# ****🔹 Dart**** String ****Class – Complete Explanation 🚀****

The **String** class in Dart is **immutable**, meaning its value cannot be changed after creation. It provides **various methods and constructors** to manipulate and format text efficiently. Let’s explore **everything about the String class**, from **basic to industrial-level usage**.

# ****📌 1. How to Create a String?****

There are **four ways** to create a string in Dart:

### ✅ **1.1 Using String Literals (Recommended)**

void main() {

String name = 'Alice'; // ✅ Using single quotes

String city = "New York"; // ✅ Using double quotes

print(name); // Output: Alice

print(city); // Output: New York

}

### ✅ **1.2 Multiline Strings (Using Triple Quotes)**

void main() {

String description = '''This is a

multiline string.''';

print(description);

}

### ✅ **1.3 Using String Constructor (Less Common)**

void main() {

String message = String.fromCharCodes([72, 101, 108, 108, 111]);

print(message); // Output: Hello

}

# ****📌 2.**** String ****Class Constructors 🏗️****

Dart provides **four constructors** to create strings dynamically.

| **Constructor** | **Description** | **Example** |
| --- | --- | --- |
| **String.fromCharCode(int charCode)** | Creates a string from a single Unicode character. | String.fromCharCode(65) → "A" |
| **String.fromCharCodes(List<int> charCodes)** | Creates a string from multiple Unicode values. | String.fromCharCodes([72, 101, 108, 108, 111]) → "Hello" |
| **String.fromEnvironment(String name, {String defaultValue})** | Retrieves a compile-time environment variable. | String.fromEnvironment("APP\_MODE", defaultValue: "production") |

✅ **Example Usage of Constructors**:

void main() {

print(String.fromCharCode(97)); // Output: a

print(String.fromCharCodes([104, 101, 108, 108, 111])); // Output: hello

}

# ****📌 3. String Properties 📏****

| **Property** | **Description** | **Example** |
| --- | --- | --- |
| **length** | Returns the number of characters in the string. | "hello".length → 5 |
| **isEmpty** | Returns true if the string is empty. | "".isEmpty → true |
| **isNotEmpty** | Returns true if the string is **not** empty. | "Dart".isNotEmpty → true |

✅ **Example Usage**:

void main() {

String text = "Flutter";

print(text.length); // Output: 7

print(text.isEmpty); // Output: false

print(text.isNotEmpty); // Output: true

}

# ****📌 4. String Methods – Powerful Text Manipulation****

## ****🔹 4.1 String Case Conversions****

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| **toUpperCase()** | Converts to uppercase. | "dart".toUpperCase() → "DART" |
| **toLowerCase()** | Converts to lowercase. | "FLUTTER".toLowerCase() → "flutter" |

✅ **Example**:

void main() {

String text = "Hello Dart";

print(text.toUpperCase()); // Output: HELLO DART

print(text.toLowerCase()); // Output: hello dart

}

## ****🔹 4.2 String Trimming & Padding****

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| **trim()** | Removes whitespace from both sides. | " Dart ".trim() → "Dart" |
| **trimLeft()** | Removes whitespace from the left. | " Dart".trimLeft() → "Dart" |
| **trimRight()** | Removes whitespace from the right. | "Dart ".trimRight() → "Dart" |
| **padLeft()** | Adds padding to the left. | "7".padLeft(3, "0") → "007" |
| **padRight()** | Adds padding to the right. | "Hi".padRight(5, "!") → "Hi!!!" |

✅ **Example**:

void main() {

String name = " Dart ";

print(name.trim()); // Output: "Dart"

print(name.trimLeft()); // Output: "Dart "

print(name.trimRight()); // Output: " Dart"

print("7".padLeft(3, '0')); // Output: 007

}

## ****🔹 4.3 Substring & Splitting****

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| **substring(start, [end])** | Extracts a part of the string. | "Flutter".substring(0, 4) → "Flut" |
| **split(String pattern)** | Splits string into a list. | "apple,banana".split(",") → ["apple", "banana"] |

✅ **Example**:

void main() {

String sentence = "Dart Programming";

print(sentence.substring(0, 4)); // Output: Dart

print("one,two,three".split(",")); // Output: [one, two, three]

}

## ****🔹 4.4 String Replacement****

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| **replaceAll(from, to)** | Replaces all occurrences of a substring. | "abc abc".replaceAll("a", "X") → "Xbc Xbc" |
| **replaceFirst(from, to, [start])** | Replaces first occurrence. | "abc abc".replaceFirst("a", "X") → "Xbc abc" |

✅ **Example**:

void main() {

String text = "I love Dart!";

print(text.replaceAll("Dart", "Flutter")); // Output: I love Flutter!

