GUIs

10 / ~~09~~ 16 / 19

Rahul Arya

Graphical User Interfaces

[Demo]

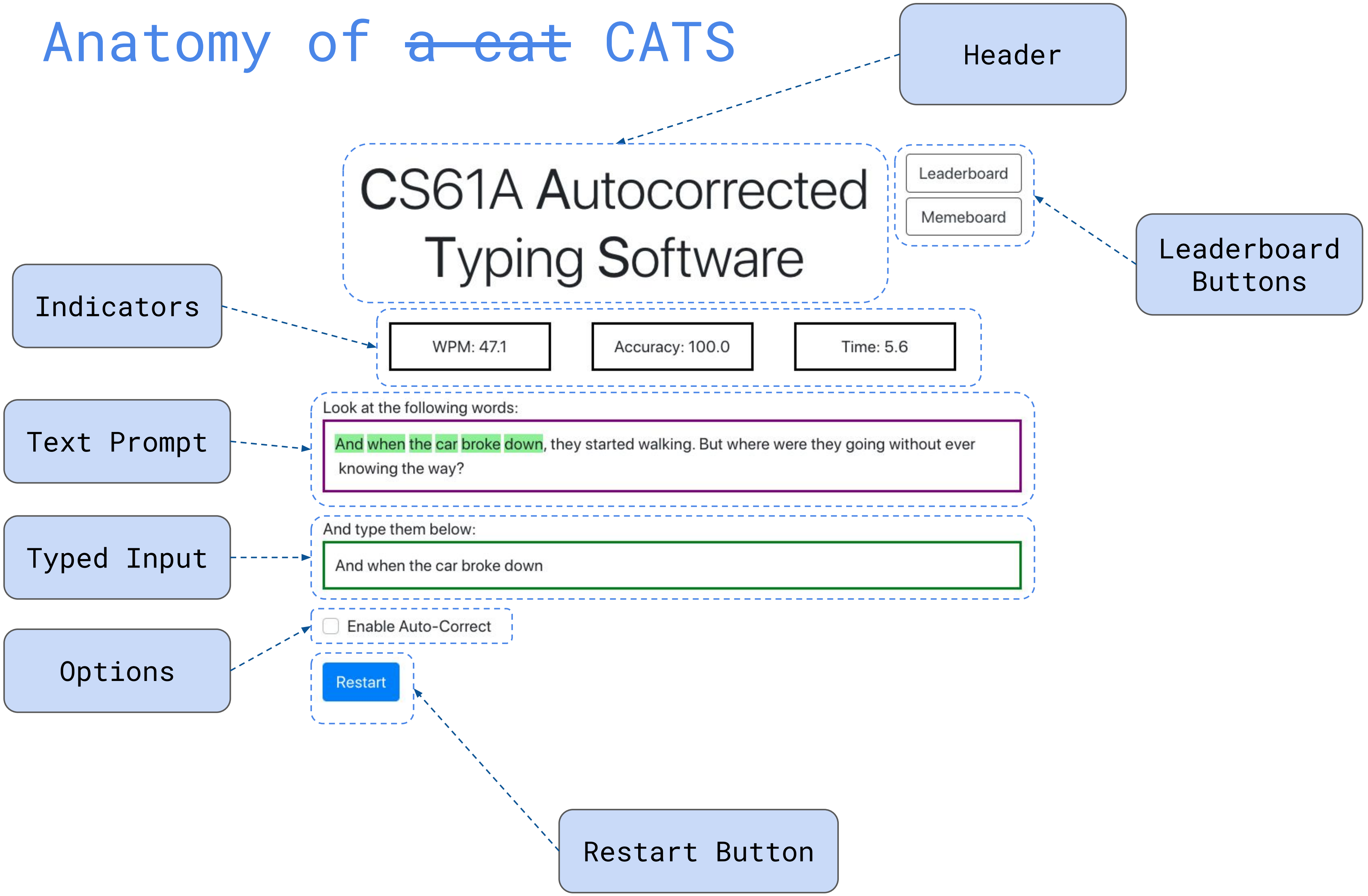
Various Platforms / Languages / Tools

- Android / iOS / Desktop / Web / ...
- Java / Swift / C# / JavaScript / ...
- Android Studio / Xcode / Visual Studio / WebStorm / ...

