R2S
Resource Software Solution

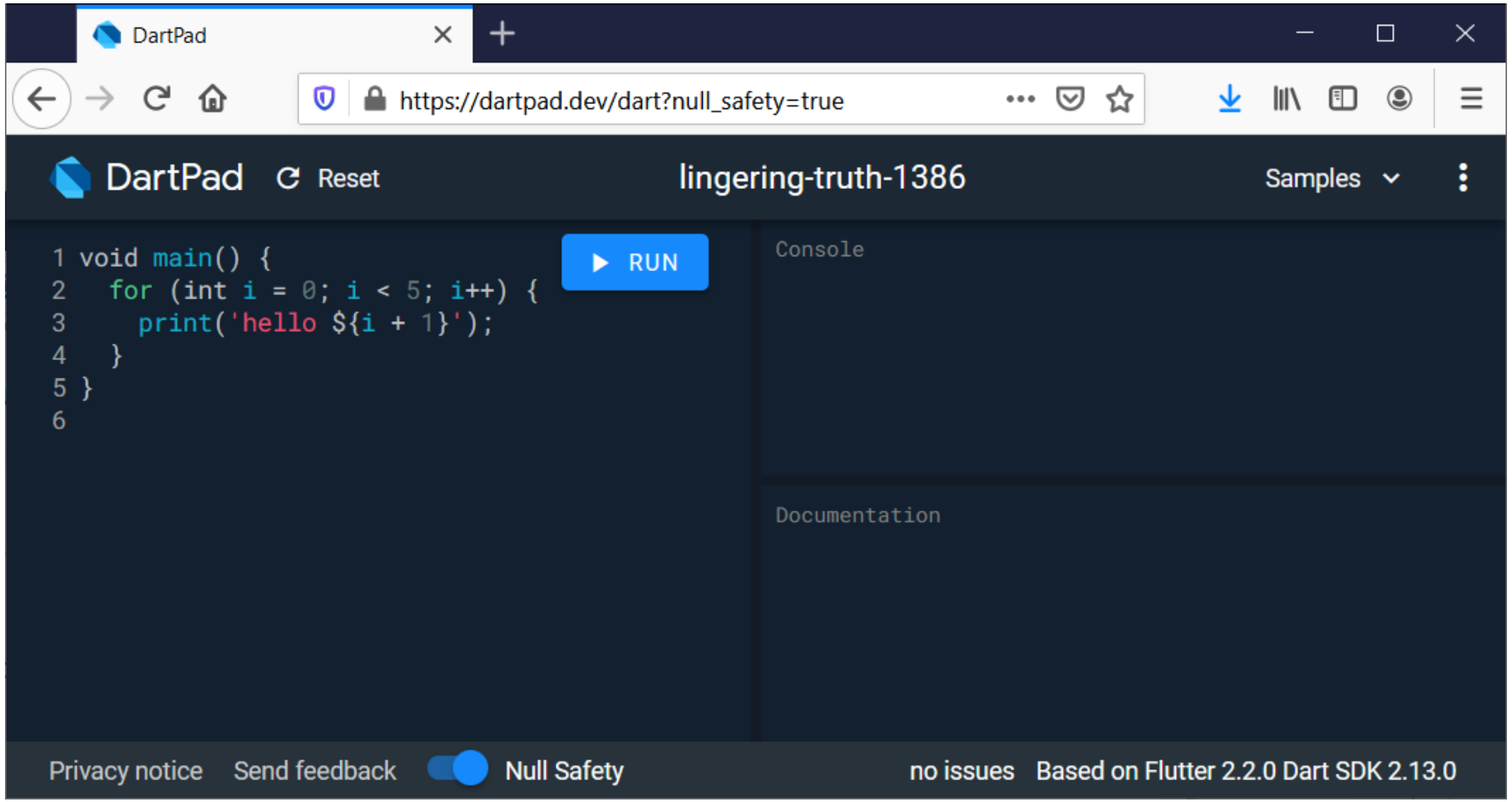
Dart Language

Contents

