




Introduction

Currying

Currying

Technique to transform function with multiple arguments into a series of functions that each take single argument



```
function add(a) {  
  return function(b) {  
    return a + b;  
  };  
}  
  
const add5 = add(5);  
console.log(add5(10)); // Outputs: 15
```

Benefits

1. Modularity

Break down functions into smaller, reusable pieces.

2. Partial Application

Create new functions by pre-filling some arguments.

Example

In this example, `logWithPrefix` is a curried function that allows you to create specialized loggers by pre-filling the prefix argument.

```
function logWithPrefix(prefix) {  
  return function(message) {  
    console.log(`${prefix}: ${message}`);  
  };  
}  
  
const infoLogger = logWithPrefix('INFO');  
const errorLogger = logWithPrefix('ERROR');  
  
infoLogger('This is an informational message.');
```

Improves code readability and reusability.

Currying in React

Event Handlers

Lets create a custom hook to store data from form.

```
import React, { useState } from 'react';

function useForm() {
  const [formData, setFormData] = useState({});

  const updateField = (field) => (value) => {
    setFormData((prev) => ({ ...prev, [field]: value }));
  };

  return [formData, updateField];
}
```

currying
function

Currying in React

Event Handlers

Using the custom hook

```
function FormComponent() {  
  const [formData, updateField] =  
    useForm();  
  
  return (  
    <>  
      <input onChange={(e) =>  
updateField('name')(e.target.value)} />  
      <input onChange={(e) =>  
updateField('email')(e.target.value)}  
      />  
    </>  
  );  
}
```

Using the
function





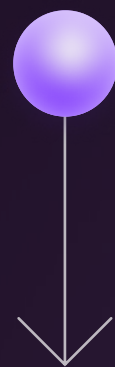
Conclusion

Currying is a powerful technique that can help you to break down functions and achieve reusability, you can write cleaner, more maintainable code.

Lets hear your thoughts

Have you used currying in your projects?

Share your insights on
comments



Let's learn from each other!