- What's the common element?
- **Component-level abstraction**

Anatomy of a cat

Anatomy of ~~a cat~~ CATS



CS61A Autocorrected Typing Software

WPM: 47.1 | Accuracy: 100.0 | Time: 5.6

Look at the following words:

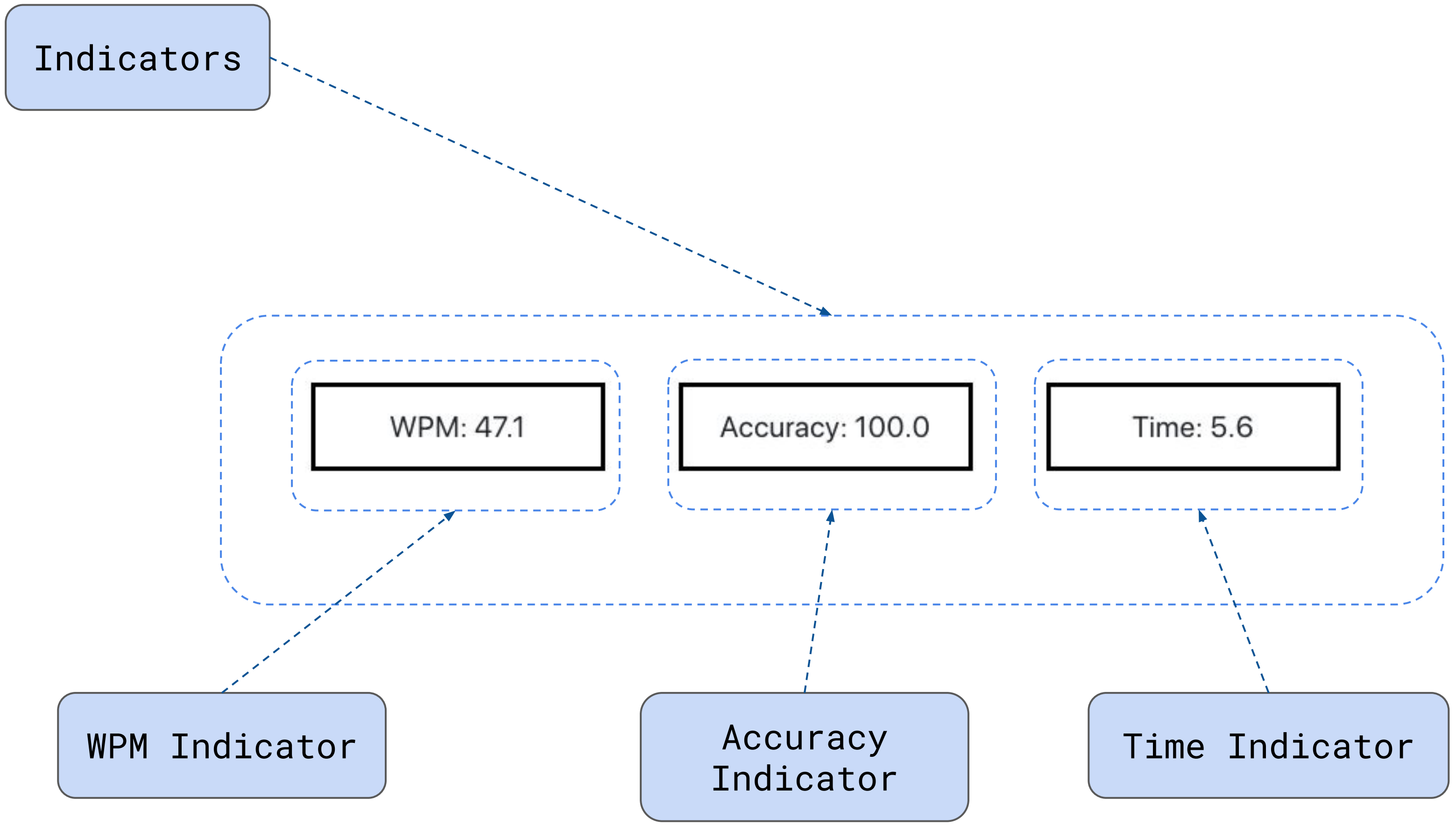
And when the car broke down, they started walking. But where were they going without ever knowing the way?

