

COMPLETE MERN STACK WEB DEVELOPMENT COURSE:

TOPICS YOU WILL LEARN IN THIS COURSE →

1. HOW THE INTERNET WORKS?
2. HOW WEBSITES WORK?
3. INTRODUCTION TO HTML.
4. INTRODUCTION TO CSS.
5. INTERMEDIATE CSS.
6. BOOTSTRAP 4
7. INTRODUCTION TO JAVASCRIPT
8. DOCUMENT OBJECT MODEL (DOM)
9. ADVANCED JAVASCRIPT AND DOM MANIPULATION
10. JQUERY
11. NODE.JS
12. EXPRESS.JS
13. APPLICATION PROGRAMMING INTERFACE
14. EJS
15. DATABASES
16. SQL
17. MONGODB
18. MONGOOSE
19. DEPLOYING YOUR WEB APPLICATION

20.BUILD YOUR OWN RESTFUL API FROM SCRATCH

21.AUTHENTICATION & SECURITY

22.REACT.JS

23.DESIGN SCHOOL 101

24.PRE REQUISITE

25.PLACEMENT QUESTIONS:

BUILDING DYNAMIC WEB APPLICATIONS WITH HTML, CSS, JAVASCRIPT, BOOTSTRAP 5, REACT.JS, NODE.JS, MONGODB, MONGOOSE, AND EXPRESS.JS

INTRODUCTION

To Access Projects Notes and Exercises : <https://drive.google.com/drive/folders/1vrj6mhOGJIRVsj-gChb4FxG1eqA7DmZC?usp=sharing>

IN THE EVER-EVOLVING LANDSCAPE OF WEB DEVELOPMENT, ONE STACK HAS RISEN TO PROMINENCE FOR ITS VERSATILITY, POWER, AND EFFICIENCY - THE MERN STACK. THE MERN STACK IS A COMPREHENSIVE COMBINATION OF TECHNOLOGIES THAT EMPOWERS DEVELOPERS TO CREATE MODERN, DYNAMIC, AND DATA-DRIVEN WEB APPLICATIONS. FROM CRAFTING ENGAGING USER INTERFACES TO HANDLING COMPLEX SERVER-SIDE LOGIC AND DATA STORAGE, THE MERN STACK EQUIPS YOU WITH THE TOOLS TO TACKLE IT ALL.

IN THIS EBOOK, WE EMBARK ON A JOURNEY THROUGH THE INTRICACIES OF THE MERN STACK. WHETHER YOU'RE AN ASPIRING DEVELOPER LOOKING TO

DIVE INTO WEB DEVELOPMENT FOR THE FIRST TIME, OR AN EXPERIENCED PROGRAMMER SEEKING TO ENHANCE YOUR SKILL SET, THIS RESOURCE IS TAILORED TO MEET YOUR NEEDS.

UNPACKING THE ACRONYM

THE MERN STACK IS AN AMALGAMATION OF FOUR ESSENTIAL COMPONENTS:

First To Access Projects and Exercises : <https://drive.google.com/drive/folders/1oZvE3TU7Kij-gRYgzigqBEAF0iUQcHUk>

To Access Notes : <https://t.me/developperHome>

And Here We Go:

M - MONGODB: THIS NOSQL DATABASE IS THE FOUNDATION OF YOUR APPLICATION'S DATA MANAGEMENT. YOU'LL LEARN TO DESIGN EFFICIENT DATABASE STRUCTURES AND HARNESS THE POWER OF MONGODB THROUGH THE MONGOOSE ODM (OBJECT DATA MODELING) LIBRARY.

E - EXPRESS.JS: AS A ROBUST WEB APPLICATION FRAMEWORK, EXPRESS.JS SIMPLIFIES SERVER-SIDE DEVELOPMENT. IT PROVIDES AN ARRAY OF FEATURES AND MIDDLEWARE THAT FACILITATE ROUTING, REQUEST HANDLING, AND MORE.

R - REACT.JS: REACT.JS IS A CUTTING-EDGE LIBRARY FOR BUILDING DYNAMIC, INTERACTIVE USER INTERFACES. WE'LL EXPLORE REACT'S COMPONENT-BASED ARCHITECTURE AND GUIDE YOU THROUGH CREATING RESPONSIVE, CLIENT-SIDE APPLICATIONS.

N - NODE.JS: NODE.JS IS A SERVER-SIDE RUNTIME ENVIRONMENT THAT ALLOWS JAVASCRIPT TO BE EXECUTED ON THE SERVER. YOU'LL HARNESS THE POWER OF NODE.JS TO BUILD THE SERVER THAT SERVES AS THE BACKBONE OF YOUR MERN APPLICATION.

WHAT TO EXPECT

IN THE FOLLOWING CHAPTERS, WE'LL BREAK DOWN EACH COMPONENT OF THE MERN STACK, GUIDING YOU THROUGH THE ESSENTIALS AND HELPING YOU BUILD A STRONG FOUNDATION IN WEB DEVELOPMENT. WE'LL COVER:

HTML AND CSS BASICS TO ENSURE YOU CAN CRAFT BEAUTIFUL, RESPONSIVE LAYOUTS.

THE POWER OF JAVASCRIPT AND HOW TO USE IT TO MAKE YOUR WEB APPLICATIONS COME TO LIFE.

BOOTSTRAP 5 FOR CREATING VISUALLY APPEALING, MOBILE-FIRST DESIGNS.

MONGODB AND MONGOOSE FOR MANAGING YOUR APPLICATION'S DATA.

EXPRESS.JS FOR CRAFTING ROBUST SERVER-SIDE LOGIC.

REACT.JS FOR DEVELOPING DYNAMIC AND INTERACTIVE USER INTERFACES.

NODE.JS FOR BUILDING AN EFFICIENT SERVER THAT COMMUNICATES WITH THE CLIENT.

AS YOU PROGRESS THROUGH THIS EBOOK, YOU'LL NOT ONLY LEARN EACH TECHNOLOGY INDIVIDUALLY BUT ALSO UNDERSTAND HOW THEY SYNERGIZE TO CREATE A COHESIVE WEB APPLICATION. WE'LL WORK ON REAL-WORLD EXAMPLES AND PROJECTS TO SOLIDIFY YOUR UNDERSTANDING.

WHETHER YOU'RE BUILDING YOUR FIRST WEB APPLICATION OR LOOKING TO ENHANCE YOUR EXISTING SKILLS, THIS EBOOK IS YOUR DEFINITIVE GUIDE TO MASTERING THE MERN STACK. SO, LET'S EMBARK ON THIS EXCITING JOURNEY TO BECOME PROFICIENT IN WEB DEVELOPMENT AND UNLOCK THE DOOR TO CREATING DYNAMIC, DATA-DRIVEN APPLICATIONS. ARE YOU READY TO MASTER THE MERN STACK?

LET'S GET STARTED!

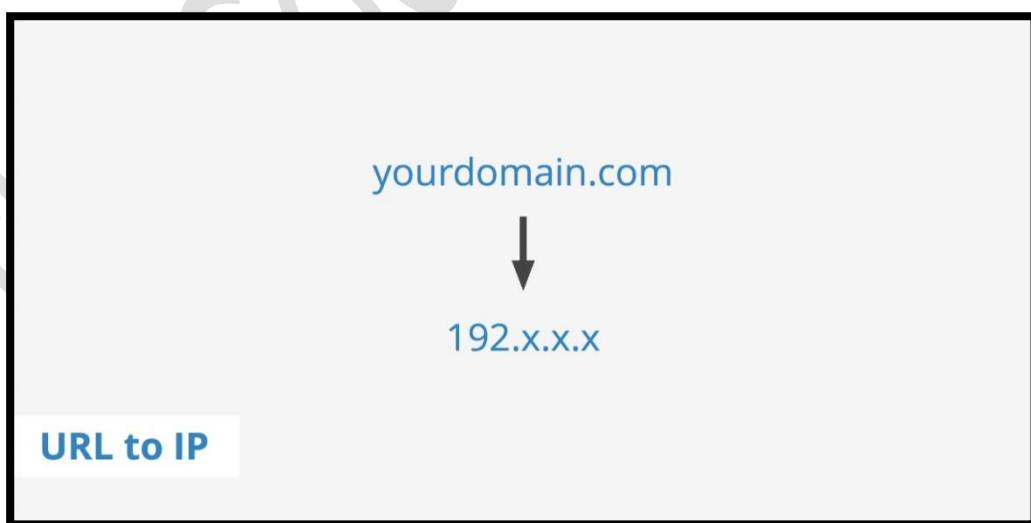
~1. HOW THE INTERNET WORKS:

Imagine you want to visit a website like "www.google.com". When you type this into your browser and hit enter, your computer becomes the "client". It sends a request to your Internet Service Provider (ISP), which is like the middleman connecting you to the larger network called the Internet. Now, your ISP needs to find out where "www.google.com" is located. It sends a message to a special server called the DNS Server. This server's job is to translate the website name into a unique address called an IP Address. It's like looking up a phone number for a person's name.

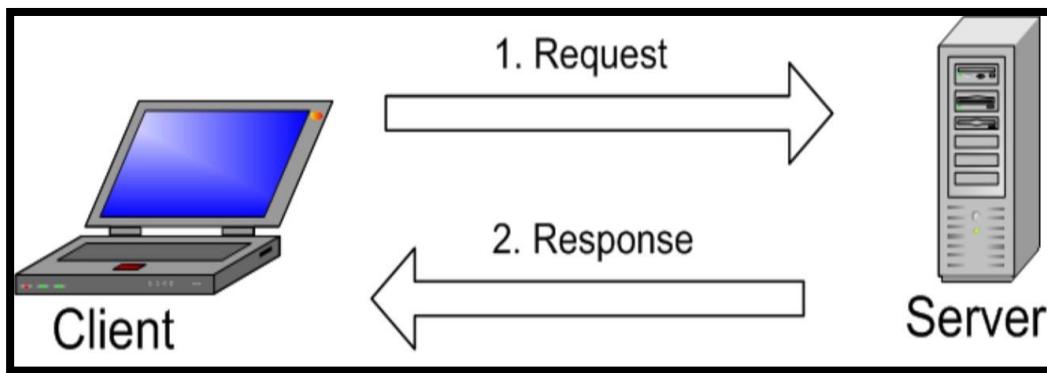
Once the DNS Server gives the IP Address for "www.google.com", your browser uses this address to send a message directly to that specific server. This server belongs to the website you want to visit.

The website's server receives your request, gathers the necessary data (like the webpage content), and sends it back to your computer.

1. *URLS AND IP ADDRESSES*: HERE'S AN IMAGE EXPLAINING HOW URLs ARE CONVERTED INTO IP ADDRESSES:



2. *CLIENT-SERVER PROCESS*: THIS IMAGE SHOWS THE STEPS OF A CLIENT MAKING A REQUEST TO A SERVER:



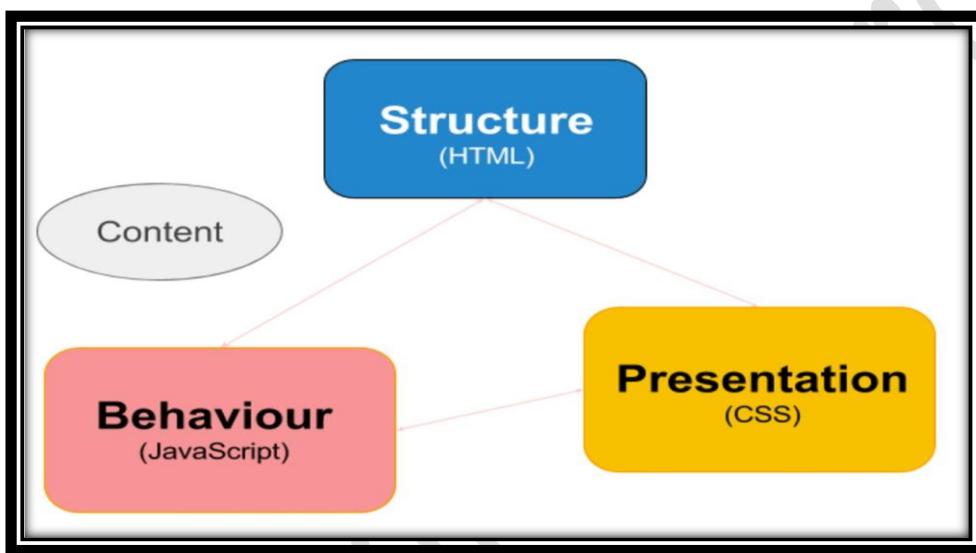
~REVISION NOTES OF HOW INTERNET WORKS.

○ CLIENT – SERVER

- Client makes a request to go to a webpage WWW.GOOGLE.COM
 - request is sent to the Internet Service Provider (ISP)
 - Then the ISP will relay the message to the DNS Server
 - DNS Server
 - Input: web url
 - Output: IP Address of website
 - IP Address is obtained
 - Browser sends message to the server based on the IP address
 - Server sends back the data
-

~2. HOW WEBSITES WORKS

Websites operate through a blend of HTML, CSS, and JavaScript. HTML establishes the structural foundation, defining content and layout. CSS then embellishes the presentation, shaping colors, fonts, and design. Meanwhile, JavaScript introduces interactivity and functionality, enabling dynamic features like form validation or animated elements. These three components synergize to produce the modern web experience, where HTML constructs the framework, CSS adorns it, and JavaScript breathes life into actions and engagement.



~REVISION NOTES OF HOW WEBSITE WORKS.

- HOW WEBSITES WORK
 - HTML
 - Structure
 - CSS
 - Styling
 - JS
 - Actions
-

~3. INTRODUCTION TO HTML

HTML, or Hypertext Markup Language, is the foundational language of the web. It forms the structure and content of web pages, enabling the presentation of text, images, links, and multimedia. HTML uses a system of tags to define various elements within a document, such as headings, paragraphs, lists, and more. These tags are enclosed in angle brackets (<>) and provide instructions to web browsers on how to display content. As you delve into HTML, you'll discover its vital role in creating the structure of websites and learn how to combine it with CSS for styling and JavaScript for interactivity, thus forming the basis of modern web development.

CODE :-

```
<!DOCTYPE html>
<html>
<head>
    <title>My First HTML Page</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph of text.</p>
    <a href="https://www.example.com">Visit Example Website</a>
</body>
</html>
```

1. `<H1>` TO `<H6>` - HEADINGS:

Headings are used to define the structure and hierarchy of content. `<h1>` is the highest level, while `<h6>` is the lowest.

EXAMPLE:

```
<h1>Main Heading</h1>
<h2>Subheading</h2>
<h3>Sub-subheading</h3>
```

2. `<P>` - PARAGRAPH:

The `<p>` tag is used to define paragraphs of text.

EXAMPLE:

3. `<A>` - ANCHOR (LINK):

The `<a>` tag is used to create hyperlinks.

EXAMPLE:

```
<a href="https://
www.example.com">Visit Example
Website</a>
```

4. `` - IMAGE:

The `` tag is used to embed images on a webpage.

EXAMPLE:

```

```

5. `` AND `` - UNORDERED LIST AND LIST ITEMS:

The `` tag is used to create an unordered (bulleted) list, and `` tags define individual list items.

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
</ul>
```

6. `` AND `` - ORDERED LIST AND LIST ITEMS:

Similar to the unordered list, but with numbered items.

EXAMPLE:

```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ol>
```

7. `
` - LINE BREAK:

The `
` tag creates a line break within text.

EXAMPLE:

```
Line 1<br>
Line 2
```

8. `<DIV>` - DIVISION:

The `<div>` tag is a container used to group and style elements.

EXAMPLE:

```
<div style="background-color:  
lightgray;">  
    <p>This is inside a div.</p>  
</div>
```

9. `` - INLINE SPAN:

The `` tag is used for styling a specific portion of text within a larger element.

EXAMPLE:

```
<ol>  
    <li>First item</li>  
    <li>Second item</li>  
    <li>Third item</li>  
</ol>
```

10. FORMS (`<FORM>`):

The `<form>` element is used to create interactive input forms. It can contain various input elements like text fields, checkboxes, radio buttons, and more.

EXAMPLE:

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

HERE'S A BRIEF OVERVIEW OF THE KEY ATTRIBUTES AND ELEMENTS ASSOCIATED WITH THE `<FORM>` TAG:

1. ACTION: SPECIFIES THE URL TO WHICH THE FORM DATA IS SENT WHEN THE USER SUBMITS IT. THIS URL TYPICALLY POINTS TO A SERVER-SIDE SCRIPT THAT PROCESSES THE FORM DATA.
2. METHOD: DEFINES THE HTTP METHOD TO BE USED WHEN SUBMITTING THE FORM DATA. THE TWO MOST COMMON METHODS ARE "GET" AND "POST." "GET" APPENDS FORM DATA TO THE URL, WHILE "POST" SENDS IT IN THE REQUEST BODY.
3. `<INPUT>`, `<TEXTAREA>`, `<SELECT>`: THESE ARE ELEMENTS THAT CAN BE PLACED WITHIN A `<FORM>` TO COLLECT VARIOUS TYPES OF USER INPUT.
4. `<BUTTON>`: YOU CAN USE BUTTONS TO SUBMIT THE FORM, RESET FORM DATA, OR TRIGGER JAVASCRIPT FUNCTIONS.
5. NAME ATTRIBUTE: ASSIGNS A NAME TO EACH FORM ELEMENT. THIS NAME IS USED TO IDENTIFY THE DATA WHEN THE FORM IS SUBMITTED, TYPICALLY AS KEY-VALUE PAIRS.
6. ID ATTRIBUTE: PROVIDES A UNIQUE IDENTIFIER FOR EACH FORM ELEMENT, MAKING IT EASIER TO TARGET AND STYLE THEM USING CSS OR MANIPULATE THEM WITH JAVASCRIPT.
7. PLACEHOLDER ATTRIBUTE: DEFINES A SHORT HINT OR EXAMPLE OF THE EXPECTED INPUT FOR TEXT FIELDS.

HERE'S A SIMPLE EXAMPLE OF A BASIC HTML FORM:

```
<FORM ACTION="PROCESS_FORM.PHP" METHOD="POST">
    <LABEL FOR="USERNAME">USERNAME:</LABEL>
```

```
<INPUT TYPE="TEXT" ID="USERNAME" NAME="USERNAME"  
PLACEHOLDER="ENTER YOUR USERNAME" REQUIRED>  
  
<LABEL FOR="PASSWORD">PASSWORD:</LABEL>  
<INPUT TYPE="PASSWORD" ID="PASSWORD" NAME="PASSWORD"  
REQUIRED>  
  
<INPUT TYPE="SUBMIT" VALUE="SUBMIT">  
</FORM>
```

11. INPUT ELEMENTS (`<INPUT>`):

`<input>` elements provide a way for users to enter data. The type attribute determines the kind of input, such as text, password, checkbox, etc.

EXAMPLE:

```
<input type="text" name="username"  
placeholder="Enter your username">  
<input type="password" name="password"  
placeholder="Enter your password">
```

12. TABLE (`<TABLE>`), TABLE ROW (`<TR>`), TABLE HEADER (`<TH>`), TABLE BODY (`<TBODY>`), TABLE FOOTER (`<TFOOT>`):

Tables are used to display tabular data. `<tr>` represents a row, `<th>` is used for header cells, and `<tbody>` and `<tfoot>` group content within the table body and footer.

EXAMPLE:

```
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>John</td>
      <td>25</td>
    </tr>
    <!-- More rows... -->
  </tbody>
  <tfoot>
    <tr>
      <td colspan="2">Total: 1
record</td>
    </tr>
  </tfoot>
</table>
```

13. HYPERLINK (`<A>`):

THE `<A>` ELEMENT CREATES HYPERLINKS, ALLOWING USERS TO NAVIGATE TO OTHER WEB PAGES OR RESOURCES.

EXAMPLE:

```
<a href="https://
www.example.com">Visit Example
Website</a>
```

14. IMAGE (``):

The `` tag is used to display images on a webpage.

EXAMPLE:

```

```

15. TEXT FORMATTING: BOLD (``), ITALICIZE (`<I>`), EMPHASIS (``):

These tags are used for text formatting. `` for bold, `<i>` for italic, and `` for emphasis (usually displayed in italics).

EXAMPLE:

```
<p>This is <b>bold</b> text.</p>  
<p>This is <i>italic</i> text.</p>  
<p>This is <em>emphasized</em> text.</p>
```

~REVISION NOTES OF INTRODUCTION TO HTML

- HYPERTEXT MARKUP LANGUAGE
- FILE STRUCTURE
 - !DOCTYPE HTML
 - SYMBOLIZES THAT THE FILE IS HTML5

- <HTML>

- EVERYTHING CONTAINED IN THE TAGS, WILL BE HTML CODE

- <HEAD>

- HOLDS INFORMATION ABOUT THE WEBPAGE

- <TITLE>

- <META>

- TELLS THE BROWSER HOW TO DISPLAY THE HTML WEB PAGE

- CHARSET=

- UTF-8

- NAME="DESCRIPTION"

- BRIEF DESCRIPTION OF THE WEB PAGE, THAT TELLS SEARCH ENGINES HOW TO DISPLAY THE PAGE HYPERLINK

- <BODY>

- TAGS

- HEADER

- <H#>

- H1,H2,H3,H4,H5,H6

- LINE BREAK

-

- WHITE SPACE

- HORIZONTAL RULE

- <HR>

- HORIZONTAL LINE

- PARAGRAPH

- <P>

- GOES INTO A NEW LINE

- ITALICIZE

- VS <I>

- USE EM OVER I, AS I IS JUST FOR STYLE, BUT EM CONVEYS

MORE

INFORMATION TO THE USER AS WELL AS TO THE BROWSER

- EMPHASIS

-

- PUTS AN EMPHASIS ON THE TAG

- ITALICIZE

- <I>
- STYLES THE TEXT
- BOLD
 - VS
 - USE STRONG OVER B, AS IT HIGHLIGHTS AN ADDED IMPORTANCE
- LISTS
 - ORDERED LIST
 - NUMBERS
 -
 - UNORDERED LIST
 - BULLET POINTS
 -
- IMAGE
 -
 - SRC=
 - CAN BE A URL TO ANOTHER SOURCE WHERE IMAGE IS ON THE INTERNET
 - CAN BE A LOCAL FILE IN THE PATH OF THE PAGE DIRECTORY
 - ALT=
 - ALTERNATIVE, IF SRC FAILS TO OBTAIN AN IMAGE TO RENDER TO THE PAGE
- HYPERLINK
 - ANCHOR
 - <A>
 - HREF=
 - URL TO LINK TO
 - DOWNLOAD
 - DOWNLOADS THE CONTENTS OF THE URL
 - TABLE
 - <TABLE>
 - <THEAD>
 - TABLE HEAD
 - <TBODY>

- TABLE BODY
 - <TFOOT>
 - TABLE FOOTER
 - <TR>
 - TABLE ROW
 - <TD>
 - TABLE DATA
 - <TH>
 - TABLE HEADER CELL
- FORMS
 - <FORM>
 - LABELS
 - <LABEL>
 - INPUTS
 - <INPUT>
 - EMAIL
 - COLOR
 - PASSWORD
 - CHECKBOX
 - DATE
 - RANGE
 - RADIO

4. INTRODUCTION TO CSS

CSS, or Cascading Style Sheets, is the design language of the web, complementing HTML's structure. It empowers developers to control the visual presentation and layout of web pages. CSS achieves this by defining rules that dictate how HTML elements should appear.

These rules encompass properties like color, typography, spacing, positioning, and responsiveness. CSS follows a cascading nature, meaning rules can be inherited or overridden, allowing for consistency and flexibility across websites. By separating content (HTML) from presentation (CSS), developers can efficiently manage large-scale projects, create visually appealing designs, and adapt layouts for various devices. As you delve into CSS, you'll explore selectors that target specific elements, learn about the box model that governs layout, and grasp responsive design principles to craft user-friendly experiences across a wide range of devices and screen sizes.

1. INLINE CSS:

Inline CSS is applied directly to an individual HTML element using the "style" attribute.

EXAMPLE:

```
<p style="color: blue;">This is a blue paragraph.</p>
```

2. INTERNAL CSS:

Internal CSS is placed within the `<style>` element in the HTML `<head>`. It applies to the entire HTML document.

EXAMPLE:

```
<head>
  <style>
    p {
      color: green;
    }
  </style>
</head>
<body>
  <p>This is a green paragraph.</p>
</body>
```

3. EXTERNAL CSS:

External CSS is stored in a separate .css file and linked to the HTML document using the `<link>` element.

EXAMPLE:

```
p {
  font-size: 16px;
}
```

```
<head>
  <link rel="stylesheet" type="text/
css" href="styles.css">
</head>
<body>
  <p>This paragraph has a font size
of 16px.</p>
</body>
```

4. ANATOMY OF CSS SYNTAX:

CSS rules consist of a selector (what to style), followed by curly braces enclosing declarations (how to style).

EXAMPLE:

```
selector {  
    property: value;  
}
```

5. CSS SELECTOR:

A selector defines which HTML elements to style. It can target elements by tag, class, or ID.

6. TAG SELECTOR:

Styles all elements of a specific tag.

Example:

```
css  
p {  
    color: red;  
}
```

7. CLASS SELECTOR:

Styles elements with a specific class attribute.

Example:

```
css  
.highlight {  
    background-color: yellow;  
}
```

8. ID SELECTOR:

Styles a unique element with a specific ID attribute.

EXAMPLE:

Example:

```
css
#header {
    font-size: 24px;
}
```

9. ID VS. CLASS SELECTOR:

IDs should be unique, while classes can be applied to multiple elements. IDs have higher specificity, potentially affecting other styles.

~REVISION NOTES OF INTRODUCTION TO CSS

- EVERY ELEMENTS IN A WEB PAGE IS ESSENTIALLY ALL COMPOSED OF A SERIES OF BOXES
- INLINE CSS
 - CSS ATTRIBUTES THAT CAN BE PLACED IN THE HTML FILE TO STYLE THE PAGE IN THE FORM OF BEING IN THE INDIVIDUAL HTML TAG THEMSELVES
- INTERNAL CSS
 - USE OF THE ○ EXTERNAL CSS
 - LINK TO THE .CSS FILE WITHIN THE HTML FILE
- ANATOMY OF CSS SYNTAX
 - SELECTOR { PROPERTY : VALUE; }
- CSS SELECTORS
 - TAG SELECTOR
 - EX. H1, IMG, BODY
 - CLASS SELECTOR
 - INSIDE HTML, OF THE HTML TAG INCLUDE CLASS= " "

- THEN IN THE CSS, SELECT THE CLASS BY INCLUDING A . IN THE FRONT o .CLASS_IDENTIFIER { PROPERTY : VALUE; }
 - ID SELECTOR
 - INSIDE HTML, OF THE HTML TAG, INCLUDE ID=" " "
 - THEN IN THE CSS, SELECT THE ID BY INCLUDING A # IN THE FRONT o #ID_IDENTIFIER { PROPERTY : VALUE; }
 - ID VS CLASS SELECTORS
 - ID - USE ONLY FOR A SINGLE HTML TAG WHERE YOU WANT TO MODIFY ITS STYLE
 - o THERE CAN ONLY BE ONE ID_IDENTIFIER FOR A GIVEN HTML FILE
 - CLASS - USE FOR A GROUP OF HTML TAGS WHERE YOU WANT TO HAVE A CONSISTENT STYLE FOR THE GROUP o A HTML TAG CAN HAVE MULTIPLE CLASSES
-

~5. INTERMEDIATE CSS

1. SPAN (``):

The `` element is used to style a specific portion of text within a larger element.

EXAMPLE:

```
<p>This is <span style="color: blue;">blue</span> text.</p>
```

2. BOX MODEL (WIDTH, HEIGHT, PADDING, BORDER, MARGIN):

The box model represents how elements are structured with content, padding, border, and margin.

EXAMPLE:

```
.box {  
    width: 200px;  
    height: 100px;  
    padding: 10px;  
    border: 1px solid black;  
    margin: 20px;  
}
```

3. DISPLAY PROPERTY (BLOCK, INLINE):

The `display` property determines how elements behave in the layout.

EXAMPLE:

```
.block {  
    display: block;  
}  
  
.inline {  
    display: inline;  
}
```

4. CSS STATIC AND RELATIVE POSITIONING:

Static is the default position. Relative positions an element relative to its normal position.

EXAMPLE:

```
.static {  
    position: static;  
}  
  
.relative {  
    position: relative;  
    top: 20px;  
    left: 30px;  
}
```

5. CENTERING ELEMENTS WITH CSS:

Centering elements horizontally and vertically.

EXAMPLE:

```
.center {  
    margin: auto; /* Horizontal  
    centering */  
    display: flex;  
    justify-content: center;  
    align-items: center; /* Vertical  
    centering */  
    height: 100vh;  
}
```

6. CENTERING ELEMENTS WITH CSS:

Centering elements horizontally and vertically.

EXAMPLE:

```
p {  
    font-size: 16px;  
    font-weight: bold;  
    font-style: italic;  
    font-family: Arial, sans-serif;  
}
```

7. CSS FLOAT AND CLEAR:

`float` positions elements side by side. `clear` prevents elements from floating around another element.

EXAMPLE:

```
.left {  
    float: left;  
    width: 50%;  
}  
.right {  
    float: right;  
    width: 50%;  
}  
.clear {  
    clear: both;  
}
```

1. FLEXBOX:

- FLEXBOX, SHORT FOR "FLEXIBLE BOX LAYOUT," IS A CSS LAYOUT MODEL THAT ALLOWS YOU TO DESIGN COMPLEX LAYOUTS WITH A MORE EFFICIENT AND PREDICTABLE WAY TO DISTRIBUTE SPACE AND ALIGN CONTENT IN A CONTAINER.
- THE MAIN PROPERTIES FOR WORKING WITH FLEXBOX ARE:
- DISPLAY: FLEX; APPLIED TO THE CONTAINER ELEMENT, THIS PROPERTY TURNS IT INTO A FLEX CONTAINER, ENABLING FLEXBOX LAYOUT FOR ITS CHILDREN.
- FLEX-DIRECTION: DETERMINES THE MAIN AXIS ALONG WHICH THE FLEX ITEMS ARE PLACED. IT CAN BE SET TO ROW, COLUMN, ROW-REVERSE, OR COLUMN-REVERSE.
- JUSTIFY-CONTENT: DEFINES HOW FLEX ITEMS ARE ALIGNED ALONG THE MAIN AXIS, CONTROLLING SPACING BETWEEN THEM. OPTIONS INCLUDE FLEX-START, FLEX-END, CENTER, SPACE-BETWEEN, AND SPACE-AROUND.
- ALIGN-ITEMS: SPECIFIES HOW FLEX ITEMS ARE ALIGNED ALONG THE CROSS AXIS, WHICH IS PERPENDICULAR TO THE MAIN AXIS. OPTIONS INCLUDE FLEX-START, FLEX-END, CENTER, BASELINE, AND STRETCH.
- FLEX: ALLOWS YOU TO DISTRIBUTE AVAILABLE SPACE AMONG FLEX ITEMS, DEFINING THEIR RELATIVE PROPORTIONS.

2. ALIGN ITEMS:

- ALIGN-ITEMS IS A CSS PROPERTY USED WITHIN THE FLEXBOX LAYOUT MODEL.
- IT DETERMINES HOW FLEX ITEMS ARE ALIGNED ALONG THE CROSS-AXIS (VERTICAL AXIS IN A ROW LAYOUT OR HORIZONTAL AXIS IN A COLUMN LAYOUT) WITHIN THE FLEX CONTAINER.
- COMMON VALUES FOR ALIGN-ITEMS INCLUDE:
- FLEX-START: ALIGNS ITEMS AT THE START OF THE CROSS-AXIS.
- FLEX-END: ALIGNS ITEMS AT THE END OF THE CROSS-AXIS.
- CENTER: CENTERS ITEMS ALONG THE CROSS-AXIS.
- BASELINE: ALIGNS ITEMS BASED ON THEIR TEXT BASELINE.

- STRETCH: STRETCHES ITEMS TO FILL THE CROSS-AXIS, WHICH IS THE DEFAULT BEHAVIOR.

SELECTOR	CSS RULE	DESCRIPTION
:ACTIVE	A:ACTIVE	ALL ACTIVE LINKS
:VISITED	A:VISITED	ALL VISITED LINKS
:LINK	A:LINK	ALL UNVISITED LINKS
::AFTER	UL::AFTER	ADD COSMETIC CONTENT AFTER EACH ELEMENT
::BEFORE	UL::BEFORE	ADD COSMETIC CONTENT BEFORE EACH ELEMENT
:CHECKED	INPUT:CHECKED	ALL <INPUT> ELEMENTS THAT ARE CHECKED
:DEFAULT	INPUT:DEFAULT	ALL <INPUT> ELEMENTS THAT ARE SPECIFIED AS DEFAULT
:ENABLED	INPUT:ENABLED	ALL <INPUT> ELEMENTS THAT ARE ENABLED
:DISABLED	INPUT:DISABLED	ALL <INPUT> ELEMENTS THAT ARE DISABLED
:VALID	INPUT:VALID	ALL <INPUT> INPUT ELEMENTS WITH A VALID VALUE
:INVALID	INPUT:INVALID	ALL <INPUT> ELEMENTS WITH AN INVALID VALUE
:IN-RANGE	INPUT:IN-RANGE	ALL <INPUT> ELEMENTS WITH A VALUE WITHIN A SPECIFIED RANGE
:OUT-OF-RANGE	INPUT:OUT-OF-RANGE	ALL <INPUT> ELEMENTS WITH A VALUE OUTSIDE THE SPECIFIED RANGE
:INDETERMINATE	INPUT:INDETERMINATE	ALL <INPUT> ELEMENTS THAT ARE IN AN INDETERMINATE STATE
:REQUIRED	INPUT:REQUIRED	ALL <INPUT> ELEMENTS WITH A "REQUIRED"

		ATTRIBUTE
:OPTIONAL	INPUT:OPTIONAL	ALL <INPUT> ELEMENTS WITH NO "REQUIRED" ATTRIBUTE
:READ-ONLY	INPUT:READ-ONLY	ALL <INPUT> ELEMENTS WITH A "readonly" ATTRIBUTE
:READ-WRITE	INPUT:READ-WRITE	ALL <INPUT> ELEMENTS WITHOUT A "readonly" ATTRIBUTE
::PLACEHOLDER	INPUT::PLACEHOLDER	ALL <INPUT> ELEMENTS WITH PLACEHOLDER TEXT
:HOVER	INPUT:HOVER	THE <INPUT> ELEMENT CURRENTLY BEING HOVERED
:FOCUS	INPUT:FOCUS	THE <INPUT> ELEMENT CURRENTLY WITH FOCUS
:EMPTY	OL:EMPTY	ALL ELEMENTS WITHOUT CHILD ELEMENTS
:FIRST-CHILD	LI:FIRST-CHILD	ALL ELEMENTS THAT ARE THE FIRST CHILD OF THEIR PARENT
:LAST-CHILD	LI:LAST-CHILD	ALL ELEMENTS THAT ARE THE LAST CHILD OF THEIR PARENT
:NTH-CHILD(N)	LI:NTH-CHILD(2)	ALL ELEMENTS THAT ARE THE SECOND CHILD OF THEIR PARENT
:NTH-LAST-CHILD(N)	LI:NTH-LAST-CHILD(2)	ALL ELEMENTS THAT ARE THE SECOND TO LAST CHILD OF THEIR PARENT
:ONLY-CHILD	LI:ONLY-CHILD	ALL ELEMENTS THAT ARE THE ONLY CHILD OF THEIR PARENT
:FIRST-OF-TYPE	LABEL:FIRST-OF-TYPE	ALL <LABEL> ELEMENTS THAT ARE THE FIRST LABEL OF THEIR PARENT
:LAST-OF-TYPE	LABEL:LAST-OF-TYPE	ALL <LABEL> ELEMENTS THAT ARE THE LAST LABEL

		OF THEIR PARENT
:NTH-OF-TYPE(N)	LABEL:NTH-OF-TYPE(2)	ALL <LABEL> ELEMENTS THAT ARE THE SECOND LABEL OF THEIR PARENT
:NTH-LAST-OF-TYPE(N)	LABEL:NTH-LAST-OF-TYPE(2)	ALL <LABEL> ELEMENTS THAT ARE THE SECOND TO LAST LABEL OF THEIR PARENT
:ONLY-OF-TYPE	LABEL:ONLY-OF-TYPE	ALL <LABEL> ELEMENTS THAT ARE THE ONLY LABEL ELEMENT OF THEIR PARENT
::FIRST-LETTER	P::FIRST-LETTER	THE FIRST LETTER OF EVERY <P> ELEMENT
::FIRST-LINE	P::FIRST-LINE	THE FIRST LINE OF EVERY <P> ELEMENT
:LANG(LANGUAGE)	P:LANG(FR)	ALL FRENCH <P> ELEMENTS
:NOT(SELECTOR)	:NOT(TABLE)	ALL ELEMENTS THAT ARE NOT A <TABLE> ELEMENT
:ROOT	:ROOT	THE DOCUMENT'S ROOT ELEMENT
::SELECTION	::SELECTION	PARTS OF AN ELEMENT THAT IS SELECTED (HIGHLIGHTED) BY THE USER, USUALLY TEXT
:TARGET	:TARGET	THE ELEMENT THAT MATCHES THE URL FRAGMENT, E.G. ELEMENT WITH ID="LIST" WHEN THE URL IS WWW.MYSITE.COM#LIST

CSS LENGTH :



UNIT

DESCRIPTION



Represents the font-size of the element.

Example: 1.5em is 50% larger than the current font.



Represents the font-size of the root, usually the `<html>` element.

Browsers commonly set 1rem = 16px, but this can be changed.



Represents a percentage of the parent element.

Example: width:50% is 50% of the parent's width.



Represents 1% of the viewport width.

Example: width:25vw is a quarter of the viewport width.



Represents 1% of the viewport height.

Example: height:50vh is a half the viewport height.



Represents 1% of the viewport's smaller dimension (can be width or height).



Represents 1% of the viewport's larger dimension (can be width or height).



Represents the x-height (height of lower-case x) of the current font.



Represents the width of the "0" (zero) character.

~REVISION NOTES OF INTERMEDIATE CSS



- COMBINES A GROUP OF HTML ELEMENTS INTO A SINGLE BOX
- USEFUL TO STRUCTURE AND DIVIDE UP WEB CONTENT

- AN INLINE ELEMENT WHERE YOU CAN USE TO PICK A SUB-SECTION OF A HTML ELEMENT, WHICH CAN THEN BE USED TO STYLE
 - THE BOX MODEL
 - WIDTH
 - HEIGHT
 - PADDING
 - BORDER
 - BORDER SIZE
 - SOLID, DASHED, ...
 - MARGIN
 - DISPLAY PROPERTY
 - WE CAN ALSO MODIFY THE DISPLAY PROPERTY FOR ANY HTML ELEMENT
 - EX. DISPLAY : INLINE;
 - BLOCK
 - ELEMENTS THAT WILL TAKE UP IT'S OWN LINE, WHERE THE WIDTH TAKES UP THE WHOLE SCREEN
 - INLINE
 - INLINE-BLOCK
 - ALLOWS OTHER ELEMENTS TO SIT TO THEIR LEFT OR RIGHT SIDE
 - NONE
 - HTML ELEMENT IS HIDDEN AS IF IT NEVER EXISTED
 - ALTERNATIVE, VISIBILITY : HIDDEN
 - THE ELEMENT CAN NOT BE SEEN, BUT IT FILLS THE ORIGINAL SIZE POSITION
 - CSS STATIC AND RELATIVE POSITIONING
 - 1. CONTENT DICTATES EVERYTHING
 - 2. ORDER OF CONTENT COMES FROM THE CODE
 - 3. PARENT - CHILDREN RELATIONSHIP
 - CHILDREN ELEMENTS ARE LAYERED ON TOP OF THE PARENT ELEMENTS
 - POSITION
 - 4 METHODS TO CHANGE THE LAYOUT POSITIONING OF ELEMENTS
 - STATIC

- ALL HTML ELEMENTS ARE STATIC BY DEFAULT
- RELATIVE
 - THE ADJUSTMENT POSITION IS APPLIED IN RELATIVE TO THE STATIC POSITIONING
 - POSITION : STATIC;
 - THEN WE CAN ONLY SEE CHANGES ONCE WE CHANGE THE COORDINATE VALUES.
 - TOP, BOTTOM, LEFT, RIGHT
 - RELATIVE POSITIONING ACTS INDEPENDENTLY ON THE HTML ELEMENT
 - ABSOLUTE
 - MOVES ELEMENT RELATIVE TO ITS PARENT ELEMENT
 - ADDS MARGINS RELATIVE TO ITS PARENT ELEMENT
 - ABSOLUTE POSITIONING ACTS DEPENDENTLY ON THE HTML ELEMENT
 - FIXED
 - THE ELEMENT WILL BE FIXED IN PLACE, EVEN WHEN THE USER SCROLLS AROUND THE WEB PAGE
 - USEFUL FOR NAVIGATION BARS
 - CENTERING ELEMENTS WITH CSS
 - TEXT-ALIGN : CENTER
 - CENTERS ALL ELEMENTS, WHERE IT DOES NOT HAVE A WIDTH SET
 - MARGIN : TOP RIGHT BOTTOM LEFT
 - AUTO - WILL BE USED TO CENTER THE ELEMENT
 - FONT STYLING
 - FONT-FAMILY
 - SERIF
 - SANS-SERIF
 - NOTE THAT THE AVAILABILITY OF FONTS VARIES BASED ON THE BROWSER AND OPERATING SYSTEM. SO IF YOU CHOOSE A FONT FOR YOUR WEB SITE, BE MINDFUL OF ITS AVAILABILITY TO YOUR END USERS.
 - EM
 - EXAMPLE.
 - FONT-SIZE: 2REM;
 - 1 EM = 16PX = 100%

- DYNAMIC SIZING OF TEXT, IN PROPORTION TO A STANDARD, BEING 16PX = 1EM
- DYNAMIC SIZING OF TEXT, RELATIVE TO THE HTML ELEMENTS PARENT
 - REM
 - DYNAMIC SIZING OF TEXT, RELATIVE TO THE ROOT
 - PREFERABLE TO USE FOR TEXTS ○ CSS FLOAT AND CLEAR
 - FLOAT
 - THE HTML ELEMENT “FLOATS”, SO THAT OTHER ELEMENTS CAN WRAP AROUND THE ELEMENT ■ CLEAR
 - OPPOSITE OF FLOAT, THE HTML ELEMENT DOES NOT ALLOW WRAPPING AROUND OF OTHER ELEMENTS.

~6. **BOOTSTRAP 5**

Bootstrap 5 was a popular front-end web framework used for building responsive and user-friendly websites and web applications. Bootstrap 5 builds on the success of its predecessor, Bootstrap 4, and introduces several new features and improvements. Here's a brief overview of Bootstrap 5:

1. Modular Design: Bootstrap 5 is designed with modularity in mind, making it easier to customize and use only the components you need for your project. This modular approach helps reduce the overall file size and improve performance.
2. No jQuery: Bootstrap 5 has dropped its dependency on jQuery, which was used in earlier versions. This decision was made to promote modern JavaScript practices and improve performance.
3. Customization: Bootstrap 5 offers extensive customization options, allowing developers to tailor the framework to their specific project needs. You can use custom CSS variables to change design elements like colors and fonts.
4. Improved Grid System: The grid system in Bootstrap has been refined to make it even more flexible and

responsive, enabling developers to create complex layouts with ease.

5. New Features: Bootstrap 5 introduced several new components and features, including an extended color palette, a more powerful and customizable navbar, and enhanced form controls.

6. Sass Support: Bootstrap 5 provides support for Sass, a popular CSS preprocessor. This allows for more efficient and organized CSS development.

7. Responsive Design: As with previous versions, Bootstrap 5 prioritizes responsive web design, ensuring that websites and applications look and function well on various devices and screen sizes.

8. Accessibility: The framework has improved accessibility features to help developers create web content that is more inclusive and usable for all users.

9. Documentation: Bootstrap 5 comes with comprehensive and well-documented resources, making it easier for developers to learn and use the framework effectively.

o INSTALLATION PROCEDURE

- 1. DOWNLOAD THE BOOTSTRAP FILES TO LOCAL AND LINK TO THE LOCAL IN HTML
 - THOUGH THIS CAN CAUSE LOW PERFORMANCE, AS THE BROWSER WILL NOT RECOGNIZE THE FILE, AND WILL DOWNLOAD IT.
 - METHOD IS NOT RECOMMENDED
- 2. INCLUDE THE INDIVIDUAL BOOTSTRAP REFERENCE LINK
- 3. INCLUDE THE START TEMPLATE FROM BOOTSTRAP, WHICH ALSO INCLUDES POPPER.JS AND JQUERY

INSTALLATION:

1. DOWNLOAD AND LOCAL LINK (NOT RECOMMENDED):

Download Bootstrap files and place them in your project directory. Link to the CSS and JS files in your HTML file's `<head>` section.

```
html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="path/to/bootstrap.min.css">
    <script src="path/to/bootstrap.min.js"></script>
</head>
<body>
    <!-- Your content here -->
</body>
</html>
```

2. INDIVIDUAL BOOTSTRAP REFERENCE LINK:

Include individual reference links for Bootstrap CSS and JS from a CDN.

```
html
  <!DOCTYPE html>
  <html>
    <head>
      <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
      <script
        src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
      </script>
      <script
        src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js">
      </script>
      <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js">
      </script>
    </head>
    <body>
      <!-- Your content here -->
    </body>
  </html>
```

3. BOOTSTRAP START TEMPLATE:

Use the Bootstrap Starter Template that includes necessary dependencies.

HTML

```
html
  <!DOCTYPE html>
  <html>
    <head>
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
      <script
        src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
      </script>
      <script
        src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
      <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js">
      </script>
    </head>
    <body>
      <!-- Your content here -->
    </body>
  </html>
```

NAVIGATION BAR:

```
html
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Logo</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNav">
    <ul class="navbar-nav ml-auto">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Features</a>
      </li>
      <!-- Add more menu items here -->
    </ul>
  </div>
</nav>
```

GRID LAYOUT SYSTEM:

```
html
<div class="container">
  <div class="row">
    <div class="col-md-6">
      <!-- Content for the first column -->
    </div>
    <div class="col-md-6">
      <!-- Content for the second column -->
    </div>
  </div>
</div>
```

BUTTONS:

```
html
<button class="btn btn-primary">Primary Button</button>
<button class="btn btn-secondary">Secondary Button</button>
<button class="btn btn-success">Success Button</button>
<!-- Add more button types as needed --&gt;</pre>
```

CAROUSEL:

```
html
<div id="myCarousel" class="carousel slide" data-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
    <!-- Add more carousel items here --&gt;
  &lt;/div&gt;
  &lt;a class="carousel-control-prev" href="#myCarousel" role="button" data-slide="prev"&gt;
    &lt;span class="carousel-control-prev-icon" aria-hidden="true"&gt;&lt;/span&gt;
    &lt;span class="sr-only"&gt;Previous&lt;/span&gt;
  &lt;/a&gt;
  &lt;a class="carousel-control-next" href="#myCarousel" role="button" data-slide="next"&gt;
    &lt;span class="carousel-control-next-icon" aria-hidden="true"&gt;&lt;/span&gt;
    &lt;span class="sr-only"&gt;Next&lt;/span&gt;
  &lt;/a&gt;
&lt;/div&gt;</pre>
```

CARDS:

```
html
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <h5 class="card-title">Card Title</h5>
    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>
  </div>
</div>
```

~REVISION NOTES OF BOOTSTRAP 5

BOOTSTRAP 5 :

- FRONT-END LIBRARY/FRAMEWORK
- BENEFITS
 - ADAPTIVE UI DISPLAY FOR VARIOUS KINDS OF SCREENS
 - CATALOG OF PRE-STYLED ELEMENTS
- INSTALLATION
 - 1. DOWNLOAD THE BOOTSTRAP FILES TO LOCAL AND LINK TO THE
 - 1. INTRODUCTION TO BOOTSTRAP 5

WHAT IS BOOTSTRAP?

ADVANTAGES OF USING BOOTSTRAP FOR WEB DEVELOPMENT.

2. GETTING STARTED

DOWNLOADING AND INCLUDING BOOTSTRAP IN YOUR PROJECT.

SETTING UP THE DEVELOPMENT ENVIRONMENT.

3. BASIC HTML STRUCTURE

THE STRUCTURE OF A BOOTSTRAP-BASED HTML DOCUMENT.

USING THE VIEWPORT META TAG FOR RESPONSIVENESS.

4. GRID SYSTEM

UNDERSTANDING THE GRID SYSTEM IN BOOTSTRAP 5.
CONTAINER, ROWS, AND COLUMNS.
GRID BREAKPOINTS AND RESPONSIVE DESIGN.

5. TYPOGRAPHY

WORKING WITH BOOTSTRAP TYPOGRAPHY CLASSES.
CUSTOMIZING FONTS AND TEXT STYLING.

6. CSS COMPONENTS

- >BUTTONS
- >FORMS AND FORM CONTROLS
- >ALERTS AND BADGES
- >TABLES
- >CARDS
- >DROPDOWNS
- >NAVBAR
- >PAGINATION
- >BREADCRUMBS

7. UTILITIES

- >SPACING UTILITIES
- >DISPLAY UTILITIES
- >TEXT UTILITIES

->VISIBILITY UTILITIES

8. CUSTOMIZATION

->USING BOOTSTRAP'S CUSTOM CSS VARIABLES.

->CREATING CUSTOM THEMES AND STYLES.

9. JAVASCRIPT COMPONENTS

->MODALS

->CAROUSELS

->ACCORDION

->POPOVERS

->TOOLTIPS

10. RESPONSIVE DESIGN

WORKING WITH RESPONSIVE CLASSES AND UTILITIES.

TESTING RESPONSIVENESS ON DIFFERENT DEVICES.

11. ACCESSIBILITY

ENSURING YOUR BOOTSTRAP-BASED SITE IS
ACCESSIBLE.

ARIA ROLES AND ATTRIBUTES.

12. SASS INTEGRATION

USING SASS WITH BOOTSTRAP 5 FOR MORE EFFICIENT
CSS DEVELOPMENT.

13. MIGRATION FROM BOOTSTRAP 4

GUIDELINES FOR MIGRATING FROM BOOTSTRAP 4 TO BOOTSTRAP 5.

14. BEST PRACTICES

TIPS AND BEST PRACTICES FOR USING BOOTSTRAP
EFFECTIVELY.

15. RESOURCES

LINKS TO OFFICIAL BOOTSTRAP DOCUMENTATION,
COMMUNITY RESOURCES, AND HELPFUL TOOLS.

16. CASE STUDIES AND EXAMPLES

PRACTICAL EXAMPLES AND CASE STUDIES SHOWCASING
THE USE OF BOOTSTRAP 5 IN REAL-WORLD PROJECTS.

- MEDIA QUERY BREAKPOINTS

- MOBILE-FIRST

- MORE PEOPLE GO ON WEBSITES ON THEIR
PHONES THAN ON THEIR
DESKTOPS
 - GOOGLE RANKS SEARCHES BASED ON MOBILE-
FRIENDLY-NESS

- @MEDIA <TYPE> <FEATURE> { ... }

- WORKS LIKE A TRUE/FALSE STATEMENT. IF
TRUE, RUN THE CODE

- <TYPE>

- PRINT

- SCREEN

- SPEECH
 - <FEATURE>
 - EXAMPLE:
- @MEDIA SCREEN (MIN-WIDTH: 900PX) { ... }
 - CODE SHOULD ONLY BE ACTIVATED IF THE MEDIA IS A SCREEN WITH A MIN-WIDTH OF 900PX
- CAN ALSO COMBINE MULTIPLE FEATURES
 - @MEDIA SCREEN (MIN-WIDTH: 900PX) AND (MAX-WIDTH: 1500PX) { ... }
 - DIFFERENT TYPES OF VIEWS
 - DEVICE
- SIZE OF THE DEVICE, FIXED
 - BROWSER SIZE
 - VIEWPORT
- SIZE OF THE SCREEN THAT YOUR WEBSITE IS BEING DISPLAYED ON
- CODE REFACTORING
 - 1. READABILITY
 - 2. MODULARITY
 - 3. EFFICIENCY
 - 4. LENGTH
- COMBINING SELECTORS
 - MULTIPLE SELECTORS
 - SELECTOR1, SELECTOR2, SELECTOR# { ... }

- STYLING IS APPLIED TO ALL SELECTORS IN THE LIST
SEPERATED BY
COMMAS
 - HIERARCHICAL SELECTORS
 - SELECTOR1 SELECTOR2 { ... }
- SELECTOR1 IS THE PARENT, SELECTOR2 IS THE CHILD
- STYLING IS PERFORMED ON THE CHILD ELEMENT
- NOTE: THIS IS NOT REALLY USED FOR ID SELECTORS,
AS THAT IS
REDUNDANT
 - COMBINED SELECTORS (NON-HIERARCHICAL)
 - SELECTOR1.SELECTOR2{ ... }
 - OR
 - SELECTOR#SELECTOR2{ ... }
- SELECTOR PRIORITY
 - SELECTOR WHICH IS MORE SPECIFIC GETS HIGHER
PRIORITY
 - ELEMENT SELECTOR
 - CLASS SELECTOR
 - TRY TO ONLY HAVE ONE CLASS FOR A HTML
ELEMENT, EVEN THOUGH
BOOTSTRAP DOESN'T FOLLOW THIS
 - ID SELECTOR
 - USE ID SPARINGLY
 - ID HAS THE HIGHEST SELECTOR PRIORITY
 - AVOID INLINE STYLING

7. INTRODUCTION TO JAVASCRIPT.

1. INTRODUCTION TO JAVASCRIPT:

- JavaScript runs in browsers as an interpreted language.
- It allows performing actions directly in the browser without sending data to the server.

2. JAVASCRIPT ALERTS:

- Display alerts using `alert("Text to Display Alert");`.

JAVASCRIPT

```
javascript
  alert("Hello, this is an alert!");
```

3. JS DATA TYPES:

- Data types include String, Boolean, and Number.
- Use `typeof<argument>` to check types.

JAVASCRIPT

```
typeof("Hello"); // Output: "string"
typeof(true);   // Output: "boolean"
typeof(42);     // Output: "number"
```

4. JS VARIABLES:

- Declare variables with `var`.
- Example: `var myName = "Anand";`.

```
javascript
var myName = "Anand";
```

5. NAMING AND NAMING CONVENTIONS FOR JS VARIABLES:

- VARIABLE NAMES CAN'T START WITH NUMBERS.
- ONLY LETTERS, NUMBERS, \$, AND _ ARE ALLOWED.
- FOLLOW CAMELCASE FORMAT.

6. STRING CONCATENATION:

- COMBINE STRINGS USING `+`.

JAVASCRIPT

```
var greeting = "Hello";
var name = "World";
var message = greeting + " " +
name; // Output: "Hello World"
```

7. STRING LENGTHS AND RETRIEVING THE NUMBER OF CHARACTERS:

- USE `STRINGVARIABLE.LENGTH` TO GET THE LENGTH OF A STRING.

JAVASCRIPT

```
javascript
var myString = "Hello";
var length = myString.length; // Output: 5
```

8. SLICING AND EXTRACTING PARTS OF A STRING:

- USE `STRING.SLICE(X, Y)` TO EXTRACT PARTS OF A STRING.

JAVASCRIPT

```
javascript
var name = "Anand";
var sliced = name.slice(0, 3); // Output: "Ana"
```

9. CHANGING CASING IN TEXT:

- USE `TOUPPERCASE()` AND `TOLOWER CASE()`.

JAVASCRIPT

```
var word = "Hello";
var upper = word.toUpperCase(); //
Output: "HELLO"
var lower = word.toLowerCase(); //
Output: "hello"
```

10. BASIC ARITHMETIC AND THE MODULO OPERATOR:

- MODULO `%` GIVES THE REMAINDER OF A DIVISION.

JAVASCRIPT

```
javascript
var remainder = 9 % 6; // Output: 3
```

11. INCREMENT AND DECREMENT EXPRESSIONS:

- USE `X++` AND `X--`.

JAVASCRIPT

```
var count = 5;
count++; // count is now 6
count--; // count is now 5
```

12. CREATING AND CALLING FUNCTIONS:

- DECLARE FUNCTIONS WITH `FUNCTION`.
- CALL FUNCTIONS USING THEIR NAME.

JAVASCRIPT

```
function sayHello() {  
    console.log("Hello!");  
}  
sayHello();
```

13. Function Parameters and Arguments:

- Functions can take parameters (placeholders for values).
- Call functions with arguments (actual values).

```
function greet(name) {  
    console.log("Hello, " + name +  
    " !");  
}  
greet("Anand"); // Output: "Hello,  
Anand!"
```

14. FUNCTION OUTPUTS AND RETURN VALUES:

- USE `RETURN` IN A FUNCTION TO PROVIDE AN OUTPUT.

JAVASCRIPT

```
javascript
function add(a, b) {
    return a + b;
}
var result = add(3, 5); // Output: 8
```

15. CONTROL STATEMENTS: IF-ELSE CONDITIONALS AND LOGIC:

- USE `IF` AND `ELSE` TO CONTROL THE FLOW BASED ON CONDITIONS.

JAVASCRIPT

```
var num = 10;
if (num > 5) {
    console.log("Number is greater
than 5.");
} else {
    console.log("Number is not greater
than 5.");
}
```

16. COMPARATORS AND EQUALITY:

- USE `===` FOR VALUE AND DATA TYPE EQUALITY.

- USE `!==` FOR NOT EQUAL.
- USE `>`, `<`, `<=`, `>=` FOR COMPARISONS.
- USE `==` FOR VALUE EQUALITY.

JAVASCRIPT

```
var x = 5;
console.log(x === "5"); // Output:
false
console.log(x == "5"); // Output:
true
```

17. COMBINING COMPARATORS:

- USE `&&` (AND), `||` (OR), `!` (NOT) FOR COMBINING CONDITIONS.

JAVASCRIPT

```
javascript
var age = 25;
var hasLicense = true;
if (age >= 18 && hasLicense) {
    console.log("You can drive!");
}
```

CONSOLE.LOG() IS A JAVASCRIPT FUNCTION USED FOR DEBUGGING AND LOGGING INFORMATION TO THE WEB BROWSER'S CONSOLE. IT'S A

FUNDAMENTAL TOOL FOR DEVELOPERS TO INSPECT AND UNDERSTAND HOW THEIR CODE IS EXECUTING. HERE'S HOW IT WORKS:

1. LOGGING DATA:

- YOU PASS ONE OR MORE VALUES, VARIABLES, OR EXPRESSIONS AS ARGUMENTS TO CONSOLE.LOG().

18. JAVASCRIPT ARRAYS:

- Create arrays with square brackets.
- Access elements using index.

JAVASCRIPT

```
javascript
var colors = ["red", "green", "blue"];
console.log(colors[0]); // Output: "red"
```

19. ADDING ELEMENTS AND INTERMEDIATE ARRAY TECHNIQUES:

- Add elements with `Array.push()`.
- Remove and retrieve elements with `Array.pop()`.

JAVASCRIPT

```
var fruits = ["apple", "banana"];
fruits.push("orange"); // ["apple",
"banana", "orange"]
var lastFruit = fruits.pop(); // "orange"
```

20. WHILE LOOPS:

- Use `while` to execute code while a condition is true.

JAVASCRIPT

```
javascript
var count = 0;
while (count < 5) {
    console.log("Count is " + count);
    count++;
}
```

21. FOR LOOPS:

- Use `for` to execute code for a specific number of iterations.

JAVASCRIPT

```
javascript
for (var i = 0; i < 5; i++) {
    console.log("Iteration " + i);
}
```

ECMASCRIPT 6 (ES6), ALSO KNOWN AS ECMASCRIPT 2015, INTRODUCED SEVERAL NEW FEATURES AND IMPROVEMENTS TO THE JAVASCRIPT LANGUAGE. HERE'S A BRIEF OVERVIEW OF THE KEY FEATURES AND CONCEPTS IN ES6:

1. BLOCK-SCOPED VARIABLES:

- LET AND CONST WERE INTRODUCED FOR DECLARING VARIABLES WITH BLOCK SCOPE, AS OPPOSED TO FUNCTION SCOPE IN THE CASE OF VAR.

2. ARROW FUNCTIONS:

- ARROW FUNCTIONS PROVIDE A CONCISE SYNTAX FOR DEFINING ANONYMOUS FUNCTIONS, MAKING THE CODE MORE READABLE.

3. TEMPLATE LITERALS:

- TEMPLATE LITERALS ALLOW YOU TO EMBED EXPRESSIONS INSIDE STRINGS USING BACKTICKS (`) AND \${}`.

4. DEFAULT PARAMETERS:

- YOU CAN DEFINE DEFAULT VALUES FOR FUNCTION PARAMETERS, SIMPLIFYING THE HANDLING OF MISSING OR UNDEFINED ARGUMENTS.

5. REST AND SPREAD OPERATORS:

- THE REST OPERATOR (...) IS USED TO COLLECT MULTIPLE FUNCTION ARGUMENTS INTO AN ARRAY. THE SPREAD OPERATOR IS USED TO SPREAD AN ARRAY INTO INDIVIDUAL VALUES.

6. DESTRUCTURING ASSIGNMENT:

- DESTRUCTURING ENABLES YOU TO EXTRACT VALUES FROM OBJECTS AND ARRAYS AND ASSIGN THEM TO VARIABLES WITH A CONCISE SYNTAX.

7. CLASSES:

- ES6 INTRODUCED CLASS SYNTAX FOR DEFINING AND CREATING OBJECTS, PROVIDING A MORE STRUCTURED WAY TO WORK WITH OBJECT-ORIENTED PROGRAMMING IN JAVASCRIPT.

8. MODULES:

- ES6 INTRODUCED A MODULE SYSTEM WITH IMPORT AND EXPORT STATEMENTS TO BETTER ORGANIZE AND MODULARIZE YOUR CODE.

9. PROMISES:

- PROMISES SIMPLIFY ASYNCHRONOUS PROGRAMMING, ALLOWING YOU TO WORK WITH ASYNCHRONOUS OPERATIONS IN A MORE READABLE AND MANAGEABLE WAY.

10. GENERATORS:

- GENERATORS ARE SPECIAL FUNCTIONS THAT CAN BE PAUSED AND RESUMED, ALLOWING YOU TO WRITE ASYNCHRONOUS CODE THAT LOOKS MORE SYNCHRONOUS.

11. ITERATORS AND ITERABLES:

- ES6 INTRODUCED THE CONCEPT OF ITERABLE OBJECTS AND ITERATORS, MAKING IT EASIER TO LOOP THROUGH DATA STRUCTURES LIKE ARRAYS, STRINGS, AND MAPS.

12. MAPS AND SETS:

- ES6 INTRODUCED TWO NEW DATA STRUCTURES, MAP AND SET, WHICH OFFER MORE FLEXIBILITY AND PERFORMANCE IN MANAGING COLLECTIONS OF DATA.

13. SYMBOL:

- THE SYMBOL DATA TYPE WAS INTRODUCED TO CREATE UNIQUE AND PRIVATE OBJECT PROPERTIES AND METHODS.

14. PROXIES AND REFLECT:

- PROXIES ALLOW YOU TO INTERCEPT AND CUSTOMIZE THE BEHAVIOR OF OBJECTS AND FUNCTIONS. REFLECT PROVIDES METHODS FOR PERFORMING OPERATIONS ON OBJECTS, MAKING IT MORE CONSISTENT AND SAFER.

15. ENHANCED OBJECT LITERALS:

- OBJECT LITERALS HAVE BEEN ENHANCED WITH FEATURES LIKE SHORTHAND PROPERTY AND METHOD DEFINITIONS.

16. PROMISES:

- PROMISES SIMPLIFY ASYNCHRONOUS OPERATIONS, MAKING IT EASIER TO HANDLE CALLBACKS AND MANAGE ASYNC TASKS IN A MORE STRUCTURED WAY.
-

~**REVISION NOTES OF INTRODUCTION TO JAVASCRIPT**

○ JAVASCRIPT RUNS ON THE BROWSER RATHER THAN THE SERVER. WHEREAS BEFORE IN ORDER TO

PERFORM ACTION, DATA AND OPERATIONS WERE BEING SENT TO THE SERVER.

○ JAVASCRIPT IS AN INTERPRETED PROGRAMMING LANGUAGE

○ JAVASCRIPT ALERTS

■ `ALERT("TEXT TO DISPLAY ALERT");`

○ JS DATA TYPES

■ DIFFERENT TYPES

● STRING

● BOOLEAN

● NUMBER

- **TYPEOF(<ARGUMENT TO CHECK TYPE>)**
- **OUTPUT OF FUNCTION**
 - “NUMBER”
 - “BOOLEAN”
 - “STRING”
 - JS VARIABLES
 - PROMPT(“DIALOGUE TO DISPLAY FOR PROMPT”);
 - TO DECLARE A VARIABLE USE, VAR.
- VAR, VARIABLES CAN LATER BE CHANGED TO A DIFFERENT VALUE
- EXAMPLE:
 - VAR MYNAME = “ANAND”;
 - NAMING AND NAMING CONVENTIONS FOR JS VARIABLES
 - VARIABLE NAMES CANNOT START WITH A NUMBER
 - VARIABLE CAN ONLY CONTAIN NUMBERS, LETTERS, \$, AND/OR _
 - VARIABLES FOLLOW A CAMEL-CASE FORMAT.
- FIRST WORD CONTAINS LOWERCASE, THEN SUBSEQUENT WORDS CONTAIN CAPITAL LETTERS FOR THEIR NAMES
 - STRING CONCATENATION
- CAN COMBINE STRINGS USING +
- EXAMPLE:
 - “HELLO” + “WORLD” -> “HELLOWORLD”
 - STRING LENGTHS AND RETRIEVING THE NUMBER OF CHARACTERS
- TO CHECK THE NUMBER OF CHARACTERS IN A STRING

- STRINGVARIABLE.LENGTH

- SLICING AND EXTRACTING PARTS OF A STRING

- SLICE FUNCTION

- STRINGSLICE(X, Y);

- SLICES THE STRING FROM X TO Y. INCLUDING X, UP TO BUT

- NOT INCLUDING Y

- EXAMPLE:

- “ANAND”.SLICE(0,3);

- “ANA”

- CHANGING CASING IN TEXT

- TOUPPERCASE()

- EXAMPLE:

- “WORD”.TOUPPERCASE()

- “WORD”

- TOLOWERCASE()

- BASIC ARITHMETIC AND THE MODULO OPERATOR IN JAVASCRIPT

- MODULO

- %

- GIVES THE REMAINDER OF A DIVISION

- EXAMPLE:

- $9 \% 6 = 3$

- INCREMENT AND DECREMENT EXPRESSIONS

- X++

- X = X + 1;

- X--

- X = X - 1;

■ CREATING AND CALLING FUNCTIONS

- DECLARING FUNCTIONS:

- FUNCTION MYFUNCTION() { <CODE TO BE EXECUTED> }

- CALLING A FUNCTION

- MYFUNCTION();

■ FUNCTION PARAMETERS AND ARGUMENTS

- EXAMPLE:

- FUNCTION GETMILK(MONEY) { ... }

■ FUNCTION OUTPUTS AND RETURN VALUES

- INCLUDE A RETURN STATEMENT IN THE FUNCTION TO MAKE IT A FUNCTION THAT

HAS AN OUTPUT

■ CONTROL STATEMENTS: IF-ELSE CONDITIONALS AND LOGIC

- IF (CONDITION) { < CODE TO RUN, IF CONDITION IS TRUE > }

- ELSE { < CODE TO RUN, IF CONDITION IS FALSE > }

■ COMPARATORS AND EQUALITY

- ===

- IS EQUAL TO

- CHECKS BOTH FOR THE VALUE AND THE DATA TYPES

- !=

- IS NOT EQUAL TO

- >, <, <=, >=
- ==
 - ONLY CHECKS IF THE VALUES EQUATE TO EACH OTHER.
 - DOESN'T TAKE INTO ACCOUNT THE DATA TYPES
 - SO, 1 == "1", WILL BE EQUAL TO TRUE
- COMBINING COMPARATORS
 - &&
- AND
 - ||
- OR
 - !
- NOT
 - JAVASCRIPT ARRAYS
- CREATING AN ARRAY
 - EXAMPLE:
 - EMPTY ARRAY
 - VAR NAMES = [];
 - PREFILLING AN ARRAY WITH ELEMENTS
 - VAR NAMES = ["BOB", "JOHN", "JENNIFER"];
 - RETRIEVING AN ELEMENT FROM AN ARRAY
 - EXAMPLE:
 - NAMES[1]; // "JOHN"
 - ARRAY ELEMENTS ALWAYS START AT 0

- RETRIEVING THE NUMBER OF ELEMENTS IN AN ARRAY
 - `ARRAY.LENGTH // RETURN THE NUMBER OF ELEMENTS`
 - CHECKING IF AN ARRAY CONTAINS A SPECIFIC ELEMENT
 - `ARRAY.INCLUDES(<ELEMENT TO CHECK IF IT EXISTS IN THE ARRAY>);`
 - RETURNS A BOOLEAN
 - ADDING ELEMENTS AND INTERMEDIATE ARRAY TECHNIQUES
 - ADDING AN ELEMENT AT THE END OF AN ARRAY
 - `ARRAY.PUSH(<ELEMENT TO PUSH INTO THE ARRAY>);`
 - REMOVE AND RETRIEVE THE LAST ELEMENT OF AN ARRAY
 - `ARRAY.POP(); // RETURNS THE LAST ELEMENT OF THE ARRAY`
 - WHILE LOOPS
 - WHILE (<SOME CONDITION IS TRUE>) { // CODE TO RUN }
 - `FOR(STARTINGCONDITION ; CONDITIONTORUNLOOPIFTRUE;`
`CONDITIONMODIFIER) { // CODE TO RUN IF CONDITION IS TRUE}`
 - EXAMPLE:
 - `FOR(I = 0; I < 100; I++) { // RUN CODE}`

8. DOCUMENT OBJECT MODEL(DOM)

1. *ADDING JAVASCRIPT TO WEBSITES*:

- *ADDING INLINE JAVASCRIPT (NOT GOOD PRACTICE)*:

- INSIDE THE `<BODY>`, ADD `ONLOAD=""`

- EXAMPLE: `<BODY ONLOAD="ALERT('HELLO!');">`

- INCLUDE A `<SCRIPT>` INSIDE THE HTML

- *EXTERNAL JAVASCRIPT*:

- INCLUDE A `<SCRIPT>` WITH A `SRC=""`

- THE PLACEMENT OF THE `<SCRIPT>` WITH THE .JS LINK MATTERS:

- INCLUDE CSS IN THE BEGINNING, INSIDE THE `<HEAD>`

- INCLUDE JAVASCRIPT AT THE BOTTOM, JUST BEFORE THE CLOSING `</BODY>` TAG

2. Intro to the Document Object Model:

- Internal Javascript:

```
// Accessing elements using DOM
var rootElement =
document.firstChild;
var headElement =
rootElement.firstChild;
var bodyElement =
rootElement.lastElementChild;
```

3. Selecting HTML Elements with JavaScript:

- getElementBy:

```
var listItems =
document.getElementsByTagName("li");
var elementsByClass =
document.getElementsByClassName("classname");
var elementById =
document.getElementById("elementId");
```

2. SELECLING HTML ELEMENTS WITH JAVASCRIPT :

.GETELEMENTBY:

JAVASCRIPT

```
var listItems = document.getElementsByTagName("li");
var elementsByClass = document.getElementsByClassName("classname");
```

```
var elementById = document.getElementById("elementId");
```

3.

- **Query Selector:**

```
var firstListItem =
document.querySelector("li:first-child");
var allListItems =
document.querySelectorAll("li");
```

4. Manipulating and Changing Styles
of HTML Elements with JavaScript:

```
var selectedElement =
document.querySelector(".selected");
selectedElement.style.color = "red";
selectedElement.style.fontSize =
"20px";
```

5. Separation of Concerns: Structure vs Style vs Behavior:

```
/* styles.css */
.highlight {
    background-color: yellow;
}
```

```
var elementToModify =
document.querySelector(".element-to-
modify");
elementToModify.classList.add("highlig
ht");
```

The Coding Wizard

6. Text Manipulation and the Text Content Property:

```
var textElement =  
document.querySelector(".text-  
element");  
var.textContent =  
textElement.textContent; // Get text  
content  
textElement.textContent = "New  
content"; // Set text content
```

7. Manipulating HTML Element Attributes:

```
var linkElement =  
document.querySelector("a");  
var hrefValue =  
linkElement.getAttribute("href");  
linkElement.setAttribute("href",  
"https://www.example.com");
```

9. Javascript Objects:

```
var person = { name: "Anand", dob:  
"12/20/1993" };  
console.log(person.name); // Output:  
Anand
```

10. Switch Statements:

```
var day = "Monday";  
switch (day) {  
    case "Monday":  
        console.log("It's Monday!");  
        break;  
    case "Tuesday":  
        console.log("It's Tuesday!");  
        break;  
    default:  
        console.log("It's another day.");  
}
```

9. ADVANCED JAVASCRIPT AND DOM MANIPULATION

11. Keyboard Event Listener:

```
document.addEventListener("keypress",  
  function(event) {  
    console.log("Key pressed: " +  
      event.key);  
  });
```

The Coding Wizard

12. Understanding Callbacks and How to Respond to Event Listener:

```
function myCallbackFunction() {  
    console.log("Callback executed!");  
}  
  
var button =  
document.querySelector("button");  
button.addEventListener("click",  
myCallbackFunction);
```

~REVISION NOTES OF DOCUMENT OBJECT MODEL (DOM)

- ADDING JAVASCRIPT TO WEBSITES
 - ADDING INLINE JAVASCRIPT (NOT GOOD PRACTICE. DON'T USE INLINE JS)
 - INSIDE THE <BODY>, ADD A ONLOAD=" <JAVASCRIPT TO EXECUTE WHEN THE PAGE LOADS> "
- EXAMPLE

- <BODY ONLOAD="ALERT('HELLO!');">

- INTERNAL JAVASCRIPT

- INCLUDE A <SCRIPT> INSIDE OF THE HTML

- EXTERNAL JAVASCRIPT

- INCLUDE A <SCRIPT> WITH A SRC=" <PATH TO .JS FILE> "
 - THE PLACEMENT OF THE <SCRIPT> WITH THE .JS LINK MATTERS!

- INCLUDE CSS IN THE BEGINNING, INSIDE THE <HEAD>

- INCLUDE JAVASCRIPT AT THE BOTTOM, JUST BEFORE THE CLOSING</BODY> TAG

- INTRO TO THE DOCUMENT OBJECT MODEL

- CATALOGS A WEBPAGE INTO INDIVIDUAL MODELS/OBJECTS WHERE WE CAN

ALTER/MANIPULATE ITS BEHAVIOR AND STYLE

- DOCUMENT IS THE ROOT OF THE WEBPAGE

- DOCUMENT.FIRSTELEMENTCHILD;

- HTML ROOT OF WEB PAGE

- DOCUMENT.FIRSTELEMENTCHILD.FIRSTELEMENTCHILD;

- <HEAD>

- DOCUMENT.FIRSTELEMENTCHILD.LASTELEMENTCHILD;

- <BODY>

- SELECTING HTML ELEMENTS WITH JAVASCRIPT

- GETELEMENTBY (BROAD SELECTOR)

- GET ELEMENTS BY TAG NAME (LI, UL, H1,)

- DOCUMENT.GETELEMENTSBYTAGNAME(< TAG >);

- GET ELEMENTS BY CLASS NAME
 - DOCUMENT.GETELEMENTSBYCLASSNAME(< CLASSNAME >);
- GET ELEMENT BY ID
 - DOCUMENT.GETELEMENTBYID(< ID >);
- QUERY SELECTOR (MORE COMPLEX AND PRECISE SELECTOR)
 - DOCUMENT.QUERYSELECTOR(< CSS SELECTOR >);
- RETURNS A SINGULAR HTML ELEMENT
- IF THE QUERY SELECTOR OBTAINS MULTIPLE ELEMENTS, THE QUERYSELECTOR ONLY RETURNS THE FIRST ELEMENT
 - DOCUMENT.QUERYSELECTORALL(< CSS SELECTOR >);
- RETURNS A LIST OF ALL THE HTML ELEMENTS THAT MATCH THE CSS SELECTOR
- MANIPULATING AND CHANGING STYLES OF HTML ELEMENTS WITH JAVASCRIPT
 - PROPERTIES MAY, AND OFTEN ARE, DIFFERENT THAN THE CSS PROPERTY.
 - TO SET THE VALUE, HAVE TO INCLUDE THE PROPERTY IN “ ”
 - CHANGING STYLE
 - <SELECTED>.STYLE.<PROPERTY> = "<NEW VALUE>";
 - CHANGING COLOR
- <SELECTED>.STYLE.COLOR = "< COLOR >";
- SEPARATION OF CONCERNS: STRUCTURE VS STYLE VS BEHAVIOR
 - CSS FILES SHOULD ONLY BE IN CHARGE OF STYLING A WEBPAGE.

- SO WE SHOULDN'T USE DOM - JAVASCRIPT TO MODIFY THE STYLE OF A WEB

PAGE.

- WHAT WE CAN DO IS, INCLUDE A STYLE MODIFIER IN THE CSS FILE, AND

USING DOM WE CAN ADD A NEW CLASS PROPERTY FOR THE .CLASSLIST TO ADD/MODIFY STYLING OF A HTML ELEMENT.

- .CLASSLIST

- RETURNS A LIST OF THE CLASSES THAT ARE ATTACHED TO THE ELEMENT

- WE CAN THEN ADD PROPERTIES TO THE LIST, TO FURTHER ADD NEW PROPERTIES FOR A HTML ELEMENT

- .CLASSLIST.ADD();

- ADD A NEW CSS STYLE PROPERTY

- .CLASSLIST.REMOVE();

- REMOVE A CSS STYLE PROPERTY

- .CLASSLIST.TOGGLE();

- TOGGLE'S A CSS STYLE PROPERTY ON/OFF.

- TEXT MANIPULATION AND THE TEXT CONTENT PROPERTY

- <SELECTED>.TEXTCONTENT

- RETURNS THE ACTUAL TEXT STRING VALUE

- <SELECTED>.INNERHTML

- RETURNS THE HTML CODE

- CAN ALSO MODIFY THE HTML

- MANIPULATING HTML ELEMENT ATTRIBUTES

- ATTRIBUTES CONSIST OF THINGS LIKE, CLASS=, HREF=, SRC=, ...
- <SELECTED>.ATTRIBUTES
 - RETURNS THE MAPPING OF ALL THE ATTRIBUTES
- <SELECTED>.GETATTRIBUTE(" < ATTRIBUTE TO GET > ");
- <SELECTED>.SETATTRIBUTE(" < WHICH ATTRIBUTE TO CHANGE > ", "< VALUE TO CHANGE THE ATTRIBUTE TO > ");
- ADVANCED JAVASCRIPT AND DOM MANIPULATION

- ADDING EVENT LISTENERS TO A BUTTON

- <EVENTTARGET>.ADDEVENTLISTENER(<TYPE>, <LISTENER>);
 - <TYPE> CONSISTS OF A VARIETY OF DIFFERENT TYPES TO CAUSE TO EXECUTE THE EVENT LISTENER
- E.G. "CLICK"
 - <LISTENER> USUALLY A JAVASCRIPT FUNCTION
- CAN ALSO BE AN ANONYMOUS FUNCTION
- HIGHER ORDER FUNCTIONS AND PASSING FUNCTIONS AS ARGUMENTS
- HOW TO PLAY SOUNDS ON A WEBSITE

- CREATE A NEW AUDIO(< FILE PATH >);
- CALL THE .PLAY(); OF THE AUDIO OBJECT
- JAVASCRIPT OBJECTS
 - JAVASCRIPT OBJECTS HAVE ONE OR MORE FIELDS : DATA
 - EXAMPLE:

- VAR ANAND = { NAME : "ANAND", DOB : "12/20/1993"}
- CONSTRUCTOR
- FUNCTION <NAME OF CLASS>(< ARGUMENTS >) { <THIS.FIELD = ARGUMENT;> }
 - USE THE 'NEW' KEYWORD TO CREATE AN OBJECT
- SWITCH STATEMENTS
 - SWITCH (EXPRESSION)
- CASE < >:
 - RUN THE CODE FOR THE CASE
 - BREAK
- DEFAULT:
 - "ELSE STATEMENT"
 - KEYBOARD EVENT LISTENER
 - DOCUMENT.ADEVENTLISTENER('KEYPRESS', HANDLER)
 - UNDERSTANDING CALLBACKS AND HOW TO RESPOND TO EVENT LISTENER
 - CALLBACK FUNCTION
 - FUNCTION THAT GETS PASSED IN AS AN INPUT

22. REACT.JS

1. Introduction to React:

React is a JavaScript library for building user interfaces.
The most basic example of a React component:

```
import React from 'react';

function App() {
  return <div>Hello, React!</div>;
}
```

2. Components:

Functional Components:

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

Class Components:

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

3. JSX (JavaScript XML):

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

4. Props:

```
function Welcome(props) {
```

```
return <h1>Welcome, {props.name}!</h1>;
}
```

```
// Usage
<Welcome name="Alice" />
```

5. State:

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

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Increment
        </button>
      </div>
    );
  }
}
```

6. Lifecycle Methods:

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

  componentDidMount() {
    this.timerID = setInterval(() => {
      this.setState({ time: this.state.time + 1 });
    }, 1000);
  }
}
```

```
componentWillUnmount() {  
  clearInterval(this.timerID);  
}  
  
render() {  
  return <div>Time: {this.state.time} seconds</div>;  
}  
}
```

7. Event Handling:

```
class Button extends React.Component {  
  handleClick() {  
    alert('Button Clicked');  
  }  
  
  render() {  
    return <button onClick={this.handleClick}>Click me</button>;  
  }  
}
```

8. Conditional Rendering:

```
function Greeting(props) {  
  if (props.isLoggedIn) {  
    return <h1>Welcome back!</h1>;  
  } else {  
    return <h1>Please sign up.</h1>;  
  }  
}
```

9. Lists and Keys:

```
function ToDoList(props) {  
  const items = props.items.map((item, index) =>  
    <li key={index}>{item.text}</li>  
  );  
  return <ul>{items}</ul>;  
}
```

}

10. Forms and Controlled Components:

```
class MyForm extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { value: "" };  
  }  
  
  handleChange(event) {  
    this.setState({ value: event.target.value });  
  }  
  
  handleSubmit(event) {  
    alert('A name was submitted: ' + this.state.value);  
    event.preventDefault();  
  }  
  
  render() {  
    return (  
      <form onSubmit={this.handleSubmit}>  
        <input type="text" value={this.state.value} onChange={this.handleChange}>  
      </form>  
    );  
  }  
}
```

11. Component Composition:

Nesting Components for complex UI:

```
function Header() {  
  return <h1>Header</h1>;  
}
```

```
function Footer() {  
  return <footer>Footer</footer>;  
}  
  
function App() {  
  return (  
    <div>  
      <Header />  
      <main>  
        <p>Content goes here.</p>  
      </main>  
      <Footer />  
    </div>  
  );  
}
```

12. Styling in React:

Using inline styles with React:

```
const style = {  
  backgroundColor: 'lightblue',  
  color: 'darkblue',  
};  
  
function StyledComponent() {  
  return <div style={style}>Styled Component</div>;  
}
```

13. State Management (Local State vs. Global State):

Using useState for local state:

```
import React, { useState } from 'react';  
  
function Counter() {  
  const [count, setCount] = useState(0);  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
    </div>  
  );  
}
```

```
<button onClick={() => setCount(count + 1)}>Increment</button>
</div>
);
}
```

14. Routing (with React Router):

Setting up routing with React Router:

```
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';

function Home() {
  return <div>Home Page</div>;
}

function About() {
  return <div>About Page</div>;
}

function App() {
  return (
    <Router>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
        </ul>
      </nav>

      <Route exact path="/" component={Home} />
      <Route path="/about" component={About} />
    </Router>
  );
}
```

15. API Integration:

Fetching data from an API:

```
import React, { useState, useEffect } from 'react';

function App() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then((response) => response.json())
      .then((data) => setData(data));
  }, []);

  return (
    <ul>
      {data.map((item) => (
        <li key={item.id}>{item.title}</li>
      ))}
    </ul>
  );
}


```

16. Context API:

Creating and consuming a context:

```
const ThemeContext = React.createContext('light');

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Header />
      <Main />
      <Footer />
    </ThemeContext.Provider>
  );
}

function Header() {
  const theme = useContext(ThemeContext);
  return <header style={{ background: theme }}>Header</header>;
}
```

17. Hooks:

Using useState and useEffect:

```
import React, { useState, useEffect } from 'react';

function Timer() {
  const [time, setTime] = useState(0);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setTime((prevTime) => prevTime + 1);
    }, 1000);

    return () => {
      clearInterval(intervalId);
    };
  }, []);

  return <div>Time: {time} seconds</div>;
}
```

18. Redux (Optional):

Setting up Redux with actions, reducers, and store.

19. Component Testing:

Writing tests for React components using testing libraries like Jest and React Testing Library.

20. Deployment:

Building and deploying a React app to a hosting platform.

These examples provide code snippets for React concepts 11 to 20. Continue to explore these topics and integrate them into your React projects for a deeper understanding.

React Notes:

- What is React?

- Front-end javascript library that makes it easier to make user interfaces
- Very popular in 2019
- Benefits
 - Re-usability
 - Organization
 - Concise code
 - Scales well
- Components
 - Contains its own html, css, js
 - Can update itself without affecting other components

- Intro to Code Sandbox and Structure to the Module

- Code Sandbox is a nice to use IDE on the browser that contains all the setup needed for coding

- Intro to JSX and Babel

- Every react html has a <div id="root">
- <ReactDOM>.render()
 - Inserts HTML into the website, through the javascript file/code
 - Arguments

- 1. What to show

- Note: only takes 1 element

- 2. Where to show

- In this case, document.getElementById('root')

- 3. Callback

- React creates JSX files
 - HTML code inside of a Javascript file
- Babel
 - Javascript compiler inside of React module
 - Takes any Javascript code and converts it into an equivalent Javascript code that can be read by any browser

- Javascript Expressions in JSX & ES6 Template Literals

- We can insert Javascript inside an HTML inside of a javascript
 - {<Javascript code>}

- Only values or expressions can be placed, not statements

- JSX Attributes & Styling React Elements

- In the JSX file, instead of using "class=", we would have to use "className="

- HTML attributes for JSX must be camelCase
- Inline Styling for React Elements
 - Even though it looks like HTML, it is not, it's JSX
 - For inline styling, we need to pass in a javascript object
- React Components
 - Components can help us break down a large codebase into smaller ones, plus the benefit of reuseability
 - To create a React Component, create a function that has the naming convention in Pascal form
 - Create a new file with .jsx extension
 - In the index.js file, import the .jsx file and use the component like how you would use an HTML element
 - Normally, React app's in their index.js have an App component, which comprises of all of the UI
 - All the components will have their own .jsx file
 - And all the components will be organized into sub-directories
- Javascript ES6 - Import, Export and Modules
 - export default <reference>
 - There can only be 1 default export for a .jsx file
 - Export multiple functions, values, objects,
 - export {<ref 1, ref 2, ...>}
 - import * as <ref> from "<.jsx file>"
 - wildcard import is discouraged, as it reduces readability
- Starting up a React app
- React Props
 - Is a method to pass in parameters from JSX html to the React functions
 - We can also pass in functions as well
 - name, imageURL, tel, email are like parameters that get passed to Card, with all the parameters inside of props
- Props
 - Creates a new javascript object with parameters
- React Dev Tools
 - Displays React DOM Tree
 - Install extension: React Developer Tools
- Mapping Data to Components
 - .map()
 - "key" - unique key value for props to distinguish array values
 - Only used by the backend in React

- Javascript ES6 Map/Filter/Reduce

- Map

- <Array>.map(<Function to perform on individual items in the array>)
 - Creates a new array by performing function on items in the original array

- Filter

- <Array>.filter(<Function conditional that takes in item, and evaluates true or false>)
 - Creates a new array by keeping the items that evaluate to true

- Reduce

- <Array>.reduce(<Function(accumulator, currentValue) to perform on each item in the array>)
 - Creates an accumulated value, based on operation to each item in the original array

- Find

- <Array>.find(<Function>)
 - Finds and returns the first item that matches the condition

- FindIndex

- <Array>.findIndex(<Function>)
 - Finds and returns the first item index that matches the condition

- Javascript ES6 Arrow Functions (Fat Arrow)

- Shorter way of writing a javascript function

- Can omit the keyword function

- Used for anonymous functions
 - 1 parameter (no need for parentheses)

- <parm> => {function body}

- 2 or more parameters (need parentheses)

- (<parm1, parm2> => {function body})

- If function body is only 1 line, then we can omit the {} and “return”

- React Conditional Rendering with Ternary Operations & AND

- Ternary Operator

- Ternary Operator is treated as an expression, so it can be placed inside the JSX file inside with HTML-looking code

- In Javascript, we can do a similar method to ternary operator by using && because if the first condition in && is true, the second condition will be executed

- If currentTime is greater than true, the <h1> will be called

- State in React - Declarative vs Imperative Programming

- Declarative
- React uses
 - UI is dependent on a conditions of a state variable
 - Given <statement>, the <view> should be ...
 - Benefits
 - Easier to reason about
 - Fewer things can go wrong
 - Cons
 - Needs to re-do a lot of work all the time
 - Expensive
 - Can create brand new ui elements and rerendering
 - Imperative
 - Getting hold of an item and setting it's attribute to a new value
 - React Hooks - useState
 - Adds dynamic interactivity
 - Must use a hook inside a component
 - Each state is local to a component
 - useState(<parm>)
 - parm - Starting state value
 - Returns an array with 2 items
 - 1. State value
 - 2.
 - Benefits
 - Will only re-render the ui element that is needed
 - Javascript Object & Array Destructuring
 - Data
 - Destructuring complex data
 - Event Handling in React
 - HTML Event Attributes
 - https://www.w3schools.com/html/html_attributes.asp
 - Event Attributes can add more event handling types, such as onmouseout, onclick, ...
 - React Forms
 - <input onChange=>
 - Controlled Components
 - <https://reactjs.org/docs/forms.html#controlled-components>
 - <form> when submitted, will refresh to make a POST or a GET request

- Will call method called onSubmit
 - event.preventDefault()
- Prevents the default next step of the event
- 1. Override <form onSubmit=>
- 2. Add preventDefault
 - Class Components vs Functional Components
 - Hooks vs Classes
- Class
 - class App extends React.Component { ... render(){
... } }
- Hook
 - function App() { ... }
 - Can only use for Components
 - Makes it easier to maintain state
 - Changing Complex State
 - We can have useState store an object to initialize
 - In the update function of useState()
- We can pass in an anonymous function, which has the previous value as the parameter
 - JavaScript ES6 Spread Operator
- Can work with arrays and object inserts
- Gives easy functionality to insert arrays and objects into other arrays and objects
 - Managing a Component Tree
 - Can pass in function pointers in props from parent components to child components

10. JQUERY

WHAT IS JQUERY?

- JQUERY IS A POPULAR JAVASCRIPT LIBRARY THAT SIMPLIFIES DOM MANIPULATION.

- TRADITIONAL DOM QUERY:

JAVASCRIPT

```
const traditionalElement = document.querySelector("h1");
```

- JQUERY EQUIVALENT:

JAVASCRIPT

```
const jQueryElement = $("h1");
```

INCORPORATING JQUERY INTO WEBSITES:

- USE A CONTENT DELIVERY NETWORK (CDN) FOR SIMPLICITY.

- PLACE THE JQUERY CDN SCRIPT ABOVE THE CLOSING `</BODY>` TAG IN YOUR HTML FILE:

HTML

```
<script  
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

SELECTING ELEMENTS WITH JQUERY:

- USE CSS SELECTORS TO TARGET ELEMENTS.
- SELECTING ALL ELEMENTS WITH A CLASS `EXAMPLE`:

JAVASCRIPT

```
const selectedElements = $(".example");
```

MANIPULATING STYLES WITH JQUERY:

- SETTING A STYLE ATTRIBUTE USING JQUERY:

JAVASCRIPT

```
$("#h1").css("color", "red");
```

SEPARATE CONCERNS:

- CREATE CSS CLASSES IN A `*.CSS` FILE.
- USE JQUERY TO APPLY STYLES USING CLASS NAMES:

JAVASCRIPT

```
$("#h1").addClass("highlight");
```

MANIPULATING TEXT WITH JQUERY:

- CHANGING THE TEXT CONTENT OF A SELECTED ELEMENT:

JAVASCRIPT

```
$("#p").text("New text content.");
```

MANIPULATING ATTRIBUTES WITH JQUERY:

- SETTING THE `SRC` ATTRIBUTE OF AN IMAGE:

JAVASCRIPT

```
$(“img”).attr(“src”, “new-image.jpg”);
```

ADDING EVENT LISTENERS WITH JQUERY:

- ADDING A CLICK EVENT LISTENER:

JAVASCRIPT

```
$(“button”).click(function() {  
    alert(“Button clicked!”);  
});
```

ADDING AND REMOVING ELEMENTS WITH JQUERY:

- ADDING A NEW ELEMENT BEFORE A SELECTED ELEMENT:

JAVASCRIPT

```
$(“<div>New element</div>”).insertBefore(“.existing-element”);
```

WEB ANIMATIONS WITH JQUERY:

- HIDING AN ELEMENT:

JAVASCRIPT

```
$(“.element”).hide();
```

- FADING IN AN ELEMENT:

JAVASCRIPT

```
$(".element").fadeIn();
```

- TOGGLING SLIDE ANIMATION:

JAVASCRIPT

```
$(".element").slideToggle();
```

ANIMATION CONTROL WITH JQUERY:

- ANIMATING MULTIPLE PROPERTIES SEQUENTIALLY:

JAVASCRIPT

```
$("#h1")  
  .slideUp()  
  .slideDown()  
  .animate({ opacity: 0.5 });
```

~DEEP REVISION NOTES OF JQUERY

○ WHAT IS JQUERY?

- THE MOST POPULAR JS LIBRARY
- TAKES THE TRADITIONAL DOM MANIPULATION METHODS, AND MAKES IT EASIER AND

EFFICIENT IN LESS LINES OF CODE

- EXAMPLE:

- TRADITIONAL

○ DOCUMENT.QUERYSELECTOR("H1")

- JQUERY
 - \$(“H1”)
 - HOW TO INCORPORATE JQUERY INTO WEBSITES
 - EASIEST WAY IS TO USE A CDN METHOD
 - CDN IS BASICALLY A NETWORK OF CONNECTED SERVERS. CDN THEN FINDS THE EASIEST AND FASTEST METHOD TO RETRIEVE FILES.
 - PLACE THE JQUERY CDN SCRIPT ABOVE THE JAVASCRIPT AT DIRECTLY ABOVE THE ENDING BODY TAG
 - <SCRIPT SRC="HTTPS://AJAX.GOOGLEAPI.COM/AJAX/LIBS/JQUERY/3.3.1/JQUERY.MIN.JS"></SCRIPT>
 - SELECTING ELEMENTS WITH JQUERY
 - JQUERY(“ < CSS SELECTOR > ”)
 - OR
 - \$(" < CSS SELECTOR > ")
 - \$() == JQUERY()
 - THERE IS NO DIFFERENCE BETWEEN SELECTING ALL ELEMENTS VS ONE ELEMENT
 - MANIPULATING STYLES WITH JQUERY
 - TO SET THE STYLE
 - <JQUERY SELECTED>.CSS(< ATTRIBUTE TO CHANGE>, <WHAT TO CHANGE IT TO>)
 - EXAMPLE

- \$(“H1”).CSS(“COLOR”, “RED”);

- TO GET THE STYLE

- <JQUERY SELECTED>.CSS(< ATTRIBUTE TO GET >);

- SEPARATE CONCERNS

- JQUERY IS IMPLEMENTED IN THE .JS FILE, AND WE CAN ALSO USE IT TO STYLE THE PAGE

- BUT STYLING THE PAGE IN A .JS FILE IS BAD PRACTICE

- SO THE CORRECT THING TO DO IS CREATE A CSS CLASS INSIDE THE STYLES.CSS, AND USE JQUERY TO SET THE STYLE TO THAT CLASS IN THE STYLES.CSS VIA .ADDCLASS()

- ADD/REMOVE CLASS

- <JQUERY SELECTED>.ADDCLASS(< CSS CLASS TO CHANGE STYLE TO, CAN BE MULTIPLE CLASSES WHICH ARE SEPARATED BY A SINGLE SPACE>);

- <JQUERY SELECTED>.REMOVECLASS(< CSS CLASS TO CHANGE STYLE TO, CAN BE MULTIPLE CLASSES WHICH ARE SEPARATED BY A SINGLE SPACE >);

- CHECK IF AN ELEMENT HAS A CLASS

- <JQUERY SELECTED>.HASCLASS(< CLASS TO CHECK IF EXISTS >)

- MANIPULATING TEXT WITH JQUERY

- 2 METHODS

- 1. <JQUERY SELECTED>.TEXT(< NEW TEXT>);

- 2. <JQUERY SELECTED>.HTML(< NEW HTML >);

- MANIPULATING ATTRIBUTES WITH JQUERY

- SET ATTRIBUTE
 - <JQUERY SELECTED>.ATTR(< ATTRIBUTE TO CHANGE>, < NEW ATTRIBUTE VALUE>)
- GET ATTRIBUTE
 - <JQUERY SELECTED>.ATTR(< ATTRIBUTE TO CHECK>)
- ADDING EVENT LISTENERS WITH JQUERY
 - CLICK LISTENER
 - <JQUERY SELECTED>.CLICK(< CALLBACK FUNCTION>);
 - KEY PRESS LISTENER
 - <JQUERY SELECTED>.KEYPRESS(< CALLBACK FUNCTION>);
 - GENERAL/FLEXIBLE METHOD
 - <JQUERY SELECTED>.ON(< EVENT > ,< CALLBACK FUNCTION>);
- ADDING AND REMOVING ELEMENTS WITH JQUERY
 - ADDING AN ELEMENT
 - ADDING A NEW ELEMENT BEFORE A SELECTED ELEMENT
 - <JQUERY SELECTED>.BEFORE(< HTML OF NEW ELEMENT >);
 - ADDING A NEW ELEMENT AFTER A SELECTED ELEMENT
 - <JQUERY SELECTED>.AFTER(< HTML OF NEW ELEMENT >);
 - PREPEND A NEW ELEMENT INSIDE OF A SELECTED ELEMENT
 - <JQUERY SELECTED>.PREPEND(< HTML OF NEW ELEMENT >);

- APPEND A NEW ELEMENT INSIDE OF A SELECTED ELEMENT

- <JQUERY SELECTED>.APPEND(< HTML OF NEW ELEMENT >);

- WEB ANIMATIONS WITH JQUERY

- HIDING AN ELEMENT

- <JQUERY SELECTED>.HIDE();
 - ELEMENT GETS REMOVED FROM THE FLOW OF THE HTML

STRUCTURE OF THE WEB PAGE

- SHOW AN ELEMENT

- <JQUERY SELECTED>.SHOW();

- TOGGLE HIDE AND SHOW

- <JQUERY SELECTED>.TOGGLE();

- FADEOUT

- HIDES THE ELEMENT BUT IN A LESS SUDDEN WAY
 - <JQUERY SELECTED>.FADEOUT();

- FADEIN

- SHOWS THE ELEMENT BUT IN A LESS SUDDEN WAY
 - <JQUERY SELECTED>.FADEIN();

- TOGGLE FADE IN AND FADE OUT

- <JQUERY SELECTED>.FADETOGGLE();

- SLIDE UP

- COLLAPSES AN ELEMENT
 - <JQUERY SELECTED>.SLIDEUP();

- SLIDE DOWN
 - UNCOLLAPSE AN ELEMENT
 - <JQUERY SELECTED>.SLIDEDOWN();
- TOGGLE SLIDE IN AND SLIDE OUT
 - <JQUERY SELECTED>.SLIDETOOGLE();
- METHOD FOR MORE CONTROL OVER ANIMATIONS
 - <JQUERY SELECTED>.ANIMATE({ < NEW CSS RULE TO ANIMATE>});
 - CAN ONLY ADD CSS RULES THAT HAVE A NUMERIC VALUE
- CAN ALSO CHAIN ANIMATION METHODS BACK TO BACK
 - EXAMPLE:
 - \$("H1").SLIDEUP().SLIDEDOWN().ANIMATE({OPACITY: 0.5});

11. NODEJS

1. INTRODUCTION TO NODE.JS:

NODE.JS IS A SERVER-SIDE RUNTIME ENVIRONMENT FOR EXECUTING JAVASCRIPT CODE. IT'S IDEAL FOR BUILDING SCALABLE, NETWORK APPLICATIONS. HERE'S A SIMPLE "HELLO, NODE.JS" EXAMPLE:

JAVASCRIPT

CODE

```
// HELLO.JS
```

```
console.log('Hello, Node.js!');
```

2. SETTING UP NODE.JS:

INSTALLING NODE.JS CAN BE DONE BY DOWNLOADING IT FROM THE OFFICIAL WEBSITE. VERIFY THE INSTALLATION WITH:

BASH

CODE

```
NODE -V
```

3. NODE.JS MODULES:

NODE.JS USES COMMONJS MODULES FOR ORGANIZING CODE. HERE'S HOW TO CREATE AND USE A SIMPLE MODULE:

JAVASCRIPT

CODE

```
// math.js
const add = (a, b) => a + b;
module.exports = add;

// app.js
const addition = require('./math');
console.log(addition(2, 3)); // Output: 5
```

4. ASYNCHRONOUS PROGRAMMING:

NODE.JS IS KNOWN FOR ITS NON-BLOCKING, EVENT-DRIVEN ARCHITECTURE.
HERE'S AN EXAMPLE OF USING CALLBACKS FOR ASYNCHRONOUS TASKS:

JAVASCRIPT

CODE

```
function fetchData(callback) {
  setTimeout(() => {
    callback('Data received');
  }, 2000);
}

fetchData((data) => {
  console.log(data); // Output: Data received
});
```

5. FILE SYSTEM (FS):

NODE.JS PROVIDES MODULES LIKE FS TO WORK WITH THE FILE SYSTEM.
HERE'S HOW TO READ A FILE:

JAVASCRIPT

CODE

```
const fs = require('fs');

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

6. HTTP AND WEB SERVERS:

NODE.JS CAN CREATE WEB SERVERS. HERE'S A BASIC HTTP SERVER EXAMPLE:

JAVASCRIPT

CODE

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, HTTP server!');
});

server.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

7. STREAMS AND BUFFERS:

STREAMS AND BUFFERS ARE CRUCIAL FOR EFFICIENTLY HANDLING DATA.
HERE'S A SIMPLE EXAMPLE OF READING A FILE USING STREAMS AND BUFFERS:

JAVASCRIPT

CODE

```
const fs = require('fs');
const readStream = fs.createReadStream('file.txt');

let data = "";

readStream.on('data', (chunk) => {
  data += chunk;
});

readStream.on('end', () => {
  console.log(data);
});
```

8. EVENT EMITTERS:

NODE.JS RELIES HEAVILY ON THE EVENT-DRIVEN ARCHITECTURE. HERE'S HOW TO CREATE AND USE AN EVENT Emitter:

JAVASCRIPT

CODE

```
const EventEmitter = require('events');
const myEmitter = new EventEmitter();

myEmitter.on('customEvent', (arg) => {
  console.log(`Event occurred with data: ${arg}`);
});
```

```
myEmitter.emit('customEvent', 'Some data');
```

9. WORKING WITH DATABASES:

NODE.JS CAN INTERACT WITH VARIOUS DATABASES. HERE'S AN EXAMPLE OF USING THE MONGODB DRIVER TO CONNECT TO A MONGODB DATABASE:

JAVASCRIPT

CODE

```
const { MongoClient } = require('mongodb');
const url = 'mongodb://localhost:27017';
const dbName = 'mydb';

MongoClient.connect(url, { useNewUrlParser: true }, (err, client) => {
  if (err) throw err;
  const db = client.db(dbName);
  // Perform database operations
  client.close();
});
```

10. PACKAGE MANAGEMENT WITH NPM:

NPM IS THE PACKAGE MANAGER FOR NODE.JS. YOU CAN INSTALL, MANAGE, AND PUBLISH PACKAGES. FOR EXAMPLE, TO INSTALL THE EXPRESS PACKAGE:

BASH

CODE

```
npm install express
```

11. MIDDLEWARE AND EXPRESS.JS:

EXPRESS.JS IS A POPULAR WEB APPLICATION FRAMEWORK FOR NODE.JS. HERE'S HOW TO USE MIDDLEWARE IN AN EXPRESS APPLICATION:

JAVASCRIPT

CODE

```
const express = require('express');
const app = express();

app.use((req, res, next) => {
  console.log('Middleware executed');
  next();
});

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

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

12. AUTHENTICATION AND AUTHORIZATION:

IMPLEMENTING USER AUTHENTICATION AND AUTHORIZATION CAN BE COMPLEX BUT IS ESSENTIAL FOR SECURE APPLICATIONS. LIBRARIES LIKE PASSPORT HELP WITH AUTHENTICATION.

