Design-systems
are a
Carnival!

One accessible component
Many pretty masks

Kathleen McMahon
@resource11

CityJS CONFERENCE
LONDON

@resource11

https://noti.st/resource11

@resource11

Twitter | Instagram | GitHub

Who are you, again?

DESIGN CODE && MOTION

Kathleen McMahon

ENGINEER • DESIGNER • SPEAKER


SPAM
Hormel

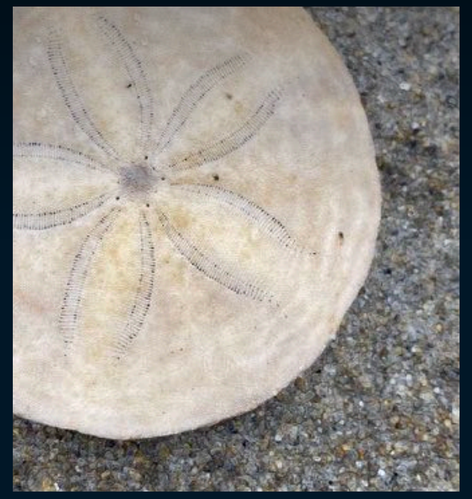

2100
RALEIGH
STANLEY


@resource11
70


SPAM
315
Hormel


@resource11


@resource11




@resource11


@resource11


@resource11


@resource11


@resource11


@resource11


@resource11


@resource11


@resource11


@resource11


@resource11

LaunchDarkly


Northwestern Mutual®


@resource11
Design Systems
are ALWAYS
the hotness

@resource11

Design-systems
are a

Carnival!


@resource11


@resource11
NOLEGGIO COSTUMI
COSTUMES RENT
LOCATION DE COSTUMES


@resource11


@resource11

@resource11


@resource11


@resource11


@resource11


@resource11


@resource11


@resource11

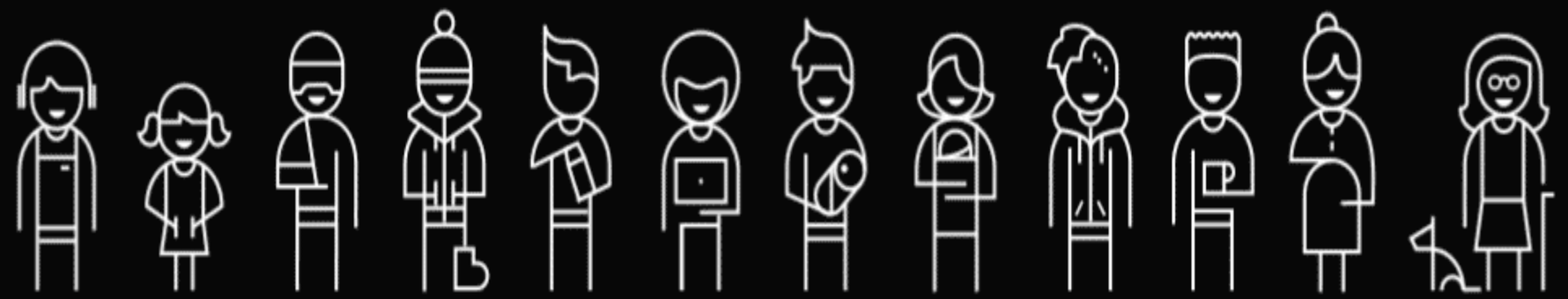

@resource11
© Serena Pe


@resource11

Is it consistent for everyone?

Our users have different needs at different times

Vision Hearing Motor Cognitive



Sensory Language Low bandwidth

Why accessible components?

@resource11

February 2024

WebAIM Million

An accessibility analysis of the top 1,000,000 home pages

@resource11

February 2024

11.8%

Increase in added
ARIA attributes

@resource11

February 2024

34.2%

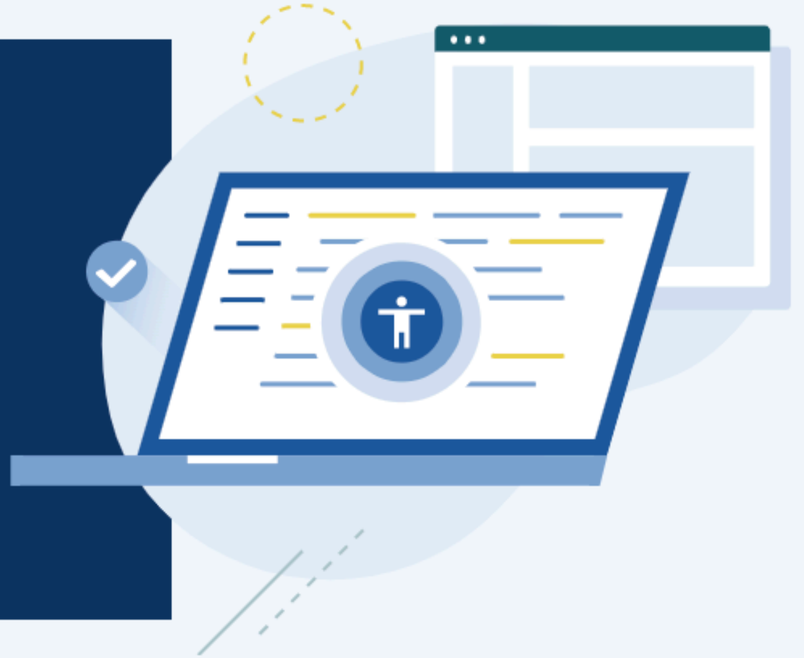
More detected errors than
those without ARIA present

What can we do about this?

ARIA Authoring Practices Guide (APG) Home

Learn to use the accessibility semantics defined by the Accessible Rich Internet Application (ARIA) specification to create accessible web experiences. This guide describes how to apply accessibility semantics to common design patterns and widgets. It provides design patterns and functional examples complemented by in-depth guidance for fundamental practices.

View Patterns

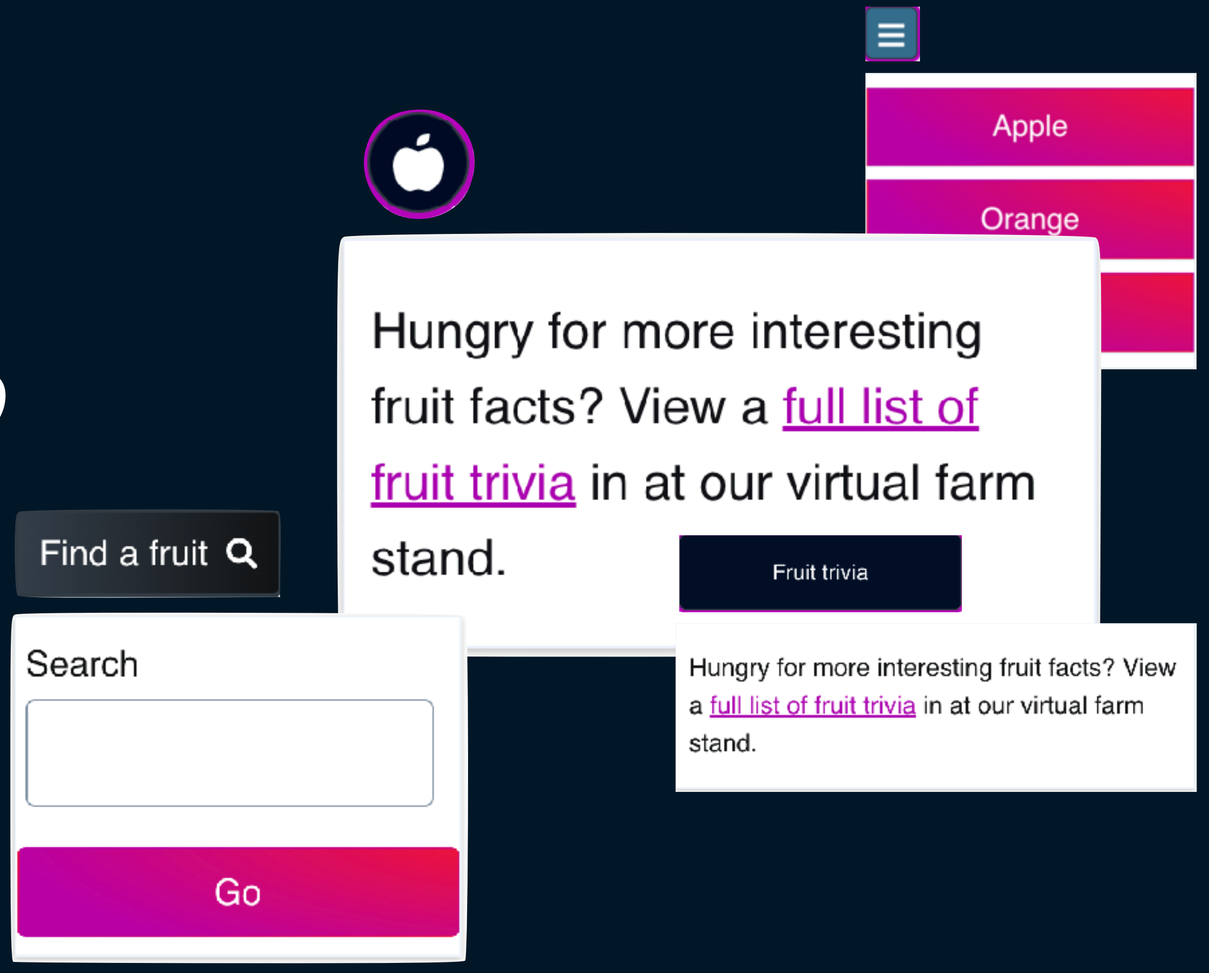


APG RESOURCES

Building blocks that help you make the web accessible

Disclosure widget

One pattern, a few uses



https://noti.st/resource11/1G5PnK/accessibility-flavored-react-components-make-your-design-system-delicious

Anatomy of a disclosure widget

A button that opens & closes a container, then returns focus to the button

checked -- and tri-state checkboxes, which allow an additional third state known as partially checked.

collection of possible values.

