Chapter 4:

Fundamental bases of

Dart programming language

Plan

1. **Introduction to Dart programming language.**
   - Dartpad.
2. **Dart basics.**
   - Data types, Variables, Operators, Conditions, Loops.
   - Functions, Anonymous functions, Callback functions.
3. **Object-oriented concepts in Dart.**
   - Class, attributes, constructor.
   - Methods, getters & setters.
   - Inheritance (extends, implements).
4. **Asynchronous programming in Dart.**

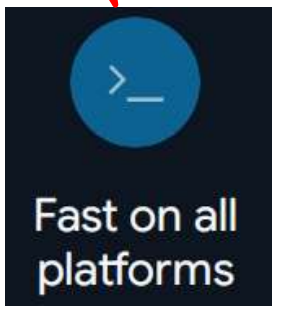
Plan

1. **Introduction to Dart programming language.**
   - Dartpad.
2. **Dart basics.**
   - Data types, Variables, Operators, Conditions, Loops.
   - Functions, Anonymous functions, Callback functions.
3. **Object-oriented concepts in Dart.**
   - Class, attributes, constructor.
   - Methods, getters & setters.
   - Inheritance (extends, implements).
4. **Asynchronous programming in Dart.**

Introduction

- A client-optimized language for fast apps on any platform.
- An Object-Oriented language.
- Free & open source.
- Developed by Google.
- First version was released on Novembe 4th, 2013.
- Features of Dart are :

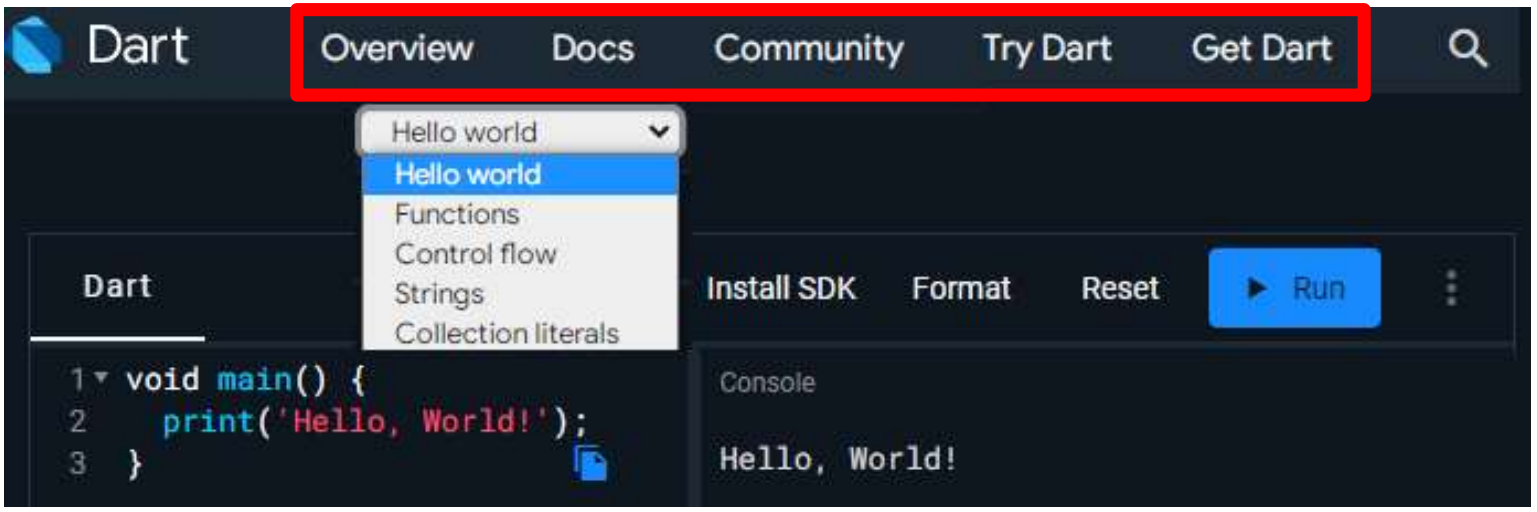
Compile to ARM & x64 machine code for mobile, desktop, and backend.
Compile to JavaScript for the web.



