Dart

**DOCUMENTO DE USO INTERNO**

La información contenida en este documento ha sido clasificada como de uso interno y, por tanto, usted deberá ser empleado de Helicon para manejar el presente documento. En caso contrario, favor de entregar este documento a cualquier empleado de Helicon. En caso de ignorar este aviso estará sujeto a las medidas legales aplicables.

Este documento es para uso exclusivo del personal de **Helicon.** Ninguna de sus partes puede ser circulada, citada o reproducida para distribución fuera de las instalaciones **Helicon.**

**HISTORIAL DE VERSIONES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Vers.** | **Descripción** | **Elaborado por:** | **Revisado por:** | **Autorizado por:** |
| 1.0 | Creación del documento para el proceso. | Nombre: Osvaldo Flores. | Nombre: Angelo Raimondi. | Nombre: Angelo Raimondi |
|  |  |  |  |  |

Contenido

[Tipos de datos 6](#_Toc29131579)

[Numbers 6](#_Toc29131580)

[Strings 6](#_Toc29131581)

[Booleans 6](#_Toc29131582)

[Lists 6](#_Toc29131583)

[Sets 7](#_Toc29131584)

[Maps 7](#_Toc29131585)

[Runes 7](#_Toc29131586)

[Symbols 7](#_Toc29131587)

[Variables 8](#_Toc29131588)

[Var y dynamic 8](#_Toc29131589)

[Final y const 8](#_Toc29131590)

[Estilos 9](#_Toc29131591)

[Operadores 10](#_Toc29131592)

[Aritméticos: 10](#_Toc29131593)

[Comparación: 10](#_Toc29131594)

[Decremento e incremento: 11](#_Toc29131595)

[Lógicos: 11](#_Toc29131596)

[Null – Aware: 11](#_Toc29131597)

[Cascade operator: 11](#_Toc29131598)

[Condicionales 12](#_Toc29131599)

[If - else 12](#_Toc29131600)

[Operador ternario 12](#_Toc29131601)

[Ciclos 12](#_Toc29131602)

[For 12](#_Toc29131603)

[For Each 12](#_Toc29131604)

[For In 12](#_Toc29131605)

[While 12](#_Toc29131606)

[Do – While 12](#_Toc29131607)

[Break – Continue 12](#_Toc29131608)

[Switch 13](#_Toc29131609)

[Assert 13](#_Toc29131610)

[Exceptions 13](#_Toc29131611)

[Try - Catch 13](#_Toc29131612)

[On Catch 13](#_Toc29131613)

[Throw 13](#_Toc29131614)

[Finally 13](#_Toc29131615)

[Funciones 14](#_Toc29131616)

[¿Qué es una función? 14](#_Toc29131617)

[First-Class-Object 14](#_Toc29131618)

[Función flecha 14](#_Toc29131619)

[Función anónima 14](#_Toc29131620)

[Parámetros requeridos y opcionales 14](#_Toc29131621)

[Parámetros con valores default 14](#_Toc29131622)

[Función main 14](#_Toc29131623)

[Diferencia entre acceder a la función y crear una referencia. 14](#_Toc29131624)

[Clases 15](#_Toc29131625)

[¿Qué es una clase? 15](#_Toc29131626)

[Objetos 15](#_Toc29131627)

[Variables de instancia 15](#_Toc29131628)

[Constructor 15](#_Toc29131629)

[Constructor con nombre 15](#_Toc29131630)

[Initializer list 15](#_Toc29131631)

[Factory constructors 15](#_Toc29131632)

[Métodos 16](#_Toc29131633)

[Getter y Setter 16](#_Toc29131634)

[Herencia 16](#_Toc29131635)

[Abstract 16](#_Toc29131636)

[Interfaces 16](#_Toc29131637)

[Mixins 16](#_Toc29131638)

[Static 16](#_Toc29131639)

[Generics 17](#_Toc29131640)

[Programación asíncrona 18](#_Toc29131641)

[Future 18](#_Toc29131642)

[Stream 18](#_Toc29131643)

[Comentarios 19](#_Toc29131644)

[Comentarios de una sola línea 19](#_Toc29131645)

[Comentarios de múltiple línea 19](#_Toc29131646)

[Comentarios de documentación 19](#_Toc29131647)

# **Tipos de datos**

## Numbers

Los números en Dart están divididos por **int** y **double**. Ambos siendo subtipos de la clase **num**. En donde int hace referencia a los números enteros (sin punto decimal) y double a los números con punto decimal.

