Final Review

CS 3550, Fall 2023

Taught by Prof. Pavel Panchekha

The structure of the final

The final will be **110 points** and take place over **two hours**

- WEB L104 (normal room) at **6–8pm** on Monday, 11 Dec
- **No notes** or electronic devices; bring pens, snacks, water

Includes midterm material on HTML, CSS, networking

- Roughly 1/3 of the final is midterm material
- Check the **midterm review slides** to review that material

Django Models

Query concepts

Explain the following **query concepts**:

Migration

1 + N problem

Aggregation

Query concepts

Explain the following **query concepts**:

Migration	How to update database to match change in models
1 + N problem	When you make one query and then one more query per returned object
Aggregation	A query that returns values computed from multiple objects in the database

Write a model for…

Imagine you're writing a **baseball video game**.

In the video game, there are teams and people: players and coaches.

Teams have a name, a manager, and a list of players.

People have names, salaries, and positions (model this as a string).

Write the **Django model** for this video game.

Write a model for…

In the video game, there are teams and people: players and coaches.

```
from django.db import models

class Team(models.Model):
    ...

class Person(models.Model):
    ...
```

Write a model for…

Teams have a name, a coach, and a list of players.

```
class Team(models.Model):
    name = models.CharField()
    manager = models.ForeignKey(Person)

class Person(models.Model):
    team = models.ForeignKey(Team)
```

Write a model for…

People have names, salaries, and positions (model this as a string).

```
class Person(models.Model):
    name = models.CharField()
    salary = models.DecimalField()
    position = models.CharField()
```

Write a model for…

Continuing with the same baseball video game:

A manager can quit, in which case the team will no longer have one

People can retire, in which case they no longer have a team

People can also pass away, in which case the Player object is deleted

Teams can be disbanded (deleted) if all the players join other teams

Add **field parameters** to each field to allow this.

Write a model for…

Add **field parameters** to each field to allow this.

```
class Team(models.Model):
   manager = models.ForeignKey(Person,
      blank=True, null=True,
      on_delete=models.SET_NULL)

class Person(models.Model):
   team = models.ForeignKey(Team,
      blank=True, null=True,
      on_delete=models.PROTECT)
```

What query method does…

Keeps a subset of the objects		Gets the first item in a query	
Removes a subset of the objects		Reverses the order of objects	
Combines the results of two queries		Checks if a specific object is in a query	
Removes duplicates from a query		Checks if any object is in a query	
Sorts the objects by a field		Counts how many objects were selected	

What query method does…

Keeps a subset of the objects	`filter`	Gets the first item in a query	`first`
Removes a subset of the objects	`exclude`	Reverses the order of objects	`reverse`
Combines the results of two queries	`union`	Checks if a specific object is in a query	`contains`
Removes duplicates from a query	`distinct`	Checks if any object is in a query	`exists`
Sorts the objects by a field	`order_by`	Counts how many objects were selected	`count`

Write a query for…

Continuing with the same baseball video game, write a query for:

All people making over $1,000,000 on the team named “Cubs”

How many teams are currently without a manager

All *players* managed by “Bud Black”, ordered by salary (highest first)

You can assume no two people or teams have the same name

Write a query for…

All people making over $1,000,000 on the team named “Cubs”

```
models.Person.objects.filter(
   team__name=“Cubs”,
   salary__gt=1000000)
```

Write a query for…

How many teams are currently without a coach

```
models.Team.objects.filter(manager=None).count()
```

Write a query for…

All *players* managed by “Bud Black”, ordered by salary (highest first)

```
models.Person.objects \
   .exclude(name=“Bud Black”)
   .filter(team__manager__name=“Bud Black”) \
   .order_by(“salary”).reverse()
```

Write a function for…

If a `Person` manages a `Team` then they should be on that team. Write a `clean` method on `Team` that checks this constraint:

Write a function for…

If a `Person` manages a `Team` then they should be on that team. Write a `clean` method on `Team` that checks this constraint:

```
class Team:
   def clean(self):
      if self.manager and self.manager.team != self:
         return ValidationError(“Manager not on team”)
```

Write a function for…

Continuing with the same baseball video game, write a function:

Named `fire_manager(team)`

Will be called when a team fires its manager

Checks all invariants on the `Team` and `Person`

Updates all necessary data and returns nothing

Write a function for…

Continuing with the same baseball video game, write `fire_manager`:

```
def fire_manager(team):
    manager = team.manager
    team.manager = None
    manager.team = None
    team.full_clean()

    team.save()
    manager.save()
```

Django Templates

What files are these defined in:

Consider a Django project “`project`” containing one app “`app`”.

