dart_programming

1. Entry Point of Dart Programs

- Every Dart program starts with the `main()` function.
- Example:

```dart
void main() {
  print('Welcome to Dart!');
}
```

- It's the entry point of a dart program, when you compile a dart code it starts from that main() fxn

2. Comments

- Text that is ignored by the compiler.
- Can be used for documentation purposes or to explain code

```dart
//This is a Single line comment

/*
This is a
multiline comment
*/

/// This is also a comment
```

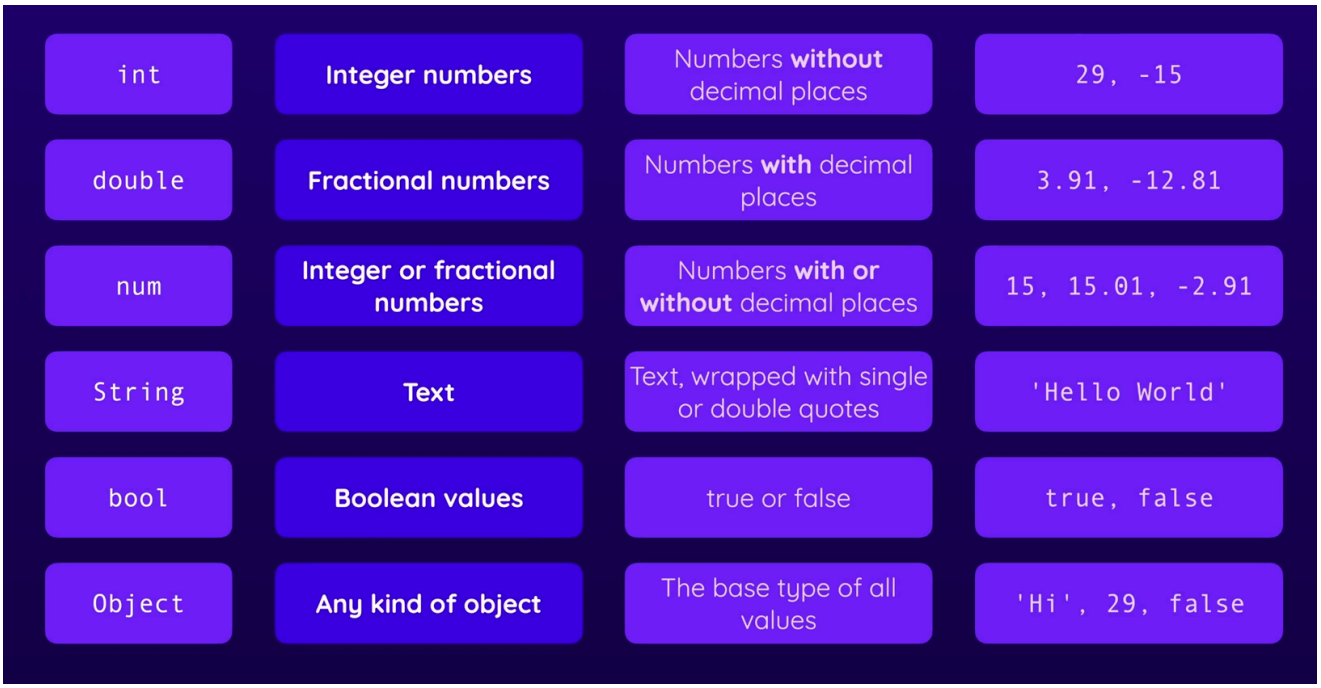
3. Data Types

- Dart is a type safe language
- All value are of certain types
- Dart can be flexible with `dynamic`.
- Common types:
  - `int`: Whole numbers (e.g., `42`).
  - `double`: Decimal numbers (e.g., `3.14`).
  - `String`: Text (e.g., `'Hello'`).
  - `bool`: True/False values (e.g., `true`).
  - `List`: Collection of items (e.g., `[1, 2, 3]`).
- Example:

```dart
    int age = 25;
    double price = 19.99;
    String name = 'Alice';
    bool isStudent = true;
    List<int> scores = [95, 88, 76]; // List of numbers

        //We can have list of mix data types also
        List<dynamic> randomData = ['Hello',12,1.99];
        // Here we have set that the data types of items can be dynamic
```

