# For the MCQ part tell me each and every component in such a way i don't miss anything and will score fully in the mcq

Based on your handwritten notes for the November 6th DSA exam, I'll create a comprehensive MCQ preparation guide covering all the topics you've listed. Let me organize this systematically to ensure you don't miss anything.

### DSA Exam - Complete MCQ Preparation Guide

### Overview of Your Exam

- **1 DSA exam** covering multiple technologies
- **1 OOPS** (Object-Oriented Programming System) section including SQL
- **MCQ Topics:** React, FastAPI, SQL, HTML, CSS, JavaScript, Shell Scripting, Python

Based on your notes mentioning "FAIS API" topics (Authentication - JWT, Pydantic, HTTP, Response Body, Backend Connection), I'll cover everything comprehensively.

### 1. React - Complete MCQ Coverage

### Core React Fundamentals

**What React Is:**

- React is a **JavaScript library** (not a framework) for building user interfaces[1]
- Developed by Facebook/Meta
- Used for creating **Single Page Applications (SPAs)**[2]
- Component-based architecture

**JSX (JavaScript XML):**

- Syntax extension for JavaScript that looks like HTML[1]
- JSX gets converted to JavaScript by **Babel**[1]
- Must have a **single root element**[1]

```
- Example: `const element = <h1>Hello {5+5}</h1>` renders as "Hello 10"[^1_1]
```

**Components:**

- Building blocks of React applications
- Two types: **Class components** and **Functional components**[3]
- Components accept **props** (properties) and return **JSX**[3]
- Components must start with capital letters

**Props vs State:**

- **Props:** Read-only data passed from parent to child components[4]
- **State:** Internal, mutable data managed within a component[4]
- State changes trigger re-renders
- Props are immutable; state is mutable

## React Hooks (Critical for MCQs)

**useState Hook:**

- Adds state to functional components[5]
- Returns array with [stateValue, setterFunction]
- Example: `const [count, setCount] = useState(0)`

**useEffect Hook:**

- Handles side effects in functional components[6] [5]
- Mimics lifecycle methods:
  - `componentDidMount`: `useEffect(() => {}, [])`
  - `componentDidUpdate`: `useEffect(() => {}, [dependency])`
  - `componentWillUnmount`: Return cleanup function[6]

**Other Important Hooks:**

- **useContext:** Access context values[7]
- **useCallback:** Memoize functions[5]
- **useMemo:** Memoize values[5]
- **useRef:** Access DOM elements or store mutable values

## React Lifecycle Methods (Class Components)

**Mounting Phase:**[5]

- `constructor()` - Initialize state
- `getDerivedStateFromProps()` - Sync state with props
- `render()` - Returns JSX
- `componentDidMount()` - After component mounts (API calls here)[5]

**Updating Phase:** [5]

- `shouldComponentUpdate()` - Control re-rendering (returns true/false)
- `getSnapshotBeforeUpdate()` - Capture values before update
- `componentDidUpdate()` - After component updates[5]

**Unmounting Phase:**

- `componentWillUnmount()` - Cleanup (remove listeners, cancel timers)[5]

### React Router (Important!)

- Library for handling **routing and navigation**[3]
- Components:
  - `<BrowserRouter>` - Wrapper component
  - `<Route>` - Defines URL-to-component mapping[3]
  - `<Link>` - Navigation without page reload

```
- `<Switch>` (v5) or `<Routes>` (v6) - Renders first matching route
```

**Event Handling:**

- Use **camelCase** syntax: `onClick`, not `onclick`[3]
- Pass function reference: `onClick={handleClick}`, not `onClick={handleClick()}`
- Synthetic events wrap browser's native events[3]

**Key Concepts for MCQs:**

- **Virtual DOM:** React creates in-memory representation for efficient updates[2] [3]
- **Reconciliation:** Process of updating real DOM from Virtual DOM
- **Keys:** Unique identifiers for list items (help React track changes)
- **Fragments:** `<React.Fragment>` or `<>` - Group elements without extra DOM nodes[3]
- **Controlled vs Uncontrolled Components:** Form inputs controlled by React state vs DOM[2]

## 2. FastAPI - Complete MCQ Coverage

### FastAPI Basics

**What is FastAPI?** [8] [9]

- Modern, high-performance **Python web framework** for building APIs
- Built on **Starlette** (ASGI framework) and **Pydantic** (data validation)[9]
- Uses **Python type hints** for automatic validation
- Generates **interactive API documentation** (Swagger UI, ReDoc) automatically[10] [8]

**Key Features:** [8] [10]

- **Asynchronous support:** Uses `async` and `await` for non-blocking operations
- **Automatic data validation:** Through Pydantic models
- **Type-driven development:** Leverages Python type hints
- **Built-in security:** OAuth2, JWT support
- **Dependency injection:** Built-in system for managing dependencies

## FastAPI Core Components

**Path Operations (Endpoints):** [10]

- Use decorators: `@app.get()`, `@app.post()`, `@app.put()`, `@app.delete()`
- Example:

```
@app.get("/items/{item_id}")
async def read_item(item_id: int):
    return {"item_id": item_id}
```

**Path Parameters vs Query Parameters:**

- **Path parameters:** Part of URL path `/items/{item_id}`
- **Query parameters:** After `?` in URL `/items?skip=0&limit=10`

