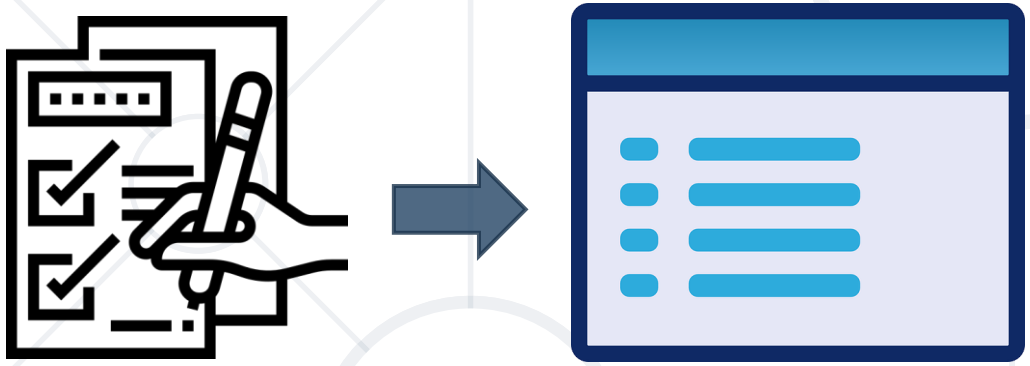
Django Forms Basics



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#python-web

Table of Contents

Web Forms

Web/HTML Form

- Web Form is an online page
  - Enables users to **input** **data**
  - This data is subsequently **transmitted** to a server for **processing**
- Web forms **emulate** traditional **paper documents** where users manually complete specific **fields**
  - They can encompass **various elements**
    - text boxes, checkboxes, select options, and a submit button, among others

Web/HTML Form

- In HTML, forms are wrapped within the **<form>** tag
- When working with forms, it's **essential** to use either the **GET** or **POST** **HTTP** methods

```
<body>

<form action="/your-url/" method="post">
    <!-- input elements -->
    <!-- submit button -->
</form>

</body>
```

Forms in Django

Django Forms

- Django offers a comprehensive **set** of **tools** and **libraries** that
  - allow developers to create **forms** using **Python** **code**
  - support all features of **HTML** **forms** in a **Pythonic** **manner**
  - **simplify** and **automate** a significant portion of the **form creation** and **handling** process

Django Forms

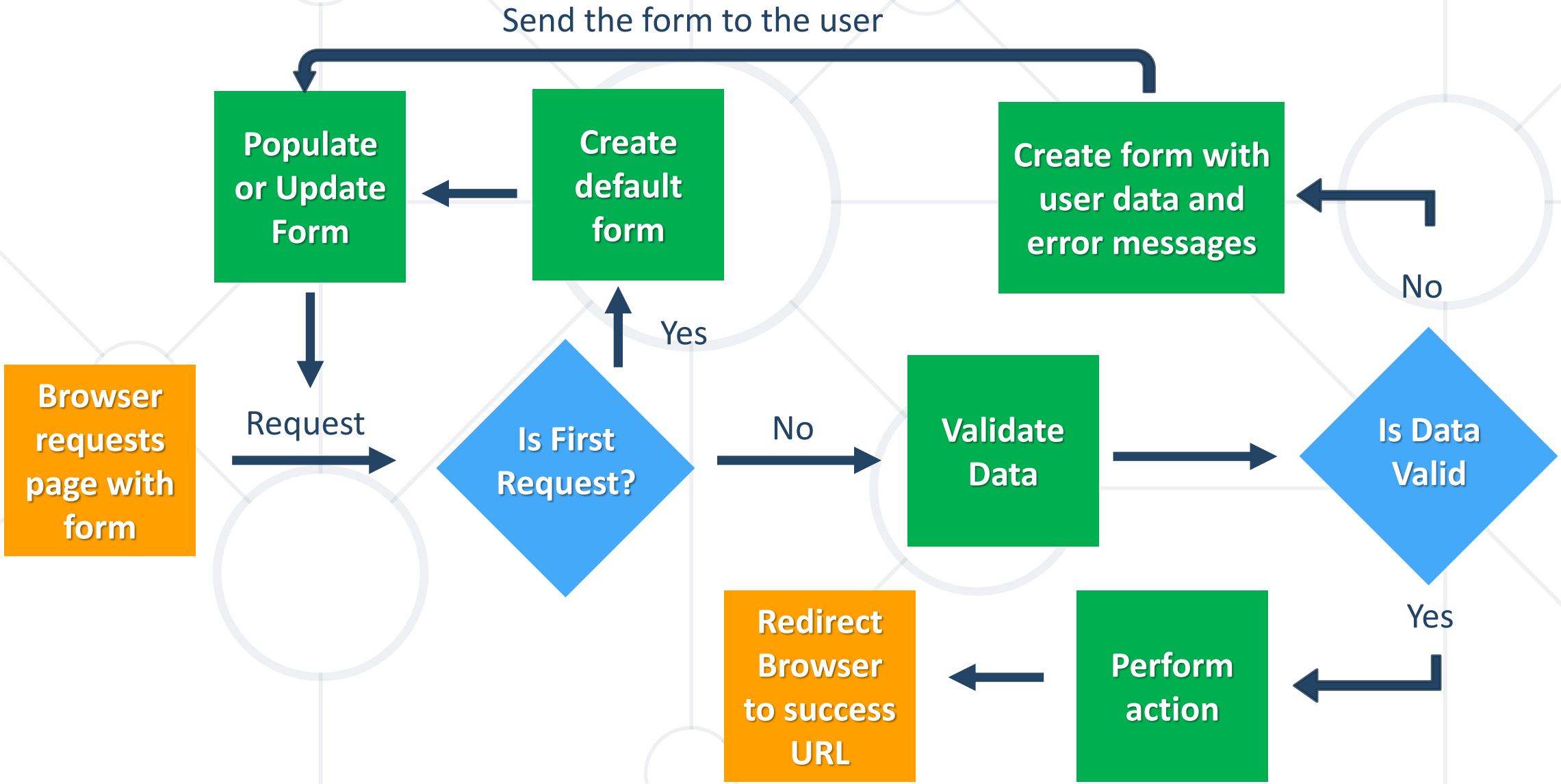
- The **form fields** in Django **correspond** to HTML form **<input>** elements