Disclosure (Show/Hide)

A disclosure is a widget that enables content to be either collapsed (hidden) or expanded (visible).

Feed

A feed is a section of a page that automatically loads new sections of content as the user scrolls.

Grid (Interactive Tabular Data and Layout Containers)

A grid widget is a container that enables users to navigate the information or interactive elements it contains using directional navigation keys, such as arrow keys, Home, and End.

Landmarks

Landmarks provide a powerful way to identify the organization and structure of a web page.

Link

A link widget provides an interactive reference to a resource.

Listbox

A listbox widget presents a list of options and allows a user to select one or more of them.

Menu and Menubar

A menu is a widget that offers a list of

Menu Button

A menu button is a button that opens a

Meter

A meter is a graphical display of a numeric

↑ Back to Top

Primary keyboard interactions

Space bar/Enter key

Escape key

Mouse clicks

<button />

<button>

Supports

Mouse clicks

Enter keypress

Spacebar keypress

ARIA support

```
  ...props
}) => {
  return (
    <button
      {...props}
      aria-controls={ariaControls}
      aria-expanded={ariaExpanded}
      aria-label={ariaLabel}
      id={buttonId}
      className={clsx(styles.root, styles[size], buttonClasses)}
      disabled={disabled}
      onClick={onClick}
      ref={buttonRef}
      type={type}
    >
      <span className={styles.btnContentWrap}>
        {children}
        {icon && (
          <Icon
            icon={icon}
            iconClasses={iconOnlyBtn ? iconOnlyBtnClasses : styles.icon}
            size={size !== ButtonSizes.small ? "lg" : "sm"}
          />
        )}
      </span>
```

```
  ...props
}) => {
  return (
    <button
      {...props}
      aria-controls={ariaControls}
      aria-expanded={ariaExpanded}
      aria-label={ariaLabel}
      id={buttonId}
      className={clsx(styles.root, styles[size], buttonClasses)}
      disabled={disabled}
      onClick={onClick}
      ref={buttonRef}
      type={type}
    >
      <span className={styles.btnContentWrap}>
        {children}
        {icon && (
          <Icon
            icon={icon}
            iconClasses={iconOnlyBtn ? iconOnlyBtnClasses : styles.icon}
            size={size !== ButtonSizes.small ? "lg" : "sm"}
          />
        )}
      </span>
```

```
  ...props
}) => {
  return (
    <button
      {...props}
      aria-controls={ariaControls}
      aria-expanded={ariaExpanded}
      aria-label={ariaLabel}
      id={buttonId}
      className={clsx(styles.root, styles[size], buttonClasses)}
      disabled={disabled}
      onClick={onClick}
      ref={buttonRef}
      type={type}
    >
      <span className={styles.btnContentWrap}>
        {children}
        {icon && (
          <Icon
            icon={icon}
            iconClasses={iconOnlyBtn ? iconOnlyBtnClasses : styles.icon}
            size={size !== ButtonSizes.small ? "lg" : "sm"}
          />
        )}
      </span>
```

State & click handlers

```
export const DisclosureWidget = ({
  buttonClasses,
  buttonIcon,
  buttonId,
  buttonSize,
  buttonText,
  children,
}) => {
  const [isOpen, setIsOpen] = useState(false);

  /**
   * Control the isOpen state.
   */
  const toggleOpen = () => {
    setIsOpen(!isOpen);
  };

  return (
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
```

```
export const DisclosureWidget = ({
  buttonClasses,
  buttonIcon,
  buttonId,
  buttonSize,
  buttonText,
  children,
  defaultExpanded,
}) => {
  const [isOpen, setIsOpen] = useState(defaultExpanded);

  /**
   * Control the isOpen state.
   */
  const toggleOpen = () => {
    setIsOpen(!isOpen);
  };

  return (
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
```

```
  buttonIcon,
  buttonId,
  buttonSize,
  buttonText,
  children,
  defaultExpanded,
}) => {
  const [isOpen, setIsOpen] = useState(defaultExpanded);

  /**
   * Control the isOpen state.
   */
  const toggleOpen = () => {
    setIsOpen(!isOpen);
  };

  return (
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
        aria-controls="dwContainer"
        aria-expanded={isOpen}
        aria-label={ariaLabel}
        buttonClasses={buttonClassNames}
```

```
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
        aria-controls="dwContainer"
        aria-expanded={isOpen}
        aria-label={ariaLabel}
        buttonClasses={buttonClassNames}
        buttonId={buttonId}
        buttonRef={buttonRef}
        icon={buttonIcon}
        iconOnlyBtnClasses={iconOnlyBtnClasses}
        iconOnlyBtn={iconOnlyBtn}
        onClick={toggleOpen}
        size={buttonSize}
        type="button"
      >
        {buttonText}
      </Button>
      <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
        {isOpen && children}
      </div>
    </div>
  );
};
```

```
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
        aria-controls="dwContainer"
        aria-expanded={isOpen}
        aria-label={ariaLabel}
        buttonClasses={buttonClassNames}
        buttonId={buttonId}
        buttonRef={buttonRef}
        icon={buttonIcon}
        iconOnlyBtnClasses={iconOnlyBtnClasses}
        iconOnlyBtn={iconOnlyBtn}
        onClick={toggleOpen}
        size={buttonSize}
        type="button"
      >
        {buttonText}
      </Button>
      <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
        {isOpen && children}
      </div>
    </div>
  );
};
```

```
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
        aria-controls="dwContainer"
        aria-expanded={isOpen}
        aria-label={ariaLabel}
        buttonClasses={buttonClassNames}
        buttonId={buttonId}
        buttonRef={buttonRef}
        icon={buttonIcon}
        iconOnlyBtnClasses={iconOnlyBtnClasses}
        iconOnlyBtn={iconOnlyBtn}
        onClick={toggleOpen}
        size={buttonSize}
        type="button"
      >
        {buttonText}
      </Button>
      <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
        {isOpen && children}
      </div>
    </div>
  );
};
```

Focus management

```
  children,
  defaultExpanded,
}) => {
  const [isOpen, setIsOpen] = useState(defaultExpanded);
  const buttonRef = useRef(null);


  /**
   * Control the isOpen state.
   */
  const toggleOpen = () => {
    setIsOpen(!isOpen);
  };

  /**
   * Close dropdown and move focus back to activator button
   * when ESC key is pressed.
   */
  const onKeyUpHandler = (e) => {
    if ((e.key === "Escape" || e.keyCode === 27) && isOpen) {
      setIsOpen(false);
      buttonRef.current.focus();
```

```
};

/**
 * Close dropdown and move focus back to activator button
 * when ESC key is pressed.
 */
const onKeyUpHandler = (e) => {
  if ((e.key === "Escape" || e.keyCode === 27) && isOpen) {
    setIsOpen(false);
    buttonRef.current.focus();
  }
};

const buttonClassNames = clsx(styles.dwButton, buttonClasses);

return (
  <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
    <Button
      aria-controls="dwContainer"
      aria-expanded={isOpen}
      aria-label={ariaLabel}
      buttonClasses={buttonClassNames}
```

```
const buttonClassNames = clsx(styles.dwButton, buttonClasses);

return (
  <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
    <Button
      aria-controls="dwContainer"
      aria-expanded={isOpen}
      aria-label={ariaLabel}
      buttonClasses={buttonClassNames}
      buttonId={buttonId}
      buttonRef={buttonRef}
      icon={buttonIcon}
      iconOnlyBtnClasses={iconOnlyBtnClasses}
      iconOnlyBtn={iconOnlyBtn}
      onClick={toggleOpen}
      size={buttonSize}
      type="button"
    >
      {buttonText}
    </Button>
    <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
```

```
const buttonClassNames = clsx(styles.dwButton, buttonClasses);

return (
  <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
    <Button
      aria-controls="dwContainer"
      aria-expanded={isOpen}
      aria-label={ariaLabel}
      buttonClasses={buttonClassNames}
      buttonId={buttonId}
      buttonRef={buttonRef}
      icon={buttonIcon}
      iconOnlyBtnClasses={iconOnlyBtnClasses}
      iconOnlyBtn={iconOnlyBtn}
      onClick={toggleOpen}
      size={buttonSize}
      type="button"
    >
      {buttonText}
    </Button>
    <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
```

CSS focus states

```css
*:focus {
  outline: 0;
  outline-offset: var(--s-5);
  box-shadow: 0 0 0 var(--s-5) var(--purple-orchid);
}
```

```css
.button {
    font-weight: var(--font-weight-bold);
    border: 0;
    border-radius: var(--border-radius);
    cursor: pointer;
    display: inline-block;
    line-height: 1;
    transition: background-color 0.2s ease;

    :hover {
        background-color: var(--color-primary);
        border: 1px solid var(--color-primary);
    }

    :focus {
        outline: 0.1em solid var(--color-primary-reverse);
        outline-offset: -2px;
        box-shadow: 0 0 0 0.2rem var(--color-focus);
    }

    :active {
        outline: 0.1em solid var(--color-aquamarine-500);
```

