

# Temas

**¿Qué es Dart?. Características del Lenguaje**

**Variables, Constantes y tipo de Datos**

**Funciones**

**Listas y Mapas**

**Operaciones Condicionales y Ciclos de Control**

**Clases**

# ¿Qué es Dart?

**Dart es un lenguaje de programación:**

- **Orientado a objetos**
- **Fuertemente Tipado (Los tipos de variables que se conocen en el momento de la compilación. Nos ayuda a escribir un código que es seguro y más predecible)**
- **Sintaxis similar a: C, Java, Javascript**

# Ventajas de Usar Dart

**Dart utiliza un ahead-of-time(AOT) que compila a código nativo, haciendo más rápida una app con Flutter.**

**Es multiplataforma, tanto para dispositivos móviles, web, escritorio y dispositivos embebidos.**

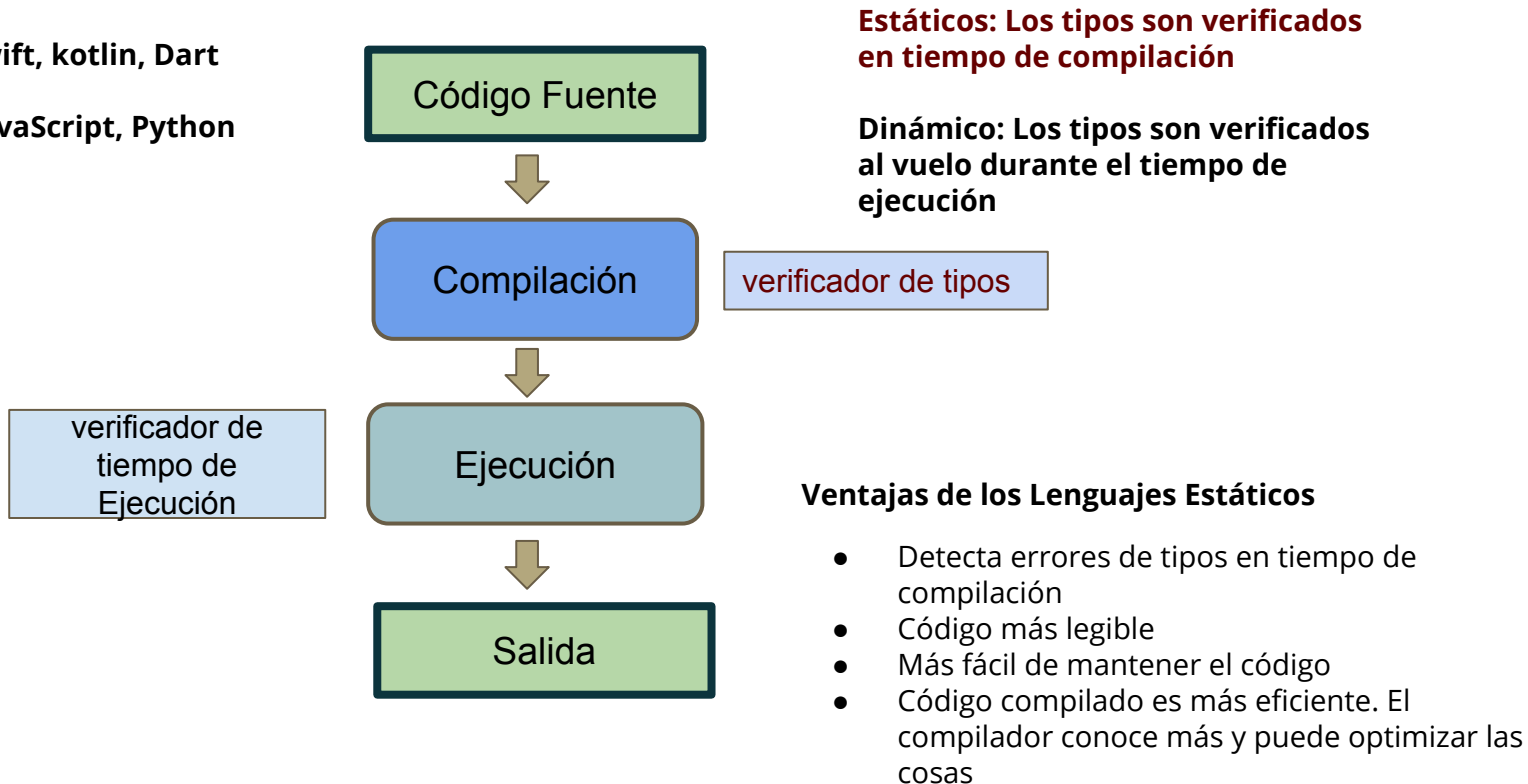
**Dart es compilado just-in-time(JIT), permitiendo mostrar más rápido los cambios en el código.**