}

# ****📌 5. String Interpolation (Embedding Variables)****

✅ **Example**:

void main() {

String name = "Alice";

int age = 25;

print("My name is $name and I am $age years old.");

}

**Output:**

My name is Alice and I am 25 years old.

# ****📌 6. Regular Expressions (RegExp)****

Dart supports **regex** for advanced string matching.

✅ **Example**:

void main() {

RegExp regex = RegExp(r'^[a-zA-Z]+$');

print(regex.hasMatch("Hello")); // Output: true

print(regex.hasMatch("Hello123")); // Output: false

}

# ****🔥 Industrial-Level Usage****

✅ **Validate Email with Regex**:

bool isValidEmail(String email) {

RegExp regex = RegExp(r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$');

return regex.hasMatch(email);

}

void main() {

print(isValidEmail("test@example.com")); // Output: true

}

# ****🎯 Final Thoughts****

* ✅ String is **immutable** in Dart.
* ✅ Multiple constructors exist (fromCharCode(), fromCharCodes(), fromEnvironment()).
* ✅ **Methods** allow powerful text processing (trimming, splitting, replacing).
* ✅ **Regular expressions (RegExp)** allow pattern matching.

Would you like **advanced real-world applications** using String? 🚀🔥

Got it! Here's a **well-structured and visually appealing** explanation of **all String constructors in Dart** with **clear formatting and examples**. 🚀

**🔹 Dart String Class – All Constructors**

In Dart, the String class provides multiple ways to create and manipulate strings. Below are **all the constructors** available in Dart for String.

**📝 Summary Table – Dart String Constructors**

**1️⃣ Default String Creation (Literals)**

📌 **Note:** Dart does not provide an explicit String() constructor.

✅ **Example:**

void main() {

String s1 = "Hello, Dart!"; // Using double quotes

String s2 = 'Welcome!'; // Using single quotes

print(s1); // Output: Hello, Dart!

print(s2); // Output: Welcome!

}

✅ **Multi-line Strings:**  
Use ''' or """ for multi-line strings.

void main() {

String multiline = '''This is

a multi-line

string.''';

print(multiline);

}

**2️⃣ String.fromCharCodes() – Create String from Unicode**

This constructor **creates a string from a list of Unicode values**.

✅ **Example:**

void main() {

List<int> unicodeValues = [72, 101, 108, 108, 111]; // Unicode for "Hello"

String result = String.fromCharCodes(unicodeValues);

print(result); // Output: Hello

}

**3️⃣ String.fromCharCode() – Create Single Character from Unicode**

This constructor **creates a string with a single character from a Unicode value**.

✅ **Example:**

void main() {

String char = String.fromCharCode(65); // Unicode 65 → 'A'

print(char); // Output: A

}

**4️⃣ String.fromEnvironment() – Get String from Environment Variables**

This constructor retrieves **compile-time environment variables**.

⚠️ **Note:** This works only in **Dart compiled environments**, not in Flutter.

✅ **Example:**

void main() {

const String apiKey = String.fromEnvironment("API\_KEY", defaultValue: "No API Key Found");

print(apiKey); // Output: No API Key Found (unless environment variable exists)

}

| **Constructor** | **Description** | **Example** |
| --- | --- | --- |
| **Default String ("", '')** | Creates a normal string | "Hello" |
| **Multi-line String (''' or """)** | Creates multi-line strings | '''Hello\nWorld''' |
| **String.fromCharCodes()** | Converts Unicode list to string | String.fromCharCodes([72,101,108]) → "Hel" |
| **String.fromCharCode()** | Converts a single Unicode to string | String.fromCharCode(65) → "A" |
| **String.fromEnvironment()** | Retrieves environment variables (compile-time) | String.fromEnvironment("API\_KEY") |

**🎯 Final Thoughts**

* Use **normal string literals ("", '')** for regular string creation.
* Use **multi-line strings (''' or """)** when handling long text.
* Use **fromCharCodes()** and **fromCharCode()** for working with Unicode characters.
* Use **fromEnvironment()** for retrieving environment variables **(only in compiled Dart programs)**.