13. ERROR HANDLING:

NODE.JS APPLICATIONS SHOULD HANDLE ERRORS GRACEFULLY. HERE'S A SIMPLE ERROR-HANDLING MIDDLEWARE IN EXPRESS:

JAVASCRIPT

CODE

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something went wrong!');
});
```

14. REAL-TIME APPLICATIONS:

FOR REAL-TIME FEATURES, YOU CAN USE WEBSOCKETS WITH LIBRARIES LIKE SOCKET.IO.

15. SCALABILITY AND CLUSTERING:

NODE.JS APPLICATIONS CAN TAKE ADVANTAGE OF MULTIPLE CORES USING CLUSTERING. THE CLUSTER MODULE ALLOWS YOU TO CREATE A CLUSTER OF NODE.JS PROCESSES.

16. RESTFUL API DESIGN:

DESIGNING A RESTFUL API IS CRUCIAL FOR BUILDING WEB SERVICES. YOU DEFINE ENDPOINTS AND USE HTTP VERBS (GET, POST, PUT, DELETE) TO PERFORM ACTIONS.

17. SECURITY BEST PRACTICES:

NODE.JS APPLICATIONS SHOULD FOLLOW BEST PRACTICES TO PREVENT SECURITY VULNERABILITIES LIKE SQL INJECTION AND XSS. SECURE DEPENDENCIES AND USE AUTHENTICATION AND AUTHORIZATION.

18. TESTING AND DEBUGGING:

UNIT TESTING IS ESSENTIAL TO VERIFY YOUR APPLICATION'S CORRECTNESS. TOOLS LIKE MOCHA AND CHAI HELP WITH TESTING. NODE.JS OFFERS BUILT-IN DEBUGGING TOOLS AND CAN BE INTEGRATED WITH IDES.

19. DEPLOYMENT AND HOSTING:

DEPLOY NODE.JS APPLICATIONS ON CLOUD PLATFORMS LIKE AWS, HEROKU, OR AZURE. DOCKER CAN BE USED TO CONTAINERIZE NODE.JS APPS.

20. NODE.JS ECOSYSTEM:

NODE.JS CAN BE USED WITH FRONT-END FRAMEWORKS LIKE NEXT.JS FOR SERVER-SIDE RENDERING AND WITH SERVERLESS PLATFORMS LIKE AWS LAMBDA FOR FUNCTION EXECUTION.

SOME NOTES FOR REVISION:

- WHAT IS NODE.JS?
 - ALLOWS THE CREATION OF A BACKEND USING JAVASCRIPT
 - FAST EXECUTION ON THE BACKEND
 - ALLOWS JAVASCRIPT TO BE RUN OUTSIDE OF THE BROWSER, INDEPENDENTLY, AND TO ALLOW JAVASCRIPT CODE TO INTERACT DIRECTLY WITH THE COMPUTER, THAT WILL ALLOW

IT TO ACCESS FILES, LISTEN TO NETWORK REQUESTS, ...

- ALLOWS THE CREATION OF FULL DESKTOP APPLICATIONS USING JAVASCRIPT
 - ALLOWS THE EXECUTION OF JAVASCRIPT PROGRAMS ON THE SERVER
 - ALL OF THE HEAVY CALCULATIONS AND EXECUTIONS HAPPEN ON THE SERVER,
 - RATHER THAN THE CLIENT'S BROWSER
 - THE NODE REPL (READ EVALUATION PRINT LOOPS)
 - HOW TO USE THE NATIVE NODE MODULES
 - CHECK OUT THE DIFFERENT TYPES OF API'S ON NODEJS.ORG/API/
 - ACCESSING FILE SYSTEMS
 - CREATE A FILE SYSTEM OBJECT
 - EXAMPLE:
 - CONST FS = REQUIRE("FS");
 - REQUIRE(<"/>);
 - NOT PART OF STANDARD JAVASCRIPT
 - IN NODEJS, A BUILT IN FUNCTION TO LOAD MODULES INTO THE JAVASCRIPT CODE
 - THE NPM PACKAGE MANAGER AND INSTALLING EXTERNAL NODE MODULES
 - NPM (NODE PACKAGE MANAGER)
 - PACKAGE MANAGERS FOR EXTERNAL MODULES
 - GETS BUNDLED WITH NODE
 - CREATING YOUR OWN PACKAGE

- RUN NODE INIT AND FOLLOW THE STEPS
 - IT WILL CREATE A PACKAGE.JSON FILE
- DOWNLOADING AN EXTERNAL NODE MODULE
- FIND THE MODULE YOU WANT FROM SEARCHING FROM NPMJS.COM
 - CHECK THE USAGE, USUALLY IT IS IN THE FORM OF NPM INSTALL <KEYWORD>
-

12. EXPRESS JS

WHAT IS EXPRESS?

- EXPRESS IS A NODE.JS FRAMEWORK DESIGNED TO SIMPLIFY WRITING WEB APPLICATIONS.
- IT REDUCES REPETITIVE CODE AND IS TAILORED FOR WEB DEVELOPERS.

CREATING OUR FIRST SERVER WITH EXPRESS:

1. INITIALIZE A NEW NODE.JS PROJECT: `NPM INIT`.
2. INSTALL EXPRESS: `NPM INSTALL EXPRESS`.
3. IN `SERVER.JS`:

JAVASCRIPT

```
javascript
// Import the express module
const express = require('express');

// Create an instance of the express application
const app = express();

// Listen on a specific port (e.g., 3000)
app.listen(3000);
```

Creating Our First Server with Express:

```
const express = require('express');
const app = express();
const port = 3000;

app.listen(port, () => {
  console.log(`Server started on port
${port}`);
});
```

HANDLING REQUESTS AND RESPONSES: THE GET REQUEST:

- THE ROOT PAGE (`/`) CORRESPONDS TO THE HOME PAGE.
- DEFINE A ROUTE TO HANDLE THE ROOT PAGE REQUEST:

JAVASCRIPT

```
APP.GET('/', function(req, res) {
  // CODE TO EXECUTE WHEN A GET REQUEST TO ROOT OCCURS
});
```

UNDERSTANDING AND WORKING WITH ROUTES:

- DEFINE ROUTES USING `APP.GET(<ROUTE>, <CALLBACK>)`.
- `<ROUTE>` SPECIFIES THE URL PATH TO RESPOND TO.

Understanding and Working with Routes:

```
app.get('/about', (req, res) => {
  res.send('About Us Page');
});

app.get('/contact', (req, res) => {
  res.send('Contact Us Page');
});
```

- EXAMPLE:

JAVASCRIPT

```
APP.GET('/ABOUT', FUNCTION(REQ, RES) {
  // CODE TO EXECUTE WHEN A GET REQUEST TO /ABOUT OCCURS
});
```

RESPONDING TO REQUESTS WITH HTML FILES:

- USE `RES.SENDFILE()` TO SEND AN HTML FILE AS A RESPONSE:

JAVASCRIPT

```
javascript
const path = require('path'); // Import the 'path' module
app.get('/contact', function(req, res) {
  res.sendFile(path.join(__dirname, 'contact.html'));
});
```

Responding to Requests with HTML Files:

```
const path = require('path');

app.get('/home', (req, res) => {
  res.sendFile(path.join(__dirname,
  'home.html'));
});
```

PROCESSING POST REQUESTS WITH BODY PARSER:

1. CREATE A FORM IN HTML:

HTML

```
html
<form action="/contact" method="POST">
  <input type="text" name="message">
  <button type="submit">Send</button>
</form>
```

Processing POST Requests with Body Parser:

```
const bodyParser = require('body-parser');

app.use(bodyParser.urlencoded({ extended: true }));

app.post('/submit', (req, res) => {
  const message = req.body.message;
  res.send(`Received: ${message}`);
});
```

2. SET UP POST HANDLING WITH EXPRESS:

JAVASCRIPT

```
javascript
const bodyParser = require('body-parser'); // Import body-parser
app.use(bodyParser.urlencoded({ extended: true })); // Use bodyParser

app.post('/contact', function(req, res) {
  const message = req.body.message; // Access the posted data
  // Code to handle the POST request
});
```

UNDERSTANDING `BODY-PARSER` MODES:

- `BODY-PARSER` HELPS PARSE INCOMING DATA.
- MODES: `BODYPARSER.TEXT()`, `BODYPARSER.JSON()`, `BODYPARSER.URLENCODED()`.

Understanding body-parser Modes:

```
app.use(bodyParser.text()); // Parse  
text data  
  
app.use(bodyParser.json()); // Parse  
JSON data  
  
app.use(bodyParser.urlencoded({ extend  
ed: true })); // Parse URL-encoded  
data
```

1. MIDDLEWARE FUNCTIONS:

MIDDLEWARE FUNCTIONS ARE AT THE HEART OF EXPRESS. THEY CAN PERFORM TASKS SUCH AS PARSING DATA, AUTHENTICATION, AND HANDLING ERRORS.

JAVASCRIPT

```
const express = require('express');  
const app = express();  
  
// Middleware to parse JSON data  
app.use(express.json());  
  
// Custom middleware  
app.use((req, res, next) => {  
  console.log('Middleware executed');  
  next();  
});
```

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

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

2. ROUTING:

EXPRESS ALLOWS YOU TO CREATE MODULAR ROUTE HANDLERS FOR DIFFERENT PATHS. THIS IS ESPECIALLY USEFUL FOR STRUCTURING LARGE APPLICATIONS.

JAVASCRIPT

CODE

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Home Page');
});

const router = express.Router();

router.get('/about', (req, res) => {
  res.send('About Page');
});

router.get('/contact', (req, res) => {
  res.send('Contact Page');
});

app.use('/pages', router);

app.listen(3000, () => {
  console.log('Server is running on port 3000');
```

});

3. ERROR HANDLING:

EXPRESS PROVIDES A WAY TO HANDLE ERRORS, INCLUDING ERRORS IN MIDDLEWARE AND ROUTES.

JAVASCRIPT

CODE

```
const express = require('express');
const app = express();

app.get('/error', (req, res, next) => {
  const err = new Error('This is a custom error');
  next(err);
});

app.use((err, req, res, next) => {
  console.error(err);
  res.status(500).send('Something went wrong');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

4. AUTHENTICATION AND AUTHORIZATION:

IMPLEMENTING USER AUTHENTICATION AND AUTHORIZATION IS AN ADVANCED TOPIC. LIBRARIES LIKE PASSPORT ARE COMMONLY USED.

5. RESTFUL API DESIGN:

EXPRESS IS COMMONLY USED FOR BUILDING RESTFUL APIs. DESIGNING YOUR ROUTES AND ENDPOINTS IS ESSENTIAL.

JAVASCRIPT

CODE

```
const express = require('express');
const app = express();

const books = [
  { id: 1, title: 'Book 1' },
  { id: 2, title: 'Book 2' },
];

// Get all books
app.get('/api/books', (req, res) => {
  res.json(books);
});

// Get a single book by ID
app.get('/api/books/:id', (req, res) => {
  const book = books.find((b) => b.id === parseInt(req.params.id));
  if (!book) return res.status(404).send('Book not found');
  res.json(book);
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

6. SECURITY BEST PRACTICES:

SECURE YOUR EXPRESS APPLICATION BY IMPLEMENTING BEST PRACTICES TO PREVENT COMMON VULNERABILITIES.

7. TEMPLATE ENGINES:

EXPRESS CAN WORK WITH TEMPLATE ENGINES LIKE EJS, PUG, OR HANDLEBARS TO RENDER DYNAMIC VIEWS.

JAVASCRIPT

CODE

```
const express = require('express');
const app = express();
app.set('view engine', 'ejs'); // EJS as the template engine

app.get('/home', (req, res) => {
  res.render('home', { title: 'My Page' });
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

8. WEBSOCKETS:

IMPLEMENT REAL-TIME FEATURES IN YOUR EXPRESS APPLICATION USING WEBSOCKETS WITH LIBRARIES LIKE SOCKET.IO.

~REVISION NOTES OF EXPRESS JS

- WHAT IS EXPRESS?

- A NODE FRAMEWORK, THAT WAS BUILT TO MAKE PROGRAMMERS WRITE LESS

- REPETITIVE CODE

- BUILT SPECIFICALLY FOR WEB DEVELOPERS

- CREATING OUR FIRST SERVER WITH EXPRESS

- NPM INIT

- NPM INSTALL EXPRESS

- IN THE SERVER.JS FILE

- //JSHINT ESVERSION:6

- PLACE THIS COMMENT ABOVE TO REMOVE WARNINGS

- CONST EXPRESS = REQUIRE('EXPRESS');

- FUNCTION THAT REPRESENTS THE EXPRESS MODULE, AND GETS Binded TO

- THE VARIABLE APP

- CONST APP = EXPRESS();

- LISTEN TO A SPECIFIC PORT

- APP.LISTEN(3000);

- HANDLING REQUESTS AND RESPONSES: THE GET REQUEST

- LOCALHOST:3000

- EQUIVALENT TO THE ROOT OF A HOME PAGE

- ROOT PAGE

- BROWSER SENDS A GET REQUEST TO THE SERVER

- APP.GET(<"LOCATION">, <CALL BACK FUNCTION, TELLS THE SERVER WHAT TO DO

- WHEN THAT REQUEST HAPPENS>)

- EXAMPLE

- APP.GET("/", FUNCTION(REQ, RES) {});

- UNDERSTANDING AND WORKING WITH ROUTES

- APP.GET(<ROUTE>, <RESPONDING CALLBACK>)

- SPECIFIES THE ROUTE THAT WE ARE GOING TO RESPOND TO
- WHEN A ROUTE IS ENCOUNTERED, THE GET() IS EXECUTED
 - <ROUTE>
- PATH
- EXAMPLE:
 - ROOT PATH
 - “/”
 - <RESPONDING CALLBACK>
- RESPONDS FOR THE SPECIFIED ROUTE
- EXAMPLE:
 - FUNCTION(REQ, RES) {
 - RES.SEND();
 - NODEMON.IO
 - WILL MONITOR FOR CHANGES IN YOUR JS CODE AND WILL AUTOMATICALLY CHANGE YOUR CODE
 - ACTIVATE NODEMON
- NODEMON <.JS FILE>
 - RESPONDING TO REQUESTS WITH HTML FILES
 - RES.SENDFILE()
- __DIRNAME
 - GIVES THE CURRENT FILE PATH
 - USAGE:
 - __DIRNAME + “<NAME OF HTML FILE>”

■ PROCESSING POST REQUESTS WITH BODY PARSER

- <FORM> HAS AN ACTION AND A METHOD
 - METHOD
 - POST, SO WE ARE SENDING DATA SOMEWHERE
 - ACTION
 - PATH/LOCATION TO INTERACT WITH (LIKE SENDING DATA VIA A POST)
 - APP.POST(<ROUTE>, <CALLBACK METHOD>)
 - SPECIFIES THAT WHEN THERE IS A POST REQUEST FOR THE ROUTE, THE CALLBACK METHOD WILL BE EXECUTED
 - BODY-PARSER
 - USED TO PARSE INCOMING POST DATA
 - TO INSTALL, CALL THE COMMAND “NPM INSTALL BODY-PARSER”
 - IN THE JS FILE, INCLUDE
 - CONST BODYPARSER = REQUIRE('BODY-PARSER');
 - APP.USE(BODYPARSER);
 - BODY-PARSER MODES
 - APP.USE(BODYPARSER.<MODE>)
 - BODYPARSER.TEXT()
 - PARSE ALL THE REQUESTS INTO TEXT/STRING
 - BODYPARSER.JSON()
 - BODYPARSER.URLENCODED()
 - USED WHEN PARSING DATA THAT IS SEND

VIA A HTML FORM

- BODYPARSER.URLENCODED({EXTENDED:
TRUE})

- ALLOWS US TO POST NESTED
OBJECTS

The Coding Wizard

13.APPLICATION PROGRAMMING INTERFACE

- API
 - INTERFACE LAYER TO INTERACT WITH AN EXTERNAL SYSTEM.
 - THE INTERACTIONS ARE VIA THE FORM OF GET AND POST REQUESTS
- USING THE REQUEST MODULE TO GET DATA FROM AN API
 - REQUEST - NPM MODULE
 - SETUP
 - CONST REQUEST = REQUIRE("REQUEST");
 - USAGE
 - REQUEST.(<URL TO REQUEST TO>, <CALLBACK>);
- <CALLBACK>

API KEY:

AN API KEY IS A CODE PASSED IN BY COMPUTER PROGRAMS CALLING AN API (APPLICATION PROGRAMMING INTERFACE) TO IDENTIFY THE CALLING PROGRAM. IT'S LIKE A PASSWORD, ALLOWING THE API PROVIDER TO TRACK AND CONTROL HOW THE API IS BEING USED. API KEYS HELP IN PREVENTING UNAUTHORIZED ACCESS AND ALLOW PROVIDERS TO MONITOR USAGE FOR BILLING AND SECURITY PURPOSES.

USING API KEYS:

1. OBTAINING AN API KEY:

To get an API key, you usually need to sign up for an account with the API provider. Once registered, you'll typically find your API key in your account settings or developer dashboard.

2. INCLUDING API KEY IN REQUESTS:

Most often, you include the API key in the headers of your API requests. This is usually done using the "Authorization" header.

JAVASCRIPT

```
const apiKey = 'your_api_key_here';
const headers = {
  'Authorization': `Bearer ${apiKey}`,
  'Content-Type': 'application/json'
};
```

The format `Bearer <API_KEY>` is common when using API keys for authentication.

3. MAKING API CALLS:

After setting up the API key in your headers, you can make API calls as usual. Here's an example using JavaScript's `fetch` function:

```
const url = 'https://api.example.com/
data';
fetch(url, {
  method: 'GET',
  headers: headers
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error =>
  console.error('Error:', error));
```

IMPORTANT NOTES:

- KEEP YOUR API KEY SECRET:

Treat your API key as sensitive information. Never expose it in client-side code that's accessible to users. If someone gains access to your API key, they might misuse it.

- SECURITY MEASURES:

API providers might offer additional security measures like IP whitelisting, allowing requests only from specific IP addresses.

- RATE LIMITING:

Many APIs implement rate limiting to prevent abuse. This restricts the number of requests you can make within a specific timeframe. Make sure to check the API documentation for rate limits.

- ENVIRONMENT VARIABLES:

It's a good practice to store your API keys in environment variables, especially if you're sharing your code with others or collaborating on a project. This keeps sensitive information separate from your codebase.

EXAMPLE SCENARIO:

Let's say you're using the OpenWeatherMap API to retrieve weather data. You'd sign up for an account, get an API key, and use it to make requests to the API endpoints.

JAVASCRIPT

```
const apiKey =
  'your_openweathermap_api_key';
const headers = {
  'Authorization': `Bearer ${apiKey}`,
  'Content-Type': 'application/json'
};

const city = 'New York';
const url = `https://
api.openweathermap.org/data/2.5/
weather?q=${city}&appid=${apiKey}`;

fetch(url, {
  method: 'GET',
  headers: headers
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error =>
  console.error('Error:', error));
```

REMEMBER TO REPLACE

``YOUR_OPENWEATHERMAP_API_KEY`` WITH YOUR ACTUAL API KEY.

1. SET UP AN API KEY:

To use an API, you typically need to sign up for the service and obtain an API key. For example, when using the Mailchimp API, you'd sign up for a Mailchimp account and get an API key from your account settings.

2. INCLUDE THE API KEY IN REQUESTS:

When making requests to the API, you usually include your API key in the request headers. This allows the API server to identify your application and grant or restrict access based on your permissions.

JAVASCRIPT

```
javascript
const apiKey = 'your_api_key_here';
const headers = {
  'Authorization': `Bearer ${apiKey}`,
  'Content-Type': 'application/json'
};
```

3. MAKING API CALLS:

Once you have the API key set up, you can use it to make requests to the API endpoints. For example, if you're using JavaScript and the `fetch` function:

JAVASCRIPT

```
javascript
const url = 'https://api.example.com/some/endpoint';
fetch(url, {
  method: 'GET', // or 'POST', 'PUT', etc.
  headers: headers
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error:', error));
```

Remember to replace `your_api_key_here` with your actual API key.

NOW, LET'S PROCEED TO THE NEXT TOPIC.

UNDERSTANDING THE JSON FORMAT AND WORKING WITH JSON:

JSON (JavaScript Object Notation) is a lightweight data interchange format that's easy for both humans and machines to read and write. It's commonly used to send and receive data between a server and a web application.

JSON FORMAT EXAMPLE:

JSON

```
json
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "address": {
    "street": "123 Main St",
    "city": "Anytown"
  },
  "interests": ["reading", "music", "travel"]
}
```

JAVASCRIPT OBJECT TO JSON STRING:

You can convert a JavaScript object to a JSON string using `JSON.stringify()`:

JAVASCRIPT

```
javascript
const person = {
  name: "John Doe",
  age: 30,
  isStudent: false
};

const jsonStr = JSON.stringify(person);
```

JSON STRING TO JAVASCRIPT OBJECT:

To convert a JSON string back to a JavaScript object, use `JSON.parse()`:

JAVASCRIPT

```
javascript
const jsonStr = '{"name": "John Doe", "age": 30, "isStudent": false}';
const person = JSON.parse(jsonStr);
```

THIS CONVERTS THE JSON STRING INTO A JAVASCRIPT OBJECT.

Remember, JSON represents data, so it's commonly used when sending or receiving data in APIs. For instance, when you interact with an API, you might receive JSON responses that you need to parse and work with in your application.

~REVISION NOTES OF API

- API

- INTERFACE LAYER TO INTERACT WITH AN EXTERNAL SYSTEM.
- THE INTERACTIONS ARE VIA THE FORM OF GET AND POST REQUESTS

- USING THE REQUEST MODULE TO GET DATA FROM AN API

- REQUEST - NPM MODULE
- SETUP
 - CONST REQUEST = REQUIRE("REQUEST");
- USAGE
 - REQUEST.(<URL TO REQUEST TO>, <CALLBACK>);

- <CALLBACK>

- FUNCTION(ERROR, RESPONSE, BODY);

- UNDERSTANDING THE JSON FORMAT AND WORKING WITH JSON

- JSON
 - JAVASCRIPT OBJECT NOTATION
 - USED TO COMMUNICATE DATA BACK AND FORTH IN THE WEB
- XML
 - EXTENSIVE MARKUP LANGUAGE
 - ALTERNATIVE TO JSON, CAME BEFORE JSON.
 - LESS FREQUENTLY USED NOW
- JAVASCRIPT OBJECT -> STRING
 - JSON.STRINGIFY(<JAVASCRIPT OBJECT>);
- STRING -> JAVASCRIPT OBJECT

- JSONPARSE(<JSON STRING EQUIVELANT>);
- SENDING MORE THAN ONE THING
 - RES.WRITE(<HTML TO RESPOND WITH>);
 - RES.SEND();
- NEWSLETTER WALKTHROUGH
 - INCORPORATED AND MODIFIED AN EXISTING BOOTSTRAP SIGN IN EXAMPLE
 - IN THE HTML, SOME FILES ARE DIRECTLY OBTAINED FROM THE INTERNET VIA THE USE OF THE WEB URLs
 - HOWEVER, SOME ARE STATIC FILES SUCH AS CSS AND IMAGES FILES.
 - IN ORDER TO USE THEM, WE MUST NOTIFY EXPRESS THAT THEY ARE STATIC
 - TO DO THIS, WE CAN CREATE A FOLDER WHERE ALL THE STATIC FILES WILL BE CONTAINED,
 - AND THEN CALL THE EXPRESS.STATIC()
 - IN THE FORM, BE SURE TO SPECIFY ACTION AND METHOD
 - SO THE PICTURE ABOVE, THE FORM WILL POST TO THE LOCATION IN ACTION
 - POSTING DATA TO MAILCHIMP SERVERS
 - SET UP MAINCHIMP ACCOUNT & OBTAIN API KEY
 - SET UP THE REQUEST URL AND HEADERS
 - AUTHENTICATION

- LOOK THROUGH THE API DOCUMENTATION ON AUTHENTICATION

```
app.post("/", function(req, res) {  
    var firstName = req.body.fName;  
    var lastName = req.body.lName;  
    var email = req.body.email;  
  
    var data = {  
        'members': [  
            {  
                email_address: email,  
                status: 'subscribed',  
                merge_fields: {  
                    FNAME: firstName,  
                    LNAME: lastName  
                }  
            }  
        ],  
    };  
  
    var jsonData = JSON.stringify(data);  
  
    console.log(firstName, lastName, email);  
  
    var options = {  
        url: 'https://us3.api.mailchimp.com/3.0/lists/c88fb2ef24',  
        method: 'POST',  
        headers: {  
            'Authorization': "anand1 8a8caf31357e4496e62d3e3690b8797b-us3"  
        },  
        body: jsonData  
    };  
  
    request(options, function(error, response, body) {  
        if (error) {  
            console.log(error);  
        } else {  
            console.log(response.statusCode);  
        }  
    });  
});
```

- HEROKU ALLOWS USERS TO DEPLOY THEIR WEBSITES ON HEROKU'S SERVERS,

SO THAT ANYONE CAN ACCESS THE WEBSITE.

- TO USE HEROKU:

- 1. MODIFY THE PRIOR APP.LISTEN(3000, <CALLBACK>); TO APP.LISTEN(PROCESS.ENV.PORT || 3000, <CALLBACK>);
 - PROCESS.ENV.PORT - HEROKU WILL HANDLE THE SETTING OF THE PORT
 - || 3000 - THIS WILL BE USED IN THE CASE THAT HEROKU IS NOT

BEING USED

- 2. DEFINE A PROCFILE
 - CREATE A FILE NAMED PROCFILE THAT IS IN THE SAME FOLDER AS THE MAIN ROOT OF THE WEBSITE
 - INSIDE, WRITE THE SAME COMMAND AS YOU WOULD WRITE IN THE TERMINAL TO EXECUTE YOUR WEBSITE
- WEB: NODE APP.JS
 - 3. INITIALIZE AND COMMIT THE WEB PROJECT USING GIT
 - IN THE WEB PROJECT ROOT DIRECTORY, EXECUTE GIT INIT
 - ADD THE FILES, GIT ADD .
 - COMMIT THE FILES, GIT COMMIT -M "COMMIT MESSAGE"
 - 4. CREATE A HEROKU REFERENCE TO YOUR WEB PROJECT, GIT
 - EXECUTE, HEROKU CREATE
 - BY DEFAULT, HEROKU CREATES A RANDOM NAME TO YOUR PROJECT
 - BUT YOU CAN PASS A PARAMETER TO SET THE NAME
 - EXECUTE, GIT PUSH HEROKU MASTER
 - WEB PROJECT IS NOW DEPLOYED ON HEROKU
 - 5. RUNNING THE WEB SITE ON HEROKU
 - 1. CAN RUN HEROKU OPEN ON THE ROOT DIRECTORY OF THE WEB

14. EJS

EJS (EMBEDDED JAVASCRIPT):

EJS IS A TEMPLATING ENGINE THAT ENABLES YOU TO GENERATE HTML WITH EMBEDDED JAVASCRIPT. IT'S OFTEN USED WITH NODE.JS AND EXPRESS TO CREATE DYNAMIC WEB PAGES BY EMBEDDING SERVER-SIDE CODE WITHIN HTML TEMPLATES.

SETTING UP EJS:

1. INSTALL EJS:

INSTALL EJS USING NPM (NODE PACKAGE MANAGER) IN YOUR NODE.JS PROJECT:

BASH

NPM INSTALL EJS

2. CONFIGURE EXPRESS:

IN YOUR EXPRESS APPLICATION, CONFIGURE EJS AS THE VIEW ENGINE:

JAVASCRIPT

```
javascript
const express = require('express');
const app = express();

app.set('view engine', 'ejs');
```

BASIC EJS TAGS:

1. OUTPUTTING VARIABLES:

USE `<%= VARIABLENAME %>` TO OUTPUT THE VALUE OF A VARIABLE IN THE HTML.

```
ejs
<h1>Welcome to <%= pageTitle %></h1>
```

2. CONDITIONAL STATEMENTS:

EJS SUPPORTS JAVASCRIPT'S CONDITIONAL STATEMENTS.

```
ejs
<% if (isLoggedIn) { %>
  <p>Welcome, <%= username %>!</p>
<% } else { %>
  <p>Please log in.</p>
<% } %>
```

3. LOOPING WITH FOREACH:

USE FOREACH LOOP TO ITERATE THROUGH AN ARRAY AND GENERATE HTML FOR EACH ELEMENT.

```
ejs
<ul>
  <% items.forEach(function(item) { %>
    <li><%= item %></li>
  <% }); %>
</ul>
```

LAYOUTS AND PARTIALS:

1. LAYOUTS:

CREATE A LAYOUT TEMPLATE (E.G., `LAYOUT.EJS`) WITH COMMON ELEMENTS LIKE HEADER AND FOOTER.

```
ejs
<!DOCTYPE html>
<html>
<head>
  <title><%= pageTitle %></title>
</head>
<body>
  <%- body %> <!-- Rendered content goes here -->
</body>
</html>
```

2. PARTIALS:

CREATE REUSABLE COMPONENTS (PARTIALS) TO INCLUDE IN YOUR TEMPLATES.

```
ejs
<!-- header.ejs -->
<header>
  <h1><%= title %></h1>
</header>
```

INCLUDE THE PARTIAL IN YOUR TEMPLATE:

```
ejs
<%- include('partials/header', { title: 'My Website' }) %>
```

PASSING DATA TO TEMPLATES:

PASS DATA FROM YOUR ROUTE TO THE EJS TEMPLATE USING `RES.RENDER():`

JAVASCRIPT

```
javascript
app.get('/profile', (req, res) => {
  const user = { username: 'john_doe', age: 30 };
  res.render('profile', { user });
});
```

INCLUDING CSS AND JS:

LINK CSS AND JS FILES LIKE YOU WOULD IN REGULAR HTML:

EJS

```
ejs
<link rel="stylesheet" href="/styles.css">
<script src="/app.js"></script>
```

ESCAPING HTML:

TO PREVENT CROSS-SITE SCRIPTING (XSS) ATTACKS, USE `<%-` TO RENDER UNESCAPED HTML:

EJS

```
<P>THIS IS <%- UNESCAPEDHTML %></P>
```

COMMENTS:

COMMENTS WITHIN EJS TEMPLATES ARE SIMILAR TO JAVASCRIPT:

EJS

```
<% // THIS IS A COMMENT %>
```

EXAMPLE USAGE:

LET'S SAY YOU HAVE AN EXPRESS ROUTE THAT RENDERS A TEMPLATE:

JAVASCRIPT

```
javascript
app.get('/greeting', (req, res) => {
  const name = 'Alice';
  res.render('greeting', { name });
});
```

AND IN YOUR `GREETING.EJS` TEMPLATE:

EJS

```
ejs
<!DOCTYPE html>
<html>
<head>
  <title>Greeting Page</title>
</head>
<body>
  <h1>Hello, <%= name %>!</h1>
</body>
</html>
```

~REVISION NOTES OF EJS

○ WHAT IS EJS?

- TEMPLATING LANGUAGE THAT LETS YOU GENERATE HTML MARKUP WITH PLAIN JAVASCRIPT
- USEFUL WHEN YOU WANT TO USE A SINGLE HTML PAGE, BUT WITH DIFFERENT CONTENTS/DATA BASED ON CONDITIONS

○ CREATING YOUR FIRST EJS TEMPLATE

- NPM INSTALL EJS
- IN .JS,
 - APP.SET('VIEW ENGINE', 'EJS');
- CREATE A NEW FOLDER CALLED 'VIEWS'
 - ADD A FILE WITH .EJS

○ .EJS FILE FOLLOWS HTML FORMAT, BUT HAS EJS VARIABLE PLACEHOLDERS

- PLACEHOLDER MARKER ●
- TO DISPLAY A EJS WEBPAGE,
 - RES.RENDER(>);

○ RUNNING CODE INSIDE THE EJS TEMPLATE

- WE CAN ADD JAVASCRIPT CODE IN THE EJS FILE.
- EACH LINE OF THE JAVASCRIPT NEEDS TO HAVE AN EJS SCRIPTLET OF
 - USE SCRIPTLET ONLY FOR CONTROL-FLOW, BUT ONLY IN SELECT SITUATIONS.

- MOST LOGIC NEEDS TO RESIDE IN THE SERVER CODE
-
- PASSING DATA FROM YOUR WEBPAGE TO YOUR SERVER
 - 1. THIS WILL BE DONE VIA A POST REQUEST FROM THE WEB PAGE.
 - 2. SERVER, LET'S SAY APP.JS, WILL THEN HANDLE THE POST REQUEST.
 - 3. IT CAN USE BODY-PARSER, TO THEN OBTAIN THE DATA BEING SEND, AND WORK WITH THE DATA
 - RES.REDIRECT()
-
- THE CONCEPT OF SCOPE IN THE CONTEXT OF JAVASCRIPT
 - VARIABLES INSIDE OF A FUNCTION, HAVE LOCAL SCOPE TO THAT FUNCTION
 - VARIABLES OUTSIDE OF A FUNCTION, HAVE GLOBAL SCOPE
 - IF/ELSE OR FOR/WHILE
 - VARIABLE CREATED WILL HAVE GLOBAL SCOPE
 - LET WILL HAVE LOCAL SCOPE
 - CONST WILL HAVE LOCAL SCOPE
 - USE LET OR CONST WHENEVER YOU CAN, OVER VAR
-
- ADDING PRE-MADE CSS STYLESHEETS TO YOUR WEBSITE
 - ALL FILES THAT WILL BE USED MUST BE SERVED UP IN EXPRESS APP.JS, THESE ARE STATIC FILES
 - STATIC FILES ARE FILES THAT CLIENTS DOWNLOAD AS THEY ARE FROM THE SERVER.
-
- UNDERSTANDING TEMPLATING VS LAYOUTS
 - TEMPLATING

- USED MAINLY WHEN YOU WANT TO REUSE THE STRUCTURE AND FORMAT OF A WEB PAGE, BUT CHANGING THE DATA WITHIN THEM

- LAYOUTS

- WHEN YOU WANT TO REUSE CERTAIN PARTS OF A WEBPAGE, LIKE THE HEADERS, FOOTERS, AND STYLING, IN MANY DIFFERENT WEBPAGES

- BUT OVERALL, THE BODY STRUCTURE DIFFERS

- USE <%- INCLUDE(<ejs NAME WITHOUT .EJS>) -%> FOR REUSING PARTS OF THE WEBPAGE

- REUSED COMPONENT PAGES ARE CALLED PARTIALS

- UNDERSTANDING NODE MODULE EXPORTS: HOW TO PASS FUNCTIONS AND DATA BETWEEN

- FILES

- USING FUNCTIONS BETWEEN FILES:

- NODE.JS CONTAINS AN OBJECT CALLED MODULE

- IF YOU WANT TO CREATE FUNCTIONS IN A CLASS THAT YOU WANT IT TO BE REUSABLE IN OTHER FILES:

- YOU CAN SET MODULE.EXPORTS.<FUNCTION REFERENCE NAME>

= <NAME OF THE FUNCTION THAT YOU WANT TO REUSE>

- MODULE.EXPORTS CAN ALSO BE SHORTENED DOWN, AND EXPORTS CAN BE USED

- TO USE THE FUNCTION:

- CREATE A REFERENCE VARIABLE, AND LINK IT TO THE FILE WHERE THE REUSABLE FUNCTIONS RESIDE

- BE SURE TO USE __DIRNAME + IF IT'S CUSTOM, BECAUSE

WE DIDN'T INSTALL USING NPM

- THEN INVOKE THE FUNCTION USING <REFERENCE VARIABLE>.<NAME OF REUSABLE FUNCTION>
 - BLOG WEBSITE CHALLENGE
 - USUALLY, THE NPM PACKAGES THAT ARE BEING USED ARE GITIGNORED, AND WHEN A NEW PERSON WORKS WITH THE PROJECT THEY WOULD THEN INITIALLY GO AHEAD AND RUN NPM INSTALL
 - EXPRESS PARAMETER ROUTING
-

15. DATABASE

DATABASES:

Databases are systems for storing and managing data. They provide methods to organize, retrieve, update, and manipulate data efficiently.

SQL VS NOSQL:

SQL (STRUCTURED QUERY LANGUAGE):

1. STRUCTURED QUERY LANGUAGE:

SQL is a language used for managing and manipulating relational databases. It's used to perform operations like selecting, updating, inserting, and deleting data.

2. MOST POPULAR:

SQL databases are widely used and have a long history. Examples include MySQL, PostgreSQL, Oracle, and SQL Server.

EXAMPLE (SQL):

Imagine a table "Users" with columns "ID," "Name," and "Age."

ID	NAME	AGE
ID	Name	Age
1	John	25
2	Alice	30

NOSQL (NOT ONLY STRUCTURED QUERY LANGUAGE):

1. NOT ONLY STRUCTURED QUERY LANGUAGE:

NoSQL databases are designed to handle unstructured or semi-structured data. They include various types like document stores, key-value stores, column-family stores, and graph databases.

2. MOST POPULAR:

NoSQL databases gained popularity due to their flexibility and scalability. Examples include MongoDB (document store) and Redis (key-value store).

EXAMPLE (NOSQL):

In a document-based NoSQL database like MongoDB, data could be structured as JSON-like documents.