Hot reload to see changes instantly on the running app

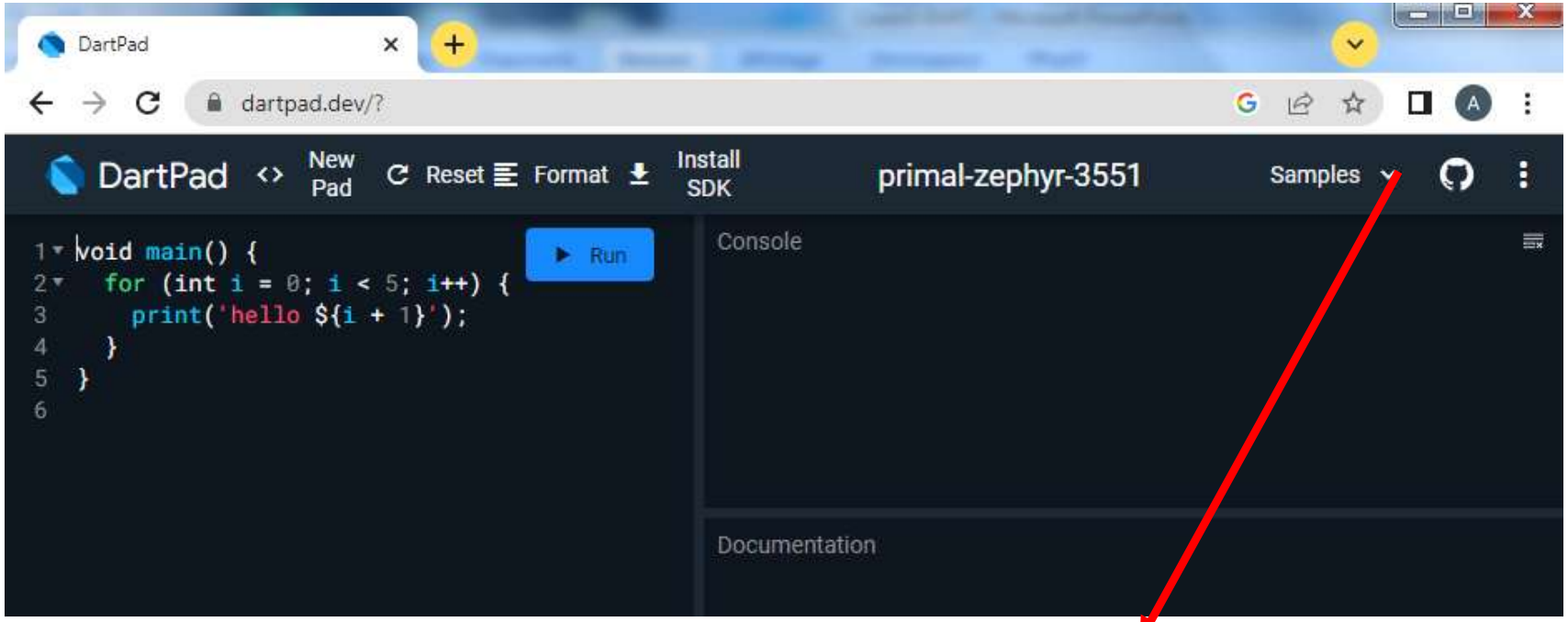
Dart website

- **Dart.dev**.
  - Providing documentation.
  - Taking access to the community.
  - Alowing to try Dart in the Browser.

