Overview of the cheat sheet

React Cheatsheet

- **JSX**: Correctly described as a syntax extension. The example illustrates basic usage.
- **Components**: Both functional and class components are well represented.
- **Props**: Properly explained with examples demonstrating how to pass data.
- **State**: Clearly distinguishes between state in class components and functional components using hooks.
- **Hooks**: Introduces `useState` and `useEffect` accurately, with valid syntax.

Node.js Cheatsheet

- **Setting Up**: Correct commands to initialize a project and install Express.
- **Basic Server**: Provides a simple server example that is correct and functional.
- **Middleware**: Correctly describes how to use middleware.
- **Routing**: Demonstrates a basic GET route effectively.
- **Handling Requests**: Correctly shows how to access URL parameters.

Django Cheatsheet

- **Setting Up**: Accurate commands to create a project and run the server.
- **Create an App**: Correctly shows how to create an app within a project.
- **Models**: Accurately illustrates how to define a model.
- **Views**: Correctly describes how to create a view function and render a template.
- **URLs**: Properly explains URL routing for Django views.
- **Templates**: Provides a basic example of a Django template.

React Cheatsheet

1. JSX

- **Definition**: A syntax extension for JavaScript that resembles HTML.
- **Usage**: Allows you to write HTML-like code within JavaScript.

jsx

```jsx
const element = <h1>Hello, world!</h1>;
```

2. Components

Functional Components: Pure functions that return JSX.

jsx

```jsx
function Greeting() {
  return <h1>Hello!</h1>;
}
```

Class Components: ES6 classes extending `React.Component`.

jsx

```jsx
class Greeting extends React.Component {
  render() {
    return <h1>Hello!</h1>;
  }
}
```

3. Props

Definition: Short for "properties", used to pass data to components.

jsx

```jsx
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

<Greeting name="Alice" />
```

4. State

- **Definition**: A built-in object that allows components to create and manage their own data.

Class Component State:

jsx

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return <h1>{this.state.count}</h1>;
  }
}
```

- **Functional Component State with Hooks**:

```
jsx
Copy code
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  return <h1>{count}</h1>;
}
```

5. Hooks

useState: A hook that allows you to add state to functional components.

jsx

```
const [state, setState] = useState(initialState);
```

useEffect: A hook for handling side effects (e.g., data fetching).

jsx

```
useEffect(() => {
  // code to run on mount/update
  return () => {
    // cleanup code
  };
}, [dependencies]);
```

Node.js Cheatsheet

1. Setting Up

***Initialize Project**:*

bash
Copy code

```
npm init -y
```

Install Express:

bash

```
npm install express
```

2. Basic Server

javascript

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

3. Middleware

***Using Middleware**:*

javascript

```
app.use(express.json()); // Parse JSON bodies
```

4. Routing

javascript

```
app.get('/api/users', (req, res) => {
  res.json([{ name: 'Alice' }, { name: 'Bob' }]);
});
```

5. Handling Requests

***Accessing Parameters**:*

javascript

```
app.get('/api/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User ID: ${userId}`);
});
```

Django Cheatsheet

1. Setting Up

***Create a Project**:*

bash

```
django-admin startproject myproject
```

Run Server:
bash

```
python manage.py runserver
```

2. Create an App

bash

```
python manage.py startapp myapp
```

3. Models

python

```
from django.db import models

class Item(models.Model):
    name = models.CharField(max_length=100)
```

```
    price = models.DecimalField(max_digits=10, decimal_places=2)
```

4. Views

python

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html', {'items':
Item.objects.all()})
```

5. URLs

python

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
]
```

6. Templates

***Basic Template** (`home.html`):*

```
<!DOCTYPE html>
<html lang="en">
<head><title>Home</title></head>
<body>
  <h1>Items</h1>
  <ul>
    {% for item in items %}
      <li>{{ item.name }} - {{ item.price }}</li>
    {% endfor %}
```

```
  </ul>
</body>
</html>
```