```
json
{
  "_id": "1",
  "name": "John",
  "age": 25
}
```

DIFFERENCES:

1. STRUCTURE:

SQL:

- Uses tables with fixed columns.
- Cells can be empty (NULL) if no data is present.
- Ensures a consistent structure for all rows.

NOSQL:

- Uses flexible JSON-like structures.
- Each document can have different fields.

2. RELATIONSHIPS:

SQL:

- Relational databases establish relationships between tables.
- Suitable for complex many-to-many relationships.
- Uses joins to retrieve data across multiple tables.

EXAMPLE (SQL RELATIONSHIPS):

Imagine tables "Users" and "Orders" linked through "user_id."

USERS:

ID	Name
1	John
2	Alice

Orders:

ID	user_id	Product
1	1	Item1
2	2	Item2

NOSQL:

- NoSQL databases typically store related data within a single document.
- Good for simpler 1-to-many relationships.

EXAMPLE (NOSQL RELATIONSHIPS):

In MongoDB, you can embed orders within a user document.

JSON

```
json
{
  "_id": "1",
  "name": "John",
  "orders": [
    { "product": "Item1" },
    { "product": "Item2" }
  ]
}
```

3. SCALABILITY:

SQL:

- Traditional SQL databases might become slower as data grows.

NOSQL:

- NoSQL databases are designed to handle massive amounts of data and can scale horizontally.

REVISION NOTES OF DATABASE

- SQL VS NOSQL
 - SQL
 - STRUCTURED QUERY LANGUAGE
 - MOST POPULAR
 - MYSQL
 - POSTGRESQL
 - NOSQL
 - NOT ONLY STRUCTURED QUERY LANGUAGE
 - MOST POPULAR
 - MONGODB
 - REDIS
 - DIFFERENCES
 - 1. STRUCTURE
 - SQL
 - TABLE FORMAT, KIND OF LIKE A EXCEL SHEET
 - MISSING CELLS, WILL CONTAIN NULL
 - CONSISTENT STRUCTURE
 - NOSQL
 - JSON OBJECT STRUCTURE
 - FLEXIBLE STRUCTURE
 - 2. RELATIONSHIPS

- SQL

- RELATIONAL
 - GOOD FOR MANY-TO-MANY RELATIONSHIPS
 - CAN HAVE MULTIPLE TABLES, AND WE CAN LINK A RELATIONSHIP BETWEEN THEM
 - WILL LEAD TO LESS REPETITIVE DATA

- NOSQL

- NON-RELATIONAL
 - CAN LEAD TO REPETITIVE DATA
 - GOOD FOR 1-TO-MANY TYPE RELATIONSHIPS
 - 3. SCALABILITY

- SQL

- BECOMES SLOWER WITH MORE ROWS OF DATA

- NOSQL

- BETTER FOR SCALABILITY

16.SQL

1. BASICS OF SQL:

- SQL IS USED TO COMMUNICATE WITH RELATIONAL DATABASES TO PERFORM TASKS LIKE QUERYING, INSERTING, UPDATING, AND DELETING DATA.

2. DATA DEFINITION LANGUAGE (DDL):

- DDL IS USED TO DEFINE AND MANAGE DATABASE STRUCTURES.
- EXAMPLES:

- Creating a table:

```
sql  
CREATE TABLE Employees (  
    ID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Age INT  
)
```

- Altering a table:

```
sql  
ALTER TABLE Employees  
ADD Salary DECIMAL(10, 2);
```

- DROPPING A TABLE:

SQL

```
DROP TABLE EMPLOYEES;
```

3. DATA MANIPULATION LANGUAGE (DML):

- DML IS USED TO MANIPULATE DATA WITHIN THE TABLES.

- EXAMPLES:

- INSERTING DATA:

SQL

```
INSERT INTO EMPLOYEES (ID, NAME, AGE) VALUES (1, 'JOHN DOE', 30);
```

- Inserting data:

sql

```
INSERT INTO Employees (ID, Name, Age) VALUES (1, 'John Doe', 30);
```

- Updating data:

sql

```
UPDATE Employees  
SET Age = 31  
WHERE ID = 1;
```

- Deleting data:

sql

```
DELETE FROM Employees  
WHERE ID = 1;
```

4. DATA QUERY LANGUAGE (DQL):

- DQL IS USED TO RETRIEVE DATA FROM TABLES.

- EXAMPLES:

- SELECTING ALL COLUMNS:

sql

```
SELECT * FROM Employees;
```

- Selecting specific columns:

sql

```
SELECT Name, Age FROM Employees;
```

- Using WHERE clause:

sql

```
SELECT * FROM Employees  
WHERE Age > 25;
```

5. DATA CONTROL LANGUAGE (DCL):

- DCL IS USED TO MANAGE PERMISSIONS AND ACCESS CONTROL.

- EXAMPLE:

- GRANTING PRIVILEGES:

SQL

```
GRANT SELECT, INSERT ON EMPLOYEES TO USER123;
```

6. AGGREGATION FUNCTIONS:

- FUNCTIONS LIKE COUNT, SUM, AVG, MAX, MIN TO SUMMARIZE DATA.

- EXAMPLE:

SQL

```
SELECT COUNT(*) FROM EMPLOYEES;
```

7. JOINS:

- COMBINING DATA FROM MULTIPLE TABLES.

- EXAMPLE:

SQL

```
SELECT Orders.OrderID, Customers.CustomerName  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

8. SUBQUERIES:

- USING ONE QUERY RESULT IN ANOTHER QUERY.
- EXAMPLE:

SQL

```
SELECT CustomerName  
FROM Customers  
WHERE CustomerID IN (SELECT CustomerID FROM Orders);
```

9. INDEXES:

- IMPROVE QUERY PERFORMANCE BY CREATING INDEXES ON COLUMNS.
- EXAMPLE:

SQL

```
CREATE INDEX idx_name ON Employees (Name);
```

10. TRANSACTIONS:

- ENSURING DATA INTEGRITY USING TRANSACTIONS.
- EXAMPLE:

SQL

```
BEGIN TRANSACTION;
```

```
-- SQL STATEMENTS
```

```
COMMIT;
```

REVISION NOTES OF SQL

○ CREATE TABLE AND INSERT DATA

■ CREATE TABLE

- CREATE TABLE <TABLE NAME> (<COLUMN NAME> <DATA TYPE>, ...

PRIMARY KEY (<COLUMN NAME TO BE THE PRIMARY KEY>));

■ DATA TYPES

- INT, STRING, MONEY, CHAR, ...

■ INSERT DATA

- INSERT INTO <TABLE NAME> VALUES (VALUE1, VALUE2, VALUE3, ...);

○ READ, SELECT, AND WHERE

■ SELECTING SPECIFIC COLUMNS

- SELECT <COLUMN(S)> FROM <TABLE>;

■ SELECTING SPECIFIC ROWS

- SELECT <COLUMN(S)> FROM <TABLE> WHERE <CONDITION>;

○ UPDATING SINGLE VALUES AND ADDING COLUMNS IN SQL

- UPDATE <TABLE> SET <COLUMN> = <NEW CELL VALUE> WHERE <COLUMN>
= <CONDITION>;

■ ADDING A NEW COLUMN

- ALTER TABLE (ADD, DELETE, OR MODIFY COLUMNS IN A TABLE)

○ ALTER TABLE <TABLE NAME> ADD <NEW COLUMN NAME>

<DATA TYPE>;

- DELETE
 - DELETE FROM <TABLE> WHERE <CONDITION>;
 - RELATIONSHIPS, FOREIGN KEYS, AND INNER JOINS
 - FOREIGN KEY
 - LINKS BETWEEN TABLE
 - INNER JOIN
-

17. MONGODB

- Install MongoDB on Mac
 - Follow directions on Mongodb
- MongoDB CRUD Operations
 - Create
 - Create a database ○ terminal: use
 - Create a collection / Insert new items into a collection
 - db..insertOne()
 - db..insertMany()
 - If the collection doesn't exist, then a new one will be created
 - Read / Query
 - terminal: show collections ○ Shows all the collections
 - db..find()
 - Displays all the data in the collection
 - db..find(,)
 - Displays all the data in the collection that meets the query criteria
 - Example of Query:
 - {name : "Pencil"}
 - {price: {\$gt: 1}} ○ Example of Projection:
 - {name : 1, address: 1}
 - 1 means true
 - 0 means false
 - We can dictate which fields we want back from the query

- So in this case, where the data has a name and an address
- Update

Db.<collection>.updateOne(<Query>, <Update To>)

- ➔ <Query>
- ➔ The date to update
- ➔ <Update To>
- ➔ {\$set: {<field>: <new value>}}

- Delete

- db.<collection>.deleteOne(Query Criteria)
- db.<collection>.deleteMany()

- Relationships in MongoDB

- One to Many Relationships are great for NoSQL - MongoDB

- Example:

- Product
 - Review 1
 - Review 2
 - Review 3 ...

- Mongo shell

- terminal: mongod

- Starts up the mongo server
- When you see “waiting for connections on port 27017” then the server has started

- terminal: mongo

- Starts up the mongo shell

- Troubleshoot:

- sudo killall -15 mongod ○ If unable to start up the server using mongod ○ MongoDB with Node.JS
 - 2 ways to incorporate MongoDB app with Node.js
- 1. Native MongoDB Driver ○ Install a MongoDB driver from the documentation
 - One that corresponds to the programming language
 - In this case, Node.js MongoDB driver
 - npm install mongodb
- 2. mongoose
 - ODM
 - Object Data Modeling library ○ Most popular package to work with MongoDB and Node.JS

18. MONGOOSE

1. INSTALLATION:

- INSTALL MONGOOSE USING NPM OR YARN.
- EXAMPLE:

BASH

```
npm install mongoose
```

2. CONNECTING TO MONGODB:

- ESTABLISH A CONNECTION TO YOUR MONGODB DATABASE USING MONGOOSE.
- EXAMPLE:

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/
mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});
```

3. DEFINING MODELS:

- DEFINE DATA STRUCTURE USING MONGOOSE SCHEMA AND CREATE A MODEL.

- EXAMPLE:

JAVASCRIPT

```
const mongoose = require('mongoose');

const userSchema = new
mongoose.Schema({
  username: String,
  email: String,
  age: Number
});

const User = mongoose.model('User',
userSchema);
```

4. CRUD OPERATIONS:

- PERFORM CREATE, READ, UPDATE, AND DELETE OPERATIONS USING MONGOOSE.

- EXAMPLES:

- Create:

```
const newUser = new User({  
    username: 'john_doe',  
    email: 'john@example.com',  
    age: 25  
});  
newUser.save();
```

...

.

- Read:

```
// Find all users  
User.find({}, (err, users) => {  
    if (err) throw err;  
    console.log(users);  
});  
  
// Find a specific user  
User.findOne({ username: 'john_doe' },  
(err, user) => {  
    if (err) throw err;  
    console.log(user);  
});
```

.

- Update:

```
User.findOneAndUpdate(  
    { username: 'john_doe' },  
    { age: 26 },  
    { new: true },  
    (err, user) => {  
        if (err) throw err;  
        console.log(user);  
    }  
);
```

- DELETE:

JAVASCRIPT

```
User.findOneAndDelete({ username:  
  'john_doe' }, (err) => {  
  if (err) throw err;  
  console.log('User deleted');  
});
```

5. VALIDATION:

- DEFINE VALIDATION RULES FOR SCHEMA FIELDS.

- EXAMPLE:

JAVASCRIPT

```
const userSchema = new
mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true
  },
  email: {
    type: String,
    required: true,
    unique: true,
    validate: {
      validator: (value) => /\S+@\S+\.
\S+/.test(value),
      message: 'Invalid email format'
    }
  },
  age: {
    type: Number,
    min: 18
  }
});
```

6. MIDDLEWARE:

- EXECUTE FUNCTIONS BEFORE OR AFTER SPECIFIC EVENTS LIKE SAVING OR REMOVING DOCUMENTS.

- EXAMPLE:

JAVASCRIPT

```
userSchema.pre('save', function(next) {
  console.log('Saving user... ');
  next();
});

userSchema.post('remove',
function(doc) {
  console.log(` ${doc.username} has
been removed.`);
});
```

7. POPULATION:

- REPLACE REFERENCES WITH ACTUAL DATA FROM OTHER COLLECTIONS.
- EXAMPLE:

JAVASCRIPT

```
const commentSchema = new
mongoose.Schema({
  text: String,
  author: {
    type:
      mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }
});
```

8. QUERY HELPERS:*

- DEFINE REUSABLE FUNCTIONS FOR QUERIES.

- EXAMPLE:

JAVASCRIPT

```
userSchema.query.byUsername =  
  function(username) {  
    return this.where({ username });  
  };  
  
User.find().byUsername('john_doe').exe  
c((err, users) => {  
  if (err) throw err;  
  console.log(users);  
});
```

9. TRANSACTIONS:

- PERFORM ATOMIC TRANSACTIONS USING MONGOOSE.

- EXAMPLE:

JAVASCRIPT

```
const session = await
mongoose.startSession();
session.startTransaction();

try {
    // Perform operations within the
    transaction
    await newUser.save({ session });
    await session.commitTransaction();
} catch (error) {
    await session.abortTransaction();
    throw error;
} finally {
    session.endSession();
}
```

REVISION NOTES OF MONGOOSE

- TERMINOLOGY

- COLLECTION

- GROUPING OF MONGODB DOCUMENTS
 - EQUIVALENT TO A TABLE IN SQL

- DOCUMENT

- A SINGLE { <KEY : VALUE>, ... }

- WHICH CAN CONSIST OF 1 OR MANY KEY : VALUE PAIRS

- INTRO TO MONGOOSE

- ALLOW NODE.JS APP TO TALK WITH MONGODB

- ALLOWS THE USER TO DEFINE THE SCHEMA FOR THE DOCUMENTS IN A PARTICULAR COLLECTION
- NPM INSTALL MONGOOSE
- CREATE REFERENCE TO MONGOOSE
 - CONST MONGOOSE = REQUIRE('MONGOOSE');
- CREATE NEW DATABASE, OR CONNECT TO AN EXISTING DATABASE
- CLOSING CONNECTION
 - MONGOOSE.CONNECTION.CLOSE();
- CREATE A MONGOOSE SCHEME
- USE THE SCHEME TO CREATE A MONGOOSE MODEL
- ADDING A NEW ITEM TO THE COLLECTION
 - CREATE A NEW OBJECT BASED ON THE MODEL
 - THEN CALL .SAVE() ON THAT NEW OBJECT
- SAVING IN BULK
 - <MODEL>.INSERTMANY(<ARRAY OF OBJECTS>, <CALLBACK>)

19. DEPLOYING YOUR WEB APPLICATION

- HOW TO DEPLOY APPS WITH A DATABASE

- WE CAN'T SIMPLY PLACE THE APP W/DATABASE ON HEROKU, AS THE APP AND

- DATABASE WILL BE SEPARATED.

- WE NEED TO PLACE BOTH THE APP AND THE DATABASE ON SERVERS

- MONGODB ATLAS

- SERVER THAT HOSTS THE DATABASE

- HOW TO SETUP MONGODB ATLAS

- 1. SETUP AN ACCOUNT WITH ATLAS

- 2. CREATE A CLUSTER

- 3. SETUP SECURITY ACCESS

- 4. CLUSTER > CONNECT > CONNECT TO AN APPLICATION > USE THE URL OBTAINED

TO CONNECT WITH NODE.JS APPLICATION

- DEPLOYING APP WITH A DATABASE TO HEROKU

20. BUILD YOUR OWN RESTFUL API FROM SCRATCH

- WHAT IS REST?

- REPRESENTATIONAL STATE TRANSFER
- CLIENT - SERVER ARCHITECTURE
 - CLIENT (ANALOGY OF A CUSTOMER) MAKES A REQUEST TO A SERVER
 - SERVER (ANALOGY OF A WAITER) RESPONDS TO THE REQUEST
- SENT THROUGH (ANALOGY OF A LANGUAGE):
 - HTTP REQUEST
 - FTP REQUEST
 - HTTPS REQUEST

- SECURED

- API (ANALOGY OF A MENU)
 - SET OF SERVICES THAT THE CLIENT CAN INTERACT WITH THE SERVER
- REST IS A ARCHITECTURAL STYLE OF DESIGNING API'S
- RULES TO BE RESTFUL
 - HTTP REQUEST VERBS

- GET

- READ

- PUT

- FULL UPDATE

- PATCH
 - PARTIAL UPDATE
- POST
 - CREATE
- DELETE
 - DELETE
 - USING SPECIFIC PATTERN OF ROUTES/ENDPOINT URLs
- /{ROUTES}
- CREATING A DATABASE WITH ROBO 3T
 - ROBO 3T
 - GUI FOR MONGODB
 - BY DEFAULT ROBO 3T WILL CONNECT TO LOCALHOST:27017
 - FIRST, WE MUST STARTUP THE MONGODB SERVER, BY RUNNING “MONGOD”
- IN THE TERMINAL
- SET UP SERVER CHALLENGE
- GET ALL ARTICLES
- POST AN ARTICLE
 - INSTALL POSTMAN
 - POSTMAN ALLOWS US TO TEST OUT OUR APIs WITHOUT CREATING THE
- ADDITIONAL COMPONENTS NEEDED
- CHAINED ROUTE HANDLERS USING EXPRESS
 - APP.ROUTE()

- GET
 - POST
 - DELETE
- USING EXPRESS ROUTE CAN REDUCE REDUNDENCY
- GET A SPECIFIC ARTICLE
 - PUT A SPECIFIC ARTICLE
 - PATCH A SPECIFIC ARTICLE
 - DELETE A SPECIFIC ARTICLE

21. AUTHENTICATION & SECURITY

○ LEVEL 1 - REGISTER USERS WITH USERNAME AND PASSWORD

■ SETUP MONGOOSE AND MONGODB

- NPM I MONGOOSE
- CONST MONGOOSE = REQUIRE("MONGOOSE");
-

```
MONGOOSE.CONNECT("MONGODB://LOCALHOST:27017/CREDENTIALS");
```

■ RUN MONGODB

- TERMINAL: MONGOD
- BY DEFAULT IT LISTENS TO PORT 27017

■ 1. CREATE SCHEMA

- CONST USERSCHEMA = { ... };

■ 2. CREATE MODEL BASED ON THE SCHEMA

```
● CONST USER = NEW MONGOOSE.MODEL("USER (SINGULAR)",  
USERSCHEMA);
```

■ SET UP POST TO /REGISTER

○ LEVEL 2 - DATABASE ENCRYPTION

■ ENCRYPTION

- THE PROCESS OF ENCODING AND DECODING MESSAGES

■ CIPHER

- AN ALGORITHM TO PERFORMING ENCRYPTIONS AND DECRYPTIONS
- INSTALL AND USE MONGOOSE-ENCRYPTION
 - 1. NPM I MONGOOSE-ENCRYPTION
 - 2. CONST ENCRYPT = REQUIRE('MONGOOSE-ENCRYPTION');
 - 3. SCHEMA SHOULD BE CREATED USED NEW MONGOOSE.SCHEMA
 - 4. TWO METHODS OF GENERATING KEYS
- METHOD 1
 - 5. MODIFYING TO ONLY ENCRYPT CERTAIN FIELDS, IN THIS CASE ONLY PASSWORDS
 - ENCRYPTION OCCURS WHEN 'SAVE'
 - DECRYPTION OCCURS WHEN 'FIND'
- USING ENVIRONMENT VARIABLES TO KEEP SECRETS SAFE
 - WHAT ARE ENVIRONMENT VARIABLES AND HOW DOES IT KEEP SECRETS SAFE?
 - BASICALLY A FILE WHERE SECRET KEYS ARE CONTAINED IN
 - DOTENV
 - VERY POPULAR NPM PACKAGE THAT IS USED FOR WORKING WITH ENVIRONMENT VARIABLES
 - INSTALLATION & USAGE
 - 1. NPM I DOTENV
 - 2. IN .JS FILE:
 - PLACE REQUIRE('DOTENV').CONFIG() AT THE VERY TOP OF THE JS FILE

■ 3. CREATE A “.ENV” FILE IN THE ROOT OF THE DIRECTORY OF PROJECT

■ 4. ADD ENVIRONMENT VARIABLES TO THE .ENV FILE

- FORMAT:

- NAME=VALUE

■ 5. ACCESSING ENVIRONMENT VALUE

- PROCESS.ENV.<KEY NAME>

■ 6. PLACE THE .ENV FILE INTO A .GITIGNORE

■ LEVEL 3 - HASHING PASSWORDS

- WE WON’T NEED AN ENCRYPTION KEY FOR HASHING

- HOW IT WORKS

- PASSWORD -> <HASH FUNCTION> -> HASH

- STORE THE HASH IN THE DATABASE

- THIS MAKES IT SO THAT WE DO NOT HAVE TO STORE THE USER’S
PASSWORD

- HASH FUNCTIONS ARE 1-WAY FUNCTION

- EASY TO “ENCRYPT” A PASSWORD, BUT ALMOST MATHEMATICALLY
IMPOSSIBLE TO GO BACK AND BE ABLE TO FIND THE PASSWORD

- MD5

- NPM PACKAGE FOR HASHING

■ HACKING 101

- 1. HASH OF THE PASSWORD IS OBTAINED

- 2. REVERSE HASH TABLE (PRE-BUILD HASHTABLES) CAN BE
GENERATED

AND A POWERFUL COMPUTER CAN BE USED TO HACK THE PASSWORD

- 3. DICTIONARY ATTACK
- LEVEL 4 - SALTING AND HASHING PASSWORDS WITH BCRYPT
 - PREVENTS DICTIONARY ATTACKS OR HASH TABLE ATTACKS
 - SALTING
 - PASSWORD + SALT (RANDOM SET OF CHARACTERS) ARE GIVEN TO THE HASH FUNCTION TO GENERATE THE HASH
 - SALT VALUE IS STORED IN THE DATABASE, ALONG WITH THE HASH
 - BCRYPT
 - INDUSTRY STANDARD HASHING ALGORITHMS THAT DEVELOPER'S USE
 - MAKES IT MUCH SLOWER TO GENERATE THE HASHES
 - ONLY WORKS FOR SPECIFIC AND STABLE VERSIONS OF NODE.JS
 - ALSO USES SALT ROUNDS
 - SALT ROUNDS
 - NUMBER OF ROUNDS/ITERATIONS TO SALT THE PASSWORD
 - MORE ROUNDS -> MORE SECURE IT IS
 - NVM
 - NODE VERSION MANAGER
 - ALLOWS USING SWITCHING BETWEEN MULTIPLE VERSIONS OF NODE
- WHAT ARE COOKIES AND SESSIONS?
 - COOKIES AND SESSIONS STORE INFORMATION FROM BROWSER INTERACTION
 - COOKIES
 - STORED ON THE BROWSER

- SESSION
 - STORED ON THE BROWSER, AS WELL AS THE SERVER
 - COOKIES ARE CREATED IN RESPONSE TO A POST REQUEST
 - COOKIES ARE SAVED, AND CAN BE RETRIEVED IN THE GET REQUEST
 - SESSION
 - PERIOD OF TIME THAT A BROWSER INTERACTS WITH A SERVER
 - COOKIES ARE USED TO MAINTAIN A SESSION
 - MAINTAINS AUTHENTICATION UNTIL LOGGING OUT, WHICH IS WHEN THE COOKIE AND THUS SESSION ENDS
- USING PASSPORT.JS TO ADD COOKIES AND SESSIONS
 - NPM INSTALL:
 - PASSPORT
 - PASSPORT-LOCAL
 - PASSPORT-LOCAL-MONGOOSE
 - EXPRESS-SESSION
 - LEVEL 6 - OAUTH 2.0
 - OPEN STANDARD FOR TOKEN BASED AUTHORIZATION
 - DELEGATE SECURITY TO LARGER COMPANIES (FACEBOOK, LINKEDIN)
 - WHY OATH?
 - 1. GRANT GRANULAR LEVEL OF ACCESS
 - CAN REQUEST SPECIFIC DATA ACCORDINGLY
 - 2. READ ONLY OR READ + WRITE ACCESS
 - 3. REVOKE ACCESS

- HOW TO SETUP OAUTH?
- 1. SETUP APP WITH 3RD PARTY (FACEBOOK, GOOGLE, LINKEDIN)
 - RETURNS AN APPID OR CLIENTID
- 2. REDIRECT TO AUTHENTICATE
- 3. USER LOGS IN
- 4. USER GRANTS PERMISSIONS
- 5. RECEIVE AUTHORIZATION CODE FROM (FACEBOOK)
- 6. EXCHANGE AUTHENTICATION CODE FOR ACCESS TOKEN
 - RECEIVE ACCESS TOKEN AND SAVE INTO OUR DATABASE
- AUTH CODE VS ACCESS TOKEN
- AUTH CODE - WORKS ONE TIME
- ACCESS TOKEN - WORKS MULTIPLE TIME

23. DESIGN SCHOOL 101

- Understanding Color Palette
 - Colors Invoke Emotion
 - Red
 - Exciting, Intensity, Love, Fast
 - Yellow
 - Attention grabbing, Joy
 - Green
 - Freshness, Safety, Growth
 - Blue
 - Stability, Trust, Serenity
 - Purple
-

24. PRE REQUISITE

- Always be learning
 - New technologies will constantly come up, you should be constantly learning so you don't stagnate.
 - Spend 1 hour everyday to learn.
 - You learn the most from being out of your comfort zone
 - Be able to figure out and research things on your own.
-

PLACEMENT QUESTIONS MOSTLY ASKED BY COMPANIES:

HTML:

1. **What does HTML stand for?**

Answer: HTML stands for Hypertext Markup Language.

2. **How do you create a hyperlink in HTML?**

Answer: You can create a hyperlink in HTML using the `<a>` (anchor) element, like this: `Link Text`.

3. **What is the purpose of HTML tags?**

Answer: HTML tags are used to define elements and structure web content. They specify how content should be displayed or formatted in a web page.

4. **What is the HTML5 `<!DOCTYPE>` declaration used for?**

Answer: The HTML5 `<!DOCTYPE>` declaration defines the document type and version of HTML, ensuring that browsers render the page correctly.

5. **How can you add a comment in HTML?**

Answer: You can add an HTML comment using the `<!-- your comment here -->` syntax.</p>

6. **Explain the difference between HTML and HTML5.**

Answer: HTML5 is the latest version of HTML, which introduced new elements, attributes, and improved support for multimedia. It is more semantically structured and supports modern web standards.

7. **What is an HTML element?**

Answer: An HTML element is a complete set of tags that define a particular part of a web page content, such as headings, paragraphs, links, images, etc.

8. **What is the purpose of the `<head>` element in an HTML document?**

Answer: The `<head>` element contains meta-information about the document, like the title, character encoding, and linked stylesheets, but it's not displayed on the web page itself.

9. **How do you create an ordered list in HTML?**

Answer: You can create an ordered list using the `` element with `` elements for each list item.

10. **What's the difference between HTML attributes and HTML elements?**

Answer: HTML elements are the building blocks of a web page, while HTML attributes are used to provide additional information about an element.

11. **What is semantic HTML, and why is it important?**

Answer: Semantic HTML is a way of writing HTML that reflects the meaning and structure of content. It's important for accessibility, search engine optimization (SEO), and clear document structure.

12. **How can you embed an image in an HTML page?**

Answer: You can embed an image using the `` element, like this: ``.

13. **What's the difference between `<div>` and `` elements?**

Answer: `<div>` is a block-level element used for grouping and styling larger sections of content, while `` is an inline element used for styling smaller portions of text or content.

14. **How do you create a table in HTML?**

Answer: You can create a table using the `<table>` element with `<tr>` for rows, `<th>` for headers, and `<td>` for data cells.

15. **What's the purpose of the `<meta>` tag in HTML?**

Answer: The `<meta>` tag is used to provide metadata about the HTML document, such as character encoding, author, and description.

16. **How do you create a form in HTML?**

Answer: You can create a form using the `<form>` element and add input elements like text fields, radio buttons, and submit buttons within it.

17. **What is the `<iframe>` element used for in HTML?**

Answer: The `<iframe>` element is used to embed external content like web pages or videos within an HTML page.

18. **How do you create a bullet list in HTML?**

Answer: You can create a bullet list using the `` element with `` elements for each list item.

19. **Explain the role of the `<a>` element's `target` attribute.**

Answer: The `target` attribute specifies how a linked page should open. For example, "_blank" opens the link in a new tab or window.

20. **What is an HTML entity, and why are they used?**

- Answer: HTML entities are codes used to represent special characters (e.g., © for ©) to ensure proper rendering in HTML documents.

21. **How do you create a line break in HTML?**

Answer: You can create a line break using the `
` element.

22. **What's the purpose of the `<input>` element in HTML forms?**

Answer: The `<input>` element is used for creating various types of form controls, such as text fields, checkboxes, and radio buttons.

23. **How can you add a video to an HTML page?**

Answer: You can add a video using the `<video>` element and specify the video file's source and options.

24. **What is the difference between `` and `` elements in HTML?**

Answer: `` is used for emphasizing the importance of text, typically displayed as bold, while `` is used for emphasizing text in a more general way, typically displayed in italics.

25. **Explain the concept of HTML validation.**

Answer: HTML validation is the process of checking if your HTML code follows the rules and standards set by the W3C. It ensures proper document structure and helps in debugging.

26. **How do you create a dropdown list (select menu) in HTML?**

Answer: You can create a dropdown list using the `<select>` element with `<option>` elements for each item.

27. **What are HTML5 semantic elements, and name a few examples.**

Answer: HTML5 introduced semantic elements like `<header>`, `<nav>`, `<section>`, `<article>`, `<footer>`, etc., which describe the purpose of the content they contain.

28. **How do you add a background image to an HTML page?**

Answer: You can add a background image to an HTML page using CSS with the `background-image` property.

29. **What is the purpose of the `<label>` element in HTML forms?**

Answer: The `<label>` element provides a text description for a form element, improving accessibility and user experience.

30. **How do you include special characters (e.g., copyright symbol) in HTML?**

Answer: You can include special characters using HTML entities, such as `©` for ©, or use the corresponding Unicode characters.

The Coding Wizard

CSS:

1. **What does CSS stand for?**

Answer: CSS stands for Cascading Style Sheets.

2. **How do you include an external CSS file in an HTML document?**

Answer: You can link an external CSS file using the `<link>` element within the HTML document's `<head>` section.

3. **Explain the difference between inline, internal, and external CSS.**

Answer: Inline CSS is applied directly to an HTML element, internal CSS is defined within a `<style>` element in the HTML document, and external CSS is in a separate .css file linked to the HTML file.

4. **What is the CSS box model?**

Answer: The CSS box model defines how elements are rendered by describing their content, padding, borders, and margins.

5. **What's the purpose of the `position` property in CSS?**

Answer: The `position` property specifies the positioning method for an element, such as relative, absolute, fixed, or static.

6. **Explain the difference between `margin` and `padding`.**

Answer: `Margin` is the space outside an element, while `padding` is the space inside an element, between the content and the border.

7. **How do you center an element horizontally in CSS?**

Answer: You can center an element horizontally by setting its margin to auto and specifying a width.

8. **What is the `z-index` property used for in CSS?**

Answer: The `z-index` property determines the stacking order of elements on a webpage, allowing you to control which element appears on top.

9. **What does the `display: none;` property do in CSS?**

Answer: `display: none;` hides an element completely, and it won't take up space on the page.

10. **What is the purpose of the `float` property in CSS?**

Answer: The `float` property is used for aligning elements to the left or right within their containing elements.

11. **What is the difference between `em` and `rem` units in CSS?**

Answer: `em` units are relative to the font size of the nearest parent element, while `rem` units are relative to the root (html) element's font size.

12. **How do you add a CSS rule for a specific HTML element with a certain class name?**

Answer: You can use the selector `classname` to select elements with a specific class, e.g., `myclass { property: value; }`.

13. **What is a CSS pseudo-class?**

Answer: A CSS pseudo-class is used to define a special state or behavior of an element, like `:hover` for mouse hover.

14. **How can you select all even or odd elements in a list using CSS?**

Answer: You can use `:nth-child(even)` or `:nth-child(odd)` to select even or odd elements in a list.

15. **What is the purpose of the `@media` rule in CSS?**

Answer: The `@media` rule allows you to apply different styles to a webpage based on the device's screen size or other criteria.

16. **What is a CSS selector and how do you use it to target elements?**

Answer: A CSS selector is a pattern used to select elements for styling. For example, to select all `

` elements, you use the selector `p`.

17. **What is the difference between `inline` and `block` elements in CSS?**

Answer: `Inline` elements flow inline with text and only take up as much width as necessary, while `block` elements create a new block formatting context and take up the full width.

18. **What is the purpose of the `box-sizing` property in CSS?**

Answer: The `box-sizing` property controls how an element's width and height are calculated, affecting how padding and borders are included.

19. **What is the CSS `:before` and `:after` pseudo-elements used for?**

Answer: `:before` and `:after` pseudo-elements allow you to insert content before and after an element's content, typically for decorative or informational purposes.

20. **How do you apply a CSS style to all links within a specific div?**

Answer: You can use a descendant selector, like `div a`, to select all links within a specific div.

21. **What is the difference between `absolute` and `relative` positioning in CSS?**

Answer: `Absolute` positioning is relative to the nearest positioned ancestor, while `relative` positioning is relative to the element's normal position in the document flow.

22. **What is the CSS `opacity` property used for?**

Answer: The `opacity` property controls the transparency of an element, where a value of 0 is completely transparent and 1 is fully opaque.

23. **What is a CSS selector specificity and how is it calculated?**

Answer: Selector specificity is a way to determine which style rule applies when multiple rules conflict. It's calculated based on the number of IDs, classes, and elements in the selector.

24. **How can you create a CSS animation?**

Answer: You can create a CSS animation using the `@keyframes` rule, specifying the animation's keyframes and properties to animate.

25. **What is the `line-height` property used for in CSS?**

Answer: The `line-height` property controls the amount of space between lines of text within an element.

26. **What is the CSS `overflow` property used for?**

Answer: The `overflow` property controls what happens when content overflows the boundaries of an element, such as scrolling or hiding.

27. **How do you create a CSS gradient background?**

Answer: You can create a gradient background using the `background-image` property with linear or radial gradients.

28. **What is the CSS `transition` property used for?**

Answer: The `transition` property allows you to smoothly change property values over a specified duration, creating smooth animations.

29. **Explain the difference between `margin: 0 auto;` and `text-align: center;` for centering elements horizontally.**

Answer: `margin: 0 auto;` centers a block-level element within its parent, while `text-align: center;` centers the text within an element.

30. **What is the purpose of the `box-shadow` property in CSS?**

Answer: The `box-shadow` property adds a shadow effect to an element, allowing you to control its size, color, and position.

JavaScript:

1. **What is JavaScript?**

Answer: JavaScript is a high-level, interpreted scripting language used for making web pages interactive and providing dynamic behavior to websites.

2. **What is the difference between JavaScript and Java?**

Answer: JavaScript and Java are distinct languages. JavaScript is a scripting language used for web development, while Java is a general-purpose programming language.

3. **How do you declare a variable in JavaScript?**

Answer: Variables in JavaScript can be declared using `var`, `let`, or `const`. For example, `var x;` or `let name = "John";`.

4. **What is a data type in JavaScript?**

Answer: A data type in JavaScript defines the type of data that a variable can hold. Common data types include numbers, strings, booleans, and objects.

5. **Explain the difference between `null` and `undefined` in JavaScript.**

Answer: `null` is an explicitly assigned value that represents the absence of any object value, while `undefined` means a variable has been declared but hasn't been assigned a value.

6. **What is a JavaScript function?**

Answer: A JavaScript function is a reusable block of code that can be called to perform a specific task. Functions are declared using the `function` keyword.

7. **How do you call a function in JavaScript?**

Answer: You call a function by using its name followed by parentheses, like `myFunction();`.

8. **What is the difference between `==` and `===` in JavaScript?**

Answer: `==` tests for equality after type coercion, while `===` tests for equality without type coercion (strict equality).

9. **What is a closure in JavaScript?**

Answer: A closure is a function that has access to its own scope, the outer function's scope, and the global scope, even after the outer function has finished execution.

10. **What is the purpose of the `this` keyword in JavaScript?**

Answer: `this` refers to the current object, typically the object that the function is a method of.

11. **How do you loop through an array in JavaScript?**

Answer: You can loop through an array using `for` loops, `while` loops, or array-specific methods like `forEach`.

12. **What is an event handler in JavaScript?**

Answer: An event handler is a function that is executed when a specific event occurs, such as a button click or a keypress.

13. **How do you create an object in JavaScript?**

Answer: You can create an object using object literal notation, constructor functions, or the ES6 class syntax.

14. **What is the difference between `null` and `undefined` in JavaScript?**

Answer: `null` is a value that represents the intentional absence of any object value, while `undefined` indicates that a variable has been declared but hasn't been assigned a value.

15. **Explain the concept of callback functions in JavaScript.**

Answer: A callback function is a function passed as an argument to another function, and it is executed after the completion of the parent function.

16. **What is an arrow function in JavaScript?**

Answer: An arrow function is a concise way to write functions in JavaScript, introduced in ES6, using the `=>` syntax.

17. **What is the purpose of the `localStorage` and `sessionStorage` in JavaScript?**

Answer: `localStorage` and `sessionStorage` are used for storing key-value pairs in a web browser, but `localStorage` persists data across sessions, while `sessionStorage` is session-specific.

18. **What is the Document Object Model (DOM) in JavaScript?**

Answer: The DOM is a programming interface for web documents. It represents the page so that programs can change the document structure, content, and style dynamically.

19. **What is an asynchronous function in JavaScript?**

Answer: An asynchronous function doesn't block the main execution thread. It allows other tasks to run while it performs time-consuming operations.

20. **How do you handle errors in JavaScript?**

Answer: Errors in JavaScript can be handled using `try`, `catch`, and `finally` blocks.

21. **What is the purpose of the `fetch` API in JavaScript?**

Answer: The `fetch` API is used to make network requests and retrieve data from web servers, often used for making HTTP requests.

22. **What is the difference between `var`, `let`, and `const` in JavaScript for variable declaration?**

Answer: `var` has function scope, `let` has block scope, and `const` is used for constant values that can't be reassigned.

23. **What is a callback hell, and how can it be avoided?**

Answer: A callback hell occurs when multiple nested callbacks make code hard to read. It can be avoided by using Promises, async/await, or modularizing code.

24. **What is event delegation in JavaScript?**

Answer: Event delegation is a technique where a single event handler is attached to a common ancestor of multiple elements, making it easier to manage events on those elements.

25. **Explain the concept of hoisting in JavaScript.**

Answer: Hoisting is a JavaScript behavior where variable and function declarations are moved to the top of their containing scope during compilation.

26. **What is a JavaScript constructor function?**

Answer: A constructor function is a function used to create and initialize objects. It is typically used with the `new` keyword.

27. **How do you add a class to an element in JavaScript?**

Answer: You can add a class to an element using the `classList` property and its `add` method, like `element.classList.add('myClass');`.

28. **What is the purpose of the `map` method in JavaScript for arrays?**

Answer: The `map` method creates a new array by applying a given function to each element of an existing array.

29. **What is the `typeof` operator in JavaScript used for?**

Answer: The `typeof` operator is used to determine the data type of a value or variable.

30. **How do you prevent variable pollution in the global scope in JavaScript?**

Answer: You can prevent variable pollution by encapsulating code in functions and using block-scoped variables ('let' and 'const') to limit the scope of variables.

The Coding Wizard

Bootstrap 5

1. **What is Bootstrap 5, and how does it differ from previous versions of Bootstrap?**

Answer: Bootstrap 5 is the latest version of the Bootstrap framework. It introduces new features and improvements, such as a more lightweight and customizable design, utility classes, and removal of jQuery dependency.

2. **Explain the key components of Bootstrap 5.**

Answer: Bootstrap 5 includes components like the grid system, navigation bars, forms, buttons, cards, modals, and more. It provides a wide range of UI elements for web development.

3. **How do you include Bootstrap 5 in a web project?**

Answer: You can include Bootstrap 5 in a project by linking to the Bootstrap CSS and JavaScript files in your HTML document, or by installing it via npm or yarn.

4. **What is the Bootstrap grid system, and how does it work?**

Answer: The Bootstrap grid system is a responsive, 12-column layout system. It allows you to create flexible and responsive page layouts by using a combination of containers, rows, and columns.

5. **Explain the purpose of responsive design in Bootstrap 5.**

Answer: Responsive design in Bootstrap 5 ensures that web pages look and function well on various devices and screen sizes. It uses a mobile-first approach, and responsive classes help adapt content to different screen sizes.

6. **What are Bootstrap utility classes, and how can you use them?**

Answer: Bootstrap utility classes are predefined classes that provide quick styling and layout adjustments. You can use them in HTML elements to apply common styles and effects.

7. **How do you create a navigation bar in Bootstrap 5?**

Answer: You can create a navigation bar by using the Bootstrap `navbar` class along with navigation components like `navbar-nav` and `nav-item`.

8. **Explain the purpose of Bootstrap forms and form controls.**

Answer: Bootstrap forms and form controls provide a structured and stylized way to create input forms, buttons, checkboxes, radio buttons, and more for user interactions.

9. **What is a Bootstrap modal, and how can you create one?**

Answer: A Bootstrap modal is a dialog box that can be used for various purposes, such as displaying additional information or collecting user input. You can create one using the `modal` class and JavaScript to control its visibility.

10. **Explain how to create a Bootstrap 5 carousel.**

Answer: You can create a carousel using the Bootstrap `carousel` class and the required HTML structure. You need to define carousel items and include navigation elements.

11. **What is the purpose of Bootstrap cards, and how can you use them for content display?**

Answer: Bootstrap cards are flexible containers used to display various types of content, such as text, images, and links. You can use them by adding the ` .card` class and customizing the card's content.

12. **How do you create a Bootstrap 5 dropdown menu?**

Answer: You can create a dropdown menu using the ` .dropdown` class and associated components like ` .dropdown-menu` and ` .dropdown-item`.

13. **What is the Bootstrap 5 button group, and when is it used?**

Answer: A Bootstrap button group is used to group a set of buttons together for better organization. It can be applied to radio buttons, checkboxes, or regular buttons.

14. **Explain the purpose of Bootstrap's responsive tables.**

Answer: Bootstrap's responsive tables adapt to different screen sizes by allowing horizontal scrolling or stacking table data. They ensure that tabular data remains readable on small screens.

15. **How do you create a responsive image in Bootstrap 5?**

Answer: To create a responsive image, add the ` .img-fluid` class to the ` img` element. This ensures that the image scales proportionally with the width of the parent container.

16. **What is the Bootstrap 5 alert component, and when should it be used?**

Answer: Bootstrap alerts are used to display messages or notifications. They come in different styles (success, info, warning, danger) and can be dismissed by users.

17. **How do you use Bootstrap badges to display additional information in your content?**

Answer: You can use Bootstrap badges to highlight and display additional information like counts, labels, or statuses. Just apply the `badge` class to the HTML element.

18. **What is the Bootstrap 5 accordion component, and when is it used for content organization?**

Answer: The Bootstrap accordion component is used to organize and hide/show content sections. It allows users to expand or collapse sections to focus on specific information.

19. **How do you create a Bootstrap 5 breadcrumb navigation trail?**

Answer: You can create a breadcrumb navigation trail using the `breadcrumb` class and `ol` and `li` elements to define the trail.

20. **Explain the purpose of Bootstrap's tooltips and popovers.**

Answer: Bootstrap tooltips provide helpful information when users hover over an element, while popovers display additional content or context when triggered.

21. **What is the role of Bootstrap's pagination component, and how can you create it?**

Answer: Bootstrap's pagination component is used to split content across multiple pages. You can create it by adding the `pagination` class to an ordered or unordered list.

22. **How do you use Bootstrap's navs and tabs for content navigation and organization?**

Answer: You can create navigation tabs using the `nav` and `nav-tabs` classes and use them to switch between different sections of content.

23. **What is the Bootstrap collapse component, and when is it used to hide or show content?**

Answer: The Bootstrap collapse component allows you to hide or show content sections. It is often used for collapsible content, such as accordions or collapsible navigation menus.

24. **How can you customize the default Bootstrap styles to match your project's design?**

Answer: You can customize Bootstrap styles by overriding the default CSS classes, using custom CSS, or creating a custom theme based on Bootstrap's variables.

25. **What is the Bootstrap 5 off-canvas component, and how can it be used for navigation menus?**

Answer: The Bootstrap off-canvas component is used for creating navigation menus that slide in from the side of the page. It is helpful for responsive design and mobile navigation.

26. **Explain the purpose of Bootstrap's responsive display classes.**

Answer: Bootstrap's responsive display classes help control the visibility of content based on screen size. They allow you to hide or show content on specific devices or screen widths.

27. ****How do you create a Bootstrap 5 progress bar, and when is it used for showing progress or completion?****

Answer: You can create a progress bar by using the `progress` class. It is used to display the progress of a task, such as file uploads or form submissions.

28. ****What is the Bootstrap 5 list group, and how can it be used for displaying lists of items?****

Answer: The Bootstrap list group is used to display lists of items. It's a versatile component for showing collections of content, such as user profiles or messages.

29. ****How do you create a Bootstrap 5 spinner for indicating loading or processing?****

Answer: You can create a spinner using the `spinner-border` or `spinner-grow` classes. Spinners are used to indicate that a task is in progress.

30. ****Explain the purpose of Bootstrap's form validation styles, and how can you use them in your forms?****

Answer: Bootstrap's form validation styles help indicate the validity of form inputs. You can apply classes like `is-valid` and `is-invalid` to form controls to display feedback to users.

ReactJS:

1. **What is React.js, and why is it used?**

Answer: React.js is an open-source JavaScript library used for building user interfaces. It is popular for its component-based architecture, performance optimization, and reusability.

2. **Explain the difference between React and React Native.**

Answer: React is used for building web applications, while React Native is used for building mobile applications for iOS and Android.

3. **What is JSX in React, and why is it beneficial?**

Answer: JSX (JavaScript XML) is a syntax extension used to write HTML-like code within JavaScript. It helps create React elements in a more readable and maintainable way.

4. **What are React components?**

Answer: React components are reusable building blocks for user interfaces. They can be class-based or functional components and encapsulate the UI and behavior.

5. **How do you create a functional component in React?**

Answer: You can create a functional component using a JavaScript function that returns JSX. For example, `function MyComponent() { return <div>Hello, World!</div>; }`.

6. **What is the state in React, and how is it managed in components?**

Answer: State is a built-in object in React components that stores mutable data. In class components, state is managed using `this.state` and `this.setState()`. In functional components, you can use the `useState` hook.

7. **What is a prop in React, and how are they passed to components?**

Answer: Props (short for properties) are a way to pass data from parent components to child components. They are passed as attributes in JSX and accessed within the child component.

8. **What is the virtual DOM in React, and how does it improve performance?**

Answer: The virtual DOM is an in-memory representation of the actual DOM. React uses it to efficiently update the real DOM by calculating and minimizing changes, resulting in improved performance.

9. **Explain the component lifecycle in React and some commonly used lifecycle methods.**

Answer: The component lifecycle consists of three main phases: mounting, updating, and unmounting. Commonly used lifecycle methods include `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.

10. **What are React hooks, and why are they used?**

Answer: React hooks are functions that allow functional components to have state and lifecycle features. They simplify state management, side effects, and lifecycle handling.

11. **What is conditional rendering in React?**

Answer: Conditional rendering in React allows components to render different content based on conditions, such as using `if` statements or the ternary operator in JSX.

12. **Explain the concept of event handling in React.**

Answer: Event handling in React involves attaching event listeners to DOM elements and defining functions to handle those events. For example, using `onClick` for handling clicks.

13. **What is React Router, and how is it used for routing in React applications?**

Answer: React Router is a popular library for handling client-side routing in React applications. It allows you to create multiple pages and manage navigation without full-page reloads.

14. **How do you make an API request in a React component?**

Answer: You can make API requests using methods like `fetch`, `axios`, or other HTTP libraries within React components, typically within lifecycle methods or `useEffect`.

15. **What are higher-order components (HOCs) in React?**

Answer: Higher-order components are functions that take a component and return a new enhanced component. They are used for code reuse and component composition.

16. **What is Redux, and what problem does it solve in React applications?**

Answer: Redux is a state management library for React that helps manage application state in a predictable and centralized way. It's often used for complex applications with a lot of shared state.

17. **What is the context API in React, and how is it used for state management?**

Answer: The context API is a built-in solution for state management in React. It allows data to be shared between components without having to pass props manually.

18. **What is the purpose of keys in React lists, and why are they important?**

Answer: Keys are used to identify elements in a React list. They help React identify which items have changed, been added, or removed efficiently.

19. **How can you optimize the performance of a React application?**

Answer: Performance optimization in React involves using techniques like lazy loading, code splitting, memoization, and minimizing unnecessary re-renders using PureComponent or shouldComponentUpdate.

20. **What is code splitting in React, and why is it useful?**

Answer: Code splitting is a technique for splitting your application's bundle into smaller chunks. It helps reduce the initial load time by only loading code when it's needed.

21. **What is SSR (Server-Side Rendering) in React, and when is it beneficial?**

Answer: SSR is a technique where React components are rendered on the server rather than in the browser. It can improve SEO and initial page load times.

22. **How does React handle forms and form inputs?**

Answer: React handles form inputs by using controlled components. Each form element's value is controlled by React state, and changes are handled with event handlers.

23. **What is the purpose of React DevTools, and how can they help in debugging?**

Answer: React DevTools is a browser extension that provides a set of debugging tools for inspecting and understanding the component hierarchy, state, and props of a React application.

24. **Explain the concept of lazy loading in React.**

Answer: Lazy loading involves loading components or resources only when they are needed, which helps reduce the initial bundle size and improve page load performance.

25. **What is the difference between `componentWillUnmount` and `useEffect` cleanup in React components?**

Answer: `componentWillUnmount` is a lifecycle method in class components used for cleanup, while `useEffect` in functional components can also handle cleanup using a return function.

26. **What is prop drilling in React, and how can it be avoided?**

Answer: Prop drilling occurs when props are passed through multiple levels of nested components. It can be avoided by using context or state management libraries like Redux.

27. ****What are error boundaries in React, and how do they work?****

Answer: Error boundaries are React components that catch errors during rendering, in lifecycle methods, and in constructors of their child components. They help display a fallback UI and log the error.

28. ****How can you integrate third-party libraries or plugins in a React application?****

Answer: You can integrate third-party libraries by installing them via npm or yarn and then importing and using them within your React components.

29. ****What is the purpose of React Fragments, and how are they used?****

Answer: React Fragments are used to group multiple elements without adding an extra parent element in the DOM. They are useful for cleaner JSX syntax.

30. ****What is the difference between controlled and uncontrolled components in React forms?****

Answer: Controlled components have their value controlled by React state and are updated through React event handlers, while uncontrolled components rely on the DOM and are accessed through refs.

Node.js:

1. **What is Node.js, and how is it different from JavaScript?**

Answer: Node.js is a runtime environment for executing JavaScript on the server. While JavaScript is a programming language, Node.js allows JavaScript to run on the server, enabling server-side applications.

2. **Explain the event-driven, non-blocking I/O model of Node.js.**

Answer: Node.js uses an event-driven, non-blocking I/O model, which means it can efficiently handle multiple requests without waiting for one to complete before moving to the next. It relies on callbacks and event loops for this.

3. **What is npm, and why is it important in Node.js development?**

Answer: npm (Node Package Manager) is a package manager for Node.js. It's important because it simplifies the installation, management, and sharing of libraries and dependencies used in Node.js projects.

4. **How can you create a simple HTTP server in Node.js?**

Answer: You can create an HTTP server in Node.js using the `http` module. Here's an example:

```
```javascript
const http = require('http');

const server = http.createServer((req, res) => {
 res.writeHead(200, { 'Content-Type': 'text/plain' });
 res.end('Hello, World!');
});

});
```

```
server.listen(8080, 'localhost');
```

```
```
```

5. **Explain the purpose of the `package.json` file in Node.js projects.**

Answer: The `package.json` file contains metadata about a Node.js project, including its dependencies, scripts, and other project-related information. It's crucial for managing and sharing projects.

6. **What is the difference between `require` and `import` in Node.js?**

Answer: `require` is the CommonJS way of importing modules in Node.js, while `import` is the ES6 way. In modern Node.js (v13+ with `--experimental-modules`), you can also use `import`.

7. **What is the purpose of the `module.exports` object in Node.js modules?**

Answer: The `module.exports` object is used to define the public interface of a Node.js module. It specifies which functions or objects can be accessed by other modules when they require the module.

8. **What is the Event Emitters module in Node.js, and how is it used?**

Answer: The Event Emitters module is a built-in module in Node.js that allows you to emit and handle custom events. It's used to implement the observer pattern and custom event handling.

9. **Explain the purpose of the `fs` module in Node.js.**

Answer: The `fs` (File System) module in Node.js is used for interacting with the file system, allowing you to perform various file-related operations, such as reading, writing, and manipulating files.

10. **What are Callbacks in Node.js, and why are they important?**

Answer: Callbacks are functions passed as arguments to other functions and are executed once the operation is complete. They are vital in Node.js for handling asynchronous operations like I/O, ensuring non-blocking code execution.

11. **What is the purpose of the `npm init` command, and how is it used to initialize a Node.js project?**

Answer: `npm init` is used to create a `package.json` file interactively. It prompts you to provide project details and sets up the initial configuration for your Node.js project.

12. **What is the `global` object in Node.js, and how is it different from the browser's `window` object?**

Answer: The `global` object in Node.js is similar to the `window` object in the browser but is accessible globally in Node.js. It contains global variables and functions available throughout your Node.js application.

13. **Explain the use of the `process` object in Node.js.**

Answer: The `process` object provides information and control over the Node.js process. It allows you to access environment variables, set up event listeners, and interact with the running Node.js application.

14. **What is the purpose of the `util` module in Node.js, and how is it used?**

Answer: The `util` module provides various utility functions that can help with debugging and formatting output. For example, it includes the `util.format` function for string formatting.

15. **What is the `Buffer` class in Node.js, and when is it commonly used?**

Answer: The `Buffer` class is a built-in module in Node.js used to handle binary data. It's often used when working with binary files, network operations, or for handling data in non-text formats.

16. **Explain the purpose of the `child_process` module in Node.js.**

Answer: The `child_process` module allows you to create and interact with child processes in Node.js. It's used for tasks such as running external commands or scripts.

17. **What is the difference between synchronous and asynchronous functions in Node.js?**

Answer: Synchronous functions block the event loop until they complete, while asynchronous functions allow the event loop to continue processing other tasks while they are running. Asynchronous functions typically use callbacks or Promises.

18. **What is the Node.js Event Loop, and how does it work?**

Answer: The Node.js Event Loop is responsible for handling asynchronous operations. It continuously checks the message queue for pending events, executes them, and then waits for more events.

19. **What are Promises in Node.js, and how do they help with handling asynchronous code?**

Answer: Promises are a way to handle asynchronous operations more cleanly and efficiently in Node.js. They represent a value that may not be available yet but will be resolved at some point, either successfully or with an error.

20. **What is the purpose of the `http` and `https` modules in Node.js?**

Answer: The `http` module is used for creating HTTP servers and making HTTP requests, while the `https` module extends this functionality to secure HTTPS (HTTP over TLS/SSL) connections.

21. **Explain the concept of middleware in Node.js frameworks like Express.**

Answer: Middleware functions in Node.js frameworks are functions that have access to the request and response objects. They are used to perform actions, modify requests or responses, and enhance the functionality of an application.

22. **What is the purpose of the `os` module in Node.js, and how is it used?**

Answer: The `os` module provides information about the operating system on which Node.js is running. It's used to access details such as CPU architecture, memory, and platform-specific data.

23. **Explain the role of the `cluster` module in Node.js.**

Answer: The `cluster` module allows you to create child processes for running multiple instances of a Node.js application. It helps leverage multi-core processors and distribute the workload.

24. **What is npm's `package-lock.json` file, and why is it used in Node.js projects?**

Answer: `package-lock.json` is an automatically generated file by npm to lock down the specific versions of installed dependencies. It ensures consistent dependency versions across different environments.

25. **What is the purpose of the `assert` module in Node.js, and how is it used for testing?**

Answer: The `assert` module is used for writing assertions to test code for correctness. It provides a set of assertion functions to check conditions and report any errors.

26. **Explain the purpose of the `crypto` module in Node.js.**

Answer: The `crypto` module provides cryptographic functionality for secure data processing, including encryption, decryption, hashing, and digital signatures.

27. **What is the purpose of the `stream` module in Node.js, and how is it used?**

Answer: The `stream` module is used for handling data streams in Node.js. It allows you to read, write, and manipulate data in chunks, making it efficient for large datasets.

28. **What is the `url` module in Node.js, and how is it used to parse and format URLs?**

Answer: The `url` module provides methods for working with URLs. You can use it to parse, format, and manipulate URLs, making it easier to work with web addresses.

29. **What is the purpose of the `querystring` module in Node.js, and how is it used for working with query parameters?**

Answer: The `querystring` module is used for working with query parameters in URLs. It allows you to parse and stringify query parameters in a URL.

30. **What is the purpose of the `dgram` module in Node.js, and how is it used for UDP (User Datagram Protocol) communication?**

Answer: The `dgram` module in Node.js is used for implementing UDP communication. It provides an interface for sending and receiving datagrams, making it suitable for low-latency, connectionless communication.

MongoDB

1. **What is MongoDB, and what kind of database is it?**

Answer: MongoDB is a NoSQL database that falls under the category of document-oriented databases. It stores data in a flexible, JSON-like format called BSON.

2. **Explain the key features of MongoDB.**

Answer: MongoDB features include a flexible schema, horizontal scalability, support for geospatial data, powerful query language, and high availability through replica sets.

3. **What is a BSON in MongoDB?**

Answer: BSON (Binary JSON) is a binary-encoded serialization of JSON-like documents used for data storage and retrieval in MongoDB.

4. **What is a collection in MongoDB, and how does it relate to a table in a relational database?**

Answer: A collection in MongoDB is a group of documents, similar to a table in a relational database. However, MongoDB does not enforce a schema for collections.

5. **Explain the difference between a document and a collection in MongoDB.**

Answer: A document is a single record or data entity in MongoDB, while a collection is a group of related documents, similar to a table in a relational database.

6. **What is a schemaless database, and how does MongoDB fit this description?**

Answer: A schemaless database does not enforce a fixed structure for data. MongoDB is schemaless because documents within a collection can have different fields.

7. **What is the primary key in MongoDB, and how is it defined?**

Answer: In MongoDB, the `_id` field acts as the primary key, uniquely identifying each document in a collection. If not provided, MongoDB generates a unique identifier for each document.

8. **Explain the concept of indexing in MongoDB and its benefits.**

Answer: Indexing in MongoDB is the process of creating data structures to improve query performance. It helps reduce query execution time by providing efficient access to documents.

9. **What are the primary types of indexes in MongoDB?**

Answer: MongoDB supports several index types, including single-field indexes, compound indexes, and geospatial indexes.

10. **Explain the purpose of the `find` method in MongoDB and how it is used to query documents.**

Answer: The `find` method is used to query documents in a collection. It allows you to specify criteria to filter documents and return the matching results.

11. **What is aggregation in MongoDB, and when is it used?**

Answer: Aggregation in MongoDB is a framework for processing and transforming data within a collection. It is used for complex operations like grouping, sorting, and performing calculations on data.

12. **What is a MongoDB document's `_id` field, and why is it important?**

Answer: The `_id` field is a unique identifier for each document in a collection. It is crucial for document retrieval, updates, and ensuring data consistency.

13. **Explain the difference between `update` and `save` methods in MongoDB.**

Answer: The `update` method modifies existing documents based on specified criteria, while the `save` method either updates an existing document or inserts a new one if it doesn't exist.

14. **What is a replica set in MongoDB, and why is it used?**

Answer: A replica set is a group of MongoDB servers that maintain the same data set. It is used for high availability, automatic failover, and data redundancy.

15. **How does sharding work in MongoDB, and what problem does it solve?**

Answer: Sharding is a technique used to distribute data across multiple servers. It addresses scalability issues by allowing large datasets to be spread across multiple machines.

16. **Explain the Map-Reduce model in MongoDB and its use cases.**

Answer: Map-Reduce is a data processing model used in MongoDB for aggregating and transforming large datasets. It is suitable for complex data analysis and reporting.

17. **What is the GridFS in MongoDB, and when is it used?**

Answer: GridFS is a specification for storing large binary files in MongoDB, typically files that are larger than 16 MB. It breaks files into smaller chunks for efficient storage and retrieval.

18. **What is the `'\$set` operator in MongoDB, and how is it used to update documents?**

Answer: The `'\$set` operator is used in MongoDB to update specific fields in a document without affecting other fields. It allows you to set or change the value of a field.

19. **Explain the role of the `'\$match` stage in MongoDB's aggregation framework.**

Answer: The `'\$match` stage filters documents in an aggregation pipeline, allowing you to select only those that match specific criteria.

20. **What is a TTL index in MongoDB, and how is it used for data expiration?**

Answer: A TTL (Time-To-Live) index in MongoDB is used to automatically delete documents that have exceeded a certain age. It is often used for session data and logs.

21. **What is connection pooling in MongoDB, and how does it improve performance?**

Answer: Connection pooling is a technique used to manage and reuse database connections efficiently. It improves performance by reducing the overhead of establishing new connections for each request.

22. **Explain the concept of horizontal scaling in MongoDB and how it is achieved.**

Answer: Horizontal scaling in MongoDB involves adding more servers to a cluster to accommodate increased data and traffic. It is achieved through the use of sharding.

23. **What is the `'\$lookup` stage in MongoDB's aggregation framework, and how is it used for joining collections?**

Answer: The `'\$lookup` stage is used to perform left outer joins between collections in an aggregation pipeline, allowing you to combine data from multiple collections.

24. **What is the purpose of the `mongod` process in MongoDB, and how does it relate to the database server?**

Answer: The `mongod` process is the primary MongoDB database process responsible for handling data storage and retrieval. It is the heart of the database server.

25. **Explain the role of indexes in MongoDB and how they impact query performance.**

Answer: Indexes in MongoDB provide a way to access and retrieve data quickly. They significantly improve query performance by allowing the database to find and retrieve documents efficiently.

26. **What is the `'\$unwind` stage in MongoDB's aggregation framework, and when is it used?**

Answer: The `'\$unwind` stage is used to deconstruct arrays within documents. It creates multiple copies of a document for each element in an array, making it easier to process array data.

27. **What is the purpose of the `mongos` process in MongoDB's sharded clusters?**

Answer: The `mongos` process is a routing service in a sharded cluster. It routes client requests to the appropriate shard and acts as an interface for querying sharded data.

28. **Explain the difference between a capped collection and a regular collection in MongoDB.**

Answer: A capped collection is a fixed-size collection where documents are automatically removed in the order they were inserted. Regular collections have no size limits and retain data indefinitely.

29 **What is a compound index in MongoDB, and how is it different from single-field indexes?**

Answer: A compound index involves multiple fields and is used to optimize queries that involve multiple criteria. It differs from single-field indexes that optimize queries based on a single field.

30. **What is a covered query in MongoDB, and how does it impact query performance?**

Answer: A covered query is a query in which all the fields in the query and the projection are covered by an index. It improves query performance by avoiding the need to load documents from disk.

The Coding Wizard

Express.js

1. **What is Express.js, and how does it relate to Node.js?**

Answer: Express.js is a web application framework for Node.js. It simplifies the process of building web applications by providing a set of features and tools.

2. **Explain the key features of Express.js.**

Answer: Express.js features include routing, middleware support, template engines, and a robust set of HTTP utility methods.

3. **How do you install Express.js in a Node.js project?**

Answer: You can install Express.js using npm (Node Package Manager) with the command: `npm install express`.

4. **What is middleware in Express.js, and how does it work?**

Answer: Middleware functions in Express.js are functions that have access to the request and response objects. They can execute code, modify the request or response, and end the request-response cycle.

5. **Explain routing in Express.js and how you define routes.**

Answer: Routing in Express.js is the process of defining how the application responds to specific HTTP requests. You define routes using HTTP methods and URL patterns.

6. **What is a route parameter in Express.js, and how is it used?**

Answer: Route parameters in Express.js are named segments in a URL. They are used to capture values from the URL and pass them as arguments to the route handler.

7. **What is the purpose of the `req` and `res` objects in Express.js?**

Answer: The `req` (request) object represents the HTTP request, while the `res` (response) object represents the HTTP response. You use these objects to handle client requests and send responses.

8. **What is the purpose of middleware like `body-parser` in Express.js, and how is it used?**

Answer: Middleware like `body-parser` is used to parse request bodies. It's essential for processing POST and PUT requests that contain data in the request body.

9. **Explain the difference between `app.use` and `app.get` in Express.js.**

Answer: `app.use` is used to apply middleware to every request, while `app.get` is used to define a route that responds only to GET requests.

10. **What is a template engine in Express.js, and why is it useful?**

Answer: A template engine in Express.js allows you to generate dynamic HTML content by rendering data with templates. It's useful for building views in web applications.

11. **What is the purpose of Express.js's error-handling middleware?**

Answer: Error-handling middleware in Express.js is used to catch errors that occur during request processing and respond with an appropriate error message or status code.

12. **What is the purpose of static files in Express.js, and how do you serve them?**

Answer: Static files are assets like images, CSS, and JavaScript. You can serve them using the built-in `express.static` middleware to make them accessible to clients.

13. **What is a RESTful API, and how can you create one with Express.js?**

Answer: A RESTful API is a set of rules for building web services. You can create one with Express.js by defining routes and route handlers that respond to HTTP methods like GET, POST, PUT, and DELETE.

14. **What is CORS, and why might you need to enable it in an Express.js application?**

Answer: CORS (Cross-Origin Resource Sharing) is a security feature that controls how web pages in one domain can request resources from a different domain. In Express.js, you might need to enable it to allow cross-origin requests.

15. **Explain the purpose of Express.js session management, and how can you implement it?**

Answer: Session management in Express.js involves tracking user sessions and authentication state. You can implement it using middleware like `express-session` to store session data on the server.

16. **What is the role of the `next` function in Express.js middleware, and how is it used?**

Answer: The `next` function is used to pass control to the next middleware or route handler in the request-response cycle. It allows middleware to execute and pass control to subsequent middleware.

17. **Explain the purpose of the `app.locals` object in Express.js, and how is it used for data sharing?**

Answer: The `app.locals` object in Express.js allows you to store application-wide data. It is accessible from all parts of your application, making it suitable for sharing data across requests.

18. **What is parameter validation in Express.js, and how can you implement it?**

Answer: Parameter validation in Express.js involves validating and sanitizing user input. You can implement it by using validation libraries like `express-validator` or custom validation functions.

19. **What is the purpose of Express.js's response methods like `send`, `json`, and `status`?**

Answer: These response methods are used to send data back to the client. `send` sends plain text or HTML, `json` sends JSON data, and `status` sets the HTTP status code of the response.

20. **Explain the concept of Express.js routing middleware.**

Answer: Express.js routing middleware is used to group routes and apply middleware functions to a specific set of routes. It's beneficial for code organization and applying middleware to multiple routes at once.

21. **What is the `req.query` object in Express.js, and how is it used for processing query parameters?**

Answer: The `req.query` object in Express.js is used to access query parameters in the URL. It allows you to extract and use parameters sent as part of the request.

22. **What is the purpose of the `express.Router` object in Express.js, and how is it used?**

Answer: The `express.Router` object is used to create modular route handlers. It allows you to define routes and middleware for specific route paths and reuse them in different parts of your application.

23. **Explain the difference between the `app.locals` and `res.locals` objects in Express.js.**

Answer: `app.locals` stores application-wide data, while `res.locals` is used to pass data from middleware to route handlers. `res.locals` data is specific to the current request.

24. **What is the purpose of Express.js's `app.param` method, and how is it used to handle route parameters?**

Answer: The `app.param` method in Express.js is used to preprocess and validate route parameters before route handlers are executed. It allows you to centralize parameter handling.

25. **Explain the role of Express.js's `app.route` method, and how is it used to group routes.**

Answer: The `app.route` method is used to create chainable route handlers for a specific route path. It allows you to group routes with a common path and apply middleware to all of them.

26. **What is the purpose of Express.js's `res.redirect` method, and how is it used for URL redirection?**

Answer: `res.redirect` is used to redirect the client to a different URL. It is often used to implement URL redirection or navigate users to a different page.

27. **Explain the purpose of route nesting in Express.js, and how is it used to create structured applications?**

Answer: Route nesting involves organizing routes and middleware into hierarchical structures. It's used to create more structured and organized applications, making it easier to manage and maintain code.

28. **What is Express.js's `req.cookies` object, and how is it used to handle cookies?**

Answer: `req.cookies` is an object used to access cookies sent by the client in the request. It allows you to read and manipulate cookie data.

29. **What is the purpose of Express.js's `app.all` method, and how is it used for handling all HTTP methods?**

Answer: `app.all` is used to define route handlers that respond to all HTTP methods. It's often used for global middleware or request logging.

30. **Explain the concept of Express.js error handling, and how can you handle errors in an Express application?**

Answer: Error handling in Express involves using error-handling middleware to catch and handle errors. You can create custom error-handling middleware and use the `next` function to pass errors to them for processing.

The Coding Wizard

60+ questions on topics

API (Application Programming Interface):

1. **What is an API, and why is it important in software development?**

Answer: An API, or Application Programming Interface, is a set of rules and protocols that allows different software applications to communicate and interact with each other. APIs are essential in software development to enable the exchange of data and functionality between different systems, services, and platforms.

2. **Explain the difference between a RESTful API and a SOAP API.**

Answer: RESTful APIs and SOAP APIs are two different architectural styles for building web services. RESTful APIs use HTTP methods (GET, POST, PUT, DELETE) and are stateless, relying on URLs for resource identification. SOAP APIs use a strict XML-based protocol and can work over various protocols, including HTTP, SMTP, and more. REST is typically simpler and more lightweight, while SOAP is more rigid and feature-rich.

3. **What is an endpoint in the context of API design?**

Answer: An endpoint is a specific URL or URI that an API exposes to allow clients to interact with a particular resource or perform a specific operation. Each endpoint corresponds to a unique function or resource within the API.

4. **What is the purpose of API authentication, and what are some common authentication methods?**

Answer: API authentication ensures that only authorized users or applications can access the API. Common authentication methods include API keys, OAuth, JWT (JSON Web Tokens), and Basic Authentication.

5. ****How can you handle versioning in APIs to ensure backward compatibility?****

Answer: API versioning involves including a version number in the API's URL or request headers. It allows for changes and updates to the API while ensuring that existing clients using older versions remain functional.

6. ****Describe the role of HTTP status codes in API responses and give examples of a few common ones.****

Answer: HTTP status codes indicate the outcome of an API request. Common examples include:

- 200 OK: The request was successful.
- 404 Not Found: The requested resource was not found.
- 500 Internal Server Error: An error occurred on the server.
- 401 Unauthorized: Authentication is required or failed.

7. ****What is rate limiting in API usage, and why is it implemented?****

Answer: Rate limiting is a technique used to restrict the number of API requests a client can make within a specified time period. It is implemented to prevent abuse, protect server resources, and ensure fair usage among clients.

8. ****What is an API key, and how is it used for API access control?****

Answer: An API key is a unique code provided to authorized clients for API access. It is included in API requests and used to identify and authenticate the client making the request.

9. ****What are webhooks, and how do they relate to APIs?****

Answer: Webhooks are a mechanism for a server to notify a client (or another server) about events or changes. They are initiated by the server, allowing real-time updates and communication between systems. Webhooks often use APIs to deliver data or trigger actions.

10. ****Explain the concept of API pagination and why it's necessary for large datasets.****

Answer: API pagination involves breaking down large datasets into smaller, manageable chunks or pages. It is necessary for large datasets to improve performance, reduce data transfer, and make it easier for clients to work with the data incrementally.

11. ****What is the OpenAPI Specification (formerly Swagger), and how can it help in API documentation?****

Answer: The OpenAPI Specification is a standardized way to describe and document RESTful APIs. It includes information about endpoints, request/response formats, and authentication. OpenAPI tools can generate interactive API documentation, making it easier for developers to understand and use the API.

12. ****What are RESTful constraints, and how do they apply to API design?****

Answer: RESTful constraints are a set of architectural principles for building RESTful web services. They include statelessness, client-server architecture, layered system, and more. Adhering to these constraints simplifies API design, making it scalable and easy to work with.

13. ****How can you handle error responses in an API to provide meaningful feedback to clients?****

Answer: API error responses should include a clear error code, message, and possibly additional information. They help clients identify the cause of the error and take appropriate action to address it.

14. **What is CORS (Cross-Origin Resource Sharing), and why is it important in API security?**

Answer: CORS is a security feature that controls how web pages in one domain can request resources from a different domain. It's important in API security to prevent unauthorized cross-origin requests that could compromise the security of the server.

15. **Describe the role of API gateways in managing and securing API traffic.**

Answer: API gateways are used to manage and secure API traffic. They can handle tasks like authentication, rate limiting, request transformation, and routing. API gateways provide a centralized point of control and security for API interactions.

WebSocket:

16. **What is WebSocket, and how does it differ from traditional HTTP communication?**

Answer: WebSocket is a communication protocol that enables bidirectional, real-time communication between a client and a server. It differs from traditional HTTP communication, which is stateless and request-response based. WebSocket allows for continuous, low-latency data exchange.

17. ****How is WebSocket used for real-time applications like chat and gaming?****

Answer: WebSocket is well-suited for real-time applications like chat and gaming because it allows servers to push updates to clients instantly. It enables interactive and dynamic user experiences.

18. ****Explain the WebSocket handshake process for establishing a connection.****

Answer: The WebSocket handshake begins with an HTTP request from the client to the server. If the server supports WebSocket, it responds with a 101 Switching Protocols status code. Once the connection is established, both parties can exchange data in full-duplex mode.

19. ****What is the role of the WebSocket protocol in data framing and communication?****

Answer: The WebSocket protocol defines a frame structure for sending data over a WebSocket connection. Each frame can be a text or binary message, a ping, a pong, or a close frame. This structure enables efficient and ordered communication.

20. ****How do you close a WebSocket connection gracefully?****

Answer: To close a WebSocket connection gracefully, a client or server sends a close frame with a specific status code and an optional reason. This informs the other party that the connection is closing, allowing them to clean up resources.

21. ****What are some benefits of using WebSocket over traditional polling techniques for real-time updates?****

Answer: WebSocket provides several advantages over traditional polling techniques, including reduced latency, lower network and server load, and the ability to push updates immediately to clients, improving the real-time experience.

22. **What is the WebSocket heartbeating mechanism, and how is it used to maintain connection health?**

Answer: WebSocket heartbeating involves sending ping and pong frames between the client and server to ensure the connection is active. If one party does not receive a pong response after sending a ping, it can consider the connection unhealthy and take appropriate action.

23. **Explain the security considerations when using WebSocket, such as preventing security vulnerabilities and DDoS attacks.**

Answer: Security measures for WebSocket include using secure WebSocket connections (WSS), implementing access control, and rate limiting to prevent abuse. Measures are taken to protect against security vulnerabilities and DDoS attacks.

24. **What are some common libraries and frameworks for implementing WebSocket in various programming languages?**

Answer: Common libraries and frameworks for WebSocket include Socket.IO, WebSocket API in HTML5, ws (Node.js), and libraries for other languages like Python, Java, and Ruby.

25. **How can you handle message broadcasting to multiple WebSocket clients connected to a server?**

Answer: To handle message broadcasting, the server can maintain a list of connected clients and loop through them to send messages to all clients or specific groups of clients, ensuring all connected clients receive the message.

SQL (Structured Query Language):

26. **What is SQL, and what is its primary purpose in database management?**

Answer: SQL, or Structured Query Language, is a domain-specific language used to manage and manipulate relational databases. Its primary purpose is to query, insert, update, and delete data in a structured and organized manner.

27. **What is a relational database, and how does it differ from other database models?**

Answer: A relational database organizes data into tables with rows and columns. It differs from other database models, such as NoSQL, by enforcing a strict structure and relationships between data.

28. **What are the basic SQL operations, and how are they used in database management?**

Answer: Basic SQL operations include SELECT (retrieving data), INSERT (adding new data), UPDATE (modifying existing data), and DELETE (removing data). These operations are used to manage and manipulate data in a relational database.

29. **Explain the difference between SQL's DDL and DML statements.**

Answer: DDL (Data Definition Language) statements are used to define and manage the structure of the database, such as creating tables and altering their structure. DML (Data Manipulation Language) statements are used to interact with and manipulate data in the database, including SELECT, INSERT, UPDATE, and DELETE operations.

30. **What is a SQL JOIN operation, and how does it work to combine data from multiple tables?**

Answer: A SQL JOIN operation is used to combine data from two or more related tables based on a specified column. It allows you to retrieve data from multiple tables in a single query, making it possible to establish relationships between them.

31. **How do you create and define a new database table in SQL using DDL statements?**

Answer: You can create a new database table using the `CREATE TABLE` statement. In this statement, you define the table's name, column names, data types, constraints, and any relationships with other tables.

32. **Explain the purpose of SQL constraints, and provide examples of common constraints.**

Answer: SQL constraints are rules applied to columns in a table to enforce data integrity. Common constraints include PRIMARY KEY (uniquely identifies each row), FOREIGN KEY (ensures data integrity between related tables), NOT NULL (requires a column to have a value), and CHECK (validates data based on a condition).

33. **What is SQL normalization, and why is it important in database design?**

Answer: SQL normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing large tables into smaller, related tables and establishing relationships between them. Normalization reduces data anomalies and improves database efficiency.

34. **How can you query a SQL database to retrieve specific data using the SELECT statement?**

Answer: You can query a SQL database using the `SELECT` statement. Specify the columns to retrieve, the table(s) to query, and optional filtering conditions using the `WHERE` clause.

35. **Explain the purpose of SQL indexes, and how do they improve query performance?**

Answer: SQL indexes are data structures that provide fast access to specific rows in a table. They improve query performance by reducing the amount of data that needs to be scanned when searching for specific records. Common types of indexes include B-tree and hash indexes.

36. **What is SQL injection, and how can you prevent it when working with user input in SQL queries?**

Answer: SQL injection is a security vulnerability that occurs when user input is improperly handled in SQL queries, allowing attackers to manipulate the database. To prevent SQL injection, use parameterized queries or prepared statements, and avoid directly embedding user input in SQL queries.

37. **Explain the difference between SQL transactions and their isolation levels.**

Answer: SQL transactions are a set of one or more SQL statements that are executed as a single unit of work. Isolation levels determine how transactions

interact with each other. Common isolation levels include READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE, each offering different levels of data consistency and concurrency control.

38. **How can you update or modify existing data in a SQL database using the UPDATE statement?**

Answer: You can update data in a SQL database using the `UPDATE` statement. Specify the table to update, set the new values for the columns, and use the `WHERE` clause to identify the rows to be updated.

39. **What is the purpose of SQL views, and how do they simplify complex queries?**

Answer: SQL views are virtual tables generated by a query. They simplify complex queries by allowing you to encapsulate them into a reusable, named view. Views make it easier to access and manipulate data without writing the same complex query multiple times.

40. **What is the SQL ORDER BY clause, and how is it used to sort query results?**

Answer: The SQL `ORDER BY` clause is used to sort query results in ascending (ASC) or descending (DESC) order based on one or more columns. It allows you to control the presentation of data in the result set.

NoSQL (Not Only SQL):

41. **What is NoSQL, and how does it differ from traditional SQL databases?**

Answer: NoSQL databases are a type of database management system that are not based on the traditional tabular, relational model. They store and retrieve data in flexible, schema-less formats, making them suitable for unstructured or semi-structured data. NoSQL databases are more horizontally scalable and better suited for certain use cases, such as big data and real-time applications.

42. **Explain the CAP theorem and how it relates to NoSQL databases.**

Answer: The CAP theorem, also known as Brewer's theorem, states that a distributed system can have at most two of the following three properties: Consistency (all nodes see the same data at the same time), Availability (every request receives a response, without guarantee that it contains the most recent version of the data), and Partition tolerance (the system continues to operate despite network partitions). NoSQL databases are often categorized based on their trade-offs within the CAP theorem.

43. **What are some common types of NoSQL databases, and when should they be used?**

Answer: Common types of NoSQL databases include document stores (e.g., MongoDB), key-value stores (e.g., Redis), column-family stores (e.g., Apache Cassandra), and graph databases (e.g., Neo4j). The choice of NoSQL database depends on the specific requirements of the application, such as data structure, scalability, and query patterns.

44. **Explain the concept of eventual consistency in NoSQL databases.**

Answer: Eventual consistency means that, in a distributed NoSQL database, updates made to the data will propagate and converge over time, ensuring that all replicas eventually reach the same state. However, there may be a temporary inconsistency period before this convergence occurs.

45. **How do NoSQL databases handle schema flexibility and dynamic data structures?**

Answer: NoSQL databases are schema-flexible and can store data with dynamic or changing structures. They do not require a fixed schema, making it easy to accommodate evolving data without the need for migrations.

46. **What is sharding in the context of NoSQL databases, and how does it improve scalability?**

Answer: Sharding involves splitting a large dataset into smaller, more manageable pieces called shards, which can be distributed across multiple servers or nodes. Sharding improves scalability by allowing data to be distributed horizontally, enabling the database to handle more data and requests.

47. **Explain the benefits of using NoSQL databases in scenarios where high availability is critical.**

Answer: NoSQL databases are often designed with high availability in mind. They support features like automatic failover, data replication, and distributed architectures, ensuring that the system remains available even in the presence of failures.

48. **What is a document store in NoSQL, and how does it organize data?**

Answer: A document store is a type of NoSQL database that stores data in a document-oriented format, such as JSON or BSON. Each document can have a

different structure, and they are typically organized into collections. Document stores are suited for applications with flexible or semi-structured data.

49. **What is the role of secondary indexes in NoSQL databases, and how do they improve query performance?**

Answer: Secondary indexes allow NoSQL databases to provide efficient and fast access to specific data within a database. They improve query performance by enabling queries on fields other than the primary key, without having to scan the entire dataset.

50. **How does data replication work in NoSQL databases, and what are its benefits?**

Answer: Data replication involves creating copies of data on multiple nodes in a NoSQL database cluster. It provides fault tolerance, data redundancy, and high availability by ensuring that data is still accessible even if some nodes fail.

SQL (Structured Query Language):

51. **What is SQL, and what is its primary purpose in database management?**

Answer: SQL, or Structured Query Language, is a domain-specific language used to manage and manipulate relational databases. Its primary purpose is to query, insert, update, and delete data in a structured and organized manner.

52. **What is a relational database, and how does it differ from other database models?**

Answer: A relational database organizes data into tables with rows and columns. It differs from other database models, such as NoSQL, by enforcing a strict structure and relationships between data.

53. **What are the basic SQL operations, and how are they used in database management?**

- **Answer:** Basic SQL operations include SELECT (retrieving data), INSERT (adding new data), UPDATE (modifying existing data), and DELETE (removing data). These operations are used to manage and manipulate data in a relational database.

54. **Explain the difference between SQL's DDL and DML statements.**

Answer: DDL (Data Definition Language) statements are used to define and manage the structure of the database, such as creating tables and altering their structure. DML (Data Manipulation Language) statements are used to interact with and manipulate data in the database, including SELECT, INSERT, UPDATE, and DELETE operations.

55. **What is a SQL JOIN operation, and how does it work to combine data from multiple tables?**

Answer: A SQL JOIN operation is used to combine data from two or more related tables based on a specified column. It allows you to retrieve data from multiple tables in a single query, making it possible to establish relationships between them.

56. **How do you create and define a new database table in SQL using DDL statements?**

Answer: You can create a new database table using the `CREATE TABLE` statement. In this statement, you define the table's name, column names, data types, constraints, and any relationships with other tables.

57. **Explain the purpose of SQL constraints, and provide examples of common constraints.**

Answer: SQL constraints are rules applied to columns in a table to enforce data integrity. Common constraints include PRIMARY KEY (uniquely identifies each row), FOREIGN KEY (ensures data integrity between related tables), NOT NULL (requires a column to have a value), and CHECK (validates data based on a condition).

58. **What is SQL normalization, and why is it important in database design?**

Answer: SQL normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves dividing large tables into smaller, related tables and establishing relationships between them. Normalization reduces data anomalies and improves database efficiency.

59. **How can you query a SQL database to retrieve specific data using the SELECT statement?**

Answer: You can query a SQL database using the `SELECT` statement. Specify the columns to retrieve, the table(s) to query, and optional filtering conditions using the `WHERE` clause.

60. **Explain the purpose of SQL indexes, and how do they improve query performance?**

Answer: SQL indexes are data structures that provide fast access to specific rows in a table. They improve query performance by reducing the amount of data that needs to be scanned when searching for specific records. Common types of indexes include B-tree and hash indexes.

****NoSQL (Not Only SQL):****

61. **What is NoSQL, and how does it differ from traditional SQL databases?**

Answer: NoSQL databases are a type of database management system that are not based on the traditional tabular, relational model. They store and retrieve data in flexible, schema-less formats, making them suitable for unstructured or semi-structured data. NoSQL databases are more horizontally scalable and better suited for certain use cases, such as big data and real-time applications.

62. **Explain the CAP theorem and how it relates to NoSQL databases.**

Answer: The CAP theorem, also known as Brewer's theorem, states that a distributed system can have at most two of the following three properties: Consistency (all nodes see the same data at the same time), Availability (every request receives a response, without guarantee that it contains the most recent version of the data), and Partition tolerance (the system continues to

operate despite network partitions). NoSQL databases are often categorized based on their trade-offs within the CAP theorem.

63. **What are some common types of NoSQL databases, and when should they be used?**

Answer: Common types of NoSQL databases include document stores (e.g., MongoDB), key-value stores (e.g., Redis), column-family stores (e.g., Apache Cassandra), and graph databases (e.g., Neo4j). The choice of NoSQL database depends on the specific requirements of the application, such as data structure, scalability, and query patterns.

64. **Explain the concept of eventual consistency in NoSQL databases.**

Answer: Eventual consistency means that, in a distributed NoSQL database, updates made to the data will propagate and converge over time, ensuring that all replicas eventually reach the same state. However, there may be a temporary inconsistency period before this convergence occurs.

65. **How do NoSQL databases handle schema flexibility and dynamic data structures?**

Answer: NoSQL databases are schema-flexible and can store data with dynamic or changing structures. They do not require a fixed schema, making it easy to accommodate evolving data without the need for migrations.

66. **What is sharding in the context of NoSQL databases, and how does it improve scalability?**

Answer: Sharding involves splitting a large dataset into smaller, more manageable pieces called shards, which can be distributed across multiple servers or nodes. Sharding improves scalability by allowing data to be

distributed horizontally, enabling the database to handle more data and requests.

67. **Explain the benefits of using NoSQL databases in scenarios where high availability is critical.**

Answer: NoSQL databases are often designed with high availability in mind. They support features like automatic failover, data replication, and distributed architectures, ensuring that the system remains available even in the presence of failures.

68. **What is a document store in NoSQL, and how does it organize data?**

Answer: A document store is a type of NoSQL database that stores data in a document-oriented format, such as JSON or BSON. Each document can have a different structure, and they are typically organized into collections. Document stores are suited for applications with flexible or semi-structured data.

69. **What is the role of secondary indexes in NoSQL databases, and how do they improve query performance?**

Answer: Secondary indexes allow NoSQL databases to provide efficient and fast access to specific data within a database. They improve query performance by enabling queries on fields other than the primary key, without having to scan the entire dataset.

70. **How does data replication work in NoSQL databases, and what are its benefits?**

Answer: Data replication involves creating copies of data on multiple nodes in a NoSQL database cluster. It provides fault tolerance, data redundancy, and high availability by ensuring that data is still accessible even if some nodes fail.

Conclusion

The Journey Towards MERN Mastery

As you come to the end of this eBook on mastering the MERN stack, we hope that you've found the knowledge and skills shared within these pages both enriching and empowering. The MERN stack is a formidable combination of technologies that can be used to create web applications of varying complexity, from simple personal websites to large-scale, data-intensive platforms. You've embarked on a journey to understand the intricacies of HTML, CSS, JavaScript, Bootstrap 5, React.js, Node.js, MongoDB, Mongoose, and Express.js, and we trust that this journey has been a rewarding one.

Reflecting on Your Achievements

Before we part ways, let's take a moment to reflect on what you've achieved:

Strong Foundation in Web Development: You've built a solid foundation in web development by mastering the core web technologies such as HTML, CSS, and JavaScript. These skills enable you to craft stunning, responsive web pages and add interactivity to your applications.

Visually Stunning Web Design with Bootstrap 5: Bootstrap 5 has provided you with the tools to create visually appealing, mobile-first designs with ease. Your websites and applications are now not only functional but also aesthetically pleasing.

Interactive User Interfaces with React.js: The power of React.js has been unlocked, allowing you to create dynamic, interactive user interfaces. You've learned to structure your applications into reusable components and efficiently

manage state and props.

Server-Side Proficiency with Node.js and Express.js: Node.js and Express.js have empowered you to build robust server-side logic. You can now handle client requests, route them to the appropriate endpoints, and seamlessly communicate with databases.

Efficient Data Management with MongoDB and Mongoose: You've dived into the world of NoSQL databases with MongoDB and gained expertise in using Mongoose for efficient data modeling. Your applications can handle and store data efficiently.

Putting It All Together in the MERN Stack: Most importantly, you've learned how to combine these technologies to create a full-stack web application. This is where the true magic of the MERN stack unfolds – you can design, develop, and deploy end-to-end solutions that address real-world problems.

The Road Ahead

Your journey with the MERN stack doesn't end here; it's merely the beginning. The world of web development is constantly evolving, and there's always more to explore and master. Here are a few suggestions for your ongoing journey:

Stay Informed: Keep yourself updated with the latest trends and developments in web development. Technology moves fast, and staying informed is key to your success.

Build Real Projects: The best way to learn and retain your skills is by building real projects. Take on personal or open-source projects to continue applying what you've learned.

Collaborate and Contribute: Join web development communities and collaborate with fellow developers. Contributing to open-source projects is a fantastic way to gain experience and give back to the community.

Never Stop Learning: The web development field is vast. Consider learning other frameworks, libraries, or languages to diversify your skill set.

Teach and Share: Sharing your knowledge with others is a great way to solidify your own understanding. Consider writing blog posts or creating tutorials to help others on their learning journey.

Final Thoughts

We want to congratulate you for your dedication and hard work in mastering the MERN stack. You now possess the skills and knowledge to bring your web development ideas to life. Remember, in the world of web development, there's always more to learn and explore. Embrace every challenge as an opportunity to grow and innovate.

Thank you for choosing this eBook as your guide. We wish you the best of luck on your journey in web development, and may your future projects be as bright as your aspirations.

Happy coding, and continue to create amazing things with the MERN stack!



Python Programming

Learn Python from Beginner to Advanced

Get started

Overview

This course is designed to teach you Python programming from the basics to advanced topics. You will learn about the syntax, data structures, functions, object-oriented programming, file handling, error handling, and additional topics like regular expressions and popular Python libraries. By the end of this course, you will be able to write Python programs and develop applications with ease.

01 Installing



Module 1: Installing Python on Windows, macOS, and Linux

01 Module 1: Installing Python on Windows, macOS, and Linux

Python is a popular programming language known for its simplicity and versatility.

Before you can start coding in Python, you need to install it on your computer. In this module, we will guide you through the installation process on different operating systems: Windows, macOS, and Linux.

Installing Python on Windows

To install Python on Windows, follow these steps:

Download Python: Visit the official Python website at <https://www.python.org/> and click on the "Downloads" tab. Choose the latest stable version suitable for your Windows architecture (32-bit or 64-bit).

Run the Installer: Once the installer is downloaded, run it. During the installation process, make sure to check the box that says "Add Python X.X to PATH". This will add Python to your system's PATH, allowing you to run Python from the command line easily. Click "Install Now" to proceed with the installation.

Verify Installation: After the installation is complete, open the Command Prompt (CMD) and type `python --version` or `python3 --version` to verify the installation. You should see the installed Python version displayed on the screen.

Installing Python on macOS

To install Python on macOS, follow these steps:

Download Python: Visit the official Python website at <https://www.python.org/> and click on the "Downloads" tab. Choose the latest stable version suitable for macOS.

Run the Installer: Once the installer is downloaded, run it and follow the prompts in the installer. Make sure to check the box that says "Add Python X.X to PATH" (X.X represents the version) during the installation process.

Verify Installation: Open the Terminal and type `python3 --version` to check the installed Python version. You should see the version displayed on the screen.

Installing Python on Linux

To install Python on Linux, follow these steps:

Install Python via Package Manager: Python is usually pre-installed on many Linux distributions. If not, you can use your package manager to install Python. For Ubuntu/Debian, use the following command: `sudo apt-get update && sudo aptget install python3`. For CentOS/Fedora, use the following command: `sudo yum install python3`.

Verify Installation: Open the Terminal and type `python3 --version` to check the installed Python version. You should see the version displayed on the screen.

Summary

In this module, we learned how to install Python on different operating systems: Windows, macOS, and Linux. By following the installation steps specific to your operating system, you can set up Python and start coding in this versatile programming language.

Conclusion Module 1 Installing Python on Windows, macOS, and Linux

Module 1: Installing Python on Windows, macOS, and Linux Module 1 focused on the installation process of Python on different operating systems. You learned how to download Python from the official website and set up the development environment. Whether you are using Windows, macOS, or

Linux, you now have the necessary knowledge to install Python and start coding.



Module 2: Setting up the Development Environment

02 Module 2: Setting up the Development Environment

In this module, we will learn how to set up the development environment for Python programming. A well-configured development environment is essential for writing, testing, and running Python code efficiently. We will cover the installation of Python, choosing a text editor or integrated development environment (IDE), and creating virtual environments for project isolation.

Installing Python

Python is an open-source programming language that is widely used for various applications. Before we can start coding in Python, we need to install it on our computer. Here are the steps to install Python on different operating systems:

Windows

Download Python:

Visit the official Python website at <https://www.python.org/>.

Click on the "Downloads" tab.

Choose the latest stable version suitable for your Windows architecture (32-bit or 64bit).

Run the Installer:

Once the installer is downloaded, run it.

During the installation process, check the box that says "Add Python X.X to PATH". This ensures that Python is added to your system's PATH, allowing you to run Python from the command line easily.

Click "Install Now" to proceed with the installation.

Verify Installation:

Open Command Prompt (CMD) and type `python --version` or `python3 --version` to verify the installation. You should see the installed Python version.

macOS

Download Python:

Visit the official Python website at <https://www.python.org/>.

Click on the "Downloads" tab.

Choose the latest stable version suitable for macOS.

Run the Installer:

Once the installer is downloaded, run it and follow the prompts in the installer.

Ensure that the box "Add Python X.X to PATH" (X.X represents the version) is checked.

Verify Installation:

Open Terminal and type `python3 --version` to check the installed Python version.

Linux

Install Python via Package Manager:

Python is usually pre-installed on many Linux distributions. If not, you can use your package manager to install Python.

For Ubuntu/Debian: `sudo apt-get update && sudo apt-get install python3`

For CentOS/Fedora: `sudo yum install python3`

Verify Installation:

Open Terminal and type `python3 --version` to check the installed Python version.

Setting up a Development Environment

Once Python is installed, we need to set up a development environment to write and run Python code. Here are the key components of a Python development environment:

Text Editor or Integrated Development Environment (IDE)

A text editor or an IDE is used to write and edit Python code. There are several options available, and you can choose the one that suits your preferences. Here are some popular choices:

- Text Editors:

Visual Studio Code (VSCode)

Sublime Text Atom

- IDEs:

PyCharm

Spyder

Jupyter Notebooks (primarily for data analysis)

Choose a text editor or IDE that you are comfortable with and install it on your computer.

Virtual Environments (Optional but Recommended)

Virtual environments are used to create isolated Python environments for different projects. They allow you to install project-specific dependencies without interfering with the global Python installation. Here's how you can create a virtual environment using the built-in `venv` module:

Open a terminal or command prompt.

Create a virtual environment:

Windows: `python3 -m venv myenv`

macOS/Linux: `python3 -m venv myenv`

Activate the virtual environment:

Windows: `myenv\Scripts\activate`

macOS/Linux: `source myenv/bin/activate`

Once the virtual environment is activated, any Python packages you install will be isolated within that environment.

Package Manager - pip

`pip` is the default package manager for Python. It allows you to install, upgrade, and manage Python packages and libraries. Here's how you can use `pip` to install Python packages:

To install a package: `pip install package_name`

-
-
- To upgrade a package: `pip install --upgrade package_name`
To uninstall a package: `pip uninstall package_name`

Make sure to run these commands within the activated virtual environment if you are using one.

Version Control (Optional)

Consider using Git for version control of your Python projects. Git allows you to track changes, collaborate with others, and revert to previous versions of your code if needed.

You can install Git from the official website at <https://git-scm.com/>.

Conclusion Module 2 Setting up the Development Environment

Module 2: Setting up the Development Environment In Module 2, you learned about the essential tools for setting up a Python development environment. You explored different text editors and integrated development environments (IDEs) that can be used to write Python code.

Additionally, you discovered the benefits of using virtual environments and package managers like pip. With a properly configured development environment, you are now ready to write and run Python code efficiently.



Module 3: Introduction to Python

03 Module 3: Introduction to Python

Basic Syntax

Indentation

Python uses indentation to define code blocks instead of curly braces or keywords like `end` or `done` in other languages. It is important to use consistent indentation to indicate blocks of code, such as in loops, conditionals, and functions.

```
# Example of indentation

if condition:
    # code block
    statement1
    statement2

else:
    # code block
    statement3
    statement4
```

Comments

Comments are used to add explanatory notes to the code. In Python, single-line comments start with the `#` symbol, while multi-line comments are enclosed between triple quotes (`'''` or `"""`).

```
# This is a single-line comment

'''

This is a
multi-line comment
'''
```

Variables and Data Types

Variables

Variables are used to store values in Python. Unlike other programming languages, Python uses dynamic typing, which means you don't need to declare the data type explicitly. You can assign a value to a variable using the assignment operator

(=).

```
# Example of variable assignment
x = 10
name = "John"
```

Data Types

Python supports various data types, including:

- Integers : whole numbers without decimal points.
- Floats : numbers with decimal points.
- Strings : sequences of characters enclosed in single or double quotes.
- Booleans : either True or False.
- Lists : ordered collections of items.
- Tuples : ordered, immutable collections of items.
- Dictionaries : unordered collections of key-value pairs.

```
# Examples of different data types
age = 25 # integer
height = 1.75 # float
name = "John" # string
is_student = True # boolean

my_list = [1, 2, 3] # list
my_tuple = (4, 5, 6) # tuple
my_dict = {"name": "John", "age": 25} # dictionary
```

Operators and Expressions

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations in Python.

- Addition: `+`
- Subtraction: `-`
- Multiplication : `*`
- Division: `/`
- Modulus: `%` (returns the remainder of a division)
- Exponentiation : `**` (raises a number to a power)

```
# Examples of arithmetic operators
x = 10 + 5 # addition
y = 10 - 5 # subtraction
z = 10 * 5 # multiplication
w = 10 / 5 # division
r = 10 % 3 # modulus
s = 2 ** 3 # exponentiation
```

Comparison Operators

Comparison operators are used to compare values and return a boolean result.

- Equal to: `==`
- Not equal to: `!=`
- Less than: `<`

- Greater than: `>`
- Less than or equal to: `<=`
- Greater than or equal to: `>=`

```
# Examples of comparison operators
x = 10 == 5 # equal to
y = 10 != 5 # not equal to
z = 10 < 5 # less than
w = 10 > 5 # greater than
r = 10 <= 5 # less than or equal to
s = 10 >= 5 # greater than or equal to
```

Logical Operators

Logical operators are used to combine boolean values and perform logical operations.

- and: returns `True` if both operands are `True`.
- or: returns `True` if at least one operand is `True`.
- not: returns the opposite boolean value.

```
# Examples of logical operators
x = True and False # and
y = True or False # or
z = not True # not
```

Expressions

Expressions are combinations of values, variables, and operators that produce a result.

They follow the order of operations, also known as PEMDAS/BODMAS

(Parentheses/Brackets, Exponents/Orders, Multiplication and Division, Addition and Subtraction).

```
# Examples of expressions
x = (10 + 5) * 2 # parentheses
y = 2 ** 3 + 4 * 5 # exponentiation and multiplication
z = 10 / (2 + 3) # division and addition
```

Input and Output

Input

The `input()` function is used to get user input in Python. It prompts the user to enter a value and returns it as a string.

```
# Example of input
name = input("Enter your name: ")
```

Output

The `print()` function is used to display output to the console. It can print strings, variables, and expressions.

```
# Example of output
print("Hello, World!")
print(name)
print(10 + 5)
```

These concepts provide a solid foundation for understanding the basic syntax, variables, data types, operators, expressions, and input/output in Python. Practice using these concepts in your code to become more comfortable with the language.

Conclusion Module 3 Introduction to Python

Module 3: Introduction to Python In Module 1, you were introduced to the fundamentals of Python programming. You learned about the overview of Python, why it is popular, and how to install Python on different operating systems. You also gained knowledge about basic syntax, variables, data types, and input/output operations. This module provided a solid foundation for your journey into Python programming.



Module 4: Basic Syntax of Python

04 Module 4: Basic Syntax of Python

In this module, we will explore the basic syntax of the Python programming language. Understanding the syntax is crucial as it forms the foundation for writing correct and efficient Python code. We will cover topics such as indentation, comments, variables and data types, operators and expressions, and input and output.

Indentation

Python uses indentation to define code blocks instead of curly braces or keywords like "end" or "done" in other programming languages. Consistent indentation is essential to indicate blocks of code, such as loops, conditionals, and functions. It is recommended to use four spaces for indentation, although some developers prefer using tabs.

Example:

```
if condition:  
    # code block 1  
    statement1  
    statement2  
else:  
    # code block 2  
    statement3  
    statement4
```

Comments

Comments are essential for documenting code and making it more readable. In Python, single-line comments start with the `#` symbol, while multi-line comments are enclosed between triple quotes (`'''` or `"""`).

Example:

```
# This is a single-line comment  
  
'''  
This is a multi-line comment.  
It can span multiple lines.  
'''  
  
"""  
This is also a multi-line comment.  
"""
```

Variables and Data Types

Variables are used to store values in Python. Unlike some other programming languages, Python uses dynamic typing, which means you don't need to declare the data type explicitly. The data type of a variable is determined based on the value assigned to it.

Example:

```
# Integer variable  
age = 25  
  
# Float variable  
pi = 3.14  
  
# String variable  
name = "John Doe"  
  
# Boolean variable  
is_student = True
```

Python supports various data types, including integers, floats, strings, booleans, lists, tuples, and dictionaries. Understanding these data types is crucial for manipulating and working with different kinds of data.

Operators and Expressions

Python provides a wide range of operators for performing arithmetic, comparison, logical, and other operations. Arithmetic operators include addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (**). Comparison operators include equal to (==), not equal to (!=), less than (<), greater than (>), less than or equal to (<=), and greater than or equal to (>=). Logical operators include `and`, `or`, and `not` for boolean operations.

Expressions in Python are combinations of values, variables, and operators that produce a result. They follow the order of operations, also known as PEMDAS/BODMAS (Parentheses/Brackets, Exponents/Orders, Multiplication and Division, Addition and Subtraction).

Example:

```
# Arithmetic expressions
result = 5 + 3 * 2 # 11
result = (5 + 3) * 2 # 16

# Comparison expressions
is_equal = 5 == 3 # False
is_greater = 5 > 3 # True

# Logical expressions
is_valid = (5 > 3) and (10 < 20) # True
is_invalid = (5 > 3) and (10 > 20) # False
```

Input and Output

Input and output operations are essential for interacting with users and displaying information. In Python, we can use the `input()` function to get user input, which returns a string. The `print()` function is used to display output to the console.

Example:

```
# Input
name = input("Enter your name: ")
age = int(input("Enter your age: ")) # Convert input to integer

# Output
print("Hello, " + name + "! You are " + str(age) + " years old.")
```

These are the fundamental concepts of the basic syntax in Python.

Understanding and practicing these concepts will enable you to write correct and efficient Python code. In the next module, we will explore data structures in Python, such as lists, tuples, and dictionaries.

Module 4: Basic Syntax of Python Module 4 covered the basic syntax of Python programming. You learned about indentation, comments, variables, and data types.

Understanding these fundamental concepts is crucial for writing clean and readable code. By practicing the concepts taught in this module, you will be able to write Python programs that are easy to understand and maintain.



Module 5: Variables and Data Types

05 Module 5: Variables and Data Types

In this module, we will explore the concept of variables and data types in Python. Variables are used to store and manipulate data, while data types define the nature of the data

being stored. Understanding variables and data types is essential for writing effective and efficient Python programs.

Variables

A variable is a named location in the computer's memory that can hold a value. In Python, variables are created by assigning a value to a name using the assignment operator (`=`). For example:

```
x = 10
```

In the above example, we create a variable named `x` and assign it the value `10`. Variables can store different types of data, such as numbers, strings, or even complex objects.

Variable Naming Rules

When naming variables in Python, there are a few rules to follow:

- Variable names must start with a letter or an underscore (`_`).
- Variable names can only contain letters, numbers, and underscores.
- Variable names are case-sensitive, meaning `x` and `X` are considered different variables.

It is good practice to choose meaningful and descriptive names for variables to improve code readability.

Data Types

Python has several built-in data types that define the nature of the data being stored in variables. The most commonly used data types include:

1. Numeric Types

Python supports three numeric types: integers, floats, and complex numbers.

- Integers: Integers are whole numbers without a fractional part. For example, `10` and `-5` are integers.
- Floats: Floats are numbers with a decimal point or an exponent. For example, `3.14` and `-2.5` are floats.
- Complex Numbers: Complex numbers are numbers with a real and imaginary part. They are written in the form `a + bj`, where `a` is the real part and `b` is the imaginary part.

2. Strings

Strings are sequences of characters enclosed in single quotes (`'`) or double quotes (`""`).

They are used to represent text in Python. For example:

```
name = "John Doe"
```

In the above example, we create a string variable named `name` and assign it the value `"John Doe"`.

3. Booleans

Booleans represent the truth values `True` and `False`. They are used for logical operations and comparisons. For example:

```
is_raining = True
```

In the above example, we create a boolean variable named `is_raining` and assign it the value `True`.

4. Lists

Lists are ordered collections of items enclosed in square brackets (`[]`). They can contain elements of different data types and can be modified (mutable). For example:

```
fruits = ["apple", "banana", "orange"]
```

In the above example, we create a list variable named `fruits` and assign it a list of strings.

5. Tuples

Tuples are similar to lists but are immutable, meaning they cannot be modified once created. They are enclosed in parentheses (`()`). For example:

```
point = (3, 5)
```

In the above example, we create a tuple variable named `point` and assign it a tuple of two integers.

6. Dictionaries

Dictionaries are unordered collections of key-value pairs enclosed in curly braces ({}). Each key-value pair is separated by a colon (:). Dictionaries are used to store and retrieve data based on keys. For example:

```
person = {"name": "John", "age": 25, "city": "New York"}
```

In the above example, we create a dictionary variable named `person` and assign it key-value pairs representing a person's name, age, and city.

Type Conversion

Python provides built-in functions to convert data from one type to another. This is known as type conversion or type casting. Some commonly used type conversion functions include:

- `int()`: Converts a value to an integer.
- `float()`: Converts a value to a float. `str()`:
- Converts a value to a string. `list()`: Converts
- a value to a list. `tuple()`: Converts a value to
- a tuple. `dict()`: Converts a value to a
- dictionary.

Type conversion is useful when performing operations that require data of a specific type or when combining data of different types.

Summary

In this module, we explored the concept of variables and data types in Python. Variables are named locations in memory that store data, and data types define the nature of the data being stored. Python supports various data types, including numeric types, strings, booleans, lists, tuples, and dictionaries. Understanding variables and data types is crucial for writing effective and efficient Python programs.

Conclusion Module 5 Variables and Data Types

Module 5: Variables and Data Types In Module 5, you delved deeper into variables and data types in Python. You learned about common data types such as integers, floats, strings, booleans, lists, tuples, and dictionaries. Understanding how to work with different data types is essential for manipulating and storing data effectively. With the knowledge gained in this module, you can now handle various types of data in your Python programs.



Module 6: Functions and Modules

06 Module 6: Functions and Modules

In this module, we will explore the concepts of functions and modules in Python. Functions are blocks of reusable code that perform specific tasks, while modules are files containing Python definitions, functions, and statements that help organize code into reusable components.

Functions

Defining Functions

Functions in Python are defined using the `def` keyword followed by the function name and parentheses. They can take parameters (arguments) and perform a specific task.

```
def greet(name):  
    print("Hello, " + name + "!")  
  
def add_numbers(a, b):  
    return a + b
```

Function Arguments and Parameters

Functions can take parameters to accept input data. There are two types of function arguments: positional arguments and keyword arguments.

Positional Arguments: Arguments passed to a function based on their position.

```
def greet(name, age):  
    print("Hello, " + name + "! You are " + str(age) + " years old.")  
  
greet("Alice", 25)
```

Keyword Arguments: Arguments identified by their parameter names.

```
def greet(name, age):  
    print("Hello, " + name + "! You are " + str(age) + " years old.")  
  
greet(age=25, name="Alice")
```

Lambda Functions

Lambda functions, also known as anonymous functions, are defined using the `lambda` keyword. They are typically used for short, simple operations.

```
add_numbers = lambda a, b: a + b
```

Modules and Packages

Modules

Modules are Python files containing Python definitions, functions, and statements.

They are used to organize code into files for better maintainability.

Creating Modules

To create a module, write functions or code in a Python file and import it into another Python script.

```
# math_operations.py
def add(a, b):
    return a + b
```

```
def subtract(a, b):
    return a - b
```

```
# main.py
import math_operations

result = math_operations.add(5, 3)
print(result)
```

Packages

Packages are collections of modules organized into directories. They help in better structuring large codebases.

```
my_package/  
└── __init__.py  
└── module1.py  
└── module2.py
```

Importing from Packages

To import modules or specific functions/classes from packages, use the `import` statement.

