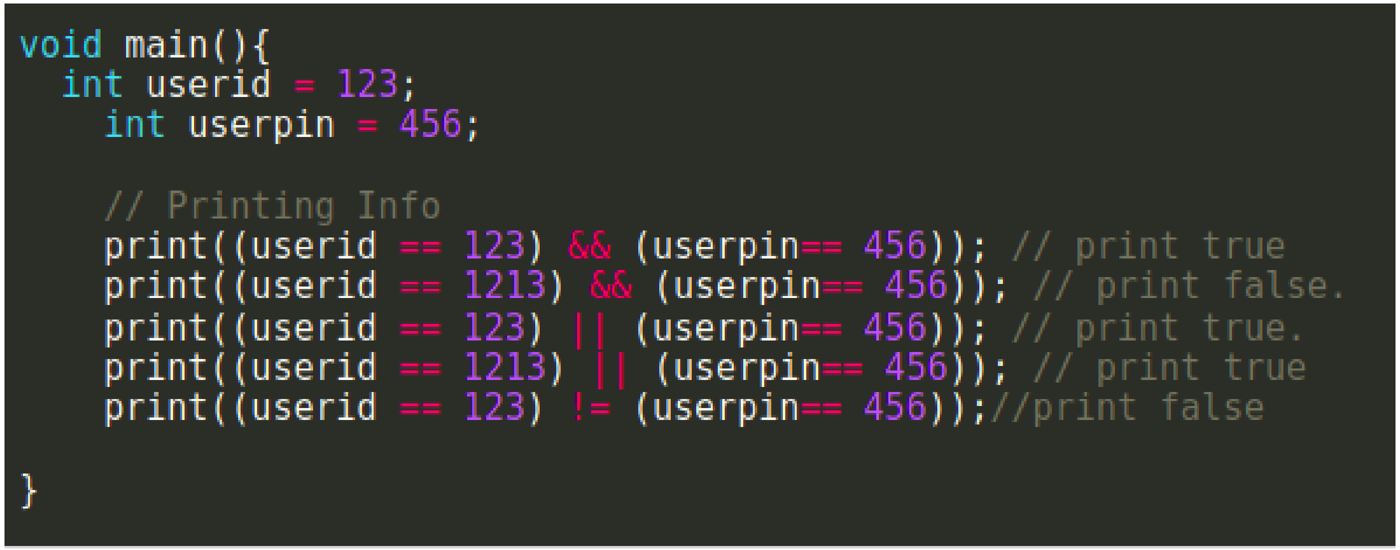
- Operators are used to perform mathematical and logical operations on the variables. Each operation in dart uses a symbol called the operator to denote the type of operation it performs. Before learning operators in the dart, you must understand the following things.
- **Operands** : It represents the data.
- **Operator** : It represents how the operands will be processed to produce a value.



```
void main() {
 // declaring two numbers
 int num1=10;
 int num2=3;

 // performing arithmetic calculation
 int sum=num1+num2;         // addition
 int diff=num1-num2;        // subtraction
 int unaryMinus = -num1;      // unary minus
 int mul=num1*num2;         // multiplication
 double div=num1/num2;      // division
 int div2 =num1~/num2;        // integer division
 int mod=num1%num2;         // show remainder

//Printing info
 print("The addition is $sum.");
 print("The subtraction is $diff.");
 print("The unary minus is $unaryMinus.");
 print("The multiplication is $mul.");
 print("The division is $div.");
 print("The integer division is $div2.");
 print("The modulus is $mod.");
}
```

