Flutter Una Introducción al SDK de Google

Dart

Hola Mundo

Para imprimir se usa print

```
void main(){

  var nombre = "Adrian";


  print("Hello $nombre");
}
```

Hola Mundo
⟸ Interpolacion de Strings, es decir inyeccion de algun valor
* Se debe inicializar cada variable ya que en caso de no estar inicializada pues Dart le asigna el tipo de dato dinamico (en este caso se asigno el tipo debido al dato

```
void main(){

  String nombre = "Adry";

  print("Hello $nombre");
}
```

Manera correcta

Tipos de datos - Números y Strings

```
void main() {
  // Números

  int empleados = 10;
  double pi = 3.141592;
  var numero = 1.0; // Para los double se ponen si o si decimales caso contrario
  // Dart lo interpreta como un numero double

  print('$empleados - $pi - $numero');

  // String - Cadenas de caracteres
  String nombre = "Elliot";
  print('$nombre');
  print(nombre[0]); // En Dart no se maneja mucho el termino de arreglo sino de Lista
  print(nombre[nombre.length-1]);//Ultimo caracter
}
```

Run

Console

```
10 - 3.141592 - 1
Elliot
E
t
```

Tipos de datos - Booleanos y condiciones

```dart
void main() {
 bool activado =true;
 print(activado);

  if ( !activado ){
    print("El motor esta Funcionando");
  } else {
    print("Está apagado");
  }
}
```

```
Console

true
Está apagado
```

Tipos de datos - Lista

```dart
void main() {

  //List numeros = [1,2,3,4,5]; // <= Lista dinamica [tiene cualquier tipo]
  List<int> numeros = [1,2,3,4,5]; // <= Lista dinamica [tiene cualquier tipo]
  print( numeros );
  //Añadir nuevos elementos
  numeros.add(6);
  // Al ser una lista dinamica se pueden añadir datos de diferente tipo, por ej:
  //numeros.add("Hello Friend");
  print( numeros );

  // Tamaño Fijo
 List masNumeros = List.filled(10,null);
 print( masNumeros );
 // masNumeros.add(1); //Esto no es correcto porque es de un largo fijo
 //Sin embargo, sí se puede modificar los valores
   masNumeros[0] = 1;
   print(masNumeros);
}
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
[null, null, null, null, null, null, null, null, null, null]
[1, null, null, null, null, null, null, null, null, null]
```

Pregunta 4:

¿Esto es permitido?

```
1  List masNumeros = List(10);
2  masNumeros.add(1);
```

No

```dart
void main (){
List masNumeros = List.filled(10,null);
masNumeros.add(1);
}
```

Si

Tipo de dato - Map

* Todo programa en Dart tiene su función main

Un Map se define por clave valor y es similar a lo que sería el diccionario

```dart
void main() {

  Map persona = {
    'nombre' : 'Adry',
    'edad'   : 22,
    'soltero': true

  };
}
```

Aquí me indica que puede tener cualquier valor como key y cualquier valor como valor de la key en sí

Sin embargo, se puede definir de qué tipo va a ser mi map, al indicar que es dynamic digo que pueda aceptar cualquier tipo de dato, por ej

```dart
void main() {

  Map<String, dynamic> persona = {
    'nombre' : 'Adry',
    'edad'   : 22,
    'soltero': true

  };
}
```

```dart
void main() {

  Map<String, dynamic> persona = {
    'nombre' : 'Adry',
    'edad'   : 22,
    'soltero': true

  };

  print(persona['nombre']);
}
```

```
Console

Adry
```

A diferencia de otros lenguajes pues para acceder a los atributos de mi Map pues uso corchetes

```dart
void main() {

  String propiedad ='Soltero';

  Map<String, dynamic> persona = {
    'nombre' : 'Adry',
    'edad'   : 22,
    'soltero': true

  };

  print(persona['nombre']);
  print(persona['edad']);
  print(persona['propiedad']);

  Map <int, String>personas = {
    1: 'Tonny',
    2: 'Peter',
    9: 'Strange'
  };

  personas.addAll({4:'Banner'});

  print (personas);

  print (personas[9]);
}
```