## Pydantic Models (Very Important!)

**What is Pydantic?** [11] [12]

- Python library for **data validation** using type hints
- Automatically validates and serializes/deserializes data [11]
- FastAPI uses Pydantic for request/response models

**BaseModel:** [13]

- Base class for creating data models
- Example:

```
from pydantic import BaseModel, Field

class Item(BaseModel):
    name: str = Field(min_length=1, max_length=100)
    price: float = Field(gt=0)  # Greater than 0
    is_available: bool = True
```

**Field Validation:** [14] [15]

- `Field()` function adds constraints:

- `min_length`, `max_length` - String length
- `gt`, `ge`, `lt`, `le` - Numeric comparisons (greater than, etc.)
- `pattern` - Regular expression validation
- `...` - Required field
- `None` - Optional field

**Validators:** [14] [13]

- `@field_validator` - Validate individual fields
- `@model_validator` - Validate entire model (cross-field validation)
- Custom validation logic for complex rules

### HTTP Methods in FastAPI

**Request Body:**

- Data sent by client in POST/PUT requests
- Defined using Pydantic models
- Automatically parsed and validated

**Response Models:**

- Define structure of API responses
- Automatic serialization to JSON
- Type safety for outputs

## 3. HTTP Protocol - Complete Coverage

### HTTP Methods (Critical!)

**GET:** [16] [17] [18]

- **Retrieve data** from server
- Should be **idempotent** (same result on multiple calls)
- No request body (use query parameters instead)
- **Safe method** (doesn't modify server state)

**POST:** [17] [18] [16]

- **Create new resources**
- Includes request body with data
- **Not idempotent** (multiple calls may create multiple resources)
- Response typically returns created resource with 201 status

**PUT:** [18] [16] [17]

- **Update or replace** existing resource
- **Idempotent** (same result on multiple calls)
- Replaces entire resource with provided data
- If resource doesn't exist, may create it

**DELETE:** [16] [17] [18]

- **Remove resource** from server
- **Idempotent**
- Returns 204 No Content on success

**PATCH:**

- **Partial update** of resource
- Only modifies specified fields
- Not always idempotent

### HTTP Status Codes (Memorize These!)

**1xx - Informational:** [19] [20]

- `100 Continue` - Request received, continue sending
- `101 Switching Protocols`

**2xx - Successful:** [20] [21] [19]

- `200 OK` - Request successful, response contains data
- `201 Created` - Resource created successfully
- `204 No Content` - Success, but no content to return
- `206 Partial Content` - Range request successful

**3xx - Redirection:** [19] [20]

- `301 Moved Permanently`
- `302 Found` (Temporary redirect)
- `304 Not Modified` (Cached version is valid)

**4xx - Client Errors:** [21] [20] [19]

- `400 Bad Request` - Invalid syntax/data
- `401 Unauthorized` - Authentication required
- `403 Forbidden` - Authenticated but no permission
- `404 Not Found` - Resource doesn't exist
- `405 Method Not Allowed` - HTTP method not supported
- `422 Unprocessable Entity` - Validation error (common in FastAPI)

- `429 Too Many Requests` - Rate limit exceeded

**5xx - Server Errors:** [20] [21] [19]

- `500 Internal Server Error` - Generic server error
- `502 Bad Gateway` - Invalid response from upstream server
- `503 Service Unavailable` - Server overloaded/down
- `504 Gateway Timeout`

### Response Body:

- Contains actual data returned by server
- Usually in JSON format
- Includes status code, headers, and content

## 4. Authentication - JWT (JSON Web Token)

### JWT Structure [22] [23]

**Three Parts (separated by dots):**

1. **Header:** [22]

```
{
  "alg": "HS256",  // Signing algorithm
  "typ": "JWT"     // Token type
}
```

2. **Payload (Claims):** [22]

```
{
  "iss": "issuer",           // Issuer
  "sub": "subject/user_id",  // Subject (user)
  "iat": 1573849217,         // Issued at (timestamp)
  "exp": 1573849817,         // Expiration time
  "jti": "unique_id"         // JWT ID (prevents replay)
}
```

3. **Signature:** [23] [22]

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secret_key
)
```

### How JWT Works[24] [23] [22]

1. **User logs in** with credentials
2. **Server verifies** and generates JWT using secret key
3. **JWT returned** to client
4. **Client stores** JWT (localStorage, cookie, sessionStorage)
5. **Subsequent requests** include JWT in Authorization header: `Authorization: Bearer <token>`
6. **Server validates** JWT signature and expiry
7. **Access granted** if valid

### JWT Key Concepts for MCQs:

- **Stateless authentication:** Server doesn't store session[24]
- **Self-contained:** Token contains all user info
- **Symmetric vs Asymmetric:**[22]
  - **Symmetric (HS256):** Same secret key for signing and verifying
  - **Asymmetric (RS256, ES256):** Private key signs, public key verifies
- **Token expiry:** Set with `exp` claim to prevent indefinite validity
- **Payload is NOT encrypted:** Don't store sensitive data (can be decoded)[22]

## 5. SQL - Complete MCQ Coverage

### SQL Basics

#### What is SQL?[25]

- **Structured Query Language** for managing databases
- Used to store, retrieve, manipulate, and delete data
- Standard language across database systems (MySQL, PostgreSQL, etc.)

#### Basic Operations (CRUD):

- **CREATE:** `CREATE TABLE`, `CREATE DATABASE`
- **READ:** `SELECT` statements
- **UPDATE:** `UPDATE` statements
- **DELETE:** `DELETE` statements

### SQL SELECT Statement[26]

