

Types of Functions

in JavaScript

(Check **Caption**)

first class
function

higher order
function

first order
function

unary
function

pure
function

currying
function

I am looking
for a frontend
developer
role, reach out
to me

First Class Function

In Javascript, functions are first class objects.

First-class functions means when functions in that language are treated like any other variable.

In such a language, a function

- can be passed as an argument to other functions,
- can be returned by another function and
- can be assigned as a value to a variable.

For example, in the below example, handler function assigned to a listener.

```
const handler = () =>
  console.log("Follow Sunil Vishwakarma");

document.addEventListener("click", handler);
```

swipe →

Higher Order Function

Higher-order function is a function that accepts another function as an argument or returns a function as a return value or both.

```
const firstOrderFunc = () =>
  console.log("Follow Sunil Vishwakarma");

const higherOrder = (ReturnFirstOrderFunc) =>
  ReturnFirstOrderFunc();

higherOrder(firstOrderFunc);
```

swipe —→

First Order Function

First-order function is a function that doesn't accept another function as an argument and doesn't return a function as its return value.


```
const firstOrder = () =>
  console.log("Follow Sunil Vishwakarma");
```

swipe —→

Unary Function

Unary function (i.e. monadic) is a function that accepts exactly one argument. It stands for a single argument accepted by a function.

Let us take an example of unary function,

```
const unaryFunction = (a) => console.log(a + 10);  
// Add 10 to the given argument and display the value
```

swipe →

Pure Function

A Pure function is a function where the return value is only determined by its arguments without any side effects. i.e, If you call a function with the same arguments 'n' number of times and 'n' number of places in the application then it will always return the same value.

Let's take an example to see the difference between pure and impure functions,

```
//Impure
let numberArray = [];
const impureAddNumber = (number) => numberArray.push(number);
//Pure
const pureAddNumber = (number) => (argNumberArray) =>
  argNumberArray.concat([number]);

//Display the results
console.log(impureAddNumber(6)); // returns 1
console.log(numberArray); // returns [6]
console.log(pureAddNumber(7)(numberArray)); // returns [6, 7]
console.log(numberArray); // returns [6]
```

swipe →

Currying Function

Currying is the process of taking a function with multiple arguments and turning it into a sequence of functions each with only a single argument.

Currying is named after a mathematician Haskell Curry. By applying currying, a n-ary function turns it into a unary function.

Let's take an example of n-ary function and how it turns into a currying function,

```
const multiArgFunction = (a, b, c) => a + b + c;
console.log(multiArgFunction(1, 2, 3)); // 6

const curryUnaryFunction = (a) => (b) => (c) => a + b + c;

curryUnaryFunction(1);
// returns a function: b => c => 1 + b + c

curryUnaryFunction(1)(2);
// returns a function: c => 3 + c

curryUnaryFunction(1)(2)(3);
// returns the number 6
```

SUNIL VISHWAKARMA
@linkinsunil

Thats a Wrap!

If you liked it, visit my profile and checkout for other
short and easy explanations

Context API vs Redux-Toolkit

Feature ▾	Context API ▾	Redux-Toolkit ▾
State Management	Not a full-fledged state management tool. Passes down values and update functions, but does not have built-in ability to store, get, update, and notify changes in values.	A full-fledged state management tool with built-in ability to store, get, update, and notify changes in values.
Usage	Best for passing static or infrequently updated values and moderately complex state that does not cause performance issues when passed using props.	Best for managing large-scale, complex state that requires asynchronous actions and side-effects.
Code Complexity	Minimal setup and low learning curve. However, can become complex when used with a large number of components and nested Contexts.	
Performance	Can cause unnecessary re-renders if the state passed down is not simple and can require the use of additional memoization techniques to optimize performance.	
Developer Tools	Does not come with pre-built developer tools but can be used with third-party tools like React DevTools.	
Community	Has a large and active community.	

React

Virtual DOM

useRef()
referencing values in React

When you want a component to remember some information, but you don't want that information to trigger new renders, you can use a ref.

Lets See into

1. How to add a ref to component?
2. How to update a ref's value?
3. How refs are different from state?
4. When to use refs?
5. Best practices for using refs?

{ Current }

⚠ Please Like & Share for no reason

VS

TF is a Virtual DOM?

real DOM

swipe →

JavaScript Evolution

ES6 ES2015

1. let and const
2. Arrow functions
3. Default parameters
4. Rest and spread operators
5. Template literals
6. Destructuring assignment
7. Classes and inheritance
8. Promises for asynchronous programming
9. Symbols for creating unique object keys
10. Iterators and generators

ES9 ES2018

1. Object.getOwnPropertyDescriptors()
2. Spread syntax for objects
3. Promise.prototype.finally()

ES10 ES2019

1. Array.prototype.flat()
2. Array.prototype.flatMap()
3. String.prototype.trimStart()
4. String.prototype.trimEnd()
5. Array.prototype.sort() (stable)

ES11 ES2020

1. BigInt
2. Nullish coalescing operator (??)
3. Optional chaining operator (?)
4. Promise.allSettled()

ES12 ES2021

1. String.prototype.replaceAll()
2. Logical assignment operators (|=, &=&, ??=)

ES13 ES2022

1. Array.prototype.lastIndexOf()
2. Object.hasOwn()
3. at() for strings and arrays
4. Top level await()

share

Redux Toolkit
Easiest Explanation Ever

React Redux Toolkit

swipe →

like and share

Instagram icon @linkinsunil LinkedIn icon @linkinsunil Twitter icon @officialskv

P.S.

Repost this if you
think your followers
will like it



Enjoyed this?

1. Follow me
2. Click the  notification
3. Never miss a post