What the hell is this?

By Sahil Kapoor
Front-End Developer at Wizikey
@invalidtoken

Agenda

- Basic definition of this
- Implicit and explicit bindings
- Hard Binding
- (Spoiler Alert) A better definition of this
- Where to go from here?
- Questions

A function's this refers to the object that was used to execute that function*

```javascript
let particle = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: function () {
    return { x: this.x, y: this.y, z: this.z };
  },
};

console.log(particle.sendCoordinates()); // ?
```

```javascript
let particle = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: function() {
    return { x: this.x, y: this.y, z: this.z };
  },
};

console.log(particle.sendCoordinates());
// { x: 20, y: 40, z: 60 }
```

Thread of execution

```
let particle = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: function () {
    return {
      x: this.x,
      y: this.y,
      z: this.z,
    };
  },
};
console.log(particle.sendCoordinates());
// { x: 20, y: 40, z: 60 }
```

Memory

Thread of execution

```
let particle = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: function () {
    return {
      x: this.x,
      y: this.y,
      z: this.z,
    };
  },
};
console.log(particle.sendCoordinates());
// { x: 20, y: 40, z: 60 }
```

Memory

Particle
{
x : 20

}

Thread of execution

```
1  let particle = {
2    x: 20,
3    y: 40,
4    z: 60,
5    sendCoordinates: function () {
6      return {
7        x: this.x,
8        y: this.y,
9        z: this.z,
10     };
11   },
12 };
13 console.log(particle.sendCoordinates());
14 // { x: 20, y: 40, z: 60 }
```

Memory

Particle
{
x : 20
y : 40

}

Thread of execution

```
1  let particle = {
2    x: 20,
3    y: 40,
4    z: 60,
5    sendCoordinates: function () {
6      return {
7        x: this.x,
8        y: this.y,
9        z: this.z,
10     };
11   },
12 };
13 console.log(particle.sendCoordinates());
14 // { x: 20, y: 40, z: 60 }
```

Memory

Particle
{
x : 20
y : 40
z : 60

}

Thread of execution

```
let particle = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: function () {
    return {
      x: this.x,
      y: this.y,
      z: this.z,
    };
  },
};
console.log(particle.sendCoordinates());
// { x: 20, y: 40, z: 60 }
```