And type them below:

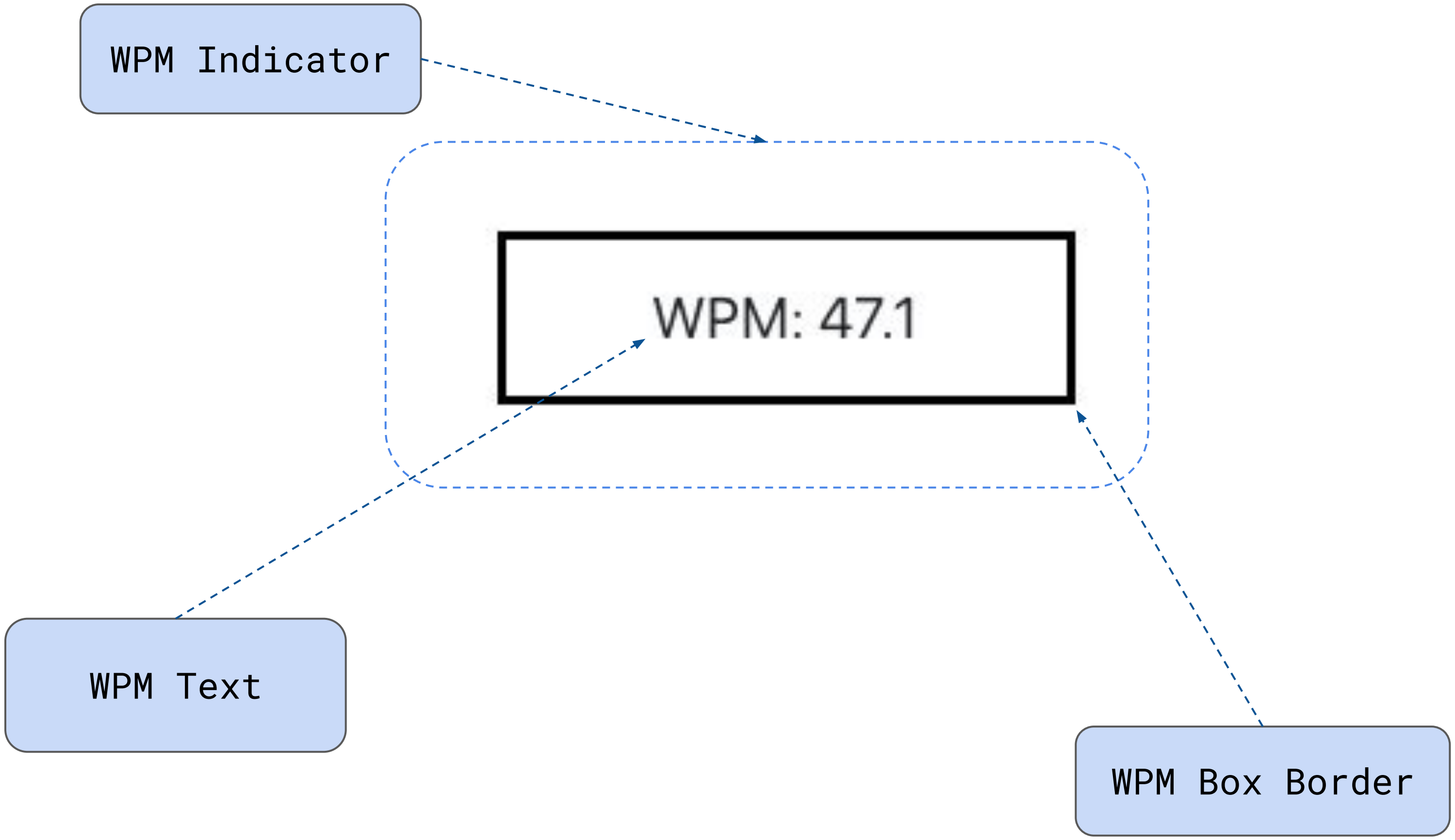
And when the car broke down

Enable Auto-Correct

