1. JavaScript - de kern

(bijna) alles is een object

.....
.....
.....
.....
.....
.....
.....
.....

1. JavaScript - de kern

dit stelt ons instaat om bijna alles te maken

(bijna) alles is een object

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

(bijna) alles is een object

CODE:

```
var user = {
    name: "test user",
    age: 25
};
```

CONSOLE:

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

(bijna) alles is een object

CODE:

```
var user = {
    name: "test user",
    age: 25
};
```

de property bag waar we
nu al steeds mee bezig zijn

CONSOLE:

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

functions zijn ook objects

CODE:

```
var getLastItem = function(array){
    return array[array.length-1];
}

console.log(getLastItem([1,2,3,4]));
```

CONSOLE:

4

.....

.....

.....

.....

.....

.....

.....

.....

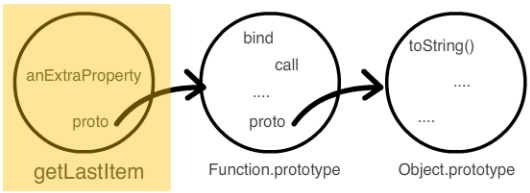
1. JavaScript - de kern

functions zijn ook objects

CODE:

```
var getLastItem = function(array){
    return array[array.length-1];
}
getLastItem.anExtraProperty = 5;
console.log(getLastItem([1,2,3,4]));
console.log(getLastItem.anExtraProperty);
```

PROTOTYPE CHAIN:



.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

functions zijn ook objects

CODE:

```
var getLastItem = function(array){
    return array[array.length-1];
}
getLastItem.anExtraProperty = 5;
console.log(getLastItem([1,2,3,4]));
console.log(getLastItem.anExtraProperty);
```

CONSOLE:

```
4
5
```

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Te gebruiken als variabelen

CODE:

```
var execute = function(func){
    func();
}
var testFunction = function(){
    console.log('executed');
}
execute( testFunction );
```

CONSOLE:

executed

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Te gebruiken als variabelen

CODE:

```
var execute = function(func){
    func();
}
var testFunction = function(){
    console.log('executed
}
execute( testFunction );
```

zonder ronde haken

CONSOLE:

executed

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Te gebruiken als variabelen

CODE:

```
var execute = function(func){
    func();
}
var testFunction = function(){
    console.log('executed');
}
execute( testFunction );
```

CONSOLE:

executed

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Te gebruiken als variabelen

CODE:

```
var execute = function(func){
    func();
}
var testFunction = function(){
    console.log('executed');
}
execute( testFunction );
```

CONSOLE:

executed

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Bruikbaar voor o.a. callbacksystemen en high-order functions

CODE:

```
function repeat(times, body) {
    for (var i = 0; i < times; i++) body(i);
}

repeat(3, function(n) {
    console.log("iteration: " + n);
});
```

CONSOLE:

iteration: 0
iteration: 1
iteration: 2

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Bruikbaar voor o.a. callbacksystemen en high-order functions

CODE:

```
function repeat(times, body) {
    for (var i = 0; i < times; i++) body(i);
}

repeat(3, function(n) {
    console.log("iteration: " + n);
});
```

CONSOLE:

iteration: 0
iteration: 1
iteration: 2

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Bruikbaar voor o.a. callbacksystemen en high-order functions

CODE:

```
function repeat(times, body) {
    for (var i = 0; i < times; i++) body(i);
}

repeat(3, function(n) {
    console.log("iteration: " + n);
});
```

CONSOLE:

```
iteration: 0
iteration: 1
iteration: 2
```

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Bruikbaar voor o.a. callbacksystemen en high-order functions

CODE:

```
function repeat(times, body) {
    for (var i = 0; i < times; i++) body(i);
}

repeat(3, function(n) {
    console.log("iteration: " + n);
});
```

CONSOLE:

```
iteration: 0
iteration: 1
iteration: 2
```

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Bruikbaar voor o.a. callbacksysteem

handig voor functionaliteiten als forEach

CODE:

```
function repeat(times, body) {
    for (var i = 0; i < times; i++) body(i);
}

repeat(3, function(n) {
    console.log("iteration: " + n);
});
```

CONSOLE:

iteration: 0
iteration: 1
iteration: 2

.....

.....

.....

.....

.....

.....

.....

.....

1. JavaScript - de kern

Bruikbaar voor o.a. callbacksysteem

handig voor functionaliteiten als forEach

CODE:

```
function loadJSON(url, callback){
    // retrieve the data, and when loaded:
    // execute the callback including the data
    // as parameter
}

loadJSON("http://bla.nl", setData);
```

CONSOLE:

.....

.....

.....