Mouse click management

```
  buttonSize,
  buttonText,
  children,
  defaultExpanded
}) => {
  const [isOpen, setIsOpen] = useState(defaultExpanded);
  const buttonRef = useRef(null);
  const contentRef = useRef(null);

  /**
   * Closes any active widgets if the mouse is clicked outside of it.
   *
   * Sends focus to the element in the disclosure
   * container that has the firstItemRef passed into its ref prop.
   */
  useEffect(() => {
    if (isOpen) {
      document.addEventListener("mouseup", clickOutsideHandler);
      document.addEventListener("keyup", clickOutsideHandler);

    } else {
      document.removeEventListener("mouseup", clickOutsideHandler);
      document.removeEventListener("keyup", clickOutsideHandler);
```

```
        buttonRef={buttonRef}
        icon={buttonIcon}
        iconOnlyBtnClasses={iconOnlyBtnClasses}
        iconOnlyBtn={iconOnlyBtn}
        onClick={toggleOpen}
        size={buttonSize}
        type="button"
      >
        {buttonText}
      </Button>
      <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
        {isOpen && children}
      </div>
    </div>
  );
};

export default DisclosureWidget;
```

```
    }
  };

  /**
   * Close dropdown when clicking outside of focus
   * area with mouse.
   */
  const clickOutsideHandler = (e) => {
    if (
      contentRef.current.contains(e.target) ||
      buttonRef.current.contains(e.target)
    ) {
      return;
    }
    setIsOpen(false);
  };

  const buttonClassNames = clsx(styles.dwButton, buttonClasses);

  return (
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
```

```
/**
 * Closes any active widgets if the mouse is clicked outside of it.
 *
 * Sends focus to the element in the disclosure
 * container that has the firstItemRef passed into its ref prop.
 */
useEffect(() => {
  if (isOpen) {
    document.addEventListener("mouseup", clickOutsideHandler);
    document.addEventListener("keyup", clickOutsideHandler);
  } else {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  }

  return () => {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  };
}, [isOpen]);

/**
```

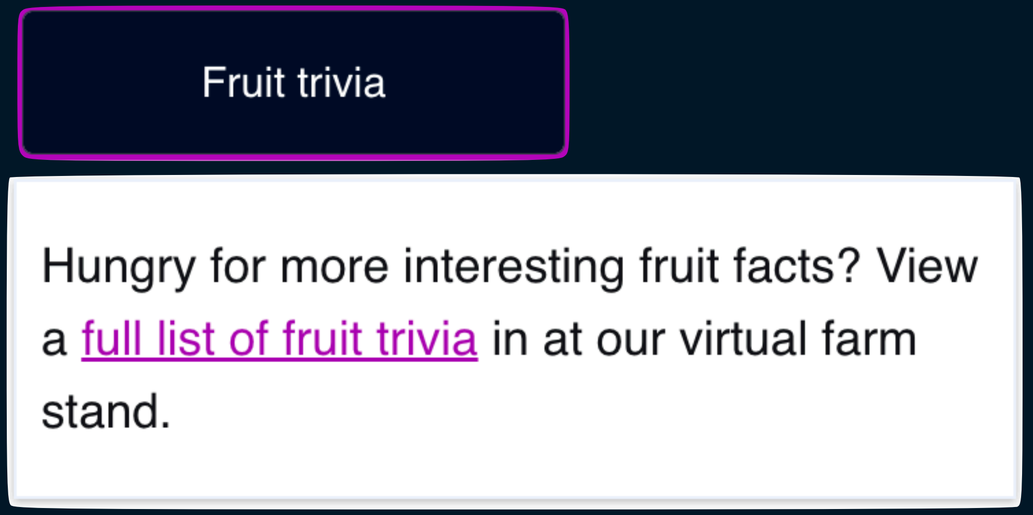
```
 * Closes any active widgets if the mouse is clicked outside of it.
 *
 * Sends focus to the element in the disclosure
 * container that has the firstItemRef passed into its ref prop.
 */
useEffect(() => {
  if (isOpen) {
    document.addEventListener("mouseup", clickOutsideHandler);
    document.addEventListener("keyup", clickOutsideHandler);

  } else {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  }

  return () => {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  };
}, [isOpen]);

/**
```

```
 * Closes any active widgets if the mouse is clicked outside of it.
 *
 * Sends focus to the element in the disclosure
 * container that has the firstItemRef passed into its ref prop.
 */
useEffect(() => {
  if (isOpen) {
    document.addEventListener("mouseup", clickOutsideHandler);
    document.addEventListener("keyup", clickOutsideHandler);

  } else {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  }

  return () => {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  };
}, [isOpen]);

/**
```

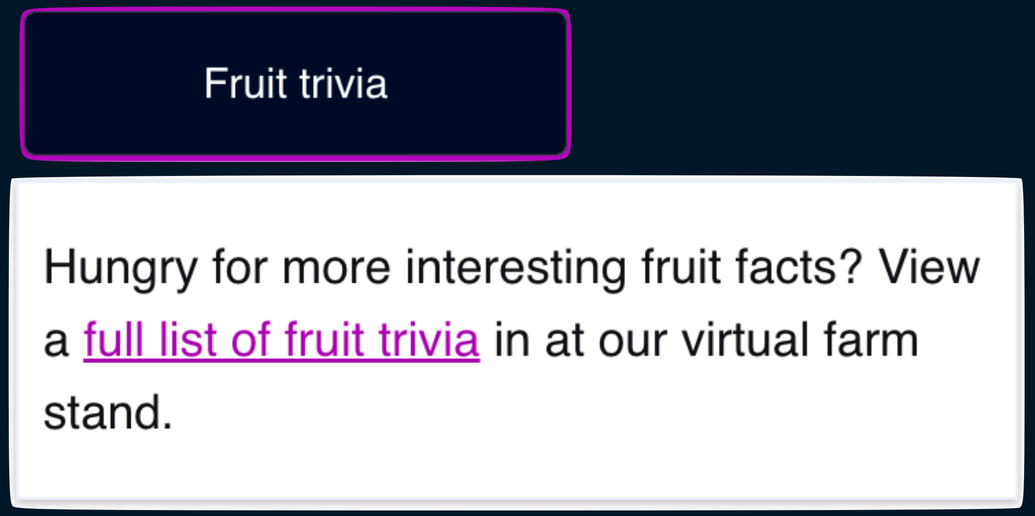
```
  /**
   * Closes any active widgets if the mouse is clicked outside of it.
   *
   * Sends focus to the element in the disclosure
   * container that has the firstItemRef passed into its ref prop.
   */
  useEffect(() => {
    if (isOpen) {
      document.addEventListener("mouseup", clickOutsideHandler);
      document.addEventListener("keyup", clickOutsideHandler);

    } else {
      document.removeEventListener("mouseup", clickOutsideHandler);
      document.removeEventListener("keyup", clickOutsideHandler);
    }

    return () => {
      document.removeEventListener("mouseup", clickOutsideHandler);
      document.removeEventListener("keyup", clickOutsideHandler);
    };
  }, [isOpen]);

  /**
```


€30
15€
18€
20€
28€
18€
€20
12€

Toggletips


Fruit trivia
Hungry for more interesting fruit facts? View a full list of fruit trivia in at our virtual farm stand.

Toggletips are NOT Tooltips

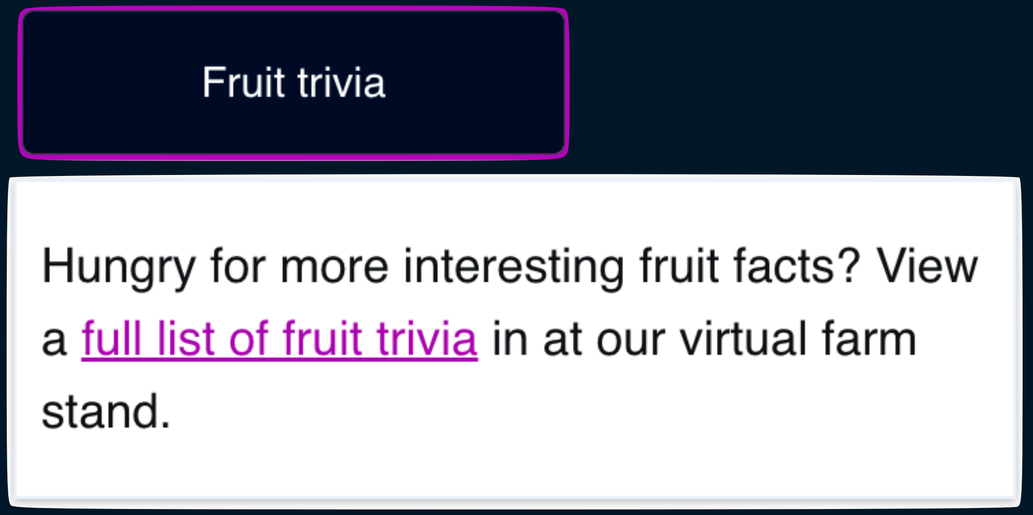


Toggletips

Contain interactive content

Supported by touch devices, non-mouse pointers, eye trackers

Toggletips


Fruit trivia
Hungry for more interesting fruit facts? View a full list of fruit trivia in at our virtual farm stand.