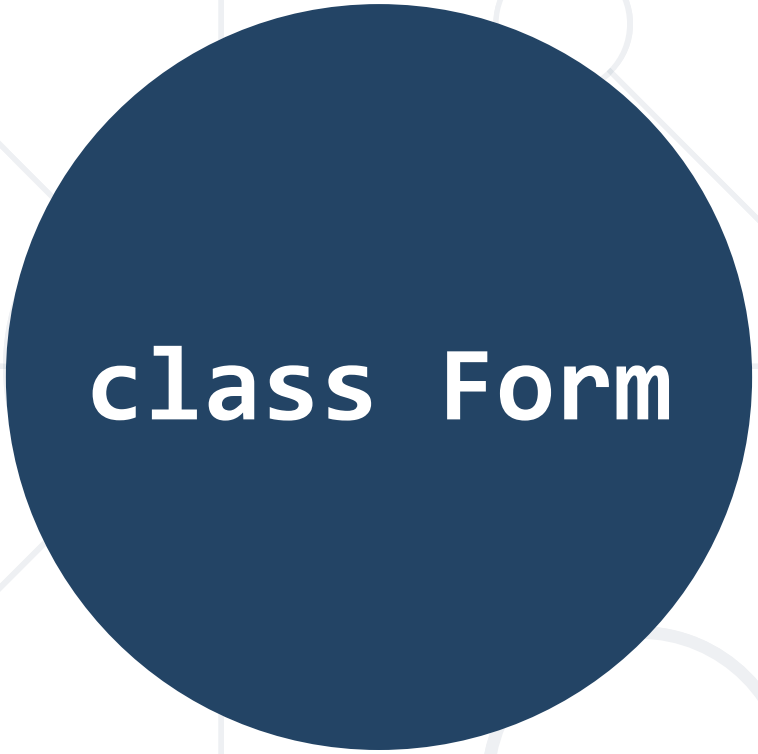
```
from django import forms

class NameForm(forms.Form):
    name = forms.CharField(label='Your Name', max_length=50)
```

```
<form action="/your-url/" method="post">
    <label for="name">Your name: </label>
    <input type="text" id="name" name="name" maxlength="50" required>
    <input type="submit" value="OK">
</form>
```

