perplexity

extend each answers a bit

1. **Define Django templates. How are they used to generate dynamic HTML content?**
   - Django templates are text strings that separate presentation logic from data, containing placeholders `{{ variable }}` and template tags `{% tag %}` for control flow.[1]
   - They generate dynamic HTML by compiling raw template code into an optimized internal representation, then rendering with context data via `Template.render(Context)`.
   - Example process: `t = Template("Hello {{name}}!"); c = Context({'name':'Fred'}); print(t.render(c))` outputs "Hello Fred!". Templates support any text format beyond HTML.[1]
2. **Explain template inheritance in Django with an example program.**
   - Template inheritance promotes DRY principle by defining a reusable base template with `{% block %}` sections that child templates override.[1]
   - Base template provides common layout (header, footer); children extend it using `{% extends "base.html" %}` and fill specific blocks.
   - Complete example:

```
<!-- base.html -->
<!DOCTYPE html>
<html>
<head><title>{% block title %}Default{% endblock %}</title></head>
<body>
    <div id="header">Common Header</div>
    {% block content %}{% endblock %}
    <div id="footer">Common Footer</div>
</body>
</html>

<!-- child.html -->
{% extends "base.html" %}
{% block title %}My Page{% endblock %}
{% block content %}
    <h1>Hello {{ name }}!</h1>
    <p>Dynamic content here.</p>
{% endblock %}
```

3. **What are context variables in Django templates? How are they passed to templates?**
   - Context variables are key-value pairs in a `Context` object, accessed in templates via `{{ key }}`, enabling dynamic content substitution.[1]

- Passed from views as dictionaries: `return render(request, 'template.html', {'name': 'Fred', 'age': 30})`; supports complex nested structures.
- Naming rules: Start with letter (A-Za-z), followed by letters/digits/underscores/dots; case-sensitive. Dot notation traverses: dicts, object attributes/methods (no args), list indices.[1]

4. **What are template filters in Django? Write a custom filter that capitalizes all text in a string.**
   - Built-in filters transform variables before display: `{{ text|lower|title }}`; chain multiple with `:` for arguments like `{{ bio|truncatewords:30 }}`.[1]
   - Custom filters created in `templatetags/` directory, registered with `@register.filter`; reload server after creation.
   - Full custom filter example:

```
# templatetags/custom_filters.py
from django import template
register = template.Library()

@register.filter(name='capitalize_all')
def capitalize_all(value):
    return value.upper() if value else ''

# In template: {{ my_text|capitalize_all }}
```

   Useful for consistent text formatting across templates.[1]

5. **How can we traverse the complex data structures using Dot Character (.)? Which data structures can be access using dot character? Explain short circuit logic.**
   - Dot (`.`) performs lookups in order: dictionary key (`person.name`), object attribute (`book.title`), method call (`user.get_full_name` no `()`/args), list index (`scores.0`).[1]
   - Supports nested traversal: `{{ book.publisher.country.name }}` for deeply nested model relationships.
   - Short-circuit logic: Stops at FIRST successful lookup type, preventing errors; e.g., if `foo` is dict with `'bar'` key, ignores object attribute `bar`.[1]

6. **Explain {% if %} template tag with an example.**
   - `{% if %}` tests variable truthiness (non-empty, non-false); supports optional `{% else %}`, logical `and/or/not`, nested tags.[1]
   - Example with logic:

```
{% if athlete_list and coach_list %}
    <p>Teams available!</p>
{% elif not cheerleader_list %}
    <p>No cheerleaders.</p>
{% else %}
    <p>Default case.</p>
{% endif %}
```

- Avoid ambiguous `and/or` chains; use nested `if` for clarity; always end with `{% endif %}`. [1]

7. **Explain {% for %} template tag with an example. List & explain magic forloop variable.**
   - `{% for %}` iterates sequences (lists, querysets): `{% for item in items %}{{ item }}{% empty %}No items{% endfor %}`; add `reversed` for backward iteration.[1]
   - Nested loops example:

```
{% for book in books %}
    Book: {{ book.title }}
    {% for author in book.authors.all %}
        - {{ author }}
    {% endfor %}
{% endfor %}
```

   - Magic `forloop` variables table:

Variable	Description	Example	
`forloop.counter`	1-based iteration (1,2,3...)	1st: 1	
`forloop.counter0`	0-based (0,1,2...)	1st: 0	
`forloop.revcounter`	Items remaining (5,4,3...)	Last: 1	
`forloop.revcounter0`	0-based reverse (4,3,2...)	Last: 0	
`forloop.first`	True on first iteration	`<b>{% if forloop.first %}First{% endif %}</b>`	
`forloop.last`	True on last	Special styling	
`forloop.parentloop`	Parent in nested loops	Access outer counter	[1]

8. **What are the disadvantages of directly embedding templates in code? How can this be overcome by separating the template file from the view?**
   - Embedding HTML strings in views leads to: frequent code changes for UI updates, violated separation (Python devs edit HTML, designers touch Python), serial development (can't work in parallel).[1]
   - Solution: Store templates in external `.html` files under `TEMPLATES['DIRS']`; Django's loader handles paths, caching, errors (`TemplateDoesNotExist`).
   - Benefits: Centralized management, no boilerplate `open/read/close`, designers edit independently, scalable for large projects.[1]
9. **How can templates be loaded using Django using get_template()? Alternatively how can the steps be reduced using render_to_response()?**
   - `from django.template.loader import get_template; t = get_template('mytemplate.html')`: Searches `TEMPLATES['DIRS']`, compiles, ready for `t.render(Context({}))`.[1]