- Where can you rename a URL
- Where can you switch from debug to production mode
- Where can you modify the generated HTML
- Where can you add a new controller
- Where can you modify the CSS

What files are these defined in:

Consider a Django project "`project`" containing one app "`app`".

Where can you rename a URL	`project/urls.py`
Where can you switch to production mode	`project/settings.py`
Where can you modify the gen'd HTML	`app/templates/xyz.html`
Where can you add a new controller	`app/views.py`
Where can you modify the CSS	`static/xyz.css`

Write a Django template

Assume there is a coach

Write a Django template to generate this HTML for a `Team` stored in the `team` variable:

```
<li>
   <h1>The Cubs</h1>
   <p>The Cubs are managed by Bud Black</p>


   <p>There are 7 players</p>


</li>
```

Write a Django template

Write a Django template to generate this HTML for a `Team` stored in the `team` variable:

```
<li>
   <h1>The {{team.name}}</h1>
   <p>The {{team.name}} are managed by
      {{team.manager.name}}</p>
   <p>There are {{team.player_set|length}}
      player{{team.player_set|length|pluralize}}</p>
</li>
```

Write a Django template

Write a Django template to generate an HTML list like this

```
<ul>
  <li>The Cubs are managed by Bud Black</li>
  <li>The Dodgers are managed by Dave Roberts</li>
  <li>The Mets do not have a manager</li>
</li>
```

Write a Django template

Write a Django template to generate an HTML list like this

```
{% for team in teams %}
<ul>
    {% if team.manager %}
    <li>The {{team.name}} are managed by {{team.manager.name}}</li>
    {% else %}
    <li>The {{team.name}} do not have a manager</li>
    {% endif %}
</li>
{% endfor %}
```

Write a Django controller

Write a controller named `managers` that returns the previous HTML, which is located in a file named `managers.html`:

Write a Django controller

Write a controller named `managers` that returns the previous HTML, which is located in a file named `managers.html`:

```
def managers(request):
   teams = models.Team.objects.all()
   return render(request, “managers.html”,
      { “teams”: teams })
```

Forms

What HTML element is used for:

Short text		A date	
Long, multi-line text		One of three exclusive options	
A checkbox		A dropdown menu	
A number		A password	
An email address		A file upload	

What HTML element is used for:

Short text	`<input>`	A date	`<input type=date>`
Long, multi-line text	`<textarea>`	One of three exclusive options	`<input type=radio>`
A checkbox	`<input type=checkbox>`	A dropdown menu	`<select>`
A number	`<input type=number>`	A password	`<input type=password>`
An email address	`<input type=email>`	A file upload	`<input type=file>`

What HTML element is used for:

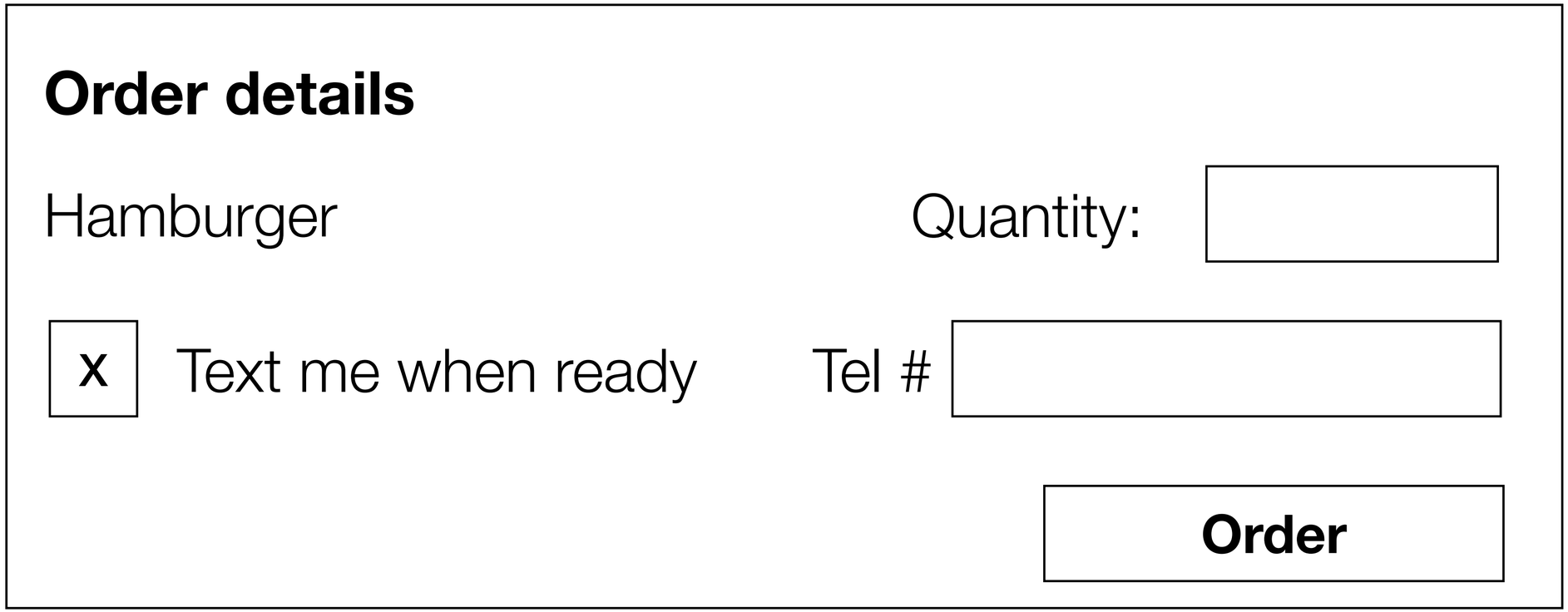
Describe what information should go into an input element	
Show error messages	
A clickable button that submits the form	

