A deep dive into data

Variables, memory, destructuring, and data types

Advanced JavaScript

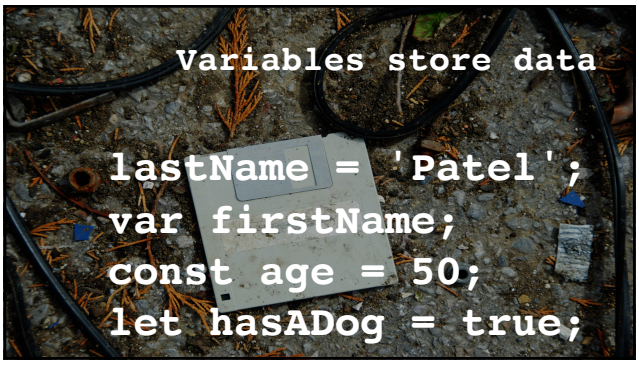
JS

1

tl;dr

- In JavaScript, variables are global by default!
- It hoists *var*s, giving them function scope but leaves lets and *const*s, giving them block scope
- It has dynamically-typed values
  - undefined - I don't have a value for this variable yet
  - number - Always a double
  - string - Not just an array of chars
  - boolean - Leads us to truthy and falsy values
  - function
  - object

3



4

To use a non-existent variable is a fatal error

```
let x = 5;

let z = x + y;  //ReferenceError: y is not defined
```

5

Hoisting

6

What happens to x?

```
1 var city = document.getElementById('city').value;
2 if (city === 'Albuquerque') {
3   var x = 10;
4 }
5 console.log("x is ", x);
```

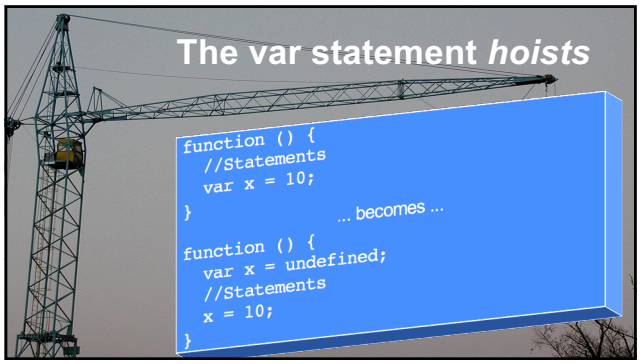
What happens on line 5?

A) x is always 10

B) x may be 10 or it may be undefined

C) A ReferenceError is thrown

7



8

Variable scoping

var, let, and const

9

Scoping refers to where a variable can be seen

Global scope

Function scope

Block scope

10

11

To give them function scope ...

1. Put them anywhere in a function
2. Use var

12

To give them block scope ...

Use let or const:

```
let foo;
let bar = 5;
const baz = 10;
```

ES 2015

13

let isn't hoisted

- This doesn't work

```
console.log(bar);
// referenceError; bar isn't defined
let bar="value";
```

- This is okay, though

```
function readThere () {
  return there
}
let there = 'foo'
console.log(readThere())
```

14

const behaves just like let

Not hoisted

Block-scoped

15

except that const is constant ...

- Values must be assigned on declaration:

```
const x = 10;

x='foo';   // throws -- TypeError

const x = 'foo'; // throws -- redeclaration
```

16

... except when it's not

```
const simpsons = ['homer', 'marge', 'bart', 'lisa'];
simpsons.push('maggie');  // totally works.
const neighbor = {
  first: "Ned"
}
neighbor.kids = ["Rod", "Todd"];  // Also works.
```

const is not immutable

17

Data types

19

JavaScript is weakly typed

- undefined
- number
- boolean
- function
- string
- object

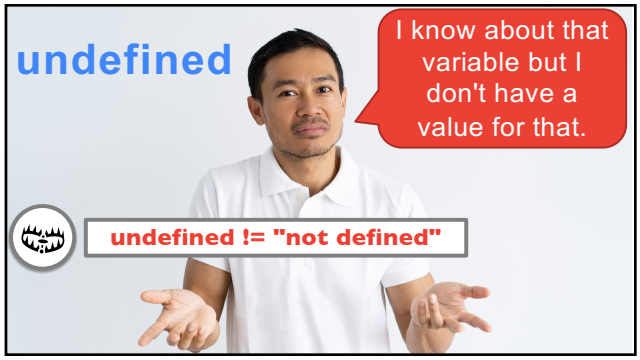
20

How do I know what I'm working with?

- The typeof operator

```
function foo(x) {
  if (       x != "object")
    throw "I need an object";
  else
    // Do stuff with that object
}
```

21

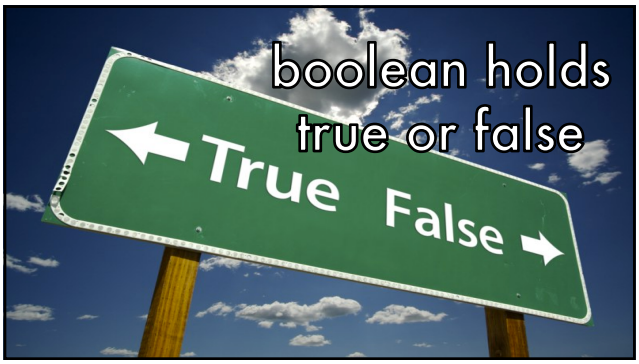


22

```
var x = 5;          // Integer
var x = 5.5;        // Floating point
var x = 0o50;       // Octal
var x = 0x37;       // Hexadecimal
var x = 0b110111; // Binary
var x = 5.0e50;     // Scientific notation
```

Numbers are very flexible

25



26

Strings and template literals

Borrowing from mustaches/handlebars

27

The old-school way of string concatenation

```
var name = 'Your name is ' + first + ' ' + last
+ '.';
var url = 'http://us.com/api/messages/' + id;
```

These are backtics or backquotes! Not single quotes.