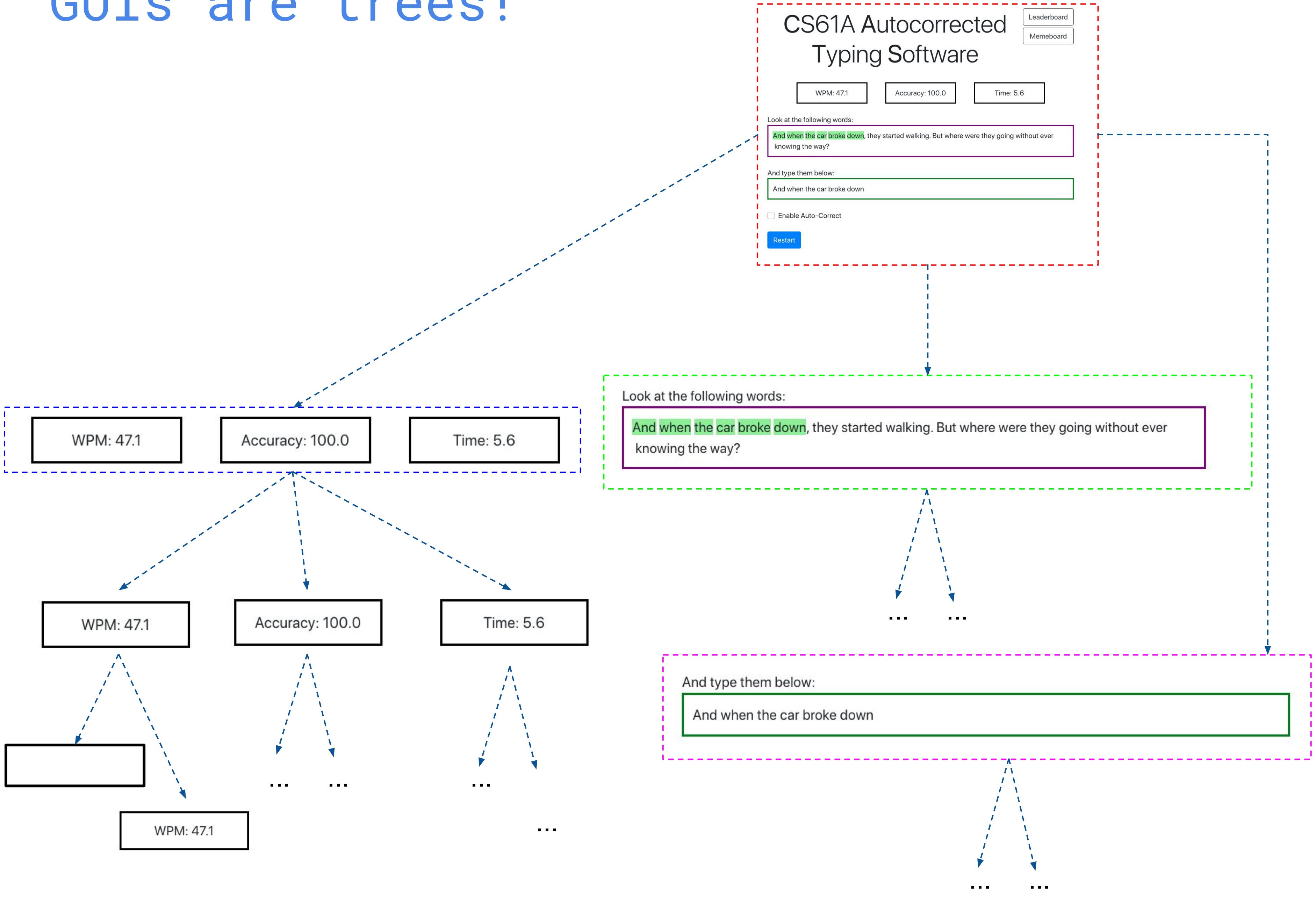
Anatomy of ~~a cat~~ CATS



Anatomy of ~~a cat~~ CATS



GUIs are trees!