- Core Data Types in Dart



4.String Data Type and String Interpolation

- Strings are sequences of characters.
- Dart Strings Use Unicode to represent characters, supporting multiple languages and symbols.(assamese,hindi)

```dart
String greeting = 'Hello, 🌍!';
// String greeting = 'Hello,\u{1F30D}' also does the same
print(greeting); // Output: Hello, 🌍!
```

- Use `${}` to embed variables or expressions. (Like javascript)
- Example:

```dart
        String name = 'Dart';
    print('Welcome to $name!'); //simple string interpolation
    print('Sum of 2 + 3 = ${2 + 3}'); //embeding expressions
```

5. Variables and Null Safety

- Variables are used to store data.
- Dart is strongly typed, meaning every variable has a type.
- Syntax: `<type> <variableName> = <value>;`
- Example of variables:

```
int age = 25;           // Integer variable
double price = 19.99;   // Floating-point number
String name = 'Alice';  // Text
bool isStudent = true;  // Boolean
```

- You can also use final or var , than dart will decide its data type

```
var score = 95; // Dart infers it as an int
final city = 'NY'; // Cannot be changed later
```

Null Safety in Dart

- Dart ensures variables are `non-nullable by default`.
- Non nullable variables needs to gets initailized if not done the compiler will show error

```
int val;
print(val);// will show error on compilation
int? val;
print(val)// will not show error :prints null
```

- In the code above i used ? that refers that the variable can be null or nullable

```
int? age = null; // Nullable
int year = 2024; // Non-nullable
```

Late Keyword

- Declares a variable that will be initialized later.
- Useful when you're sure the variable will be assigned before use.

```
late String description;
description = 'This is Dart';
```

```
print(description); // Output: This is Dart
```

- Its your responsibilty to initialize the variable before use , otherwise will show error

6. Flow Control: If-Else

- Used for decision-making.
- Example:

```
    int score = 85;

        if (score > 90) {
            print('Excellent!');
    }else if (score > 70) {
            print('Good Job!');
    }else{
            print('Keep Trying!');
    }
```

- If the condition under if satisfies the code block inside it will execute the else part will not get executed, but if the condition under `if` does not satisfy (i.e., evaluates to `false`), the code block inside `else` will execute instead.

7. Functions

- Reusable blocks of code.
- Syntax:

```
<return-type> fxn-name (parameters){
        // body
        //anything you want to do
}
//paremeters: anything you want to give to the fxn
```

- Example:

```
    int add(int a, int b) {
            return a + b;
    }

    void main() {
```

```
    print(add(3, 4)); // Outputs: 7
}
```

- When we want to use a fxn we made we need to call them as shown above

8. Parameters in Functions

- Used to pass values into functions as shown above.
- Types of parameters in dart
- **Required Positional Parameters:** These parameters are mandatory and must be provided in the exact order they are declared in the function definition.
- Syntax:

```
  void greet(String name) {
  print("Hello, $name!");
}
// While calling this fxn you need to give a paramter of type String
```

2. Optional Positional Parameters

- These parameters are optional and can be omitted during the function call. If not provided, they take a default value (usually `null`).
- Syntax:

```
  void greet(String name, [String title]) {
  if (title != null) {
    print("Hello, $title $name!");
  } else {
    print("Hello, $name!");
  }
}
// Here while calling greet() you can pass only one parameter or two
```

3. Named Parameters

- **Definition:** These parameters are optional and can be provided in any order by specifying their names. They can also have default values.
- Syntax:

```
void greet(String name, {String title = "Mr."}) {
  print("Hello, $title $name!");
```

```
    }
    // Named parameter , is optional if not given Mr will be by default
```

- Three types of parameters combined example

```
void greet(String name, [String title], {String salutation = "Dear"}) {
  // name: Required positional parameter (must be provided)
  // title: Optional positional parameter (can be omitted)
  // salutation: Named parameter (can be omitted or provided with a name)

  if (title != null) {
    print("$salutation $title $name");
  } else {
    print("$salutation $name");
  }
}

void main() {
  greet("Alice"); // Output: Dear Alice
  greet("Bob", "Dr."); // Output: Dear Dr. Bob
  greet("Charlie", salutation: "Hello"); // Output: Hello Charlie
  greet("David", "Prof.", salutation: "Greetings"); // Output: Greetings
Prof. David
}
```

