Title: Variable Scope through example code

Answer:

1. **A globally-scoped variable**

```
// global scope
var a = 1;

function one() {
  alert(a); // alerts '1'
}
```

2. **Local scope**

```
// global scope
var a = 1;

function two(a) {
  // local scope
  alert(a); // alerts the given argument, not the global value of '1'
}

// local scope again
function three() {
  var a = 3;
  alert(a); // alerts '3'
}
```

3. **Intermediate**: *No such thing as block scope in JavaScript* (ES5; ES6 introduces `let` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let))

   a.

```
var a = 1;

function four() {
  if (true) {
    var a = 4;
  }

  alert(a); // alerts '4', not the global value of '1'
}
```

   b.

```
var a = 1;

function one() {
  if (true) {
    let a = 4;
  }

  alert(a); // alerts '1' because the 'let' keyword uses block scoping
}
```

4. Intermediate: *Object properties*

```
var a = 1;

function Five() {
  this.a = 5;
}

alert(new Five().a); // alerts '5'
```

5. Advanced: *Closure*

```
var a = 1;

var six = (function() {
  var a = 6;

  return function() {
    // JavaScript "closure" means I have access to 'a' in here,
    // because it is defined in the function in which I was defined.
    alert(a); // alerts '6'
  };
})();
```

6. Advanced: *Prototype-based scope resolution*

```
var a = 1;

function seven() {
  this.a = 7;
}

// [object].prototype.property loses to
// [object].property in the lookup chain. For example...

// Won't get reached, because 'a' is set in the constructor above.
seven.prototype.a = -1;

// Will get reached, even though 'b' is NOT set in the constructor.
seven.prototype.b = 8;

alert(new seven().a); // alerts '7'
alert(new seven().b); // alerts '8'
```

7. Global+Local: *An extra complex Case*

```
var x = 5;

(function () {
    console.log(x);
    var x = 10;
    console.log(x);
})();
```

This will print out `undefined` and `10` rather than `5` and `10` since JavaScript always moves variable declarations (not initializations) to the top of the scope, making the code equivalent to:

```
var x = 5;

(function () {
    var x;
    console.log(x);
    x = 10;
    console.log(x);
})();
```

8. Catch clause-scoped variable

```
var e = 5;
console.log(e);
try {
    throw 6;
} catch (e) {
    console.log(e);
}
console.log(e);
```

This will print out `5`, `6`, `5`. Inside the catch clause `e` shadows global and local variables. But this special scope is only for the caught variable. If you write `var f;` inside the catch clause, then it's exactly the same as if you had defined it before or after the try-catch block.

Tags: variables, javascript