Operaciones con cada tipo de dato visto en clase

```
art > main
import 'dart:math';

Run | Debug
void main(List<String> args) {
  print("bool");
  bool isTrue = true;
  bool isFalse = false;
  print(isTrue && isFalse); // false
  print(isFalse || isTrue); // true
  print(!isTrue); // false
  print(isFalse == isTrue); // false
  print(isFalse != isTrue); // true
```

```
C:\Users\GHMX\Desktop\Dart-Ops>dart main.dart
bool
false
true
false
false
true
```

```
print("int");
int a = 10;
int b = 5;
print(a + b); // 15
print(a - b); // 5
print(a * b); // 50
print(a / b); // 2.0
print(a ~/ b); // 2
print(a % b); // 0
print(-a); // -10
print(++a); // 11
print(--a); // 10
print(a++); // 10
print(a--); // 11
print(a & b); // 0
print(a | b); // 15
print(a ^ b); // 15
print(~a); // -11
print(a << 1); // 20
print(a >> 1); // 5
print(a == b); // false
print(a != b); // true
print(a > b); // true
print(a < b); // false
print(a >= b); // true
print(a <= b); // false
print(pow(a, b)); // 100000
print(sqrt(a)); // 3.1622776601683795
print(log(a)); // 2.302585092994046
print(sin(a)); // -0.5440211108893698
print(cos(a)); // -0.8390715290764524
print(tan(a)); // 0.6483608274590866
```

```
int
15
5
50
2.0
2
0
-10
11
10
10
11
0
15
15
-11
20
5
false
true
true
false
true
false
100000
3.1622776601683795
2.302585092994046
-0.5440211108893698
-0.8390715290764524
0.6483608274590866
```

```dart
print("double");
double c = 10.0;
double d = 5.0;
print(c + d); // 15.0
print(c - d); // 5.0
print(c * d); // 50.0
print(c / d); // 2.0
print(c ~/ d); // 2
print(c % d); // 0.0
print(-c); // -10.0
print(++c); // 11.0
print(--c); // 10.0
print(c++); // 10.0
print(c--); // 11.0
print(c == d); // false
print(c != d); // true
print(c > d); // true
print(c < d); // false
print(c >= d); // true
print(c <= d); // false
print(pow(c, d)); // 100000.0
print(sqrt(c)); // 3.1622776601683795
print(log(c)); // 2.302585092994046
print(sin(c)); // -0.5440211108893698
print(cos(c)); // -0.8390715290764524
print(tan(c)); // 0.6483608274590866
```

```
double
15.0
5.0
50.0
2.0
2
0.0
-10.0
11.0
10.0
10.0
11.0
false
true
true
false
true
false
100000.0
3.1622776601683795
2.302585092994046
-0.5440211108893698
-0.8390715290764524
0.6483608274590866
```

```dart
print("String");
String str = "Hello";
String str2 = " World ";
print(str + str2); // Hello World
print("$str $str2"); // Hello  World
print(str * 3); // HelloHelloHello
print(str[0]); // H
print(str == str2); // false
print(str != str2); // true
print(str.length); // 5
print(str.isEmpty); // false
print(str.isNotEmpty); // true
print(str.contains("H")); // true
print(str.startsWith("W")); // false
print(str.endsWith("o")); // true
print(str.indexOf("l")); // 2
print(str.lastIndexOf("l")); // 3
print(str.substring(0, 3)); // Hel
print(str.substring(3)); // lo
print(str.split("l")); // [He, , o]
print(str.replaceAll("l", "L")); // HeLLo
print(str.toUpperCase()); // HELLO
print(str.toLowerCase()); // hello
print(str2.trim()); // World
print(str2.trimLeft()); // World
print(str2.trimRight()); //  World
print(str.padLeft(10, "0")); // 00000Hello
print(str.padRight(10, "0")); // Hello00000
print(str.codeUnitAt(0)); // 72
```

```
String
Hello World
Hello  World
HelloHelloHello
H
false
true
5
false
true
true
false
true
2
3
Hel
lo
[He, , o]
HeLLo
HELLO
hello
World
World
 World
00000Hello
Hello00000
72
```

