JS

8 Types of JS Functions

Named Functions

- Named function is the function that we define it in the code and then call it whenever we need it by referencing its name and passing some arguments to it.
- Named functions are useful if we need to call a function many times to pass different values to it or run it several times.

Example

function name

```
function isEven(num) {
  return num % 2 === 0;
}
```

calling using function name

```
console.log(isEven(24)); // => true
console.log(isEven(11)); // => false
```

Arrow Functions

In ES6, arrow functions provide a shorthand syntax for defining functions.
Here we do not use the “function” keyword and use the arrow symbol.

Example

```
// Traditional function expression
const add = function(a, b) {
  return a + b;
};
console.log(add(2, 3)); // 5

// Arrow function
const add = (a, b) => {
  return a + b;
};
console.log(add(2, 3)); // 5

// Arrow function with implicit return
const add = (a, b) => a + b;

console.log(add(2, 3)); // 5
```

defined without function keyword and with => notation

Anonymous function

The anonymous functions don't have names.
They need to be tied to something: variable or an event to run.

Example

function without name assigned to a varibale

```
const add = function(a, b) {
  return a + b;
};
```

calling using variable name

```
console.log(add(2, 3)); // 5
```

Immediately invoked function expression(IIFE)

- IIFEs are functions that are executed immediately upon definition.
- They help create private scopes and module patterns, preventing variables from leaking into the global scope.

Example

```
(function() {
  console.log('IIFE executed!');
})();
```

(); represents IIFE which we dont need to call expilicitly using function name

```
(() => {
  console.log('IIFE executed!');
})();
```

Callback Functions

Functions passed as arguments to other functions, commonly used in asynchronous operations.

Example

```
// function
function greet(name, callback) {
    console.log('Hi' + ' ' + name);
    callback();
}

// callback function
function callMe() {
    console.log('I am callback function');
}

// passing function as an argument
greet('Coder Aishya', callMe);

-----------------------------------------------------------------------
OUTPUT:
Hi Coder Aishya
I am callback function
```

Higher-Order Functions

Functions that accept other functions as arguments or return functions.

some examples of Higher Order functions are map(), filter(), reduce()

Example

```
const numbers = [1, 2, 3, 4, 5];

const doubledNumbers = numbers.map(function(number) {
  return number * 2;
});

console.log(doubledNumbers);
// Output: [2, 4, 6, 8, 10]
```

Generator Functions

Functions that can be paused and resumed, using the function* syntax and yield keyword.

Example

```
function* generateSequence() {
  yield 1;
  yield 2;
  yield 3;
}

const generator = generateSequence();
console.log(generator.next().value); // 1
console.log(generator.next().value); // 2
console.log(generator.next().value); // 3
```

Async Functions

Functions that return a Promise and use await to pause execution until the Promise is resolved.

Example

```
async function fetchDataAsync() {
  let data = await fetch('https://api.example.com/data');
  data = await data.json();
  return data;
}

fetchDataAsync().then(data => {
  console.log(data);
});
```