


# ifElse

Using ifElse in functional pipelines (3 min. read)


We're used to seeing `if` statements like this

```
const hasAccess = true;




if (hasAccess) {
  console.log('Access granted.');
```



```
} else {
  console.log('Access denied.');
```








```
}

  
```

An increasingly popular alternative is the ternary statement.

```
const hasAccess = true;
const message = hasAccess ? 'Access granted.' : 'Access denied.';

console.log(message);
```






Ramda provides `ifElse`, letting you handle branching logic with functions.

```
import { ifElse } from 'ramda';

const hasAccess = true;

const logAccess = ifElse(
  () => hasAccess,
  () => console.log('Access granted.'),
  () => console.log('Access denied.')
);

logAccess();
```

One advantage is that you can package the logic away into a function. Instead of hardcoding the `hasAccess` variable, make it a parameter.

```
import { ifElse } from 'ramda';

const logAccess = ifElse(
  (hasAccess) => hasAccess,
  () => console.log('Access granted.'),
  () => console.log('Access denied.')
);

logAccess(true);
```

And instead of the `console.log` side-effect, purify it by simply returning your desired result.

```
import { ifElse } from 'ramda';

const logAccess = ifElse(
  (hasAccess) => hasAccess,
  () => 'Access granted.',
  () => 'Access denied.'
);

const result = logAccess(true);

console.log({ result });
```

This makes a point-free style easier to achieve.

```
import { always, equals, ifElse } from 'ramda';

const logAccess = ifElse(
  equals(true),
  always('Access granted.'),
  always('Access denied.')
);

const result = logAccess(true);

console.log({ result });
```



And the end result's easier to test!