# Pipe

You already compose functions. Pipe lets you compose by flowing them left-to-right, improving readability. (4 min. read)

So far we’ve learned about HOFs, data-last functions, currying, and point-free style. What do they all tie back to?

**Function Composition**.

Their entire existence is to make composition easier for us. That fact becomes even more obvious when your functions perform multiple steps involving data.

## Back to Bobo

Let’s revisit the first exercise from our **Function Composition** section: uppercase and reverse Bobo’s name. A broken-up solution would look something like this

```
const getFirstName = (user) => {...}
const uppercaseString = (string) => {...}
const reverseString = (string) => {...}

const upperAndReverseFirstName = (user) => {
  const name = getFirstName(user);
  const uppercasedName = uppercaseString(name);

  return reverseString(uppercasedName);
};
```

See how we need those intermediate variables to track the result as it goes through our function?

```
more complex === more variables
```

The more steps your function has, the more of those variables you’ll need. It becomes noisy and limits how declarative your code can be.

You could try nesting them, but I think that looks worse

```
const upperAndReverseFirstName = (user) => (
  reverseString(uppercaseString(getFirstName(user)));
);
```

How about this?

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

const upperAndReverseFirstName = pipe(
  getFirstName,
  uppercaseString,
  reverseString
);

const result = upperAndReverseFirstName({
  firstName: 'Bobo'
});

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

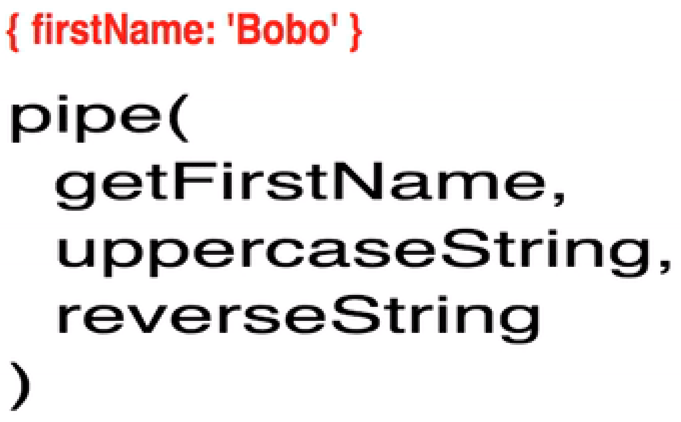
`pipe` lets you compose functions from **left-to-right**, structuring them like a sequence of events. The leftmost function runs first, then passes its output to the next function and so on.

The result is a list that closely resembles the spec you'd write up when brainstorming a solution

1. Get the first name
2. Uppercase it
3. Reverse it

```
pipe(
  getFirstName,
  uppercaseString,
  reverseString
)
```

See the resemblance? Here's an animation of that code, with Bobo running through it.



```
{ firstName: 'Bobo' }

pipe(
  getFirstName,
  uppercaseString,
  reverseString
)
```

Bobo running through the pipe

Here’s another interactive example. We’re transforming a number.

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

const doMath = pipe(
  // double
  (x) => x * 2,

  // triple
  (x) => x * 3,

  // square
  (x) => x * x,

  // increment
  (x) => x + 1
);

const result = doMath(2);

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