# 📌 Topic: ****Django Form Validate Specific Field and All at Once****

# ****Code 31****

# ****Video 37****

## 🔹 1. Why Validation is Needed?

When users fill out forms, they might:

* Enter invalid data (short names, wrong email format).
* Skip required fields.
* Enter unsafe data (XSS attacks, SQL injection attempts).

👉 Django forms help us **validate automatically** (like checking if an email looks valid).  
👉 But sometimes we want **custom rules** (e.g., Name must have at least 4 characters, Password must have 7 digits).

## 🔹 2. Validation Methods in Django Forms

### (A) Field-specific validation

* Django allows you to validate a single field with a method:  
  clean\_<fieldname>(self)
* Runs **only for that specific field**.
* Must return the validated value or raise ValidationError.

### (B) Form-wide validation (all fields together)

* Use clean(self) method.
* Runs **after all field-level validations**.
* Used when validation **depends on multiple fields** (e.g., comparing passwords, conditional checks).
* Must return cleaned\_data.
* Errors are added with self.add\_error(field, "message").

# 🧑‍💻 Example 1: Field-specific Validation

# forms.py

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

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

    name **=** forms**.**CharField()

    email **=** forms**.**EmailField()

    city **=** forms**.**CharField()

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

*# Validate only 'name' field*

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

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

**if** len(name\_value) **<** 4**:**

**raise** forms**.**ValidationError("Enter at least 4 characters for Name")

**return** name\_value

*# Validate only 'email' field*

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

        email\_value **=** *self***.**cleaned\_data['email']

**if** **not** email\_value**.**endswith('@gmail.com')**:**

**raise** forms**.**ValidationError("Email must end with @gmail.com")

**return** email\_value

✅ Explanation:

* If user enters Tom → Validation error (name too short).
* If user enters abc@yahoo.com → Validation error (not @gmail.com).
* Django will show these errors in the form.

# 🧑‍💻 Example 2: Form-wide Validation (all at once)

# forms.py

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

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

    name **=** forms**.**CharField()

    email **=** forms**.**EmailField()

    city **=** forms**.**CharField()

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

*# Validate all fields together*

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

        cleaned\_data **=** super()**.**clean()   *# get already validated data*

        name\_value **=** cleaned\_data**.**get('name')

        email\_value **=** cleaned\_data**.**get('email')

        password\_value **=** cleaned\_data**.**get('password')

*# Check Name length*

**if** name\_value **and** len(name\_value) **<** 4**:**

*self***.**add\_error('name'**,** "Please enter at least 4 characters for Name")

*# Check Email domain*

**if** email\_value **and** **not** email\_value**.**endswith('@gmail.com')**:**

*self***.**add\_error('email'**,** "Email must end with @gmail.com")

*# Check Password length*

**if** password\_value **and** len(password\_value) **<** 7**:**

*self***.**add\_error('password'**,** "Password must be at least 7 characters long")

**return** cleaned\_data

✅ Explanation:

* clean() runs **after all field validations**.
* Useful when validation depends on multiple fields.
* self.add\_error(field, message) → attaches error to that field.
* Always return cleaned\_data.

# 🧩 How to Show Errors in Template

**<**!-- register**.**html --**>**

**<**form method**=**"POST" novalidate**>**

    {**%** csrf\_token **%**}

    {{ form**.**as\_p }}   **<**!-- automatically shows validation errors --**>**

**<**button *type***=**"submit"**>**Register**</**button**>**

**</**form**>**

If errors exist → Django will automatically render them below the input fields.

# 📌 Full Example (with view and template)

### forms.py

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

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

    name **=** forms**.**CharField()

    email **=** forms**.**EmailField()

    city **=** forms**.**CharField()

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

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

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

**if** len(name\_value) **<** 4**:**

**raise** forms**.**ValidationError("Name must be at least 4 characters long")

**return** name\_value

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

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

        email\_value **=** cleaned\_data**.**get('email')

        password\_value **=** cleaned\_data**.**get('password')

**if** email\_value **and** **not** email\_value**.**endswith('@gmail.com')**:**

*self***.**add\_error('email'**,** "Email must end with @gmail.com")

**if** password\_value **and** len(password\_value) **<** 7**:**

*self***.**add\_error('password'**,** "Password must be at least 7 characters long")

**return** cleaned\_data

### views.py

**from** django**.**shortcuts **import** render

**from** **.**forms **import** Registration

**def** register(**request**)**:**

**if** request**.**method **==** "POST"**:**

        form **=** Registration(request**.**POST)

**if** form**.**is\_valid()**:**

            print("Form is valid:"**,** form**.**cleaned\_data)

**else:**

            print("Form errors:"**,** form**.**errors)

**else:**

        form **=** Registration()

**return** render(request**,** "register.html"**,** {"form"**:** form})

### register.html

**<**!DOCTYPE html**>**

**<**html**>**

**<**head**>**

**<**title**>**Registration**</**title**>**

**</**head**>**

**<**body**>**

**<**h1**>**Register**</**h1**>**

**<**form method**=**"POST"**>**

        {**%** csrf\_token **%**}

        {{ form**.**as\_p }}   **<**!-- shows fields **+** errors --**>**

**<**button *type***=**"submit"**>**Submit**</**button**>**

**</**form**>**

**</**body**>**

**</**html**>**

# 📌 Extra Things You Missed

✅ 1. **Default Error Messages**  
You can override Django’s default error messages:

name **=** forms**.**CharField(**error\_messages=**{'required'**:** 'Name cannot be empty!'})

✅ 2. **Non-field Errors**  
Sometimes you want an error not linked to a single field (e.g., password1 != password2).

**from** django**.**core**.**exceptions **import** ValidationError

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

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

    pass1 **=** cleaned\_data**.**get('password1')

    pass2 **=** cleaned\_data**.**get('password2')

**if** pass1 **!=** pass2**:**

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

→ These errors show up in {{ form.non\_field\_errors }}.

✅ 3. **Custom Validators**  
Instead of writing clean\_<field>(), you can use reusable validators:

**from** django**.**core**.**exceptions **import** ValidationError

**def** validate\_gmail(**value**)**:**

**if** **not** value**.**endswith('@gmail.com')**:**

**raise** ValidationError("Email must be Gmail")

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

    email **=** forms**.**EmailField(**validators=**[validate\_gmail])

✅ 4. **Order of Validation Execution**

1. Field-specific validation (clean\_<field>())
2. Built-in validators (EmailField, CharField, etc.)
3. Form-wide clean() method

# 🎯 Final Takeaways

* **clean\_<field>()** → validate a single field only.
* **clean()** → validate the whole form (multiple fields).
* Use self.add\_error(field, message) to attach error to a field.
* Use ValidationError for raising errors.
* Use form.errors in views for debugging.
* Template automatically shows errors if using {{ form.as\_p }}.
* Can also define **non-field errors** and **custom validators**.