# 📘 ****Django Form Validators****

# ****Code 32****

# ****Video 38****

## ✅ 1. What are Validators?

* A **validator** is a function or method that checks whether the input provided by the user is **valid** or not.
* Validators help ensure **data integrity** (valid data is saved/processed).
* If the input is invalid, Django will:
  + raise a **ValidationError**
  + show an error message in the form.

## ✅ 2. Types of Validators in Django

There are two main types:

### 🔹 (A) **Built-in Validators** (already provided by Django)

Some commonly used ones:

1. **MaxLengthValidator** – ensures the field value doesn’t exceed the given length.
2. **MinLengthValidator** – ensures the field value has at least the required length.
3. **EmailValidator** – checks if input is a valid email format.
4. **URLValidator** – checks if input is a valid URL.
5. **RegexValidator** – checks if input matches a specific pattern (regular expression).
6. **validate\_slug** – checks if input is a valid slug (letters, numbers, hyphens, underscores only).
7. **validate\_email** – validates email address format.

👉 These are used by passing them inside the validators=[...] argument of a form field.

### 🔹 (B) **Custom Validators** (you create your own function)

* A validator is simply a function that:
  + takes a value as input
  + raises a **ValidationError** if invalid
  + does nothing if valid

👉 Useful when built-in validators are not enough.

## ✅ 3. Example Code (with Full Explanation)

Forms.py

**from** django **import** forms

**from** django**.**core **import** validators

*# -----------------------------*

*# Custom Validator Function*

*# -----------------------------*

*# This function checks if email starts with 's'*

**def** start\_with\_s(**value**)**:**

*# value = user input*

**if** value[0]**.**lower() **!=** 's'**:**   *# lower() to also handle uppercase 'S'*

**raise** forms**.**ValidationError('Email should start with "s".')

*# -----------------------------*

*# Django Form with Validators*

*# -----------------------------*

class **Registration**(*forms***.***Form*)**:**

*# ---------------------*

*# Built-in Validators*

*# ---------------------*

    name **=** forms**.**CharField(

**validators=**[

            validators**.**MaxLengthValidator(10)**,**  *# max length = 10 chars*

            validators**.**MinLengthValidator(3)    *# min length = 3 chars*

        ]

    )

*# ---------------------*

*# Custom Validator*

*# ---------------------*

    email **=** forms**.**EmailField(

**validators=**[start\_with\_s]  *# calls custom function*

    )

*# ---------------------*

*# Normal Field*

*# ---------------------*

    city **=** forms**.**CharField()

*# ---------------------*

*# Password with Widget*

*# ---------------------*

*# Using PasswordInput widget hides user input with dots (••••)*

    password **=** forms**.**CharField(**widget=**forms**.**PasswordInput())

## ✅ 4. How Django Executes Validators

When form validation runs (form.is\_valid()):

1. Django first checks **field type** (e.g., CharField, EmailField).
   * If invalid → raises built-in error (e.g., "Enter a valid email").
2. Then it applies **validators list** for that field.
   * Executes each validator function in order.
   * If any raises ValidationError, that error is shown in form.

## ✅ 5. Other Ways to Add Validation

Besides validators list, Django gives extra ways:

### (A) clean\_<fieldname>() method

* Add custom validation for a **specific field** inside the form class.

class **Registration**(*forms***.***Form*)**:**

    name **=** forms**.**CharField()

**def** clean\_name(*self*)**:**

        data **=** *self***.**cleaned\_data['name']

**if** **not** data**.**isalpha()**:**   *# check only alphabets*

**raise** forms**.**ValidationError("Name must contain only letters")

**return** data

### (B) clean() method

* Runs **after all fields are validated**.
* Used to validate **multiple fields together**.

class **Registration**(*forms***.***Form*)**:**

    password **=** forms**.**CharField(**widget=**forms**.**PasswordInput())

    confirm\_password **=** forms**.**CharField(**widget=**forms**.**PasswordInput())

**def** clean(*self*)**:**

        cleaned\_data **=** super()**.**clean()

        pwd **=** cleaned\_data**.**get("password")

        cpwd **=** cleaned\_data**.**get("confirm\_password")

**if** pwd **!=** cpwd**:**

**raise** forms**.**ValidationError("Passwords do not match")

## ✅ 6. Widgets vs Validators

* **Widgets** = only affect **UI (input display)**, like PasswordInput hides characters.
* **Validators** = check if the **input value is valid**.

So in your code:

password = forms.CharField(widget=forms.PasswordInput())

* widget=PasswordInput hides the password in the browser.
* But validation still needs to be added separately (like min length).

## ✅ 7. Where Validators Are Useful

* Registration forms (username length, password rules, email format).
* Contact forms (validate phone number, email).
* Search forms (validate query format).
* Checkout forms (credit card number, postal code).

## ✅ 8. Key Points for Notes

* Validators are **functions that validate user input**.
* Django has many **built-in validators**.
* You can create **custom validators** for special rules.
* Three validation methods:
  1. **Field validators list**
  2. **clean\_<fieldname>() method**
  3. **clean() method**
* Validation happens when you call form.is\_valid().
* If invalid → form.errors contains error messages.