## Strings

Un string por definición es una cadena de caracteres, representan una cadena de Unicode UTF-16 unidades. Puede ser escrito con comillas dobles o comillas simples.

Se pueden poner valores dentro de una cadena de caracteres como expresiones con ${expresión} o identificadores en donde no hay necesidad de utilizar llaves $identificador. A este concepto se le conoce como **string interpolation**.

Una **expresión** es una combinación de variables o constantes. (5+3), (x+y+z), (5+z).

Un **identificador** puede ser una variable, una constante, algún objeto que contenga una referencia. (x), (y), (z).

Dos o más strings pueden concatenarse (unirse) mediante el uso del operador +. Si son dos cadenas literales no hay necesidad de usar el operador.

Se soporta el uso de **multi-line string** utilizando triple comilla doble o simple, con el fin de crear un string de multi línea.

Así mismo puede crearse un **raw string** agregando r como prefijo de nuestra cadena de caracteres.

## Booleans

Para representar los valores booleanos se utiliza **bool**. Solo aceptan dos valores, true y false.

*Los condicionales solo comparan las expresiones mediante booleanos.*

## Lists

Una **lista** es una colección ordenada de objetos. Las listas en otros lenguajes son conocidas como arrays.

Cada uno de los elementos de una lista tiene un **index**, en todas las listas su index inicial es 0 y su final es la cantidad de elementos – 1.

Las listas tienen varios métodos; algunos métodos son:

add() → Inserta un elemento al final.

length() → Devuelve la cantidad de elementos.

addAll() → Inserta múltiples elementos.

removeAt() → Remueve un elemento mediante el index.

Las colecciones cuentan con dos operadores muy útiles, **spread operator** y **null-aware operator**:

**…** → Retorna a los elementos de una lista.

**…?** → Si la lista es nula no los retorna.

## Sets

Un **set** es una colección desordenada de objetos únicos. Tiene relación con los conjuntos matemáticos, en donde rechazará los elementos que ya existan en su conunto.

*Tanto sets como lists son subclases de* ***Iterable****.*

## Maps

Un **map** es una colección que asocia un key y un valor, en donde el key es un elemento único y el valor puede ser utilizado múltiples veces.

## Runes

Los **runes** son de ayuda para acceder a más caracteres de los que un string puede, mediante valores hexadecimales expresados en un string podemos acceder a emojis, entre otras cosas.

**\uXXXX** → En donde XXXX es un hexadecimal de 4 dígitos.

**\u{XXXX}** → En donde XXXX será un valor hexadecimal de menos o más de 4 dígitos.

Symbols

Los **symbols** sirven para manejar información del archivo o librería, mediante un mecanismo que puede conseguir el tipo de dato al tiempo de ejecución, el número de métodos en una clase, el numero de constructores que tiene o el número de parámetros de una función.

# **Variables**

## Var y dynamic

Las variables almacenan referencias, cuando le asignamos un valor a cierta variable, si no especificamos el tipo de dato mediante genéricos entonces la variable infiere cual será el tipo de dato.

El concepto de **inferir** hace referencia a que la variable sabrá cual es el tipo de dato que se está alojando.

Por defecto todas las variables sin inicializar tendrán el valor de **null**.

Una variable puede ser:

**var** → Una vez se asigna o infiere el tipo de dato, no podrá cambiarlo.

**dynamic** → Puede cambiar el tipo de dato sin importar que ya haya sido otro asignado.

## Final y const

Ambas variables funcionan como constantes, con la única diferencia es que **final** es una constante en tiempo de ejecución y **const** es una constante en tiempo de compilación.

Además de crear variables constantes, se puede inicializar valores constantes con **const**.

# **Estilos**

Los identificadores en Dart se podrán escribir de tres maneras.

**UpperCamelCase** La primera letra de cada palabra será en mayúscula, incluyendo la primera palabra.

**lowerCamelCase** La primera letra de cada palabra será en mayúscula, excepto la primera palabra.

**lowercase\_with\_underscores** Cada palabra será en minúscula (incluso los acrónimos) y separar las palabras con \_.

Usar **UpperCamelCase** en: clases, enums, typedefs, tipos de datos en parámetros, tipos en genéricos, extensiones.