CRUD con listas y mapas

```dart
print("List");
List<int> list = [1, 2, 3, 4, 5];
List<int> list2 = [6, 7, 8, 9, 10];
print(list + list2); // [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(list.where((element) => element > 3)); // (4, 5)
list.forEach((element) => print(element)); // 1 2 3 4 5
print(list.reduce((value, element) => value + element)); // 15
print(list.length); // 5
print(list.isEmpty); // false
print(list.isNotEmpty); // true
print(list.contains(1)); // true
print(list.indexOf(1)); // 0
print(list.lastIndexOf(1)); // 0
print(list.sublist(0, 3)); // [1, 2, 3]
print(list.sublist(3)); // [4, 5]
print(list.reversed); // (5, 4, 3, 2, 1)
// CREATE
List<int> list3 = [11, 12, 13];
// READ
print(list3[0]); // 11
print(list3.elementAt(2)); // 13
```

```dart
// UPDATE
list3.add(14);
print(list3); // [11, 12, 13, 14]
list3.insert(0, 8);
print(list3); // [8, 11, 12, 13, 14]
list3.insertAll(1, [9, 10]);
print(list3); // [8, 9, 10, 11, 12, 13, 14]
list3.addAll([15, 16]);
print(list3); // [8, 9, 10, 11, 12, 13, 14, 15, 16]
list3.replaceRange(0, 2, [17, 18]);
print(list3); // [17, 18, 10, 11, 12, 13, 14, 15, 16]
// DELETE
list3.remove(17);
print(list3); // [18, 10, 11, 12, 13, 14, 15, 16]
list3.removeAt(0);
print(list3); // [10, 11, 12, 13, 14, 15, 16]
list3.removeLast();
print(list3); // [10, 11, 12, 13, 14, 15]
list3.removeRange(0, 2);
print(list3); // [12, 13, 14, 15]
list3.removeWhere((element) => element == 14);
print(list3); // [12, 13, 15]
list3.clear();
print(list3); // []
```

```
List
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
(4, 5)
1
2
3
4
5
15
5
false
true
true
0
0
[1, 2, 3]
[4, 5]
(5, 4, 3, 2, 1)
11
13
[11, 12, 13, 14]
[8, 11, 12, 13, 14]
[8, 9, 10, 11, 12, 13, 14]
[8, 9, 10, 11, 12, 13, 14, 15, 16]
[17, 18, 10, 11, 12, 13, 14, 15, 16]
[18, 10, 11, 12, 13, 14, 15, 16]
[10, 11, 12, 13, 14, 15, 16]
[10, 11, 12, 13, 14, 15]
[12, 13, 14, 15]
[12, 13, 15]
[]
```

```dart
print("Set");
Set<int> set = {1, 2, 3, 4, 5, 6};
Set<int> set2 = {6, 7, 8, 9, 10};
print(set.length); // 6
print(set.isEmpty); // false
print(set.isNotEmpty); // true
print(set.contains(1)); // true
set.forEach((element) => print(element)); // 1 2 3 4 5 6
print(set.where((element) => element > 3)); // (4, 5, 6)
print(set.any((element) => element > 3)); // true
print(set.every((element) => element > 3)); // false
print(set.firstWhere((element) => element > 3)); // 4
print(set.union(set2)); // {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(set.intersection(set2)); // {6}
print(set.difference(set2)); // {1, 2, 3, 4, 5}
print(set.lookup(1)); // 1
// CREATE
Set<int> set3 = {6, 7, 8, 9, 10};
// READ
print(set3.elementAt(1)); // 7
print(set3.first); // 6
print(set3.last); // 10
```

```dart
// UPDATE
set3.add(6);
print(set3); // {6, 7, 8, 9, 10}
set3.addAll({11, 12});
print(set3); // {6, 7, 8, 9, 10, 11, 12}
// DELETE
set3.remove(7);
print(set3); // {6, 8, 9, 10, 11, 12}
set3.removeWhere((element) => element == 12);
print(set3); // {6, 8, 9, 10, 11}
set3.clear();
print(set3); // {}
```

```
Set
6
false
true
true
1
2
3
4
5
6
(4, 5, 6)
true
false
4
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{6}
{1, 2, 3, 4, 5}
1
7
6
10
{6, 7, 8, 9, 10}
{6, 7, 8, 9, 10, 11, 12}
{6, 8, 9, 10, 11, 12}
{6, 8, 9, 10, 11}
{}
```

```dart
print("Map");
Map<String, int> map = {"one": 1, "two": 2, "three": 3};
print(map.length); // 3
print(map.isEmpty); // false
print(map.isNotEmpty); // true
print(map.containsKey("one")); // true
print(map.containsValue(3)); // true
map.forEach((key, value) => print("$key: $value")); // one: 1 two: 2 three: 3
// CREATE
Map<String, int> map2 = {"four": 4, "five": 5};
// READ
print(map2["four"]); // 4
print(map2.keys); // (four, five)
print(map2.values); // (4, 5)
print(map2.entries); // (MapEntry(four: 4), MapEntry(five: 5))
// UPDATE
map2["six"] = 6;
print(map2); // {four: 4, five: 5, six: 6}
map2.addAll({"seven": 7, "eight": 8});
print(map2); // {four: 4, five: 5, six: 6, seven: 7, eight: 8}
map2.update("four", (value) => 7);
print(map2); // {four: 7, five: 5, six: 6, seven: 7, eight: 8}
map2.updateAll((key, value) => value + 1);
print(map2); // {four: 8, five: 6, six: 7, seven: 8, eight: 9}
// DELETE
map2.remove("four");
print(map2); // {five: 5, six: 6, seven: 7, eight: 8}
map2.removeWhere((key, value) => key == "six");
print(map2); // {five: 5, seven: 7, eight: 8}
map2.clear();
print(map2); // {}
```

```
Map
3
false
true
true
true
one: 1
two: 2
three: 3
4
(four, five)
(4, 5)
(MapEntry(four: 4), MapEntry(five: 5))
{four: 4, five: 5, six: 6}
{four: 4, five: 5, six: 6, seven: 7, eight: 8}
{four: 7, five: 5, six: 6, seven: 7, eight: 8}
{four: 8, five: 6, six: 7, seven: 8, eight: 9}
{five: 6, six: 7, seven: 8, eight: 9}
{five: 6, seven: 8, eight: 9}
{}
```