Django Forms Handling





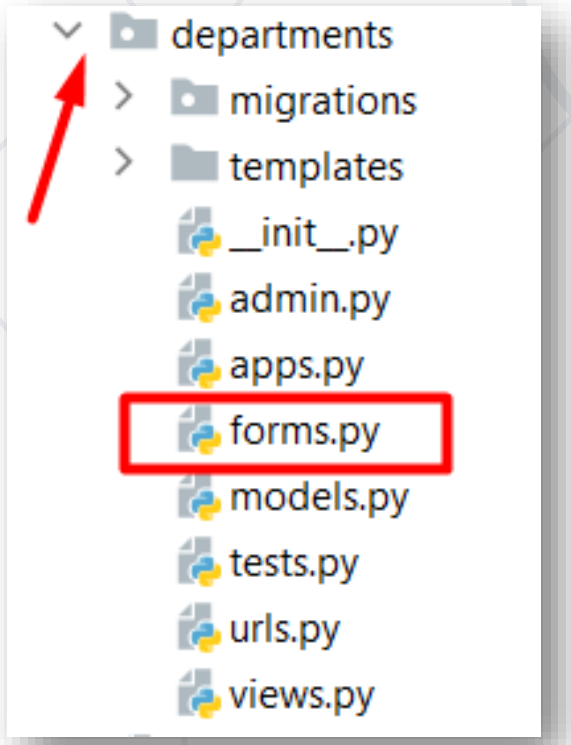
Django Form Class

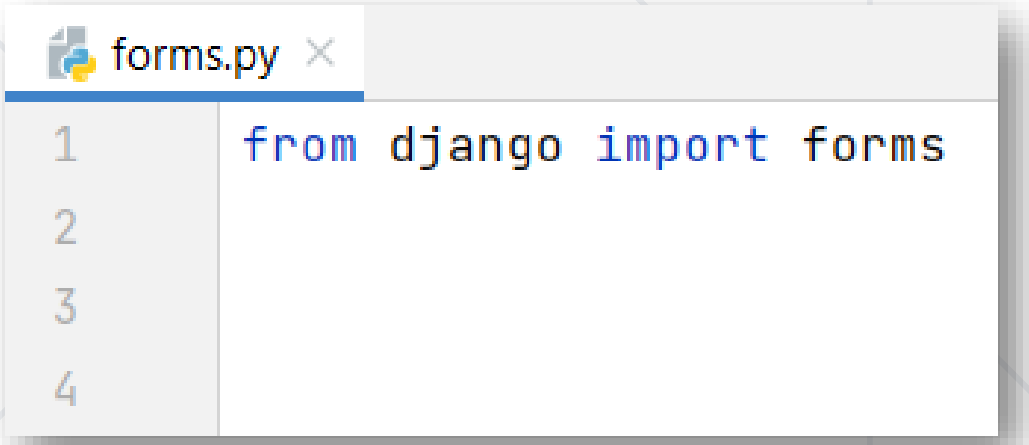
Django Form Class

- The Django **Form Class** serves several **key** functions
  - Defines the **form fields**, specifying what **data** the form will collect
  - Determines the **behavior** and **appearance** of the form
  - Handles **validation** when the form is **submitted**, ensuring the **data meets** the **specified criteria**

Creating Django Forms

- Create a `forms.py` **file** in the app directory
- Import the **forms module**





Creating Django Forms

- To create a form in Django:
  - **Inherit** from the **Form** class
  - **Add** the form **fields**

```
from django import forms

class NameForm(forms.Form):
    name = forms.CharField(...)
```

- **Form** and **Model classes** **share** most **field types** and some **common arguments**

Handling Django Forms

- Create a **view** with a corresponding URL path

views.py

```
from .forms import NameForm

def add_new_name(request):
    if request.method == "GET":
        form = NameForm()

    if request.method == "POST":
        form = NameForm(request.POST)
        if form.is_valid():
            #  do something with the data
            #  redirect to the desired page

    return render(request, "index.html", {"form": form})
```

Generates an empty form

Binds the collected data to the form

Checks if the data is valid

Returns the empty form or invalid data with errors

Flat is Better than Nested

- In many cases, you only need to **instantiate** a form **once**

views.py

```
from .forms import NameForm

def add_new_name(request):
    form = NameForm(request.POST or None)
    if form.is_valid():
        #  do something with the data
        #  redirect to the desired page

    return render(request, "index.html", {"form": form})
```

if the request method is not a POST request, generate and render an empty form

Django Template with Form

- Create a **template** with the form

index.html

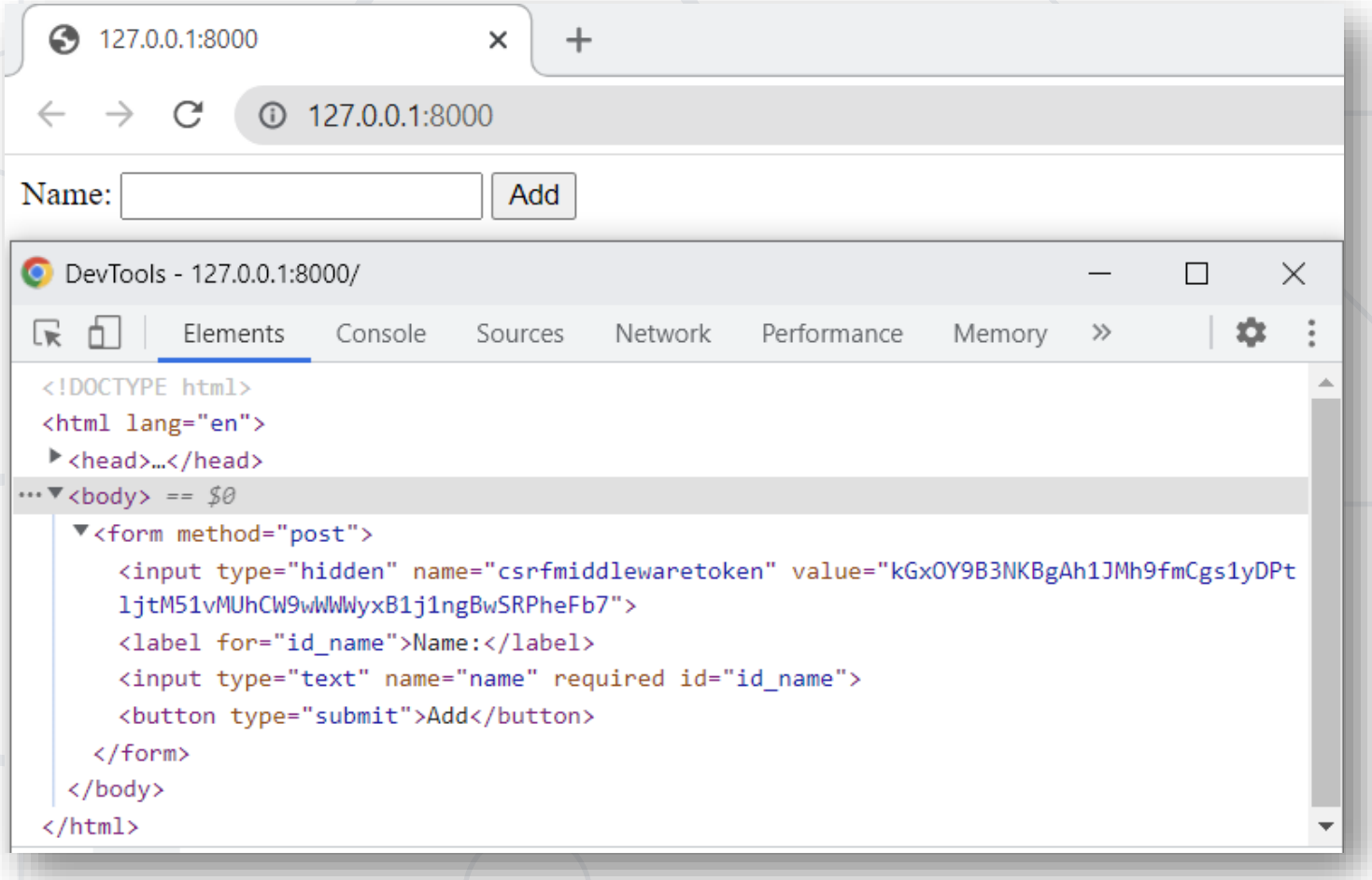
```
<body>
  <form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit">Add</button>
  </form>
</body>
```

