**🧩 What is a Reactive Form?**

Think of a **form** like the input boxes on a website — for example:  
👉 Name, Email, Address, etc.

Now, there are **two ways** to create forms in Angular:

1. **Template-driven forms** – created mostly in the HTML file.
2. **Reactive forms** – created mostly in the **TypeScript (component)** file using **code**.

So, **Reactive Form = form made using TypeScript code**  
It gives us **more control** and makes **validation and logic easier**.

## Why it’s called “Reactive”?

Because it **reacts automatically** when a user types something or changes any input.  
For example:  
If a user types a wrong email → The form will show an error immediately.

Angular gives 3 main classes to create and manage forms:

### 1 FormControl

Represents **one single input field** (like Name, Email, etc.)

name = new FormControl('', Validators.required);

Explanation:

* FormControl → creates one control.
* '' → the initial value (empty here).
* Validators.required → means this field must not be empty.

**FormControl can:**

* Store the current value.
* Check if it’s valid or not.
* Watch for any changes.

💬 Example in real life:  
Imagine a single textbox for “Name”. FormControl will **watch** what the user types and check if it’s valid.

### 2 FormGroup

Used to **group multiple FormControls together** — like combining **Name, Email, and Password** into one form.

**Example:**

addressForm = new FormGroup({

street: new FormControl(),

city: new FormControl(),

state: new FormControl()

});

Explanation:

* FormGroup → acts like a **container** for multiple FormControls.
* Inside {}, we define controls like street, city, and state.

💬 Example in real life:  
A form section for “Address” — where you have **street**, **city**, and **state** fields.

### 3 FormArray

Used when you need **multiple controls of the same type** — for example, a list of hobbies, skills, phone numbers, etc.

hobbies = new FormArray([

new FormControl('reading'),

new FormControl('coding')

]);

📝 Explanation:

* FormArray → is a list (array) of FormControls or FormGroups.
* You can **add, remove, or reorder** controls as needed.

💬 Example in real life:  
A person may have **many hobbies** — so you can add more input boxes dynamically.

| **Concept** | **What it Represents** | **Example** | **Real-life Example** |
| --- | --- | --- | --- |
| **FormControl** | Single input field | new FormControl('') | One Name textbox |
| **FormGroup** | Group of FormControls | new FormGroup({...}) | Full Address section |
| **FormArray** | List of controls | new FormArray([...]) | Hobbies list (add more) |

## ✅ Key Points to Remember

* Reactive forms are **created in the component file** (not in HTML).
* They are **model-driven** — meaning the form structure is written as **code**.
* They are **powerful** for handling **validation, changes, and errors** easily.

## 🧩 What are Validators?

**Validators** are like **rules** for your form fields.  
They **check** if the user’s input is **correct or not**.

👉 Example:  
If the Name box is empty → it shows an error.  
If the Email is not in proper format → it shows an error.

In short:  
🧠 **Validator = rule to check user input**

## ✅ 1. Built-in Validators in Angular

Angular already gives us **ready-made validators** for common checks.

Here are the most used ones 👇

| **Validator** | **What it does** | **Example** |
| --- | --- | --- |
| **required** | Field must not be empty | Validators.required |
| **minLength** | Must have at least given number of characters | Validators.minLength(3) |
| **maxLength** | Must not exceed given number of characters | Validators.maxLength(10) |
| **pattern** | Must match a pattern (like only letters, or phone format) | Validators.pattern('[a-zA-Z ]\*') |
| **email** | Checks if it’s a valid email address | Validators.email |
| **min** | Number must be greater than or equal to this value | Validators.min(18) |
| **max** | Number must be less than or equal to this value | Validators.max(60) |

Example with multiple validators:

studentForm = new FormGroup({

name: new FormControl('', [Validators.required, Validators.minLength(3)]),

email: new FormControl('', [Validators.required, Validators.email]),

age: new FormControl('', [Validators.min(18), Validators.max(60)])

});

## 2. Custom Validators

Sometimes, you need **your own rule** — not covered by built-in ones.

For example:  
👉 A password must contain the word “@123”.

You can **create your own validator function**.

**Steps:**

1. Create a function that takes a control (input) as argument.
2. Inside, check if the input value meets your rule.

Example :

import { AbstractControl } from '@angular/forms';

function passwordValidator(control: AbstractControl) {

const value = control.value;

if (value && !value.includes('@123')) {

return { invalidPassword: true }; // ❌ invalid

}

return null; // ✅ valid

}

Then use it like this:

password = new FormControl('', [passwordValidator]);