10. Anonymous Functions

- Functions without a name.
- Syntax:

```
(parameters) {
  // Function body
}
```

- Example:

```
void main() {
  // Kind of storing the fxn in a variable
  var square = (int x){ return x * x;};

  // Call the anonymous function
  var result = square(5); // result will be 25
  print(result);
}
```

- Example 2

```
void main() {
List<int> numbers = [1, 2, 3, 4, 5];

numbers.forEach(
// passing a anonymous fxn
  (number){
          print(number);
        }
);
}
```

One Liner Fxn Syntax

- You can also write simple fxns like this

```
// Syntax: returnType functionName(parameters) => expression;

int add(int a, int b) => a + b;
```

11. Classes and Objects

- **Class:** A blueprint for creating objects. Defines properties (data) and methods (actions).
  class Car { ... }`
- **Object:** An instance of a class. Represents a specific entity with its own data.
  `Car myCar = Car();`
- **Properties:** Variables within a class that store data.
  `String? model;`
  `int? year;`
- **Methods:** Functions within a class that define the object's behavior.
  `void drive() { ... }`
- Classes are like a box that contains fxns and variables and objects are like copies of that box but with specific values in that variables
- Example:

```
// A class
class Car {
//properties or fields
  String? model;
  int? year;
//method or a class fxn
```

```
  void drive() {
    print('The $model is driving.');
  }
}

void main() {
  Car myCar = Car(); // object creation
  myCar.model = 'Toyota Camry';  // accessing propertirs
  myCar.year = 2023;
  myCar.drive(); // Output: The Toyota Camry is driving.
}
```

- Good Resource : https://docs.oracle.com/javase/tutorial/java/concepts/index.html

12. Getters and Setters

- Used to access and modify private fields of a class.
- Example:

```
class Person {
        late String _name;

        String get getname => _name;  // getter used to get name
        set setname(String value) => _name = value; // setter used to set
name
}

void main() {
        var person = Person();
        person.setname = 'Alice';
  // cant access person._name like this since its a private property
        print(person.getname); // Alice
}
```

13. Inheritance and `super`

Inheritance

- **Concept:** Inheritance allows a class (called a **subclass** or **child class**) to inherit properties and methods from another class (called a **superclass** or **parent class**).
- That is we can use properties and methods of a class in an another class.
- Syntax: `class Subclass extends Superclass { ... }`

- Example:

```
class Animal {
  void eat() {
    print('Animal is eating.');
  }
}

class Dog extends Animal {
  void bark() {
    print('Dog is barking.');
  }
}
```

super keyword

- Used within a subclass to access members (properties and methods) of the superclass.
- Example:

```
class Animal {
  String? color;

  void displayColor() {
    print('Animal color: $color');
  }
}

class Cat extends Animal {
  void displayInfo() {
    super.color = 'White'; // Accessing superclass's property
    super.displayColor(); // Calling superclass's method
  }
}
```

`@override` Annotation

- Indicates that a method in a subclass overrides a method with the same name in the superclass.
- Example:

```
class Animal {
  void makeSound() {
    print('Animal makes a sound.');
  }
}
```

```dart
class Dog extends Animal {
  @override
  void makeSound() {
    print('Dog barks.');
  }
}
```

IMPORTANT LINKS

- Learn Dart :https://dart.dev/language
- Dart documentation : https://api.dart.dev/
- Online dart code editor : https://dartpad.dev/
- **SETUP ENVIRONMENT (OFFICIAL DOCS)**: https://docs.flutter.dev/get-started/install
- **SETUP ENVIRONMENT VIDEO (MAC)**: https://www.youtube.com/watch?v=QG9bw4rWqrg
- **SETUP ENVIRONMENT VIDEO (WINDOWS)**: https://www.youtube.com/watch?v=6AfMhjexLDg
- **SETUP ENVIRONMENT VIDEO (LINUX)**:https://www.youtube.com/watch?v=mtqTnGAAHw0