.....

.....

.....

.....

.....

2. this object

.....

.....

.....

.....

.....

.....

.....

.....

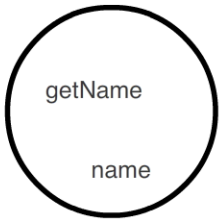
2. this object

Het 'this' object

CODE:

```
var user = {
    name: 'new user',
    getName: function(){return this.name;}
}
console.log( user.getName() );
```

OBJECTS:



user

.....

.....

.....

.....

.....

.....

.....

.....

2. this object

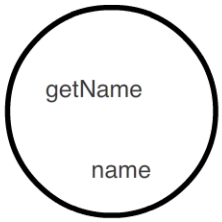
Het 'this' object

CODE:

```
var user = {
    name: 'new user',
    getName: function(){return this.name;}
}
console.log( user.getName() );
```

this verwijst naar de context waarin een method wordt aangeroepen

OBJECTS:



user

.....

.....

.....

.....

.....

.....

.....

.....

2. this object

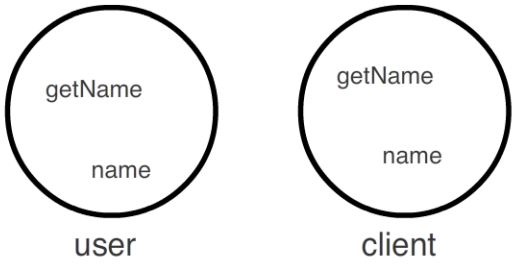
Het 'this' object

CODE:

```
var user = {
    name: 'new user',
    getName: function(){return this.name;}
}
var client = {name:'ns'};
client.getName = user.getName;
console.log( client.getName() );
```

this verwijst naar de context waarin een method wordt aangeroepen

OBJECTS:



.....

.....

.....

.....

.....

.....

.....

.....

2. this object

Het 'this' object

CODE:

```
var user = {
    name: 'new user',
    getName: function(){return this.name;}
}
var client = {name:'ns'};
client.getName = user.getName;
console.log( client.getName() );
```

zelfde functie maar aanroep
context bepaald this

OBJECTS:

.....

.....

.....

.....

.....

.....

.....

.....

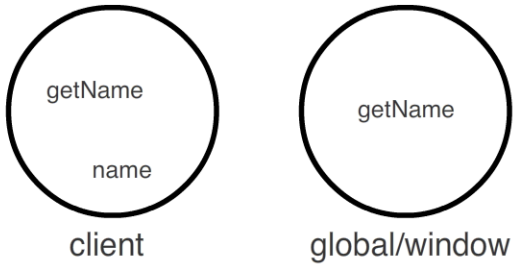
2. this object

Het 'this' object

CODE:

```
var user = {name: 'new user',
    getName: function(){return this.name;}
}
var client = {name:'ns'};
var getName = user.getName;
console.log( getName() );
```

OBJECTS:



.....

.....

.....

.....

.....

.....

.....

.....

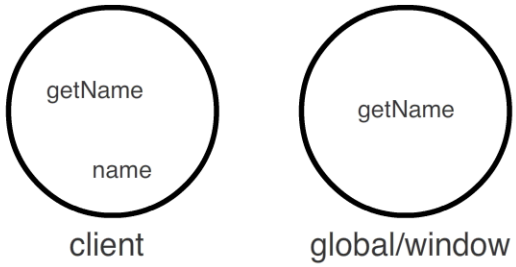
2. this object

Het 'this' object

CODE:

```
var user = {name: 'new user',
    getName: function(){return this.name;}
}
var client = {name:'ns'};
var getName = user.getName;
console.log( getName() );
```

OBJECTS:



.....

.....

.....

.....

.....

.....

.....

.....

2. this object

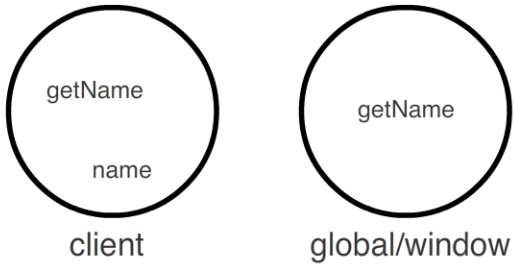
Het 'this' object

CODE:

```
var user = {name: 'new user',
    getName: function(){return this.name;}
}
var client = {name:'ns'};
var getName = user.getName;
console.log( getName() );
```

geen context betekent
global context dus window

OBJECTS:



2. this object

Gaan meer met 'this' te maken krijgen... maar eerst een opdracht

CODE:

OBJECTS:

.....

.....

.....

.....

.....

.....

.....

.....