What HTML element is used for:

Describe what information should go into an input element	`<label>`
Show error messages	`<button>`
A clickable button that submits the form	`<output>`

Write an HTML form

Write HTML for this form; it should submit to `/order/new`

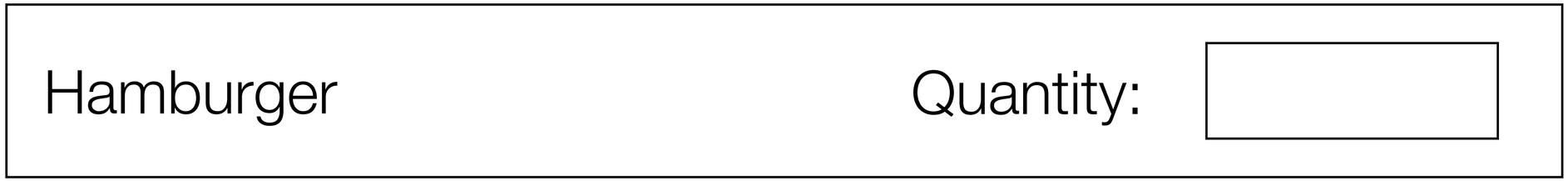


Write an HTML form

Order details

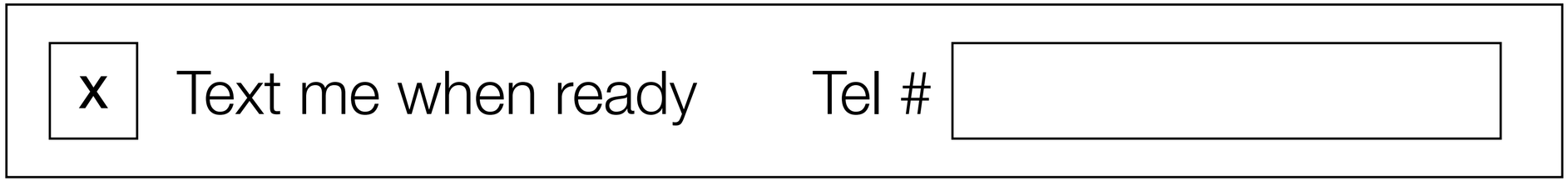
```
<form method=post action=/order/new>
   <h1>Order details</h1>
   <!-- … -->
</form>
```

Write an HTML form



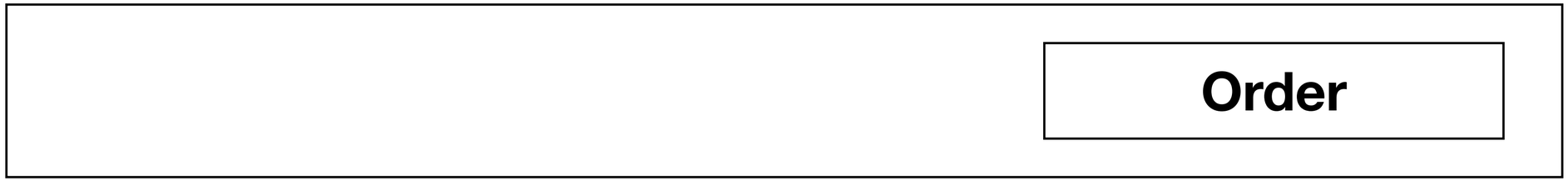

```
<p>Hamburger
  <label for=quantity>Quantity:</label>
  <input id=quantity name=quantity type=number></p>
```