```
SELECT column1, column2 FROM table_name WHERE condition;
SELECT * FROM users;  -- Select all columns
SELECT DISTINCT city FROM customers;  -- Remove duplicates
```

### SQL Clauses[26]

**WHERE Clause:**

```
SELECT * FROM users WHERE age > 25;
SELECT * FROM products WHERE price BETWEEN 10 AND 50;
```

**ORDER BY:**

```
SELECT * FROM employees ORDER BY salary DESC;  -- Descending
SELECT * FROM employees ORDER BY name ASC;     -- Ascending (default)
```

**LIMIT/TOP:**

```
SELECT * FROM products LIMIT 10;  -- MySQL
SELECT TOP 10 * FROM products;    -- SQL Server
```

### SQL Operators[26]

**Comparison:** `=`, `!=`, `>`, `<`, `>=`, `<=`

**Logical:**

- `AND`, `OR`, `NOT`

```
SELECT * FROM users WHERE age > 18 AND city = 'NYC';
```

**IN Operator:**

```
SELECT * FROM users WHERE city IN ('NYC', 'LA', 'Chicago');
```

**LIKE (Pattern Matching):**

```
SELECT * FROM users WHERE name LIKE 'J%';   -- Starts with J
SELECT * FROM users WHERE name LIKE '%son'; -- Ends with son
SELECT * FROM users WHERE email LIKE '%@gmail.com';
```

**BETWEEN:**

```
SELECT * FROM products WHERE price BETWEEN 100 AND 500;
```

**IS NULL / IS NOT NULL:**

```
SELECT * FROM users WHERE phone IS NULL;
```

## SQL Aggregate Functions[26]

- **COUNT():** Count rows
- **SUM():** Sum numeric values
- **AVG():** Average value
- **MIN():** Minimum value
- **MAX():** Maximum value

```
SELECT COUNT(*) FROM orders;
SELECT AVG(salary) FROM employees;
SELECT MAX(price) FROM products;
```

**GROUP BY:**

```
SELECT city, COUNT(*) FROM users GROUP BY city;
SELECT category, AVG(price) FROM products GROUP BY category;
```

**HAVING Clause:**

```
SELECT city, COUNT(*) as count
FROM users
GROUP BY city
HAVING count > 5;  -- Filter after grouping
```

## SQL Joins (Very Important!)[27]

**INNER JOIN:**

```
SELECT orders.id, customers.name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.id;
```

- Returns only matching rows from both tables

**LEFT JOIN (LEFT OUTER JOIN):**

```
SELECT customers.name, orders.amount
FROM customers
```

```
LEFT JOIN orders ON customers.id = orders.customer_id;
```

- Returns all rows from left table, matching rows from right (NULL if no match)

**RIGHT JOIN:**

- Returns all rows from right table, matching from left

**FULL OUTER JOIN:**

- Returns all rows from both tables

**SELF JOIN:**

- Join table with itself

```
SELECT e1.name as Employee, e2.name as Manager
FROM employees e1
INNER JOIN employees e2 ON e1.manager_id = e2.id;
```

## SQL Subqueries [28] [29]

**Subquery in WHERE:**

```
SELECT name FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

**Subquery in FROM (Derived Table):**

```
SELECT * FROM (
  SELECT name, salary FROM employees WHERE dept = 'IT'
) AS it_employees;
```

**Subquery with IN:**

```
SELECT name FROM customers
WHERE id IN (SELECT customer_id FROM orders WHERE amount > 1000);
```

**Subquery with JOIN:** [29]

```
SELECT s.name, sub.total_orders
FROM students s
INNER JOIN (
  SELECT student_id, COUNT(*) as total_orders
  FROM enrollments GROUP BY student_id
) sub ON s.id = sub.student_id;
```

### SQL Constraints[26]

- **PRIMARY KEY:** Unique identifier for rows
- **FOREIGN KEY:** Links tables together
- **UNIQUE:** Ensures all values are different
- **NOT NULL:** Column must have a value
- **CHECK:** Validates data (e.g., age > 0)
- **DEFAULT:** Sets default value if none provided

```
CREATE TABLE users (
  id INT PRIMARY KEY,
  email VARCHAR(100) UNIQUE NOT NULL,
  age INT CHECK (age >= 18),
  status VARCHAR(20) DEFAULT 'active'
);
```

## 6. HTML - Complete MCQ Coverage

### HTML Basics

**DOCTYPE:**[30]

- HTML5 doctype: `<!DOCTYPE html>`
- Tells browser which HTML version to use

**HTML Structure:**

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <!-- Content here -->
  <script src="script.js"></script>
</body>
</html>
```

**Common HTML Tags:**

```
- `<h1>` to `<h6>` - Headings
```

- `<p>` - Paragraph
- `<a href="">` - Hyperlink[30]
- `<img src="" alt="">` - Image

- `<div>` - Container (block-level)
- `<span>` - Container (inline)

```
- `<ul>`, `<ol>`, `<li>` - Lists
```

```
- `<table>`, `<tr>`, `<td>`, `<th>` - Tables
```

### HTML Forms (Important!)

**Form Element:** [31] [32]