# Dart: Lenguaje Estáticamente Tipado

**Dart es un lenguaje Fuertemente tipado**

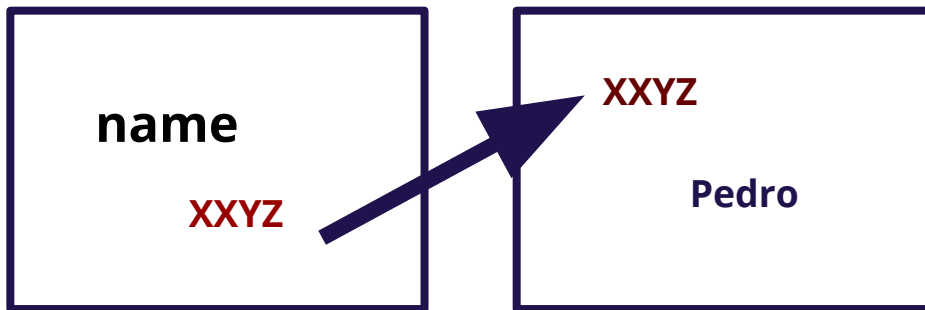
**Estáticos:** Swift, kotlin, Dart

**Dinámico:** JavaScript, Python



# ¿Qué es una Variable?

Las variables son como cajas que tienen una referencia a un valor.



```
var name = 'Pedro';
```

# Variables. Tipos de Variables (entre otras)

**números: num, int , double**

**Cadena: String**

**booleanos: bool**

**De cualquier tipo: Dynamic**

**Listas: list**

**Mapas:map**

Se puede inicializar un objeto de cualquiera de estos tipos especiales usando un literal. Por ejemplo, "esto es una cadena" es un literal de string, y "true" es un literal booleano.

Cada variable en Dart se refiere a un objeto,

# Variables. ¿Cómo nombrarlas?

El nombre de una variable:

- No puede ser una palabra clave
- Puede contener letras mayúsculas y minúsculas
- Puede contener números. No pueden comenzar con un número
- No puede contener espacios y caracteres especiales, excepto el guión bajo (\_) y el signo de dólar (\$)

Se puede seguir un estilo:

->Siempre debes usar letras minúsculas y cada nueva palabra dentro del nombre de la variable debe tener una letra mayúscula. Ejemplo: `var myVariable="hola";`

# Ejemplos:

`var nombreUno = "Pedro"; // Se infiere que la variable es de tipo String`

`String nombreDos = "Juan"; // Se especifica el tipo de la variable String`

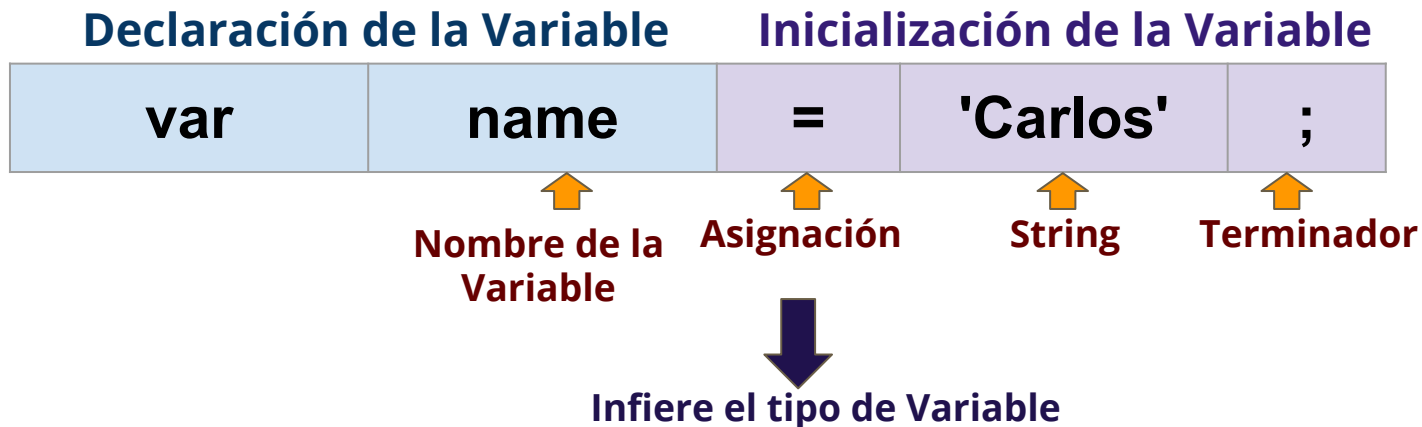
`bool esValido=true; //Se especifica el tipo de la variable booleana(V o F)`

