Expressions

Where React comes alive!

tl;dr

- If intelligent processing of JSX is needed as opposed to static JSX declarations, we include JavaScript expressions inside curly braces ("{" and "}")
- These allow us to run JavaScript inside our JSX to ...
  - conditionally display elements
  - display multiple elements by iterating an array
- If what you're trying to do is too complex, you can always call a function that outputs JSX

Say we're reading a list of people ...

people.json

```
{
"name": { "first": "maëlia", "last": "dupuis" },
"email": "maëlia.dupuis@example.com",
"cell": "06-76-31-32-56",
"picture": { "large": "md65.jpg"}
},
{
"name": { "first": "susanne", "last": "scott" },
"email": "susanne.scott@example.com",
"cell": "081-007-7340",
},
{
"name": { "first": "babür", "last": "çörekçi" },
"email": "babür.çörekçi@example.com",
"cell": "(743)-870-9450",
"picture": { "large": "bc65.jpg" }
}
```

ListPeople.js

```
export function ListPeople(props) {
  const people = props.people;
  return (
    <section>
      <ul>
        <li>
          <img src={p.img} />
          {p.first} {p.last}
        </li>
      </ul>
    </section>
  )
}
```

... and displaying it

This code won't work.
How do you enumerate the list?
How do you conditionally display?

Hey! We could run some JavaScript inside the JSX! Then we could use conditionals, loops, and call functions!

Expressions are JavaScript inside of JSX

- We will need to run some JavaScript inside of the JSX

Expressions are ... well ... JavaScript expressions

ListPeople.js

```
export function ListPeople(props) {
  const [p]= props.people;
  return (
    <section>
      <ul>
        <li>
          <img src={p.img} />
          {p.first} {p.last}
        </li>
      </ul>
    </section>
  )
}
```

You can add JavaScript to JSX if you put it in curly braces

Expressions must be a <u>single</u> JavaScript expression

- ... not a statement
- ... not a block
- ... not an assignment
- ... not a line of code

Generally, expressions are something you'd find on the right side of an "="

An expression evaluates to a <u>single</u> thing which is then substituted back into the JSX

Not allowed:

- if (foo === bar) doIt();
- while (foo === bar) doIt();
- function () { doIt(); }
- foo = bar
- expr1 ; expr2

- JSX expressions must be just that ... expressions as opposed to statements. One per set of curly braces. In them you can reference variables, use operators, and call functions (Thus, Rap, JSX can be used in methods other than the render method. Cool.

ListPeople.js

```
export function ListPeople(props) {
 const people = props.people;
 return (
   <section>
     { people.length ? <People /> : null }
     <ul>
       <li>
         <img src={p.img} />
         {p.first} {p.last}
       </li>
     </ul>
   </section>
 )
}
```

Expressions can contain JSX

And that JSX can have an expression,

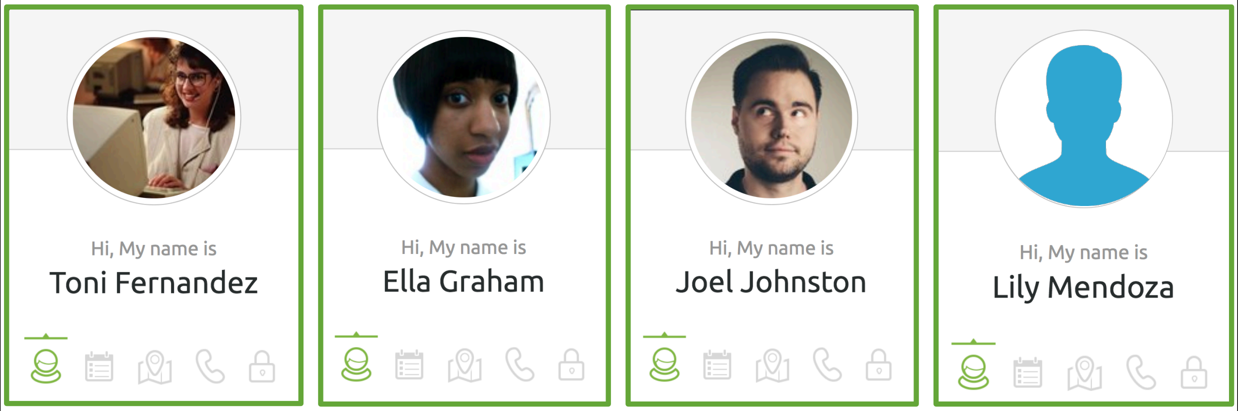
which can have more JSX,
which can have an expression,
which can have more JSX,
which can have an expression,
which can have more JSX,
which can have an expression,
which can have more JSX,
which can have an expression,
which can have more JSX,
which can have an expression,
which can have more JSX,
which can have an expression,
... ad nauseum

JSX is super useful for ...

- Conditional rendering
- Looping
- Calling functions

Conditional rendering

Say ListPeople.js has a photo



- If the user has a photo, show it. If not, show a generic placeholder image.