```
<form action="/submit" method="POST">
  <!-- Form inputs here -->
</form>
```

**Input Types:** [32] [33]

- `text` - Single-line text (default)
- `password` - Hidden text
- `email` - Email validation
- `number` - Numeric input
- `checkbox` - Multiple selections
- `radio` - Single selection from group
- `date` - Date picker
- `file` - File upload
- `submit` - Submit button
- `reset` - Reset form
- `button` - Generic button
- `hidden` - Hidden field
- `tel` - Telephone number
- `url` - URL validation
- `color` - Color picker
- `range` - Slider
- `search` - Search field

**Input Attributes:** [31]

- `value` - Initial/default value
- `placeholder` - Hint text
- `required` - Must be filled

- `readonly` - Cannot be modified
- `disabled` - Cannot interact
- `min`, `max` - Numeric limits
- `minlength`, `maxlength` - String length limits
- `pattern` - Regex validation
- `multiple` - Allow multiple values (file, email)
- `checked` - Default checked (checkbox/radio)
- `name` - Identifies field when submitted
- `id` - Unique identifier

### HTML5 Semantic Elements:[34]

- `<header>` - Page/section header
- `<nav>` - Navigation links
- `<main>` - Main content
- `<article>` - Self-contained content
- `<section>` - Thematic grouping
- `<aside>` - Sidebar content
- `<footer>` - Page/section footer[30]

**External Resources:**

- CSS: `<link rel="stylesheet" href="style.css">`[30]

```
- JavaScript: `<script src="script.js"></script>`[^1_30]
```

## 7. CSS - Complete MCQ Coverage

### CSS Basics

**What is CSS?**

- **Cascading Style Sheets** for styling HTML
- Controls layout, colors, fonts, spacing

**CSS Selectors:**

```
/* Element selector */
p { color: blue; }

/* Class selector */
.container { width: 100%; }
```

```
/* ID selector */
#header { background: gray; }

/* Attribute selector */
input[type="text"] { border: 1px solid; }

/* Pseudo-class */
a:hover { color: red; }
```

### CSS Box Model[35]

Every HTML element is a box with:

1. **Content** - The actual content
2. **Padding** - Space between content and border
3. **Border** - Edge around padding
4. **Margin** - Space outside border

```
div {
  width: 300px;
  padding: 20px;
  border: 5px solid black;
  margin: 10px;
}
/* Total width = 300 + 40 + 10 + 20 = 370px */
```

### CSS Position Property[30]

- `static` - Default, normal flow
- `relative` - Positioned relative to normal position
- `absolute` - Positioned relative to nearest positioned ancestor
- `fixed` - Positioned relative to viewport
- `sticky` - Hybrid of relative and fixed

### CSS Flexbox (Important!)[36] [37] [38]

**Container Properties:**

```
.container {
  display: flex;
  flex-direction: row;      /* row, column, row-reverse, column-reverse */
  justify-content: center;  /* flex-start, flex-end, space-between, space-around */
  align-items: center;      /* flex-start, flex-end, center, stretch */
  flex-wrap: wrap;          /* nowrap, wrap, wrap-reverse */
}
```

**Item Properties:**

```
.item {
  flex-grow: 1;     /* Growth factor */
  flex-shrink: 1;   /* Shrink factor */
  flex-basis: 200px; /* Initial size */
  flex: 1 1 200px;  /* Shorthand */
}
```

**Key Flexbox Concepts:** [39] [40]

- **One-dimensional layout** (either row OR column) [39]
- Main axis vs cross axis
- Items can grow/shrink to fill space
- Great for responsive designs

## CSS Grid (Important!) [40] [39]

**Container Properties:**

```
.grid-container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);  /* 3 equal columns */
  grid-template-rows: auto auto;
  gap: 10px;                              /* Space between items */
}
```

**Item Properties:**

```
.grid-item {
  grid-column: 1 / 3;   /* Span columns 1-2 */
  grid-row: 1 / 2;      /* Span row 1 */
}
```

**Grid vs Flexbox:** [41] [40] [39]

- **Grid:** Two-dimensional layout (rows AND columns) [40] [39]
- **Flexbox:** One-dimensional layout (row OR column) [39] [40]
- Use Grid for page layouts, Flexbox for components

## CSS Display Property:

- `block` - Takes full width, starts new line
- `inline` - Takes only necessary width, no line break
- `inline-block` - Like inline but can set width/height
- `none` - Element not displayed
- `flex` - Flexbox container

- `grid` - Grid container

### CSS Units:

- `px` - Pixels (fixed)
- `%` - Percentage of parent
- `em` - Relative to parent font size
- `rem` - Relative to root font size
- `vh/vw` - Viewport height/width percentage

### CSS z-index:[30]

- Controls stacking order of positioned elements
- Higher values appear on top
- Only works with positioned elements (not static)

**CSS Visibility:**[30]

- `visibility: hidden` - Hides but takes space
- `display: none` - Hides and removes from flow

## 8. JavaScript - Complete MCQ Coverage

### JavaScript Basics

**Variables:**

- `var` - Function-scoped, can be redeclared
- `let` - Block-scoped, cannot be redeclared
- `const` - Block-scoped, cannot be reassigned

**Data Types:**

- Primitive: `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, `bigint`
- Reference: `object`, `array`, `function`

### ES6 Features (Important!)[42] [43]

**Arrow Functions:**[44] [45]