Toggletips

Hungry for more interesting fruit facts? View a full list of fruit trivia in at our virtual farm stand.

Icons

Informative or decorative

Informative icons need descriptive text

Decorative icons need to be hidden from assistive technology

```
return (
    <button
      {...props}
      aria-controls={ariaControls}
      aria-expanded={ariaExpanded}
      aria-label={ariaLabel}
      id={buttonId}
      className={clsx(styles.root, styles[size], buttonClasses)}
      disabled={disabled}
      onClick={onClick}
      ref={buttonRef}
      type={type}
    >









    </button>
```

```
return (
    <button
      {...props}
      aria-controls={ariaControls}
      aria-expanded={ariaExpanded}
      aria-label={ariaLabel}
      id {buttonId}
      {...(({ styles.btn, styles[size], buttonClasses)}
      {children}disabled}
      {icon && (nClick}
        <FontAwesomeIcon>
          icon={icon}
          iconClasses={iconOnlyBtn ? iconOnlyBtnClasses : styles.icon}
          size={size !== ButtonSizes.small ? "lg" : "sm"}
        />
      )}
    </span>
    </button>
```

```
  const buttonClassNames = clsx(styles.dwButton, buttonClasses);

  return (
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
        aria-controls="dwContainer"
        aria-expanded={isOpen}
        aria-label={ariaLabel}
        buttonClasses={buttonClassNames}
        buttonId={buttonId}
        buttonRef={buttonRef}
        icon={buttonIcon}
        iconOnlyBtnClasses={iconOnlyBtnClasses}
        iconOnlyBtn={iconOnlyBtn}
        onClick={toggleOpen}
        size={buttonSize}
        type="button"
      >
        {buttonText}
      </Button>
      <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
        {isOpen && children}
      </div>
    </div>
  );
};
```

```
  const buttonClassNames = clsx(styles.dwButton, buttonClasses);

  return (
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
        aria-controls="dwContainer"
        aria-expanded={isOpen}
        aria-label={ariaLabel}
        buttonClasses={buttonClassNames}
        buttonId={buttonId}
        buttonRef={buttonRef}
        icon={buttonIcon}
        iconOnlyBtnClasses={iconOnlyBtnClasses}
        iconOnlyBtn={true}
        onClick={toggleOpen}
        size={buttonSize}
        type="button"
      >
        {buttonText}
      </Button>
      <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
        {isOpen && children}
      </div>
    </div>
  );
};
```

```
  const buttonClassNames = clsx(styles.dwButton, buttonClasses);

  return (
    <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
      <Button
        aria-controls="dwContainer"
        aria-expanded={isOpen}
        aria-label={ariaLabel}
        buttonClasses={buttonClassNames}
        buttonId={buttonId}
        buttonRef={buttonRef}
        icon={buttonIcon}
        iconOnlyBtnClasses={iconOnlyBtnClasses}
        iconOnlyBtn={true}
        onClick={toggleOpen}
        size={buttonSize}
        type="button"
      >
        {buttonText}
      </Button>
      <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
        {isOpen && children}
      </div>
    </div>
  );
};
```

```
export default function ToggleTip() {
  return (
    <DisclosureWidget
      aria-label="Fruit trivia"
      buttonClasses={clsx(styles.disclosureButton, styles.toggleTipButton)}
      buttonIcon="apple-alt"
      iconClasses={clsx(styles.btnIcon)}
      iconOnlyBtn={true}
    >
      <div className={clsx(styles.floatingContainer)}>
        <p className={styles.toggleTipContainer}>
          Hungry for more interesting fruit facts? View a{" "}
          <a href="https://google.com">full list of fruit trivia</a> in at our
          virtual farm stand.
        </p>
      </div>
    </DisclosureWidget>
  );
}
```

```
export default function ToggleTip() {
  return (
    <DisclosureWidget
      aria-label="Fruit trivia"

      buttonIcon="apple-alt"

      iconOnlyBtn={true}
    >
      <div className={clsx(styles.floatingContainer)}>
        <p className={styles.toggleTipContainer}>
          Hungry for more interesting fruit facts? View a{" "}
          <a href="https://google.com">full list of fruit trivia</a> in at our
          virtual farm stand.
        </p>
      </div>
    </DisclosureWidget>
  );
}
```

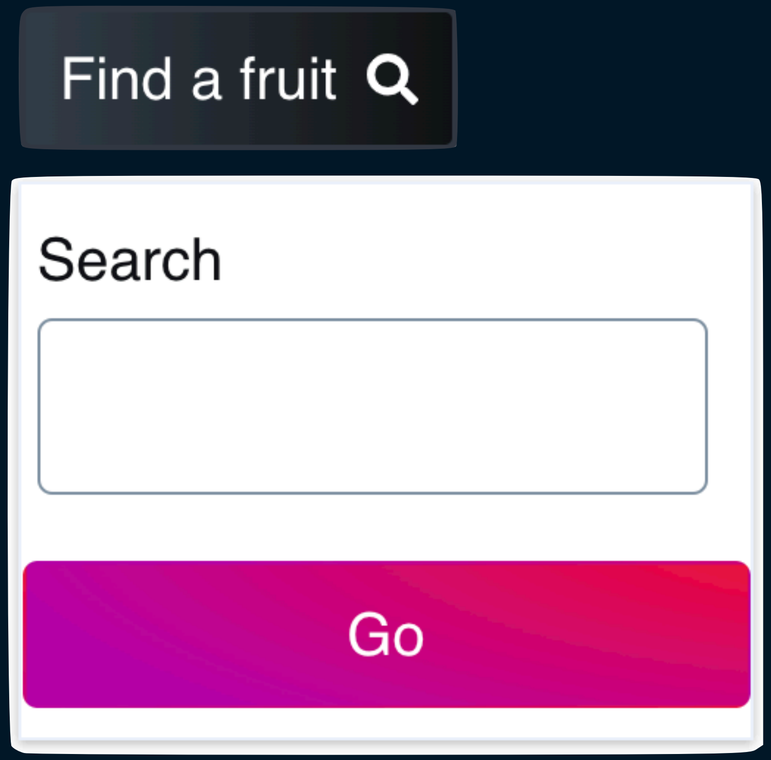
```
export default function ToggleTip() {
  return (
    <DisclosureWidget
      aria-label="Fruit trivia"
      buttonClasses={clsx(styles.disclosureButton
      iconClasses={clsx(styles.btnIcon)}
      iconOnlyBtn={true}
    >
      <div className={clsx(styles.floatingContainer)}>
        <p classNames={styles.toggleTipContainer}>
          Hungry for more interesting fruit facts? View a{" "}
          <a href="https://google.com">full list of fruit trivia</a> in at our
          virtual farm stand.
        </p>
      </div>
    </DisclosureWidget>
  );
}
```

```
export default function ToggleTip() {
  return (
    <DisclosureWidget
      aria-label="Fruit trivia"
      buttonClasses={clsx(styles.disclosureButton, styles.toggleTipButton)}
      buttonIcon="apple-alt"
      iconClasses={clsx(styles.btnIcon)}
      iconOnlyBtn={true}
    >
      <div className={clsx(styles.floatingContainer)}>
        <p className={styles.toggleTipContainer}>
          Hungry for more interesting fruit facts? View a{" "}
          <a href="https://google.com">full list of fruit trivia</a> in at our
          virtual farm stand.
        </p>
      </div>
    </DisclosureWidget>
  );
}
```

Toggletip

Hungry for more interesting fruit facts? View a full list of fruit trivia in at our virtual farm stand.