Run

Console

```
Adry
22
null
{1: Tonny, 2: Peter, 9: Strange, 4: Banner}
Strange
```

Documentation

Funciones en Dart

Para las funciones en Dart pues es recomendable tambien asignar el tipo de dato que la funcion me va a retornar un valor de tipo dinamico es decir cualquier cosa. Y debemos de evitar que Dart infiera todos los datos ya que se pierde funcionalidad y tambien estamos susceptibles a ciertas fallas

```dart
void main() {

  String mensaje = saludar(texto: 'Hola,' ,nombre: 'Adrian' );

  print(mensaje);

}

String saludar({required String texto, required String nombre }) {
  //print('Hello Friend');
  return '$texto $nombre';
}
```

Run

Console

```
Hola, Adrian
```

Para lo que sería la asignación de nombre a los argumentos se hace uso de llaves {} en mi funcion, esto indica que cuando se llame a mi función debo llamar los parámetros tal cual y como están en la función original.
*Ojo que por versiones es necesario usar el required

En Dart también existen las funciones de flecha como en JS

```dart
void main() {

  String mensaje = saludar2(texto: 'Hola,' ,nombre: 'Adrian' );

  print(mensaje);

}

String saludar({required String texto, required String nombre }) {
  //print('Hello Friend');
  return '$texto $nombre';
}

String saludar2({required String texto, required String nombre }) => '$texto $nombre';
```

Run

Console

```
Hola, Adrian
```

Aquí implicitamente esta la palabra return

Clases en Dart

Para crear una nueva instancia es opcional usar el new

```dart
void main() {

  final heroeDelTiempo = new Heroe(poder: "Fuego Din", nombre:"Link"); // el new en Dart es opcional
  // con el final indico que la variable no va a cambiars es como una constante

  print(heroeDelTiempo); // tambien funciona como print(heroeDelTiempo.toString());


}


class Heroe {

  String nombre = '';
  String poder = ''; //  String? poder; de las 2 maneras se puede hacer



  Heroe ({String nombre='Sin Nombre',String poder=''}) {//constructor
    this.nombre = nombre;
    this.poder = poder;
  }

  String toString(){
    return 'nombre: ${this.nombre } - poder: ${ this.poder }';
  }
}
```

Run

Console

```
nombre: Link - poder: Fuego Din
```

Documentation

{String poder}

Es importante analizar que aqui en el constructor de heroe se estan haciendo uso de las llaves, esto con la finalidad de que cuando se cree una nueva instancia pues no exista un orden definido sino mas bien simplemente se llama al atributo en el constructor para setear el valor como se lo hace en la linea 3

Tambien en el metodo toString pues se hace uso de $ y {} para la interpolacion de caracteres es decir para la combinacion de los valores de variables dentro de una cadena

Forma corta de definir propiedades de las clases

```dart
class Heroe {

  String nombre = '';
  String poder = ''; //  String? poder; de las 2 maneras se puede hacer

//   Heroe ({String nombre='Sin Nombre',String poder=''}) {//constructor
//     this.nombre = nombre;
//     this.poder = poder;
//   }

  Heroe({required this.nombre, required this.poder}); // alternativa de constructor


  String toString()=> 'nombre: $nombre - poder: $poder';
//   String toString(){
//     return 'nombre: ${this.nombre } - poder: ${ this.poder }';
//   }
}
```

Constructores con Nombre

```dart
import 'dart:convert';

void main() {

//   final wolverine = new Heroe('Logan','Regeneracion');

  final rawJson = '{"name": "Logan", "power":"Regeneracion"}';
  Map parsedJson = json.decode(rawJson);

  final wolverine =   Heroe.fromJson(parsedJson);

  print(wolverine.nombre);
  print(wolverine.poder);

}


class Heroe {

  String nombre = '';
  String poder = '';

  Heroe(this.nombre, this.poder);

  //Constructor Personalizado, se llamara en este caso fromJson
  Heroe.fromJson(Map parsedJson){
    nombre = parsedJson['name'];
    poder = parsedJson['power'];
  }

}
```