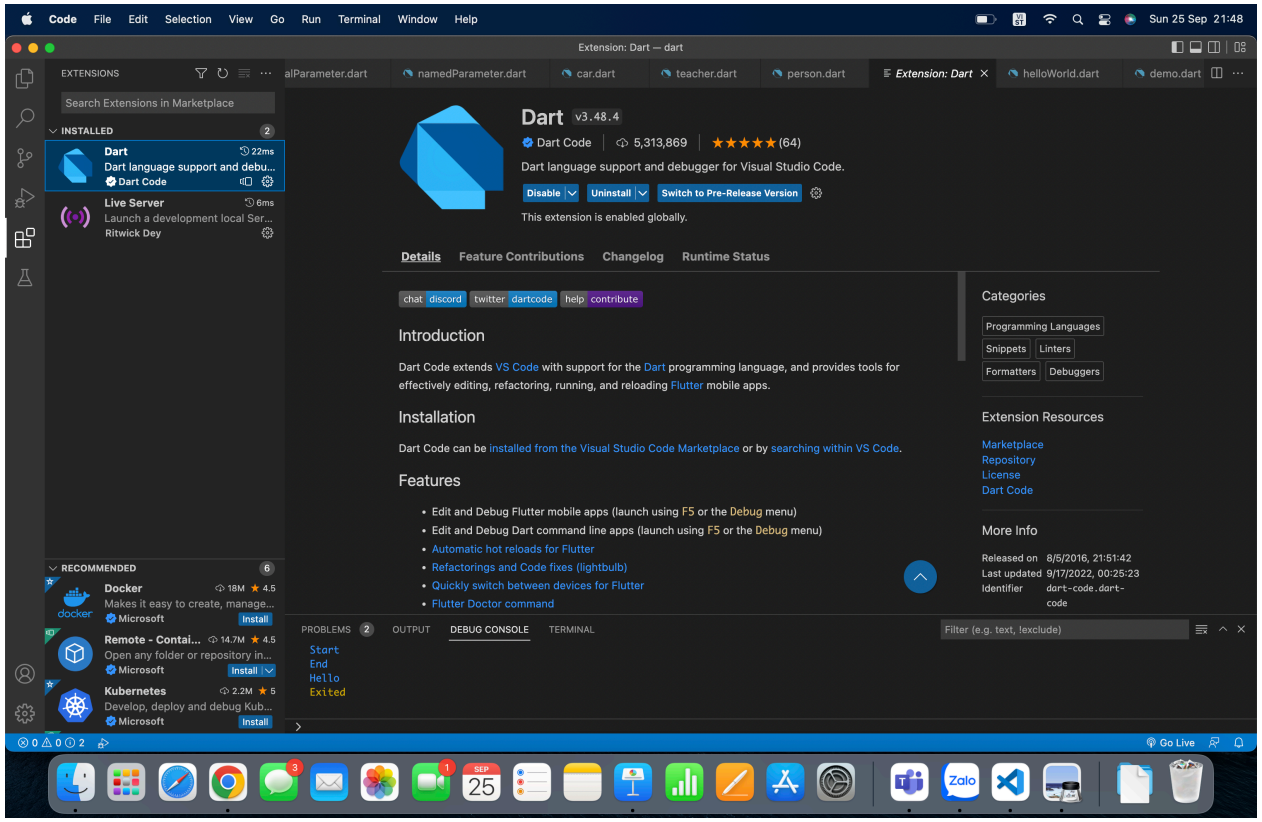
1. Dart basic
2. Functions in Dart
3. OOP in Dart
4. Async Programming