`int count; // Las variables no inicializadas tienen un valor null`

`dynamic nombreTres = 'Carlos'; // La variable podrá ser de cualquier tipo`



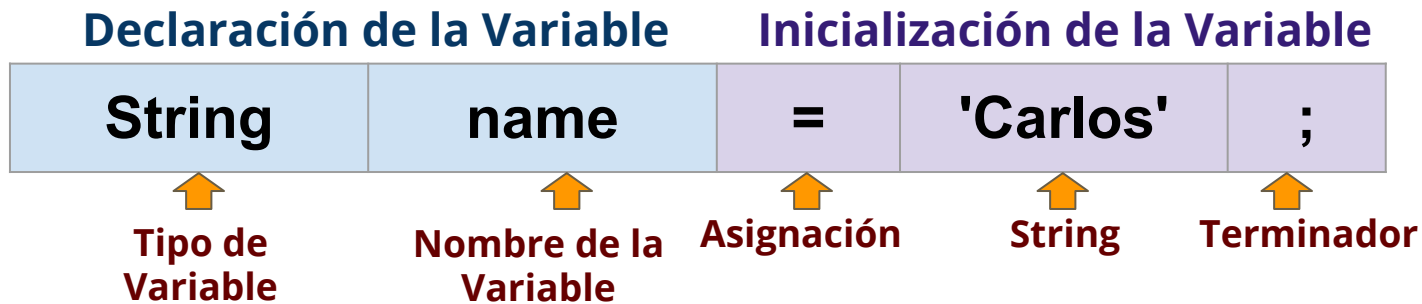
# Declaración de Variables con Inferencia de Tipo



**Ejemplos:**

```
var edad=20;  
var altura=1.80;  
var cadena="hola c";  
var testB=true;
```

# Declaración de Variable con Tipo



**Ejemplos:**

```
int edad=20;  
double altura=1.80;  
String cadena="hola c";  
bool testB=true;
```

# String->Concatenar - Interpolar

```
String nombre='Juan';  
String apellido='Perez';  
int edad=20;  
double altura=1.80;
```

## Concatenar

```
print("Mi nombre es : "+nombre+" "+apellido);
```

**Consola:**

Mi nombre es : Juan Perez

## Interpolar

```
print("Mi nombre es : $nombre $apellido");
```

**Consola:**

Mi nombre es : Juan Perez

```
print("Mi nombre es : $nombre $apellido, mi  
edad es: $edad, mialtura $altura");
```

**Consola:**

Mi nombre es : Juan Perez, mi edad es: 20,  
mialtura 1.8

# Interpolación con llaves

```
print("El siguiente año voy a cumplir $edad 1 años de edad");
```

```
print("El siguiente año voy a cumplir ${edad+1} años de edad");
```

## Consola

El siguiente año voy a cumplir 20 1 años de edad

El siguiente año voy a cumplir 21 años de edad

**\*Se puede evaluar una expresión dentro de un String con la sintaxis `${...}`**

# Ejercicio

**Dadas las siguientes Variables**

```
double temperatura=20;
```

```
int valor=2;
```

```
String pizza=' pizza';
```

```
String pasta=' pasta';
```

**Escribir un programa que produzca la siguiente salida:**

La temperatura es de 20 grados

2 más 2 es 4

Yo quiero una pizza y pasta

# Solución

```
void main() {  
    double temperatura = 20;  
    int valor = 2;  
    String pizza = ' pizza';  
    String pasta = ' pasta';  
  
    print("La temperatura es de ${temperatura} grados");  
    print("${valor} más ${valor} es ${valor+valor}");  
    print("Quiero una $pizza y $pasta");  
}
```

# Caracteres de Escape

```
void main() {
```

```
    print("La Pizza de 'Don Pizza' es rica");
```

```
    print('La Pizza de \'Don Pizza\' es rica');
```



Carácter de Escape

```
    print('\\');
```

```
    print('\$');
```

```
    print(r'C:\Windows\System');
```



Carácter de Escape

```
}
```

# Ejercitación

Definir la salida sin ejecutar:

```
void main() {  
    int a=10;  
    print('$a');  
    print('\$a');  
    print('\$$a');  
}
```



# Solución

```
void main() {  
    int a=10;  
    print('$a');  
    print('\$a');  
    print('\$$$a');  
}
```

Console

```
10  
$a  
$10
```

# Multilineas con String

```
void main() {  
    print('Esta es una sentencia corta.\n'  
          'siguiente línea.\n'  
          'última línea.\n'  
          );  
  
    print("""  
Esta es una sentencia corta.  
siguiente línea.  
última línea.  
        """);  
}
```

## Console

Está es una sentencia corta.  
siguiente línea.  
última línea.