Django form

Cross-Site Request Forgery token for security

Displaying Forms

- Start the development server



Django Form Fields

Django Form Fields

- The **crucial** aspect of creating a **form** lies in **defining** its **fields**
- Each field incorporates **custom validation logic**
- Fields can take **common arguments**
- Some fields accept **field-specific arguments**

More form fields: *https://docs.djangoproject.com/en/4.2/ref/forms/fields/#built-in-field-classes*

Form Field Arguments

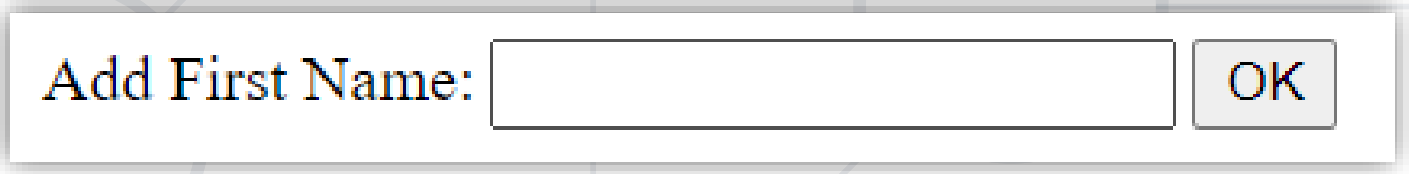
- By **default**, each Field class in Django assumes that a **value is required**
  - If you pass an **empty value**, it will raise a **ValidationError**
- You have the option to **specify** that a field is **not required**

```
first_name = forms.CharField(required=False)
```

Form Field Arguments

- You have the option to assign a **"user-friendly" label** to a field
- This **label** is used when the field is **displayed** within a **form**

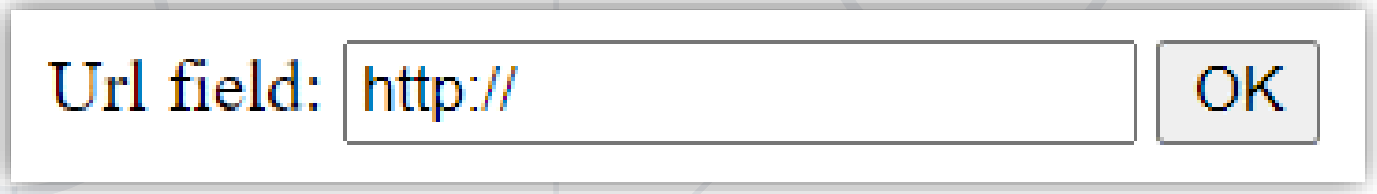
```
first_name = forms.CharField(label="Add First Name")
```



Form Field Arguments

- The **`initial`** **argument** lets you **specify** the **initial value** to use
  - when rendering this Field in an unbound Form

```
url_field = forms.URLField(initial='http://')
```



Form Field Arguments

- `help_text` attribute is used to display the **help text**
- It will be displayed **along** with the field in a form

```
first_name = forms.CharField(help_text='Add your first name')
```

First name:

Add your first name

More Field Arguments: *https://docs.djangoproject.com/en/4.2/ref/forms/fields/#core-field-arguments*

Built-in Widgets

Django Widget

- It is Django's **representation** of an HTML **input element**
- **Widgets** handle:
  - **Rendering** of HTML
  - **Extraction** of data
- Django automatically assigns **default widgets**
  - Based on the **type** of **data**
  - Each form field has a **corresponding** built-in **widget**
- You can **specify** a **different widget** for a field
  - By using the `widget` argument on the **field definition**

Built-in Widgets

- **`CharField`** uses **`TextInput`** widget by **default**
  - Renders as: **`<input type="text" ...>`**

- You can specify a field that uses a **larger `Textarea` widget**

```
comment = forms.CharField(
            widget=forms.Textarea
        )
```

Built-in Widgets

- **NumberInput**
  - HTML input type: "number"
- **EmailInput**
  - HTML input type: "email"
- **PasswordInput**
  - HTML input type: "password"