ToggleSearch

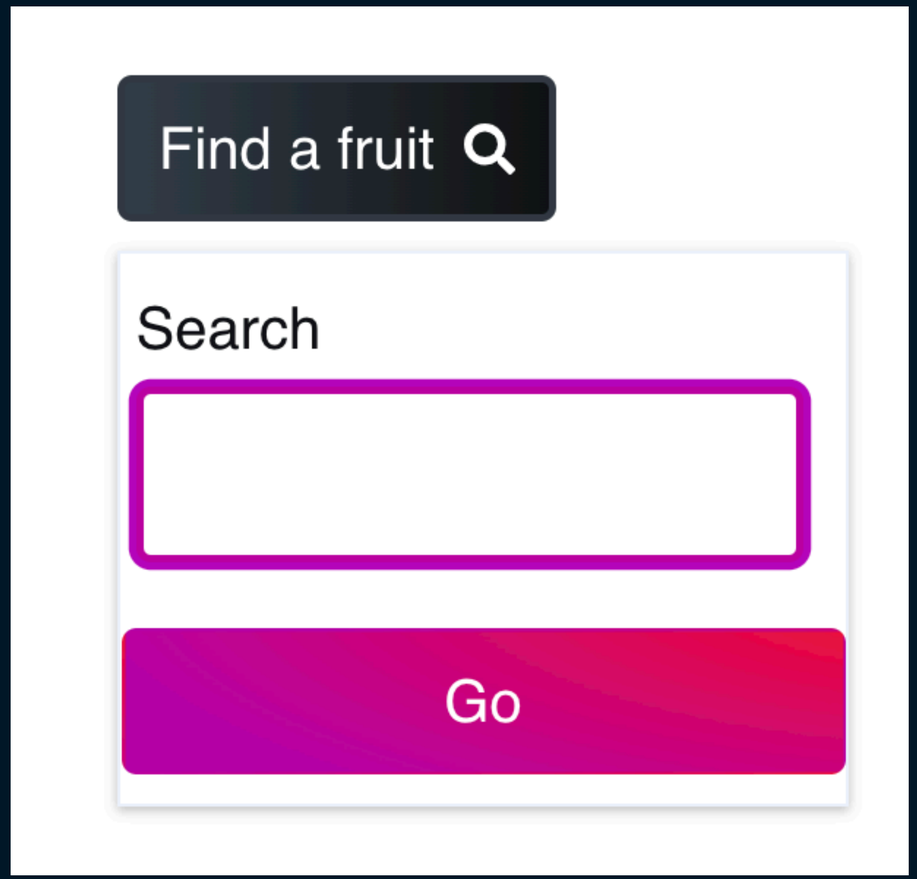


```
export default function ToggleSearch() {
  const firstItemRef = useRef(null);
  return (
    <DisclosureWidget
      firstItemRef={firstItemRef}
      buttonClasses={clsx(styles.toggleSearchButton)}
      buttonText="Find a fruit"
      buttonIcon="search"
      buttonSize="small"
    >
      <form
        onSubmit={(e) => e.preventDefault()}
        className={clsx(styles.floatingContainer, styles.disclosureNavList)}
      >
        <Input
          icon="search"
          inputRef={firstItemRef}
          label="Search"
          name="value"
          type="search"
        />
        <Button size="small">Go</Button>
      </form>
    </DisclosureWidget>
  );
}
```

```
export default function ToggleSearch() {
  const firstItemRef = useRef(null);
  return (
    <DisclosureWidget
      firstItemRef={firstItemRef}
      buttonClasses={clsx(styles.toggleSearchButton)}
      buttonText="Find a fruit"
      buttonIcon="search"
      buttonSize="small"
    >
      <form
        onSubmit={(e) => e.preventDefault()}
        className={clsx(styles.floatingContainer, styles.disclosureNavItem)}
      >
        <Input
          icon="search"
          inputRef={firstItemRef}
          label="Search"
          name="value"
          type="search"
        />
        <Button size="small">Go</Button>
      </form>
    </DisclosureWidget>
  );
}
```

```
export default function ToggleSearch() {
  const firstItemRef = useRef(null);
  return (
    <DisclosureWidget
      firstItemRef={firstItemRef}
      buttonClasses={clsx(styles.toggleSearchButton)}
      buttonText="Find a fruit"
      buttonIcon="search"
      buttonSize="small"
    >
      <form
        onSubmit={(e) => e.preventDefault()}
        className={clsx(styles.floatingContainer, styles.disclosureNavList)}
      >
        <Input
          icon="search"
          inputRef={firstItemRef}
          label="Search"
          name="value"
          type="search"
        />
        <Button size="small">Go</Button>
      </form>
    </DisclosureWidget>
  );
}
```

Targeted focus management

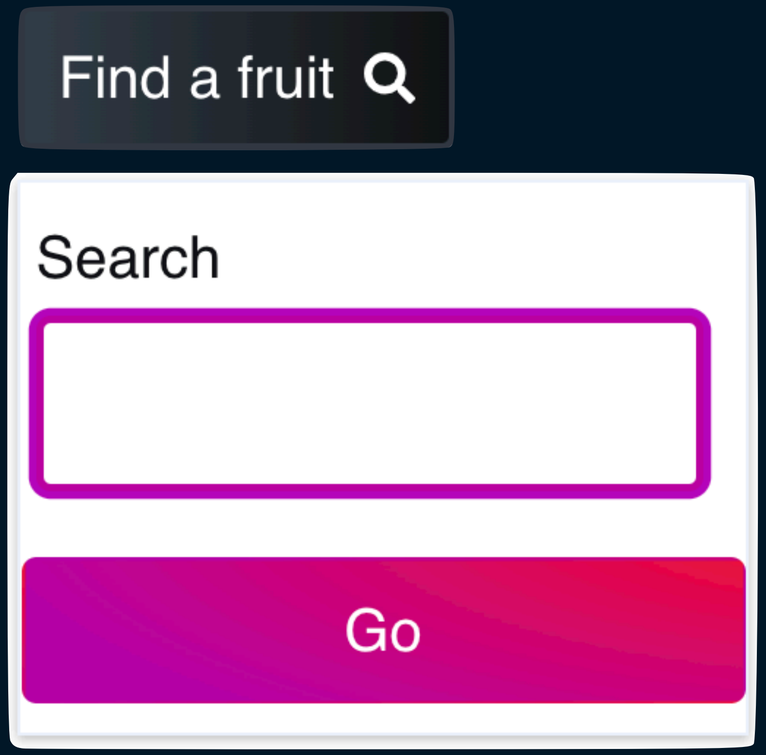


```
export default function ToggleSearch() {
  const firstItemRef = useRef(null);
  return (
    <DisclosureWidget
      firstItemRef={firstItemRef}
      buttonClasses={clsx(styles.toggleSearchButton)}
      buttonText="Find a fruit"
      buttonIcon="search"
      buttonSize="small"
    >
      <form
        onSubmit={(e) => e.preventDefault()}
        className={clsx(styles.floatingContainer, styles.disclosureNavList)}
      >
        <Input
          icon="search"
          inputRef={firstItemRef}
          label="Search"
          name="value"
          type="search"
        />
        <Button size="small">Go</Button>
      </form>
    </DisclosureWidget>
  );
}
```

```
export default function ToggleSearch() {
  const firstItemRef = useRef(null);
  return (
    <DisclosureWidget
      firstItemRef={firstItemRef}
      buttonClasses={clsx(styles.toggleSearchButton)}
      buttonText="Find a fruit"
      buttonIcon="search"
      buttonSize="small"
    >
      <form
        onSubmit={(e) => e.preventDefault()}
        className={clsx(styles.floatingContainer, styles.disclosureNavList)}
      >
        <Input
          icon="search"
          inputRef={firstItemRef}
          label="Search"
          name="value"
          type="search"
        />
        <Button size="small">Go</Button>
      </form>
    </DisclosureWidget>
  );
}
```

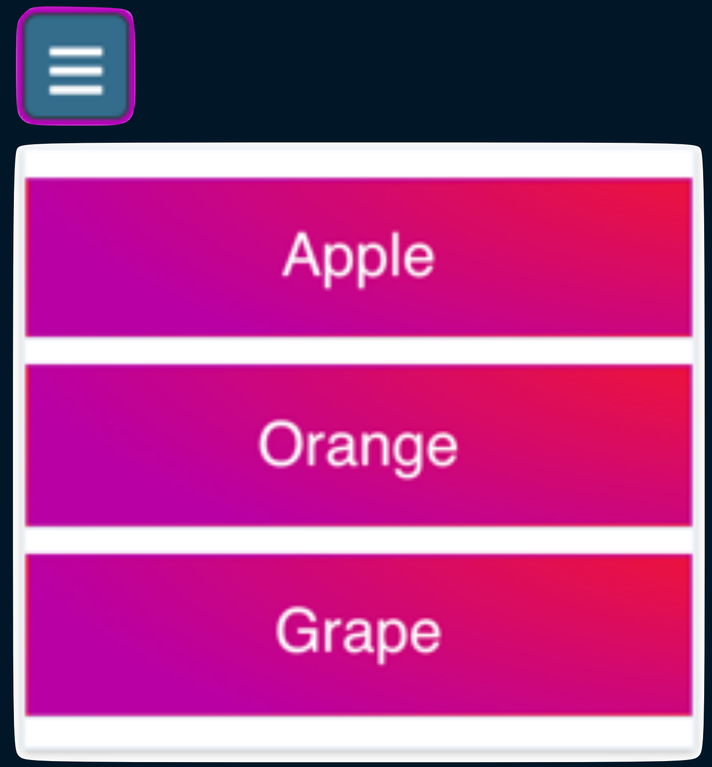
```
  */
useEffect(() => {
  if (isOpen) {
    document.addEventListener("mouseup", clickOutsideHandler);
    document.addEventListener("keyup", clickOutsideHandler);

    if (firstItemRef) {
      firstItemRef.current.focus();
    }
  } else {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  }

  return () => {
    document.removeEventListener("mouseup", clickOutsideHandler);
    document.removeEventListener("keyup", clickOutsideHandler);
  };
}, [isOpen, firstItemRef]);

/**
 * Control the isOpen state.
 */
```