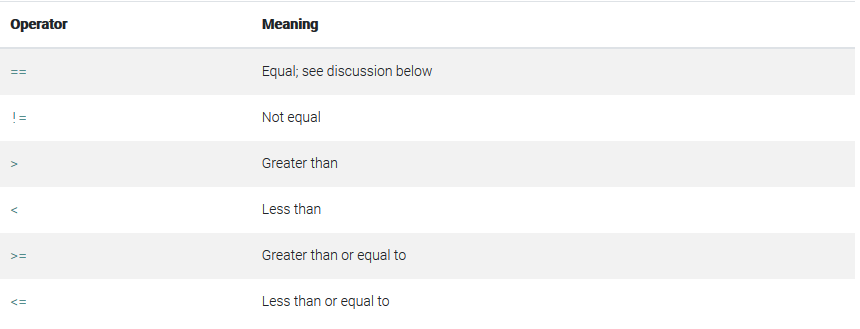
Usar **lowerCamelCase** en: atributos, top-level definitions, variables, constantes, final y parámetros con nombre.

Usar **lowercase\_with\_underscores** en: librerías, paquetes, directorios, códigos fuente, prefijos en nombres de importación.

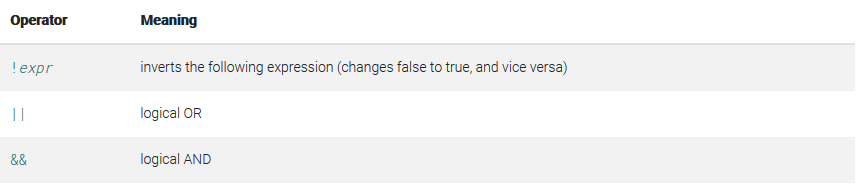
# **Operadores**

Dart soporta los siguientes operadores:

## Aritméticos:

Comparación:

## Decremento e incremento:

Lógicos:

## Null – Aware:

**?.** → Si lo de la izquierda no es nulo ejecutara el método sino retorna nulo.

**??** → Retorna el valor de la izquierda, si este es nulo retornará el de la derecha. **??=** → Si el de la izquierda es nulo asigna el valor de la derecha.

## Cascade operator:

El cascade operator nos permite hacer una secuencia de operaciones en el mismo objeto. En adición al llamar funciones podemos acceder a variables del mismo objeto.

# **Condicionales**

## If - else

En Dart puedes controlar tu código mediante el uso de condicionales (**if – else**), en donde si la condición se cumple entra a cierta línea de código y en dado caso de que no entonces opcionalmente se puede agregar una contraparte.

## Operador ternario

Así mismo Dart soporta el operador ternario, el cual si la condición se cumple retorna el primer valor sino el segundo. Nos será de ayuda para realizar un código más sencillo de leer.

# **Ciclos**

## For

Un ciclo **for** repetirá cierta parte de código un número determinado de veces.

## For Each

Si estas tratando con un objeto *iterable* puedes utilizar el método **forEach**(), el cual recorrerá dicho objeto.

## For In

Las clases **iterables** como List y Set soportan el uso del **for in**. El cual recorre dicho iterable teniendo acceso en cada iteración a un objeto.

## While

Un ciclo **while** repetirá cierta parte de código hasta que se cumpla cierta condición. Primero evalúa la condición, después ejecuta el código.

## Do – While

Un ciclo **do-while** repetirá cierta parte del código hasta que se cumpla cierta condición. Primero ejecuta el código y después evalúa la condición.

## Break – Continue

Un **break** hace que se detenga el ciclo en el que se está trabajando.

Un **continúe** hace que el ciclo pase a la siguiente iteración.

# **Switch**

Un **switch** evalúa una expresión para comparar en que caso entrará, si encuentra un caso ejecuta esa parte de código. Si la expresión no entra en ninguno de los casos entonces no ejecutará nada. De manera opcional se puede agregar un caso **default** el cual se ejecutará si la expresión no entra a ningún caso.

En el switch los casos son definidos mediante la palabra reservada **case**.

# **Assert**

En Dart existen los **asserts** los cuales evalúan una condición booleana, en caso de ser falsa dicha condición retornará un *exception*.

Se puede agregar un mensaje opcional en caso de ser necesario.

# **Exceptions**

## Try - Catch

Los exceptions son errores que indican que algo inesperado pasó. Para poder capturar un exception es necesario usar la palabra reservada **try** la cual nos ayudará a ejecutar cierto código y en caso de error, este será capturado. Se puede capturar dicho error con la palabra reservada **catch** y trabajar con él.

## On Catch

Para capturar un error en específico utilizamos la palabra reservada **on.**

## Throw

Se puede tirar algún mensaje en caso de error con la palabra reservada **throw**.