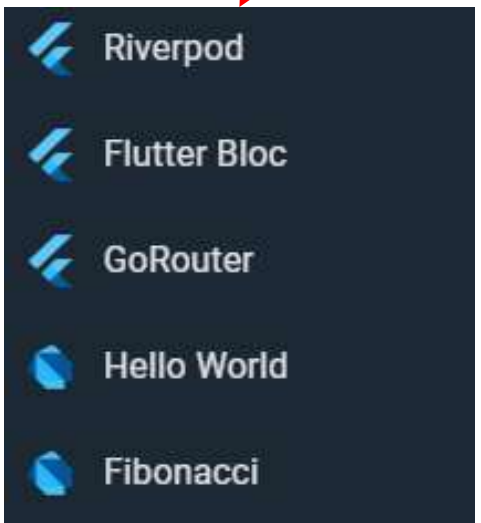


The dartpad

- Website: **Dart.pad.dev**

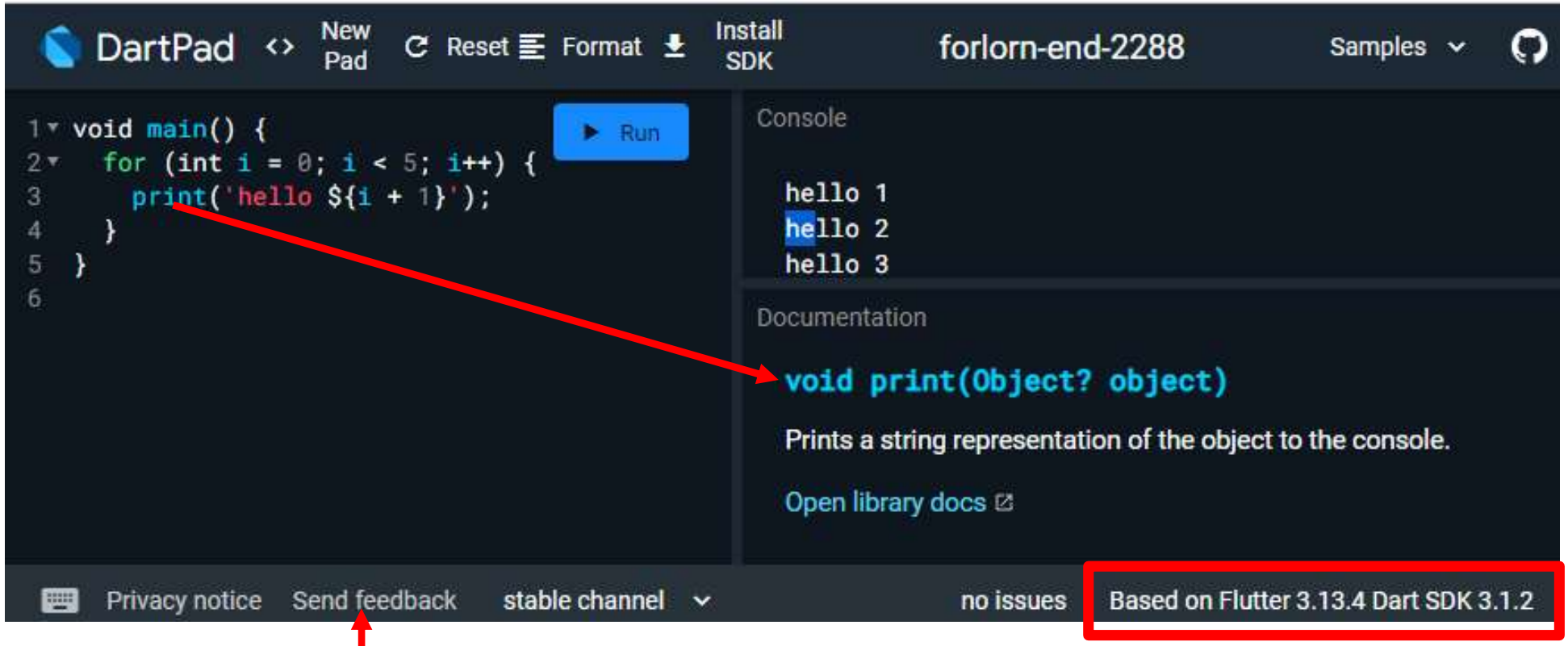


- Dart and Flutter samples:



The dartpad

- Documentation:



- Sending feedbacks:

Plan

1. **Introduction to Dart programming language.**
   - Dartpad.
2. **Dart basics.**
   - Data types, Variables, Operators, Conditions, Loops.
   - Functions, Anonymous functions, Callback functions.
3. **Object-oriented concepts in Dart.**
   - Class, attributes, constructor.
   - Methods, getters & setters.
   - Inheritance (extends, implements).
4. **Asynchronous programming in Dart.**

Basics of Dart language

- **Data types :**
  - **Numbers:** num, int, double. // num is a super type of int and double
  - **Strings:** 'hi', "hi".
  - **Booleans:** true, false.
  - **Lists (arrays):** [0,1,2].
  - **Sets:** {'A','B','C'}.
  - **Enums:** Color.red, Color.green , Color.blue.
  - **Maps :** {'key': value}.

Map<String, String> capitals = { 'France': 'Paris', 'Germany': 'Berlin', 'Japan': 'Tokyo', };

Basics of Dart language

- **Data types :**

  - **List :**

    - // Declaration of an empty and growable list:
      List <int> myList = **List.empty**(growable:true);
    - // Declaration of a fixed-size list initialized with zeros:
      List <int> myList = **List.filled**(size,0);
    - // Declaration of a list with predefined values:
      List <int> myList = [4,10,3];
    - // Assigning a value to a specific index:
      myList [index] = value;
    - // Adding an element to the list:
      myList.add (10);
    - // Removing an element by value:
      myList.remove (10);
    - // Removing an element by index:
      myList.removeAt (10);

Basics of Dart language

- **Data types :**
  - **Map :**
    - // Declaration of an empty map:
      Map<String, int> myMap = Map();
    - // Declaration of a map with predefined key–value pairs:
      Map<String, int> myMap = { 'key1': 10, 'key2': 20, 'key3': 30 };
    - // Adding or updating a key–value pair:
      myMap['key4'] = 40;
    - // Adding multiple key–value pairs at once:
      myMap.addAll({ 'key5': 50, 'key6': 60 });
    - // Removing a key–value pair by key:
      myMap.remove('key1');
    - // Removing all elements from the map:
      myMap.clear();

Basics of Dart language

- **Data types :**
  - **Enumeration :**
    - // Declaration of an enumeration:
      enum myEnumStatut { OK, KO }
    - // Access all values:
      myEnumStatut.values.
    - // Get the index of a specific value:
      myEnumStatut.OK.index.
    - // Access a value by its index:
      myEnumStatut.values[0].

Basics of Dart language

- **Data types :**
  - **List vs. Set vs. Enumeration :**

	List	Set	Enum
Definition	An ordered collection of elements.	An unordered collection of ***unique*** elements.	A fixed set of named constant values.
Duplicates allowed	Allowed	Not allowed	Not allowed
Access by index	Yes (list[0])	No	Yes (MyEnum.values[0])
Order	Yes (maintains insertion order)	No order	Yes

Basics of Dart language

- **Variables :** declaring with an explicit type.

DataType VariableName = InitialValue;

- **Examples of declaration:**

  - **double** nb = 10.0;
  - **int** nb = 10 ;
  - **num** nb = 10 ;
  - **string** name = 'Myname'+'is';
  - **bool** cond = true;

Basics of Dart language

- **Variables :** declaring with type inference.

  - **var** name = 'Myname'; // type is specified when assigning data to variable
    => name is a String.
    => name = 10.0; => Error.

  - **dynamic** x= 'Myname';

    x = 10.0; // No error because dynamic variable may have any type of value during program's running.

Basics of Dart language

- **Variables :** using "**final**" & "**const**" keywords.
  - **final** name = 'Myname'; // Its value can't be
  - **const** name = 'Myname'; // changed in the entire
    // code.

    name = 'First'; => Error : constant can't be assigned a value.

const is a constant at a compile time, while **final** is a constant at a run time.

```
void main() {
  const String data = "Flutter";
  print(data);
}
```

```
void main() {
  final DateTime date = DateTime.now();
  print(date);
}
```

Basics of Dart language

- **Variables :** using "**late**" keyword.

