Week 3

Prof. Nitin Chandrachoodan
Department of EE, IIT Madras

Notes by Adarsh (23f2003570)

L3.1 Overview of MVC (13:46)

Model View Controller

Model-View-Controller (MVC) is a design pattern commonly used in software development to separate an application into three interconnected components. This separation helps manage complexity, promotes organized code, and enhances maintainability. Here's a breakdown of each component:

1. **Model**:
    - Represents the data and business logic of the application.
    - Manages the data, including retrieval, storage, and manipulation.
    - Notifies the View when changes occur, so the user interface can update accordingly.
2. **View**:
    - Represents the user interface elements of the application.
    - Displays data from the Model and sends user commands to the Controller.
    - Updates the user interface when the Model changes.
    - View is what a machine, human or entity can see. Even API outputs are views.
        - View is any "representation" useful to another entity.
3. **Controller**:
    - Acts as an intermediary between the Model and View.
    - Processes user input, interacts with the Model, and updates the View.
    - Contains the application logic to respond to user actions.

How It Works Together:

- When a user interacts with the View (e.g., clicks a button), the event is sent to the Controller.
- The Controller processes the input, possibly updating the Model.
- The Model then updates its state and notifies the View of the change.
- The View refreshes to reflect the current state of the Model.

Advantages of MVC:

- **Separation of Concerns**: Each component has distinct responsibilities, making it easier to manage and develop.
- **Testability**: You can test components independently, improving reliability.
- **Reusability**: Models and Views can be reused across different applications.
- **Flexibility**: You can change one component (e.g., the View) without affecting others.

Some SmallTalk trivia

Smalltalk-80 is a programming language and environment that introduced several innovative features. Here are some of its key features:

1. **Object-Oriented Programming**:
    - Everything in Smalltalk-80 is an object, including classes, methods, and even control structures.
    - Supports encapsulation, inheritance, and polymorphism, making it a pure object-oriented language.
2. **Dynamic Typing**:
    - Smalltalk-80 uses dynamic typing, meaning types are determined at runtime.
    - This allows for greater flexibility in programming and easier prototyping.
3. **Message Passing**:
    - Objects communicate through message passing, where one object sends a message to another to invoke behavior.
    - This promotes loose coupling and enhances modularity.
4. **Integrated Development Environment (IDE)**:
    - Smalltalk-80 comes with a rich graphical IDE that includes a workspace, inspectors, and browsers.
    - The IDE allows for real-time coding, debugging, and object exploration.
5. **Reflection**:
    - Smalltalk-80 supports reflection, enabling programs to inspect and modify their own structure and behavior at runtime.
    - This feature facilitates powerful metaprogramming capabilities.
6. **Garbage Collection**:
    - Automatic memory management through garbage collection helps prevent memory leaks.
    - This allows developers to focus on design and functionality without worrying about manual memory management.

7. **Unified Syntax**:
    - Smalltalk-80 has a simple and consistent syntax, which contributes to its readability and ease of use.
    - The syntax is based on sending messages to objects, reducing complexity.
8. **Collections and Iterators**:
    - The language provides rich built-in collection classes (like arrays, sets, and dictionaries) with powerful iteration mechanisms.
    - This makes it easy to manipulate groups of objects.
9. **Visual Programming**:
    - Smalltalk-80 supports visual programming through its user interface, allowing developers to create graphical user interfaces easily.
    - The environment encourages experimentation and exploration through direct manipulation.
10. **Multitasking**:
    - The environment supports lightweight processes (also known as "fibers" or "threads") for concurrent programming.
    - This allows for responsive applications and better resource utilization.

Smalltalk-80 significantly influenced the development of modern programming languages and environments, especially in terms of object-oriented design and interactive programming. Its design principles continue to resonate in contemporary software development practices.

L3.2 Views (12:30)

L3.3 User Interface Design (11:52)

Best Practices in User Interface Design

1. Consistency

- **Example**: Use the same color scheme, typography, and button styles throughout the application.
- **Benefit**: Consistency helps users understand how to interact with your application more easily. For instance, if a blue button represents "Submit" on one page, it should do the same on all other pages.

2. Clarity

- **Example**: Use clear and concise labels for buttons and input fields. Instead of "Submit," consider using "Send Message" for a contact form.

- **Benefit**: Clarity ensures users know what actions to take and what information is required. Avoid jargon and technical terms that may confuse users.

3. Feedback

- **Example**: When a user clicks a button, provide immediate visual feedback (e.g., change the button color or show a loading spinner).
- **Benefit**: Feedback informs users that their action has been registered and the system is responding. This reduces anxiety and enhances the user experience.

4. Affordance

- **Example**: Design buttons to look clickable (e.g., using shadows and rounded edges) and links to look like text hyperlinks (underlined, different color).
- **Benefit**: Affordance cues help users understand how to interact with interface elements without needing instructions.

5. Hierarchy and Layout

- **Example**: Use headings, subheadings, and whitespace to create a clear visual hierarchy in forms or content pages.
- **Benefit**: A well-structured layout guides users through the content, making it easier to find important information quickly.

6. Accessibility

- **Example**: Ensure that all images have alt text, use sufficient color contrast, and provide keyboard navigation options.
- **Benefit**: Accessibility ensures that all users, including those with disabilities, can effectively use the application, broadening your audience.

7. Minimize User Input

- **Example**: Use dropdowns or checkboxes instead of text fields whenever possible. For example, if asking for a date, provide a date picker.
- **Benefit**: Reducing the amount of input required makes it easier for users to complete forms and reduces the likelihood of errors.

8. Error Prevention and Recovery

- **Example**: Validate user input in real-time and provide clear error messages that guide users on how to correct issues.
- **Benefit**: Preventing errors before they occur enhances usability, and clear recovery messages help users feel supported when mistakes happen.

9. Mobile Responsiveness

- **Example**: Design for various screen sizes using responsive layouts that adjust content dynamically based on device size.
- **Benefit**: A responsive design ensures that users have a good experience on all devices, whether they're using a desktop, tablet, or smartphone.

10. User Testing

- **Example**: Conduct usability tests with real users to gather feedback on your design and identify areas for improvement.
- **Benefit**: User testing provides insights into how actual users interact with your interface, helping you make informed design decisions based on real-world behavior.

Aesthetics

Aesthetics in web applications refers to the visual appeal and overall design quality of a website or application. Aesthetics play a crucial role in user experience (UX) by influencing how users perceive, interact with, and feel about a digital product. Here are some key aspects of aesthetics and best practices to enhance them:

Key Aspects of Aesthetics

1. **Color Scheme**: The choice of colors can evoke emotions and establish brand identity.
2. **Typography**: Font selection, size, and spacing contribute to readability and visual hierarchy.
3. **Layout and Spacing**: The arrangement of elements affects flow and usability, while spacing improves clarity and focus.
4. **Imagery and Graphics**: High-quality images and graphics enhance visual interest and can communicate messages quickly.
5. **Consistency**: Uniform design elements foster a cohesive look and feel throughout the application.

Best Practices for Aesthetics in Web Applications

1. **Create a Cohesive Color Palette**:
    - **Example**: Use a primary color for key actions (like buttons) and complementary colors for backgrounds and text.
    - **Benefit**: A well-thought-out color palette enhances brand recognition and user experience, creating a visually appealing interface.
2. **Choose Readable Typography**:
    - **Example**: Use a maximum of two or three font families, and ensure they are legible at various sizes. Use headings and subheadings to create hierarchy.
    - **Benefit**: Consistent typography improves readability and guides users

through content effectively.

3. **Utilize White Space**:
    - **Example**: Incorporate adequate margins and padding around elements to prevent overcrowding.
    - **Benefit**: White space enhances clarity, directs focus to important content, and creates a sense of elegance.
4. **Design with Visual Hierarchy**:
    - **Example**: Use size, color, and placement to emphasize important elements (e.g., make call-to-action buttons larger and more colorful).
    - **Benefit**: A clear visual hierarchy helps users navigate the interface and prioritize information effectively.
5. **Incorporate Quality Imagery**:
    - **Example**: Use high-resolution images, illustrations, and icons that align with your brand identity.
    - **Benefit**: Quality visuals can engage users, reinforce your message, and enhance the overall aesthetic appeal.
6. **Maintain Consistency**:
    - **Example**: Use the same style for buttons, icons, and headings across the application.
    - **Benefit**: Consistency fosters familiarity, making the application easier to use and more visually appealing.
7. **Use Intuitive Navigation**:
    - **Example**: Design navigation menus that are clear and easy to access, with visual cues like hover effects.
    - **Benefit**: A well-designed navigation system enhances usability and helps users find information quickly.
8. **Consider Responsive Design**:
    - **Example**: Ensure that the layout adapts smoothly to different screen sizes and devices.
    - **Benefit**: Responsive design provides a consistent aesthetic experience across devices, improving user satisfaction.
9. **Leverage Microinteractions**:
    - **Example**: Use subtle animations for button clicks, form submissions, or notifications.
    - **Benefit**: Microinteractions add a layer of engagement, making the experience feel more interactive and responsive.
10. **Test with Real Users**:

- **Example**: Gather feedback on aesthetic elements from users during usability testing sessions.
- **Benefit**: User feedback helps identify design elements that resonate well and those that may need adjustment to improve visual appeal.

Accessibility

Accessibility in web design refers to the practice of creating websites that can be used by all people, including those with disabilities. Ensuring accessibility means that everyone, regardless of their abilities or disabilities, can perceive, understand, navigate, and interact with the web. Here are key concepts and best practices for making a website accessible:

Key Concepts

1. **Disabilities**: Consider different types of disabilities, such as visual impairments, hearing impairments, motor disabilities, and cognitive disabilities.
2. **Web Content Accessibility Guidelines (WCAG)**: These guidelines provide a framework for making web content more accessible. They are organized around four principles: Perceivable, Operable, Understandable, and Robust (POUR).

Best Practices for Web Accessibility

1. **Use Semantic HTML**:
    - **Example**: Use appropriate HTML elements like `<header>`, `<nav>`, `<main>`, and `<footer>` to define different parts of your webpage.
    - **Benefit**: Semantic HTML provides meaning to content, making it easier for assistive technologies (like screen readers) to interpret the structure.
2. **Provide Text Alternatives**:
    - **Example**: Use `alt` attributes for images to describe their content (e.g., `<img src="image.jpg" alt="A scenic view of a mountain range">`).
    - **Benefit**: Text alternatives ensure that visually impaired users can understand the content conveyed by images.
3. **Ensure Keyboard Navigation**:
    - **Example**: Make sure all interactive elements (like buttons, links, and forms) can be accessed and operated using a keyboard.
    - **Benefit**: Users with motor disabilities often rely on keyboard navigation, so it's crucial to ensure that they can use your site without a mouse.
4. **Use High Color Contrast**:
    - **Example**: Ensure that text contrasts well with its background (e.g., dark text on a light background) and meets recommended contrast ratios (e.g., 4.5:1 for normal text).
    - **Benefit**: High contrast helps users with visual impairments read content

more easily.

5. **Implement Responsive Design**:
    - **Example**: Use flexible layouts and scalable text sizes to ensure that your website works well on various devices and screen sizes.
    - **Benefit**: Responsive design makes it easier for users with different devices and preferences to access content.
6. **Provide Clear and Descriptive Links**:
    - **Example**: Instead of using "click here," use descriptive text like "Download the accessibility report."
    - **Benefit**: Descriptive links give context to users and assistive technologies, helping them understand where a link will take them.
7. **Use Headings and Lists Effectively**:
    - **Example**: Use heading tags (<h1>, <h2>, etc.) to create a clear content hierarchy and <ul> or <ol> for lists.
    - **Benefit**: Proper use of headings and lists aids in navigation and helps screen reader users understand the content structure.
8. **Provide Captions and Transcripts**:
    - **Example**: Include captions for videos and transcripts for audio content.
    - **Benefit**: This makes multimedia content accessible to users who are deaf or hard of hearing.
9. **Ensure Form Accessibility**:
    - **Example**: Label all form elements clearly using <label> tags, and provide error messages that explain what needs to be corrected.
    - **Benefit**: Accessible forms help all users understand what information is required and make it easier to submit forms correctly.
10. **Test with Real Users**:
    - **Example**: Conduct usability testing with users who have disabilities to identify barriers they encounter.
    - **Benefit**: Real user feedback can provide valuable insights and help you make informed improvements to your site's accessibility.

Systematic process for developing User Interfaces

Developing user interfaces (UIs) in a systematic manner helps ensure that the end product is user-friendly, functional, and visually appealing. Here's a structured process you can follow:

1. Define Goals and Requirements

- Identify the purpose of the application.
- Gather requirements from stakeholders and users.
- Define target user personas and their needs.

2. Conduct User Research

- Perform surveys, interviews, and usability testing with potential users.
- Analyze user behavior and preferences to inform design decisions.
- Study competitors to identify best practices and gaps.

3. Create User Personas

- Develop detailed profiles representing different user types.
- Include demographics, goals, motivations, and challenges.
- Use personas to guide design choices and prioritize features.

4. Develop User Scenarios and User Stories

- Create scenarios that describe how users will interact with the interface.
- Write user stories to capture specific functionalities from the user's perspective.
- Ensure that all scenarios align with user goals.

5. Sketch and Wireframe

- Create low-fidelity sketches to visualize layout and structure.
- Develop wireframes to outline UI elements and content organization.
- Focus on functionality rather than aesthetics at this stage.

6. Design Prototypes

- Build interactive prototypes (low-fidelity or high-fidelity) to simulate user interaction.
- Use tools like Figma, Adobe XD, or Sketch for prototyping.
- Incorporate key functionalities to gather user feedback effectively.

7. Conduct Usability Testing

- Test prototypes with real users to identify usability issues.
- Observe user interactions and gather qualitative and quantitative data.
- Use findings to iterate on the design, making necessary adjustments.

8. Design Visual Elements

- Develop a visual design system, including color palettes, typography, and iconography.
- Ensure consistency with branding and visual hierarchy.

- Apply design elements to the prototype, enhancing aesthetics.

9. Implement the Design

- Collaborate with developers to translate designs into functional interfaces.
- Ensure adherence to design specifications and best practices.
- Conduct regular check-ins to address any implementation challenges.

10. Conduct User Acceptance Testing (UAT)

- Involve users in testing the final product to validate that it meets their needs.
- Gather feedback on both functionality and aesthetics.
- Make final adjustments based on user input.

11. Launch the Product

- Prepare for deployment, ensuring all elements are functional and tested.
- Monitor the launch for any immediate issues or user feedback.
- Provide training or documentation as needed.

12. Gather Post-Launch Feedback

- Continue to collect user feedback after launch to identify areas for improvement.
- Analyze usage data to understand user behavior and preferences.
- Plan for future updates and enhancements based on insights.

13. Iterate and Improve

- Use feedback to inform future design iterations and updates.
- Continuously enhance the UI based on user needs and technological advancements.
- Stay informed about design trends and emerging best practices.

User Interaction vs User Interface

1. Keyboards are examples of user interaction, whereas a screen is a user interface.
2. Touchscreens are examples of user interaction, whereas haptic is a user interface.
    1. Note You got this wrong

L3.4 Usability Heuristics

10 usability heuristics

Jakob Nielsen's Usability Heuristics with **real-world examples** to illustrate each principle:

1. Visibility of System Status

The system should always keep users informed about what is happening through appropriate feedback within a reasonable time.

- **Example:** When you upload a file on **Google Drive**, a progress bar shows how much of the upload is complete, indicating the status of the task. This feedback reassures the user that the upload is happening in real-time.

2. Match Between System and the Real World

The system should speak the users' language, with words, phrases, and concepts familiar to the user, following real-world conventions.

- **Example:** On **Airbnb**, the booking interface uses terms like "Check-in" and "Check-out" which mimic hotel processes in the real world, making the system intuitive for users.

3. User Control and Freedom

Users should have the ability to undo mistakes and easily backtrack from unwanted states.

- **Example:** In **Microsoft Word**, there's an "Undo" button that allows users to reverse changes. This feature helps users recover from mistakes such as accidental deletions.

4. Consistency and Standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. The interface should follow platform and industry standards.

- **Example: Facebook** maintains a consistent layout across its platform. For instance, the "Like" button appears in the same place under posts whether on mobile or desktop, reinforcing its recognizable function.

5. Error Prevention

The design should help users avoid errors by guiding them toward correct actions and preventing problems before they occur.

- **Example: Gmail** warns users when they mention an "attachment" in the email body but forget to actually attach a file. This prevents an error from happening.

6. Recognition Rather Than Recall

Minimize the user's memory load by making objects, actions, and options visible. The user should not need to remember information from one part of the dialogue to another.

- **Example:** On **Amazon**, users don't have to remember previously viewed products. The platform provides a "Recently Viewed" section, allowing easy recognition rather than recall.

7. Flexibility and Efficiency of Use

Accelerators, unseen by novice users, should allow expert users to speed up interactions. The system should cater to both inexperienced and advanced users.

- **Example: Adobe Photoshop** offers keyboard shortcuts for advanced users, like "Ctrl + Z" for undoing actions, while beginners can rely on the menu to perform the same tasks more slowly.

8. Aesthetic and Minimalist Design

Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information competes with relevant units and diminishes their relative visibility.

- **Example: Google Search** has a minimalist design, with a clean interface that focuses users on the search bar and relevant search results, avoiding clutter or distractions.

9. Help Users Recognize, Diagnose, and Recover from Errors

Error messages should be expressed in plain language (no codes), precisely indicating the problem, and suggesting a solution.

- **Example: Instagram** gives a clear error message when the wrong password is entered, saying, "Incorrect password. Try again," with a "Forgot Password?" link, making it easy to recover.

10. Help and Documentation

Even though it is better if the system can be used without documentation, help and documentation should be easy to search, focused on the user's task, list concrete steps, and not be too large.

- **Example: WordPress** includes a help section in its dashboard, providing users with quick access to guides and tutorials when building websites. The documentation is detailed and easy to search.

These real-world examples help demonstrate how Jakob Nielsen's usability heuristics are applied across different platforms and systems to improve user experience.

L3.5 Tools Part 1

Tools used in User Interface design

1. Figma

- **Purpose:** Figma is a web-based UI/UX design tool used for creating prototypes, wireframes, and interactive designs. It supports real-time collaboration, making it ideal for teams.
- **Example:** Designers at **Uber** use Figma to create wireframes and design the interface for their mobile apps. Its collaborative features allow teams to work simultaneously on a single design, ensuring smooth coordination among UI/UX designers and developers.

2. Sketch

- **Purpose:** Sketch is a vector-based design tool for UI and UX design. It's primarily used for creating high-fidelity designs and prototypes for websites and mobile apps.
- **Example: Netflix** uses Sketch to design responsive interfaces for their streaming platform. Designers can quickly create artboards for various screen sizes, ensuring a seamless experience on both mobile and desktop devices.

3. Adobe XD

- **Purpose:** Adobe XD is a versatile UI/UX design tool used for wireframing, prototyping, and designing interfaces. It offers integration with other Adobe Creative Cloud tools.
- **Example: Airbnb** designers use Adobe XD to prototype new features for their website and mobile app. With XD's interactive features, they can simulate the user flow and test the designs before development.

4. InVision

- **Purpose:** InVision is a tool used for creating interactive prototypes and collaborating with stakeholders. It's also popular for gathering feedback and managing design workflows.
- **Example: Spotify** uses InVision to share prototypes with team members and gather feedback on early-stage designs for both their desktop and mobile apps. InVision's commenting feature helps streamline the review process.

5. Axure RP

- **Purpose:** Axure RP is a powerful tool for creating wireframes, flowcharts, and interactive prototypes. It's especially useful for designing complex interfaces with dynamic interactions.
- **Example:** Axure is often used by enterprise software companies like **Salesforce** to design and prototype complex user flows in their CRM systems. Axure's ability to

create detailed prototypes helps test functionality before coding begins.

6. Balsamiq

- **Purpose:** Balsamiq is a wireframing tool used to create low-fidelity mockups. It's known for its simplicity and ease of use, making it ideal for quick sketch-style designs.
- **Example:** Startups often use Balsamiq in the early stages of product development to sketch out ideas. For instance, a team working on a new **e-commerce app** might use Balsamiq to create rough wireframes of the shopping cart, navigation, and checkout process.

7. Marvel

- **Purpose:** Marvel is a web-based tool for creating prototypes, wireframes, and user testing. It's known for its simplicity and is often used for user testing early in the design process.
- **Example: BuzzFeed** uses Marvel to test different layouts for their articles. They can quickly prototype new designs and gather feedback from user testing sessions to ensure the best layout is chosen before development.

8. Framer

- **Purpose:** Framer is an interactive design tool that allows designers to build high-fidelity prototypes with advanced animations and micro-interactions.
- **Example: Instagram**'s design team uses Framer to prototype and test animations for in-app interactions such as swipe gestures, like animations, and transitioning between stories. Framer's ability to prototype dynamic interactions helps in refining the app's responsiveness.

9. Principle

- **Purpose:** Principle is a design tool focused on creating animated and interactive UI prototypes, often used to show how an interface behaves when interacted with.
- **Example:** Designers at **Apple** use Principle to create fluid animations and interactions for their mobile app interfaces, showing exactly how a button or transition should behave when tapped, swiped, or dragged.

10. Zeplin

- **Purpose:** Zeplin is a collaboration tool that bridges the gap between designers and developers. It allows designers to export designs and generate specifications that developers can use to implement the UI accurately.
- **Example:** After creating a design in **Sketch** or **Figma**, designers at **Slack** export their designs to Zeplin, where developers can view design specifications, such as font sizes, colors, and padding, ensuring that the final product matches the design.

11. Canva

- **Purpose:** Canva is a user-friendly tool for creating basic UI designs, presentations, social media graphics, and more. While not as advanced as Sketch or Figma, it's useful for quick design needs.
- **Example:** Marketing teams at **small businesses** use Canva to design simple landing pages, social media banners, or app icons, especially when they don't need complex functionality or interactive prototypes.

12. Overflow

- **Purpose:** Overflow is a tool for creating user flow diagrams and visualizing how users navigate through an app or website.
- **Example:** Designers at **Dropbox** use Overflow to map out user journeys through their file storage platform, ensuring that each screen and interaction is logically connected and easy to follow.

13. Miro

- **Purpose:** Miro is an online collaborative whiteboarding tool that is used for brainstorming, wireframing, and mapping out ideas during the design process.
- **Example:** During remote design sprints, teams at **Google** use Miro to sketch out early wireframes and map out the information architecture of new features. It helps in visualizing the entire design process and gathering feedback from various stakeholders.

14. Adobe Illustrator

- **Purpose:** Adobe Illustrator is a vector-based design tool used for creating icons, logos, and detailed illustrations for UI elements.
- **Example:** UI designers at **LinkedIn** might use Adobe Illustrator to create custom icons for the platform's interface. These icons are then imported into tools like Figma or Sketch for use in the overall design.

15. FlowMapp

- **Purpose:** FlowMapp is a tool used to design user flows and sitemaps, helping to organize the structure of a website or app.
- **Example: Etsy** designers use FlowMapp to create sitemaps that organize all the sections of their e-commerce platform, ensuring a seamless flow from homepage to checkout, making it easy for users to navigate the site.

L3.6 Tools - Part II

PyHTML

PyHTML is a Python library that allows you to generate HTML content using Python code. It helps create HTML pages dynamically by writing Python scripts instead of manually writing HTML. This can be especially useful when you need to create web pages programmatically or generate HTML as part of a larger Python application.

Key Features of PyHTML:

1. **Pythonic HTML Generation:** You can use Python functions and logic to create HTML elements and structure.
2. **Modular HTML Construction:** Break down complex HTML structures into reusable Python functions.
3. **Dynamic Content Creation:** Combine dynamic data with HTML templates.
4. **Clean, Minimal Code:** Avoid the verbosity of mixing HTML and Python by keeping everything within Python's syntax.

Example of PyHTML in Action:

Here's a simple example of using PyHTML to generate a basic HTML page with a heading, paragraph, and a list.

```
from pyhtml import html, head, title, body, h1, p, ul, li

# Define the HTML structure using PyHTML elements
def generate_webpage():
    return html(
        head(
            title("My PyHTML Webpage")
        ),
        body(
            h1("Welcome to My Webpage"),
            p("This page is generated using PyHTML in Python."),
            ul(
                li("First item"),
                li("Second item"),
                li("Third item")
            )
        )
    )

# Render the HTML as a string
webpage_html = str(generate_webpage())

# Output the HTML string (can be saved to a file or served in a web
print(webpage_html)
```

Output HTML:

```
<html>
  <head>
    <title>My PyHTML Webpage</title>
  </head>
  <body>
    <h1>Welcome to My Webpage</h1>
    <p>This page is generated using PyHTML in Python.</p>
    <ul>
      <li>First item</li>
      <li>Second item</li>
      <li>Third item</li>
    </ul>
  </body>
</html>
```

How to Use PyHTML:

1. **Install PyHTML:** If you don't have it installed yet, you can install it via pip:

   ```
   pip install pyhtml
   ```

2. **Run the Script:** After writing the script, you can run it in your Python environment, and the generated HTML can be printed out, saved to a file, or used within a web application.

Use Cases for PyHTML:

- **Dynamic HTML Page Generation:** For web apps where the content or structure changes dynamically.
- **Automating Reports:** Create HTML reports with Python-generated data.
- **Email Templates:** Generate HTML email content programmatically.
- **Web Scraping/Data Processing Applications:** When you need to output processed data as HTML.

Iteration in PyHTML

```python
from pyhtml import html, head, title, body, h1, ul, li

# Sample data to iterate over
items = ["Apple", "Banana", "Cherry", "Date", "Elderberry"]

# Function to generate HTML using iteration
def generate_fruit_list():
    return html(
        head(
            title("Fruit List Page")
        ),
        body(
            h1("Fruits I Like"),
            ul(
                # Using a list comprehension with li to create list
                [li(item) for item in items]
            )
        )
    )

# Render the HTML as a string
fruit_list_html = str(generate_fruit_list())

# Output the generated HTML
print(fruit_list_html)
```

String Template

```
from string import Template

# Template string with placeholders
template = Template("<li>$item</li>")

# List of items to iterate over
items = ["Apple", "Banana", "Cherry", "Date", "Elderberry"]

# Iterating over items and substituting values in the template
html_list_items = ""
for item in items:
    html_list_items += template.substitute(item=item)

# Creating the final HTML by embedding the list items
final_html = Template("""
<html>
  <head>
    <title>$title</title>
  </head>
  <body>
    <h1>$header</h1>
    <ul>
      $list_items
    </ul>
  </body>
</html>
""").substitute(title="Fruit List", header="Fruits I Like", list_it

# Output the generated HTML
print(final_html)
```

Jinja2 Templating

```
from jinja2 import Template

# Sample hierarchical data (e.g., categories and subcategories)
categories = [
    {
        "name": "Electronics",
        "subcategories": [
            {
                "name": "Mobile Phones",
                "subcategories": [
                    {"name": "Smartphones", "subcategories": []},
                    {"name": "Feature Phones", "subcategories": []}
                ]
            },
            {"name": "Laptops", "subcategories": []},
        ]
    },
    {
        "name": "Clothing",
        "subcategories": [
            {"name": "Men", "subcategories": []},
            {"name": "Women", "subcategories": []}
        ]
    },
]

# Jinja2 template with a recursive loop
template_str = """
<ul>
    {% for category in categories recursive %}
        <li>
            {{ category.name }}
            {% if category.subcategories %}
                <ul>
                    {{ loop(category.subcategories) }}
                </ul>
            {% endif %}
        </li>
    {% endfor %}
</ul>
"""

# Compile the template
template = Template(template_str)

# Render the template with the data
rendered_html = template.render(categories=categories)

# Output the generated HTML
print(rendered_html)
```

L3.7 Accessibility (20:07)

W3C Accessibility guidelines

The **W3C Web Content Accessibility Guidelines (WCAG)** are a set of recommendations aimed at making web content more accessible, especially for people with disabilities. The guidelines are categorized into four key principles: **Perceivable, Operable, Understandable, and Robust (POUR)**. Here's a breakdown of these principles with examples:

1. Perceivable: Information must be presentable to users in ways they can perceive.

- **Text Alternatives**: Provide text alternatives for non-text content (e.g., images).
  - **Example**: Use `alt` text for images to describe what the image shows, so screen readers can interpret it.
- **Time-based Media**: Provide alternatives for time-based media (e.g., videos).
  - **Example**: Offer transcripts for audio content and captions for video content.
- **Adaptable Content**: Ensure content can be presented in different ways (e.g., simpler layout) without losing information or structure.
  - **Example**: Use proper HTML tags (e.g., headings and lists) so assistive technologies can reformat the content effectively.
- **Distinguishable**: Make it easier for users to see and hear content, including separating foreground from background.
  - **Example**: Ensure sufficient color contrast between text and background (e.g., dark text on a light background).

2. Operable: User interface components and navigation must be operable by all users.

- **Keyboard Accessibility**: All functionality should be available from a keyboard.
  - **Example**: Users should be able to navigate through a website using only the keyboard (e.g., with the Tab key).
- **Enough Time**: Give users enough time to read and use content.
  - **Example**: Avoid time limits, or provide ways to extend time on pages that auto-refresh or expire.
- **Seizure and Physical Reactions**: Do not design content in a way that is known to cause seizures.
  - **Example**: Avoid using flashing or rapidly flickering content (no content should flash more than three times per second).
- **Navigable**: Provide ways to help users navigate, find content, and determine where they are.
  - **Example**: Ensure the website has clear headings, consistent navigation

menus, and breadcrumb links.

3. Understandable: Information and the operation of the user interface must be understandable.

- **Readable Text**: Make text readable and understandable.
  - **Example**: Use plain language and avoid jargon that could be difficult for some users to understand.
- **Predictable Web Pages**: Make web pages appear and operate in predictable ways.
  - **Example**: Ensure that if a user interacts with an element (e.g., clicking a link), they know what will happen. For instance, links should be clearly identified as such, and clicking them should open the expected page.
- **Input Assistance**: Help users avoid and correct mistakes.
  - **Example**: Provide error suggestions if the user fills out a form incorrectly (e.g., highlighting which field was missed and offering hints).

4. Robust: Content must be robust enough that it can be reliably interpreted by a wide variety of user agents, including assistive technologies.

- **Compatible with Assistive Technologies**: Maximize compatibility with current and future user agents, including assistive technologies.
  - **Example**: Ensure the use of standard HTML practices so that screen readers and other tools can correctly interpret the website structure and content.

Examples of WCAG Success Criteria

Example 1: Color Contrast (Perceivable)

- **Issue**: A website uses light gray text on a white background.
- **Solution**: Increase the contrast by using black text on a white background or dark gray text on a light gray background to meet WCAG contrast requirements (minimum ratio of 4.5:1 for normal text).

Example 2: Keyboard Navigation (Operable)

- **Issue**: A drop-down menu can only be activated by hovering the mouse.
- **Solution**: Ensure that the menu can also be opened and navigated via keyboard (e.g., by pressing the Tab key and using arrow keys).

Example 3: Form Input Error (Understandable)

- **Issue**: A user submits a form without entering a required field, and the page just reloads without any feedback.
- **Solution**: Provide error messages clearly indicating which fields are required and give suggestions on how to correct the errors.

Example 4: Screen Reader Compatibility (Robust)

- **Issue**: A website's navigation bar is coded in a way that is unreadable by screen readers.
- **Solution**: Use semantic HTML (e.g., nav tags for navigation and proper headings) so assistive technologies can interpret the structure and content correctly.

These guidelines ensure that websites and web applications are more inclusive, enhancing the user experience for everyone, especially individuals with disabilities.

Extra Lectures

1. Execute another python file

```
exec (open('hello.py').read())
```

2. Python Arguments

```
import sys
sys.argv()
```

3. Python Entrypoint

__name__ is the current scope.

```
def main():
    print('Hello World')
if __name__ == '__main__':
    main()
```

String Templating

Named Arguments

```
TEMPLATE = """
Hello {name}
"""

print(TEMPLATE.format(name="Thej"))
```

Positional Arguments

```
TEMPLATE = """
Hello {} OR Hello {0}.. Just try this to get a better understanding
"""

print(TEMPLATE.format("Thej"))
```

String Formatting

```
TEMPLATE = """
This is a Decimal {value:d}, and this is Hex {value: x}
"""
```

4. Virtual Environments

```
python3 -m venv .venv-name
source .venv-name/bin/activate

which python ## Local python will show up

pip install ....
pip freeze > requirements.txt
```

5. Jinja2 Templating

```
from jinja2 import template
```