Run

Console

```
Logan
Regeneracion
```

Documentation

```
Heroe Heroe.from
```

Getters y Setters

```dart
void main (){

  final cuadrado = new Cuadrado();

  cuadrado.lado=10;

  print(cuadrado);

  print('Area: ${cuadrado.area }');
}



class Cuadrado {

  double _lado = 0.0; //  Si se pone un guion bajo a las propiedades de clase
//   double _area = 0.0; //  se hacen privadas, es decir funcionan solo dentro de esa clase

  set lado(double valor){ // Setter

    if (valor <=0.0){ //para este caso se hace una validacion
      throw('El lado no puede ser menor o igual a 0');
    }

    _lado = valor;
  }

  double get area => _lado*_lado;


  toString()=>'Lado: $_lado';
}
```

Run

Console

```
Lado: 10
Area: 100
```

Documentation

```
class Cuad
```

Clases abstractas

Una clase abstracta me va a permitir obligar a otras clases a implemetar metodos y propiedades definidos en la clase abstracta.

```dart
void main (){

 final perro = new Perro(); // Las clases abstractas no pueden ser creadas con new
 perro.emitirSonido();

 final gato = new Gato();
 gato.emitirSonido();

}

abstract class Animal {

  int patas= 0;

  void emitirSonido();
}

class Perro implements Animal { //Aqui se obliga a implementar los metodos de la Animal

  int patas=0;
  int colas=0;

  void emitirSonido()=>print("GUAUUUU!!!");

}

class Gato implements Animal {


  int patas=0;

  void emitirSonido()=>print("MIAUUU!!!");
}
```

Run

Console

```
GUAUUUU!!!
MIAUUU!!!
```

Documentation

```
int patas
```

Extends

Es herencia, similar a Java

```dart
void main() {

  final heroeDelTiempo = new Heroe();
  heroeDelTiempo.nombre = "Link";

  final gerudo = new Villano();
  gerudo.nombre="Ganondorf";

  print(heroeDelTiempo);

}
// Al extender la clase puede que otro programador al crear una nueva instancia
// no sepa que la clase no es para crear instancias sino solo para heredar sus
// propiedades entonces se obliga al programador a solo extender las propiedades de
// la clase pues haciendola abstracta

abstract class Personaje {
  String poder="";
  String nombre="";

}

class Heroe extends Personaje{


  int valentia=0;

}

class Villano extends Personaje{

  int maldad=0;
}
```

