Null saftey:

Null safety is a feature introduced in Dart 2.6 that helps prevent null-related errors at compile time. It enforces the types of variables and expressions, ensuring that you can't accidentally access a variable that might be null. This leads to more robust and predictable code.

1. Non-nullable types:

- By default, variables are non-nullable, meaning they cannot hold null values. This is enforced by specifying the type explicitly:

Example:

```
String name = "John"; // Name cannot be null
int age = 30; // Age cannot be null
```

2. Nullable types:

- You can explicitly mark a type as nullable using the **?** operator. This allows the variable to hold either a value of the specified type or null:

Example:

```
String? nickname; // Nickname can be null or a String
nickname = "Jack"; // Ok, setting a String value
nickname = null; // Ok, setting null
```

Null-aware operators

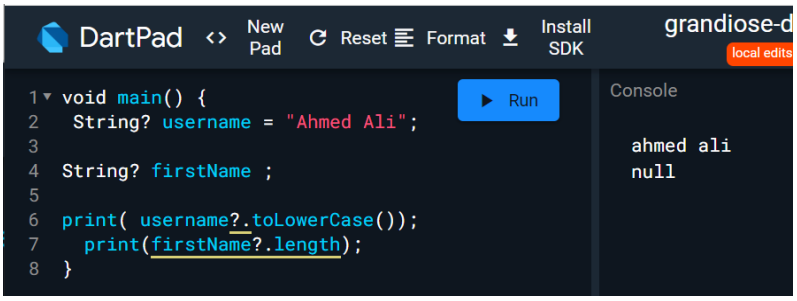
Null-aware operators are your best friends in Dart when dealing with nullable types and preventing null-related errors. They offer concise and safe ways to access properties, call methods, and perform operations on nullable values. Here's a breakdown of the most common ones:

Operator	**Meaning**
??	If-null operator
??=	Null-aware assignment operator
?.	Null-aware access & method invocation operator
!	Null assertion operator
?..	Null-aware cascade operator

`?[]`	Null-aware index operator
`...?`	Null-aware spread operator

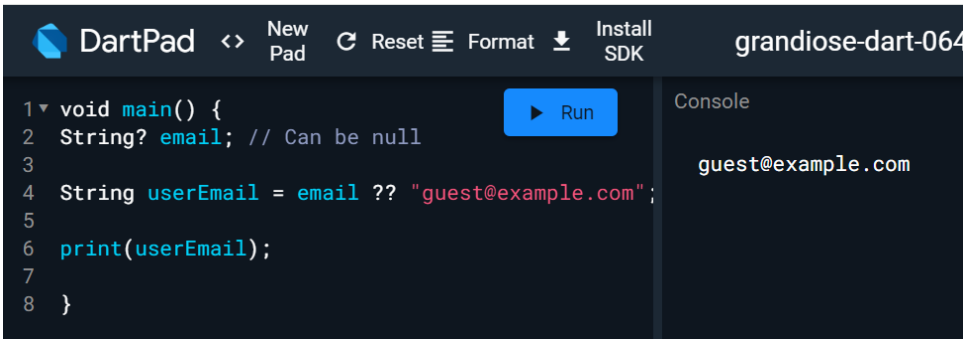
1. ?.(null-safe access operator):

This operator safely accesses properties of a potentially null object. If the object is null, it returns null instead of throwing an error.



2. Null-coalescing operator (??):

This operator provides a default value if the left operand is null. It's useful for setting default values in case of null values.

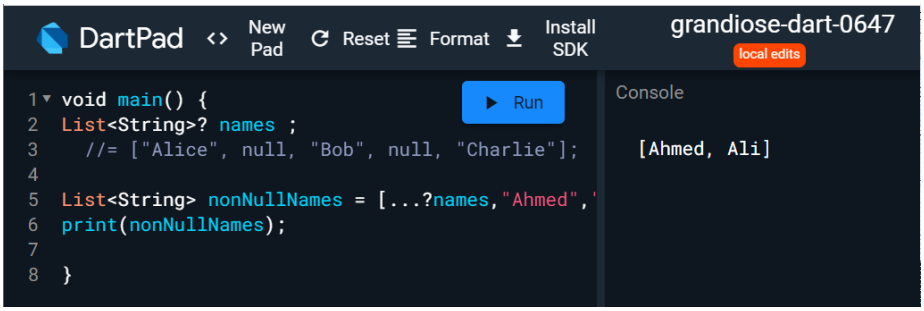


3. Null-aware assignment operator (??=):

This operator assigns a value to a variable only if it's null. It's handy for initializing variables based on conditions.

4. Null-aware spread operator (...?)

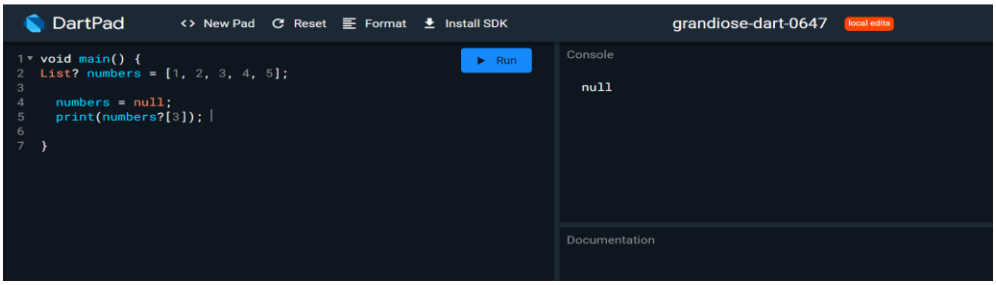
This operator safely spreads elements from an iterable that might contain null values. It skips null elements and only spreads non-null ones.



5. Null-aware cascade operator (?..)

6-Null-aware index operator(?[])

The null-aware index operator ?[] allows you to access an element of a list when the list might be null



7- Null assertion operator (!)

The ! operator tells the compiler that you are sure a variable or expression is not null and allows you to access its properties and methods directly, bypassing null checks.