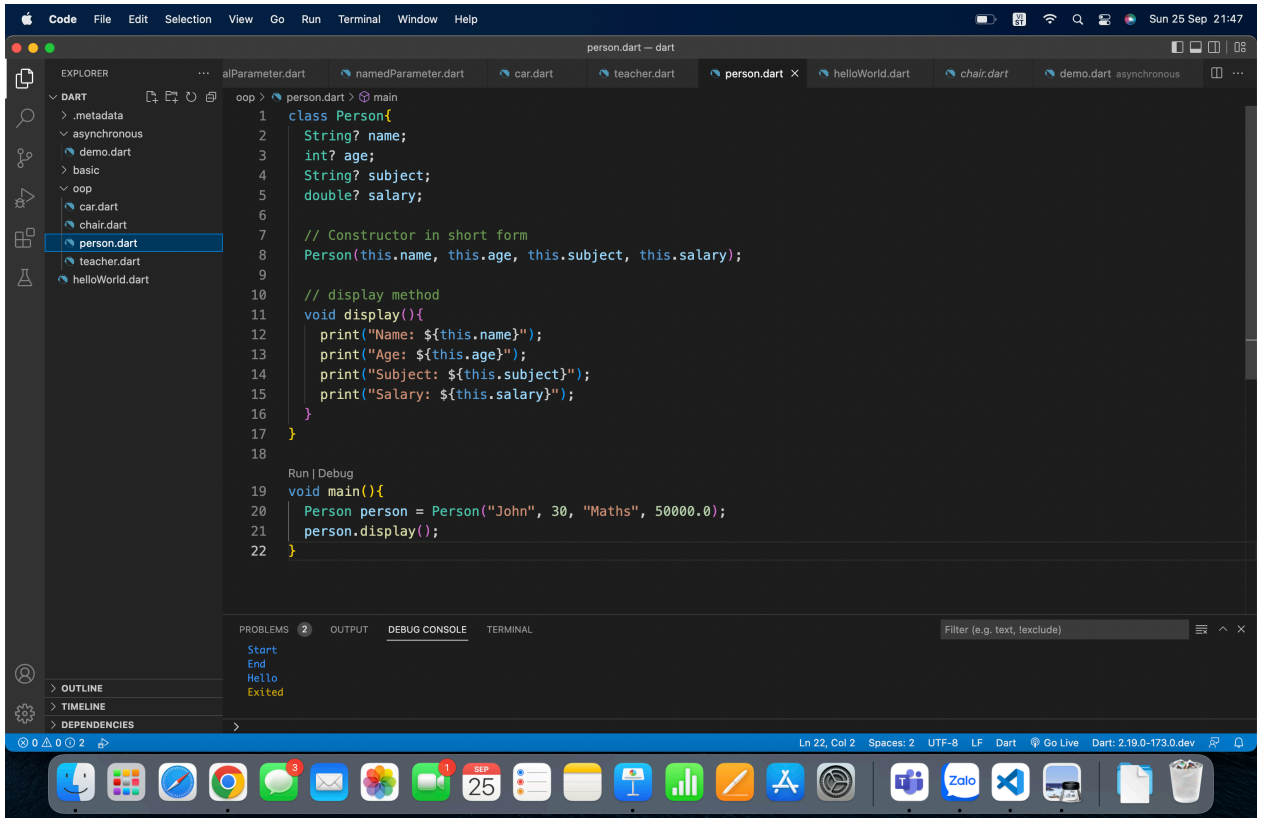
Online IDE

- Url: https://dartpad.dev



Visual Studio Code IDE





Users > kyle > Library > Application Support > Code > User > {} settings.json > abc dart.sdkPath

```
{
    "editor.fontSize": 16,
    "security.workspace.trust.untrustedFiles": "open",
    "editor.minimap.enabled": false,
    "dart.sdkPath": "/Users/kyle/Documents/R2S/Training/Materials/Flutter/Dart language/dart-sdk"
}
```

General

- Naming convention: lowerCamelCase
- Comment code: like JavaScript, C, C# or Java
- main() entry point

```
1  void main() {
2    // Incorrect Way
3    var fullname = "John Doe";
4    // Correct Way
5    var fullName = "John Doe";
6    const pi = 3.14;
7
8    print(fullName);
9  }
```

Variables 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.

```
// here variable name contains value John.
var name = "John";
```

- Types Of Variables
  - **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 value. E.g. 10.0, -10.2, 85.698 [Decimal is included]
  - **num**: For storing any types of number. E.g. 10, 20.2, -20 [both int and double]
  - **bool**: For storing true or false value. E.g. true, false [Only stores true or false values]
  - **var**: For storing any value. E.g. ‘Bimal’, 12, ‘z’, true