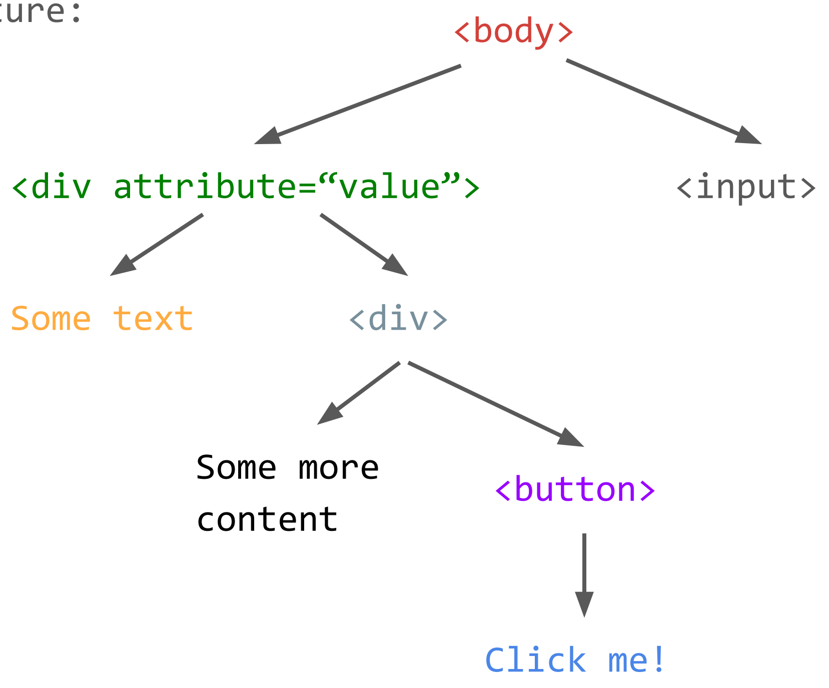
CS61A Autocorrected Typing Software
Leaderboard
Memeboard
WPM: 47.1
Accuracy: 100.0
Time: 5.6
Look at the following words:
And when the car broke down, they started walking. But where were they going without ever knowing the way?
And type them below:
And when the car broke down
Enable Auto-Correct
Restart
WPM: 47.1
Accuracy: 100.0
Time: 5.6
Look at the following words:
And when the car broke down, they started walking. But where were they going without ever knowing the way?
WPM: 47.1
Accuracy: 100.0
Time: 5.6
And type them below:
And when the car broke down
WPM: 47.1
...

Web Development

Web Development in 2 minutes

- Why web development? Easy to pick up, play around in your browser, runs on pretty much every device!
- **HTML**
  - Describes the organization of a web page
  - Made up of “tags” in a tree structure:

```
<body>
     <div attribute=“value”>
          Some text
          <div>
               Some more content
               <button>
                    Click me!
               </button>
          </div>
     </div>
     <input />
</body>
```



[Demo]

Web Development in 2 minutes

- **JavaScript**
- At a high-level, similar-“ish” to Python
- Just new syntax - semicolons, braces, indentation optional!

Syntax	**Python**	**JavaScript**
Variable assignment	`x = 5`	`let x = 5;`
Variable *reassignment*	`x = 5`	`x = 5;`
Function declaration	`def func(arg1, arg2):` `cat = arg1 + arg2` `return cat`	`let func = (arg1, arg2) => {` `let cat = arg1 + arg2;` `return cat;` `};`
Class declaration	`class CS61A(CSClass):` `def __init__(self, prof):` `super().__init__()` `self.prof = prof` `def gobears(self, gostr):` `return gostr + self.prof`	`class CS61A extends CSClass {` `constructor(prof) {` `super();` `this.prof = prof;` `}` `gobears(gostr) {` `return gostr + this.prof;` `};` `}`

[Demo]

Web Development in 2 minutes

- **CSS**
- Describes “style” / appearance of a website
- Colors, animations, layout
- Will not discuss further, since it’s specific to the web

- [extra] If you’re interested, a great CSS tutorial is at MDN: https://developer.mozilla.org/en-US/docs/Web/CSS

React

(reactjs.org)

What problems does React solve?