Would you like more **advanced use cases** or **real-world applications**? Let me know! 🚀🔥

============================================================================================================

**🔹 Dart int Class – All Constructors**

In Dart, the int class provides multiple ways to create integers. Below are **all constructors** for int with clear explanations and examples. 🚀

**1️⃣ Direct Integer Assignment (Literals)**

Dart allows direct assignment of integers using **literals**.

📌 **Note:** There is no explicit int() constructor for normal integer assignment.

✅ **Example:**

void main() {

int num1 = 100; // ✅ Direct integer assignment

int num2 = -25; // ✅ Negative integer

print(num1); // Output: 100

print(num2); // Output: -25

}

**2️⃣ int.parse() – Convert String to Integer**

This constructor **converts a string to an integer**.

⚠️ **Note:** If the string contains invalid numbers, it throws a FormatException.

✅ **Example:**

void main() {

int num1 = int.parse("123"); // ✅ Converts string to int

int num2 = int.parse("-456"); // ✅ Works with negative numbers

print(num1); // Output: 123

print(num2); // Output: -456

}

🚨 **Handling Invalid Strings with Try-Catch:**

void main() {

try {

int num = int.parse("ABC"); // ❌ Error: Invalid number

print(num);

} catch (e) {

print("Invalid input! Error: $e"); // Output: Invalid input! Error: FormatException

}

}

**3️⃣ int.tryParse() – Safe String to Integer Conversion**

Unlike int.parse(), this method **returns null instead of throwing an error** if the input is invalid.

✅ **Example:**

void main() {

int? num1 = int.tryParse("123"); // ✅ Returns 123

int? num2 = int.tryParse("XYZ"); // ❌ Invalid input → returns null

print(num1); // Output: 123

print(num2); // Output: null (safe handling)

}

**4️⃣ int.fromEnvironment() – Get Integer from Environment Variables**

This constructor retrieves **compile-time environment variables** for integers.

⚠️ **Note:** Works only in **compiled Dart programs**, not in Flutter.

✅ **Example:**

void main() {

const int apiVersion = int.fromEnvironment("API\_VERSION", defaultValue: 1);

print(apiVersion); // Output: 1 (unless an environment variable exists)

}

# ****📝 Summary Table – Dart**** int ****Constructors****

| **Constructor** | **Description** | **Example** |
| --- | --- | --- |
| **Direct Assignment (int num = 10;)** | Assigns an integer directly | int x = 100; |
| **int.parse(String)** | Converts a string to an integer | int.parse("123") → 123 |
| **int.tryParse(String)** | Converts a string to an integer safely (returns null if invalid) | int.tryParse("ABC") → null |
| **int.fromEnvironment(String)** | Retrieves an integer from environment variables (compile-time) | int.fromEnvironment("API\_VERSION") |

**🎯 Final Thoughts**

* Use **direct assignment (int x = 10;)** for regular integer values.
* Use **int.parse()** when converting valid strings to integers.
* Use **int.tryParse()** to avoid errors when parsing invalid numbers.
* Use **int.fromEnvironment()** for environment-based configurations.

Would you like more **advanced integer operations** or **real-world applications**? Let me know! 🚀🔥

**🎯 Final Thoughts**

* Use **direct assignment (int x = 10;)** for regular integer values.
* Use **int.parse()** when converting valid strings to integers.
* Use **int.tryParse()** to avoid errors when parsing invalid numbers.
* Use **int.fromEnvironment()** for environment-based configurations.

**==================================================================================================================================**

**🔹 Dart bool Class – All Constructors**

The bool class in Dart represents **Boolean values** (true or false). Unlike other types like int or String, bool has **limited constructors** since it only holds two possible values.

**1️⃣ Direct Boolean Assignment (Literals)**

Dart allows assigning true or false directly to a boolean variable.