- `render_to_response()` shortcut: `from django.shortcuts import render_to_response; return render_to_response('template.html', {'data': value})`.
- Comparison: `get_template` = 4 steps (load+context+render+HttpResponse); `render_to_response` = 1 line, auto-handles all, ideal for simple views.[1]

10. **Explain {% ifequal %} and {% ifnotequal %} template tags with an example.**
    - `{% ifequal var1 var2%}` compares exact equality (strings, numbers, variables); `{% endifequal %}` closes; supports `{% else %}`.[1]
    - `{% ifnotequal %}` inverse for inequality checks.
    - Example:

```
{% ifequal reorder "yes" %}
    <p>Ship the book!</p>
{% else %}
    <p>Hold shipment.</p>
{% endifequal %}

{% ifnotequal user_type "guest" %}
    Welcome back!
{% endifnotequal %}
```

    - Limits: No lists/dicts/booleans (use `{% if %}`); strings can be quoted literals.[1]
11. **What is use of locals()? When can we use it and when to avoid?**
    - `locals()` returns dict of all local variables in view scope: `return render(request, 'template.html', locals())` auto-passes `current_date`, `user_list`, etc.[1]
    - Use cases: Rapid prototyping, views with 10+ simple variables, quick debugging.
    - Avoid: Production (exposes internals like `request`, temp vars; namespace pollution; security risks with sensitive data; prefer explicit dict for control).[1]
12. **What is Object-Relational Mapping (ORM) in Django? How does it simplify database interaction?**
    - Django ORM maps Python classes (models) to DB tables/rows; fields to columns; provides high-level Python API instead of raw SQL.[1]
    - Simplifies: Auto-generates SQL for CRUD (`Model.objects.create()`=INSERT), relationships, filtering (`filter(published=True)`), aggregations, migrations.
    - Benefits: Database-agnostic (SQLite→PostgreSQL switch easy), secure (prevents SQL injection), Pythonic querysets (lazy evaluation, chaining).[1]
13. **Write a Django model for a book store application that includes fields for books, authors and publisher.**

```
from django.db import models

class Publisher(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
```

```
    country = models.CharField(max_length=50)
    website = models.URLField()

    def __str__(self):
        return self.name

class Author(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()

    def __str__(self):
        return f"{self.first_name} {self.last_name}"

class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher, on_delete=models.CASCADE)
    publication_date = models.DateField()

    def __str__(self):
        return self.title
```

14. **How are model relationships (one-to-one, many-to-one, many-to-many) handled in Django? Provide examples.**
    - One-to-one (`OneToOneField`): Unique link, e.g., `profile = models.OneToOneField(User, on_delete=models.CASCADE)`; access `user.profile`.[1]
    - Many-to-one/ForeignKey: `publisher = models.ForeignKey(Publisher, on_delete=models.CASCADE)`; reverse `publisher.book_set.all()`.
    - Many-to-many (`ManyToManyField`): Auto-join table, e.g., `authors = models.ManyToManyField(Author)`; access `book.authors.all()`, `author.book_set.all()`.[1]
15. **How do you configure a database in a Django project? Explain the key fields that need to be set in the settings.py file.**

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',  # sqlite3, mysql, oracle
        'NAME': BASE_DIR / 'db.sqlite3',  # or DB name
        'USER': 'myuser',
        'PASSWORD': 'mypassword',
        'HOST': 'localhost',  # or IP
        'PORT': '5432',  # default ports per engine
        'OPTIONS': {},  # engine-specific
    }
}
```

    - Key fields: `ENGINE` (backend), `NAME` (DB file/name), auth (`USER/PASSWORD`), connection (`HOST/PORT`). Test: `python manage.py dbshell`.[1]
16. **Describe the steps required to activate a Django app in a project. Write the commands needed to validate your models, generate SQL statements, and synchronize the**

database using Django's management commands.

- Step 1: Add `'books'` to `INSTALLED_APPS` in `settings.py`.[1]
- Step 2: Validate: `python manage.py check` (syntax/logic errors).
- Step 3: Generate migrations: `python manage.py makemigrations` (creates SQL in `migrations/`).
- Step 4: Apply/sync: `python manage.py migrate` (runs SQL, creates tables).
- Bonus: `python manage.py showmigrations` to check status.[1]

17. **How can you insert, update, retrieve and order data from a database using Django? Provide examples to demonstrate each operation.**
    - Insert: `p = Publisher(name='Apress'); p.save()` or `Publisher.objects.create(name='OReilly')`.[1]
    - Update: `p = Publisher.objects.get(id=1); p.name = 'New Name'; p.save()`; bulk: `Publisher.objects.filter(country='USA').update(website='new.com')`.
    - Retrieve: `Publisher.objects.all()`, `Publisher.objects.filter(name__startswith='A')`, `Publisher.objects.get(id=1)` (raises DoesNotExist).
    - Order: `Publisher.objects.order_by('name')` asc, `order_by('-name')` desc; multiple: `order_by('country', 'name')`.[1]
18. **Write a Django query to filter records from a database where the 'status' field is 'active'.**

```
# Basic exact match
Book.objects.filter(status='active')

# Chained/complex
active_books = Book.objects.filter(status='active', publication_date__year=2023).orde

# Count
Book.objects.filter(status='active').count()
```

19. **How can changes to a Database Schema be done using Django?**
    - Modify models (add/edit/delete fields/relationships).[1]
    - Generate migration: `python manage.py makemigrations` (detects changes, creates timestamped file).
    - Apply: `python manage.py migrate` (ALTER TABLE, etc.); `--fake` for manual sync.
    - Advanced: `python manage.py makemigrations --empty` custom migration; rollback `migrate app 0001`; inspect `sqlmigrate app 0002`.[1]

⁂

1. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/114655640/24f153fc-8bd2-4511-989c-6190e2436534/m3_fsd.pptx