```
from my_package import module1  
  
result = module1.add(5, 3)  
print(result)
```

Summary

In this module, we explored the concepts of functions and modules in Python. Functions are blocks of reusable code that perform specific tasks, while modules are files containing Python definitions, functions, and statements that help organize code into reusable components.

We learned how to define functions, use different types of function arguments, create modules, and import them into other Python scripts. Additionally, we explored the concept of lambda functions and how they can be used for simple operations.

Understanding functions and modules is essential for writing modular and reusable code in Python. Practicing the concepts covered in this module will enhance your ability to create and use functions, as well as organize your code into modules and packages.

Module 7: Functions and Modules Module 7 introduced you to the concepts of functions and modules in Python. You learned how to define functions, pass arguments, and use lambda functions for simple operations. Additionally, you explored the creation and usage of modules and packages, which allow you to organize and reuse your code effectively. With the knowledge gained in this module, you can write modular and reusable Python code.



Module 7: Control Flow

07 Module 7: Control Flow

Control flow refers to the order in which statements are executed in a program. In Python, control flow is managed through various control structures, such as conditional

statements and loops, which allow you to make decisions and repeat actions based on certain conditions. Understanding control flow is essential for writing efficient and flexible programs.

Conditional Statements

Conditional statements allow you to execute different blocks of code based on certain conditions. In Python, the most commonly used conditional statements are the `if`, `elif`, and `else` statements.

if Statement

The `if` statement is used to execute a block of code if a condition is true. It follows the syntax:

```
if condition:  
    # code to be executed if condition is true
```

For example, let's say we want to check if a number is positive:

```
num = 10  
  
if num > 0:  
    print("The number is positive")
```

In this example, the code inside the `if` block will only be executed if the condition is true.

if-else Statement

The `if-else` statement allows you to execute one block of code if a condition is true, and another block of code if the condition is false. It follows the syntax:

```
if condition:  
    # code to be executed if condition is true
```

```
else:  
    # code to be executed if condition is false
```

For example, let's say we want to check if a number is positive or negative:

```
num = -5  
  
if num > 0:  
    print("The number is positive")  
else:  
    print("The number is negative")
```

In this example, if the condition `num > 0` is true, the code inside the `if` block will be executed. Otherwise, the code inside the `else` block will be executed.

if-elif-else Statement

The `if-elif-else` statement allows you to check multiple conditions and execute different blocks of code based on the first condition that is true. It follows the syntax:

```
if condition1:  
    # code to be executed if condition1 is true  
elif condition2:  
    # code to be executed if condition2 is true  
else:  
    # code to be executed if all conditions are false
```

For example, let's say we want to check if a number is positive, negative, or zero:

```
num = 0
```

```
if num > 0:  
    print("The number is positive")  
elif num < 0:  
    print("The number is negative")  
else:  
    print("The number is zero")
```

In this example, if the condition `num > 0` is true, the code inside the first `if` block will be executed. If that condition is false and the condition `num < 0` is true, the code inside the `elif` block will be executed. If both conditions are false, the code inside the `else` block will be executed.

Loops

Loops allow you to repeat a block of code multiple times. In Python, there are two types of loops: `for` loops and `while` loops.

for Loop

The `for` loop is used to iterate over a sequence (such as a list, tuple, string, or range) or an iterable object. It follows the syntax:

```
for item in sequence:  
    # code to be executed for each item in the sequence
```

For example, let's say we want to print each element in a list:

```
fruits = ["apple", "banana", "orange"]
```

```
for fruit in fruits:  
    print(fruit)
```

In this example, the code inside the `for` loop will be executed for each element in the `fruits` list.

while Loop

The `while` loop is used to repeatedly execute a block of code as long as a condition is true. It follows the syntax:

```
while condition:  
    # code to be executed as long as the condition is true
```

For example, let's say we want to count from 1 to 5 using a `while` loop:

```
count = 1  
  
while count <= 5:  
    print(count)  
    count += 1
```

In this example, the code inside the `while` loop will be executed as long as the condition `count <= 5` is true. The `count` variable is incremented by 1 in each iteration to avoid an infinite loop.

break and continue Statements

The `break` statement is used to terminate a loop prematurely when a certain condition is met. It allows you to exit the loop and continue with the next statement after the loop. For example:

```
fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    if fruit == "banana":
        break
    print(fruit)
```

In this example, the loop will terminate when the `fruit` variable is equal to "banana", and the code will continue with the next statement after the loop.

The `continue` statement is used to skip the current iteration of a loop and continue with the next iteration. For example:

```
fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    if fruit == "banana":
        continue
    print(fruit)
```

In this example, when the `fruit` variable is equal to "banana", the current iteration will be skipped, and the loop will continue with the next iteration.

These control flow structures allow you to make decisions, iterate through sequences, and control the flow of execution in your Python programs. Experimenting with different conditions and loops will help solidify your understanding of these concepts.

[Conclusion](#) [Module 7 Control Flow](#)

Module 7: Control Flow Module 6 focused on control flow in

Python. You learned about conditional statements such as if, elif, and else, as well as loops like for and while. Additionally, you explored the concepts of break and continue statements. Understanding control flow is crucial for making decisions and iterating through sequences in your programs.

By mastering control flow, you can create more dynamic and interactive Python applications.



Module 8: File Handling and Error Handling

08 Module 8: File Handling and Error Handling

In this module, we will explore the concepts of file handling and error handling in Python. File handling allows us to work with files, such as reading from and writing to them, while error handling helps us manage and handle potential errors that may occur during program execution.

File Handling

Opening and Closing Files

To work with files in Python, we need to open them using the `open()` function. This function takes the filename and the mode in which we want to open the file as parameters. The mode can be 'r' for reading, 'w' for writing, 'a' for appending, and more.

```
# Opening a file for reading
file = open('filename.txt', 'r')
```

After we are done working with a file, it is important to close it using the `close()` method to free up system resources .

```
# Closing the file
file.close()
```

Reading Text Files

To read the contents of a text file, we can use the `read()` or `readline()` methods. The `read()` method reads the entire file, while the `readline()` method reads one line at a time.

```
# Reading from a file
file = open('filename.txt', 'r')
content = file.read()
print(content)
file.close()
```

Writing Text Files

To write to a text file, we can use the `write()` method. This method allows us to write content to the file. If the file does not exist, it will be created. If it already exists, the existing content will be overwritten.

```
# Writing to a file
file = open('filename.txt', 'w')
file.write('Hello, world!')
file.close()
```

Working with File Paths

When working with files, it is important to handle file paths properly. We can use the `os.path` module or the `pathlib` module to handle file paths. These modules provide functions and classes to manipulate file paths.

```
import os.path

# Working with file paths
path = os.path.join('folder', 'filename.txt')
print(path)
```

Error Handling

Handling Exceptions

In Python, we can handle exceptions using try-except blocks. A try block contains the code that might raise an exception, while an except block handles the exception if it occurs.

```
try:  
    # Code that might raise an exception  
    result = 10 / 0  
except ZeroDivisionError:  
    # Handling the ZeroDivisionError exception  
    print("Cannot divide by zero")
```

Handling Specific Exceptions

We can handle specific exceptions by specifying the type of exception in the except block. This allows us to handle different errors differently.

```
try:  
    # Code that might raise an exception  
    file = open('filename.txt', 'r')  
except FileNotFoundError:  
    # Handling the FileNotFoundError exception  
    print("File not found")
```

Using Finally Block

The finally block is used to specify code that should be executed regardless of whether an exception occurs or not. This block is useful for cleaning up resources or performing necessary actions.

```
try:  
    # Code that might raise an exception  
    file = open('filename.txt', 'r')  
except FileNotFoundError:  
    # Handling the FileNotFoundError exception  
    print("File not found")  
finally:  
    # Code that should be executed regardless of exceptions  
    file.close()
```

Summary

In this module, we learned about file handling and error handling in Python. File handling allows us to work with files, such as reading from and writing to them. Error handling helps us manage and handle potential errors that may occur during program execution. By understanding these concepts, we can effectively work with files and handle exceptions in our Python programs.

Conclusion Module 8 File Handling and Error Handling

Module 8: File Handling and Error Handling In Module 8, you learned about file handling and error handling in Python. You discovered how to read and write files, work with file paths, and parse CSV and JSON data. Additionally, you explored error handling techniques using try-except blocks and handling specific exceptions. Understanding file handling and error handling is crucial for working with external data and ensuring the robustness of your programs.



Practical Exercises

Let's put your knowledge into practice

09 Practical Exercises

In this lesson, we'll put theory into practice through hands-on activities. Click on the items below to check each exercise and develop practical skills that will help you succeed in the subject.

Python Overview

Write a short paragraph explaining what Python is and why it is popular in the programming community.

Installing Python on Windows

Write step-by-step instructions on how to download and install Python on a Windows operating system.

Installing Python on macOS

Write step-by-step instructions on how to download and install Python on a macOS operating system.

Installing Python on Linux

Write step-by-step instructions on how to install Python on a Linux

operating system using the package manager.

Setting up a Text Editor

Recommend and provide instructions for setting up a text editor for Python programming .

Creating a Virtual Environment

Explain the concept of a virtual environment and provide instructions for creating and activating a virtual environment using venv.

Indentation in Python

Explain the importance of indentation in Python and provide examples of correct and incorrect indentation .

Variable Declaration and Data Types

Write a Python program that declares variables of different data types (integer , float , string) and prints their values.



Wrap-up

Let's review what we have just seen so far

10 | Wrap-up

- ✓ **Module 1: Introduction to Python** In Module 1, you were introduced to the fundamentals of Python programming. You learned about the overview of Python, why it is popular, and how to install Python on different operating systems. You also gained knowledge about basic syntax, variables, data types, and input/output operations. This module provided a solid foundation for your journey into Python programming.
- ✓ **Module 2: Installing Python on Windows, macOS, and Linux** Module 2 focused on the installation process of Python on different operating systems. You learned how to download Python from the official website and set up the development environment. Whether you are using Windows, macOS, or Linux, you now have the necessary knowledge to install Python and start coding.

- ✓ **Module 3: Setting up the Development Environment** In Module 3, you learned about the essential tools for setting up a Python development environment. You explored different text editors and integrated development environments (IDEs) that can be used to write Python code. Additionally, you discovered the benefits of using virtual environments and package managers like pip. With a properly configured development environment, you are now ready to write and run Python code efficiently.
- ✓ **Module 4: Basic Syntax of Python** Module 4 covered the basic syntax of Python programming. You learned about indentation, comments, variables, and data types. Understanding these fundamental concepts is crucial for writing clean and readable code. By practicing the concepts taught in this module, you will be able to write Python programs that are easy to understand and maintain.
- ✓ **Module 5: Variables and Data Types** In Module 5, you delved deeper into variables and data types in Python. You learned about common data types such as integers, floats, strings, booleans, lists, tuples, and dictionaries. Understanding how to work with different data types is essential for manipulating and storing data effectively. With the knowledge gained in this module, you can now handle various types of data in your Python programs.
- ✓ **Module 6: Control Flow** Module 6 focused on control flow in Python. You learned about conditional statements such as if, elif, and else, as well as loops like for and while. Additionally, you explored the concepts of break and continue statements. Understanding control flow is crucial for making decisions and iterating through sequences in your programs. By mastering control flow, you can create more dynamic and interactive Python applications.
- ✓ **Module 7: Functions and Modules** Module 7 introduced you to the concepts of functions and modules in Python. You learned how to define functions, pass arguments, and use lambda functions for simple operations. Additionally, you

explored the creation and usage of modules and packages, which allow you to organize and reuse your code effectively. With the knowledge gained in this module, you can write modular and reusable Python code.

- ✓ **Module 8: File Handling and Error Handling** In Module 8, you learned about file handling and error handling in Python. You discovered how to read and write files, work with file paths, and parse CSV and JSON data. Additionally, you explored error handling techniques using try-except blocks and handling specific exceptions. Understanding file handling and error handling is crucial for working with external data and ensuring the robustness of your programs.



Quiz

Check your knowledge answering some questions

Question 1/17

What is Python ?

- A programming language
 - A type of snake
 - A computer brand
-

Question 2/17

Which of the following is true about Python ?

- It is difficult to learn
 - It is popular for web development
 - It is only used for scientific computing
-

Question 3/17

How do you install Python on Windows ?

- Download the installer from the official Python website and run it
 - Use the package manager to install Python
 - Python is pre-installed on Windows
-

Question 4/17

What is the purpose of the 'print' function in Python ?

- To read user input
 - To display output to the console
 - To perform mathematical calculations
-

Question 5/17

Which of the following is a valid data type in Python?

- Integer
 - Float
 - String
-

Question 6/17

What is the purpose of indentation in Python?

- To define code blocks
 - To perform mathematical calculations
 - To import modules
-

Question 7/17

What is the result of $5 + 3 * 2$?

- 16
- 11
- 13

Question 8/17

What is the result of 'Hello' + 'World'?

- HelloWorld
 - Hello World
 - WorldHello
-

Question 9/17

What is the purpose of a for loop in Python?

- To execute a block of code repeatedly
 - To make decisions based on conditions
 - To define a function
-

Question 10/17

What is the purpose of a function in Python?

- To store data
 - To perform a specific task
 - To import modules
-

Question 11/17

How do you import a module in Python?

- import module_name
 - from module_name import function_name
 - module_name.function_name
-

Question 12/17

What is the purpose of file handling in Python?

- To read and write files
 - To perform mathematical calculations
 - To import modules
-

Question 13/17

What is the purpose of error handling in Python?

- To handle exceptions and errors
 - To perform mathematical calculations
 - To import modules
-

Question 14/17

What is the purpose of regular expressions in Python?

- To perform pattern matching
- To perform mathematical calculations
- To import modules

Question 15/17

What is the purpose of the NumPy library in Python?

- For numerical computing
 - For web development
 - For file handling
-

Question 16/17

What is the purpose of the Pandas library in Python?

- For data manipulation and analysis
 - For web development
 - For file handling
-

Question 17/17

What is the purpose of the Matplotlib library in Python?

- For data visualization
 - For web development
 - For file handling
-

Submit

Question 1: Calculate the area of a rectangle

python

Copy code

```
def rectangle_area(length, width):  
    area = length * width  
    return area
```

Example usage

```
length = float(input("Enter the length of the rectangle: "))  
width = float(input("Enter the width of the rectangle: "))  
result = rectangle_area(length, width)  
print("The area of the rectangle is:", result)
```

Question 2: Check if a number is even or odd

python

Copy code

```
def check_even_odd(number):  
    if number % 2 == 0:  
        return "Even"  
    else:  
        return "Odd"
```

Example usage

```
num = int(input("Enter a number: "))
```

```
result = check_even_odd(num)
print(f"The number is {result}.")
```

Question 3: Find the maximum of three numbers

python

Copy code

```
def find_maximum(a, b, c):
    return max(a, b, c)
```

Example usage

```
num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

num3 = float(input("Enter the third number: "))

result = find_maximum(num1, num2, num3)

print("The maximum number is:", result)
```

Question 4: Swap the values of two variables

python

Copy code

```
def swap_variables(a, b):
    a, b = b, a
    return a, b
```

Example usage

```
var1 = input("Enter the first variable: ")
```

```
var2 = input("Enter the second variable: ")  
result1, result2 = swap_variables(var1, var2)  
print("After swapping, variable 1:", result1)  
print("After swapping, variable 2:", result2)
```

Question 5: Check if a year is a leap year

python

Copy code

```
def is_leap_year(year):  
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):  
        return True  
    else:  
        return False
```

Example usage

```
year = int(input("Enter a year: "))  
if is_leap_year(year):  
    print(f"{year} is a leap year.")  
else:  
    print(f"{year} is not a leap year.")
```

Question 6: Convert Celsius to Fahrenheit

python

Copy code

```
def celsius_to_fahrenheit(celsius):
    fahrenheit = (celsius * 9/5) + 32
    return fahrenheit

# Example usage
temp_celsius = float(input("Enter the temperature in Celsius: "))
temp_fahrenheit = celsius_to_fahrenheit(temp_celsius)
print(f"The temperature in Fahrenheit is: {temp_fahrenheit:.2f}")
```

Question 7: Basic calculator program

python

Copy code

```
def calculator(num1, num2, operator):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '/':
        return num1 / num2
    else:
        return "Invalid operator"
```

Example usage

```
number1 = float(input("Enter the first number: "))
```

```
number2 = float(input("Enter the second number: "))

op = input("Enter the operator (+, -, *, /): ")

result = calculator(number1, number2, op)

print(f"The result is: {result}")
```

Question 8: Print the first 10 natural numbers

python

Copy code

```
for i in range(1, 11):
```

```
    print(i)
```

Question 9: Find the factorial of a number

python

Copy code

```
def factorial(n):
```

```
    if n == 0 or n == 1:
```

```
        return 1
```

```
    else:
```

```
        return n * factorial(n-1)
```

```
# Example usage
```

```
num = int(input("Enter a number: "))

result = factorial(num)

print(f"The factorial of {num} is: {result}")
```

Question 10: Guess-the-number game

python

[Copy code](#)

```
import random
```

```
def guess_the_number():
```

```
    secret_number = random.randint(1, 100)
```

```
    attempts = 0
```

```
    while True:
```

```
        guess = int(input("Guess the number (between 1 and 100): "))
```

```
        attempts += 1
```

```
        if guess == secret_number:
```

```
            print(f"Congratulations! You guessed the number in {attempts} attempts.")
```

```
            break
```

```
        elif guess < secret_number:
```

```
            print("Too low. Try again.")
```

```
        else:
```

```
            print("Too high. Try again.")
```

```
# Example usage
```

```
guess_the_number()
```

Question 11: Check if a string is a palindrome

python

Copy code

```
def is_palindrome(s):
```

```
    s = s.lower()
```

```
    return s == s[::-1]
```

```
# Example usage
```

```
string = input("Enter a string: ")
```

```
if is_palindrome(string):
```

```
    print("The string is a palindrome.")
```

```
else:
```

```
    print("The string is not a palindrome.")
```

Question 12: Find the length of a list

python

Copy code

```
def list_length(lst):
```

```
    return len(lst)
```

```
# Example usage
```

```
my_list = [1, 2, 3, 4, 5]
```

```
result = list_length(my_list)
```

```
print(f"The length of the list is: {result}")
```

Question 13: Count the occurrences of a character in a string

python

Copy code

```
def count_char_occurrences(s, char):  
    return s.count(char)
```

Example usage

```
string = input("Enter a string: ")  
character = input("Enter a character: ")  
result = count_char_occurrences(string, character)  
print(f"The character '{character}' appears {result} times in the string.")
```

Question 14: Reverse a list

python

Copy code

```
def reverse_list(lst):  
    return lst[::-1]
```

Example usage

```
my_list = [1, 2, 3, 4, 5]  
result = reverse_list(my_list)  
print("Reversed list:", result)
```

Question 15: Concatenate two lists

python

Copy code

```
def concatenate_lists(list1, list2):  
    return list1 + list2
```

Example usage

```
list_a = [1, 2, 3]  
list_b = [4, 5, 6]  
result = concatenate_lists(list_a, list_b)  
print("Concatenated list:", result)
```

Question 16: Remove duplicates from a list

python

Copy code

```
def remove_duplicates(lst):  
    return list(set(lst))
```

Example usage

```
my_list = [1, 2, 2, 3, 4, 4, 5]  
result = remove_duplicates(my_list)  
print("List with duplicates removed:", result)
```

Question 17: Find the smallest element in a list

python

Copy code

```
def find_smallest_element(lst):  
    return min(lst)
```

Example usage

```
my_list = [5, 2, 8, 1, 7]  
result = find_smallest_element(my_list)  
print("The smallest element in the list is:", result)
```

Question 18: Check if a number is prime

python

Copy code

```
def is_prime(num):  
    if num < 2:  
        return False  
    for i in range(2, int(num**0.5) + 1):  
        if num % i == 0:  
            return False  
    return True
```

Example usage

```
number = int(input("Enter a number: "))
```

```
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

Question 19: Generate the Fibonacci sequence up to n terms

python

Copy code

```
def fibonacci_sequence(n):
    sequence = [0, 1]
    while len(sequence) < n:
        sequence.append(sequence[-1] + sequence[-2])
    return sequence
```

Example usage

```
terms = int(input("Enter the number of Fibonacci terms to generate: "))
result = fibonacci_sequence(terms)
print("Fibonacci sequence:", result)
```

Question 20: Check if a number is a perfect square

python

Copy code

```
def is_perfect_square(num):
    return num > 0 and (int(num**0.5))**2 == num
```

```
# Example usage  
number = int(input("Enter a number: "))  
if is_perfect_square(number):  
    print(f"{number} is a perfect square.")  
else:  
    print(f"{number} is not a perfect square.")
```

Question 21: Find the common elements between two lists

python

Copy code

```
def common_elements(list1, list2):  
    return list(set(list1) & set(list2))
```

Example usage

```
list_a = [1, 2, 3, 4]  
list_b = [3, 4, 5, 6]  
result = common_elements(list_a, list_b)  
print("Common elements:", result)
```

Question 22: Count the number of vowels in a string

python

Copy code

```
def count_vowels(s):
```

```
vowels = "aeiouAEIOU"  
return sum(1 for char in s if char in vowels)
```

```
# Example usage  
string = input("Enter a string: ")  
result = count_vowels(string)  
print(f"The number of vowels in the string is: {result}")
```

Question 23: Calculate the LCM of two numbers

python

Copy code

```
def calculate_lcm(x, y):  
    from math import gcd  
    return abs(x * y) // gcd(x, y)
```

```
# Example usage  
num1 = int(input("Enter the first number: "))  
num2 = int(input("Enter the second number: "))  
result = calculate_lcm(num1, num2)  
print(f"The LCM of {num1} and {num2} is: {result}")
```

Question 24: Extract the file extension from a filename

python

Copy code

```
def extract_file_extension(filename):
```

```
return filename.split('.')[−1]
```

Example usage

```
file_name = input("Enter a filename: ")  
result = extract_file_extension(file_name)  
print(f"The file extension is: {result}")
```

Question 25: Find and print the largest element in a list

python

Copy code

```
def find_largest_element(lst):  
    return max(lst)
```

Example usage

```
my_list = [10, 5, 8, 15, 3]  
result = find_largest_element(my_list)  
print("The largest element in the list is:", result)
```

Question 26: Check if a string has balanced parentheses

python

Copy code

```
def balanced_parentheses(s):  
    stack = []  
    opening = set("({[")  
    mapping = {')': '(', '}': '{', ']': '['}
```

```
for char in s:  
    if char in opening:  
        stack.append(char)  
    else:  
        if not stack or stack.pop() != mapping[char]:  
            return False  
  
    return not stack
```

```
# Example usage  
expression = input("Enter a string with parentheses: ")  
if balanced_parentheses(expression):  
    print("The parentheses are balanced.")  
else:  
    print("The parentheses are not balanced.")
```

Question 27: Calculate the sum of digits in a number

```
python  
Copy code  
def sum_of_digits(number):  
    return sum(int(digit) for digit in str(abs(number)))
```

```
# Example usage  
num = int(input("Enter a number: "))  
result = sum_of_digits(num)
```

```
print(f"The sum of the digits in {num} is: {result}")
```

Question 28: Find and replace a word in a text file

python

Copy code

```
def find_replace_in_file(file_path, old_word, new_word):
```

```
    try:
```

```
        with open(file_path, 'r') as file:
```

```
            content = file.read()
```

```
            new_content = content.replace(old_word, new_word)
```

```
            with open(file_path, 'w') as file:
```

```
                file.write(new_content)
```

```
    print(f"Word '{old_word}' replaced with '{new_word}' in the file.")
```

```
except FileNotFoundError:
```

```
    print("File not found.")
```

```
# Example usage
```

```
file_path = input("Enter the path to the text file: ")
```

```
old_word = input("Enter the word to replace: ")
```

```
new_word = input("Enter the new word: ")
```

```
find_replace_in_file(file_path, old_word, new_word)
```

Question 29: Find the intersection of two lists

python

Copy code

```
def find_intersection(list1, list2):  
    return list(set(list1) & set(list2))
```

Example usage

```
list_a = [1, 2, 3, 4]  
list_b = [3, 4, 5, 6]  
result = find_intersection(list_a, list_b)  
print("Intersection of the two lists:", result)
```

Question 30: Check if a number is an Armstrong number

python

Copy code

```
def is_armstrong_number(num):  
    order = len(str(num))  
    sum_of_digits = sum(int(digit) ** order for digit in str(num))  
    return num == sum_of_digits
```

Example usage

```
number = int(input("Enter a number: "))  
if is_armstrong_number(number):  
    print(f"{number} is an Armstrong number.")
```

```
else:
```

```
    print(f"{number} is not an Armstrong number.")
```

Feel free to reach out if you have any questions or if you'd like further clarification on any of the solutions!

Question 31: Simple Contact Management System

```
python
```

[Copy code](#)

```
class Contact:
```

```
    def __init__(self, name, phone, email):
        self.name = name
        self.phone = phone
        self.email = email
```

```
class ContactManager:
```

```
    def __init__(self):
        self.contacts = []
```

```
    def add_contact(self, contact):
        self.contacts.append(contact)
```

```
    def view_contacts(self):
```

```
        for contact in self.contacts:
```

```
            print(f"Name: {contact.name}, Phone: {contact.phone}, Email: {contact.email}")
```

```
# Example usage  
contact1 = Contact("John Doe", "123-456-7890", "john@example.com")  
contact2 = Contact("Jane Doe", "987-654-3210", "jane@example.com")
```

```
contact_manager = ContactManager()  
contact_manager.add_contact(contact1)  
contact_manager.add_contact(contact2)  
contact_manager.view_contacts()
```

Question 32: Organize Files by Extensions

python

Copy code

```
import os  
import shutil
```

```
def organize_files(directory):  
    for filename in os.listdir(directory):  
        if os.path.isfile(os.path.join(directory, filename)):  
            extension = filename.split('.')[ -1]  
            extension_folder = os.path.join(directory, extension)  
  
            if not os.path.exists(extension_folder):  
                os.makedirs(extension_folder)  
  
            shutil.move(os.path.join(directory, filename), os.path.join(extension_folder,  
filename))
```

```
# Example usage  
organize_files('/path/to/directory')
```

Question 33: Basic Tic-Tac-Toe Game

python

Copy code

```
def print_board(board):  
    for row in board:  
        print(" ".join(row))  
  
def check_winner(board, player):  
    # Check rows, columns, and diagonals for a win  
    for i in range(3):  
        if all(cell == player for cell in board[i]) or all(board[j][i] == player for j in range(3)):  
            return True  
        if all(board[i][j] == player for j in range(3)) or all(board[i][2 - i] == player for i in range(3)):  
            return True  
    return False  
  
def play_tic_tac_toe():  
    board = [[" " for _ in range(3)] for _ in range(3)]  
    current_player = "X"  
  
    for _ in range(9):  
        print_board(board)
```

```
row = int(input(f"Player {current_player}, enter row (0-2): "))

col = int(input(f"Player {current_player}, enter column (0-2): "))

if board[row][col] == " ":
    board[row][col] = current_player

if check_winner(board, current_player):
    print(f"Player {current_player} wins!")
    break

current_player = "O" if current_player == "X" else "X"
else:
    print("Cell already taken. Try again.")

# Example usage
play_tic_tac_toe()
```

Feel free to ask for more code snippets or explanations for other questions!

User

continue 20 answers more

ChatGPT

Certainly! Here are additional code snippets for the next set of questions:

Question 34: Calculate Average Word Length in a Sentence

python

Copy code

```
def average_word_length(sentence):
    words = sentence.split()
    total_length = sum(len(word) for word in words)
    return total_length / len(words) if len(words) > 0 else 0
```

Example usage

```
sentence = input("Enter a sentence: ")
result = average_word_length(sentence)
print(f"The average word length is: {result}")
```

Question 35: Extract Metadata from Audio Files

python

Copy code

```
from pydub import AudioSegment
```

```
def extract_audio_metadata(file_path):
```

```
    audio = AudioSegment.from_file(file_path)
    metadata = {
        "duration": len(audio) / 1000, # Convert to seconds
        "channels": audio.channels,
        "sample_width": audio.sample_width,
        "frame_rate": audio.frame_rate,
    }
    return metadata
```

```
# Example usage  
file_path = "/path/to/audio/file.mp3"  
result = extract_audio_metadata(file_path)  
print("Audio Metadata:", result)
```

Question 36: Download and Parse an RSS Feed

python

Copy code

```
import feedparser
```

```
def download_and_parse_rss(feed_url):  
    feed = feedparser.parse(feed_url)  
    for entry in feed.entries:  
        print(f"Title: {entry.title}, Link: {entry.link}")
```

Example usage

```
feed_url = "https://example.com/rss_feed.xml"  
download_and_parse_rss(feed_url)
```

Question 37: Convert Decimal to Binary

python

Copy code

```
def decimal_to_binary(decimal):  
    return bin(decimal).replace("0b", "")
```

```
# Example usage

decimal_number = int(input("Enter a decimal number: "))

binary_result = decimal_to_binary(decimal_number)

print(f"The binary representation is: {binary_result}")
```

Question 38: Matrix Multiplication

python

Copy code

```
import numpy as np
```

```
def matrix_multiplication(matrix1, matrix2):

    result = np.dot(matrix1, matrix2)

    return result

# Example usage

matrix_a = np.array([[1, 2], [3, 4]])

matrix_b = np.array([[5, 6], [7, 8]])

result_matrix = matrix_multiplication(matrix_a, matrix_b)

print("Result of Matrix Multiplication:")

print(result_matrix)
```

Question 39: Basic Banking System

python

Copy code

```
class Account:
```

```
def __init__(self, account_number, balance):
    self.account_number = account_number
    self.balance = balance

class BankingSystem:
    def __init__(self):
        self.accounts = []

    def create_account(self, account_number, initial_balance):
        account = Account(account_number, initial_balance)
        self.accounts.append(account)

    def deposit(self, account_number, amount):
        account = next((acc for acc in self.accounts if acc.account_number ==
                      account_number), None)
        if account:
            account.balance += amount
            print(f"Deposited {amount} into account {account_number}. New balance:
{account.balance}")
        else:
            print(f"Account {account_number} not found.")

# Example usage
bank = BankingSystem()
bank.create_account("12345", 1000)
bank.deposit("12345", 500)
```

Question 40: Calculate Area of Geometric Shapes

python

Copy code

```
import math
```

```
def calculate_area(shape, *args):
```

```
    if shape == "circle":
```

```
        return math.pi * args[0]**2
```

```
    elif shape == "rectangle":
```

```
        return args[0] * args[1]
```

```
    elif shape == "triangle":
```

```
        return 0.5 * args[0] * args[1]
```

```
# Example usage
```

```
shape_type = input("Enter shape type (circle, rectangle, triangle): ")
```

```
result = calculate_area(shape_type, 3, 4) # Example dimensions, adjust as needed
```

```
print(f"The area of the {shape_type} is: {result}")
```

Feel free to ask for more code snippets or explanations for other questions!

User

continue 10 more

ChatGPT

Certainly! Here are the code snippets for the next set of questions:

Question 41: Generate Word Cloud from Text File

```
python
```

Copy code

```
from wordcloud import WordCloud
```

```
import matplotlib.pyplot as plt
```

```
def generate_word_cloud(text_file_path):
```

```
    with open(text_file_path, 'r') as file:
```

```
        text = file.read()
```

```
        wordcloud = WordCloud(width=800, height=400,  
background_color='white').generate(text)
```

```
    plt.figure(figsize=(10, 5))
```

```
    plt.imshow(wordcloud, interpolation='bilinear')
```

```
    plt.axis('off')
```

```
    plt.show()
```

```
# Example usage
```

```
text_file_path = "/path/to/text/file.txt"
```

```
generate_word_cloud(text_file_path)
```

Question 42: Second Most Frequent Element in a List

```
from collections import Counter
```

```
def second_most_frequent(lst):
```

```
    counts = Counter(lst)
```

```
sorted_counts = sorted(counts.items(), key=lambda x: x[1], reverse=True)

if len(sorted_counts) > 1:
    return sorted_counts[1][0]
else:
    return None

# Example usage
my_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
result = second_most_frequent(my_list)
print(f"The second most frequent element is: {result}")
```

Question 43: Interact with SQLite Database

```
import sqlite3

def interact_with_database():
    connection = sqlite3.connect("my_database.db")
    cursor = connection.cursor()

    # Create a table
    cursor.execute("""CREATE TABLE IF NOT EXISTS users
                    (id INTEGER PRIMARY KEY, name TEXT, age INTEGER)""")

    # Insert data
    cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("John Doe", 30))
    cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("Jane Doe", 25))
```

```
# Query data
cursor.execute("SELECT * FROM users")
data = cursor.fetchall()
for row in data:
    print(row)
```

```
# Commit and close
connection.commit()
connection.close()
```

```
# Example usage
interact_with_database()
```

Question 44: Image Resizing

```
from PIL import Image
```

```
def resize_images(input_folder, output_folder, size=(300, 300)):
    for filename in os.listdir(input_folder):
        if filename.endswith('.jpg', '.png'):
            image_path = os.path.join(input_folder, filename)
            image = Image.open(image_path)
            resized_image = image.resize(size)

            output_path = os.path.join(output_folder, filename)
            resized_image.save(output_path)
```

```
# Example usage
```

```
input_folder = "/path/to/input/images"
output_folder = "/path/to/output/images"
resize_images(input_folder, output_folder)
```

Question 45: Find and Print Prime Numbers

```
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def find_prime_numbers(limit):
    primes = [num for num in range(2, limit) if is_prime(num)]
    return primes
```

```
# Example usage
limit = int(input("Enter the limit: "))
result = find_prime_numbers(limit)
print("Prime numbers up to", limit, "are:", result)
```

Question 46: Generate Network Graph using Networkx

```
import networkx as nx
import matplotlib.pyplot as plt
```

```
def generate_network_graph():

    G = nx.Graph()

    G.add_nodes_from([1, 2, 3, 4])

    G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 1)])


    pos = nx.spring_layout(G)

    nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=1000, font_size=10,
    font_color='black', font_weight='bold', edge_color='gray')

    plt.show()

# Example usage
generate_network_graph()
```

Question 47: Analyze and Visualize Data from CSV

```
import pandas as pd

import matplotlib.pyplot as plt


def analyze_and_visualize_csv(csv_file_path):

    df = pd.read_csv(csv_file_path)

    # Perform analysis (adjust as needed)
    summary_stats = df.describe()

    # Visualize data (adjust as needed)
    df.plot(kind='bar', x='Category', y='Value', color='blue')

    plt.title('Data Visualization')

    plt.xlabel('Category')
```

```
plt.ylabel('Value')
plt.show()

# Example usage
csv_file_path = "/path/to/data.csv"
analyze_and_visualize_csv(csv_file_path)
```

Question 48: Web Scraping

```
import requests
from bs4 import BeautifulSoup

def scrape_website(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extract data from the website (adjust as needed)
    titles = soup.find_all('h2')
    for title in titles:
        print(title.text)

# Example usage
website_url = "https://example.com"
scrape_website(website_url)
```

Question 49: Convert Roman Numerals to Integers

```
def roman_to_int(roman):
    roman_numerals = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
    result = 0

    for i in range(len(roman)):
        if i > 0 and roman_numerals[roman[i]] > roman_numerals[roman[i - 1]]:
            result += roman_numerals[roman[i]] - 2 * roman_numerals[roman[i - 1]]
        else:
            result += roman_numerals[roman[i]]

    return result

# Example usage
roman_numeral = input("Enter a Roman numeral: ")
result = roman_to_int(roman_numeral)
print(f"The integer equivalent is: {result}")
```

Question 50: Generate Pascal's Triangle

```
def generate_pascals_triangle(rows):
    triangle = [[1] * (i + 1) for i in range(rows)]

    for i in range(2, rows):
        for j in range(1, i):
            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j]
```

```
return triangle
```

```
# Example usage
```

```
num_rows = int(input("Enter the number of rows for Pascal's Triangle: "))  
result = generate_pascals_triangle(num_rows)  
for row in result:  
    print(row)
```

Conclusion

Congratulations!

Congratulations on completing this course! You have taken an important step in unlocking your full potential. Completing this course is not just about acquiring knowledge; it's about putting that knowledge into practice and making a positive impact on the world around you.



Share this course