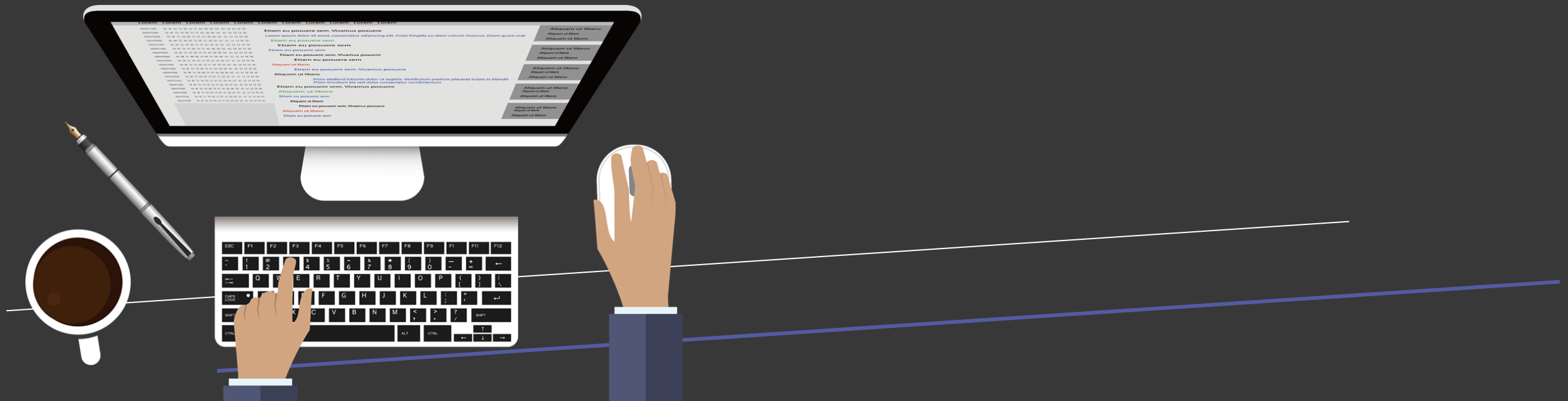
A Comprehensive Guide to Dart Programming:

Object-Oriented Principles, Functional Concepts



Object-Oriented Programming in Dart

- Dart is an object-oriented programming language.
- The main goal of oops concepts in dart is to reduce programming complexity & can execute many tasks at once.
- Object-Oriented Programming (OOP) in Dart offers several benefits, making it easier to build scalable, maintainable, and reusable software.

Oops

- Defining classes and creating objects
- Understanding class members: properties, methods, and constructors
- The Three Pillars of OOPs:
  - Inheritance
  - Ploymorphism
  - Abstraction
  - Encapsulation

Class

A class in terms of Oops is the blueprint of objects.But compared to other languages, class have less importance in dart.Class can be used for declaring, allocating space, storing, sharing objects.

The class keyword is followed by the class name. The rules for identifiers must be considered while naming a class.

Inside Class there are 4 types of attributes:

- fields
- getters/setters
- Constructors
- functions

```
class class_name {
   <fields>
   <getters/setters>
   <constructors>
   <functions>
}
```

Inside Class there are 4 types of attributes:

- fields : Fields are variables that store data for objects.
- getters/setters :getters are methods that retrieve the value of a property, while setters are methods that update the value of a property, helping to control access to class fields.
- Constructors :Constructors are special functions that create instances of classes.
- functions :Functions refers to the action an object can take. It can also be called as methods.

```
class Student{

   // field
  String name =”Name”;

  //function
   void detail(){

    print(name);

}

}
```

Object

Objects are instances of classes and have properties (attributes) and methods (actions).