  - **late** int X; // Declaring a variable that will be initialized at a later time.

    print($X); // Error.

  - X = computeSum (10,5); // Later initialization.

    print($X); // OK.

late is used to delay the initialization of a variable (after its declaration).

Basics of Dart language

- **Variables :** Null Safety feature (**Nullable / non Nullable**).

  - **String** name;//Declaring a **non nullable** variable.
    print("Hi $name");=>Error non-nullable variable

  - **String ?** name ; // Declaring a **nullable** variable
    // Initialized to null by default.
    print ("Hi $name"); => Hi null

  - **String ?** name = 'My Name '; // Initialized to non-null.
    name = null; // Can be re-assigned to null.

Basics of Dart language

- **Operators:**

  - = : **assignment**.
  - +,- ,*,/,%,~/ : **arithmetic**.
  - = =, >, <, >= ,<= ,!= : **comparison**.
  - &&, | |, ! : **logical**.
  - X>2 **?** 'V ' **:** 'F ': **ternary conditional expression**.
  - Y=X **??** 5 : **non null conditional expression**.

    // if X is non null then Y = X else Y = 5.
  - X **is** int, X **is!** bool: **type test operators**.

    //checking data type of X.
  - X **as** int: **cast operator.** // cast X to int

Basics of Dart language

- **Operators: Null assertion operator.**

  - The operator (**!**) is used to cast a potentially **nullable** variable into a **non-nullable** one.

```
int ? someValue; // Nullable variable
someValue = 30; // Assign a non null value
int data = someValue !; // This is valid as someValue is non-nullable
```

  - Telling Dart that the variable "**someValue**" is **nullable**, but it is safe to assign it to the **non-nullable** variable **"data"** as "**someValue**" will not be null at **runtime**.

Basics of Dart language

- **Loops :**
  - **for** (int i = 0;i<5;i++) {…};
  - **for** (var e **in** myList) {…};
  - **for** (var e **in** myMap.values{…};//or myMap.keys
  - myList.**forEach** ((e){…});
  - myMap.**forEach** ((key, val){…});
  - myEnum.values.**forEach** ((e){…});
  - **while** (myCond) **do** {…} ;
  - **do** {…} **while** (myCond);

Basics of Dart language

- **Conditional structures:**

  - **if** (myCond) {…};
  - **if** (myCond) {…} **else** {…};
  - **switch** (myVar) { **case** "Val1": … ; **break**;
    case "Val2": … ; **break**;
    default : … ; **break**;
    }

Basics of Dart language

- **Comments :**
  - **// : single-line**
  - **/* */ : multi-lines**
- **Print :**
  - **print** ("Hi"); or **print** ('Hi');
  - **print** ("Hi $name ");
  - **print** ("Hi $ {X+1} ");

Basics of Dart language

- **Functions :** declarations.

/* Standard syntax */

- **returnType** myFct (**paramType** myPar1,..){
  … // Function body
  return result }//declaration

/* Arrow syntax (for single-expression function */

- **returnType** myFct (**paramType** myPar1,..) =>
  expression; // declaration
- **int** myFct (int myPar) => myPar*2;

Basics of Dart language

- **Functions :** declarations.

/* No value returned */

➢ **void** myFct (**paramType** myPar1,..){
… // Function body }//declaration

/* Nullable return type */

➢**returnType ?** myFct (**paramType** myPar1,..){
… // Function body
return null }//declaration

/* Omit types */

➢ myFct (myPar1, myPar2){
… // Function body
return result; };

Basics of Dart language

- **Functions :** parameter features.

/*Positional parameters */

- The call must respect the declaration order of parameters.
- myFct (**paramType** myPar1, **paramType** valMyPar2){…}; //declaration
- myFct (valMyPar1, valMyPar2); //Call (use)

/* Optional parameters */

- myFct (**paramType** myPar1,[**paramType** myPar2]){…} //declaration
- myFct (valMyPar1); //Call (use) => myPar2 = null.

