Variables in Dart Programming

1️⃣ Introduction to Variables

What is a Variable?

A **variable** is a named location in memory used to store data that can be used and changed during program execution.

```
int age = 25;
```

Here, `age` is a variable that stores the value `25`.

Why Variables Are Needed in Programs

Variables are needed because programs must:

- Store user input
- Perform calculations
- Remember values for later use
- Make programs dynamic and interactive

Without variables, a program would always produce the same output.

Variable as Memory Storage

You can think of a variable as a **box in computer memory**:

- Variable name → Label on the box
- Variable value → Item inside the box
- Data type → Type of box (what kind of value it can store)

```
String name = "Asthica";
```

The box labeled `name` stores the text "Asthica".

3️⃣ Dart Data Types (Basics Only)

Why Dart Needs Data Types

Dart uses data types to understand:

- What kind of value is being stored
- How much memory is required
- What operations are allowed on the value

Using data types helps Dart catch errors **early** and makes programs safer and faster.

Commonly Used Data Types

🔹 `int`

Used to store **whole numbers**.

```
int age = 25;
int year = 2026;
```

🔹 `double`

Used to store **decimal numbers**.

```
double price = 99.99;
double temperature = 36.5;
```

🔹 `String`

Used to store **text or characters**.

```
String name = "Asthica";
String city = "Delhi";
```

🔹 `bool`

Used to store **true or false** values.

```dart
bool isLoggedIn = true;
bool isCompleted = false;
```

Type Safety Concept

Dart is a **type-safe language**, which means:

- A variable can store only the type of value it was declared with
- Wrong type assignments are caught at compile time

❌Wrong:

```dart
int count = "ten";
```

✅Correct:

```dart
int count = 10;
```

Type safety helps prevent bugs and unexpected program crashes.

---

4. Declaring Variables

4.1 Using Explicit Data Types

You can declare a variable by specifying its data type.

```dart
String city = "Delhi";
```

4.2 Using `var`

When you use `var`, Dart automatically infers the type based on the assigned value.

```dart
var year = 2025; // int
var country = "India"; // String
```

> Once assigned, the type cannot be changed.

Allowed Characters in Variable Names

In Dart, variable names:

- Must start with a **letter (a–z, A–Z)** or underscore `_`
- Can contain **letters, numbers, and underscores**
- **Cannot start with a number**
- **Cannot contain spaces**
- **Cannot use Dart keywords** (like `int`, `class`, `if`)

✅Valid examples:

```
int age;
String user_name;
var total1;
```

❌Invalid examples:

```
int 1age;
String user name;
var class;
```

---

camelCase Convention

Dart follows the **camelCase** naming convention for variables.

Rule:

- First word starts with a lowercase letter
- Each new word starts with an uppercase letter

✅Examples:

```
String userAge;
double totalPrice;
bool isLoggedIn;
```

Use Meaningful Variable Names

Variable names should clearly describe **what data they store**.

❌Poor naming:

```
int x1;
String abc;
```

✅Meaningful naming:

```
int userAge;
String productName;
double totalPrice;
```

Meaningful names make your code easier to understand for **you and others**.

---

5. Variable Naming Rules & Conventions

Rules

- Must start with a letter or `_`
- Cannot start with a number
- No spaces allowed
- Cannot use keywords

Best Practices

- Use meaningful names
- Follow camelCase

❌`x1`, `data123` ✅`userAge`, `totalPrice`

6️⃣ `final` and `const` (Very Important)

In Dart, `final` and `const` are used to create **constants**—variables whose values cannot be changed once assigned.

Understanding the difference between them is essential for writing **efficient and safe Dart code**.

🔹 `final` → Runtime Constant

A `final` variable:

- Can be assigned **only once**

- Gets its value **at runtime**
- Is useful when the value is not known at compile time

```
final currentTime = DateTime.now();
final String country = "India";
```

Once assigned, the value of a `final` variable **cannot be changed**.

🔹 `const` → Compile-Time Constant

A `const` variable:

- Is assigned **at compile time**
- Must have a value that is known **before the program runs**
- Is deeply immutable

```
const pi = 3.14;
const String appName = "Dart Basics";
```

Key Difference Between `final` and `const`

`final`	`const`
Runtime constant	Compile-time constant
Assigned once	Assigned once
Value decided during execution	Value decided before execution
Value decided during execution	Value decided before execution

---

❌Common Mistake

```
const date = DateTime.now(); // ❌ Error
```

✅Correct Usage

```
final date = DateTime.now(); // ✅ Correct
```

7️⃣ Null Safety in Dart (Must Learn)

Null safety is one of the most important features of Dart. It helps prevent **app crashes caused by null values**.

