Web 講義 5 ReactJS

基本的に https://ja.react.dev/ を参照すること。

React の機能

・コンポーネントの再利用
・インタラクティブな機能(ボタンなどの動作を指す)

課題 1

公式サイトのチュートリアル「三目並べ」をやってみよう。

コンポーネントの作成

・app/components フォルダを作成する。
Profile コンポーネントを作成してみよう。
・app/components/Profile.tsx ファイルを作成する。
大文字で作成するのが流儀。
(基本形)

```
type Props={

}
export const Profile:FC<Props> = () => {
    return(<></>)
}
```

(Profile.tsx)

```
type Props={
    name:string;
    age:number;
    bio:string
}

export const Profile:FC<Props> =
({name,age,bio}) => {
    return(<div>
        <p>{name},{age}</p>
        <p>{bio}</p>
    </div>)
}
```

・コンポーネントの利用
(page.tsx)

```
export default function Page() {
  return (<main>
    <Profile/>
  </main>)
}
```

ただし、このままだと Props が渡せていない。

```
export default function Page() {
  return (<main>
    <Profile name="灘校太郎" age={21} bio="
よろ"/>
    <Profile name="灘校次郎" age={22} bio="
よろ"/>
    <Profile name="灘校三郎" age={23} bio="
よろ"/>
  </main>)
}
```

このようにコンポーネントを再利用できる。

課題 2

Tailwind の知識を生かして Profile コンポーネントのレイアウトを作成してみよう。思いつかない人は以下のレイアウトを再現してみよう。

課題 3

公式サイトの「UI の記述」を読み、実践しよう。

リストのレンダリング(rendering)

JavaScript ではリストは配列や Map を指す。繰り返し処理をして、Profile の表示を簡略化してみよう。

```
const people = [
  {
    name: "灘校太郎",
    age: 20,
    bio: "僕は灘校です。"
  },
  {
    name: "灘校次郎",
    age: 18,
    bio: "僕は次郎です よろしくね"
  },
  {
    name: "灘校三郎",
    age: 16,
    bio: "僕は三郎です。"
  }
]

export default function Page() {
  return (<main>
    {
      people.map((person, index) =>
        <Profile key={index}
          name={person.name}
          age={person.age}
          bio={person.bio} />)
    }
  </main>)
}
```

課題 5

フルーツの配列「["リンゴ","オレンジ"…]」を作成し、それらについて「(フルーツ)大好き」という p タグの文章を生成してみよう。

React コンポーネントの変数

次のようなコンポーネントを考えてみる。

(components/Counter.tsx)

```
type Props={}
export const Counter:FC<Props>=()=>{
    const [num,setNum]=useState(0)

    return(<div>
        <p>Counter</p>
        <p>{num}</p>
        <button
onClick={()=>setNum(num+1)}>+</button>
    </div>)
}
```

```
export const Counter:FC<Props>=()=>{
    let num=0
    return(<div>
        <p>Counter</p>
        <p>{num}</p>
        <button onClick={()=>{
            num+=1
        }}>+</button>
    </div>)
}
```

課題 6

二つの場合で実際に動作を確認しよう。
Point: JSX 内で使用する変数は状態管理をする必要がある。

Counter コンポーネントを次のように書き換えてみよう。

```
export const Counter:FC<Props>=()=>{
    const [num,setNum]=useState(0)
    useEffect(()=>{
        document.title=`Counter: ${num}`
    },[num])
```

```
    return(<div>
        <p>Counter</p>
        <p>{num}</p>
        <button
onClick={()=>setNum(num+1)}>+</button>
    </div>)
}
```

課題 7

useEffect について調べてみよう。

発展

```
export default function Page() {
  return (<main>
    <Timer/>
  </main>)
}

const Timer:FC=()=>{
  const now=new Date()
  return(
    <p>{now.toString()}</p>
  )
}
```

Test コンポーネントを分析してみると now 変数を宣言し、現在の時間を p タグとして返している。

課題 8

ブラウザで動作を確認しよう。再更新のショートカットキーは Ctrl+R である。

このように、Test コンポーネントは最初の描画時しか更新されない。更新するためには値が変わったことを React に通知する必要がある。

実際に毎秒更新するのは少し難しいためコードサンプルだけ置いておく。

```
export const Timer: FC = () => {
  const [now, setNow] = useState(new
Date())
  const interval = useRef<NodeJS.Timeout>()

  useEffect(() => {
    interval.current = setInterval(() => {
      setNow(new Date())
    }, 1000)
    return () => {
      clearInterval(interval.current)
    }
  }, [])

  return (
    <p>{now.toString()}</p>
  )
}
```

課題 9

useRef フックについて調べてみよう。

このようにフックを使うことでたくさんの処理ができる。また、フックを自作していくこともできるので、興味のある人はやってみよう。

例えば、chakra-ui というライブラリの中に useToast というフックがある。
公式サイトを見てみよう。