CONTEXT IN FUNCTIONS

WHAT IS IT?

`this`

```javascript
function log () {
    console.log(this);
}
```

Global context

```
this === window // true

function log () {
    console.log(this);
}

log(); // window | undefined in 'use strict';
```

Function context

```
function log () {
    console.log(this);
}

log(); // window | undefined in 'use strict';
const obj = {};
obj.log = log;
obj.log(); // obj
```

Might be lost

```
function log () {
    console.log(this);
}

const obj = {};
obj.log = log;
(false || obj.log)(); // window | undefined in 'use strict';
```

Might be lost 2

```javascript
const methods = {
    greet () {
        console.log(this);
    }
};
methods.greet(); // methods
const lostContext = methods.greet;
lostContext(); // Window
```

CONTEXT RESCUE OPERATION

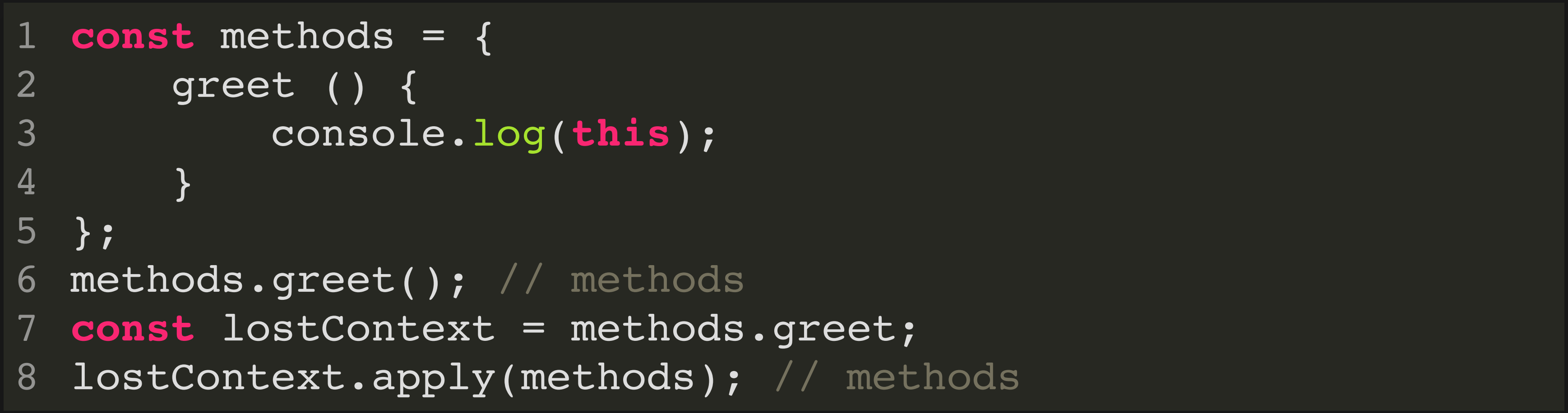
`call`

```
const methods = {
    greet () {
        console.log(this);
    }
};
methods.greet(); // methods
const lostContext = methods.greet;
lostContext.call(methods); // methods
```

With arguments

```
const methods = {
    greet (name) {
        console.log(this, name);
    }
};
methods.greet('Stefan');
const lostContext = methods.greet;
lostContext.call(methods, 'Stefan');
```

apply


```javascript
const methods = {
    greet () {
        console.log(this);
    }
};
methods.greet(); // methods
const lostContext = methods.greet;
lostContext.apply(methods); // methods
```

With arguments

```
const methods = {
    greet (name) {
        console.log(this, name);
    }
};
methods.greet('Stefan');
const lostContext = methods.greet;
lostContext.apply(methods, ['Stefan']);
```

bind

```javascript
const methods = {
    greet (name) {
        console.log(this, name);
    }
};
methods.greet(); // methods
const rescuedContext = methods.greet.bind(methods);
rescuedContext('Oleg'); // methods
```

With arguments

```javascript
const methods = {
    greet (name) {
        console.log(this, name);
    }
};
methods.greet(); // methods
const rescuedContext = methods.greet.bind(methods, 'Oleg');
rescuedContext(); // methods
```

ARROW FUNCTIONS

```
const NonArrowHi = function (name) { console.log(`Hi ${name}
```

```javascript
const arrowHi = (name) => { return console.log(`Hi ${name}`)
const arrowHiShort = (name) => console.log(`Hi ${name}`);
const hi = name => console.log(`Hi ${name}`);
```

CONTEXT IN ARROW FUNCTIONS

Always parent context

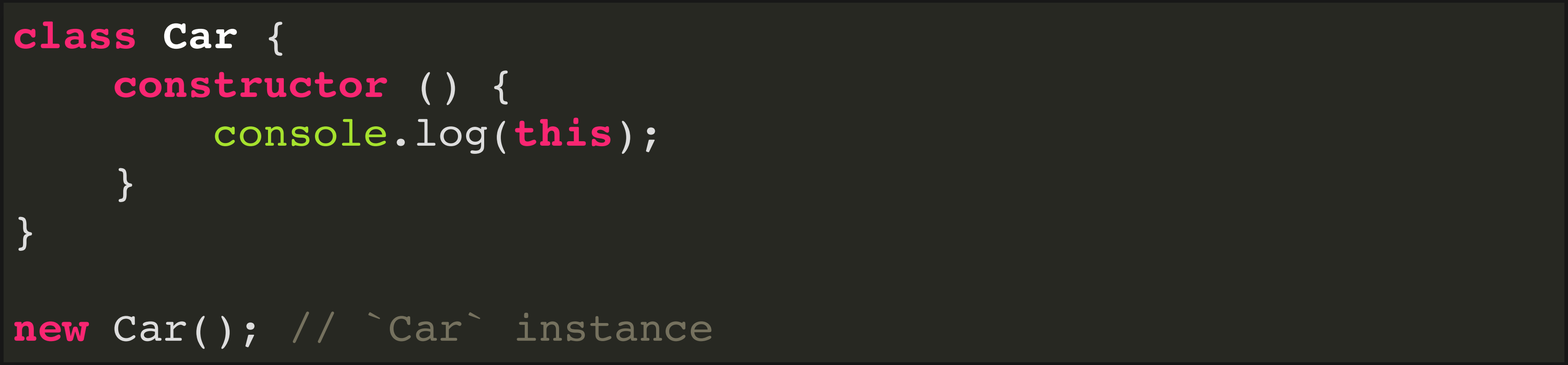
```
const getThis = () => console.log(this);
getThis() // window
const obj = { getThis };
obj.getThis(); // window
obj.getThis.call({ tellMeWhy: true }); // window
obj.getThis.apply({ tellMeWhy: true }); // window
obj.getThis.bind({ tellMeWhy: true })(); // window
```

```javascript
const parent = function papa () {
    const getThis = () => console.log(this, arguments);
    return getThis();
};
parent() // window []
parent.call({ tellMeWhy: true }) // { tellMeWhy: true } []
parent.apply({ gr: 1 }, ['argument']) // { gr: 1 } 'argument'
```

```
const obj = {
    method: () => {
        console.log(this);
    }
};

obj.method(); // window
```

CONTEXT IN CONSTRUCTOR



```javascript
class Car {
    constructor () {
        console.log(this);
    }
}

new Car(); // `Car` instance
```

is referenced to object, which is just being created

CONTEXT SUM UP

- Global context
- Function context
- Object method context
- Arrow function context
- Constructor context

MOST KNOWN CONTEXT PATTERN

`currying`

a.k.a partial execution pattern

```
const partial = (func, arguments) => {
    return func.bind(this, ...arguments)
};

const multiply = (mult, a, b) => (a + b) * mult;

const currying = partial(multiply, [2, 5]);
currying(10);
```