Variables in Dart

- Syntax
- Example

type variableName [= value];

```
void main() {
    //Declaring Variables
    String name = "John";
    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("Age is $age");
    print("Height is $height");
    print("Married Status is $isMarried");
}
```

Constant in Dart

- Constant is the **type of variable** whose **value never changes**. In programming, changeable values are mutable and unchangeable values are **immutable**.
- 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 in Dart

- Numbers

```
int counter = 0;
double price = 0.0;
price = 125.00;
```

- Strings

```
// Strings
String defaultMenu = 'main';

// String concatenation
String combinedName = 'main' + ' ' + 'function';
String combinedNameNoPlusSign = 'main' ' ' 'function';

// String multi-line
String multilineAddress = '''
  123 Any Street
  City, State, Zip
''';
```

Data Types in Dart

- Booleans

```
// Booleans
bool isDone = false;
isDone = true;
```

- Lists

```
// Lists - In Dart List is an array
List listOfFilters = ['company', 'city', 'state'];
listOfFilters.forEach((filter) {
   print('filter: $filter');
});
// Result from print statement
// filter: company
// filter: city
// filter: state
```

Data Types in Dart

Resource Software Solution

- Maps

```
// Maps - An object that associates keys and values.
// Key: Value - 'KeyValue': 'Value'
Map mapOfFilters = {'id1': 'company', 'id2': 'city', 'id3': 'state'};

// Change the value of third item with Key of id3
mapOfFilters['id3'] = 'my filter';

print('Get filter with id3: ${mapOfFilters['id3']}');
// Result from print statement
// Get filter with id3: my filter
```

Data Types in Dart

- var keyword: In dart, **var** automatically finds a data type. In simple terms, var says if you don't want to specify a data type, I will find a data type for you.

```
void main() {
    var name = "John Doe"; // String
    var age = 20; // int

    print(name);
    print(age);
}
```

Data Types in Dart

- Dynamically typed

```
void main() {
  dynamic myVariable = 50;
  myVariable = "Hello";
  print(myVariable);
}
```

- Var typed

```
void main() {
  var myVariable = 50; // You can also use int instead of var
  myVariable = "Hello"; // this will give error
  print(myVariable);
}
```

Operators in Dart

- Types Of Operators
  1. Arithmetic Operators
  2. Increment and Decrement Operators
  3. Assignment Operators
  4. Logical Operators
  5. Type Test Operators

Arithmetic Operators

Resource Software Solution

Operator Symbol	Operator Name	Description
+	Addition	For adding two operands
–	Subtraction	For subtracting two operands
–expr	Unary Minus	For reversing the sign of the expression
*	Multiplication	For multiplying two operands
/	Division	For dividing two operands and give output in double
~/	Division	For dividing two operands and give output in integer
%	Modulus	Remainder After Integer Division
++	Increment	Increase Value By 1. For E.g a++;
––	Decrement	Decrease Value By 1. For E.g a–;

Arithmetic Operators

- Example

```
1  void main() {
2    // declaring two numbers
3    int num1=10;
4    int num2=3;
5
6    // performing arithmatic calulation
7    int sum=num1+num2;        // addition
8    int diff=num1-num2;       // subtraction
9    int unaryMinus = -num1;     // unary minus
10   int mul=num1*num2;        // multiplication
11   double div=num1/num2;     // division
12   int div2 =num1~/num2;      // integer division
13   int mod=num1%num2;        // show remainder
14
15   //Printing info
16   print("The addition is $sum.");
17   print("The subtraction is $diff.");
18   print("The unary minus is $unaryMinus.");
19   print("The multiplication is $mul.");
20   print("The division is $div.");
21   print("The integer division is $div2.");
22   print("The modulus is $mod.");
23 }
```

Increment and Decrement Operators

Resource Software Solution

Operator Symbol	Operator Name	Description
`++var`	Pre Increment	Increase Value By 1. var = var + 1 **Expression value is var+1**
`--var`	Pre Decrement	Decrease Value By 1. var = var - 1 **Expression value is var-1**
`var++`	Post Increment	Increase Value By 1. var = var + 1 **Expression value is var**
`var--`	Post Decrement	Decrease Value By 1. var = var - 1 **Expression value is var**