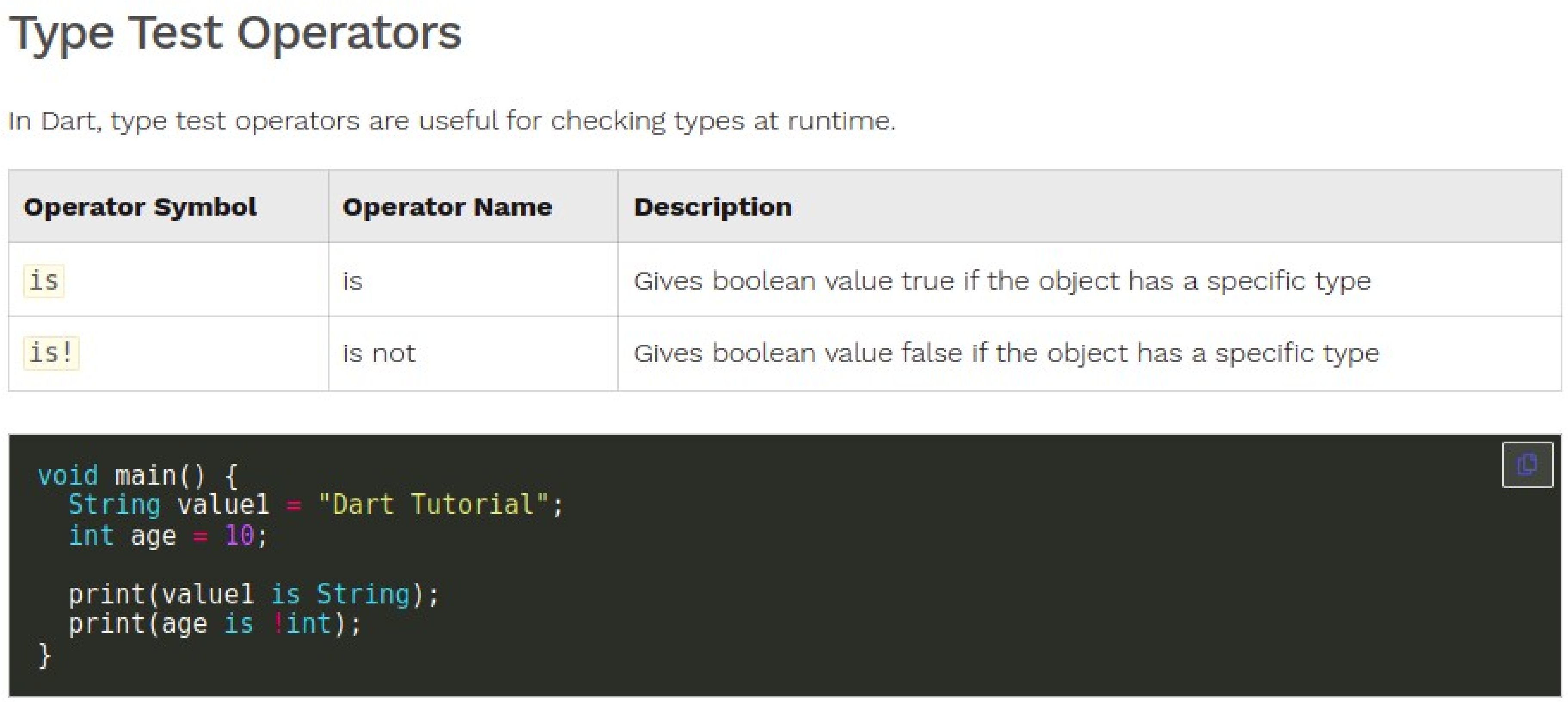


```
void main() {
  double age = 24;
  age+= 1;  // Here age+=1 means age = age + 1.
  print("After Addition Age is $age");
  age-= 1;  //Here age-=1 means age = age - 1.
  print("After Subtraction Age is $age");
  age*= 2;  //Here age*=2 means age = age * 2.
  print("After Multiplication Age is $age");
  age/= 2;  //Here age/=2 means age = age / 2.
  print("After Division Age is $age");
}
```



```
void main(){
  int userid = 123;
    int userpin = 456;

    // Printing Info
    print((userid == 123) && (userpin== 456)); // print true
    print((userid == 1213) && (userpin== 456)); // print false.
    print((userid == 123) || (userpin== 456)); // print true.
    print((userid == 1213) || (userpin== 456)); // print true
    print((userid == 123) != (userpin== 456));//print false

}
```

Type test operators



Type Test Operators

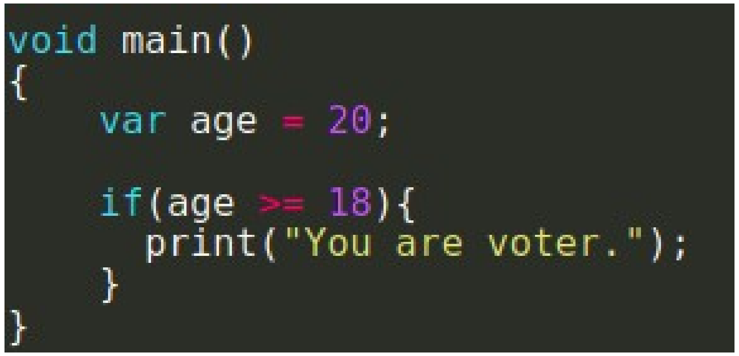
In Dart, type test operators are useful for checking types at runtime.

