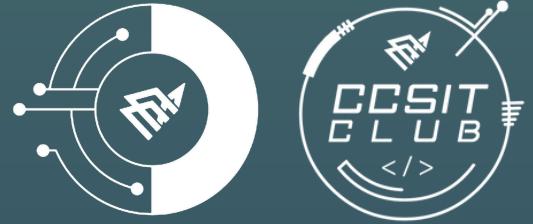


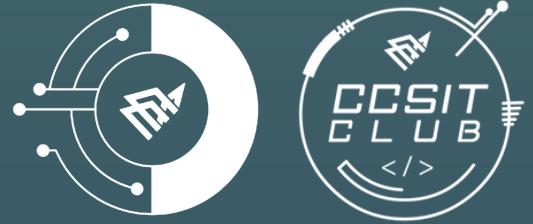
★ 1st Week ★

Dart Fundamentals



We will cover :

- App development
- What is Dart?
- Syntax Basics
- Control Flow
- Functions
- OOP



App development

:Cross-platform Apps .1

محورة باستخدام إطار عمل (Framework) واحد لبناء التطبيق لكلا النظمتين من قاعدة كود واحدة.
أشهر إطار العمل (Frameworks) حالياً هي:

(Dart لغة Flutter •

.(JavaScript لغة React Native •

: Native Apps .1

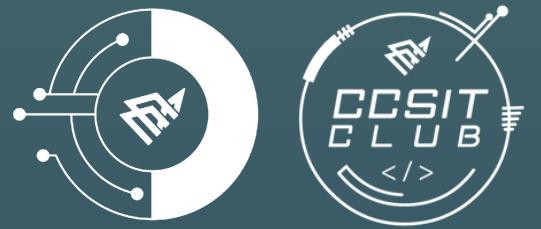
محورة خصيصا لنظام تشغيل معين مثل iOS أو Android وتعمل فيه **فقط**.
يتم استخدام اللغات الرسمية لكل نظام تشغيل:

iOS → Xcode + Swift or Java •

Android → Android Studio + Kotlin or objective-C •

- ✓ تقليل التكاليف والوقت، سهولة الصيانة.
- ✗ في بعض الحالات قد يكون الأداء أقل قليلاً من ال-native apps

- ✓ أداء عالي، تكامل تام مع النظام.
- ✗ تكلفة أعلى، زمن تطوير أطول.



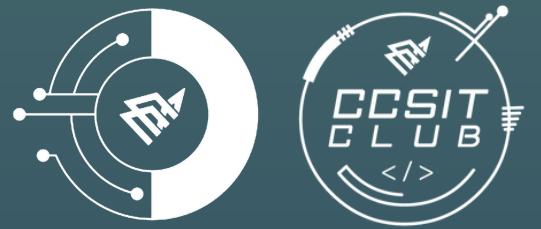
What is Dart?

لغة برمجة كائنة (General-Purpose) عامة الغرض (Object-Oriented) مفتوحة المصدر طورتها Google عام 2011، بنيتها بسيطة وأدائها عالي.

تستخدم في عدة مجالات مثل :

- تطوير تطبيقات الهاتف (iOS و Android) باستخدام Flutter.
- تطبيقات سطح المكتب (Windows, macOS, Linux).
- تطبيقات الويب.
- تطوير الخوادم (Backend).

لكن انتشارها الأكبر جاء من استخدامها مع Flutter لتطوير تطبيقات الهواتف الذكية، وهو إطار عمل مفتوح المصدر أطلقته Google عام 2017، ويُستخدم لتصميم وبرمجة Native Apps بآداء ممتاز مشابه للـ Native Apps.



Syntax Basics

1. Variables :

- String, int, double, bool
- List
- Map

2. Constants :

- final (runtime)
- const (compile time)

4. Operators:

- Arithmetic: + - * / %
- Logical: && || !
- Comparison: == != > < >= <=
- Assignment: = += -= *=

5. Input & Output:

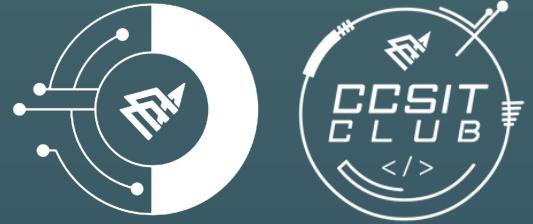
- Output: print('Hello');
- Input:
`var name = stdin.readLineSync();
(import dart:io)`

3. Comments :

- // Single line
- /* Multi-line */

6. String Interpolation :

- `print('Hello $name');`
- `print('Sum = ${a + b}');`



Control Flow

1. Conditional Statements:

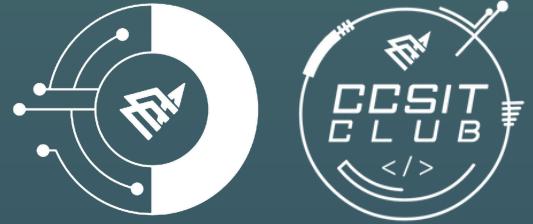
- if, else if, else
- switch / case / default

2. Loops:

- for (classic loop)
- while (repeat while condition is true)
- do-while (executes at least once)

3. Loop Control:

- continue (skip to next iteration)
- break (exit the loop)



Functions

1. Functions with Return Value:

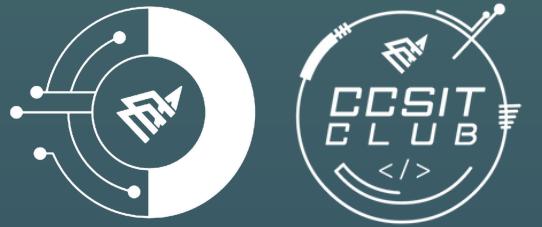
- Must specify return type and use return keyword
- **Example:** int add(int a, int b) {
 return a + b;
}

2. Void Functions:

- No return value
- **Example:** void greet() {
 print('Hello');
}

3. Arrow Functions:

- Short form for single expression
- **Example:**
int add(int a, int b) => a + b;

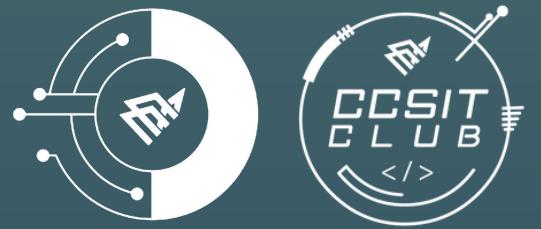


OOP

Object-Oriented Programming يرتب الكود باستخدام objects و classes

- يجعل الكود:
 - أكثر ترتيب
 - قابل لإعاده الاستخدام
 - أسهل للتعديل والتوصيف

- الأربع مباديء أساسية:
 - Encapsulation
 - Abstraction
 - Inheritance
 - Polymorphism



OOP

1. Classes & Objects:

- **Class:** a template for creating objects.
- **Object:** an instance of a class
- **Example:**

```
class Person {  
    String name;  
    int age;
```

```
void greet() {  
    print('Hello, my name is $name');  
}  
}
```

```
void main() {  
    var p = Person();  
    p.name = 'Ali';  
    p.age = 20;  
    p.greet();  
}
```

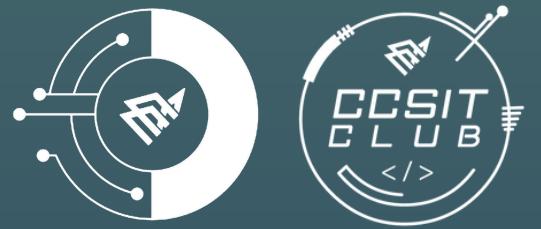
2. Constructors:

- Used to initialize objects when created.
- Default constructor is provided automatically if not defined.
- **Example:**

```
class Person {  
    String name;  
    int age;
```

```
Person(this.name, this.age); // Constructor
```

```
void info() {  
    print('$name is $age years old');  
}  
}
```



OOP

3. Encapsulation (Getters & Setters):

- Encapsulation is hiding internal details and controlling access.
- Use private variables (_variable) and public getters/setters.
- **Example:**

```
class BankAccount {  
    double _balance = 0;  
  
    double get balance => _balance;  
  
    void deposit(double amount) {  
        _balance += amount;  
    }  
}
```

* _balance can't be accessed directly outside the class*

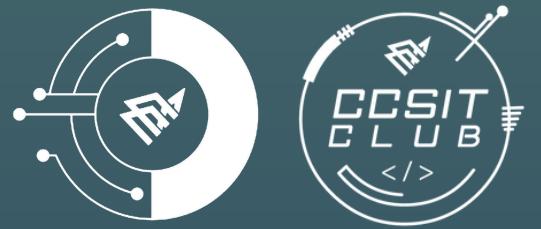
4. Inheritance:

- Allows one class to inherit properties and methods from another.
- Use (extends) keyword.
- **Example:**

```
class Animal {  
    void sound() => print('Animal makes sound');  
}  
class Dog extends Animal {  
    void bark() => print('Woof!');  
}
```

```
void main() {  
    var dog = Dog();  
    dog.sound();  
    dog.bark();  
}
```

Dog inherits from Animal



OOP

5. Polymorphism & Abstraction:

- **Polymorphism:** Using a parent reference to call methods of different child classes
- **Abstraction:** Hiding complex details using abstract classes or interfaces, use (abstract) keyword
- **Example:**

```
abstract class Shape {  
    void draw();  
}  
class Circle extends Shape {  
    void draw() => print('Drawing Circle');  
}  
class Square extends Shape {  
    void draw() => print('Drawing Square');  
}
```

Abstract classes can't be instantiated directly

```
void main() {  
    Shape shape = Circle();  
    shape.draw(); // Drawing Circle  
}
```