Eg:
Here we created a class with one field(str) and one function (print_str)

On main method we declared a variable(obj) with value equal to that class name

And we accessed that class field and function through that variable.

```
Run | Debug
3  void main(List<String> arguments)
4  {
5    var obj = name();
6    print('Calling a field from main method ------- ${obj.str}');
7    obj.print_str();
8  }
9
10 class name
11 {
12   String str = 'akshay';
13   void print_str()
14   {
15     print('Excuting a function from initialised class ------- $str');
16   }
17 }
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL
Calling a field from main method ------- akshay
Excuting a function from initialised class ------- akshay
Exited
```

```
//accessing an attribute
obj.field_name

//accessing a function
obj.function_name()
```

Functions

- Functions are the building blocks of readable, maintainable, and reusable code.

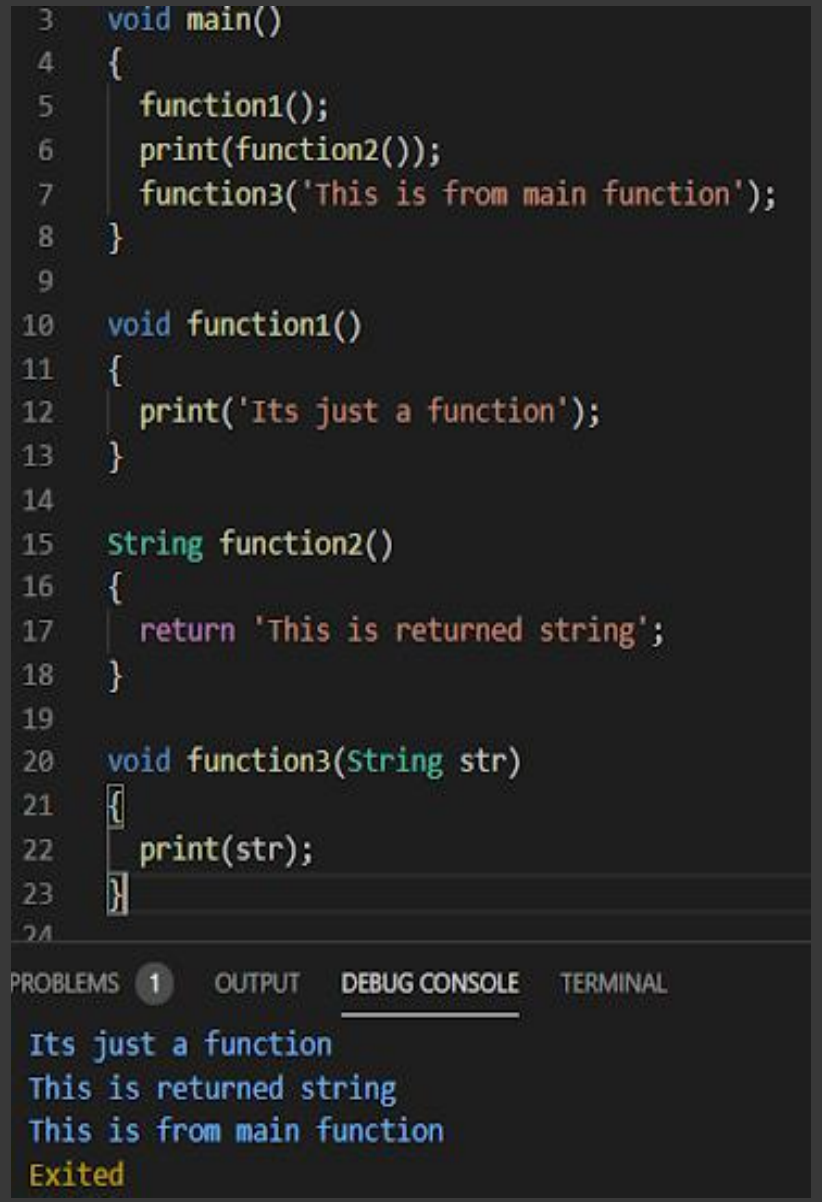
Function

```
Return_typefuc_name(parameter_list){

        //statement(s)

         return value;
}
```

- Defining a function - a function definition requires when and how a specific task would be done.
- Calling a function - A function must be called in order to execute it.
- Returning functions - functions may also return values back along with the control, to the caller.
- Parameterised functions - Parameters are a mechanism to pass values between the function.we can pass more than one parameter or what our object we want. But to receive, we must define data type.