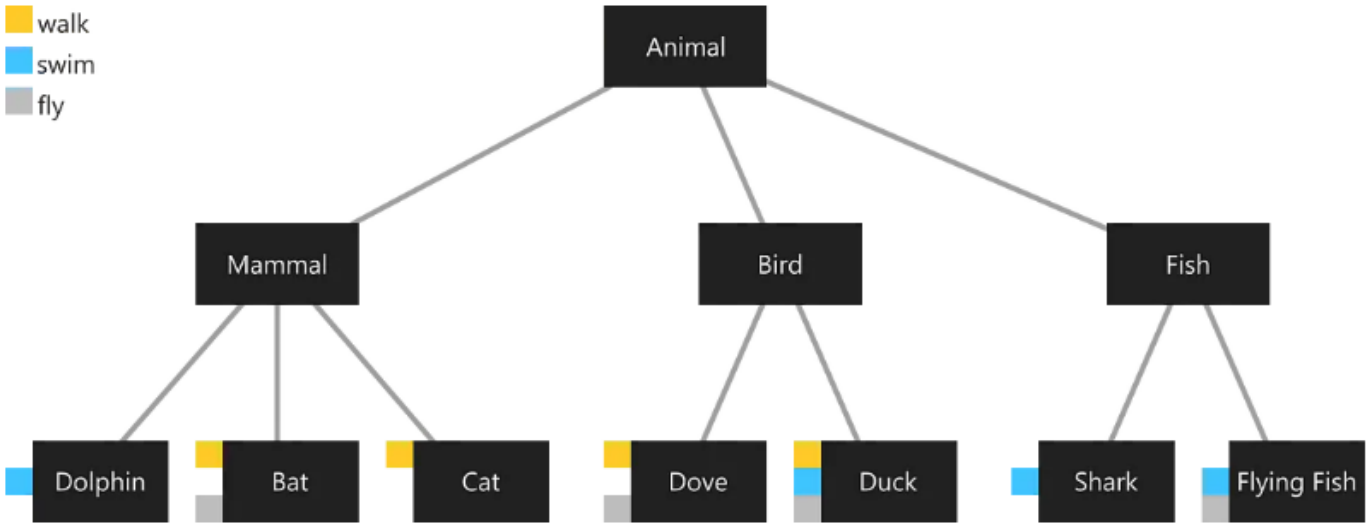
Mixins

https://medium.com/flutter-community/dart-what-are-mixins-3a72344011f3

Es una forma de darle a las clases o de solo asignar lo que nosotros necesitamos para ciertas clases.

Por ejemplo nosotros vamos a tener nuestra clase Animal de la cual se van a extender tres clases como los mamiferos, los pajaros o las aves y los peces.
Pero cada uno tiene diferentes tipos de actividades por ejemplo el delfin es un mamifero pero solo puede nadar, un murcielago que puede caminar y volar pero no nadar y el gato solo puede caminar

Es como las interfaces en Java




```dart
abstract class Animal {

}

abstract class Mamifero extends Animal {}

abstract class Ave extends Animal {}

abstract class Pez extends Animal {}

// Los Mixin son como las interfaces en Java, es decir para que se hereden de mas de
// un padre

abstract mixin class Volador {
  volar ()=> print("Estoy Volando!!");
}

abstract mixin class Caminante {

  caminar ()=> print("Estoy Caminando!!");
}

abstract mixin class Nadador {

  nadar ()=> print("Estoy Nadando!!");
}

class Delfin extends Mamifero with Nadador {}
class Murcielago extends Mamifero with Caminante, Volador{}
class Gato extends Mamifero with Caminante {}
class Paloma extends Ave with Caminante, Volador {}
class Pato extends Ave with Caminante, Volador, Nadador{}
class Tiburon extends Pez with Nadador {}
class PezVolador extends Pez with Nadador, Volador {}
```

```dart
class PezVolador extends Pez with Nadador, Volador {}

void main (){
  final pato = new Pato();
  pato.volar();

  final pezVolador = new PezVolador();
  pezVolador.volar();
}
```

Futures

Es un tema muy importante debido a las tareas asíncronas que se hacen, es decir es como las promesas en JS

Se los define como tarea asincrona que se hace en un hilo independiente al hilo principal que estamos ejecutando y cuando se resuelve es decir cuando ya obtenemos el valor de regreso ahi nosotros podemos obtenerlo y seguir ejecutando otras partes de nuestro programa

```dart
void main (){

  print('Estamos a punto de pedir datos');

  // Para consumir ese sting de mi peticion Future se usa el then
  httpGet("https://api.nasa.com/alientes").then((data){
      print(data);
  });

  print("Ultima Linea");
}



//Se simula una peticion HTTP
Future <String> httpGet(String url){ // se indica que se va a retornar un String <String>

// El delayed es una funcion de los Future que se realiza cuando
// termina cierta cantidad de tiempo
// Como segundo argumento se tiene un Callback o la funcion que yo necesito ejecutar
// luego de ese tiempo.. funcion anonima
  return Future.delayed(new Duration (seconds:4),(){
    return "Hola Mundo";
  });

}
```

```
Estamos a punto de pedir datos
Ultima Linea
Hola Mundo  Luego de 4 sg se ejecuta
```

Async - Await

¿Que pasa si yo quiero que primero se ejecute mi promesa y luego demas codigo?