ListPeople.js

```
export function ListPeople(props) {
 const [p] = props.people;
 const ph = "/img/placeholder.jpg";
 return (
  <section>
   <img src={if (p.img) p.img else ph} />
   <div>
    <p>Hi, my name is</p>
    <p>{p.first} {p.last}</p>
   </div>
  </section>
  )
}
```

We can't do this

ListPeople.js

```
export function ListPeople(props) {
 const [p] = props.people;
 const ph = "/img/placeholder.jpg";
 return (
  <section>
   <img src={p.img ? p.img : ph} />
   <div>
    <p>Hi, my name is</p>
    <p>{p.first} {p.last}</p>
   </div>
  </section>
  )
}
```

A ternary will work

ListPeople.js

```
export function ListPeople(props) {
 const [p] = props.people;
 const ph = "/img/placeholder.jpg";
 return (
  <section>
   <img src={p.img || ph} />
   <div>
    <p>Hi, my name is</p>
    <p>{p.first} {p.last}</p>
   </div>
  </section>
  )
}
```

short-circuiting will work

ListPeople.js

```
export function ListPeople(props) {
 const [p] = props.people;
 const ph = "/img/placeholder.jpg";
 return (
  <section>
   {p.img && <img src={p.img} />}
   <div>
    <p>Hi, my name is</p>
    <p>{p.first} {p.last}</p>
   </div>
  </section>
  )
}
```

or short-circuiting will work

- If we don't have an image, just don't put anything in the JSX

Looping

Remember, this is an expression. A single statement.

- while is not a single statement
- for is not a single statement

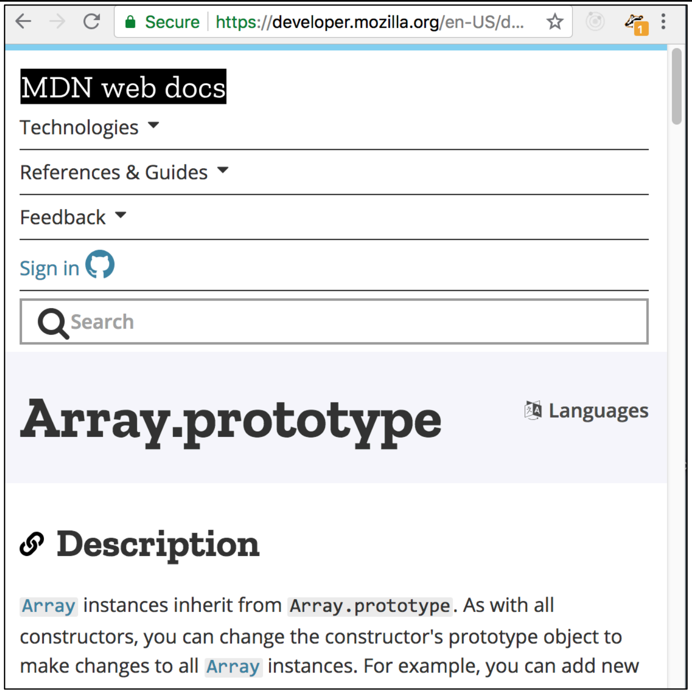
ListPeople.js

```
export function ListPeople(props) {
 return (
  <section>
   {for (let p of props.people)
    <Person person={p} />
   }
  </section>
 )
}
```

We can't do this

But there are a bunch of JavaScript Array.prototype.* methods that will iterate an array and operate on each thing

- concat()
- filter()
- flat()
- flatMap()
- join()
- slice()



Array.prototype

Description

Array instances inherit from Array.prototype. As with all constructors, you can change the constructor's prototype object to make changes to all Array instances. For example, you can add new

Array.prototype.map will return a new array of elements, each having been transformed by the function.

```
const olderPeople = people.map(
 person => person.age += 10);
```

So ... if you return JSX from map, you get a list of JSX elements

ListPeople.js

```
export function ListPeople(props) {
 return (
  <section>
   {props.people.map((p) => {
    return <Person person={p} />
   })}
   {/* Or more concisely... */}
   {props.people.map(
      p => <Person person={p} />)}
  </section>
 )
}
```

.map() will work great!

What happened to change this list?

from this … … to this

John Stamos
Bob Saget
Dave Coulier
Mary-Kate Olsen
Candace Cameron

1. △ "Bob" to "Dave"
2. △"Dave" to "Mary-Kate"
3. △ "Mary-Kate" to Candace
4. △"Candace" to "Bob"

John Stamos
Dave Coulier
Mary-Kate Olsen
Candace Cameron
Bob Saget

How can it keep track of what really happened?

How about now?

from this … … to this

102	John Stamos
334	Bob Saget
721	Dave Coulier
395	Mary-Kate Olsen
412	Candace Cameron

102	John Stamos
334	Dave Coulier
721	Mary-Kate Olsen
395	Candace Cameron
412	Bob Saget

It needs a unique ID for each VD element.

Tip: Do not use index

- Using index is easy and it makes the warning message go away.

```
{people.map((person, index) =>
    <Person
       {...person}
       key={index}
       />
   )
}
```

- But when you do this, the keys aren't consistent from render to render.

Now, we've talked about conditionals and looping but we had a 3rd time when expressions are super useful ...

Calling functions

Calling functions

A function call is a single expression

- If the logic you want is too complex for a single expression, you can call a function
- Functions can be as complex as you like!

If it's too complex to be a single expression, but a whole function seems like overkill, remember that an iife is a single expression!

The function must return JSX

ListPeople.js

```
export function ListPeople(props) {
 return <section>
   {getSomePeople(props.people)}
  </section>
}
function getSomePeople(people) {
 const ppl = [];
 for (let p of people) {
  p.birthdate === today && ppl.push(p);
  if (p.name.city.startsWith("New"))
   ppl.push(p);
 }
 return ppl.map(p => <Person pers={p} />)
}
```

tl;dr

- If intelligent processing of JSX is needed as opposed to static JSX declarations, we include JavaScript expressions inside curly braces ("{" and "}")
- These allow us to run JavaScript inside our JSX to ...
  - conditionally display elements
  - display multiple elements by iterating an array
- If what you're trying to do is too complex, you can always call a function that outputs JSX