cond

Using cond in functional pipelines (5 min. read)

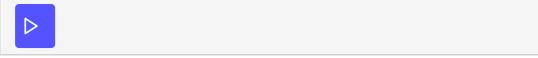
Sometimes we have too many conditions, making switch statements a great choice.

We *could* mimic this with pipe and when, but coding for a default case is tough.

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

const findAnimal = pipe(
  when(equals('lion'), always('Africa and India')),
  when(equals('tiger'), always('China, Russia, India, Vietnam, and many more')),
  when(equals('hyena'), always('African Savannah')),
  when(equals('grizzly bear'), always('North America')),
);

console.log(findAnimal('cow'));
```





Or we can use **cond**, which is **built** into languages like Lisp. It takes an array of **if/then** statements, which are arrays themselves.

The first function is the predicate, and the second function is what to run if the predicate returns true.

Here's an example

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

const findAnimal = cond([
    [equals('lion'), always('Africa and India')],
    [equals('tiger'), always('China, Russia, India, Vietnam, and many more')],
    [equals('hyena'), always('African Savannah')],
    [equals('grizzly bear'), always('North America')]
]);

console.log(findAnimal('lion'));
```

It runs through each array. If the array's first function returns true, the array's second function is called and the logic's cut off.

But how do we support the default case?

```
default:
return 'Not sure, try Googling it!';
```

Have a function that always returns true at the very end. If nothing else runs, it will succeed and be called!

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

const findAnimal = cond([
    [equals('lion'), always('Africa and India')],
    [equals('tiger'), always('China, Russia, India, Vietnam, and many more')],
    [equals('hyena'), always('African Savannah')],
    [equals('grizzly bear'), always('North America')],
    [always(true), always('Not sure, try Googling it!')]
]);

console.log(findAnimal('cow'));
```