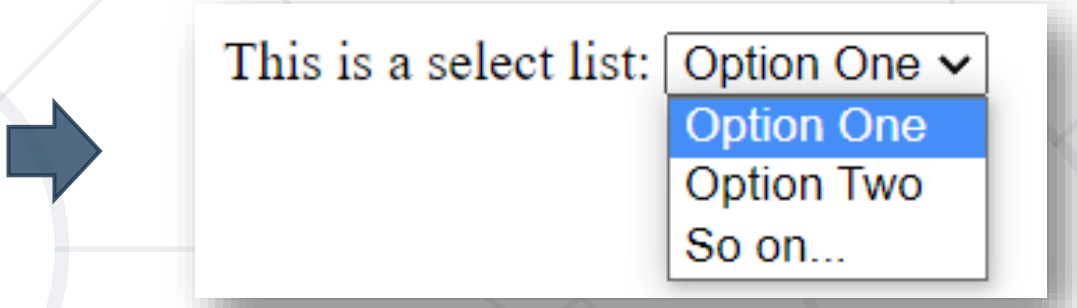
Built-in Widgets

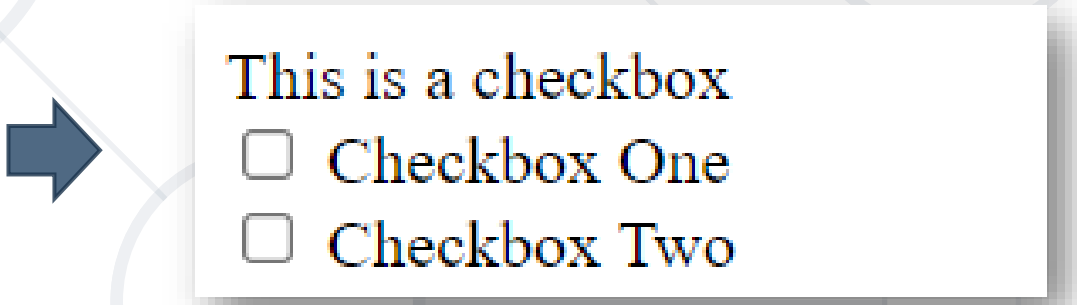
- **URLInput**
  - HTML input type: "url"
- **DateInput**
  - HTML input type: "text"
- **DateTimeInput**
  - HTML input type: "text"

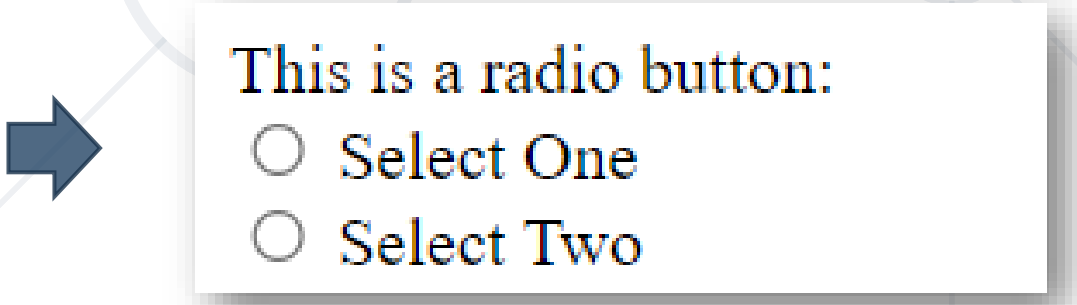
More built-in widgets: *https://docs.djangoproject.com/en/4.2/ref/forms/widgets/#built-in-widgets*

Select, Checkbox and Radio Button

- A **select list** allows you to choose options from a drop-down menu
- A **checkbox** allows you to select options from a list of options
- A **radio button** allows you to select only one option from a list of options







Select List Widget

- `Select` is the default **widget** for `ChoiceField`
- It **can** also be used with **other fields**

```
class SelectOptionForm(forms.Form):
    CHOICES = (
        ('1', 'Option One'),
        ('2', 'Option Two'),
    )

    choice_field = forms.ChoiceField(choices=CHOICES)
    char_field = forms.CharField(widget=forms.Select(choices=CHOICES))
```

The same as in the model field

Checkbox Widget

- `CheckboxInput` is the default widget for `BooleanField`
- Returns **True**, if it is checked

```
class CheckboxForm(forms.Form):
    checkbox_field = forms.BooleanField(required=False)
```

- To create a `checkbox` that can be either **checked** or **unchecked**
  - Set the attribute `required` to `False`

Radio Button Widget

- `RadioSelect` is similar to the Django **Select** widget
  - It can be used with a `ChoiceField`

```
class RadioButtonForm(forms.Form):
    CHOICES = (...)

    choices_field = forms.ChoiceField(
        choices=CHOICES,
        widget=forms.RadioSelect(),
    )

    char_field = forms.CharField(
        widget=forms.RadioSelect(choices=CHOICES),
    )
```

Django Widget Attributes

- **Widgets** in Django provide a way to **specify** **HTML attributes** using Python code

```
comment = forms.CharField(
    widget=forms.Textarea(
        attrs={'cols': 80, 'rows': 20,
               'class': 'special',
               'title': 'Add a comment'}))
```