Basics of Dart language

- **Functions :** parameter features.

/* Named parameters */

- myFct ({**paramType** myPar1, **paramType** myPar2}) {…};
  //declaration
- myFct (**myPar2**: valMyPar2 , **myPar1:** valMyPar1);
  //Call (use); //Position is not important

Named parameters are **optional** by default, but under null safety, each variable must either have **a value** or be explicitly marked as **nullable**. Three options:

1. Define as nullable (?).
2. Use default value.
3. Mark as Required.

Basics of Dart language

- **Functions :** parameter features.

/* Named parameters: define as nullable (1) */

- myFct ({paramType ? myPar1,paramType ? myPar2}) {…}; //declaration
- myFct ( ); //Call (use) => myPar1= null, myPar2=null.
- myFct (**myPar1**: valMyPar1); // => myPar2 = null.

/* Named parameters: use default values (2) */

- myFct ({paramType myPar1= valMyPar1, paramType myPar2 = valMyPar2}) {…}; //declaration
- myFct (**myPar2**: val); //Call (use)
  // => myPar1 = valMyPar1.

Basics of Dart language

- **Functions :** parameter features.

/* Named parameters: mark as required (3) */

➢ myFct ({**required** paramType myPar1,
required paramType myPar2}) {…}; //declaration

➢ myFct (**myPar2**: valMyPar2, **myPar1**: valMyPar1); //Call (use).
// => myPar1= valMyPar1, myPar2= valMyPar2.

➢myFct (**myPar1** : valMyPar1); // Error: Missing required argument myPar2.

Basics of Dart language

- **Anonymous functions :** are a nameless functions, defined without specifying a name or a return type.

```
/* Standard syntax */
(paramType1 param1, paramType2 param2, …) {
                        // Function body
                            return result; // optional
                        }
/* Arrow syntax */
(paramType1 param1, paramType2 param2, …) => expression;
```

Example :

```
var sum = (int a, int b) => a + b;
var sum = (int a, int b) {     var R = a + b;
                                return R;
                            };
```

Basics of Dart language

- **Callback function :** is a function passed into another function as an argument.

```
/* Without arguments */
➢ myFct ({Function myCallback}) {myCallback ( ); } ;
                                                    // declaration
➢ myFct (myCallback : ( ) {…}) ; //Invoke.

/* With arguments (a string) */
➢ myFct ({Function myCallback}) {myCallback ("Hello"); } ;
                                                    // declaration
➢ myFct (myCallback : (String text) {… }) ;//Invoke.
```

Callback functions are used to pass data from one function to another.

Basics of Dart language

- **Callback function : example.**
  - An ordinary function:

```
void myCallback (String? name){
        if(name! = ""){
                print('Nice To meet you ${name}');
        }else { print('I don\'t know your name');}
}
```

  - Create a function that accept a function as argument:

```
answerQuestion (String answer , Function callback) {
        callback (answer);
}
```

  - Invoke the function using the callback function:

```
var question ="what is your name?"; print ("$question");
answerQuestion("myName" ,myCallback);
```

Basics of Dart language

- **Callback function : example of anonymous callback function.**

```
int add(int x, int y) {
  return x + y;
}
```


```
(int x, int y) {
  return x + y;
}
```

Anonymous function

- Create a function that accept a function as argument:

```
answerQuestion (Function callback) {
        callback ("myName" );
}
```

- Invoke the function using anonymous callback function:

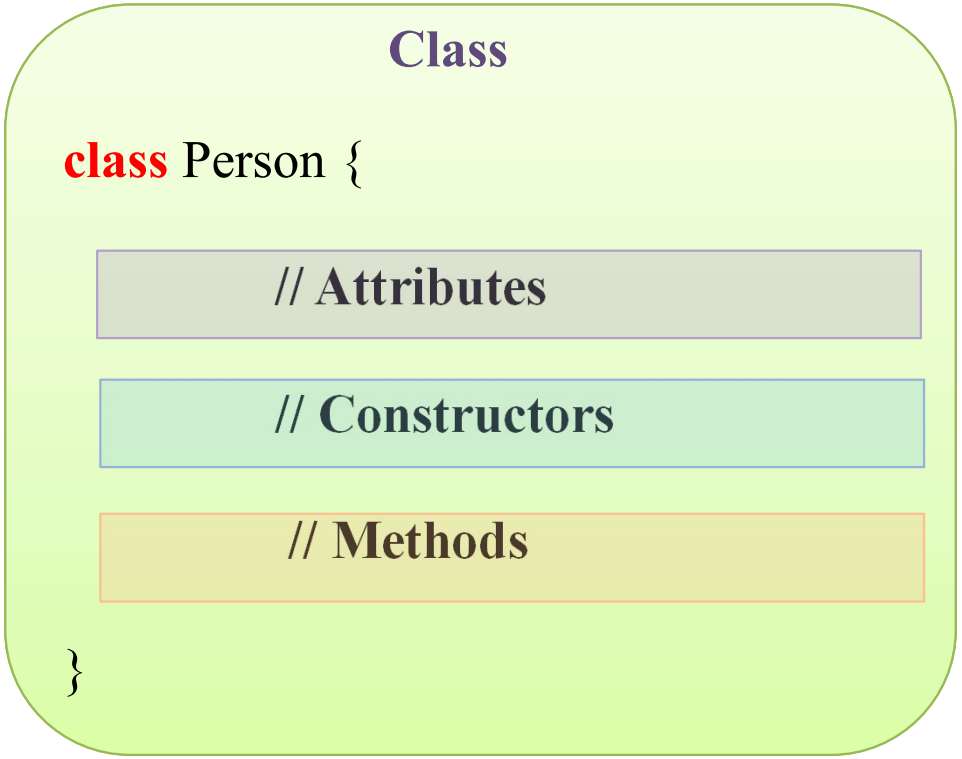
```
var question ="what is your name?"; print ("$question");
answerQuestion((String name) {
if(name != ""){
        print('Nice To meet you ${name}');
}else {
        print('I don\'t know your name');}
});
```

Plan

1. **Introduction to Dart programming language.**
   - Dartpad.
2. **Dart basics.**
   - Data types, Variables, Operators, Conditions, Loops.
   - Functions, Anonymous functions, Callback functions.
3. **Object-oriented concepts in Dart.**
   - Class, attributes, constructor.
   - Methods, getters & setters.
   - Inheritance (extends, implements).
4. **Asynchronous programming in Dart.**

POO basics of Dart

- **Class:** example "Person".



POO basics of Dart

- **Attribute:**

```
String name = "" ;  // required
int age = 0 ; // required
double ? size;  // optional
```

- **Constructor:**

```
// First way (long) with positional parameters
Person (String name, int age, double size) {
  this.name=name;
  this.age=age;
  this.size=size;
 };
// Second way (faster)
 Person (this.name ,this.age,this.size);
```

Creating an instance:

```
void main( ) {
  Person me = Person("Michel",30 );
}
```

POO basics of Dart

- **Constructor:**

```
// Third way with named parameters:

Person({this.name="",this.age=0,this.size});
Person({ required this.name, required this.age,this.size});
```

Specifying default (initial) values for non nullable attributes.

Using "Required" keyword for non nullable attributes.

Creating an instance:

```
void main( ) {
        Person me= Person(name:"Mickal",age:30);
        Person you= Person(size:170, name:"Mickal",age:30);
}
```

Optional attribute may not be specified and attribute order is not important.