Aitrich Academy

Parameter function Types

- **Optional positional parameter** - To specify optional positional parameters, use square [] brackets
- **Optional named parameter** - Unlike positional parameters, the parameter's name must be specified while the value is being passed. Curly brace {} can be used to specify optional named parameters.
- **Optional parameter with default value** - Function parameters can also be assigned values by default. However, such parameters can also be explicitly passed values.

```dart
void main()
{
  PositionedParameters('Parameter 1', 'parameter 2');
  PositionedParameters('Parameter 1');
  print('');
  NamedParameters(two: 'parameter 2', one: 'parameter 1');
  print('');
  DefaultParameter('parameter 1');
  DefaultParameter('parameter 1', two: 'parameter 2');
}

void PositionedParameters(String one, [String two]) {
  print(one);
  print(two);
}

void NamedParameters({String one, String two}) {
  print('Parameter one : $one');
  print('parameter two : $two');
}

void DefaultParameter(String one, {String two = 'hyy'}) {
  print('parameter one : $one');
  print('parameter two : $two');
}
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL

```
Parameter 1
parameter 2
Parameter 1
null

Parameter one : parameter 1
parameter two : parameter 2

parameter one parameter 1
parameter two hyy
parameter one parameter 1
parameter two parameter 2
Exited
```

Constructors

- Constructors are function which we will create with the name of class itself and,
- Which will execute automatically when the class is called.
- There are named constructors in dart, which will allow us to create and call by the default name we create. This constructor will only execute only if its called by that name.

```dart
void main(List<String> args) {
  var x = cs();
  x.function();
  var xx = cs.name();
  xx.function();
}

class cs {
  cs() {
    print('This is from Default Contructor');
  }
  cs.name() {
    print('This is from named constructor');
  }
  void function() {
    print('This is from function');
  }
}
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL

```
This is from Default Contructor
This is from function
This is from named constructor
This is from function
Exited
```