Why Dart Avoids Null Errors

In many programming languages, accessing a `null` value causes runtime crashes such as:

- App suddenly stopping
- Blank screens in mobile apps
- Unexpected errors in production

Dart avoids these problems by **forcing developers to handle null values consciously**.

Non-nullable Variables

By default, variables in Dart **cannot be null**.

```
String name = "Dart";
// name = null; ❌ Error
```

This ensures the variable **always has a valid value**, making the program safer.

Nullable Variables Using `?`

If a variable **can be empty or missing**, you must explicitly allow null using `?`.

```
String? nickname;
nickname = null; // ✅ Allowed
```

This tells Dart:

> "I know this value may be null, and I will handle it safely."

Real-Life Crash Example (Without Null Safety)

Imagine a user profile app where the user has not entered a nickname.

❌Unsafe code:

```
String? nickname;
print(nickname.length); // ❌ Crash
```

The app crashes because Dart tries to access `.length` on a null value.

Safe Code With Null Safety

```
String? nickname;
print(nickname?.length); // ✅ No crash
```

Or using a default value:

```
print(nickname ?? "No nickname");
```

Key Takeaway

- Non-nullable variables are **safe by default**
- Nullable variables must be **handled carefully**
- Null safety prevents **real-world app crashes**

8️⃣ Null-Aware Operators (Intro Level)

Null-aware operators help you **work safely with nullable variables** and avoid runtime crashes caused by `null` values.

🔹 `?.` Safe Access Operator

Used to access a property or method **only if the value is not null**.

```
String? name;
print(name?.length); // Output: null (no crash)
```

If `name` is null, Dart safely stops execution instead of crashing.

🔹 `??` Default Value Operator

Used to provide a **fallback value** when a variable is null.

```
String? city;
print(city ?? "Unknown City");
```

If `city` is null, Dart prints `"Unknown City"`.

🔹 `??=` Null-Aware Assignment Operator

Assigns a value **only if the variable is currently null**.

```
String? country;
country ??= "India";
print(country);
```

If `country` already has a value, it will not be overwritten.

🔹 `!` Null Assertion Operator (⚠️Use with Warning)

Tells Dart that you are **100% sure** the value is not null.

```
String? name = "Asthica";
print(name!.length);
```

⚠️**Danger:** If the value is actually null, the app will crash.

```
String? name;
print(name!.length); // ❌ Runtime error
```

Use `!` only when you are absolutely certain the value is not null.

Key Takeaway

- Prefer `?.` and `??` for safety
- Use `??=` for lazy initialization
- Avoid `!` unless truly necessary

---

9. `late` Keyword

Used when a variable will be initialized later but before use.

```
late String title;
title = "Dart Variables";
print(title);
```

⚠️Accessing a `late` variable before initialization causes a runtime error.

---

10. Dynamic Variables

Allows changing data types at runtime.

```
dynamic value = 10;
value = "Ten";
value = true;
```

⚠️Avoid overuse to maintain type safety.

1️⃣1️⃣Variable Scope (Beginner Level)

Variable scope defines **where a variable can be accessed** and **how long it lives in memory**.

Understanding scope helps avoid bugs and unexpected behavior in programs.

🔹 Local Variables

Local variables are declared **inside a function or a block**.

- Accessible only within that function or block
- Created when the function starts
- Destroyed when the function ends

```
void main() {
  int count = 5; // local variable
  print(count);
}
```

❌This will cause an error:

```
void main() {
  int count = 5;
}

print(count); // Error: count is not accessible here
```

🔹 Global Variables

Global variables are declared **outside all functions**.

- Accessible throughout the program
- Created when the program starts
- Exist until the program ends

```
int total = 100; // global variable

void main() {
  print(total);
}
```

Lifetime & Accessibility (Quick Summary)

Variable Type	Where Declared	Accessibility	Lifetime
Local	Inside function/block	Only inside that block	Until block ends
Global	Outside functions	Entire program	Until program ends

---

12. Common Mistakes

- Using variables without initialization
- Misusing `dynamic`
- Ignoring null safety rules
- Overusing nullable variables

---

13. End of Topic: Coding Practice

1. Declare variables to store your name, age, and city.
2. Create a `final` variable and explain why it cannot change.
3. Write a program using a nullable variable with a default value.
4. Identify and fix the error:

```
int value;
print(value);
```

1. Write a program using `var`, `final`, and `String?` in one file.