Tabla de casting de datos

Tipo de dato original	Tipo de dato destino	Operación	Ejemplo
int	double	.toDouble()	int x = 10; double y = x.toDouble();
int	String	.toString()	int x = 10; String y = x.toString();
int	bool	No es posible realizar una conversión directa de booleanos a números.	int number = 42; bool booleanValue = (number != 0);
double	int	.toInt()	double x = 10.0; int y = x.toInt();
double	String	.toString()	double x = 10.0; String y = x.toString();
double	bool	No es posible realizar una conversión directa de booleanos a números.	double number = 42.0; bool booleanValue = (number != 0);
String	int	int.parse()	String x = "10"; int y = int.parse(x);
String	double	double.parse()	String x = "10.5"; double y = double.parse(x);
String	bool	No es posible realizar una conversión directa de String a bool.	String x = "true"; bool y = (x == "true");
dynamic	Tipo específico	as	dynamic x = 10; double y = x as double;

Es posible convertir entre diferentes tipos de colecciones, como Listas, Sets y Maps. Sin embargo, la conversión directa entre ellos puede requerir un poco de trabajo.

List	Set	Set.from(list)	List<int> list = [1, 2, 3, 4, 5]; Set<int> set = Set.from(list);
Set	List	toList()	Set<int> set = {1, 2, 3, 4, 5}; List<int> list = set.toList();
List	Map	map()	List<String> list = ['a', 'b', 'c', 'd', 'e']; Map<String, int> map =

			list.asMap().map((i, v) => MapEntry(v, i));
Map	List	entries.toList() o values.toList()	Map<String, int> map = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}; List<MapEntry<String, int>> list = map.entries.toList();
Map	Set	Se puede crear un Set a partir de las claves del Map	Map<String, int> map = {'a': 1, 'b': 2, 'c': 3}; Set<String> set = map.keys.toSet();

Set	Map	Se puede crear un Map con las claves del Set y luego asignar un valor a cada clave	Set<String> set = {'a', 'b', 'c'};
			Map<String, int> map = {};
			
			int i = 0;
			for (String s in set) {
			  map[s] = i;
			  i++;
			}

La propiedad runtimeType se utiliza para averiguar el tipo de tiempo de ejecución del objeto. Ejemplo: miVariable.runtimeType

Operadores de prueba de tipo: is (verdadero si el objeto es el tipo especificado), is! (falso si el objeto tiene el tipo especificado)

const vs final

const se usa para crear una constante que se evalúa en tiempo de compilación, es adecuado usar const cuando se desea una constante que sea compartida y accedida en todo el código, estas constantes deben tener un valor asignado en el momento de su declaración.

final se usa para crear una constante que se evalúa en tiempo de ejecución, es adecuado usar final cuando se desea una constante que solo se necesita en un ámbito local, y no es necesario compartirla con otras partes del código.