Embedded search widget

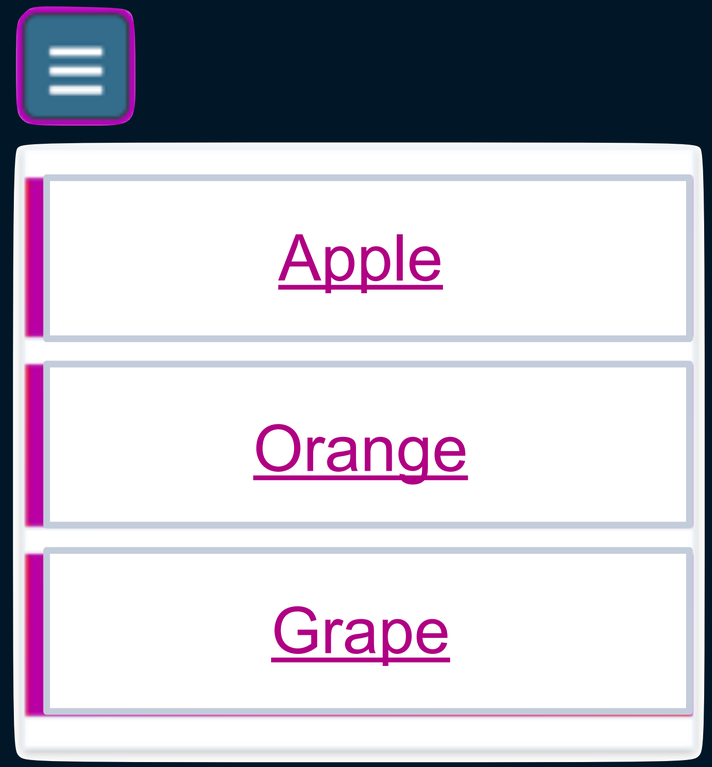


This is where it gets tricky

Hamburger navigation



Hamburger navigation



Hey

Links

Whoa

```
export default function ToggleMenuLinks() {
    return (
        <DisclosureWidget
            aria-label="fruit choices"
            buttonClasses={clsx(styles.disclosureButton, styles.chooseFruitButton)}
            buttonIcon="bars"
            iconBtnClasses={clsx(styles.btnIcon)}
            iconOnlyBtn={true}
        >
            <ul className={clsx(styles.floatingContainer, styles.disclosureMenuList)}>
                {items.map((item, i) => {
                    return (
                        <li className={styles.disclosureMenuListItem} key={i}>
                            <a href={item.link}>{item.title}</a>
                        </li>
                    );
                })}
            </ul>
        </DisclosureWidget>
    );
}
```

```
export default function ToggleMenuLinks() {
    return (
        <DisclosureWidget
            aria-label="fruit choices"
            buttonClasses={clsx(styles.disclosureButton, styles.chooseFruitButton)}
            buttonIcon="bars"
            iconBtnClasses={clsx(styles.btnIcon)}
            iconOnlyBtn={true}
        >
            <ul className={clsx(styles.floatingContainer, styles.disclosureMenuList)}>
                {items.map((item, i) => {
                    return (
                        <li className={styles.disclosureMenuListItem} key={i}>
                            <a href={item.link}>{item.title}</a>
                        </li>
                    );
                })}
            </ul>
        </DisclosureWidget>
    );
}
```

```
export default function ToggleMenuLinks() {
    return (
        <DisclosureWidget
            aria-label="fruit choices"
            buttonClasses={clsx(styles.disclosureButton, styles.chooseFruitButton)}
            buttonIcon="bars"
            iconBtnClasses={clsx(styles.btnIcon)}
            iconOnlyBtn={true}
        >
            <ul className={clsx(styles.floatingContainer, styles.disclosureMenuList)}>
                {items.map((item, i) => {
                    return (
                        <li className={styles.disclosureMenuListItem} key={i}>
                            <a href={item.link}>{item.title}</a>
                        </li>
                    );
                })}
            </ul>
        </DisclosureWidget>
    );
}
```

How about

Buttons

Whoa

```
export default function ToggleMenuButtons() {
    return (
        <DisclosureWidget
            aria-label="fruit choices"
            buttonClasses={clsx(styles.disclosureButton, styles.chooseFruitButton)}
            buttonIcon="bars"
            iconBtnClasses={clsx(styles.btnIcon)}
            iconOnlyBtn={true}
        >
            <ul className={clsx(styles.floatingContainer, styles.disclosureMenuList)}>
                {items.map((item, i) => {
                    return (
                        <li className={styles.disclosureMenuListItem} key={i}>
                            <Button>{item.link}>{item.title}</Button>
                        </li>
                    );
                })}
            </ul>
        </DisclosureWidget>
    );
}
```

```
export default function ToggleMenuButtons() {
    return (
        <DisclosureWidget
            aria-label="fruit choices"
            buttonClasses={clsx(styles.disclosureButton, styles.chooseFruitButton)}
            buttonIcon="bars"
            iconBtnClasses={clsx(styles.btnIcon)}
            iconOnlyBtn={true}
        >
            <ul className={clsx(styles.floatingContainer, styles.disclosureMenuList)}>
                {items.map((item, i) => {
                    return (
                        <li className={styles.disclosureMenuListItem} key={i}>
                            <Button>{item.title}</Button>
                        </li>
                    );
                })}
            </ul>
        </DisclosureWidget>
    );
}
```

```
export default function ToggleMenuButtons() {
    return (
        <DisclosureWidget
            aria-label="fruit choices"
            buttonClasses={clsx(styles.disclosureButton, styles.chooseFruitButton)}
            buttonIcon="bars"
            iconBtnClasses={clsx(styles.btnIcon)}
            iconOnlyBtn={true}
        >
            <ul className={clsx(styles.floatingContainer, styles.disclosureMenuList)}>
                {items.map((item, i) => {
                    return (
                        <li className={styles.disclosureMenuListItem} key={i}>
                            <Button>{item.title}</Button>
                        </li>
                    );
                })}
            </ul>
        </DisclosureWidget>
    );
}
```

Yes...

Roving Focus

Whoa

```
import { useCallback, useState, useEffect } from "react";

function useRoveFocus(length) {
    const [currentFocus, setCurrentFocus] = useState(0);

    const handleKeyDown = useCallback(
        (e) => {
            if (e.keyCode === 40) {
                // Down arrow
                e.preventDefault();
                setCurrentFocus(currentFocus === length - 1 ? 0 : currentFocus + 1);
            } else if (e.keyCode === 38) {
                // Up arrow
                e.preventDefault();
                setCurrentFocus(currentFocus === 0 ? length - 1 : currentFocus - 1);
            }
        },
        [length, currentFocus, setCurrentFocus]
    );

    useEffect(() => {
        document.addEventListener("keydown", handleKeyDown, false);
        return () => {
            document.removeEventListener("keydown", handleKeyDown, false);
        };
    }, [handleKeyDown]);

    return [currentFocus, setCurrentFocus];
}

export default useRoveFocus;
```

```
import { useCallback, useState, useEffect } from "react";

function useRoveFocus(length) {
    const [currentFocus, setCurrentFocus] = useState(0);

    const handleKeyDown = useCallback(
        (e) => {
            if (e.keyCode === 40) {
                // Down arrow
                e.preventDefault();
                setCurrentFocus(currentFocus === length - 1 ? 0 : currentFocus + 1);
            } else if (e.keyCode === 38) {
                // Up arrow
                e.preventDefault();
                setCurrentFocus(currentFocus === 0 ? length - 1 : currentFocus - 1);
            }
        },
        [length, currentFocus, setCurrentFocus]
    );

    useEffect(() => {
        document.addEventListener("keydown", handleKeyDown, false);
        return () => {
            document.removeEventListener("keydown", handleKeyDown, false);
        };
    }, [handleKeyDown]);
```

```
function useRoveFocus(length) {
    const [currentFocus, setCurrentFocus] = useState(0);

    const handleKeyDown = useCallback(
        (e) => {
            if (e.keyCode === 40) {
                // Down arrow
                e.preventDefault();
                setCurrentFocus(currentFocus === length - 1 ? 0 : currentFocus + 1);
            } else if (e.keyCode === 38) {
                // Up arrow
                e.preventDefault();
                setCurrentFocus(currentFocus === 0 ? length - 1 : currentFocus - 1);
            }
        },
        [length, currentFocus, setCurrentFocus]
    );

    useEffect(() => {
        document.addEventListener("keydown", handleKeyDown, false);
        return () => {
            document.removeEventListener("keydown", handleKeyDown, false);
        };
    }, [handleKeyDown]);
```

Yo

TagPalooza

I’m clever

@resource11

```
const ListItemWhoa = ({
    character,
    focus,
    index,
    setFocus,
    tag = "button",
    ...props
}) => {
    const ref = useRef(null);

    useEffect(() => {
        if (focus) {
            // Move element into view when it is focused
            ref.current.focus();
        }
    }, [focus]);

    const handleSelect = useCallback(() => {
        alert(`${character}`);
        // setting focus to that element when it is selected
        setFocus(index);
    }, [character, index, setFocus]);

    const Tag = `${tag}`;

    return (
        <li>
            <Tag
                {...props}
                role="menuitem"
                onClick={handleSelect}
                onKeyPress={handleSelect}
                ref={ref}
                tabIndex={focus ? 0 : -1}
            >
                {character}
            </Tag>
        </li>
    );
};

export default ListItemWhoa;
```

```
        setFocus(index);
    }, [character, index, setFocus]);

    const Tag = `${tag}`;

    return (
        <li>
            <Tag
                {...props}
                role="menuitem"
                onClick={handleSelect}
                onKeyPress={handleSelect}
                ref={ref}
                tabIndex={focus ? 0 : -1}
            >
                {character}
            </Tag>
        </li>
    );
};

export default ListItemWhoa;
```

```
        setFocus(index);
    }, [character, index, setFocus]);

    const Tag = `${tag}`;

    return (
        <li>
            <Tag
                {...props}
                role="menuitem"
                onClick={handleSelect}
                onKeyPress={handleSelect}
                ref={ref}
                tabIndex={focus ? 0 : -1}
            >
                {character}
            </Tag>
        </li>
    );
};

export default ListItemWhoa;
```

@resource11

Check out my

Fancy List

Separation of concerns, baby.

```
import React from "react";
import clsx from "clsx";
import ListItem from "./ListItem";
import useRoveFocus from "./utils/useRoveFocus";
import masks from "./data/masks";
import styles from "./css/DisclosureWidget.module.css";

const List = () => {
    const [focus, setFocus] = useRoveFocus(masks.length);
    return (
        <ul
            aria-labelledby="menubutton"
            className={clsx(styles.floatingContainer, styles.disclosureMenuList)}
        >
            {masks.map((mask, index) => (
                <ListItem
                    key={mask}
                    setFocus={setFocus}
                    index={index}
                    focus={focus === index}
                    mask={mask}
                    tag="a"
                />
            ))}
        </ul>
    );
};
export default List;
```

```
import styles from "./css/DisclosureWidget.module.css";

const List = () => {
    const [focus, setFocus] = useRoveFocus(masks.length);
    return (
        <ul
            aria-labelledby="menubutton"
            className={clsx(styles.floatingContainer, styles.disclosureMenuList)}
        >
            {masks.map((mask, index) => (
                <ListItem
                    key={mask}
                    setFocus={setFocus}
                    index={index}
                    focus={focus === index}
                    mask={mask}
                    tag="a"
                />
            ))}
        </ul>
    );
};
export default List;
```

Some

Ternary magic?

