Core JavaScript Mastery Guide

Great question 👌 — **Types of Functions in JavaScript** is a core topic, but many people only learn the "surface-level" types. Since you want **expert-level understanding**, I'll break this into:

1. **Based on Definition Style** (how functions are declared/written)
2. **Based on Behavior & Purpose** (how they work internally)
3. **Special/Advanced Function Types** (unique features in JS)

🧠 Types of Functions in JavaScript

1️⃣ Based on Definition Style

(a) Function Declaration (a.k.a. Function Statement)

```
js

function greet(name) {
  return "Hello " + name;
}
console.log(greet("Anoop"));
```

✔️ **Hoisted fully** → can be called before definition.
✔️ Has its own `this` and `arguments`.

(b) Function Expression

```
js

const greet = function(name) {
  return "Hi " + name;
};
console.log(greet("Anoop"));
```

❌ Not hoisted with body.
👉 Variable is hoisted as `undefined`.

(c) Anonymous Function

Function without a name → usually used in callbacks.

```js
setTimeout(function() {
  console.log("Runs after 2 sec");
}, 2000);
```

⚠ Not reusable, only executed where used.

(d) Named Function Expression

Function expression but with a name.

```js
const factorial = function fact(n) {
  if (n <= 1) return 1;
  return n * fact(n - 1);
};
console.log(factorial(5)); // 120
```

✔ Useful for recursion inside expressions.

(e) Arrow Function (ES6+)

```js
const greet = (name) => "Hello " + name;
```

✔ Shorter syntax
✔ Lexical `this` (inherits from parent scope)
❌ No `arguments` object

(f) Immediately Invoked Function Expression (IIFE)

Executes immediately after creation.

```js
(function() {
  console.log("I run instantly!");
})();
```

✔️ Used for private scope & avoiding global pollution.

(g) Function Constructor

Functions can be created dynamically using `new Function()`.

```js
const sum = new Function("a", "b", "return a + b");
console.log(sum(2, 3)); // 5
```

⚠ Rarely used (like `eval`, not recommended).

2️⃣ Based on Behavior & Purpose

(a) Pure Functions

Always give the same output for the same input, with no side effects.

```js
function square(x) {
  return x * x;
}
```

✔️ Predictable, testable.

(b) Impure Functions

Depend on or modify external state.

```js
```

```
let count = 0;
function increment() {
  return ++count;
}
```

⚠ Can cause unexpected behavior in large programs.

(c) Callback Functions

Functions passed as arguments to other functions.

```
js

function processUser(name, callback) {
  console.log("Processing " + name);
  callback();
}
processUser("Anoop", () => console.log("Done!"));
```

(d) Higher-Order Functions

Functions that take functions as arguments OR return functions.

```
js

function multiplier(factor) {
  return function(num) {
    return num * factor;
  };
}
const double = multiplier(2);
console.log(double(5)); // 10
```

(e) Recursive Functions

A function that calls itself.

```
js
```

```
function factorial(n) {
  if (n <= 1) return 1;
  return n * factorial(n - 1);
}
console.log(factorial(5)); // 120
```

(f) Generator Functions

Special functions that can pause and resume (`yield`).

```
js

function* counter() {
  yield 1;
  yield 2;
  yield 3;
}
const gen = counter();
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
```

(g) Async Functions (ES8)

Return a Promise, allow use of `await`.

```
js

async function fetchData() {
  let response = await fetch("https://jsonplaceholder.typicode.com/posts/1");
  return response.json();
}
```

3️⃣ Special/Advanced Types

(a) Constructor Functions (for objects)

```
js
```

```
function Person(name) {
  this.name = name;
}
const p1 = new Person("Anoop");
console.log(p1.name);
```

✔️ Used before ES6 `class`.

(b) Methods

Functions attached to objects.

```
js

const obj = {
  greet() {
    console.log("Hello from object");
  }
};
obj.greet();
```

(c) Getters and Setters

Special methods for accessing/modifying properties.

```
js

const user = {
  firstName: "Anoop",
  lastName: "Yadav",
  get fullName() {
    return this.firstName + " " + this.lastName;
  }
};
console.log(user.fullName); // "Anoop Yadav"
```

(d) Closures (Functions with Preserved Scope)

```js
function outer() {
  let count = 0;
  return function inner() {
    return ++count;
  };
}
const counter = outer();
console.log(counter()); // 1
console.log(counter()); // 2
```

✔️ Keeps memory alive.
⚠️ Can cause memory leaks if misused.

📊 Summary Table

Category	Examples
Definition Style	Declaration, Expression, Anonymous, Named, Arrow, IIFE, Constructor
Behavior & Purpose	Pure, Impure, Callback, Higher-order, Recursive, Generator, Async
Special	Constructor functions, Methods, Getters/Setters, Closures

---

👉 So, we have **3 levels of classification**:

- By *how they're written*
- By *what they do*
- By *special use cases*

Do you want me to **draw a visual mind map (diagram)** of all these function types for easier recall, or should I prepare **real-world use cases & tricky code challenges** for each type?