Para eso usamos el async y el await. El async me ayuda a mi a transformar una funcion en una tarea asincrona y el await me va a permitir esperar hasta que se resuelva la misma

```
void main () async{

  print('Estamos a punto de pedir datos');

  String data = await httpGet("https://api.nasa.com/alientes");

  print (data);

  print("Ultima Linea");
}
```

Run

Console

```
Estamos a punto de pedir datos
Hola Mundo
Ultima Linea
```

Es importante que si queremos usar el await pues a fuerza debe estar el async y no se pueden hacer constructores de una clase asincronos

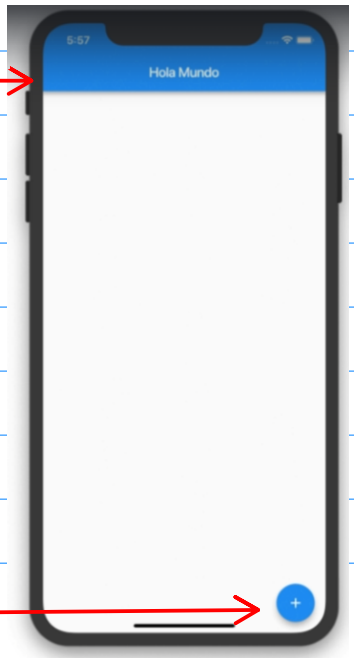
Fundamentos de Flutter

Aprender Flutter requiere mucha práctica

Widget No es mas que una clase que puede tener argumentos posicionales o argumentos con nombre.
- Todo en Flutter, son widgets a excepcion de las clases que se usan para mantener la información y los modelos de datos
- Entiendase como si fueran bloques de LEGO, que tienen funciones específicas

Por ejemplo si yo quisiera poner el AppBar en la parte de arriba pues tendria el codigo asi

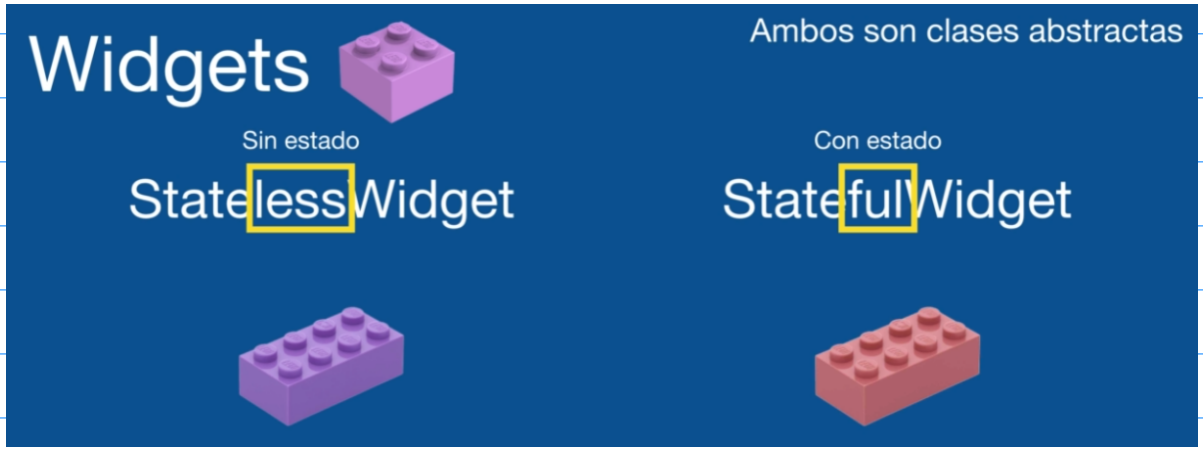
```
AppBar(
  title: Text('Hola Mundo'),
)
```