Memory

Particle
{
x : 20
y : 40
z : 60
sendCoordinates :
}

function ( ) {

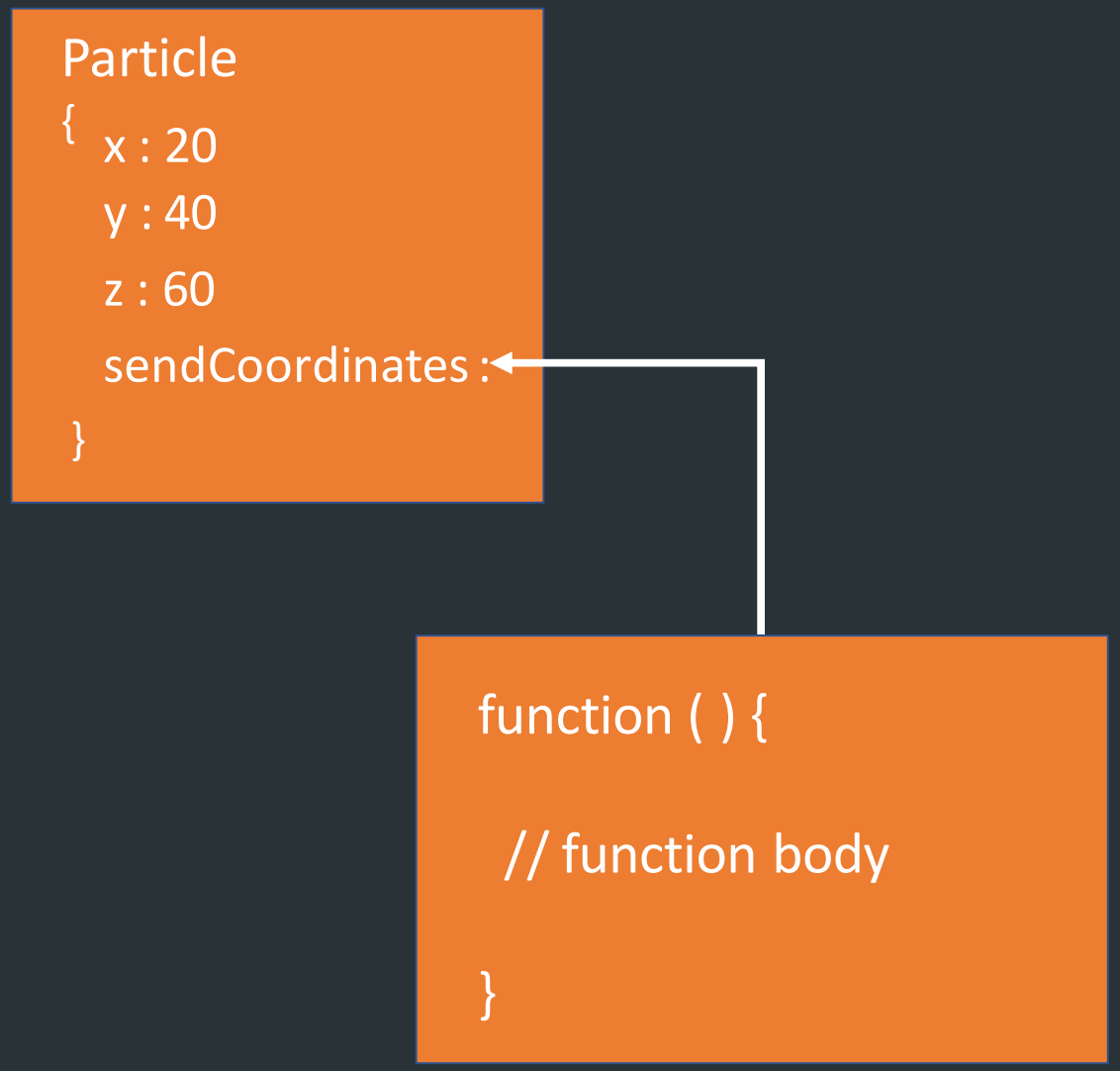
// function body

}

Thread of execution

```
1  let particle = {
2    x: 20,
3    y: 40,
4    z: 60,
5    sendCoordinates: function () {
6      return {
7        x: this.x,
8        y: this.y,
9        z: this.z,
10     };
11   },
12 };
13 console.log(particle.sendCoordinates());
14 // { x: 20, y: 40, z: 60 }
```

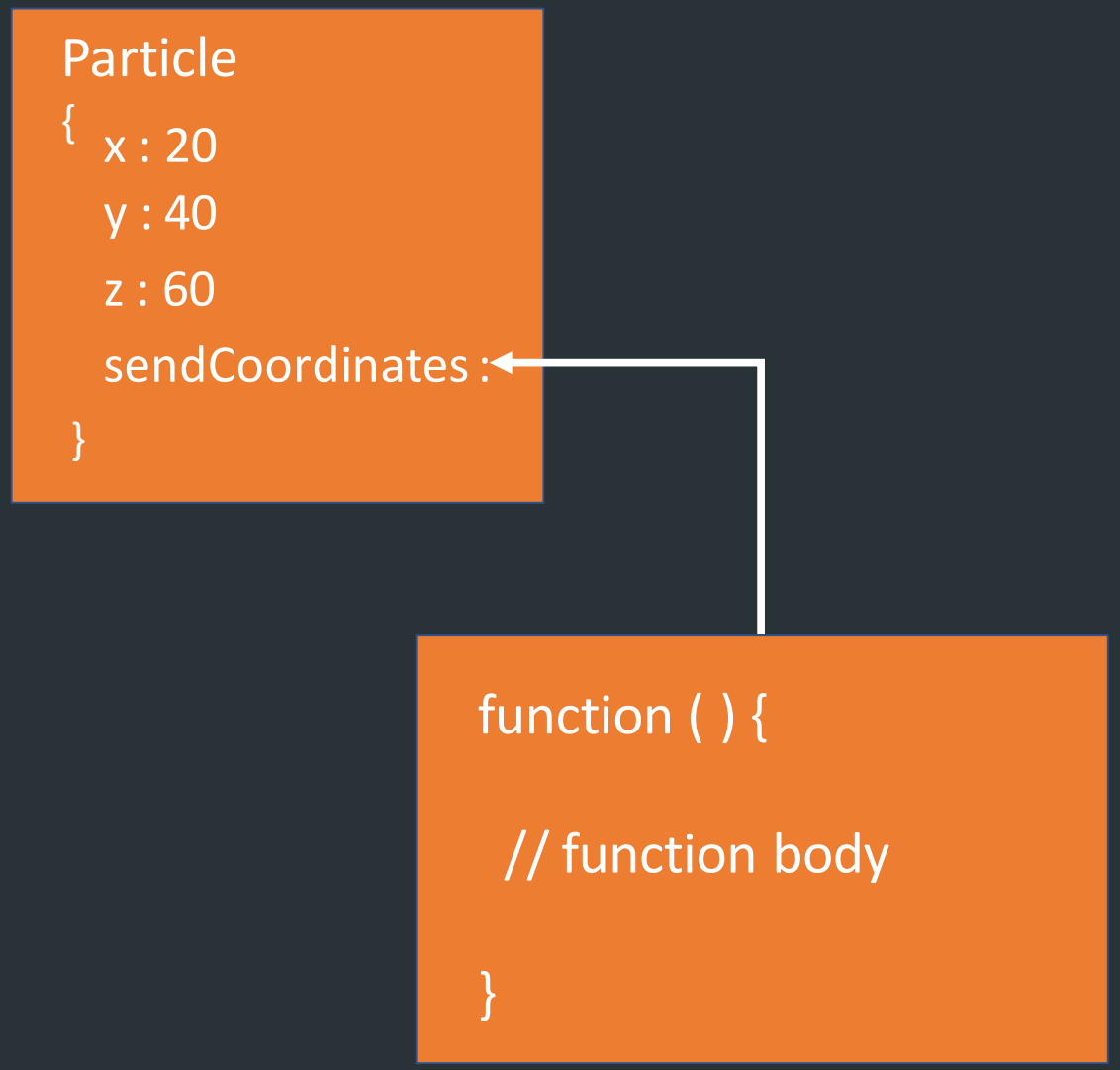
Memory



Thread of execution

```
1  let particle = {
2    x: 20,
3    y: 40,
4    z: 60,
5    sendCoordinates: function () {
6      return {
7        x: this.x,
8        y: this.y,
9        z: this.z,
10     };
11   },
12 };
13 console.log(particle.sendCoordinates());
14 // { x: 20, y: 40, z: 60 }
```

Memory



Thread of execution

```
1  let particle = {
2    x: 20,
3    y: 40,
4    z: 60,
5    sendCoordinates: function () {
6      return {
7        x: this.x,
8        y: this.y,
9        z: this.z,
10     };
11   },
12 };
13 console.log(particle.sendCoordinates());
14 // { x: 20, y: 40, z: 60 }
```

Memory

Particle
{
x : 20
y : 40
z : 60
sendCoordinates :
}

function ( ) {

// function body

}

Thread of execution

```
1  let particle = {
2    x: 20,
3    y: 40,
4    z: 60,
5    sendCoordinates: function () {
6      return {
7        x: this.x,
8        y: this.y,
9        z: this.z,
10     };
11   },
12 };
13 console.log(particle.sendCoordinates());
14 // { x: 20, y: 40, z: 60 }
```

Memory

Particle
{
x : 20
y : 40
z : 60
sendCoordinates :
}

function ( ) {

// function body

}

Thread of execution

```
let particle = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: function () {
    return {
      x: this.x,
      y: this.y
      z: this.z,
    };
  },
};
console.log(particle.sendCoordinates());
// { x: 20, y: 40, z: 60 }
```

Memory

Particle
{
x : 20
y : 40
z : 60
sendCoordinates :
}

function ( ) {

// function body

}

What we learned?

A function declared inside an object is never owned by that object. The property of that object just has a reference to that function.

```
function returnCoordinated() {
  return { x: this.x, y: this.y, z: this.z };
}

let particle1 = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: returnCoordinated,
};

let particle2 = {
  x: 800,
  y: 222,
  z: 888,
  sendCoordinates: returnCoordinated,
};

console.log(particle1.sendCoordinates()); // ?
console.log(particle2.sendCoordinates()); // ?
```

```javascript
function returnCoordinates() {
  return { x: this.x, y: this.y, z: this.z };
}

let particle1 = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: returnCoordinates,
};

let particle2 = {
  x: 800,
  y: 222,
  z: 888,
  sendCoordinates: returnCoordinates,
};

console.log(particle1.sendCoordinates());
// { x: 20, y: 40, z: 60 }

console.log(particle2.sendCoordinates());
// { x: 800, y: 222, z: 888 }
```

```javascript
function returnCoordinates() {
  return { x: this.x, y: this.y, z: this.z };
}

let particle1 = {
  x: 20,
  y: 40,
  z: 60,
  sendCoordinates: returnCoordinates,
};

let particle2 = {
  x: 800,
  y: 222,
  z: 888,
  sendCoordinates: returnCoordinates,
};

console.log(particle1.sendCoordinates());
// { x: 20, y: 40, z: 60 }

console.log(particle2.sendCoordinates());
// { x: 800, y: 222, z: 888 }
```

Implicit Binding

Implicit Binding

Implicit Binding – When you call a function through an object, the this keyword inside that function will refer to that object.*

```
function sayHi(name, interest) {
  return `Hi, ${this.name} I am ${name} and I love ${interest}`;
}

let person = {
  name: "Ankush",
};

sayHi.call(person, "sahil", "coding");
// Hi, Ankush I am sahil and I love coding

sayHi.apply(person, ["sahil", "coding"]);
// Hi, Ankush I am sahil and I love coding
```

Explicit Binding

Explicit Binding

```javascript
let Character1 = {
  name: "Jon Snow",
  height: 5.8,
  house: "stark",
  greet: function (name, house) {
    return `Hi, ${name} of house ${house}, I am ${this.name}`;
  },
};

let Character2 = {
  name: "Cersei Lannister",
  height: 5.6,
  house: "Lannister",
  greet: function (name) {
    return `I don't care about your house ${name}, I am ${this.name}`;
  },
};

Character1.greet.call(Character2, "Theon", "Greyjoy"); // ?
Character1.greet.apply(Character2, ["Theon", "Greyjoy"]); // ?
```

```javascript
let Character1 = {
  name: "Jon Snow",
  height: 5.8,
  house: "stark",
  greet: function (name, house) {
    return `Hi, ${name} of house ${house}, I am ${this.name}`;
  },
};

let Character2 = {
  name: "Cersei Lannister",
  height: 5.6,
  house: "Lannister",
  greet: function (name) {
    return `I don't care about your house ${name}, I am ${this.name}`;
  },
};

Character1.greet.call(Character2, "Theon", "Greyjoy");
// Hi, Theon of house Greyjoy, I am Cersei Lannister

Character1.greet.apply(Character2, ["Theon", "Greyjoy"]);
// Hi, Theon of house Greyjoy, I am Cersei Lannister
```

Explicit Binding overrides implicit binding

```
function greet() {
  console.log(`Hello ${this}`);
}

greet.call("Sahil");
// Hello Sahil

greet.call("Ankush");
// Hello Ankush
```

```javascript
let Particle1 = {
  x: 10,
  y: 30,
  z: 40,
  sendCoordinates: function () {
    return { x: this.x, y: this.y, z: this.z };
  },
};

let copy1 = Particle1.sendCoordinates;
let copy2 = Particle1.sendCoordinates.bind(Particle1);

copy1(); // ?
copy2(); // ?
```

```javascript
let Particle1 = {
  x: 10,
  y: 30,
  z: 40,
  sendCoordinates: function () {
    return { x: this.x, y: this.y, z: this.z };
  },
};

let copy1 = Particle1.sendCoordinates;
let copy2 = Particle1.sendCoordinates.bind(Particle1);

copy1();
// { x: undefined, y: undefined, z: undefined }

copy2();
// { x: 10, y: 30, z: 40 }
```

Hard Binding

Default Binding

What is the value of this?

```
1  function foo() {
2    console.log(this);
3  }
```

The value of this does not depend based on how the function was created, it depends on how the function is executed.

A better definition of this.

this is JavaScript's way of providing a dynamic context to a function when that function is executed.

Where to go from here?

- The new Binding.
- Try writing polyfill for call, apply and bind.
- Read about the precedence of bindings.
- Read about arrow functions and the concept of lexical this.
- Read about delegation pattern in JavaScript and how this is used there.

Thank You

@invalidtoken