call()

Consider the following example:

```
let obj = {things: 3};
let addThings = function(a, b, c){
 return this.things + a + b + c;
};
```

In the above code, we have an object `obj` and a function `addThings` which are NOT related to each other. But as we write the following line of code, `this.things` in the function `foo()` gets its binding.

```
console.log( addThings.call(obj, 1,4,6) );
// Output
//14
```

This is because the first parameter in `.call()` is the context in which `this` must bind trough which `this.things=3` and thus `return this.things+a+b+c` gives `3+1+4+6 = 14`.

The `call()` method does not make a **copy** of the function it is being called on, it's just calls the function and binds the context with `this`.

apply()

It is similar to `call()`. The only difference is that we can pass arguments using `apply()` through an array also.

The same example can be done using `apply()` in the following way:

```
let obj = {things: 3};
let addThings = function(a, b, c){
 return this.things + a + b + c;
};
let arr = [1,4,6];
console.log( addThings.apply(obj, arr) );

//Output
//14
```

The above code binds the `this.things` in addThings() function and but does not need to take the other three elements separately, it can take only one argument i.e. array containing all the other arguments.

bind()

It **copies** the context function and then binds `this` to the **context**. It returns the copy of function with different context with which it is bind.

For example:

```
let obj = {things: 3};
let addThings = function(a, b, c){
 return this.things + a + b + c;
};
console.log( addThings.bind(obj, 1,4,6)() );

//Output
//14
```

In the above code, `this` is bind with the object `obj`. Notice that `bind` returns a function that is invoked in `console.log(..)` statement.

All the above methods are used to bind `this` to different contexts. But there are minor differences. `call()` and `apply()` has a difference in accepting the arguments but they both use the same function. On the other hand, `bind()` copies the function with the required context and returns that function which needs to be called.