Props i React med TypeScript

- **Props (properties)** är ett sätt att **skicka data** från en **'förälder'-komponent** till en **'barn'-komponent** i React.
- Props ska betraktas som **read-only** – en komponent ska inte ändra sina egna props.
- De skickas till komponenter som **attribut** i JSX.

Exempel: Props i React (utan TypeScript)

```
const Welcome = (props) => {
  return <h1>Hej, {props.name}!</h1>;
};
```

```
const App = () => {
  return <Welcome name="Håkan" />;
};
```

- Här får `Welcome`-komponenten **en prop** (`name`) från `App`-komponenten.

Props med TypeScript

- I TypeScript kan vi **typa props** med **type** eller **interface**.
- Detta gör koden **säkrare** och förhindrar oväntade typer.

Exempel: Typade props i React med TypeScript

```tsx
//kod i Welcome.tsx
type WelcomeProps = {
  name: string;
};

const Welcome = ({ name }: WelcomeProps) => {
  return <h1>Hej, {name}!</h1>;
};

export default Welcome;
```

```tsx
//kod i App.tsx
import Welcome from './components/Welcome'

const App = () => {
  return <Welcome name="Håkan" />;
};
```

Typ och de-structuring

- `WelcomeProps` definierar att `name` **måste** vara en `string`.
- Props 'de-structas' direkt i funktionsparametern.

Flera props och valfria props

- Man kan skicka flera props och även göra vissa props **valfria** med `?`.

```
type WelcomeProps = {
  name: string;
  age?: number; // Valfri prop
};

const Welcome = ({ name, age }: WelcomeProps) => {
  if (age) {
    return <h1>Hej, {name}! Du är {age} år gammal.</h1>;
  } else {
    return <h1>Hej, {name}!</h1>;
  }
};
```

I föräldrakomponenten

- Anropa med eller utan `age`

```
const App = () => {
  return (
    <>
      <Welcome name="Håkan" age={53} />
      <Welcome name="Kasper" />
    </>
  );
};
```

Props med standardvärden

- Man kan sätta **defaultvärden** för props om de saknas.

Exempel: Standardvärde för props

```
type WelcomeProps = {
  name: string;
  age?: number;
};

const Welcome = ({ name, age = 30 }: WelcomeProps) => {
  return <h1>Hej, {name}! Du är {age} år gammal.</h1>;
};

const App = () => {
  return <Welcome name="Håkan" />; // age blir 30 som standard
};
```

- `age = 30` sätts som **standardvärde** om `age` inte skickas med.

Använda Children i React

- `children` är en **inbyggd prop** i React som låter en komponent **omsluta** andra komponenter eller innehåll.
- `children` kan typas olika beroende på vad den ska innehålla.

Exempel: Children med `React.ReactNode`

```
import { ReactNode } from "react";
type Props = {
  children: React.ReactNode;//allmän typ
};

const Page = ({ children }: Props) => (
  <div>
    {children}
  </div>
);

const App = () => (
  <Page>
    <p>Detta är en paragraf inuti Page-komponenten.</p>
  </Page>
);
```

- HTML-content i `children`

Exempel: Children som endast en sträng

Om vi vill begränsa `children` till endast text kan vi typa det som `string`:

```
type TitleProps = {
  children: string;
};

const Title = ({ children }: TitleProps) => (
  <h1>{children}</h1>
);

const App = () => (
  <Title>Detta är en titel</Title>
);
```

Här kan `children` endast vara en sträng, inget annat.

Använda Event-handlers med Props i React

- Event-handlers kan skickas som props och typas i TypeScript.
- För att få rätt typ, håll muspekaren över event-attributet (t.ex. `onClick`) i VS Code.

Exempel: Knapp med `onClick`-prop

```
type ButtonProps = {
  onClick: (event: React.MouseEvent<HTMLButtonElement>) => void;
};

const Button = ({ onClick }: ButtonProps) => (
  <button onClick={onClick}>
    Klicka här
  </button>
);

const App = () => (
  <Button onClick={(event) => console.log("Knappen klickades!", event)} />
);
```

Här skickas en `onClick`-funktion direkt i props som en arrow function.