## Finally

Para asegurarnos que algún código corra haya o no algún error se utiliza la palabra reservada **finally**.

# **Funciones**

## ¿Qué es una función?

Una **función** es un bloque de código que realiza cierta tarea, se le asigna un tipo de dato y un nombre para llamar a dicha función.

Una función debe de tener un tipo de dato el cual va a **retornar**, por defecto las funciones son del tipo void (null), es decir no retorna nada.

Al llamar a una función podemos enviar información, dicha información enviada recibe el nombre de **argumento**.La información recibida dentro de la función se le como **parámetro**.

## First-Class-Object

Las funciones pueden ser asignadas a variables o pasadas como argumentos a otras funciones.

## Función flecha

Si la función solo contiene una expresión se puede hacer una **función flecha** la cual retornará una sola expresión dependiendo el tipo de dato de la función.

## Función anónima

Si deseas enviar como parámetro una función y no quieres crear una nueva función, puedes enviar una **función anónima**.

## Parámetros requeridos y opcionales

Los parámetros que recibe una función pueden ser *requeridos* u *opcionales*.

Si los parámetros son **requeridos**:

Solo se pueden recibir de manera **posicional** es decir con cierto orden.

Si los parámetros son **opcionales**:

Se pueden recibir con **nombre** y al pasarlos como argumento. Habrá que especificar el nombre.

Se pueden recibir de manera *posicional*.

## Parámetros con valores default

Así mismo los parámetros pueden tener **valores default** los cuales pueden ser alterados si se lo indican como argumento.

## Función main

En Dart todas las aplicaciones deben de tener una función main. Aquí se ejecutará nuestra aplicación.

## Diferencia entre acceder a la función y crear una referencia.

Cuando llamamos a una función **con paréntesis** estamos llamando al contenido de la función para que se ejecute, pero cuando escribimos dicha función **sin paréntesis** estamos enviando una referencia de dicha función teniendo la oportunidad de asignar esa función a otra parte.

# **Clases**

## ¿Qué es una clase?

Una **clase** es una plantilla la cual dicta como estará compuesto un objeto y nos servirá para crear instancias. Una clase puede tener variables de instancia, métodos, entre otras cosas.

## Objetos

Un **objeto** es una instancia (copia) de la clase que se usará como plantilla.

*Se le conoce como* ***abstracción*** *al concepto de convertir cosas del mundo real a clases.*

## Variables de instancia

Se le llama **variables de instancia** a las variables y constantes que se declaren en una clase. Las variables de instancia pueden ser **privadas** agregando un (\_) antes del nombre del identificador, de tal manera que solo podemos acceder a ellas desde el archivo en el que estamos trabajando.

Al igual que con los tipos de datos, creamos referencias a dichas clases mediante el uso de variables o constantes. Una vez tengamos una referencia a la clase, podemos acceder a las variables de instancia y métodos utilizando (**.**).

Se pueden inicializar dichas variables de instancia con un **constructor**.

## Constructor

El constructor será llamado cada que creemos una instancia de dicha clase. Un constructor por defecto no recibe parámetros hasta que lo indiquemos.

El constructor se crea escribiendo el nombre de la clase e indicando que parámetros va a recibir. Solo puede existir un constructor default, si se desea agregar más constructores se tendrá que asignarles nombre.

## Constructor con nombre

En Dart los constructores pueden tener nombre escribiendo el nombre de la clase, un punto(.) y el nombre del identificador.

Al igual que las funciones los constructores pueden tener parámetros opcionales y requeridos.

## Initializer list

Se pueden inicializar las variables de instancia antes de que arranque el contenido el constructor utilizando (**:**) seguido de las variables de instancia. Esto es conocido como **initializer list**.

Se pueden validar dichas variables de instancia mediante el uso de asserts.

Se puede redirigir a otro constructor mediante el uso del operador (:).

## Factory constructors

Dart soporta **factory constructors** los cuales pueden retornar instancias de subclases o incluso un nulo si es necesario.

*Reciben el nombre de* ***métodos*** *las funciones u operaciones que contenga una clase.*

## Métodos

Un método es una función, pero de una clase en particular. Los métodos de una clase pueden acceder a las variables de instancia con la palabra reservada **this** seguida de un punto (**.**).

## Getter y Setter

**Get** y **set** son métodos especiales que nos permiten acceder a las propiedades de nuestro objeto.

Se va usar **get** para retornar el tipo de dato al que quieren acceder.

Utilizaremos **set** para modificar alguna propiedad de nuestro objeto.