...Hey now.

@resource11

```jsx
import React, { useRef, useState, useEffect } from "react";
import PropTypes from "prop-types";
import clsx from "clsx";
import Button from "./Button";
import List from "./List";
import styles from "./css/styles.module.css";

/**
 * A control that shows or hides a container of contents.
 */

export const DisclosureWidgetJuliaChild = ({
    "aria-controls": ariaControls,
    "aria-expanded": ariaExpanded,
    "aria-label": ariaLabel,
    buttonClasses,
    buttonIcon,
    buttonId,
    buttonSize,
    buttonText,
    children,
    defaultExpanded,
    firstItemRef,
    iconOnlyBtnClasses,
    iconOnlyBtn,
    isList
}) => {
    const [isOpen, setIsOpen] = useState(defaultExpanded);
    const buttonRef = useRef(null);
    const contentRef = useRef(null);

    /**
     * This useEffect hook listens for mouse and keyboard events
     * of the active disclosure widget,
     */
    useEffect(() => {
        if (isOpen) {
            document.addEventListener("mouseup", clickOutsideHandler);
            document.addEventListener("keyup", clickOutsideHandler);

            if (firstItemRef) {
                firstItemRef.current.focus();
            }
        } else {
            document.removeEventListener("mouseup", clickOutsideHandler);
            document.removeEventListener("keyup", clickOutsideHandler);
        }

        return () => {
            document.removeEventListener("mouseup", clickOutsideHandler);
            document.removeEventListener("keyup", clickOutsideHandler);
        };
    }, [isOpen, firstItemRef]);

    /**
     * Control the isOpen state.
     */
    const toggleOpen = () => {
        setIsOpen(!isOpen);
    };

    /**
     * Close dropdown and move focus back to activator button
     * when ESC key is pressed.
     */
    const onKeyUpHandler = (e) => {
        if ((e.key === "Escape" || e.keyCode === 27) && isOpen) {
            setIsOpen(false);
            buttonRef.current.focus();
        }
    };

    /**
     * Close dropdown when clicking outsite of focus
     * area with mouse.
     */
    const clickOutsideHandler = (e) => {
        if (
            contentRef.current.contains(e.target) ||
            buttonRef.current.contains(e.target)
        ) {
            return;
        }
        setIsOpen(false);
    };

    const buttonClassNames = clsx(styles.button, buttonClasses);

    return (
        <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
            <Button
                aria-controls="dwContainer"
                aria-expanded={isOpen}
                aria-label={ariaLabel}
                buttonClasses={buttonClassNames}
                buttonId={buttonId}
                buttonRef={buttonRef}
                icon={buttonIcon}
                iconOnlyBtnClasses={iconOnlyBtnClasses}
                iconOnlyBtn={iconOnlyBtn}
                onClick={toggleOpen}
                size={buttonSize}
                type="button"
            >
                {buttonText}
            </Button>
            {isOpen && isList ? (
                <div ref={contentRef} id="dwContainer">
                    <List />
                </div>
            ) : (
                <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
                    {isOpen && children}
                </div>
            )}
        </div>
    );
};

export default DisclosureWidgetJuliaChild;

DisclosureWidgetJuliaChild.defaultProps = {
    defaultExpanded: false,
    buttonText: ""
};

DisclosureWidgetJuliaChild.propTypes = {
    children: PropTypes.node,
    className: PropTypes.string,
    defaultExpanded: PropTypes.bool,
    disclosureButton: PropTypes.node,
    firstItemRef: PropTypes.instanceOf(Object)
};
```

```
    return (
        <div onKeyUp={onKeyUpHandler} className={styles.dwWrapper}>
            <Button
                aria-controls="dwContainer"
                aria-expanded={isOpen}
                aria-label={ariaLabel}
                buttonClasses={buttonClassNames}
                buttonId={buttonId}
                buttonRef={buttonRef}
                icon={buttonIcon}
                iconOnlyBtnClasses={iconOnlyBtnClasses}
                iconOnlyBtn={iconOnlyBtn}
                onClick={toggleOpen}
                size={buttonSize}
                type="button"
            >
                {buttonText}
            </Button>

            {isOpen && isList ? (
                <div ref={contentRef} id="dwContainer">
                    <List />
                </div>
            ) : (
                <div className={styles.dwContainer} ref={contentRef} id="dwContainer">
                    {isOpen && children}
                </div>
            )}
        </div>
    );
};

export default DisclosureWidget;
```


@resource11
Maybe this is
a bad idea...

What pattern is this?

Menu and Menubar

A menu is a widget that offers a list of choices to the user, such as a set of actions or functions.

Menu Button

A menu button is a button that opens a menu as described in the Menu and Menubar Pattern.

Meter

A meter is a graphical display of a numeric value that varies within a defined range.

Radio Group

A radio group is a set of checkable buttons, known as radio buttons, where no more than one of the buttons can be checked at a time.

Slider

A slider is an input where the user selects a value from within a given range.

Slider (Multi-Thumb)

A multi-thumb slider implements the Slider Pattern but includes two or more thumbs, often on a single rail.

Spinbutton

A spinbutton is an input widget that restricts its value to a set or range of discrete values.

Switch

A switch is an input widget that allows users to choose one of two values: on or off.

Table

Like an HTML table element, a WAI-ARIA table is a static tabular structure containing one or more rows that each contain one or more cells; it is not an interactive widget.

WAI-ARIA Roles, States, and Properties

- A menu is a container of items that represent choices. The element serving as the menu has a role of either menu or menubar.
- The items contained in a menu are child elements of the containing menu or menubar and have any of the following roles:
  - menuitem
  - menuitemcheckbox
  - menuitemradio
- If activating a menuitem opens a submenu, the menuitem is known as a parent menuitem. A submenu's `menu` element is:
  - Contained inside the same `menu` element as its parent `menuitem`.
  - Is the sibling element immediately following its parent `menuitem`.
- A parent menuitem has aria-haspopup set to either `menu` or `true`.
- A parent menuitem has aria-expanded set to `false` when its child menu is not visible and set to `true` when the child menu is visible.
- One of the following approaches is used to enable scripts to move focus among

Menu and Menubar Pattern

About This Pattern

A menu is a widget that offers a list of choices to the user, such as a set of actions or functions. Menu widgets behave like native operating system menus, such as the menus that pull down from the menubars commonly found at the top of many desktop application windows. A menu is usually opened, or made visible, by activating a menu button, choosing an item in a menu that opens a sub menu, or by invoking a command, such as `Shift + F10` in Windows, that opens a context specific menu. When a user activates a choice in a menu, the menu usually closes unless the choice opened a submenu.

A menu that is visually persistent is a menubar. A menubar is typically horizontal and is often used to create a menu bar similar to those found near the top of the window in many desktop applications, offering the user quick access to a consistent set of commands.

A common convention for indicating that a menu item launches a dialog box is to append "..." (ellipsis) to the menu item label, e.g., "Save as ...".

Examples

Editor Menubar Example: Demonstrates menu radios and menu checkboxes in submenus of a menubar that provides text formatting commands for a text field.

Navigation Menubar Example: Demonstrates a menubar that provides site navigation.