- Manipulating the DOM tree directly is a pain as it gets more complex
- The “component tree” of our GUI doesn’t line up with the DOM tree in the browser

Solutions

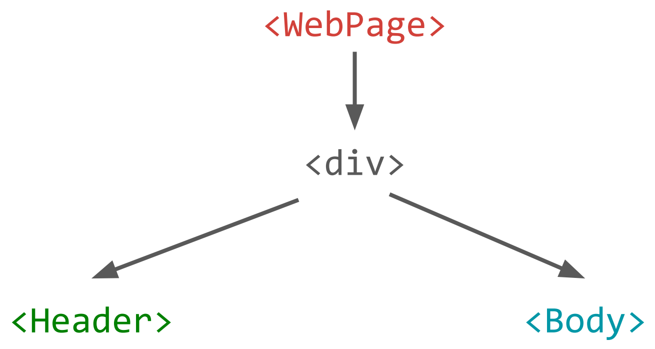
- React enforces abstraction barriers between components
    - Each node in the “component tree” is its own **class**, so components can’t depend on implementation details of other components
- Below the abstraction barrier, React (efficiently) generates and updates the DOM tree as the component tree changes

React Components and JSX

- React components must:
  - Inherit from React.Component
  - Have a render() method that describes its children / subtree
  - render() typically describes its subtree using JSX

Example:

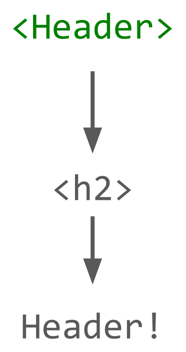
```
class WebPage extends React.Component {
     // render is a function of no arguments
     render() {
          return (
               <div>
                    <Header />
                    <Body />
               </div>
          );
     };
}
```

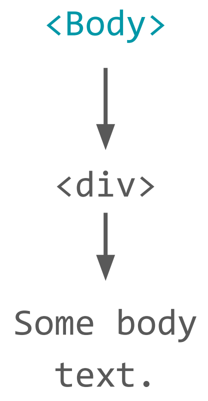


React Components and JSX

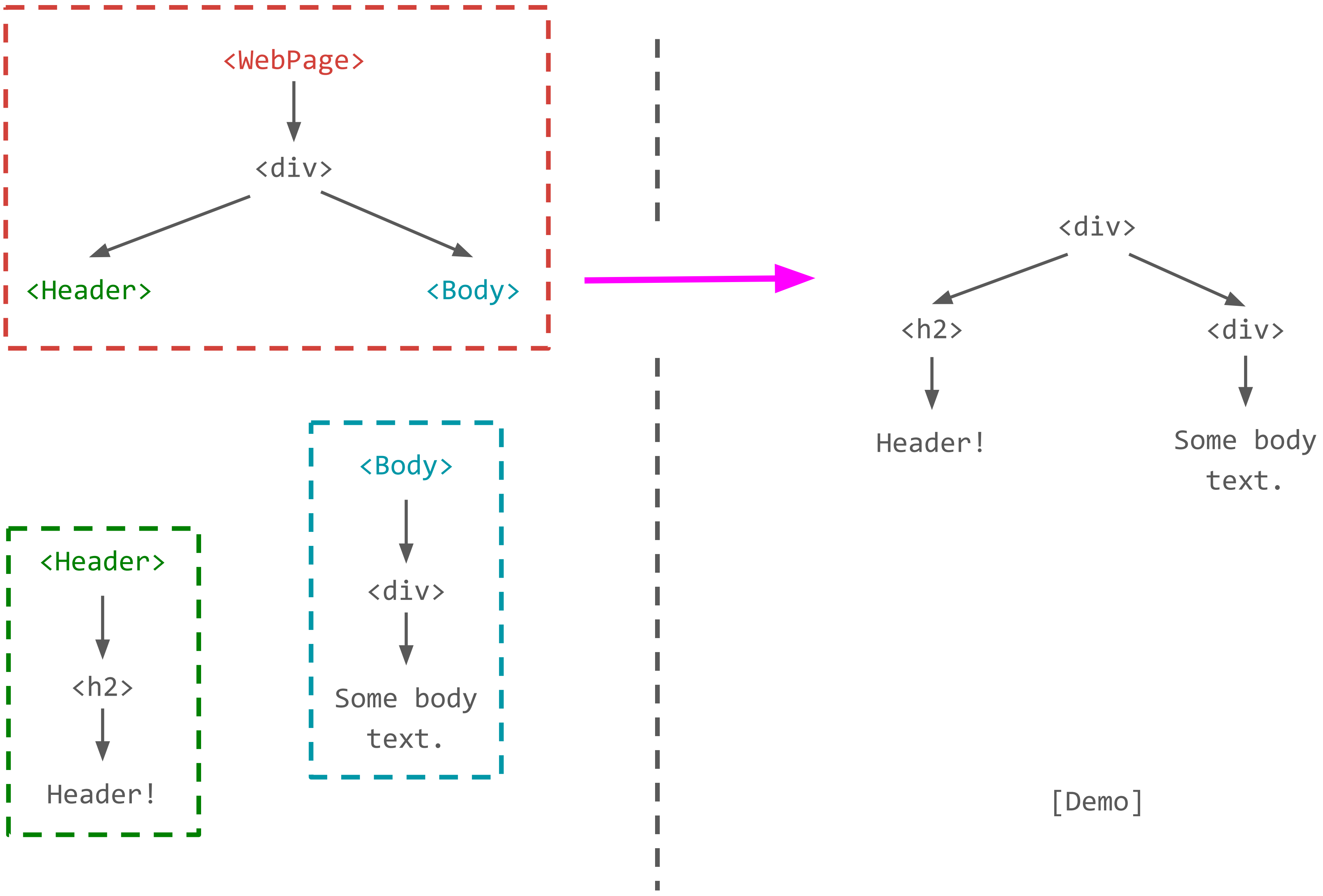
```
class Header extends React.Component {
     render() {
          return (
               <h2>
                    Header!
               </h2>
          );
     };
}

class Body extends React.Component {
     render() {
          return (
               <div>
                    Some body text.
               </div>
          );
     };
}
```





React Components and JSX



[Demo]

More JSX

Render a list of components:

```
class WebPage extends React.Component {
     render() {
          let bodyList = [];
          let i = 0;
          while (i < 3) {
               bodyList.push(<Body />);
               i += 1;
          }
          return (
               <div>
                    <Header />
                    {bodyList}
               </div>
          );
     };
}
```

Include an expression in JSX:

```
class WebPage extends React.Component {
     render() {
          return (
               <div>
                    1 + 2 is
                    {“ ”}
                    {1 + 2}
               </div>
          );
     };
}
```

Passing information to child components

- The parent component may need to pass information to the child components
- Solution: props
- Props are essentially “arguments” for a component
- Received by the component’s constructor
- Stored in a dictionary in the attribute `this.props`

[Demo]

Passing information to child components

```
class WebPage extends React.Component {
     render() {
          return (
               <div>
                    <Header />
                    <Button
                         text="some text"
                    />
               </div>
          );
     };
}
```

```
class Button extends React.Component {
     render() {
          return (
               <div>
                    <button>
                         {this.props.text}
                    </button>
               </div>
          );
     };
}
```

[Demo]

Passing information to child components

```
class WebPage extends React.Component {
     render() {
          let buttonList = [];
          let i = 0;
          while (i < 3) {
               buttonList.push(
                    <Button
                         text={“Button #” + i}
                    />
               );
               i += 1;
          }
          return (
               <div>
                    <Header />
                    {buttonList}
               </div>
          );
     };
}
```

[Demo]

Responding to user input

- So far, we can display information, but not respond to interaction!
- Want code to run when the user does something e.g. clicks a button, types some text, etc.
- Solution: event handlers
- Functions that are called when an “event” occurs - often some form of user interaction
- Can be specified using JSX:

```
<button onClick={handleClick}>
     {this.props.text}
</button>
```

- handleClick will be called when the <button> is clicked

[Demo]

Responding to user input

```
class Button extends React.Component {
     let handleClick = () => {
          alert(“Clicked! I am ” + this.props.text);
     };
     render() {
          return (
               <div>
                    <button onClick={handleClick}>
                         {this.props.text}
                    </button>
               </div>
          );
     };
}
```

[Demo]

Persistent State

- We know how to call a function when an event happens
- But our functions don’t do anything persistent!
- We need to give our components some sort of memory

- In Python, we’d use an instance attribute
  - Initialized in the constructor
  - Updated in the event handler

- Problem!
- The component does not rerender - React does not know when we update an attribute
- Can use the `forceUpdate()` method to fix

[Demo]

Responding to user input

```
class Button extends React.Component {
     constructor(props) {
          super(props);
          this.numberOfClicks = 0;
     }
     let handleClick = () => {
          this.numberOfClicks += 1;
          this.forceUpdate();
     };
     render() {
          return (
               <div>
                    <button onClick={handleClick}>
                         {“Clicked ” + this.numberOfClicks + “times !”}
                    </button>
               </div>
          );
     };
}
```

[Demo]

Persistent State

- `forceUpdate()` is a solution, but it’s not the best one
- We shouldn’t need to tell React when to update, that breaks the abstraction barrier - components should not know about “updates”

- Components should notify React when their <u>state</u> changes, and React can decide when an update is needed
- A component’s render method should only rely on its <u>state</u>
- When the state changes, a render should happen at some point

Persistent State

- State is stored in the `this.state` instance attribute, initialized in the constructor
- Updated using the `this.setState()` method, so React knows when updates happen

[Demo]

Responding to user input

```
class Button extends React.Component {
     constructor(props) {
          super(props);
          this.state = {
               numberOfClicks: 0,
          }
     }
     render() {
          let handleClick = () => {
               this.setState({
                    numberOfClicks: this.state.numberOfClicks + 1
               });
          };
          return (
               <div>
                    <button onClick={handleClick}>
                         {“Clicked ” + this.state.numberOfClicks + “times !”}
                    </button>
               </div>
          );
     };
}
```

[Demo]

Event Handlers as Props

- Often, we want the parent component to update its state in response to an event handler on the child
- Example: When a button is clicked, the header should update a counter
- Event handler must be in the parent component to update state
- But must be bound to an element in the child component

- <u>Solution</u>: Pass the event handler as a prop to the child

[Demo]

Responding to user input

```
class WebPage extends React.Component {
      ...
      let handleClick = () => {
            this.setState({
                  numberOfClicks: this.state.numberOfClicks + 1
            });
      };
      ...
      buttonList.push(
            <Button
                  onClick={handleClick}
            />
      );
      ...
}

class Button extends React.Component {
      ...
      let handleClick = () => {
            this.props.onClick();
      };
      ...
}
```

[Demo]

Summary + Thinking in React

- Directly manipulating the DOM tree gets complicated and messy fast - better to deal with a GUI as a tree of isolated components
- Components are classes that inherit from `React.Component` and that have a `render()` method
- Abstraction barriers isolate implementation of each component
- React updates the DOM tree below the abstraction barrier

- Data flows <u>down</u> the component tree in the form of <u>props</u>
- User input is captured using <u>event handlers</u>
- State is updated using setState() so React knows to re-render the DOM Tree
- Event handlers can be passed down the tree <u>as props</u> for events to flow <u>up</u> the component tree

Next Steps

- Interested in React / GUIs? Awesome!
- Check out the cats project GUI at https://github.com/Cal-CS-61A-Staff/cats-gui

- MDN JavaScript tutorial is a good, rigorous introduction to JavaScript for a 61A student
    - https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps
- Official React tutorial is excellent, goes into a lot more depth
    - https://reactjs.org/
- Resources are available for Android / iOS development as well