Write an HTML form




```
<p><input id=textme name=textme type=checkbox>
   <label for=textme>Text me when done</label>
   <label for=tel>Tel #</label>
   <input id=tel name=tel type=tel></p>
```

Write an HTML form




```
<button>Order</button>
```

Write an HTML form

Write a controller for the previous form; it should create an `Order` object (containing a string `product` and integer `quantity`) and, if the user requested a text message, an `Alert` object (containing an `order` to store the `Order` and a string `tel_no`). Assume all necessary fields are present.

Redirect the user to `/order/ID/`, where `ID` is the `Order` object's ID.

Write an HTML form

Write a controller for the previous form…

```
def order_new(request):
    # …
```

Write an HTML form

it should create an `Order` object (containing a string `product` and integer `quantity`)

```
# …
quantity = request.POST[“quantity”]
order = models.Order(product=“Hamburger”, quantity=quantity)
order.save()
```

Write an HTML form

and, if the user requested a text message,

```
# …
if “textme” in request.POST:
   # …
```

Write an HTML form

an `Alert` object (containing an `order` to store the `Order` and a string `tel_no`).

```
# …
if “textme” in request.POST:
   tel_no = request.POST[“tel”]
   alert = models.Alert(order=order, tel_no=tel)
   alert.save()
```

Write an HTML form

Redirect the user to `/order/ID/`, where `ID` is the `Order` object's ID.

```
# …
return redirect(“/order/” + str(order.id) + “/“)
```

Form validation

Write a text input element that is required and allows up to 50 characters:

Write CSS that makes invalid input elements have a red border:

Form validation

Write a text input element that is required and allows up to 50 characters:

```
<input required maxlength=50>
```

Write CSS that makes invalid `<input>` elements have a red border:

```
input:invalid { border: 2px solid red; }
```

Deployment

What do these programs / commands do:

NGINX	
APT	
JournalCtl	
SSH	
Linux	

What do these programs / commands do:

NGINX	A gateway web server
APT	Install other programs
JournalCtl	Check system notifications / logs
SSH	Allow remote terminal connections
Linux	An operating system

Cloud facts / knowledge

Name three:

- Top clouds
- Cloud computing companies
- Instance parameters

Cloud facts / knowledge

Name three:

Top clouds	AWS, GCP, Azure
Cloud computing companies	Amazon, Google, Microsoft
Instance parameters	Memory, storage, CPU speed

Cloud facts / knowledge

Explain:

- What an “instance” is
- What a “bare-metal” instance is
- What “three nines” availability means

Cloud facts / knowledge

Explain:

What an “instance” is	A virtual computer you rent
What a “bare-metal” instance is	A physical computer you rent
What “three nines” availability means	It’s up 99.9% of the time

How much do these things cost (include units)

A domain	
An IPv4 address	
An IPv6 address	
Outbound bandwidth	
An HTTPS certificate	

How much do these things cost (include units)

A domain	$5–10/yr
An IPv4 address	$40–60
An IPv6 address	free
Outbound bandwidth	$50–100/TB
An HTTPS certificate	free

Security

Threat models

Suppose you've built an application much like Uber, for people to request, pay for, and see previous taxi rides. It is used by users, drivers, and your company's employees.

Describe 3 threats you face from users.

Threat models

Suppose you've built an application much like Uber, for people to request, pay for, and see previous taxi rides. It is used by users, drivers, and your company's employees.

Describe 3 threats you face from users.

Users may try to not pay for a ride, or get their money back

Users may want to see another user's ride history

Users may want to impersonate a company employee

Users and groups

Suppose this taxi application has the following model:

```
class Ride(models.Model):

    driver = models.ForeignKey(User, …)

    passenger = models.ForeignKey(User, …)
```

Write a condition for when a `user` should be able to view a given `ride`. You can use `is_user, is_driver`, `is_employee` functions.