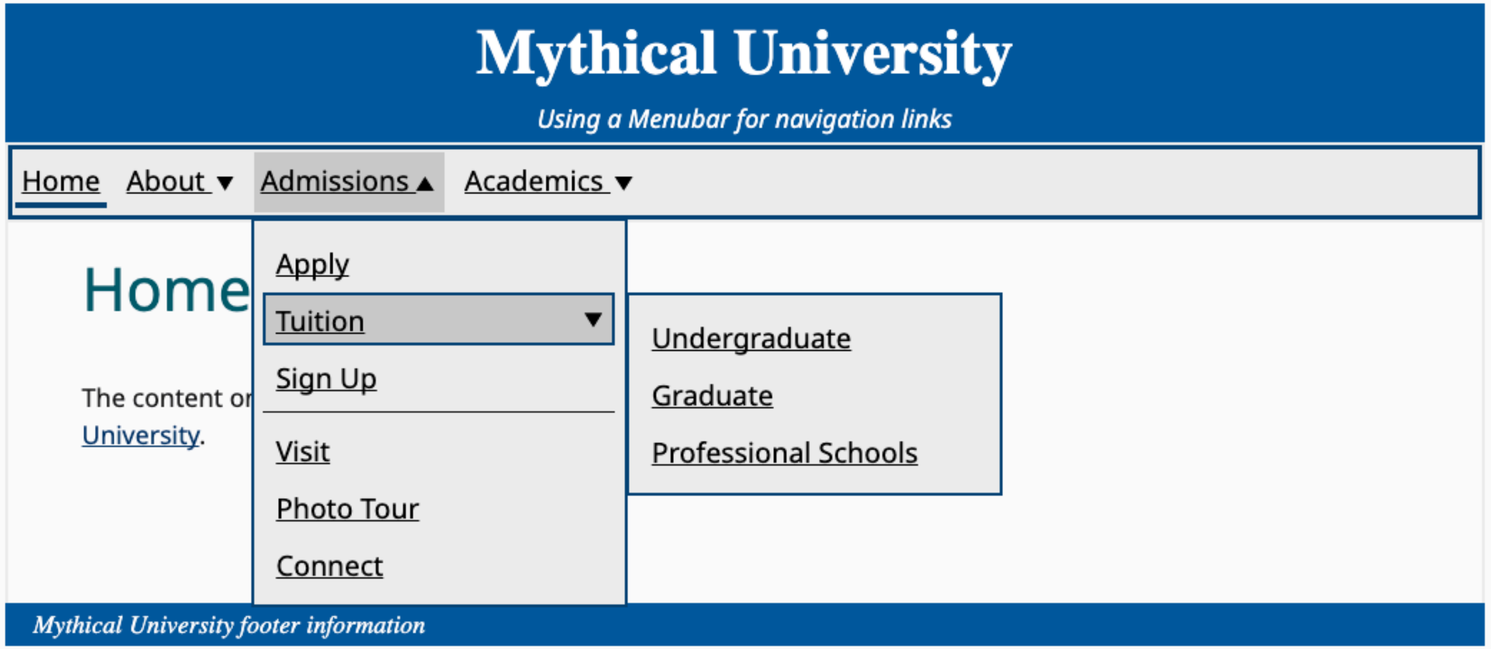
Keyboard Interaction

The following description of keyboard behaviors assumes:

1. A horizontal `menubar` containing several `menuitem`, `menuitemradio`, or `menuitemcheckbox` elements.
2. Some `menuitem` elements in the `menubar` have child submenus that contain vertically arranged items.

Example

Open In CodePen



Accessibility Features

1. Menu items that trigger navigation move focus to the target page title:
   - An important aspect of designing a navigation menu experience is where keyboard focus moves when an item that triggers navigation is activated and the menu closes. If activating a menubar item changes content on the page without triggering a browser page load, i.e., works like typical single-page apps, the focus position after the content load significantly effects efficiency for keyboard and assistive technology users.
   - This example behaves like a single page app and activating a menu item that loads new content moves focus to the beginning of the new content, which is a level one heading with content that matches the name of the activated menu item. Focusing on the heading informs screen reader users that navigation is complete and confirms the destination.
   - To view other pages, keyboard users need to navigate back to the menubar . To optimize keyboard efficiency, this example locates the menubar immediately before the content display area in the Tab sequence.
2. To help communicate that the arrow keys are available for directional navigation within the menubar and its submenus, a border is added to the menubar container when focus is within the menubar.

Page Contents

Menu and Menubar

A menu is a widget that offers a list of choices to the user, such as a set of actions or functions.

Menu Button

A menu button is a button that opens a menu as described in the Menu and Menubar Pattern.

Meter

A meter is a graphical display of a numeric value that varies within a defined range.

Radio Group

A radio group is a set of checkable buttons, known as radio buttons, where no more than one of the buttons can be checked at a time.

Slider

A slider is an input where the user selects a value from within a given range.

Slider (Multi-Thumb)

A multi-thumb slider implements the Slider Pattern but includes two or more thumbs, often on a single rail.

Spinbutton

A spinbutton is an input widget that restricts its value to a set or range of discrete values.

Switch

A switch is an input widget that allows users to choose one of two values: on or off.

Table

Like an HTML table element, a WAI-ARIA table is a static tabular structure containing one or more rows that each contain one or more cells; it is not an interactive widget.

↑ Back to Top

Menu Button Pattern

About This Pattern

A menu button is a button that opens a menu as described in the Menu and Menubar Pattern. It is often styled as a typical push button with a downward pointing arrow or triangle to hint that activating the button will display a menu.

Examples

Action Menu Button Example Using aria-activedescendant: A button that opens a menu of actions or commands where focus in the menu is managed using aria-activedescendant.

Action Menu Button Example Using element.focus(): A menu button made from an HTML `button` element that opens a menu of actions or commands where focus in the menu is managed using `element.focus()`.

Navigation Menu Button: A menu button made from an HTML `a` element that opens a menu of items that behave as links.

Keyboard Interaction

- With focus on the button:
  - `Enter`: opens the menu and places focus on the first menu item.
  - `Space`: Opens the menu and places focus on the first menu item.
  - (Optional) `Down Arrow`: opens the menu and moves focus to the first menu item.
  - (Optional) `Up Arrow`: opens the menu and moves focus to the last menu item.
- The keyboard behaviors needed after the menu is open are described in the Menu and Menubar Pattern.

Disclosure (Show/Hide) Pattern

About This Pattern

A disclosure is a widget that enables content to be either collapsed (hidden) or expanded (visible). It has two elements: a disclosure button and a section of content whose visibility is controlled by the button. When the controlled content is hidden, the button is often styled as a typical push button with a right-pointing arrow or triangle to hint that activating the button will display additional content. When the content is visible, the arrow or triangle typically points down.

Examples

Disclosure (Show/Hide) of Image Description

Disclosure (Show/Hide) of Answers to Frequently Asked Questions

Disclosure (Show/Hide) Navigation Menu

Disclosure (Show/Hide) Navigation Menu with Top-Level Links

Keyboard Interaction

When the disclosure control has focus:

- `Enter`: activates the disclosure control and toggles the visibility of the disclosure content.
- `Space`: activates the disclosure control and toggles the visibility of the disclosure content.

WAI-ARIA Roles, States, and Properties

- The element that shows and hides the content has role button.
- When the content is visible, the element with role `button` has aria-expanded set to `true`. When the content area is


@resource11
Whoa Nellie...


@resource11


@resource11
Don’t mix
the masks

Paolo Nicolello — Unsplash

Lorenzo Dominici — Unsplash

A solution is here

Open UI

The purpose of the Open UI, a W3C Community Group, is to to allow web developers to style and extend built-in web UI components and controls, such as `<select>` dropdowns, checkboxes, radio buttons, and date/color pickers.

To do that, we'll need to fully specify the component parts, states, and behaviors of the built-in controls, as well as necessary accessibility requirements, and provide test suites to ensure compatibility. We'll also implement polyfills for our extensible web UI controls.

Today, component frameworks and design systems reinvent common web UI controls to give designers full control over their appearance and behavior. We hope to make it unnecessary to reinvent built-in UI controls, but for those who choose to do so, we expect that these design systems will benefit from Open UI's specifications and test suites.

Long term, we hope that Open UI will establish a standard process for developing high-quality UI controls suitable for addition to the web platform.

For complete outline of goals and process view our full charter.

Contribute

Open UI wouldn't be possible without the support of our awesome community! Get involved on Discord and Github.

popover

HTML

Living Standard — Last Updated 4 April 2024

← 6.11 Drag and drop — Table of Contents — 7 Loading web pages →

6.12 The popover attribute

All HTML elements may have the **popover** content attribute set. When specified, the element won't be rendered until it becomes shown, at which point it will be rendered on top of other page content.

Example

The popover attribute is a global attribute that allows authors flexibility to ensure popover functionality can be applied to elements with the most relevant semantics.

The following demonstrates how one might create a popover sub-navigation list of links, within the global navigation for a website.


@resource11


@resource11

Wrap-up

@resource11

Design-systems
are a

Carnival!

Our users are diverse

Vision Hearing Motor Cognitive

Sensory Language Low bandwidth


@resource11
Don’t mix
the masks

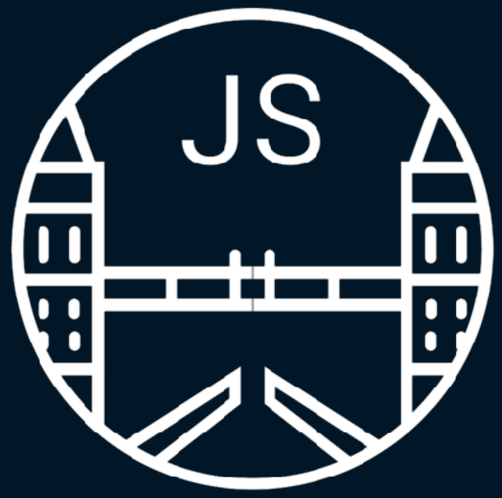

@resource11


@resource11


@resource11


@resource11
Design Systems
are ALWAYS
the hotness

@resource11



Thank you.

https://noti.st/resource11/

Slide deck posted after the talk

@resource11

Twitter | Instagram | GitHub