POO basics of Dart

- **Methods:**

Attributes

```
String name = "" ;  // required
int age = 0 ; // required
double ? size;  // optional
bool isChild ; // required
```

Constructor

```
Person({ required this.name, required this.age,this.size, this.isChild = false});
```

Methods

```
void childChecking ( ) {
  if (age < 12){ this.isChild = true;} }

String description ( ) {
   return " i am ${this.name}";}

void updateSize (double newSize ) {
   this.size = newSize ;}
```

Methods calls:

```
void main( ) {
        Person me= Person(name:"Mickal",age:30 );
        me. childChecking ( );
        me.description ( );
        me.updateSize( 195.5);
}
```

POO basics of Dart

- **Getter & Setter methods for private attribute:**

Attributes

```
String name = "" ;  // required
int age = 0 ; // required
double ? size;  // optional
bool isChild ; // required
String _adress = "";  // private attribute
```

Getter & Setter

```
String get getAdress => _adress;
set setAdress (String newAdress){
      _adress = newAdress ;
}
```

Constructor

```
Person({this.name="",this.age= 0,this.size});
```

OR

```
Person ({String name="", int age=0,double? size, String adress= ""}): _adress=adress;
```

```
void main( ) {
      Person me= Person(name:"Mickal",age:30 );
// Call the setter that assigns "TLEMCEN" to adress.
      me. setAdress ="TLEMCEN" ;
 // Call the getter that returns _adress value.
      String myAdress = me.getAdress;
}
```

POO basics of Dart

- **Inheritance : extends.**

```
class Employee extends Person {

// Attribute
String profession =" ";
// Constructor
Employee (empName) : super  (name: empName);
// Method
showProfession(){
            Print("$name is a $profession");}
}
```

```
void main( ) {
        Employee firstEmployee = Employee  ("OMAR");
                firstEmployee.profession= "Teacher "; // Use child class attribute
                firstEmployee.showProfession(); // Call child class method
                firstEmployee.setAdress="ORAN" ; // Call parent class method
 }
```

POO basics of Dart

- **Inheritance : extends.**

```
class Person { canTeach ( ) { print ( "What can $name teach ? "); } }

class Employee extends Person {
        // Method
        @override
        canTeach ( ) {
                        super. canTeach ( ) ;
                        print(" $name can teach mobile apps");
                }
}
```

```
void main( ) {
        Employee firstEmployee = Employee  ("OMAR");
        firstEmployee.canTeach();
 }
```

Output:

```
What can OMAR teach ?
 OMAR can teach mobile apps
```

POO basics of Dart

- **Inheritance : implements.**

```
class Person { void canTeach ( ) ;} }

class Employee implements Person {
        @override
        canTeach ( ) {
                        print(" $name can teach mobile apps");
                      }
```

```
void main( ) {
        Employee firstEmployee = Employee  ("OMAR");
        firstEmployee.canTeach();
 }
```

Output:

```
OMAR can teach mobile apps
```

Plan

1. **Introduction to Dart programming language.**
   - Dartpad.
2. **Dart basics.**
   - Data types, Variables, Operators, Conditions, Loops.
   - Functions, Anonymous functions, Callback functions.
3. **Object-oriented concepts in Dart.**
   - Class, attributes, constructor.
   - Methods, getters & setters.
   - Inheritance (extends, implements).
4. **Asynchronous programming in Dart.**

Asynchronous programming in Dart

- An **asynchronous** operation lets a program continue executing while waiting for a long-running task (e.g., HTTP request, file I/O).

- When an application calls an asynchronous method, the **main thread is not blocked**.

- Dart is **single-threaded**, but using **"await"** allows an asynchronous function to pause execution until the task completes, without blocking the program.

Asynchronous programming in Dart

- Asynchronous involves the use of **future**, **async** and **await** keywords.

  future: getting a value sometime in the future.

  async: allows performing an operation asynchronously.

  await: allows to delay the execution of an async function.

- **Example:**

```
Future <void> testFunction() async {
      await Future.delayed(Duration(seconds:2),( ) {
                                    print("value 1");
                                    print("value 2");
                                    });}
main() {
      testFunction();
      print("main print statement");}
```

Output:

```
main print statement
value 1
value 2
```