📌 **Note:** Dart does **not** have an explicit bool() constructor for creating boolean values.

✅ **Example:**

void main() {

bool isActive = true; // ✅ Direct assignment

bool isCompleted = false;

print(isActive); // Output: true

print(isCompleted); // Output: false

}

**2️⃣ bool.fromEnvironment() – Get Boolean from Environment Variables**

This constructor retrieves **Boolean values from environment variables** at **compile-time**.

⚠️ **Note:** Works only in **Dart compiled programs**, not in Flutter.

✅ **Example:**

void main() {

const bool isDebug = bool.fromEnvironment("DEBUG\_MODE", defaultValue: false);

print(isDebug); // Output: false (unless an environment variable exists)

}

**📝 Summary Table – Dart bool Constructors**

| **Constructor** | **Description** | **Example** |
| --- | --- | --- |
| **Direct Assignment (bool x = true;)** | Assigns a boolean value directly | bool isReady = true; |
| **bool.fromEnvironment(String)** | Retrieves a boolean value from environment variables | bool.fromEnvironment("DEBUG\_MODE") |

**🎯 Final Thoughts**

* Use **direct assignment (bool x = true;)** for general boolean variables.
* Use **bool.fromEnvironment()** for compile-time configuration settings.
* **Dart does not have bool.parse() or bool.tryParse()** like int or double.

**==================================================================================================================================**

**🔹 Dart double Class – All Constructors**

The double class in Dart represents **floating-point numbers** (decimal values). It provides **multiple constructors** for creating and converting numbers. Below are all the available constructors with explanations and examples. 🚀

**1️⃣ Direct Double Assignment (Literals)**

Dart allows direct assignment of decimal numbers.

📌 **Note:** Dart does **not** have an explicit double() constructor for normal assignments.

✅ **Example:**

void main() {

double pi = 3.14; // ✅ Direct decimal value

double negative = -2.5; // ✅ Negative values are allowed

print(pi); // Output: 3.14

print(negative); // Output: -2.5

}

**2️⃣ double.parse() – Convert String to Double**

This constructor **converts a numeric string to a double**.

⚠️ **Note:** If the string is invalid, it throws a FormatException.

✅ **Example:**

void main() {

double num1 = double.parse("45.67"); // ✅ Converts string to double

double num2 = double.parse("-89.99"); // ✅ Works with negative numbers

print(num1); // Output: 45.67

print(num2); // Output: -89.99

}

🚨 **Handling Invalid Strings with Try-Catch:**

void main() {

try {

double num = double.parse("ABC"); // ❌ Error: Invalid number

print(num);

} catch (e) {

print("Invalid input! Error: $e"); // Output: Invalid input! Error: FormatException

}

}

**3️⃣ double.tryParse() – Safe String to Double Conversion**

Unlike double.parse(), this method **returns null instead of throwing an error** if the input is invalid.

✅ **Example:**

void main() {

double? num1 = double.tryParse("123.45"); // ✅ Returns 123.45

double? num2 = double.tryParse("XYZ"); // ❌ Invalid input → returns null

print(num1); // Output: 123.45

print(num2); // Output: null (safe handling)

}

**4️⃣ double.fromEnvironment() – Get Double from Environment Variables**

This constructor retrieves **compile-time environment variables** for floating-point numbers.

⚠️ **Note:** Works only in **compiled Dart programs**, not in Flutter.

✅ **Example:**

void main() {

const double taxRate = double.fromEnvironment("TAX\_RATE", defaultValue: 5.5);

print(taxRate); // Output: 5.5 (unless an environment variable exists)

}

**📝 Summary Table – Dart double Constructors**

| **Constructor** | **Description** | **Example** |
| --- | --- | --- |
| **Direct Assignment (double x = 3.14;)** | Assigns a floating-point number directly | double pi = 3.14; |
| **double.parse(String)** | Converts a string to a double | double.parse("3.14") → 3.14 |
| **double.tryParse(String)** | Converts a string to a double safely (returns null if invalid) | double.tryParse("XYZ") → null |
| **double.fromEnvironment(String)** | Retrieves a double from environment variables (compile-time) | double.fromEnvironment("TAX\_RATE") |

**🎯 Final Thoughts**

* Use **direct assignment (double x = 2.5;)** for regular decimal values.
* Use **double.parse()** when converting valid strings to floating-point numbers.
* Use **double.tryParse()** to avoid errors when parsing invalid numbers.
* Use **double.fromEnvironment()** for environment-based configurations **(only in compiled Dart programs)**.