Está es una sentencia corta.  
siguiente línea.  
última línea.

# Operaciones con String(mayúsculas y minúsculas)

```
void main() {  
    String titulo="Curso de Dart";  
    print(titulo);  
    titulo=titulo.toLowerCase();  
    print(titulo);  
    titulo=titulo.toUpperCase();  
    print(titulo);  
}
```

**Respuesta**

Curso de Dart  
curso de dart  
CURSO DE DART

# Conceptos Importantes

**Todo lo que se puede colocar en una variable es un objeto y cada objeto es una instancia de una clase**

**Los números, las funciones y los nulos son objetos. Todos los objetos heredan de la clase Object.**

**Aunque Dart es fuertemente tipado, las anotaciones de tipo son opcionales porque Dart puede inferir tipos.**

**Cuando quieras decir explícitamente que no se espera ningún tipo, usa el tipo especial dynamic.**

# Constantes: final y const

**Si no se necesita cambiar un valor entonces usar: final o const**

```
final name="hola c";
```

```
const PI=3.14;
```

## **Diferencia entre final y const:**

La variable final solo puede ser seteada una vez y es inicializada cuando se accede. Se puede inicializar en tiempo de ejecución.

Se define una variable como const cuando es constante en tiempo de compilación. Es decir, se inicializa en el código.

**Una variable de instancia puede ser final pero no puede ser const.**

Para que sea const se tiene que declarar como: static const

# Diferencia entre final y const

La diferencia esencial es que una variable **const** **no puede ser algo a calcular a la hora de la ejecución**, es decir, deben ser inicializadas con un valor constante, un ejemplo sería la fecha actual, no se puede hacer:

```
const date = new DateTime.now();
```

```
Const variables must be initialized with a constant value.
```

```
Try changing the initializer to be a constant expression.
```

Se podría utilizar, por ejemplo, para algo como:

```
const dias_de_la_semana = 7;
```

Por otro lado, una variable **final**, aunque **inmutable** (es decir, su valor no puede cambiar), sí que se puede inicializar con algo en tiempo de ejecución:

```
final date = new DateTime.now();
```

En resumen, las variables **final** se pueden instanciar en tiempo de ejecución y los **const** NO

# ¿Cuándo usar final o const?

## Usar final:

Si no sabe cuál será su valor en tiempo de compilación.

Por ejemplo, cuando puede necesitar obtener datos de una API, esto sucede cuando ejecuta su código.

## Usar const:

Si está seguro de que no se cambiará un valor al ejecutar su código.

Por ejemplo, cuando se declara una oración que siempre permanece igual.

# Operadores

## Operadores de igualdad y relacionales

```
== // Es igual
!= // No es igual
> // Mayor que
< // Menor que
>= // Mayor o igual que
<= // Menor o igual que
```

## Operadores Aritméticos

```
+ // suma
- // resta
-expr // negación
* // multiplicación
/ // división
~/ // divide y retorna un resultado entero
% // módulo, retorna el resto de la división
```

## Operadores de Test de Tipos

```
as // Typecast
is // True si el objeto tiene el tipo especificado
is! // False si el objeto tiene el tipo especificado
```

## Operadores de Incremento y Decremento

```
++var // var = var + 1, retorna var + 1
var++ // var = var + 1, retorna var
--var // var = var - 1, retorna var - 1
var-- // var = var - 1, retorna var
```

## Operadores Lógicos

```
!expr // invierte la expresión, de false a true, y viceversa.
|| // or lógico
&& // and lógico
```



# Operaciones condicionales

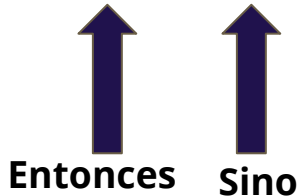
La condición If-Else evalúa cierta condición y ejecuta un bloque código en caso de que la condición sea válida en caso contrario utiliza otro bloque de código o incluso puede realizar otra evaluación anidada.

```
var precio = 500;  
if (precio == 500) {  
    print("Precio promedio");  
} else if (precio > 500) {  
    print("Precio mayor del  
promedio");  
} else {  
    print("Precio normal");  
}
```