- Note: **Mixing** the **main code logic** with the **front end** is discouraged due to considerations of **maintainability** and **separation of concerns**

Django ModelForm Class

The ModelForm Class

- In a database-driven application, there are instances where the **forms mirror** the **models**
  - **Field types** are **already specified** in the model
- Using the `ModelForm` can help **prevent redundant definitions**
- This approach **automatically generates** a **form** based on a specific model

Form vs ModelForm

- **Form**:
  - Independent of a model
  - Does not directly interact with models
  - E.g., search form, contact form, subscription form
- **ModelForm**:
  - Converts a model into a form
  - Directly interacts with models by adding or editing them
  - E.g., registration form, newsletter article form, blog post form

Create a Django Model Form

- First, create a **model** with some fields

models.py

```
from django.db import models

class Name(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models. CharField(max_length=50)
```

Create a Django Model Form

Software University

forms.py

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
        fields = '__all__'
```

Inherit from the ModelForm class

Specify the model you want to create a form for

Add the fields from the model to the form

Create a Django Model Form

Software University

- Create a **view** with a corresponding URL path

views.py

```
from .forms import NameForm

def add_new_name(request):
    if request.method == "GET":
        form = NameForm()

    if request.method == "POST":
        form = NameForm(request.POST)
        if form.is_valid():
            form.save()
            #  redirect to the desired page

    return render(request, "index.html", {"form": form})
```

Create a Django Model Form

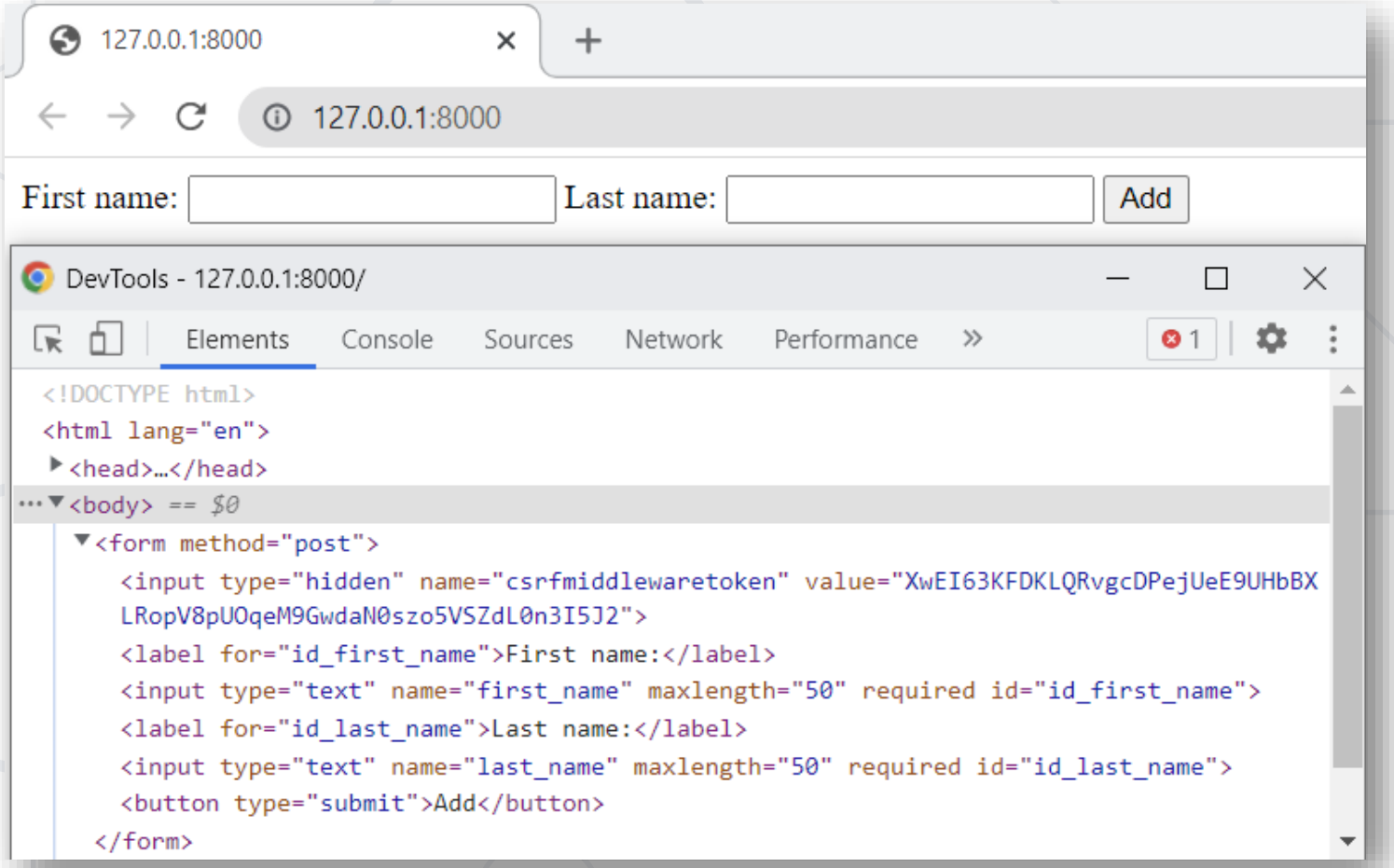
- Create a **template** with the form

index.html