Would you like **advanced examples** or **real-world applications** for double in Dart? 🚀🔥

**🔹 Dart List Class – All Constructors**

In Dart, a List is an **ordered collection** that can store multiple values of the same or different types. The List class provides multiple constructors for creating lists in different ways. Below are **all the List constructors** with explanations and examples. 🚀

**1️⃣ List Literal (Recommended)**

Dart allows creating a list using square brackets []. This is the most commonly used method.

✅ **Example:**

void main() {

List<int> numbers = [1, 2, 3, 4, 5]; // ✅ List of integers

List<String> names = ["Alice", "Bob", "Charlie"]; // ✅ List of strings

print(numbers); // Output: [1, 2, 3, 4, 5]

print(names); // Output: [Alice, Bob, Charlie]

}

**2️⃣ List() (Deprecated)**

Previously, Dart allowed creating lists using List(), but this is now **deprecated**.

🚨 **Don't use this:**

void main() {

// List<int> numbers = List(); // ❌ Error: This is not allowed anymore

}

**3️⃣ List.filled() – Create a Fixed-Length List**

This constructor **creates a list with a fixed length** and fills it with an initial value.

✅ **Example:**

void main() {

List<int> fixedList = List.filled(5, 0); // Creates a list of 5 elements, all initialized to 0

print(fixedList); // Output: [0, 0, 0, 0, 0]

}

🚀 **Making it Growable:**

void main() {

List<int> growableList = List.filled(3, 1, growable: true);

growableList.add(5);

print(growableList); // Output: [1, 1, 1, 5]

}

**4️⃣ List.generate() – Create a List with Dynamic Values**

This constructor generates a list dynamically using a function.

✅ **Example:**

void main() {

List<int> squares = List.generate(5, (index) => index \* index);

print(squares); // Output: [0, 1, 4, 9, 16]

}

**5️⃣ List.unmodifiable() – Create an Immutable List**

This constructor creates a **read-only (immutable) list** that cannot be modified.

✅ **Example:**

void main() {

List<String> readOnlyList = List.unmodifiable(["A", "B", "C"]);

print(readOnlyList); // Output: [A, B, C]

// readOnlyList.add("D"); // ❌ Error: Cannot modify unmodifiable list

}

**6️⃣ List.empty() – Create an Empty List**

This constructor creates an empty list, either **fixed-length** or **growable**.

✅ **Growable List (Recommended):**

void main() {

List<int> emptyList = List.empty(growable: true);

emptyList.add(10);

print(emptyList); // Output: [10]

}

✅ **Fixed-Length Empty List (Not Modifiable):**

void main() {

List<int> fixedEmptyList = List.empty(); // ❌ Cannot add elements

}

**📝 Summary Table – Dart List Constructors**

| **Constructor** | **Description** | **Example** |
| --- | --- | --- |
| **List Literal ([])** | Creates a simple list | List<int> nums = [1, 2, 3]; |
| **List.filled(int length, value)** | Creates a fixed-length list | List.filled(5, 0) → [0, 0, 0, 0, 0] |
| **List.generate(int length, (index) => value)** | Creates a list dynamically | List.generate(5, (i) => i \* 2) → [0, 2, 4, 6, 8] |
| **List.unmodifiable([...])** | Creates a read-only list | List.unmodifiable(["A", "B"]) |
| **List.empty(growable: true)** | Creates an empty growable list | List.empty(growable: true) |
| **List.empty()** | Creates an empty fixed-length list (not growable) | List.empty() |

**🎯 Final Thoughts**

* Use **list literals ([])** for most cases.
* Use **List.filled()** when you need a fixed-length list with default values.
* Use **List.generate()** for **dynamic list creation**.
* Use **List.unmodifiable()** when you need a **read-only list**.
* Use **List.empty(growable: true)** for an **empty list that can grow**.

Would you like **advanced list operations** or **real-world examples**? 🚀🔥