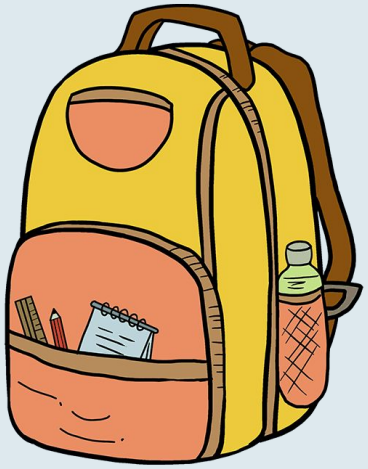
ALL ABOUT

THIS

Reminder - Closures

Closures are **functions** which remember the environment they were created in.

They carry a “backpack” with all the variables of the parent function which created it.



Nested Functions and Closure - Reminder

```
function get_count_self_calls(){
  var counter = 0;

    return function private(){
     counter++;
     console.log(counter);
   }
}

var count_self_calls = get_count_self_calls();
count_self_calls(); // 1
count_self_calls(); // 2
```

Context

Every function is executed with a context.

Context: **an object** that contains the variables and properties that the function can use.

Default JS Context in the browser: the Window object.

```
function foo(){ console.log("global context")};
```

Object Context: if a function is defined on an object, the object is the context of the function.

```
var obj = {
  name: "Bob",
  foo: function(){
    console.log("the object is my context");
  }
};
```

Context

Now the property < `name: "Bob"` > is part of foo's context.

```
var obj = {
  name: "Bob",
  foo: function(){
    console.log("the object is my context");
  }
};
```

But what is **THIS** ?

Context

THIS – is a reference to the context of the function.

```
function foo(){
  var scope_var = 5;
  this.context_var = 3;
  console.log(this);
}
```

`this` contains the `context_var` that was defined on the **context**.

Context

Object Literals

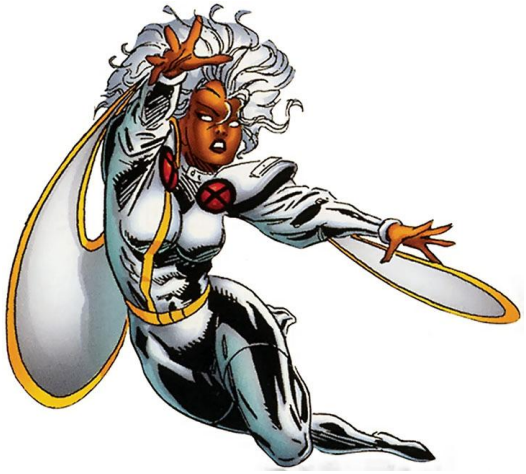
```
var obj = {
  name: "Bob",
};
```

Every property we assign to an object is saved on the context (=this).

```
obj.foo = function(){
  console.log(this.name);
}


obj.foo (); // What will be the result?
// Bob
```

Context

Object Literals - exercise



```
var storm = {
  // add code here
}
```

```
function printSuperPower(){
  console.log("my super power is " + this.superpower);
}
```

Your Task:

Invoke the printSuperPower function using the storm object as the context of the function.

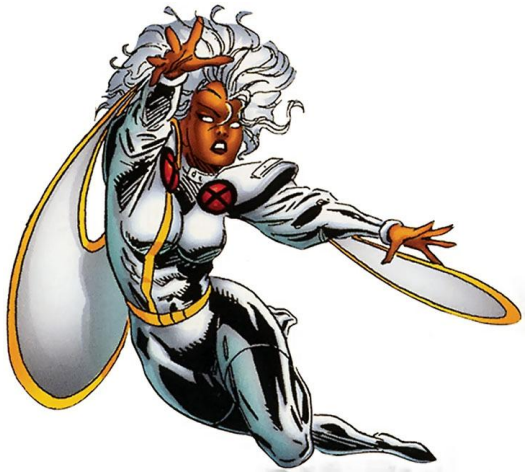
So your output should be: *my super power is flying*
Well, Storm also controls the weather, so, whatever y

Context

Object Literals - solution

```
var storm = {
    superpower: "flying",
    print: printSuperPower
}

function printSuperPower(){
    console.log("my super power is " + this.superpower);
}

storm.print()
```

We binded the function's context to the **storm** object.



Context

In Events

```
var element = document.body;
var foo = function(){
  console.log(this);
}

element.addEventListener("click", foo);
What will happen?
// will log the body element <body></body>
```

Context

Dynamic This

```
var obj = {
  name: "Bob",
};

var name = "Global scope";

var foo = function () {
  console.log(this.name);
}
obj.foo = foo; //function assignment
obj.foo();
// Bob

foo();
// Global scope
```

"this" value depends on who's invoking the function

Foo is invoked by obj

Foo is invoked by the window object (global scope)

Context

Changing the Context

```
var printName = function () {
  console.log(this.name);
}
```

What does printName function do?