```
<body>
  <form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit">Add</button>
  </form>
</body>
```

Create a Django Model Form

- Start the development server



Update a Model Instance Using a Form

Software University

- Create an update **view** with a corresponding URL path

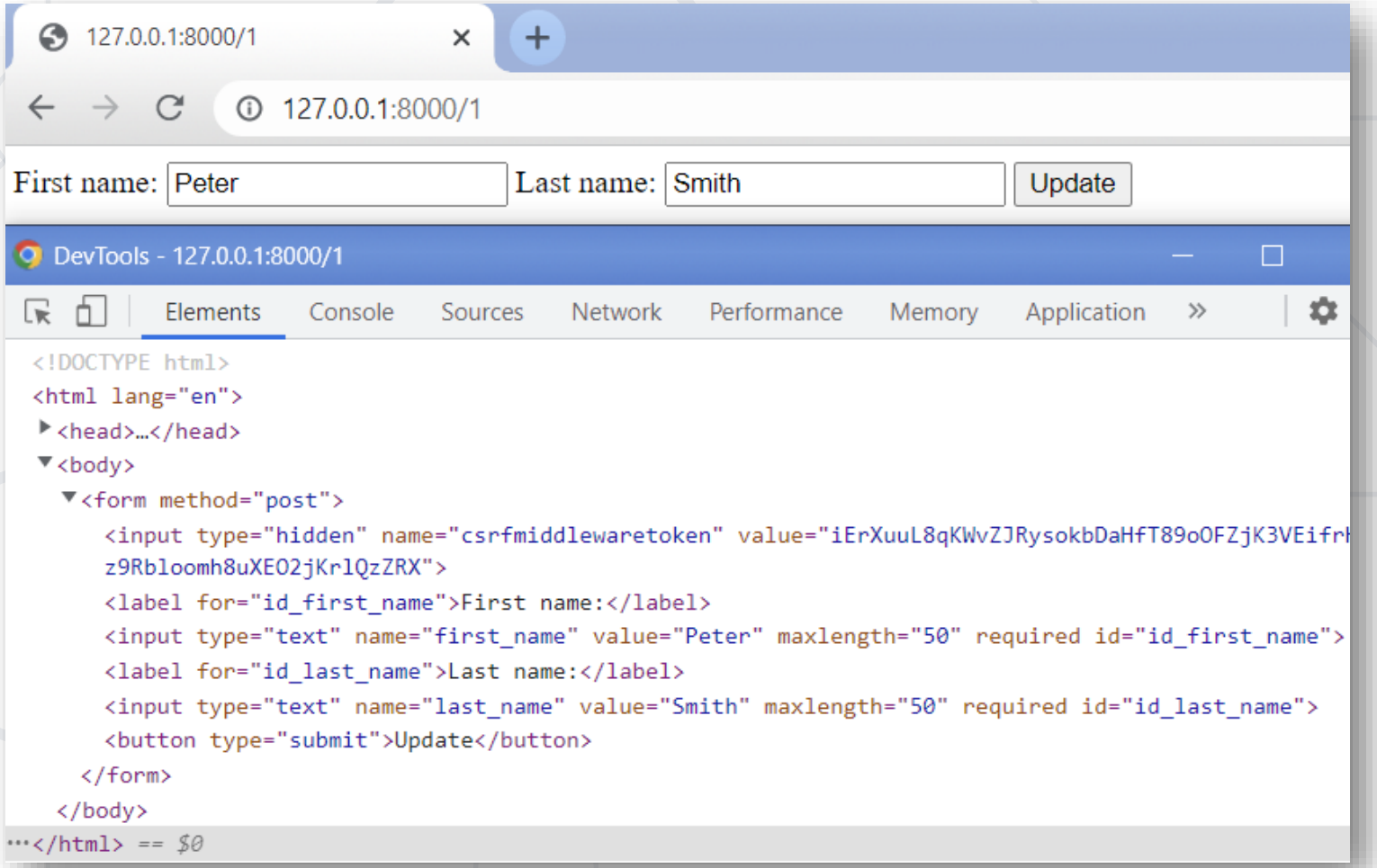
views.py

```
from .forms import NameForm
from .models import Name
from django.shortcuts import get_object_or_404

def update_name(request, pk):
    name = get_object_or_404(Name, pk=pk)
    form = NameForm(request.POST or None, instance=name)
    if form.is_valid():
        form.save()
        # redirect to the desired page
    return render(request, 'update.html', {'form': form})
```

Update a Model Instance Using a Form

- Start the development server



ModelForm Meta Options

Class Meta

- Similar to the Model, a `ModelForm` class contains an **inner** `Meta` class with pre-defined **options**
- These **Meta options** control the **behavior** and **appearance** of the form
- A comprehensive list of these options can be found in Django's `ModelFormOptions` class, in its `__init__()` method

Model Option

- When configuring a **ModelForm**, it's essential to **specify** the **model** that will be used to **generate** the **form**
- This value should be set to the **Model** class **itself**, **not** an **instance** of it

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
```

Fields Option

- It's crucial to **explicitly define** the **fields** that will be edited in the form
  - Failing to do so can potentially result in **security vulnerabilities**
- You can use `__all__` to **include all fields** from the model

