Flutter cheat Sheet

Basics of Dart

Dart is a general-purpose programming language developed by Google. It's designed to be easy to learn, fast, and scalable, with features that make it suitable for building a wide range of applications, including web, mobile, desktop, and server-side applications.

Here's an introduction to some key aspects of Dart:

1. **Object-Oriented:** Dart is an object-oriented language, which means it supports concepts like classes, objects, inheritance, encapsulation, and polymorphism.
2. **Strongly Typed:** Dart is statically typed, meaning variable types are known at compile time. However, Dart also supports type inference, allowing you to omit type annotations in many cases where the type can be inferred from the context.
3. **Asynchronous Programming:** Dart provides built-in support for asynchronous programming using futures and async/await syntax. This makes it easy to write non-blocking code for tasks like I/O operations and network requests.
4. **Mixins and Extensions:** Dart supports mixins, which are a way to reuse code across multiple classes without inheritance. It also supports extensions, which allow you to add new methods to existing classes without modifying their source code.
5. **Platform Independence:** Dart is designed to be platform-independent, meaning you can use it to build applications that run on various platforms, including web browsers, mobile devices (with Flutter), desktops, and servers.

Flutter cheat Sheet

Basics of Dart

- ### Dart Syntax :-

Dart syntax is similar to languages like C, Java, and JavaScript, making it relatively easy to learn for developers familiar with those languages. Here are some key syntax elements:

- **Variables**: Variables in Dart can be declared using the **var**, **dynamic**, or specific type keywords like **int**, **double**, **String**, etc.
- **Functions**: Dart supports functions as first-class citizens, meaning functions can be assigned to variables, passed as arguments, and returned from other functions.
- **Control Flow**: Dart provides control flow statements like **if**, **else**, **switch**, **for**, **while**, and **do-while** for implementing conditional and looping logic.
- **Classes and Objects**: Dart is an object-oriented language, so classes and objects play a central role. Classes encapsulate data and behavior, while objects are instances of classes.
- **Libraries and Packages**: Dart code is organized into libraries, which are collections of related code. Dart packages are a way to distribute and reuse libraries. The Dart package manager, **pub**, is used for managing dependencies and publishing packages.

- ### Dart Usage :-

- Web Development: Dart can be used for web development, both on the client-side and server-side. On the client-side, Dart can be compiled to JavaScript using the Dart SDK's dart2js compiler. On the server-side, Dart can be used to build web servers using frameworks like Aqueduct and Angel.
- Mobile Development: Dart is the primary programming language for building mobile apps with Flutter, Google's UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase.
- Desktop Development: Dart can also be used for desktop application development, with projects like Flutter Desktop and dart:ui providing support for building cross-platform desktop applications.

Flutter cheat Sheet

Basics of Dart

- **Dart Variables and Data Types :-**

In Dart, you can declare variables using var, dynamic, or specific data type keywords such as int, double, String, bool, etc.

```
var name = 'John'; // inferred as String
dynamic age = 30; // dynamic type
int count = 5; // integer
double price = 10.5; // double
bool isStudent = true; // boolean
```

- **Dart Input and Output :-**

To take input from the user, you can use the stdin object from the dart:io library. For output, you can use the print() function.

```
import 'dart:io';
void main() {
  stdout.write('Enter your name: ');
  String name = stdin.readLineSync()!;
  print('Hello, $name!');   }
```

- **Dart Nullable Types: :-**

Dart 2.12 introduced nullable types using ? syntax. By default, all variables are non-nullable. You can make a variable nullable by appending ?.

```
String? nullableName;
int? nullableAge = null;
```

Basics of Dart

- ### Dart Functions :-

Functions in Dart are declared using the void keyword for functions that don't return a value, or a data type for functions that return a value.

```
void greet() {
print('Hello, World!');
}
 int add(int a, int b) {
return a + b;            }
```

- ### Dart classes, Objects, and Constructors :-

You can define classes in Dart using the class keyword. Constructors are defined using the same name as the class.

```
class Person {
 String name;
 int age;
// Constructor
 Person(this.name, this.age);

 // Method
 void introduce() {
  print('My name is $name and I am $age years old.');
 }
}
void main() {
 var person = Person('John', 30);
 person.introduce();
}
```

Basics of Dart

- ### Dart Inheritance :-

Dart supports single inheritance. You can extend a class using the **extends** keyword

```
class Student extends Person {
  String school;

  Student(String name, int age, this.school) : super(name, age);

  void study() {
    print('$name is studying at $school.');
  }
}
```

- ### Dart Abstraction and Mixin Classes :-

You can create abstract classes using the abstract keyword. Mixin classes are used to add functionality to a class without using inheritance.

```
abstract class Animal {  void makeSound();  }

mixin CanFly {  void fly() {  print('Flying...');   }  }

class Bird extends Animal with CanFly {  @override
  void makeSound() {
    print('Chirp chirp');  }  }
```

Flutter cheat Sheet

Basics of Dart

- **Dart Keywords :-**

Dart has various keywords such as **if**, **else**, **for**, **while**, **switch**, **case**, **break**, **continue**, **return**, **class**, **extends**, **implements**, **with**, **abstract**, **static**, **final**, **const**, etc.

- **Dart Higher-Order Functions :-**

Dart supports higher-order functions, which are functions that can take other functions as parameters or return functions.

```
void myFunction(int a, int b, Function operation) {
print(operation(a, b));
}

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

void main() {
myFunction(5, 3, add); // prints 8
}
```