Users and groups

Suppose this taxi application has the following model:

```
class Ride(models.Model):

    driver = models.ForeignKey(User, …)

    passenger = models.ForeignKey(User, …)
```

Write a condition for when a `user` should be able to view a given `ride`. You can use `is_user, is_driver`, `is_employee` functions.

```
ride.driver == user or ride.passenger == user \
    or is_employee(user)
```

Users and groups

In the example from before, suppose there is a `Group` named `“Employees”` that all company employees are a part of. Write the `is_employee` function:

Users and groups

In the example from before, suppose there is a `Group` named `“Employees”` that all company employees are a part of. Write the `is_employee` function:

```
def is_employee(user):
    return user.groups.filter(name=“Employees”).exists()
```

Define these terms

Authentication	
Authorization	
Cookie	
Injection vulnerability	
CSRF vulnerability	

Define these terms

Authentication	Determining who is using a computer system
Authorization	Determining whether a given user can take a certain action
Cookie	Data stored by your browser to indicate your identity
Injection vulnerability	When generated code doesn’t escape attacker-controlled data
CSRF vulnerability	When you process attacker-controller form submissions

JavaScript

Script tags

Write a `script` element that includes `/static/main.js` as a module and runs after the page finishes loading.

Script tags

Write a `script` element that includes `/static/main.js` as a module and runs after the page finishes loading.

```
<script type=module src=/static/main.js></script>
```

What jQuery method…

Creates a new element		Gets the value of an input element	
Selects descendants		Removes a class from an element	
Adds an element as the last child of another		Gets the children of an element	
Adds an element as the previous sibling of another		Gets the text content of an element	
Deletes an element		Gets the value of the `data-xyz` attribute	

What jQuery method…

Creates a new element	`$("<…>")`	Gets the value of an input element	`.val()`
Selects descendants	`.find("…")`	Removes a class from an element	`.removeClass(…)`
Adds an element as the last child of another	`.append("…")`	Gets the children of an element	`.children()`
Adds an element as the previous sibling of another	`.before("…")`	Gets the text content of an element	`.text()`
Deletes an element	`.remove()`	Gets the value of the `data-xyz` attribute	`.data("xyz")`

Write JavaScript/jQuery code that:

Deletes the last child of every `<tr>` element on the page:

Prints the textual contents of every `<button>`:

Write JavaScript/jQuery code that:

Deletes the last child of every `<tr>` element on the page:

```
$(“tr :last-child”).remove()
```

Prints the textual contents of every `<button>`:

```
let $buttons = $(“button”)
for (let $btn of $buttons) {
   console.log($btn.text());
}
```

Write JavaScript/jQuery code that:

Toggles the `open` class on `#details` when the `#toggle` is clicked

Write JavaScript/jQuery code that:

Toggles the `open` class on `#details` when the `#toggle` is clicked

```
$(“#toggle”).on(“click”, () => {
   $(“#details”).toggleClass(“open”);
});
```

Write JavaScript/jQuery code that:

Prevents any form from being submitted unless `#signature` has a value of “Joanne Smith”. If it doesn’t, replace the contents of the `<output>` element inside that form with “No signature”.

Write JavaScript/jQuery code that:

Prevents any form from being submitted unless `#signature` has a value of “Joanne Smith”. If it doesn’t, replace the contents of the `<output>` element inside that form with “No signature”.

```
$(“form”).on(“submit”, (e) => {
  if ($(“#signature”).val() != “Joanne Smith”) {
    $(e.target).find(“output”).text(“No signature”);
    e.preventDefault();
  }
});
```

Write JavaScript/jQuery code that:

Makes a GET request to `/updates` and prints the response to the console

Write JavaScript/jQuery code that:

Makes a GET request to `/updates` and prints the response to the console

```
$.ajax(“/updates”, {
   success: (data) => {
      console.log(data);
   }
})
```

Write JavaScript/jQuery code that:

Makes a POST request to `/thing/new/` with the data in `#theform` and prints to the console if there's an error.

Write JavaScript/jQuery code that:

Makes a POST request to `/thing/new/` with the data in `#theform` and prints to the console if there's an error.

```
$.ajax("/thing/new/", {
   method: "POST",
   data: new FormData($("#theform")[0]),
   error: () => {
      console.error("Something went wrong");
   }
})
```

Write JavaScript/jQuery code that:

Consider this jQuery code:

```
let data = await $.ajax("/3/submissions/")
for (let submission of data) {
  let user_id = submission.user.id;
  let user = await $.ajax("/user/" + user_id);
  do_thing(user);
}
```

Rewrite it so that all requests to `/user/ID` are made in parallel.

Write JavaScript/jQuery code that:

```
let data = await $.ajax(“/3/submissions/”)
let reqs = []
for (let submission of data) {
  let user_id = submission.user.id;
  reqs.push($.ajax(“/user/“ + user_id));
}
for (let req of reqs) {
  do_thing(await req);
}
```