```
// Traditional function
function add(a, b) {
  return a + b;
}

// Arrow function
```

```
const add = (a, b) => a + b;

// With single parameter (parentheses optional)
const square = x => x * x;

// With multiple statements
const multiply = (a, b) => {
  const result = a * b;
  return result;
};
```

**Key Points:**

- Shorter syntax
- No `this` binding (inherits from parent scope)
- Cannot be used as constructors
- Implicit return for single expressions

**Template Literals:**

```
const name = "John";
const greeting = `Hello, ${name}!`;  // Use backticks
const multiline = `Line 1
Line 2`;
```

**Destructuring:**

```
// Array destructuring
const [a, b] = [1, 2];

// Object destructuring
const { name, age } = { name: "John", age: 30 };
```

**Spread Operator:**

```
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];  // [1, 2, 3, 4, 5]

const obj1 = { a: 1, b: 2 };
const obj2 = { ...obj1, c: 3 };  // { a: 1, b: 2, c: 3 }
```

**Promises:**[44]

```
const promise = new Promise((resolve, reject) => {
  if (success) {
    resolve(data);
  } else {
    reject(error);
  }
```

```
});

promise
  .then(result => console.log(result))
  .catch(error => console.log(error));
```

**Async/Await:**

```
async function fetchData() {
  try {
    const response = await fetch(url);
    const data = await response.json();
    return data;
  } catch (error) {
    console.error(error);
  }
}
```

## JavaScript DOM Manipulation

**Selecting Elements:**

```
document.getElementById('id')
document.getElementsByClassName('class')
document.getElementsByTagName('div')
document.querySelector('.class')
document.querySelectorAll('.class')
```

**Event Handling:**

```
element.addEventListener('click', function(event) {
  console.log('Clicked!');
});
```

**Common Events:**

- `click`, `dblclick`
- `mouseover`, `mouseout`
- `keydown`, `keyup`, `keypress`
- `submit`, `change`, `input`
- `load`, `DOMContentLoaded`

### JavaScript Array Methods:

```
arr.push(item)         // Add to end
arr.pop()              // Remove from end
arr.shift()            // Remove from beginning
arr.unshift(item)      // Add to beginning
arr.map(fn)            // Transform each element
arr.filter(fn)         // Filter elements
arr.reduce(fn, init)   // Reduce to single value
arr.forEach(fn)        // Loop through elements
arr.find(fn)           // Find first matching element
arr.includes(item)     // Check if contains item
```

### JavaScript Object Methods:

```
Object.keys(obj)       // Array of keys
Object.values(obj)     // Array of values
Object.entries(obj)    // Array of [key, value] pairs
Object.assign(target, source)  // Copy properties
```

## 9. Shell Scripting - Complete MCQ Coverage

### Shell Scripting Basics[46] [47]

#### What is Shell?

- Command-line interface for interacting with OS
- Common shells: Bash, Zsh, Ksh, Csh

**Shebang:**

```
#!/bin/bash
# Tells OS which interpreter to use
```

**Variables:**[46]

```
name="John"
echo "Hello, $name"
echo "Hello, ${name}"

# Read input
read -p "Enter name: " username
```

## Shell Commands[47] [46]

**File Operations:**

- `ls` - List files
- `cd` - Change directory
- `pwd` - Print working directory
- `mkdir` - Create directory
- `rm` - Remove files/directories
- `cp` - Copy files
- `mv` - Move/rename files
- `cat` - Display file contents
- `touch` - Create empty file
- `chmod` - Change file permissions[47]

**File Permissions:**[47]

- `chmod 755 file` - rwxr-xr-x
- `chmod +x file` - Add execute permission

**Text Processing:**

- `grep` - Search text patterns
- `sed` - Stream editor
- `awk` - Text processing
- `sort` - Sort lines
- `uniq` - Remove duplicates
- `wc` - Word count

## Shell Script Control Structures[46] [47]

**If Statement:**

```
if [ $age -gt 18 ]; then
  echo "Adult"
elif [ $age -eq 18 ]; then
  echo "Just 18"
else
  echo "Minor"
fi
```

**Comparison Operators:**

- `-eq` - Equal to
- `-ne` - Not equal

- `-gt` - Greater than
- `-lt` - Less than
- `-ge` - Greater than or equal
- `-le` - Less than or equal

**String Comparisons:**

- `=` - Equal
- `!=` - Not equal
- `-z` - Empty string
- `-n` - Not empty

**Loops:**

```
# For loop
for i in 1 2 3 4 5; do
  echo $i
done

# While loop
while [ $count -lt 10 ]; do
  echo $count
  count=$((count + 1))
done

# Until loop
until [ $count -eq 10 ]; do
  echo $count
  count=$((count + 1))
done
```

**Case Statement:**

```
case $choice in
  1)
    echo "Option 1"
    ;;
  2)
    echo "Option 2"
    ;;
  *)
    echo "Invalid"
    ;;
esac
```

**Functions:**