## Operador ternario

```
var precio = 500;  
var esBarato = precio < 500 ? true : false;
```

Entonces Sino



# Switch

Evalúa una variable según diversos casos. Se pueden utilizar etiquetas como **continue** para saltar a otro caso dentro del switch.

```
switch(number) {  
    case 10:  
        print("Es un 10!");  
        continue cien;  
        break;  
    case 20:  
        print("Es un 20!");  
        break;  
    cien:  
    case 100:  
        print("Es un 100!");  
        break;  
    default:  
        print("Es otro número!");  
}
```

# Ciclos de Control

**Imprime una lista del 0 al 9**

```
for (int i = 0; i < 10; i++) {  
    print(i);  
}
```

**Recorrer una lista usando el método `forEach()` de la lista con una función de flecha.**

```
var lista = ['Uno', 'Dos', 'Tres'];  
lista.forEach((num) => {  
    print(num)  
});
```

**Recorrer una lista usando la palabra reservada `in` que toma el valor actual del recorrido de la lista.**

```
var lista = ['Uno', 'Dos', 'Tres'];  
for (String num in lista) {  
    print(num);  
}
```

**El ciclo `while` evalúa una condición y si es válida ejecuta un bloque de código mientras la condición siga afirmativa. Por ejemplo, Imprimir una lista del 0 al 9.**

```
var num = 0;  
while (num < 10) {  
    print(num);  
    num++;  
}
```

# Función

**Las funciones son los bloques de construcción de código para permitir que este sea: legible, mantenible y reutilizable.**

**Una función es un conjunto de instrucciones para realizar una tarea específica.**

**Las funciones organizan el programa en bloques lógicos de código.**

**Una vez definidas, las funciones pueden ser llamadas para acceder al código. Esto hace que el código sea reutilizable.**

**Además, las funciones facilitan la lectura y el mantenimiento del código del programa.**

# Propiedades de las Funciones en Dart

## Las funciones en Dart son Objetos

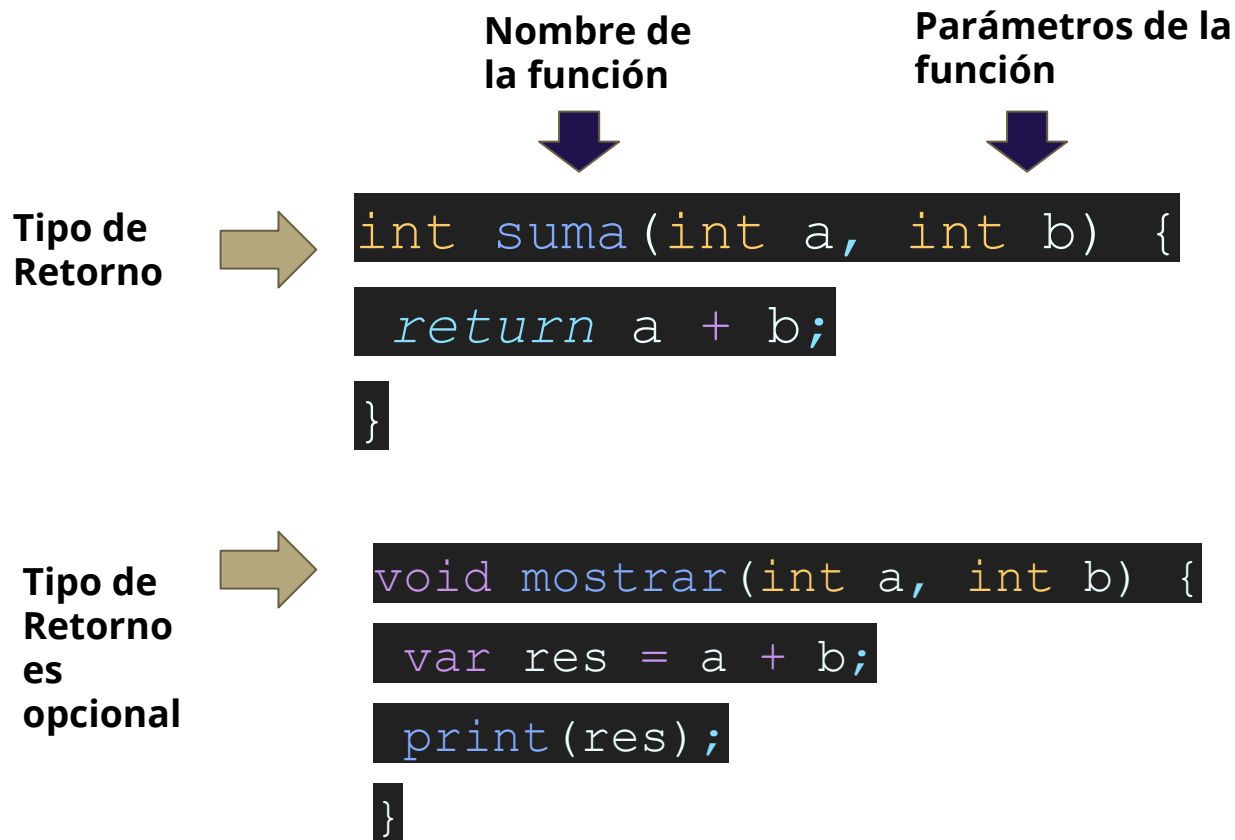
Las funciones pueden ser asignadas a una variable o pasadas como parámetro a otras funciones.

## Todas las funciones en dart retornan un valor, salvo si la declaramos void

Si no se especifica un valor de retorno entonces la función retorna un null. Se declara void la función si no necesitamos que no retorne un valor.

## Especificar el tipo de retorno es opcional pero es recomendable hacerlo por convención de codificación

# Función



# Funciones

En Dart las funciones también son objetos, lo que significa que las funciones se pueden asignar a variables e incluso podemos pasarlas como parámetros de otras funciones como un objeto de tipo Function.

```
main(List<String> arguments) {  
  var operacion1 = operacion(5, 3, suma);  
  var operacion2 = operacion(5, 3, resta);  
  print(operacion1);  
  print(operacion2);  
}
```

```
int operacion(int a, int b, Function func) {  
  return func(a, b);  
}
```

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

```
int resta(int a, int b) {  
  return a - b;  
}
```

# Función Flecha (Arrow Functions)

\*Si la función ejecuta una sola expresión o sentencia, se puede usar la sintaxis de la flecha ( => )

*flecha*

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

*nombre*

*parámetros*

*retorno directo*

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

*retorno*

The diagram illustrates the components of both function syntaxes. For the arrow function 'int sumflecha(int a, int b)=>(a+b);', labels point to 'sumflecha' (nombre), '(int a, int b)' (parámetros), and '=>(a+b)' (retorno directo). The word 'flecha' is written above the arrow symbol. For the regular function 'int sum(int a,int b) { return a+b; }', labels point to 'sum' (nombre), '(int a,int b)' (parámetros), and 'return a+b;' (retorno).



# Función Flecha

```
void main() {
```

```
    int resp=sum(2,3);
```



```
    print(resp);
```

```
int sum(int a,int b) {
```

```
    return a+b;
```

```
}
```

```
int resp2=sumflecha(2,3);
```



```
print(resp2);
```

```
}
```

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

# Función Flecha (Función Lambda)

También podemos escribir las funciones de una forma más abreviada, siempre y cuando sea una sola sentencia a ejecutar, por lo que el ejemplo anterior puede escribirse de la siguiente manera:

```
int operacion(int a, int b, Function func) {  
    return func(a, b);  
}
```



```
int operacion(int a, int b, Function func) => func(a, b);
```

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



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

```
int resta(int a, int b) {  
    return a - b;  
}
```



```
int resta(int a, int b) => a - b;
```

# Función Anónima con la Función =>

```
var dup1 = (double x) {  
    return 2.0 * x;  
};
```

```
var dup2 = (double x) => 2.0 * x;
```

```
main(List<String> arguments) {  
    print(dup1(2));  
    print(dup2(2));  
}
```

# Parámetros

// Parámetros Requeridos

```
void printCities(String name1, String name2, String  
name3) {}
```

Requeridos

// Parámetros Opcionales Posicionales

```
void printCountries(String name1, [String name2,  
String name3]) {}
```