Increment and Decrement Operators

- Example

```
1  void main() {
2    // declaring two numbers
3    int num1 = 0;
4    int num2 = 0;
5
6    // performing increment / decrement operator
7
8    // pre increment
9    num2 = ++num1;
10   print("The value of num2 is $num2");
11
12   // reset value to 0
13   num1 = 0;
14   num2 = 0;
15
16   // post increment
17   num2 =  num1++;
18   print("The value of num2 is $num2");
19
20 }
```

Assignment Operators

Operator Type	Description
=	Assign a value to a variable
+=	Adds a value to a variable
-=	Reduces a value to a variable
*=	Multiply value to a variable
/=	Divided value by a variable

```
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 Aubtraction 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");
}
```

Relational Operators

Resource Software Solution

Operator Symbol	Operator Name	Description
>	Greater than	Used to check which operand is bigger and gives result as boolean
<	Less than	Used to check which operand is smaller and gives result as boolean
>=	Greater than or equal to	Used to check which operand is bigger or equal and gives result as boolean
<=	Less than or equal to	Used to check which operand is smaller or equal and gives result as boolean
==	Equal to	Used to check operands are equal to each other and gives result as boolean
!=	Not equal to	Used to check operand are not equal to each other and gives result as boolean

Logical Operators

Operator Type	Description
&&	This is ‘and’, return true if all conditions are true
||	This is ‘or’. Return true if one of the conditions is true
!	This is ’not’. return false if the result is true and vice versa

```
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 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

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

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

Condition in Dart

- if else

```
if (condition1) {
   statements1;
} else if (condition2) {
   statements2;
} else if (condition3) {
  statements3;
}
.
.
else {
   statementsN;
}
```

```dart
void main() {
  int num1 = 1200;
  int num2 = 1000;
  int num3 = 150;

  if(num1 > num2  && num1 > num3){
      print("Num 1 is greater: $num1");
  }

  if(num2 > num1 && num2 > num3){
      print("Num2 is greater: $num2");
  }

  if(num3 > num1 && num3 > num2){
      print("Num3 is greater: $num3");
  }
}
```

Condition in Dart

- Ternary operator **condition ? exprIfTrue : exprIfFalse**
- If Else Vs Ternary Operator

```
void main() {
  int num1 = 10;
  int num2 = 15;
  int max = 0;
  if (num1> num2) {
    max = num1;
  } else {
    max = num2;
  }
  print("The greatest number is $max");
}
```

```
void main() {
  int num1 = 10;
  int num2 = 15;
  int max = (num1 > num2) ? num1 : num2;
  print("The greatest number is $max");
}
```

Condition in Dart

- switch case

```
switch (expression) {
    case value1:
        // statements
        break;
    case value2:
        // statements
        break;
    case value3:
        // statements
        break;
    default:
        // default statements
}
```

```dart
void main() {
  var dayOfWeek = 5;
  switch (dayOfWeek) {
    case 1:
      print("Day is Sunday.");
      break;
    case 2:
      print("Day is Monday.");
      break;
    case 3:
      print("Day is Tuesday.");
      break;
    case 4:
      print("Day is Wednesday.");
      break;
    case 5:
      print("Day is Thursday.");
      break;
    case 6:
      print("Day is Friday.");
      break;
    case 7:
      print("Day is Saturday.");
      break;
    default:
      print("Invalid Weekday.");
      break;
  }
}
```

Loops in Dart

- for loop

```
// Standard for loop
List listOfFilters = ['company', 'city', 'state'];
for (int i = 0; i < listOfFilters.length; i++) {
   print('listOfFilters: ${listOfFilters[i]}');
}
// Result from print statement
// listOfFilters: company
// listOfFilters: city
// listOfFilters: state
```

- for-in

```
// or for-in loop
List listOfNumbers = [10, 20, 30];
for (int number in listOfNumbers) {
   print('number: $number');
}
// Result from print statement
// number: 10
// number: 20
// number: 30
```

Loops in Dart

- while

```
while (condition) {
      // statement(s);
}
```

- do-while

```
do {
   // statement(s)
} while (condition);
```

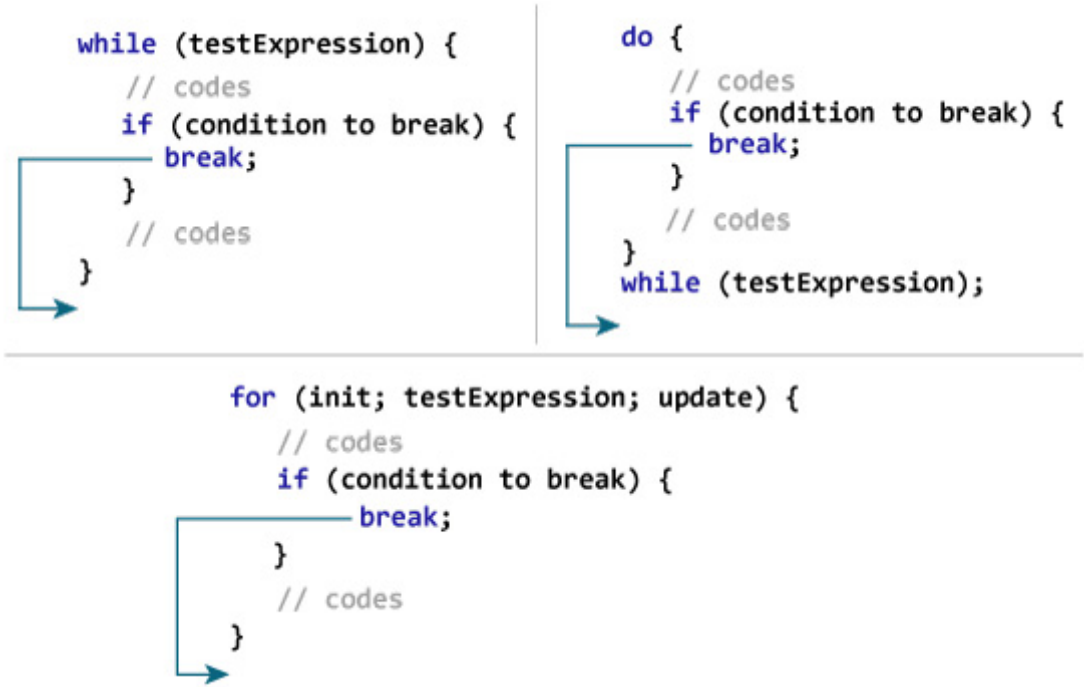
```
void main() {
  int i = 1;
  while (i <= 10) {
    print(i);
    i++;
  }
}
```

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

break Statement

- Sometimes you will need to break out of the loop immediately without checking the condition. You can do this using **break statement**.

```
void main() {
  for (int i = 1; i <= 10; i++) {
    if (i == 5) {
      break;
    }
    print(i);
  }
}
```



continue Statement

- Sometimes you will need to skip an iteration for a specific condition. You can do this utilizing **continue statement**.

```
void main() {
  for (int i = 1; i <= 10; i++) {
    if (i == 5) {
      continue;
    }
    print(i);
  }
}
```

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Functions in Dart

- They are created when some **statements** are **repeatedly** occurring in the program. The function helps **reusability of the code** in the program.
- Syntax

```
returntype functionName (parameter1,parameter2, ...) {
  // function body
}
```

```dart
// this function add two numbers
int add(int a, int b) {
  int sum = a + b;
  return sum;
}

void main() {
  int num1 = 10;
  int num2 = 20;

  int total = add(num1, num2);
  print("The sum is $total.");
}
```

Functions in Dart

- Providing **default value** on parameter

```dart
void add (int num1, int num2, [int num3=0]) {
  int sum;
  sum = num1 + num2 + num3;

  print("The sum is $sum");
}

Run | Debug
void main(){
  add(10, 20);
  add(10, 20, 30);
}
```

Functions in Dart

- To use **named parameters** to clarify the parameter's meaning in function calls. **Curly braces {}** are used to specify named parameters.
- In the example below, function **printInfo** takes two named parameters. You can pass value in any order.

```dart
void printInfo({String? name, String? gender}) {
  print("Hello $name your gender is $gender.");
}

void display({required String name, required String gender}) {
  print("Hello $name your gender is $gender.");
}

Run | Debug
void main() {
  // you can pass values in any order in named parameters.
  printInfo(gender: "Male", name: "John");
  printInfo(name: "Sita", gender: "Female");
  display(name: "Kỳ Lê", gender: "Male");
}
```

```dart
void printInfo({String name = "", String gender = ""}) {
  print('Hello $name your gender is $gender');
}
```

OOP in Dart

- **Object-oriented programming** (OOP) is a programming method that **uses objects** and their interactions to design and program applications
- In **OOP**,
  1. An object can be anything, such as a **person**, a **bank account**, a **car**, or a **house**.
  2. Each object has its attributes (or **properties**) and behavior (or **methods**).
  3. For example, a **person object** may have the attributes **name**, **age** and **height**, and the behavior **walk** and **talk**.
- Features Of OOP
  1. Class
  2. Object
  3. Constructor
  4. Abstraction

Class in Dart

- In object-oriented programming, a class is a **blueprint** for **creating objects**. A class defines the properties and methods that an object will have.
- For example, a class called **Student** might have properties like **name**, **age** and methods like **study**, **exam**.

Declaring Class in Dart

- You can declare a class in dart using the **class** keyword followed by class name and braces {}.
- Syntax

```
class ClassName {
  // properties or fields
  // methods or functions
}
```

```
class Person {
  String? name;
  String? phone;
  bool? isMarried;
  int? age;

  void displayInfo() {
    print("Person name: $name.");
    print("Phone number: $phone.");
    print("Married: $isMarried.");
    print("Age: $age.");
  }
}
```

Declaring Class in Dart

- Challenge: Create a class **Book** with three properties: **name**, **author**, and **prize**. Also, create a method called **display**, which prints out the values of the three properties.

Object in Dart

- An object is an **instance** of a class. You can create **multiple objects** of the **same class**
- Syntax:

ClassName objectName = ClassName();

```dart
class Car {
  String? name;
  String? color;
  int? numberOfSeats;

  void start() {
    print("$name Car Started.");
  }
}

void main(){
    // Here car is object of class Car.
    Car car = Car();
    car.name = "BMW";
    car.color = "Red";
    car.numberOfSeats = 4;
    car.start();

    // Here car2 is another object of class Car.
    Car car2 = Car();
    car2.name = "Audi";
    car2.color = "Black";
    car2.numberOfSeats = 4;
    car2.start();
}
```

Object in Dart

- Challenge: Create a Class Camera with properties: **name**, **color**, **megapixel**. Create a method called **display** which prints out the values of the three properties. Create two objects of the class Camera and call the method display.

Constructor in Dart

- **A constructor** is a special method used to initialize an object. It is called automatically when an object is created, and it can be used to set the initial values for the object's properties.
- For example, the following code creates a **Person** class object and sets the initial values for the **name** and **age** properties.

```
Person person = Person("John", 30);
```

Constructor in Dart

- Without Constructor: If you don't define a constructor for class, then you need to set the values of the properties manually.
- For example, the following code creates a **Person** class object and sets the values for the **name** and **age** properties.

```
Person person = Person():
person.name = "John";
person.age = 30;
```

Constructor in Dart

- Things to remember
  1. The constructor's name should be the same as the class name.
  2. Constructor doesn't have any return type.
- Syntax

```
class ClassName {
  // Constructor declaration: Same as class name
  ClassName() {
    // body of the constructor
  }
}
```

Constructor in Dart

- Example

```dart
class Teacher {
  String? name;
  int? age;
  String? subject;
  double? salary;

  // Constructor
  Teacher(String name, int age, String subject, double salary) {
    this.name = name;
    this.age = age;
    this.subject = subject;
    this.salary = salary;
  }
  // Method
  void display() {
    print("Name: ${this.name}");
    print("Age: ${this.age}");
    print("Subject: ${this.subject}");
    print("Salary: ${this.salary}\n"); // \n is used for new line
  }
}

void main() {
  // Creating teacher1 object of class Teacher
  Teacher teacher1 = Teacher("John", 30, "Maths", 50000.0);
  teacher1.display();

  // Creating teacher2 object of class Teacher
  Teacher teacher2 = Teacher("Smith", 35, "Science", 60000.0);
  teacher2.display();
}
```

Constructor in Dart

- Example: Write constructor single line

```dart
class Person{
  String? name;
  int? age;
  String? subject;
  double? salary;

  // Constructor in short form
  Person(this.name, this.age, this.subject, this.salary);

  // display method
  void display(){
    print("Name: ${this.name}");
    print("Age: ${this.age}");
    print("Subject: ${this.subject}");
    print("Salary: ${this.salary}");
  }
}

void main(){
  Person person = Person("John", 30, "Maths", 50000.0);
  person.display();
}
```

Constructor in Dart

- Example: Constructor with named parameters

```dart
class Chair {
  String? name;
  String? color;

  // Constructor
  Chair({this.name, this.color});

  // Method
  void display() {
    print('Name: $name');
    print('Color: $color');
  }
}

Run | Debug
void main() {
  Chair chair = Chair();
  chair.display();

  Chair chair2 = Chair(name: 'Chair', color: 'Red');
  chair2.display();
}
```

Constructor in Dart

- Challenge: Create a class **Patient** with three properties **name**, **age**, and **disease**. The class has one constructor. The constructor is used to initialize the values of the three properties. Also, create an object of the class **Patient** called **patient**. Print the values of the three properties using the object.

Abstraction in Dart

- Abstraction is the class that contains one or more or may not contain **abstract methods** (methods without implementation).
- Key Features
  1. Abstract class cannot be **initialized**.
  2. It is declared with **abstract** keyword.
  3. It can be inherited using the **extend** keyword then the **abstract methods** need to be **implemented**.

Abstraction in Dart

- Syntax

```
abstract class ClassName {
  //Body of abstract class

  method1();
  method2() {
    //body method
  }
}
```

```dart
abstract class Vehicle {
  void printVehicleName();

  void printSupplierName() {
    print('Supplier: TATA Motors');
  }
}
```

```dart
class Truck extends Vehicle {
  @override
  void printVehicleName() {
    print('This is a truct');
  }
}
```

```dart
import 'truck.dart';

Run | Debug
void main() {
  var truck = Truck();
  truck.printVehicleName();
  truck.printSupplierName();
}
```

Asynchronous Programming in Dart

Resource Software Solution

- What is synchronous programming

The program is **executed line by line**, one at a time. Synchronous operation means a task that needs to be solved before proceeding to the next one.

- Example of synchronous programming

```
main() {
    print("First Operation");
    print("Second Big Operation");
    print("Third Operation");
    print("Last Operation");
}
```

Asynchronous Programming in Dart

Resource Software Solution

- What is asynchronous programming

Program execution continues to the next line **without** waiting to complete other work. It simply means **Don't wait**. It represents the task that doesn't need to solve before proceeding to the next one.

- Example of synchronous programming

```
main() {
    print("First Operation");
    Future.delayed(Duration(seconds:3),()=>print('Second Big Operation'));
    print("Third Operation");
    print("Last Operation");
}
```

Asynchronous Programming in Dart

- Why we need Asynchronous
  1. To fetch data from Internet,
  2. To write something to Database,
  3. To read data from File, and
  4. To download file etc.

Asynchronous Programming in Dart

- Async and Await in Dart
  1. You can use the **async** keyword before a function body to make it **asynchronous.**
  2. You can use the **await** keyword to get the completed **result of an asynchronous** expression.

```
main() {
      print("Start");
      getData();
      print("End");
}

void getData() async {
  String data = await middleFunction();
  print(data);
}

Future<String> middleFunction(){
  return Future.delayed(Duration(seconds:5), ()=> "Hello");
}
```