And the new way ...

```
let name = `Your name is ${first} ${last}.`
let url = `http://us.com/api/messages/${id}`
```

28

The backtics allow us to have multiline strings

```
var roadPoem =
`Then took the other, as just as fair,
 And having perhaps the better claim
 Because it was grassy and wanted wear,
 Though as for that the passing there
 Had worn them really about the same,`;
```

29

And the most useful of data types ...

Objects!

32

Overview

- There are generally two types of objects

1. Declared
   - One copy
   - Like a static object
2. Created
   - Is instantiated
   - Like an instance object

33

JavaScript Objects are simple hashes

- Merely a collection/hash/dictionary of key-value pairs. Very simple.

firstName	"Rufus"
middleName	"Xavier"
lastName	"Sarsparilla"
age	10
sister	rafaela
pet	rhinoceros

- Not based on classes at all.

34

We can create objects on the fly using curly braces … no need for classes

```
const obj = {
  firstName: "Meg",
  lastName: "Griffin",
  pathology: "Rejection",
};
```

Trailing commas are encouraged!

36

Property shorthands

```
const firstName = "Meg";
const lastName = "Griffin";
const pathology = "Rejection";
const obj = {
 firstName, lastName, pathology,
};
```

Note: No colons!

37

Properties are similarly created dynamically

```
const person1 = {
  lastName: "Griffin",
  firstName: "Stewie",
  city: "Quahog",
  state: "RI",
};
document.write(`The person is
  ${person1.firstName} ${person1.lastName}`);
document.write("He is from " + person1["city"]);
var prop="state";
document.write("He lives in " + person1[prop]);
```

40

41

Remember, objects can contain anything

```
var family = {
  name: "The Griffins",       // A string
  incomeInDollars: 34000,     // A number
  dog: brian,                   // Another object
  parents: [ peter, lois ],   // An array
  playTheme: function () {
    return "It seems to me " +
      "that all you see is ...";
  }  // A function
}
•
```

42

Object spread

44

how_not_to_make_a_copy_of_an_object.js

```
import {first, last} from 'name';

console.log(name);  // {first:"Jo",last:"Kim"}
const n2 = name;
console.log(n2); // {first:"Jo",last:"Kim"}
n2.first = "Al";
console.log(name); // {first:"Al",last:"Kim"}
```

Why?!?

45

The spread operator (...) *spreads* out the elements in an object

Takes a blob of properties and flattens them into all of the properties in the object.

46

how_to_make_a_copy_of_an_object.js

```
import {first, last} from 'name';

console.log(name);   // {first:"Jo",last:"Kim"}
const n2 = {...name};
console.log(n2);     // {first:"Jo",last:"Kim"}
n2.first = "Al";
console.log(name);   // {first:"Jo",last:"Kim"}
console.log(n2);     // {first:"Al",last:"Kim"}
```

47

object_spread.js

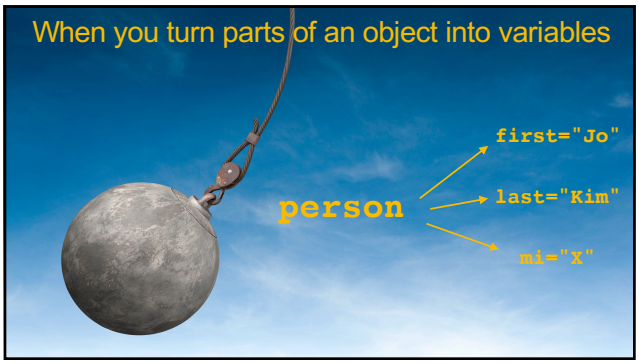
```
import {first, last} from 'name';

const address = {street:"123 Elm", city:"NY",
                 state:"NY",zip:"01234"};
const person = { first, last, address };
console.log(person);  // Address is an object
const p2 = { first, last, ...address };
console.log(p2); // Address is spread out!
```

48

Destructuring

49



50

Destructuring

- Just put the object on the left side of the "="

```
let {prop1:v1, prop2:v2} = someObject;
```

- Where prop1 & prop2 are properties of someObject
- And v1 & v2 will become the new variables.

- Note: You're not creating an object. You're matching based on the shape of the object.

51

An example of object destructuring

```
let {username, password} = req.body;
// same as
// let username = req.body.username;
// let password = req.body.password;
```

52

Nested destructuring

foo.js

```
const person = {
 addy: {city: 'Newark', state: 'NJ' },
 name: { first: 'Jo', last: 'Li', m: 'X' },
};

const {name:{first}} = person;
console.log(first);  // "Jo"
```

53

Destructuring in a function

foo.js

```
function makeFullName({first,last,mi}) {
 return `${first} ${mi}. ${last}`;
}
const p1 = getPerson();
console.log(makeFullName(p1));
const p2 = {first:"Jo",last:"Li",mi:"X"};
console.log(makeFullName(p2));
```

55

tl;dr

- In JavaScript, variables are global by default!
- It hoists *var*s, giving them function scope but leaves lets and *const*s, giving them block scope
- It has dynamically-typed values
  - undefined - I don't have a value for this variable yet
  - number - Always a double
  - string - Not just an array of chars
  - boolean - Leads us to truthy and falsy values
  - function
  - object

58