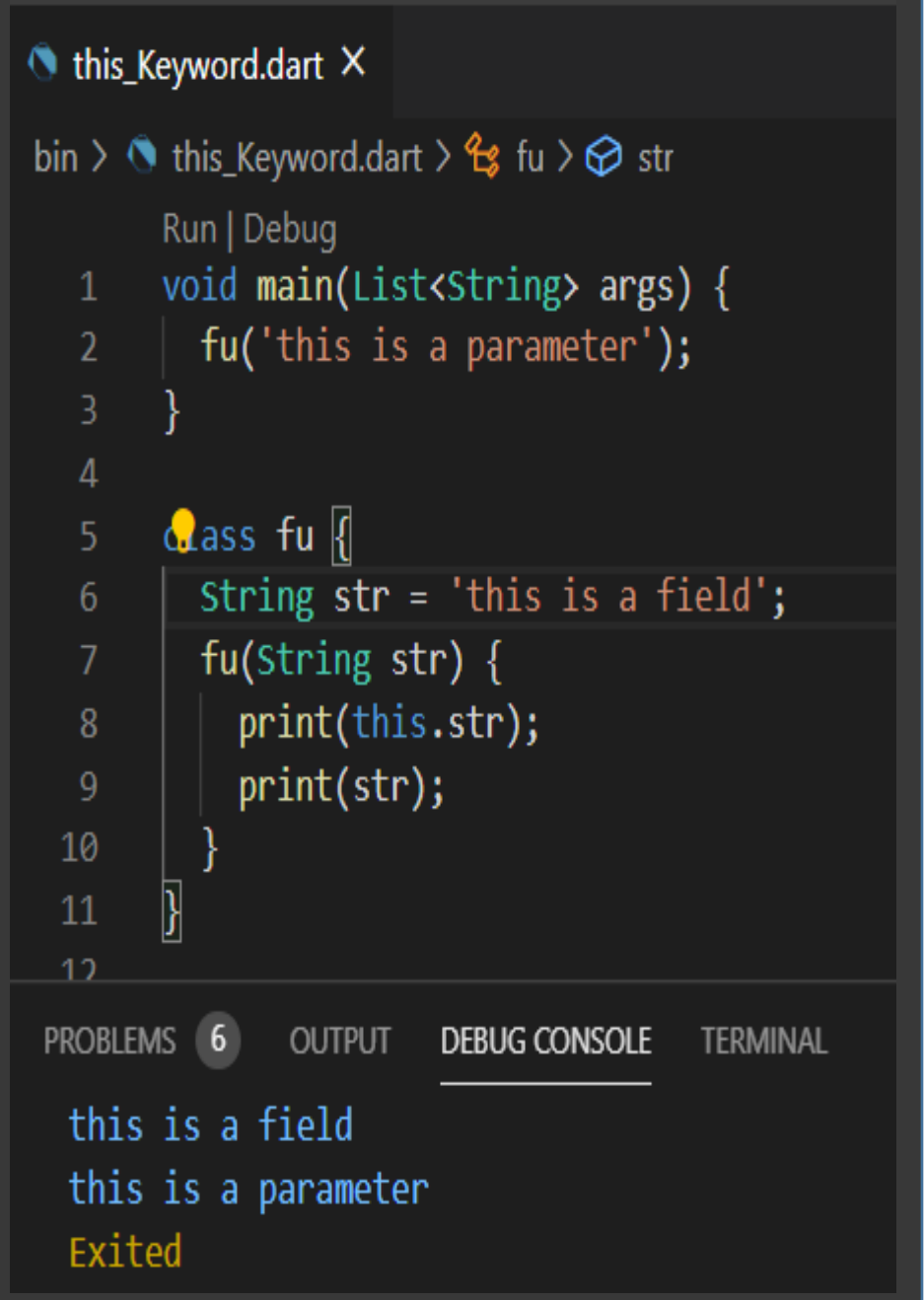
this keyword

The keyword this refers to the current instance of the class.

In the example, we have two string values with name str,

One already defined in the class
Second is received from the parameter

By only using this keyword we printed the value from the class.



Setters/Getters

- Getters or accessors are defined using get keyword.
- Setters or mutators are defined using set keyword.

(The setter must be executed first, otherwise getters will get a null value)

```
3   void main() {
4     var obj = cs();
5     obj.Name = 'AKR';
6     print(obj.Name);
7   }
8
9   class cs {
10    String name;
11
12    set Name(String s) {
13      this.name = s;
14    }
15
16    String get Name {
17      return name;
18    }
19  }
20
```

PROBLEMS 9 OUTPUT DEBUG CON

```
AKR
Exited
```

Class Inheritance

Class Inheritance is used to access a class(parent class) by another class (child class).

The inherited class objects by making the object of derived class.

Three are commonly used Inheritance:

- ✓ Single Inheritance
- ✓ Multi-level Inheritance
- ✓ Hierarchial Inheritance

```
bin > Inheritance.dart > cs2
    Run | Debug
1   void main(List<String> args)
2   {
3     var obj = cs();
4     print(obj.cs_String);
5     print(obj.str);
6     print(obj.str1);
7   }
8
9   class cs extends cs1
10  {
11    String cs_String = 'This is from derived class';
12  }
13
14  class cs1 extends cs2
15  {
16    String str = 'This is from class one';
17  }
18
19  class cs2
20  {
21    String str1 = 'This is from class two';
22  }
```

```
PROBLEMS 7   OUTPUT   DEBUG CONSOLE   TERMINAL
This is from derived class
This is from class one
This is from class two
Exited
```