## Herencia

Se utiliza la palabra reservada **extends**. Si una clase **a** extiende a **b,** todas las *propiedades*, *variables*, *métodos* implementados en **b** también estarán en **a**. Adicionalmente se puede sobrescribir las funciones de **b**.

En Dart las clases pueden extender a una sola clase.

Cuando se intenta utilizar un método o una variable de instancia que no existe se puede sobrescribir el método **noSuchMethod().**

*Todas las clases heredan de la clase Object, teniendo acceso a sus atributos y métodos.*

## Abstract

Se utiliza la palabra reservada **abstract** antes de class. Una clase abstracta no puede ser instanciada, pero puede ser heredada. Dentro de una clase abstracta se pueden definir métodos abstractos los cuales necesitan ser sobrescritos una vez extendamos de dicha clase abstracta.

## Interfaces

Se utiliza la palabra reservada **implements**. Si una clase **a** implementa a **b**, todas las propiedades, variables, métodos definidos en **b** deben de ser implementadas en **a**. Al implementar una clase es necesario sobrescribir todos sus métodos y variables de instancia.

## Mixins

Se utiliza la palabra reservada **with**. Los **mixins** nos sirven para para implementar propiedades sin la necesidad de heredar cierta clase. Se pueden crear como una clase normal, clase abstracta o con la palabra **mixin**. Los mixins no pueden heredar a otra clase.

Se puede implementar más de un mixin en donde se prioriza el orden en que se añaden, siendo el ultimo el más específico.

La palabra reservada **on** seguida de una clase en los mixins nos permite restringir a que si se va a implementar dicho mixin sea necesario que se extienda a la clase que se mencionó.

## Static

La palabra reservada **static** puede ser implementada en variables y métodos.

Las **variables estáticas** suelen ser utilizadas como constantes de toda la clase; no son inicializados hasta que son utilizados.

Los **métodos estáticos** NO operan en una instancia, no tienen acceso al this.

# **Generics**

La idea de los genéricos es permitir el uso de cualquier tipo de dato como parámetro para métodos, clases, interfaces, etc.

Los genéricos son de mucha ayuda para reducir el código duplicado.

Además, las clases pueden recibir un tipo de dato y trabajar con él para ser una clase genérica.

Para especificar uno o más tipos cuando se utiliza constructor se pone el tipo de dato entre **< >**.

# **Programación asíncrona**

Las operaciones asíncronas nos permiten trabajar en otras tareas en lo que esperamos a que otra operación termine.

*Pedir o enviar datos a algún api*, *leer datos de un archivo* son algunas operaciones asíncronas comunes.

## Future

Un **Future** representa el resultado de una operación asíncrona. Cuenta con dos estados *uncompleted* y *completed.*

Cuando se llama a una función asíncrona esta retorna un **uncompleted** future; el future estará esperando a que la operación asíncrona termine o tire un error.

Si la operación asíncrona tiene éxito, el **completed** future retornara un future con el valor dependiendo el tipo especificado. De otra forma, retornará un error.

Para definir una función asíncrona utilizaremos la palabra reservada **async** antes de las llaves.

Para esperar a un future hasta que pase al estado *completed* utilizamos la palabra reservada **await** antes de la operación asíncrona.

*Todas las funciones asíncronas serán síncronas hasta que encuentren el primer await.*

## Stream

Un **stream** es una secuencia de eventos asíncronos, como un iterable.

La diferencia entre *future* y *stream* es que el future va a retornar un solo objeto (**return**) mientras que el stream va a retornar más de un objeto (**yield**).

# **Comentarios**

Los comentarios sirven para documentar nuestro código y hacerlo más entendible, dichos comentarios van a ser ignorados por el compilador.

## Comentarios de una sola línea

Los comentarios de una sola línea comienzan con **//**. Todo entre y el fin de línea será ignorado por el compilador.

## Comentarios de múltiple línea

Los comentarios de múltiple línea comienzan con **/\*** y terminan con **/\***. Todo dentro de **/\*** y **\*/** será ignorado por el compilador.

## Comentarios de documentación

Los comentarios de documentación son comentarios de múltiple línea o una sola línea, los cuales comienzan con **///** o **/\*\***. Usando **///** en líneas consecutivas.

Dart ignora todo el texto del comentario excepto los corchetes. Mediante el uso de corchetes podemos referirnos a clases, métodos, variables, funciones, parámetros, etc.

Para transformar el código en Dart y generar una documentación en HTML, se puede utilizar el SDK’s de documentación.