```
function greet() {
  echo "Hello, $1"
}
```

```
greet "John"
```

**Redirections:** [47]

- `>` - Redirect output (overwrite)
- `>>` - Redirect output (append)
- `<` - Redirect input
- `2>` - Redirect stderr
- `&>` - Redirect stdout and stderr

### Special Variables:

- `$0` - Script name
- `$1, $2, ...` - Arguments
- `$#` - Number of arguments
- `$@` - All arguments as separate strings
- `$?` - Exit status of last command
- `$$` - Process ID

## 10. Python - Complete MCQ Coverage

### Python Basics [48] [49]

**Python Characteristics:**

- **High-level** programming language
- **Interpreted** (not compiled)
- **Dynamically typed**
- **Object-oriented**
- Created by **Guido van Rossum**

**Variables and Data Types:**

```
# Dynamic typing
x = 10         # int
x = "hello"    # str
x = [1, 2, 3]  # list

# Multiple assignment
a, b, c = 1, 2, 3
```

**Data Types:** [49]

- `int`, `float`, `complex` - Numbers
- `str` - String
- `list` - Mutable ordered sequence
- `tuple` - Immutable ordered sequence
- `dict` - Key-value pairs
- `set` - Unordered unique elements
- `bool` - True/False
- `NoneType` - None

### Python Operators:

**Arithmetic:** `+`, `-`, `*`, `/`, `//` (floor division), `%`, `**` (power)

**Comparison:** `==`, `!=`, `>`, `<`, `>=`, `<=`

**Logical:** `and`, `or`, `not`

**Membership:** `in`, `not in`

**Identity:** `is`, `is not`

### Python Control Flow:[50]

**If Statement:**

```
if condition:
    # code
elif another_condition:
    # code
else:
    # code
```

**Loops:**

```
# For loop
for i in range(10):
    print(i)

for item in list:
    print(item)

# While loop
while condition:
    # code
```

**Break and Continue:**

- `break` - Exit loop

- `continue` - Skip to next iteration

### Python Functions:

```
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"

# Lambda functions
square = lambda x: x ** 2
```

### Python Data Structures:

#### Lists:[48]

```
lst = [1, 2, 3]
lst.append(4)       # Add to end
lst.insert(0, 0)    # Insert at position
lst.pop()           # Remove last
lst.remove(2)       # Remove value
lst.sort()          # Sort in place
```

#### Dictionaries:

```
d = {"name": "John", "age": 30}
d["city"] = "NYC"   # Add key
d.get("name")       # Get value (safe)
d.keys()            # Get keys
d.values()          # Get values
d.items()           # Get key-value pairs
```

#### Sets:[48]

```
s = {1, 2, 3}
s.add(4)            # Add element
s.remove(2)         # Remove element
s1.union(s2)        # Union
s1.intersection(s2) # Intersection
```

### Python Modules:[48]

#### Importing:

```
import module_name
from module_name import function_name
from module_name import *
import module_name as alias
```

#### Common Modules:

- `os` - Operating system interface
- `sys` - System-specific parameters
- `math` - Mathematical functions
- `random` - Random number generation
- `datetime` - Date and time
- `json` - JSON encoding/decoding
- `re` - Regular expressions

### Python File Operations:

```
# Read file
with open('file.txt', 'r') as f:
    content = f.read()

# Write file
with open('file.txt', 'w') as f:
    f.write("Hello")

# Append
with open('file.txt', 'a') as f:
    f.write("World")
```

### Python Exception Handling:

```
try:
    # code that may raise exception
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
except Exception as e:
    print(f"Error: {e}")
finally:
    print("Always executes")
```

### Python List Comprehensions:

```
# Basic
squares = [x**2 for x in range(10)]

# With condition
evens = [x for x in range(20) if x % 2 == 0]

# Nested
matrix = [[i*j for j in range(3)] for i in range(3)]
```

### Python Shell Commands:[51] [52] [53]

**Using os module:**

```
import os
os.system('ls -l')          # Execute command
os.getcwd()                 # Current directory
os.chdir('/path')           # Change directory
os.listdir('.')             # List files
```

**Using subprocess:**[51]

```
import subprocess
result = subprocess.run(['ls', '-l'], capture_output=True, text=True)
print(result.stdout)
```

## 11. Backend Connection & REST API Architecture[54] [55] [56]

### REST API Architectural Constraints:[56]

1. **Client-Server Architecture:**[56]
   - Separation of concerns
   - Client handles UI, server handles data
   - Independent development and scaling
2. **Statelessness:**[55]
   - Each request contains all necessary information
   - No server-side session storage
   - Improves scalability
3. **Cacheability:**[55]
   - Responses marked as cacheable/non-cacheable
   - Improves performance
   - Reduces server load
4. **Uniform Interface:**
   - Standardized communication
   - Resource-based URLs
   - Standard HTTP methods
5. **Layered System:**[55] [56]
   - Client can't tell if connected directly to server
   - Intermediaries (proxies, load balancers) possible

    - Improves scalability and security
6. **Code on Demand (Optional):**
    - Server can send executable code to client
    - JavaScript, applets, etc.

### REST API Best Practices:[57] [58] [59]

**Resource Naming:**

- Use **nouns**, not verbs: `/users`, not `/getUsers`
- Use **plural** nouns: `/users/123`
- Use **hierarchical** structure: `/users/123/orders`
- Use **lowercase** with hyphens: `/user-profiles`

**HTTP Methods Usage:**

- GET for retrieval (read-only)
- POST for creation
- PUT for full update/replace
- PATCH for partial update
- DELETE for removal

**API Versioning:**[58] [59]

- Include version in URL: `/v1/users`, `/v2/users`
- Prevents breaking changes for existing clients
- Maintains backward compatibility

**Error Handling:**[58]

- Return appropriate status codes
- Include structured error messages