Super keyword

- super.myFunction() calls the parent's version of the function from within the child class.
- This allows for both shared functionality (in the parent) and custom implementation (in the child).
- The order of execution is determined by the placement of super.myFunction() within the child's function.

```
1   void main(List<String> args)
2   {
3     var obj = Parentclass();
4     obj.function();
5   }
6   class Parentclass extends ChildClass {
7     void function()
8     {
9       print('This is from Derived class');
10    }
11  }
12  class ChildClass
13  {
14    void function() {
15      print('This is from Child class');
16    }
17  }
18
```

```
PROBLEMS 8  OUTPUT  DEBUG CONSOLE  TERMINAL
This is from Derived class
Exited
```

```
1   void main(List<String> args)
2   {
3     var obj = Parentclass();
4     obj.function();
5   }
6   class Parentclass extends ChildClass {
7     void function()
8     {
9       super.function();
10      print('This is from Derived class');
11    }
12  }
13  class ChildClass
14  {
15    void function() {
16      print('This is from Child class');
17    }
18  }
```

```
PROBLEMS 8  OUTPUT  DEBUG CONSOLE  TERMINAL
This is from Child class
This is from Derived class
Exited
```

Abstract class and method

Abstract Method

- To make a method abstract, use semicolon (;)Instead of method body
- Abstract method can only exit with Abstract class
- You need to override Abstract methods in subclass

Abstract Class

- Use abstract keyword to declare Abstract Class.
- Abstract class can have Abstract Methods, Normal Methods and Instance Variables as well
- The Abstract Class cannot be instantiated, you cannot create objects. But can be inherited

```
Run | Debug
1  void main() {
2    int x = 10, y = 5;
3    Answer().getValue(x, y);
4    Answer().addition(x, y);
5    Answer().subtraction(x, y);
6  }
7
8  abstract class Calculation {
9    void getValue(int x, int y);
10
11   void addition(int x, int y) {
12     print('Addition ${x + y}');
13   }
14
15   void subtraction(int x, int y) {
16     print('Subtraction ${x - y}');
17   }
18 }
19
20 class Answer extends Calculation {
21   @override
22   void getValue(int x, int y) {
23     print('x = $x, y = $y');
24   }
25 }
26
```

```
PROBLEMS 18  OUTPUT  DEBUG CONSOLE  TERMINAL
Connecting to VM Service at ws://127.0.0.1:5811
x = 10, y = 5
Addition 15
Subtraction 5
Exited
```

Polymorphism

Polymorphism is the ability of an object to take many forms.
There are two types of polymorphism

- Compile time
- Run time

The example here shows the ability of an object name obj to take many forms.

Interface

- Dart does not have special syntax to declare INTERFACE
- An INTERFACE in dart is Normal Class
- An INTERFACE is used when you need concrete implementation of all its function within the subclass, it is mandatory to override all methods in implementing class.
- You can implement multiple classes but you cannot extend multiple classes during inheritance.

(Interface importance comes in flutter when we go to cross platforms)

```dart
Run | Debug
void main(List<String> args) {
  Mobile();
}

abstract class Message {
  void text();
  void number();
}

abstract class Call {
  void call();
  void recievecall();
}

class Mobile implements Message, Call {
  Mobile() {
    print('The mobile must do these functions');
  }
  @override
  void call() {}

  @override
  void number() {}

  @override
  void recievecall() {}

  @override
  void text() {}
}
```

Effective Dart: Writing Clean and Maintainable Code

A

Write clean, readable, and maintainable Dart code by following best practices.

B

Employing effective naming conventions, code formatting, and documentation

C

Utilizing code review processes to refine code quality and consistency

Conclusion

- Recap of the key takeaways from the presentation:
  - Class & Object
  - Constructors
  - Parameter function

- The Three Pillars of OOPs:
  - Inheritance
  - Ploymorphism
  - Abstraction