Core JavaScript Mastery Guide

Perfect! Let's do a **deep dive into Named Function Expressions (NFE)** — one of the trickiest topics for interviews.

1️⃣ Named Function Expression — Definition

A **Named Function Expression (NFE)** is a function expression that **has a name**, different from the variable it is assigned to:

```
js

const greet = function sayHello(name) {
  console.log("Hello " + name);
};
```

- `greet` → the variable used to call the function.
- `sayHello` → internal name **only visible inside the function**.

2️⃣ Syntax

Basic Example

```
js

const factorial = function fact(n) {
  if(n <= 1) return 1;
  return n * fact(n-1); // recursion using internal name
};

console.log(factorial(5)); // 120
```

- Here `fact` is the internal name.
- `factorial` is the variable you use externally.

Key Points

1. **Internal Name Visibility**

```js
const greet = function sayHi(name) {
  console.log(sayHi); // function itself
};
console.log(sayHi); // ❌ ReferenceError
```

- Only accessible **inside** the function.

2. **Recursive Calls**

- NFEs are very useful for **recursion**, even if the function is assigned to another variable.

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

3. **Stack Traces**

- Debugging is easier because **error messages show the internal name** (`factorial`) instead of `anonymous`.

3️⃣ Differences vs Function Expressions

Feature	Function Expression	Named Function Expression
Name	Optional	Mandatory internal name
Scope of Name	None (if anonymous)	Only inside the function
Recursion	Use variable name	Can use internal name
Stack Trace	“anonymous”	Shows internal name
Hoisting	Variable hoisted (value not)	Same behavior

4️⃣ Tricky Examples (Interview Level)

Example 1: Recursion inside NFE

```
js

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

- Works even if the variable `f` is reassigned:

```
js

const f = function factorial(n) {
  if(n <= 1) return 1;
  return n * factorial(n-1);
};

const g = f;
f = null;

console.log(g(5)); // 120 ✅ uses internal name `factorial`
```

Example 2: Name visibility

```
js

const greet = function sayHi(name) {
  console.log("Hello " + name);
  console.log(sayHi); // works inside
};
greet("Anoop"); // Hello Anoop
console.log(sayHi); // ❌ ReferenceError
```

Example 3: Named Expression in Callback

```
js
```

```
[1, 2, 3].forEach(function process(n) {
  console.log(n * 2);
});
```

- `process` can be used **inside the function for recursion** if needed.
- Outside → undefined.

5️⃣ Key Interview Points

1. NFEs are **mostly used for recursion or debugging**.
2. Internal name is **scoped inside the function only**.
3. Helps avoid problems if the variable is **reassigned externally**.
4. Can combine with `var/let/const` assignment:

```
js

var f = function factorial(n) {...};
const g = function factorial(n) {...};
```

5. **Hoisting behaves like normal function expressions** — the variable is hoisted but value is undefined.

✅ Summary

- **Anonymous Function Expression** → no internal name.
- **Named Function Expression** → internal name exists and scoped **inside** the function.
- NFEs are crucial for **recursion**, **reassignment safety**, and **debugging**.

If you want, next we can **deep dive into Anonymous Functions** with tricky callbacks, closures, and IIFE examples — where most interview questions appear.

Do you want me to go into **Anonymous Functions** next?