Logs the name property of it's context.

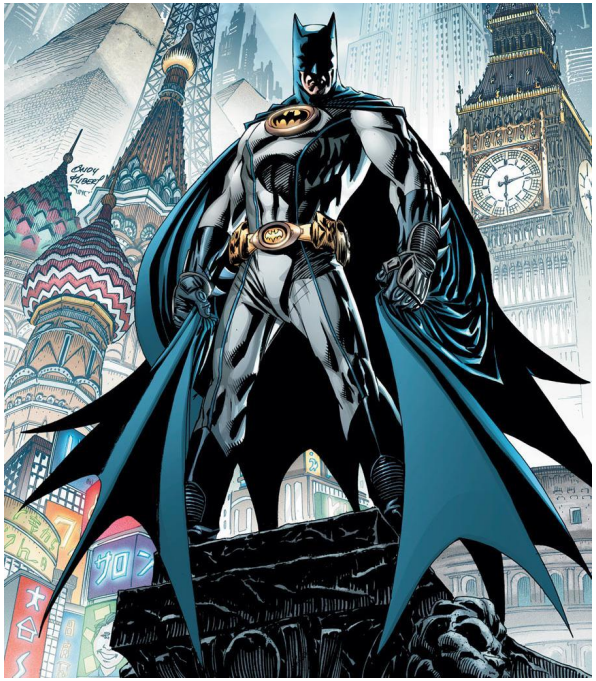
What context does it have?

Depends on how we invoke the function.

Bind

What if…

We want to bind a context to any function?



```
var printName = function () {
  console.log(this.name);
}

var batman = {
  name: "Bruce Wayne"
}

var describe = function(printHero, superPower){
  printHero();
  console.log("super power: " + superPower);
}

var printBatman = printName.bind(batman);
//set up batman as a context for the function
describe(printBatman, "none"); // Bruce Wayne. Super power: none
```

Bind()

The bind() function:

Binds a context to a function and returns the same function with the **BINDED** context.

Usage:

```
var bindedFunc = someFunc.bind(contextObj);
```

Bind

Another Example

```
var obj = {
  value: 10,
  add1: function(){
    return this.value + 1;
  }
}

obj.add1(); //what is the output?
11
var foo = obj.add1;
foo(); //what is the output?

NaN
```

But why?

When we save the function we loose the context of the object. The context depends on who is invoking the function.

When the function is invoked in the global scope the context is _____ ? The window.

Bind

Let's do something else!

```
var obj = {
  value: 10,
  add1: function(){
    return this.value + 1;
  }
}
```

```
var bigValue = {
   value: 999
}
```

We want to call add1 but use the value in bigValue.

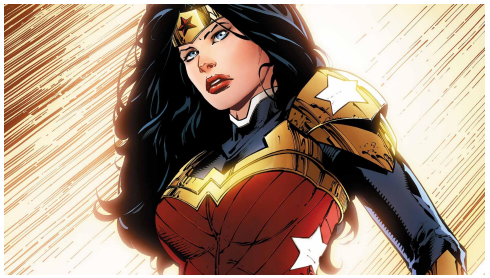
Can anyone help?

```
var foo = obj.add1.bind(bigValue);
foo();
```

Bind - Practice Time

We have an array of heroes:

```
var wonderWoman = {
  name: "Diana Prince"
}

var batman = {
  name: "Bruce Wayne"
}

var superHeroes = [wonderWoman, batman];

function printName() {
  console.log("My name is " + this.name);
}
```



Note: you cannot change the super heroes objects

We want to print the heroes names. Implement the printHeroes function:

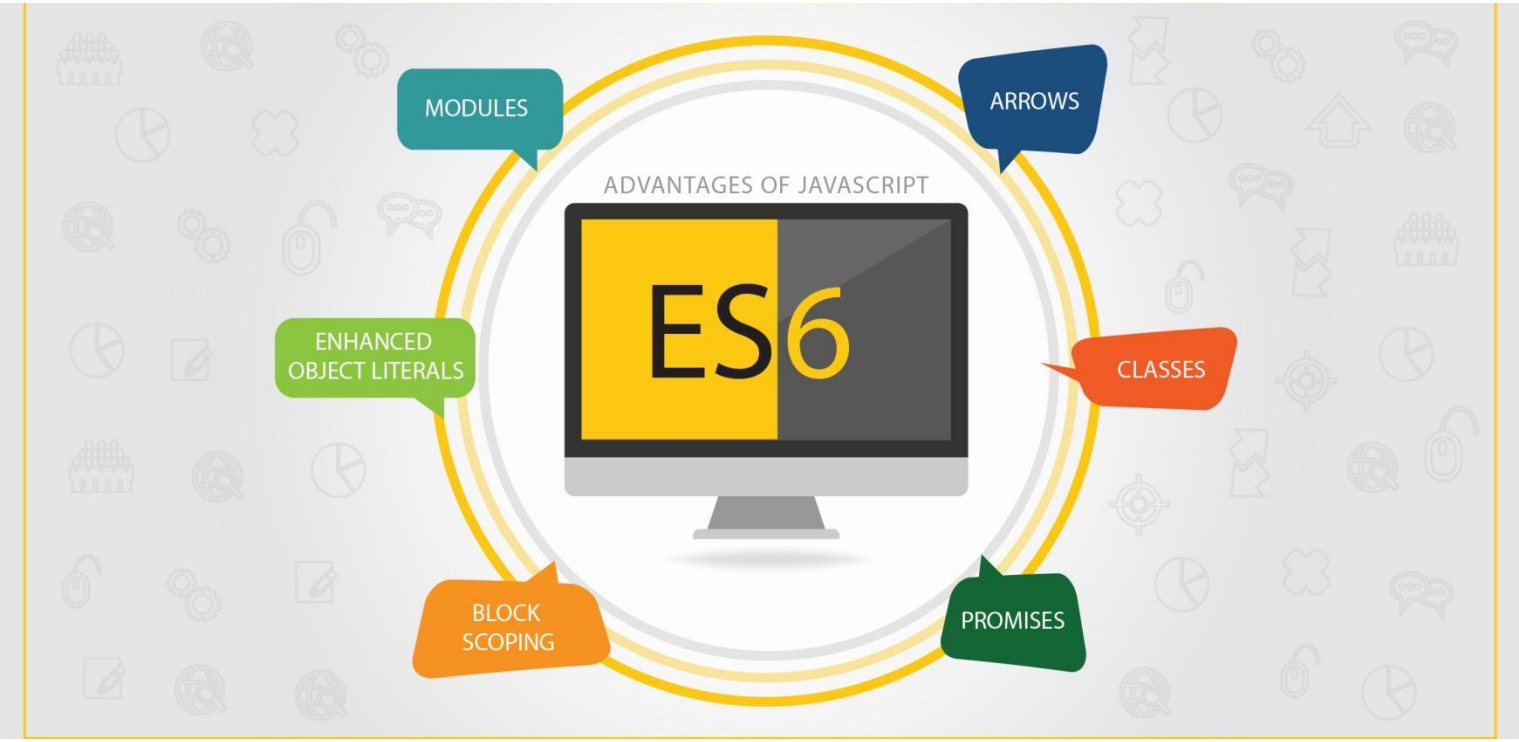
```
function printHeroes(heroes, printFunc){
  //add code here
}

printHeroes(superHeroes, printName);
```

Bind - Practice Time - Solution

```
var batman = {
    name: "Bruce Wayne"
}

var wonderWoman = {
    name: "Diana Prince"
}

var superHeroes = [wonderWoman, batman];

function printName() {
    console.log("My name is " + this.name);
}

function printHeroes(heroes, printFunc) {
    for (var i = 0; i < heroes.length; i++) {
        var bindedPrintFunc = printFunc.bind(heroes[i]);

        bindedPrintFunc();
    }
}

printHeroes(superHeroes, printName);
```

In each iteration, we bind the printName function to a different hero object

EcmaScript 6



Scope Review

What will be printed?

```
var x = 6;

function foo(){
  var x = 4;
  var y = 5;
  console.log(x);
  console.log(y);
}

foo();
console.log(x);
console.log(y);
```

```
Result:
4
5
6
Reference error
```

Scope – Another Example

What will be printed?

```
var x = 6;

if (x > 2){
  var x = 4;
  var y = 5;
}

console.log(x);
console.log(y);
```

```
Result:
4
5
```

Let and Const

Definition

Const – an immutable (cannot be changed) variable (constant).

Let – a block scope variable.

Const example

What will be the result of this code being executed?

```
const x = "ani";
x = "at";
x;
```

1. We cannot reassign a value to a const.
2. If we do an error will be thrown and x will remain the same.

Let

Definition

We said that Let is a
block scope
variable.

Blocks are code lines nested inside curly braces

```
if (true) {
  /** block starts
  ...
  block ends */
}
```

Let Example

What will be printed?

```
let x = 6;

if (x > 2){
  let x = 4;
  let y = 5;
}

console.log(x);
console.log(y);
```

Result:
6
Error y is not defined

Let Example

What will be printed?

```
let x = 6;

if (x > 2){
  let x = 4;
  let y = 5;
  console.log(x);
}
```

Result:

4

Questions

```
console.log("Questions?");
```