```
{
  "error": "ValidationError",
  "message": "Email format is invalid",
  "status": 400
}
```

**Pagination:**[57]

```
GET /users?page=2&limit=20
```

**Filtering and Sorting:**[57]

```
GET /products?category=electronics&sort=price&order=asc
```

**Rate Limiting:** [57]

- Prevent abuse
- Return 429 status code when exceeded

### Additional Tips for MCQ Preparation:

### Study Strategy:

1. **Understand concepts deeply** - Don't just memorize
2. **Practice with examples** - Code snippets help
3. **Focus on comparisons** - "X vs Y" questions are common
4. **Know syntax** - Exact syntax matters in MCQs
5. **Remember default values** - Often tested

### Common MCQ Patterns:

- **Definition questions:** "What is X?"
- **Syntax questions:** "Which is correct syntax?"
- **Output questions:** "What will this code output?"
- **Comparison questions:** "What's the difference between X and Y?"
- **Best practice questions:** "What's the best way to...?"
- **Error identification:** "What's wrong with this code?"

### Time Management:

- Read questions carefully
- Eliminate obviously wrong answers first
- Don't spend too long on one question
- Mark difficult questions and return later
- Review all answers if time permits

### Key Areas to Focus:

1. **React:** Hooks, lifecycle, state vs props
2. **FastAPI:** Pydantic validation, async, decorators
3. **HTTP:** Status codes, methods, REST principles
4. **JWT:** Structure, workflow, security

5. **SQL:** Joins, subqueries, aggregations
6. **HTML:** Forms, input types, semantic tags
7. **CSS:** Flexbox, Grid, Box Model
8. **JavaScript:** ES6 features, async/await, DOM
9. **Shell:** Control structures, commands, permissions
10. **Python:** Data structures, control flow, modules

This comprehensive guide covers all the topics from your syllabus. Focus on understanding the concepts and their practical applications. Practice with real code examples and test yourself with sample MCQs. Good luck with your exam on November 6th!

⁂


1. https://www.gyansetu.in/blog/react-mcqs-with-answers/
2. https://interviewkickstart.com/blogs/interview-questions/react-js-mcqs-answers
3. https://www.sanfoundry.com/react-js-mcq-multiple-choice-questions/
4. https://www.interviewbit.com/react-mcq/
5. https://www.geeksforgeeks.org/reactjs/reactjs-lifecycle-components/
6. https://www.geeksforgeeks.org/reactjs/mimicking-lifecycle-methods-with-hooks-in-react/
7. https://www.freecodecamp.org/news/react-lifecycle-methods-and-hooks-for-beginners/
8. https://www.geeksforgeeks.org/python/fastapi-introduction/
9. https://deepnote.com/blog/ultimate-guide-to-fastapi-library-in-python
10. https://pyimagesearch.com/2025/03/17/getting-started-with-python-and-fastapi-a-complete-beginners-guide/
11. https://docs.pydantic.dev/latest/concepts/models/
12. https://www.netguru.com/blog/data-validation-pydantic
13. https://realpython.com/python-pydantic/
14. https://www.datacamp.com/tutorial/pydantic
15. https://betterstack.com/community/guides/scaling-python/pydantic-explained/
16. https://www.linkedin.com/pulse/http-methods-explained-understanding-get-post-put-delete-shaaban-nqyyf
17. https://apidog.com/blog/http-methods/
18. https://api7.ai/learning-center/api-101/http-methods-in-apis
19. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status
20. https://learn.microsoft.com/en-us/troubleshoot/developer/webapps/iis/health-diagnostic-performance/http-status-code
21. https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
22. https://www.loginradius.com/blog/engineering/guide-to-jwt
23. https://www.geeksforgeeks.org/web-tech/json-web-token-jwt/
24. https://auth0.com/learn/json-web-tokens

25. https://www.interviewbit.com/sql-mcq/
26. https://www.sanfoundry.com/1000-sql-questions-answers/
27. https://www.w3schools.com/sql/sql_join.asp
28. https://mode.com/sql-tutorial/sql-sub-queries/
29. https://www.geeksforgeeks.org/sql/sql-subquery/
30. https://cbmceindia.com/blog/details/html-css-mcqs-with-answers/
31. https://www.w3schools.com/html/html_form_attributes.asp
32. https://www.w3schools.com/tags/att_input_type.asp
33. https://www.geeksforgeeks.org/html/html-input-type-attribute/
34. https://testbook.com/objective-questions/mcq-on-html-css--600137280141c5473516d8cb
35. https://www.w3schools.com/css/css_boxmodel.asp
36. https://www.w3schools.com/css/css3_flexbox.asp
37. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout/Basic_concepts_of_flexbox
38. https://css-tricks.com/snippets/css/a-guide-to-flexbox/
39. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout/Relationship_of_grid_layout_with_other_layout_methods
40. https://www.geeksforgeeks.org/css/comparison-between-css-grid-css-flexbox/
41. https://www.simplilearn.com/tutorials/css-tutorial/css-grid-vs-flexbox
42. https://www.w3schools.com/js/js_es6.asp
43. https://scribbler.live/2024/04/12/ES6-Features-with-Examples.html
44. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
45. https://www.w3schools.com/js/js_arrow_function.asp
46. https://www.sanfoundry.com/linux-shell-programming-mcq-1/
47. https://testbook.com/objective-questions/mcq-on-shell-programming--61f8d371669162b150866194
48. https://www.sanfoundry.com/1000-python-questions-answers/
49. https://www.interviewbit.com/python-mcq/
50. https://www.geeksforgeeks.org/computer-science-fundamentals/control-flow-statements-in-programming/
51. https://www.simplilearn.com/shell-scripting-in-python-article
52. https://www.freecodecamp.org/news/run-python-script-how-to-execute-python-shell-commands-in-terminal/
53. https://www.geeksforgeeks.org/python/executing-shell-commands-with-python/
54. https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design
55. https://aws.amazon.com/what-is/restful-api/
56. https://masteringbackend.com/hubs/backend-engineering/rest-api-architectural-constraints
57. https://www.designgurus.io/blog/8-rest-api-interview-questions-every-developer-should-know
58. https://www.browserless.io/blog/rest-api-interview-questions-answers-2025
59. https://www.interviewbit.com/rest-api-interview-questions/

60. https://www.w3schools.com/tags/ref_httpmessages.asp
61. https://react.dev/reference/react/hooks
62. https://www.sigmacomputing.com/blog/advanced-sql-joins
63. https://www.w3schools.com/html/html_form_attributes_form.asp
64. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/input
65. https://www.w3schools.com/html/html_form_input_types.asp
66. https://www.mytectra.com/tutorials/javascript/es6-features-let-const-arrow-functions-etc
67. https://www.geeksforgeeks.org/javascript/es6-arrow-function/
68. https://www.dofactory.com/html/input-attributes
69. https://mrcet.com/downloads/digital_notes/HS/OOP_10122018.pdf
70. https://www.scribd.com/document/927263566/Dsa
71. https://www.youtube.com/watch?v=RvYKa5S6BRY
72. https://www.merge.dev/blog/rest-api-interview-questions
73. https://www.youtube.com/watch?v=EsGM1XKSO-g
74. https://www.redhat.com/en/blog/python-scripting-intro
75. https://www.turing.com/interview-questions/rest-api
76. https://www.geeksforgeeks.org/dsa/control-structures-in-programming-languages/
77. https://roadmap.sh/questions/rest-api
78. https://www.tutorialspoint.com/instance-control-flow-in-java
79. https://www.interviewbit.com/blog/python-commands/
80. https://www.simplilearn.com/rest-api-interview-questions-answers-article
81. https://www.w3schools.com/dsa/dsa_intro.php
82. Final_Syllabus.jpg
83. https://www.scribd.com/document/878924156/React-Js-Mcqs
84. https://www.sanfoundry.com/1000-database-management-system-questions-answers/
85. https://fastapi.tiangolo.com/tutorial/
86. https://www.w3schools.com/sql/sql_quiz.asp
87. https://fastapi.tiangolo.com
88. https://www.placementpreparation.io/mcq/sql/
89. https://www.w3schools.com/react/react_quiz.asp
90. https://www.freecodecamp.org/news/fastapi-quickstart/
91. https://www.youtube.com/watch?v=_zMh8BQ3twM
92. https://ravirathore.com/reactjs/reactjs-mcq-questions-2024-for-developers/
93. https://www.youtube.com/watch?v=rvFsGRvj9jo
94. https://www.sanfoundry.com/1000-javascript-questions-answers/
95. https://www.scribd.com/document/288294464/HTML-CSS-JAVASCRIPT-MCQ
96. https://www.interviewbit.com/javascript-mcq/
97. https://www.igmguru.com/blog/python-mcqs

98. https://www.interviewbit.com/html-mcq/
99. https://www.geeksforgeeks.org/python/python-multiple-choice-questions/
100. https://pub.towardsai.net/pydantic-a-data-engineers-guide-to-data-validation-ca88a8d9bb2f
101. https://www.w3schools.com/html/html_quiz.asp
102. https://www.koenig-solutions.com/bash-shell-scripting-quiz-questions-answers
103. https://www.simplilearn.com/tutorials/nodejs-tutorial/jwt-authentication
104. https://www.echoapi.com/blog/understanding-and-using-http-methods-get-post-put-delete/
105. https://www.youtube.com/watch?v=XvFmUE-36Kc&vl=en
106. https://www.w3schools.com/tags/ref_httpmethods.asp
107. https://jwt.io/introduction
108. https://restfulapi.net/rest-architectural-constraints/
109. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods
110. https://www.youtube.com/watch?v=mbsmsi7l3r4
111. https://blog.wahab2.com/api-architecture-best-practices-for-designing-rest-apis-bf907025f5f
112. https://www.geeksforgeeks.org/node-js/different-kinds-of-http-requests/
113. https://frontegg.com/blog/jwt-authentication
114. https://www.alooba.com/skills/concepts/sql-19/joins-and-subqueries/
115. https://umbraco.com/knowledge-base/http-status-codes/
116. https://retool.com/blog/the-react-lifecycle-methods-and-hooks-explained
117. https://www.w3schools.com/react/react_lifecycle.asp
118. https://stackoverflow.com/questions/16317814/sql-subquery-in-aggregate-function
119. https://restfulapi.net/http-status-codes/
120. https://legacy.reactjs.org/docs/hooks-overview.html
121. https://forumde.in/oracle_live_sql