Es como si fuesen etiquetas para poner cosas específicas

```
FloatingActionButton(
  onPressed: null,
  child: Icon(Icons.add),
)
```

Existen 2 tipos de Widgets, que ambos con clases abstractas



Un Statefull nos va a permitir saber al Widget el estado de sí mismo, por ejemplo si yo debo darle seguimiento a propiedades de la clase que eventualmente pueden cambiar.
Un Stateless no le importa saber si una propiedad cambia, es más no podría cambiar

Ejemplo:

```
class ContactoScreen {
  String nombre = 'Fernando';
```

Statefull => Puede redibujarse a si mismo cuando sucede algun cambio

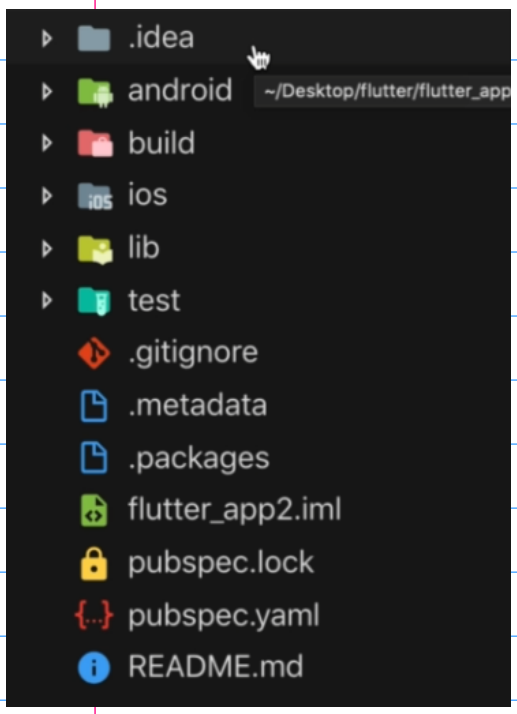
```
class ContactoScreen {
  final nombre = 'Fernando';
```

Stateless => No puede hacerlo de la misma manera porque ni siquiera tiene ese metodo
Boton - boton con icono

Árbol de Widgets

Una aplicacion de Flutter no es mas que una gran cantidad de Widgets, unos dentro de otros. Siempre habrá uno principal que es en el cual se desprenden los otros, se los puede poner horizontal o vertical, sobrepuestos, etc.

Estructura de un proyecto en Flutter



- La carpeta .idea simplemente se utiliza para mantener cierta configuración del entorno de trabajo del editor propiamente.
- El gitignore es un archivo de configuracion que le dice a Git a que archivos hay que dar seguimiento en el repositorio y a cuales no
- La carpeta build hace alusion a lo que vamos ejecutar como tal de nuestra aplicacion
- La carpeta android es la app de android que es codigo de java
- La carpeta de ios contiene todo el proyecto de xcode y lo necesario para que corra
- La carpeta lib es donde nosotros vamos a pasar la mayor parte del tiempo ya que aqui es donde se crea la aplicacion de Flutter
- La carpeta test es practicamente para hacer pruebas de nuestra aplicación de Flutter
- El metada no es tan importante para nosotros pero si para Flutter, es decir este archivo da un seguimiento a las propiedades de nuestro proyecto de Flutter, capacidad, etc. No es recomendable modificar este archivo
- packages bueno no es recomendable modifcarlo tampoco
- flutter_app2.iml es practicamente el nombre del proyecto
- pubspec.lock dice de qué manera es construido mi archivo de pubspec.yaml tampoco debe modificarse, sin embargo el .yaml si lo vamos a modifcar varias veces, es como el package.json en Node.. nos permite instalar de forma remota dependencias de desarrollo, assets. json. imagenes, etc.

Es muy importante mencionar que en este pubspec.yaml son bastante importantes las tabulaciones
Finalmente el README es un archivo que es como la descripcion del proyecto..