Opcionales

Posicionales

Nombrados

Por defecto

**\*solo los últimos  
parámetros**

// Parámetros Opcionales Nombrados

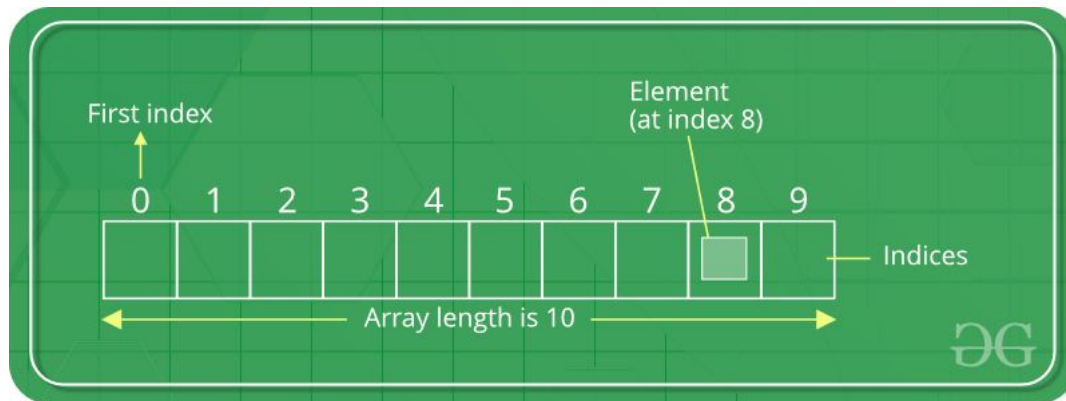
```
void printCountries(String name1, {String name2,  
String name3}) {}
```

// Parámetros Opcionales Por Defecto

```
void printCountries(String name1, String name2,  
{String name3="valorDefecto"}) {}
```

# Listas

List se utiliza para representar una colección de objetos. Es un grupo ordenado de objetos. Las bibliotecas centrales de Dart son responsables de la existencia de la clase List, su creación y manipulación.



**Index del elemento** representa la posición de los datos específicos y cuando el elemento de la lista de ese índice se llama el elemento se muestra. Generalmente, el elemento de la lista se llama desde su índice.

# Listas

```
List myList = new List(); // empty list
```

```
var myList = [];
```

```
myList.add("item 1");
```

```
myList.add("item 2");
```

```
myList[0] = "ITEM 1"; // replaces "item 1";
```

```
print(myList[1]); // "item 2";
```

```
print(myList.length); // 2
```

# Listas Iterar

```
void main() {
```

```
    var miObjeto1=new MiObjeto();  
    var miObjeto2=new MiObjeto();
```

```
    List myList = new List(); // empty list  
    myList.add(miObjeto1);  
    myList.add(miObjeto2);
```

```
    myList.forEach( (item) { //  
        MiObjeto objitem=item;  
        print(objitem.atributo);  
    } );  
}
```

```
class MiObjeto {  
    var atributo=23;  
}
```

# Mapas

Un mapa es un objeto que asocia claves y valores.

Tanto las claves como los valores pueden ser cualquier tipo de objeto.

Cada key es única.

```
var gifts = {  
  // Key:  Value  
  'first': 'partridge',  
  'second': 'turtledoves',  
  'fifth': 'golden rings'  
};
```

```
var nobleGases = {  
  2: 'helium',  
  10: 'neon',  
  18: 'argon',  
};
```



# map

<https://dzone.com/refcardz/core-dart?chapter=4>

## // Map Literal syntax

```
Map myMap = {"key1":"value1", "key2":"value2"};
```

## // or

```
var myMap = {"key1":"value1", "key2":"value2"};
```

```
var myMap = {}; // empty map
```

## // Map constructor syntax

```
Map myMap = new Map(); // empty map
```

## // or

```
var myMap = new Map(); // empty map
```

# map Iterar

```
void main() {
```

```
    var ejMap=new Map<String, dynamic>();
```

```
    ejMap.addAll({"key1":"value1", "key2":"value2"});
```

```
    ejMap["key3"] = "value3";
```

```
    ejMap.forEach( (key, value) {
```

```
        print("$key = $value");
```

```
    } );}
```

# ¿Qué es una Clase?

Una clase es una plantilla para la creación de objetos según un modelo predefinido.

**Las clases se utilizan para representar entidades o conceptos**

**Cada clase es un modelo que define:**

- **Un conjunto de variables -> el estado**
- **Métodos (funciones) apropiados para operar con dichos datos -> el comportamiento.**

**Cada objeto creado a partir de la clase se denomina instancia de la clase.**

# Ejemplo

## Clase Punto

### Variables

x

y

color

### Métodos (o funciones)

Crear

Mostrarse

Ocultarse

moverse

## Objeto 1

### Variables

21

45

verde

## Objeto 2

### Variables

102

20

rojo

# Ejemplo

## Clase



### Clase Persona

#### Variables

nombre  
apellido  
edad

#### Métodos

Persona(nombre, apellido, edad)  
hablar()  
caminar()  
correr

## Objeto



### Juan:Persona

#### Variables

nombre="Juan"  
apellido="Perez"  
edad=15;

#### Métodos

Persona(nombre,  
apellido, edad)  
hablar()  
caminar()  
correr

# Clases en Dart

**Todas las clases descienden de Objeto , la clase base de todos los objetos Dart.**

**Una clase tiene miembros (variables y métodos) y utiliza un constructor para crear un objeto.**

**Si no se declara un constructor, se proporcionará automáticamente un constructor por defecto.**

**El constructor por defecto que se proporciona no tiene argumentos.**

# ¿Qué es un constructor y por qué se necesita?

Un constructor tiene el mismo nombre que la clase, con parámetros opcionales. Los parámetros sirven para obtener valores cuando se inicializa una clase por primera vez.

```
void main() {  
    var persona = Persona("Juan");  
    print(persona.nombre);  
}
```

```
class Persona {  
    String nombre;  
    // Constructor - Mismo nombre que clase  
    Persona(this.nombre);  
}
```

# Constructores

## Constructores predeterminados

Si no declara un constructor, se le proporciona un constructor predeterminado. El constructor por defecto no tiene argumentos e invoca al constructor sin argumentos de la superclase.

## Los constructores no se heredan

Las subclases no heredan constructores de su superclase. Una subclase que declara que no hay constructores sólo tiene el constructor por defecto (sin argumento, sin nombre).

## Constructores nombrados

Utilice un constructor nombrado para implementar múltiples constructores para una clase o para proporcionar claridad adicional:



# Getters y Setters

Son métodos especiales que proporcionan acceso de lectura y escritura a las propiedades de un objeto.

Dart encapsula las variables de instancia con sus métodos de acceso get y set. Cada variable de instancia tiene un getter implícito, más un setter si es apropiado.

Se pueden personalizar->Se pueden crear propiedades adicionales implementando getters y setters, usando las palabras clave get y set:

# Herencia

Para que una clase herede los atributos y métodos de otra clase usamos la palabra reservada `extends` de la siguiente forma.

```
class Persona {
    String _nombre;
    int _edad;

    Persona(this._nombre, this._edad);

    set edad(int edad) => this._edad = edad;
    int get edad => _edad;

    String get mostrarNombre => this._nombre;
}

class Empleado extends Persona {
    double _sueldo;
    Empleado(String nombre, int edad, this._sueldo) : super(nombre, edad);

    int get edad => _edad + 1;
    set sueldo(double sueldo) => this._sueldo = sueldo;
    double get sueldo => _sueldo;
}
```

Las subclases no heredan constructores de su superclase. Una subclase que declara que no hay constructores sólo tiene el constructor por defecto (sin argumento, sin nombre).

# Interfaces

**Dart no tiene una sintaxis especial para declarar una interface.**

**Una interface en Dart es una clase normal**

**Una interfaz se utiliza cuando se necesita una implementación concreta de todas sus funciones dentro de una subclase**

- **Es obligatorio anular(`@override`) todos los métodos en la clase que los implementa**

**Se pueden implementar múltiples clases pero:**

- **No se pueden extender múltiples clases durante la herencia**

# Métodos Abstractos

Para hacer un método abstracto, utilizamos un punto y coma (;) en lugar del cuerpo del método.

**Los métodos abstractos sólo pueden existir en clases abstractas.**

**Es necesario anular y sobrescribir(@override) los métodos abstractos en las subclases**

# Clases Abstractas y Métodos Abstractos

Usar la palabra clave `abstract` para declarar una clase abstracta.

Una clase abstracta no puede ser instanciada. No se pueden crear objetos. Útiles para definir interfaces.

Una clase abstracta puede tener: métodos abstractos, métodos normales y variables de instancia

Permiten a sus clases hijas redefinir sus variables y sus funciones

Las funciones definidas pueden o no estar implementadas

También las variables de la clase padre se redefinen en la clase hija

Las clases hijas heredan de una clase abstracta usando la palabra `extends` y el nombre de la clase padre

# Otros Temas Importantes

**Mixins** (<https://ptagicodecamp.github.io/dart-mixins.html>)

**Genéricos**

**Programación Asíncrona (Future)**

# Tarea

**Crear una clase Producto como se muestra a continuación:.**

```
class Producto {  
    final int id;  
    final double precio;  
    final String nombre;  
    Producto(this.id, this.precio, this.nombre);  
  
    @Override  
    String toString() {  
        return "Precio de ${this.nombre} is \${this.precio}";  
    }  
}
```

**Elaborar una lista de 4 objetos Producto.**

**Mostrar los valores de sus variables por pantalla si el precio del producto es mayor o igual a 100.00 y menor a 1000.00.**