Operator Symbol	Operator Name	Description
`is`	is	Gives boolean value true if the object has a specific type
`is!`	is not	Gives boolean value false if the object has a specific type

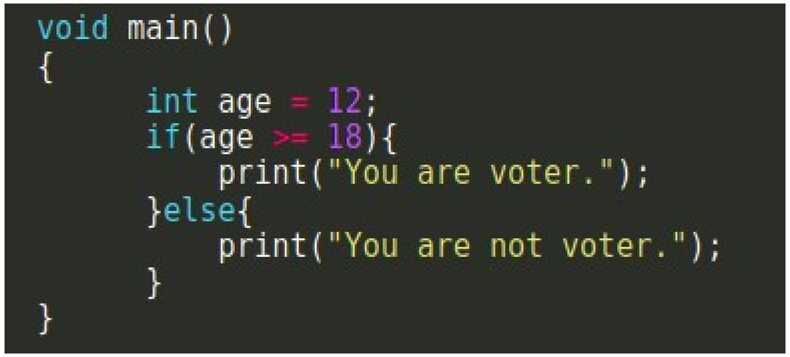
```dart
void main() {
  String value1 = "Dart Tutorial";
  int age = 10;

  print(value1 is String);
  print(age is !int);
}
```

Flow Statement

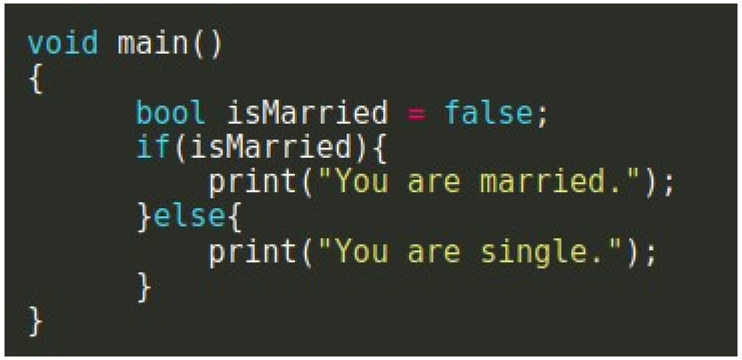
- Types Of Condition
  - If Condition
  - If-Else Condition
  - If-Else-If Condition
  - Switch case



```
void main()
{
    var age = 20;

    if(age >= 18){
      print("You are voter.");
    }
}
```



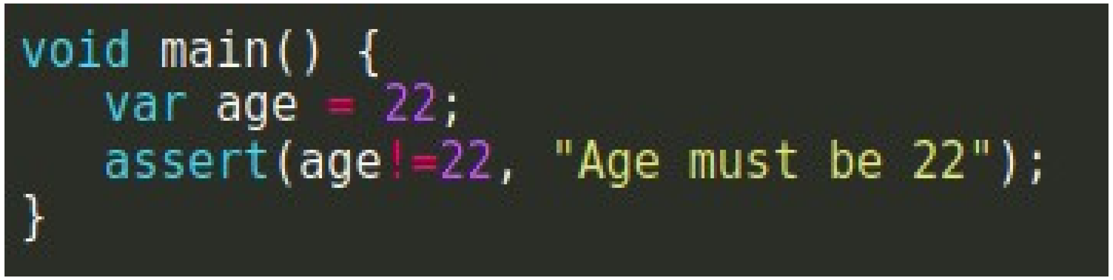
```
void main()
{
        int age = 12;
        if(age >= 18){
            print("You are voter.");
        }else{
            print("You are not voter.");
        }
}
```



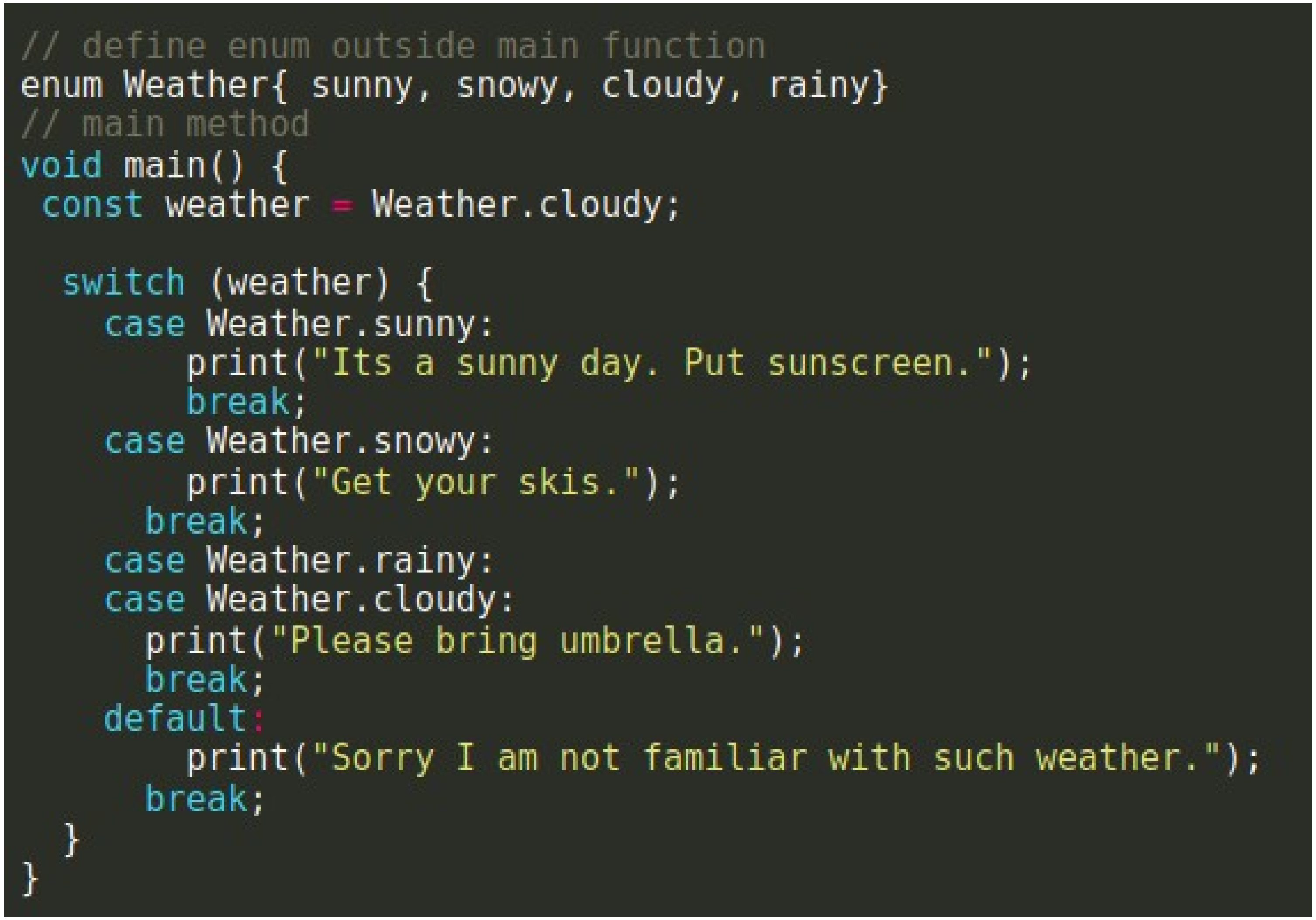
```
void main()
{
        bool isMarried = false;
        if(isMarried){
            print("You are married.");
        }else{
            print("You are single.");
        }
}
```

Assert in dart

- While coding, it is hard to find errors in big projects, so dart provide a assert method to check for the error.
- It takes conditions as an argument. If the condition is true, nothing happens. If a condition is false, it will raise an error.

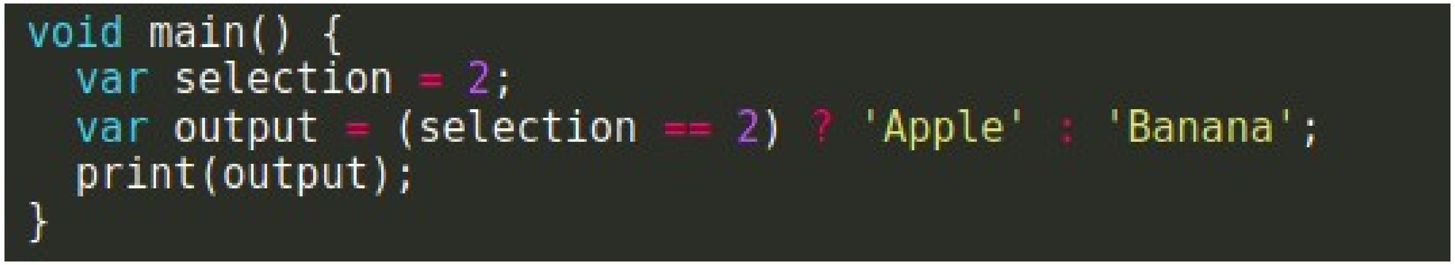


```
void main() {
   var age = 22;
   assert(age!=22, "Age must be 22");
}
```



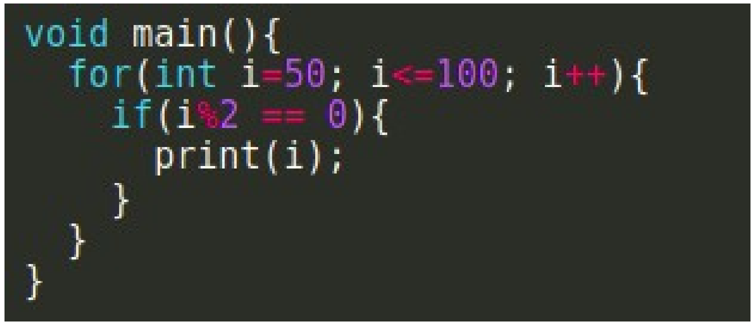
```
// define enum outside main function
enum Weather{ sunny, snowy, cloudy, rainy}
// main method
void main() {
 const weather = Weather.cloudy;

  switch (weather) {
    case Weather.sunny:
        print("Its a sunny day. Put sunscreen.");
        break;
    case Weather.snowy:
        print("Get your skis.");
      break;
    case Weather.rainy:
    case Weather.cloudy:
      print("Please bring umbrella.");
      break;
    default:
        print("Sorry I am not familiar with such weather.");
      break;
  }
}
```

Ternary operator

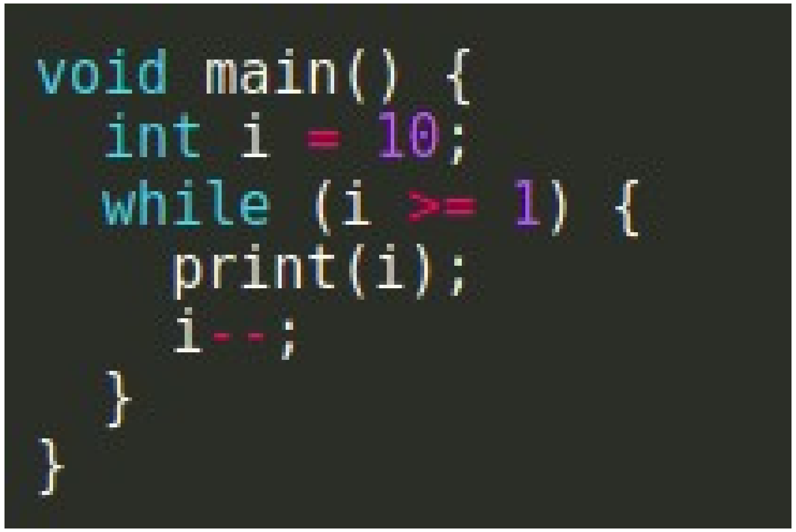


```
void main() {
  var selection = 2;
  var output = (selection == 2) ? 'Apple' : 'Banana';
  print(output);
}
```

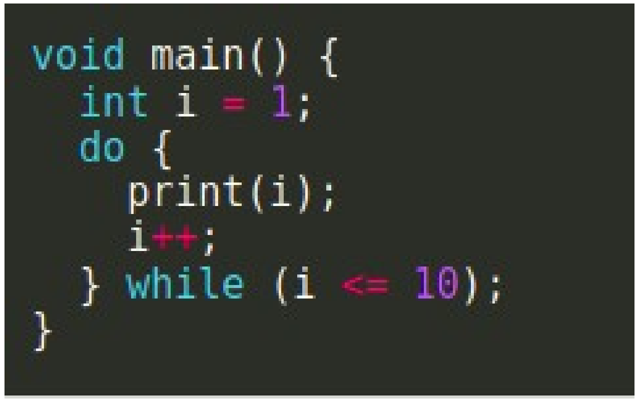
Loops



```
void main(){
  for(int i=50; i<=100; i++){
    if(i%2 == 0){
      print(i);
    }
  }
}
```



```
void main() {
  int i = 10;
  while (i >= 1) {
    print(i);
    i--;
  }
}
```



```
void main() {
  int i = 1;
  do {
    print(i);
    i++;
  } while (i <= 10);
}
```

- Reference

  https://dart-tutorial.com/introduction-and-basics/
-
- https://dartpad.dev/