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
        fields = ['first_name', 'last_name']
```

Use iterable like list or tuple

Exclude Option

- Frequently, it's more convenient to **specify** which **fields** should be **excluded** from the form

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
        exclude = ['last_name']
```

ModelForm Field Types

- Each model field has a **corresponding** default form field

Model Field	Form Field
CharField	CharField with max_length set
IntegerField	IntegerField
FloatField	FloatField
BooleanField	BooleanField, or NullBooleanField if null=True
ForeignKey	ModelChoiceField
ManyToManyField	ModelMultipleChoiceField

Full table: *https://docs.djangoproject.com/en/4.2/topics/forms/modelforms/#field-types*

Overriding the Default Fields

- You have the flexibility to **change** the **field type** for the model
- Use the **`widgets`** option
  - A dictionary mapping **field names** to **widget classes/instances**

```
class CommentForm(forms.ModelForm):
    class Meta:
        ...
        widgets = {
            'comment': Textarea(),
        }
```

Overriding the Default Fields

- You can **specify** a **different label** for a field
- Use the `labels` option
  - A dictionary mapping **field names** to **strings**

```
class NameForm(forms.ModelForm):
    class Meta:
        ...
        labels = {
            'first_name': 'Add Your First Name',
        }
```

Overriding the Default Fields

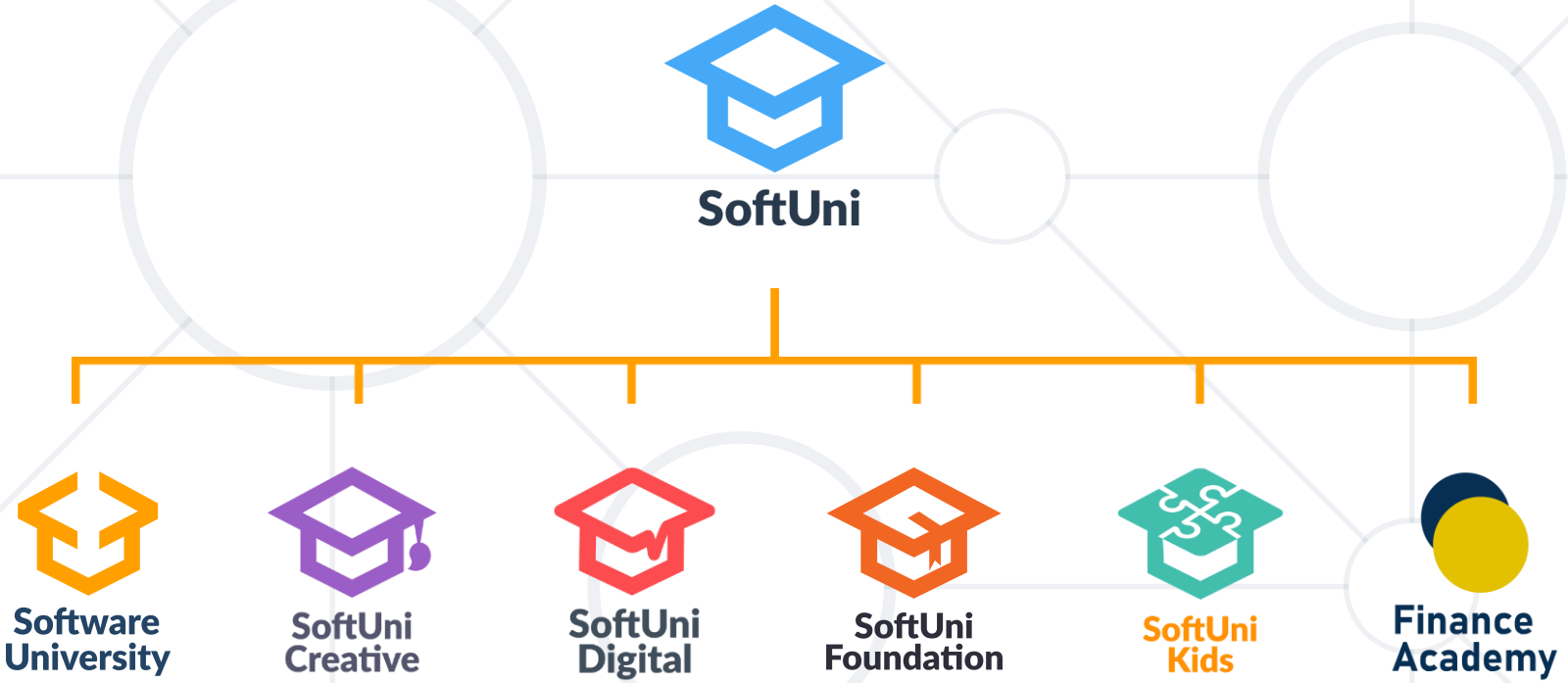
- You can **add help texts** for fields
- Use the `help_texts` option
  - A dictionary mapping **field names** to **strings**

```
class EmailForm(forms.ModelForm):
    class Meta:
        ...
        help_texts = {
            'email': 'Please, provide a valid email address',
        }
```

Summary

- Django **Forms** (forms.**Form**)
- Form **Fields**
  - Built-in **Widgets**
- **ModelForm** (forms.**ModelForm**)
- ModelForms **Meta Options**
  - model, fields, exclude, labels, widgets

Questions?



© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg