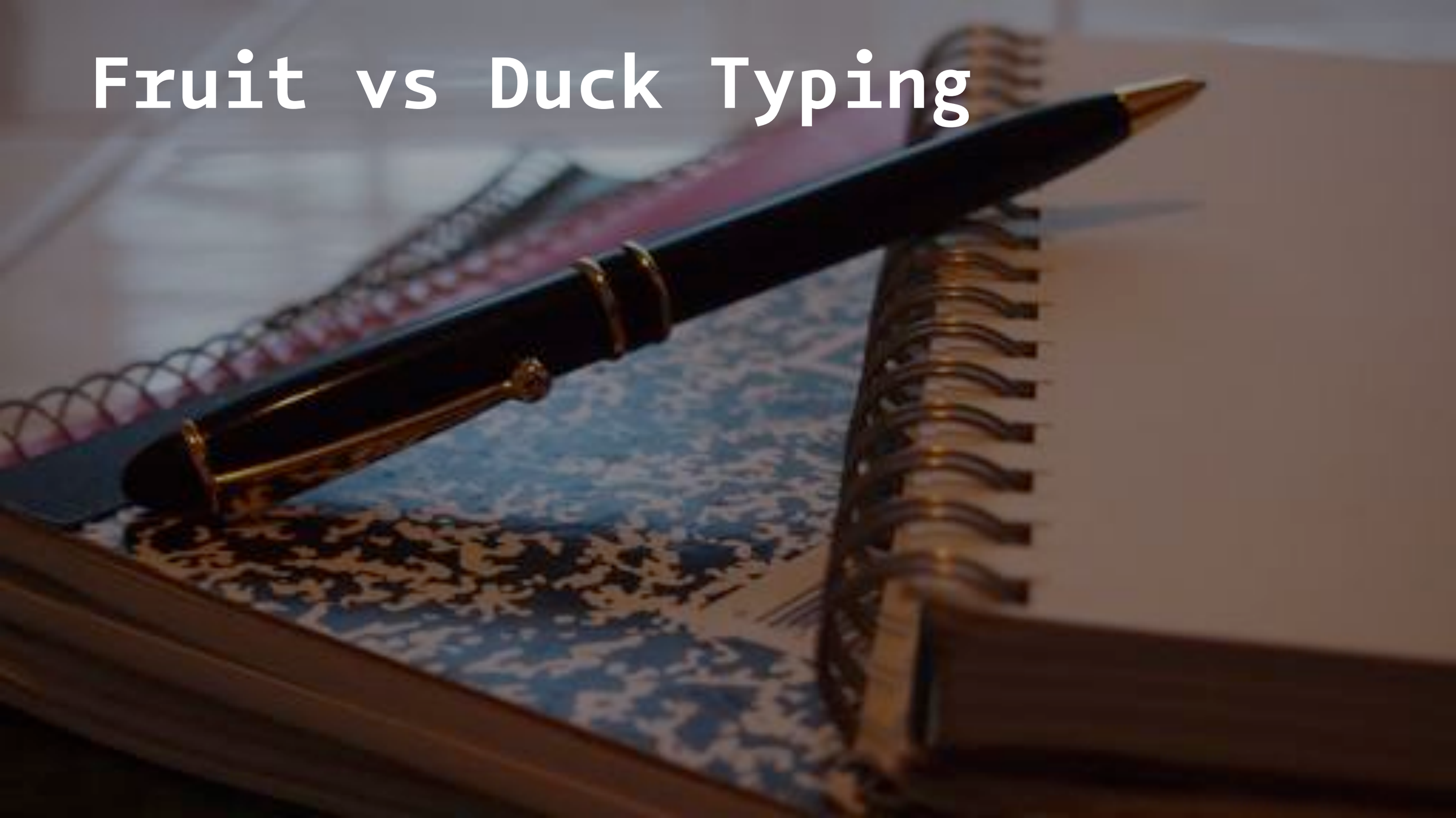
the art of the javascript metaobject protocol

Adapt or Perish


20

Fruit vs Duck Typing




Smith's
FRUIT MARKET
Est 1940
Fifth Generation Orchardist
304-856-3763

Fruit Typing:

Fruit

Not Fruit


"fruit" is deeper than its
Taste, Smell or Color

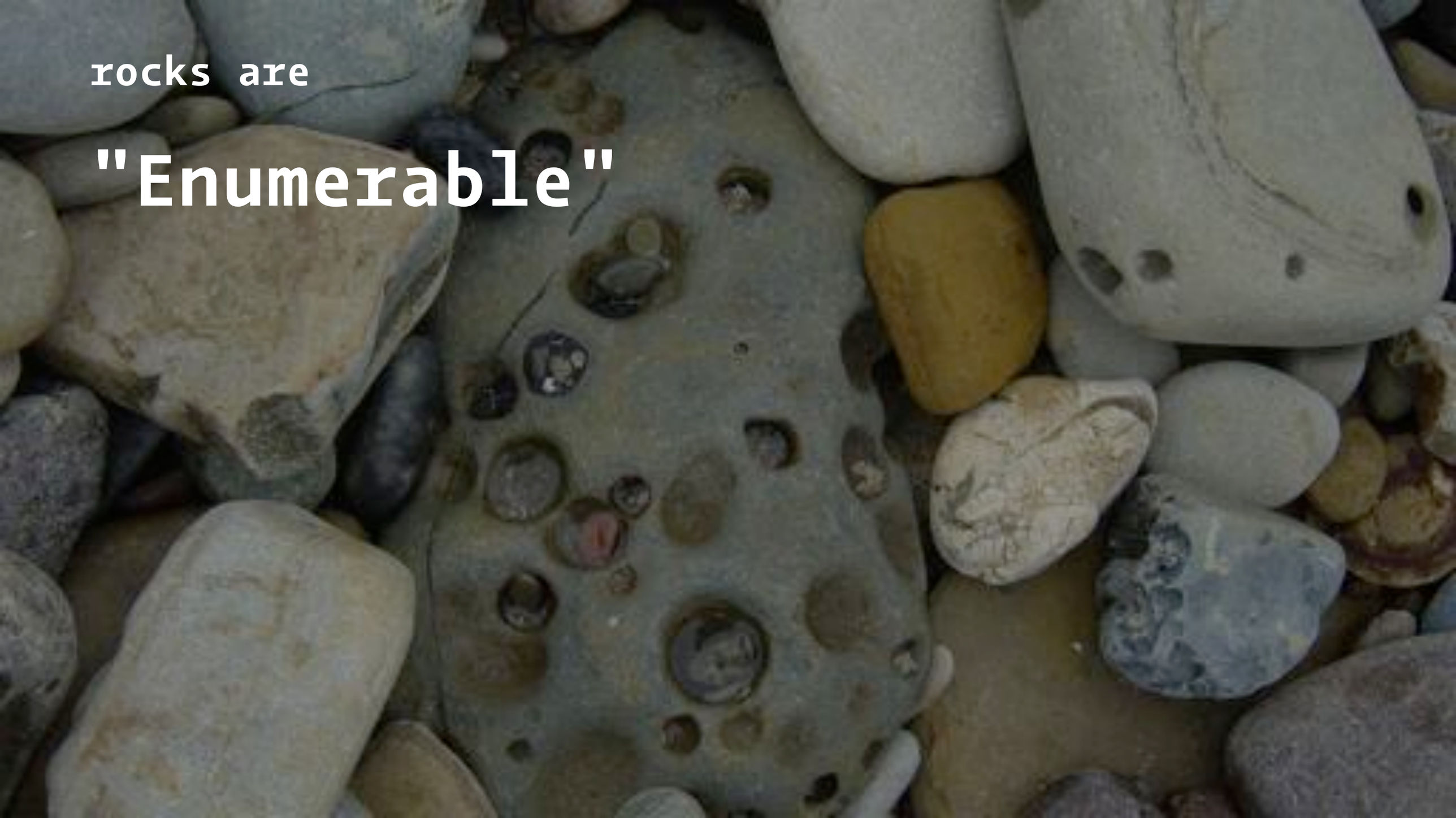
Duck Typing:


1
2
3
4
5

"Enumerable"


VFGAS
INSIDE
Dieter S. Gunda


people are
"Enumerable"
GAS
INSIDE


rocks are
"Enumerable"


duck typing is like
"Enumerable," not like "Fruit"

Duck typing is a style of typing in which an object's methods and properties determine the valid semantics,

...rather than its inheritance from a particular class or implementation of an explicit interface.

a problem we usually

Solve with Duck Typing:

Conway's Game of Life

```
function StandardCell () {
  this._neighbours = [];
  this._alive      = false;
}
```

```
StandardCell.prototype.neighbours =
  function neighbours (neighbours) {
    return this._neighbours;
  };

StandardCell.prototype.setNeighbours =
  function setNeighbours (neighbours) {
    this._neighbours = neighbours.slice(0);
    return this;
  };
```

```
StandardCell.prototype.alive =
  function alive () {
    return this._alive;
  };

StandardCell.prototype.setAlive =
  function setAlive (alive) {
    this._alive = alive;
    return this;
  };
```


Today
4 billion years
11 billion ye

moving through

Time

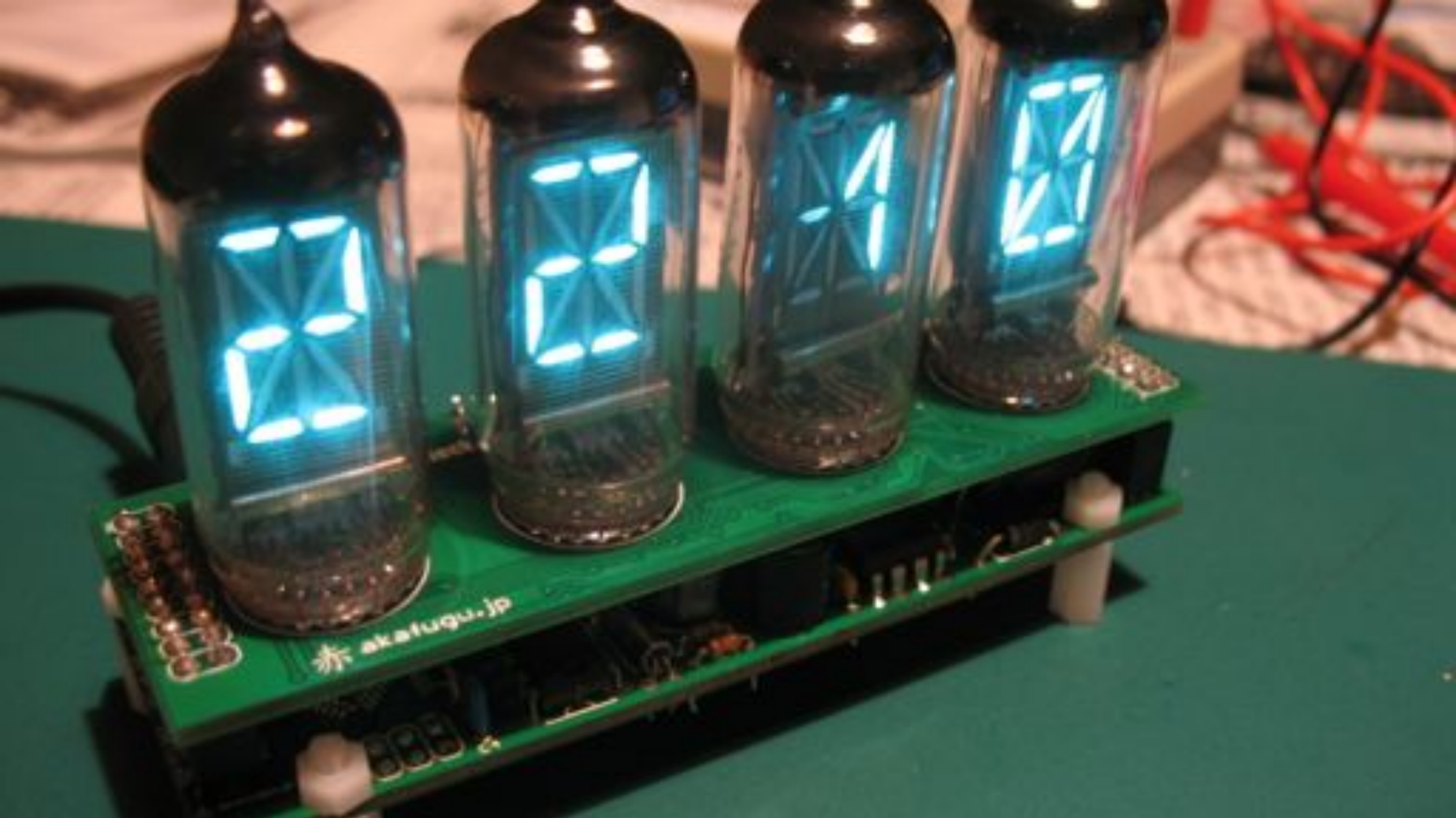
```
StandardCell.prototype.nextAlive =
  function nextAlive () {
    var alives =
      this._neighbours.filter(function (n) {
        return n.alive();
      }).length;
    if (this.alive()) {
      return alives === 2 ||
             alives == 3;
    }
    else {
      return alives == 3;
    }
  };
```

```
Universe.prototype.iterate =
  function iterate () {
    var aliveInNextGeneration = this.cells().map(
      function (c) {
        return [c, c.nextAlive()];
      }
    );

    aliveInNextGeneration.forEach(function (a) {
      var cell = a[0],
          next = a[1];

      cell.setAlive(next);
    });
  };
```


drawing
Life

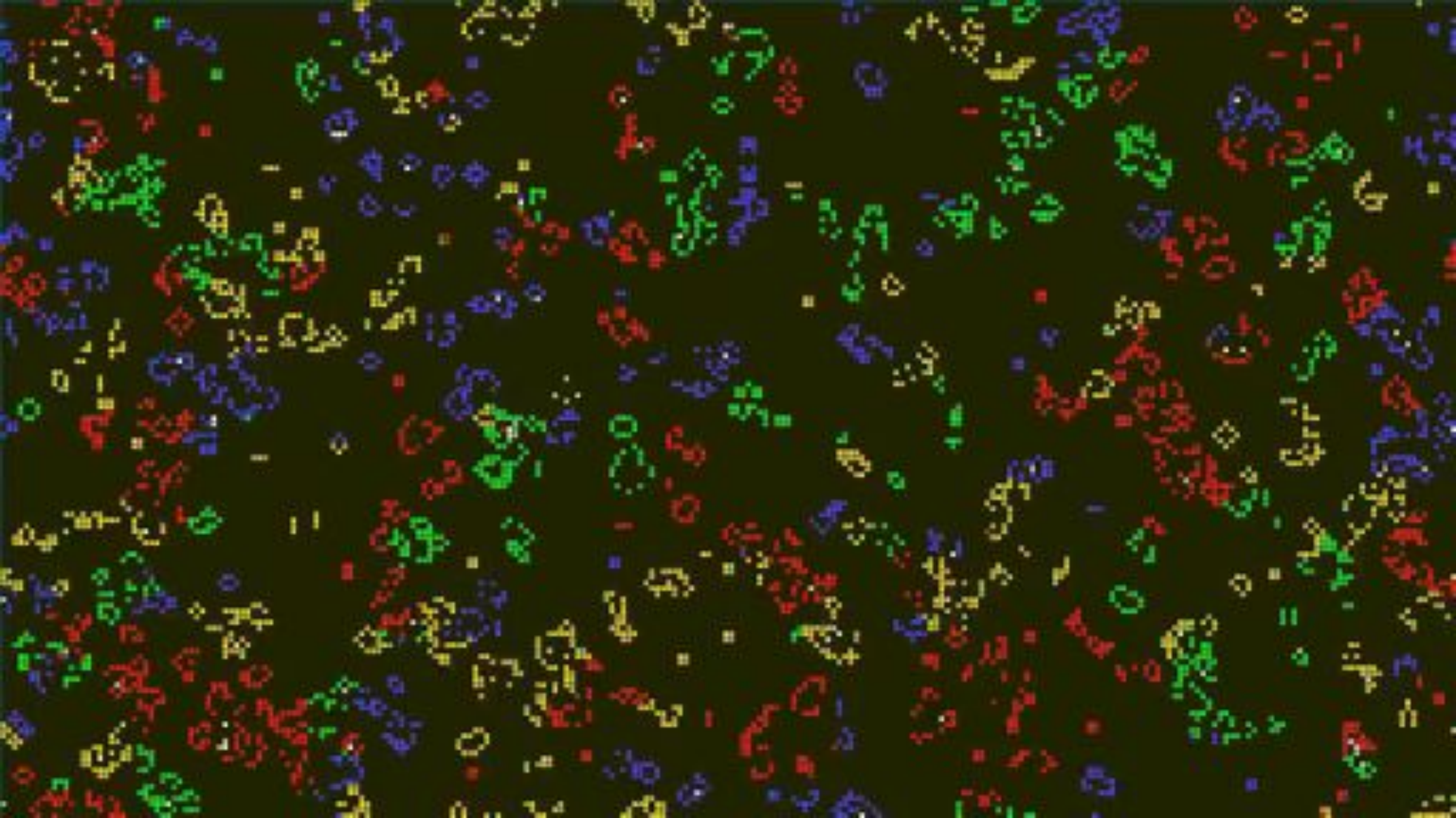
```
View.prototype.drawCell =
  function drawCell (cell, x, y) {
    var xPlus = x + this.cellSize(),
        yPlus = y + this.cellSize()
    this._canvasContext.clearRect(x, y, xPlus, yPlus);
    this._canvasContext.fillStyle = this.cellColour(cell);
    this._canvasContext.fillRect(x, y, xPlus, yPlus);
    return self;
  };
```

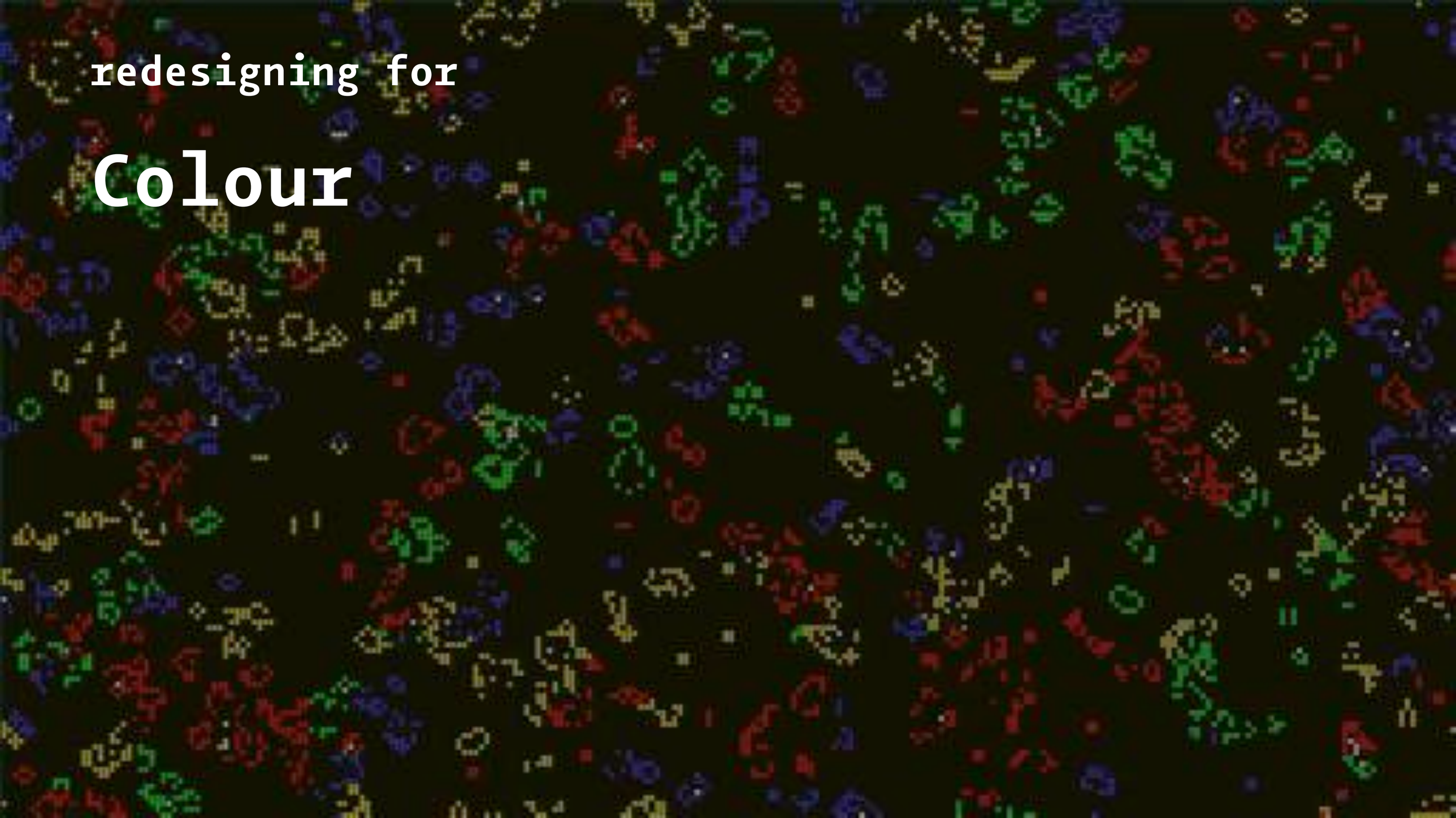
```
View.prototype.cellColour =
  function cellColour (cell) {
    return cell.alive()
           ? WHITE
           : BLACK;
  };
```


akafuqu.jp


CHANGE

Ch-ch-ch-changes!




redesigning for
Colour

```
function ColourCell () {
  this._neighbours = [];
  this._age        = 0;
}

ColourCell.prototype.neighbours =
  StandardCell.prototype.neighbours;


ColourCell.prototype.setNeighbours =
  StandardCell.prototype.setNeighbours;
```

```
ColourCell.prototype.age =
  function age () {
    return this._age;
  };

ColourCell.prototype.setAge =
  function setAge (age) {
    this._age = age;
    return this;
  };
```

moving through time

in Colour

```
ColourCell.prototype.nextAge =
  function next () {
    var alives =
      this._neighbours.filter(function (n) {
        return n.age() > 0;
      }).length;
    if (this.age() > 0) {
      return (alives === 2 || alives == 3)
             ? (this.age() + 1)
             : 0;
    }
    else {
      return (alives == 3)
             ? (this.age() + 1)
             : 0;
    }
  };
```

```
Universe.prototype.iterate =
  function iterate () {
    var ageInNextGeneration = this.cells().map(
      function (c) {
        return [c, c.nextAge()];
      }
    );

    ageInNextGeneration.forEach(function (a) {
      var cell = a[0],
          next = a[1];

      cell.setAge(next);
    });
  };
```

drawing life

in Colour

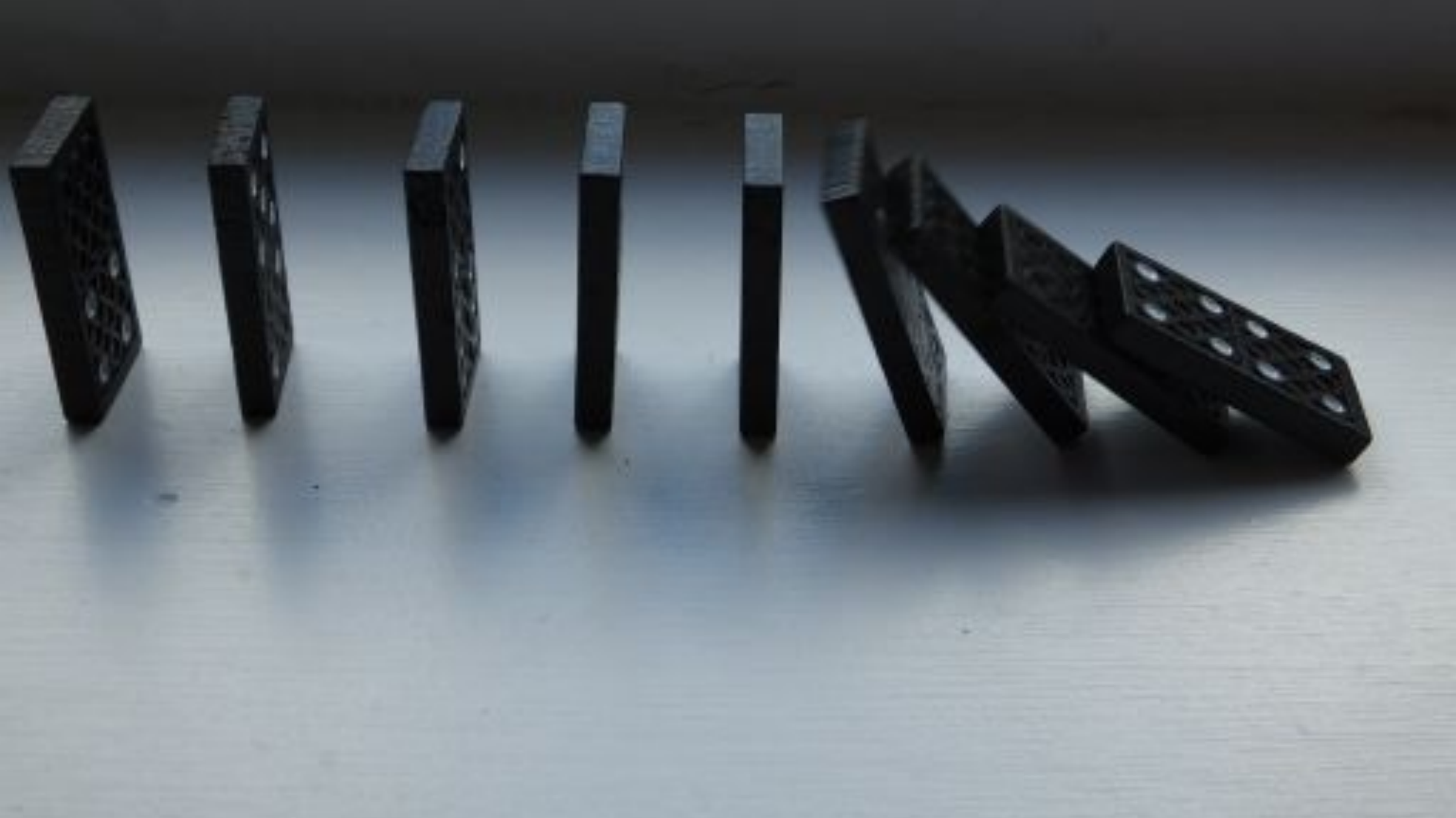
```
var COLOURS =
  [ BLACK, GREEN, BLUE, YELLOW, WHITE, RED ];

View.prototype.cellColour =
  function cellColour (cell) {
    return COLORS[
                  (cell.age() >= COLOURS.length)
                  ? (COLOURS.length - 1)
                  : cell.age()
                ];
  };

// ...
```

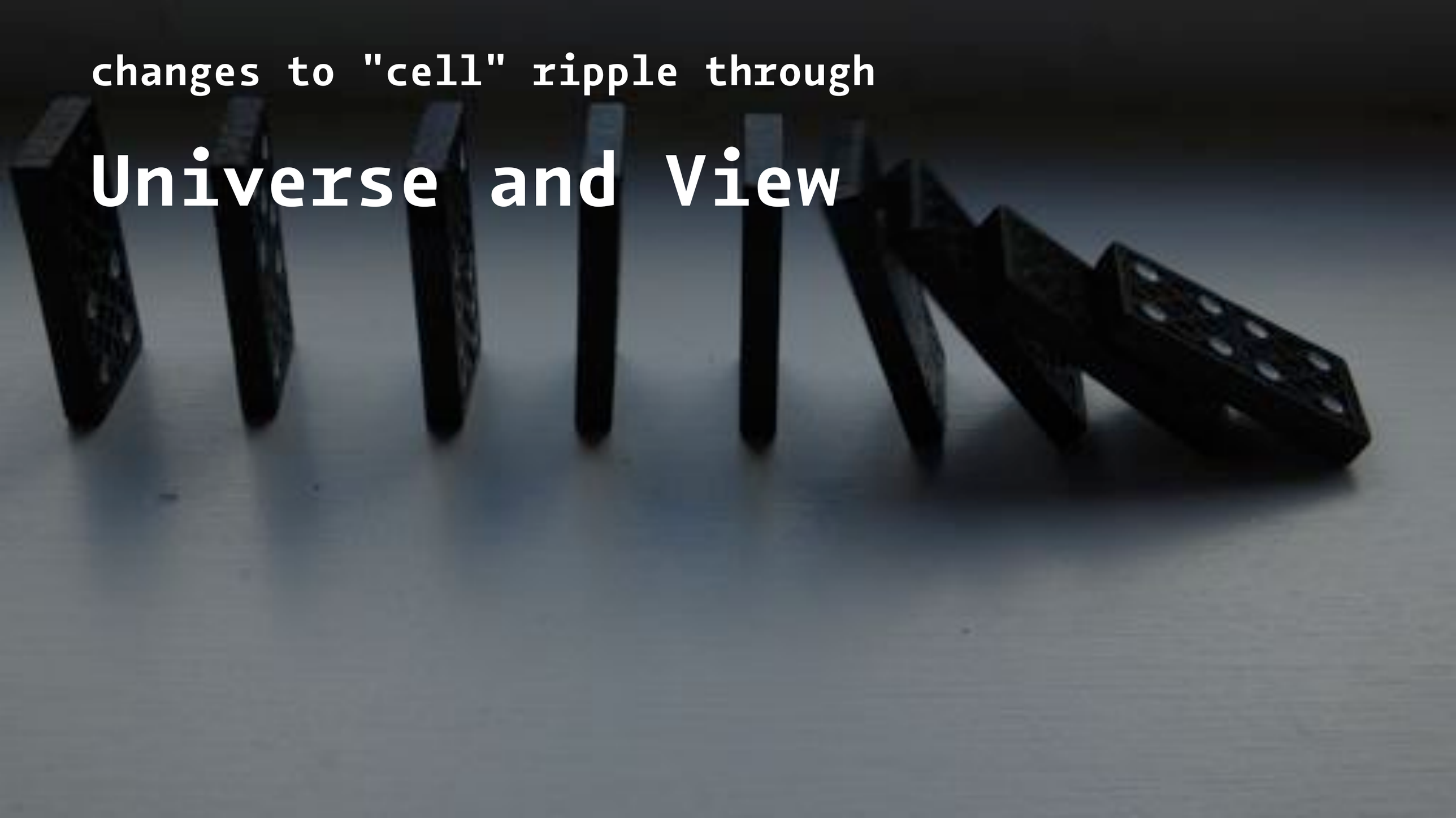

85


something
Doesn't Fit

```
Object.keys(StandardCell.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'alive',
    'setAlive',
    'nextAlive' ]
```

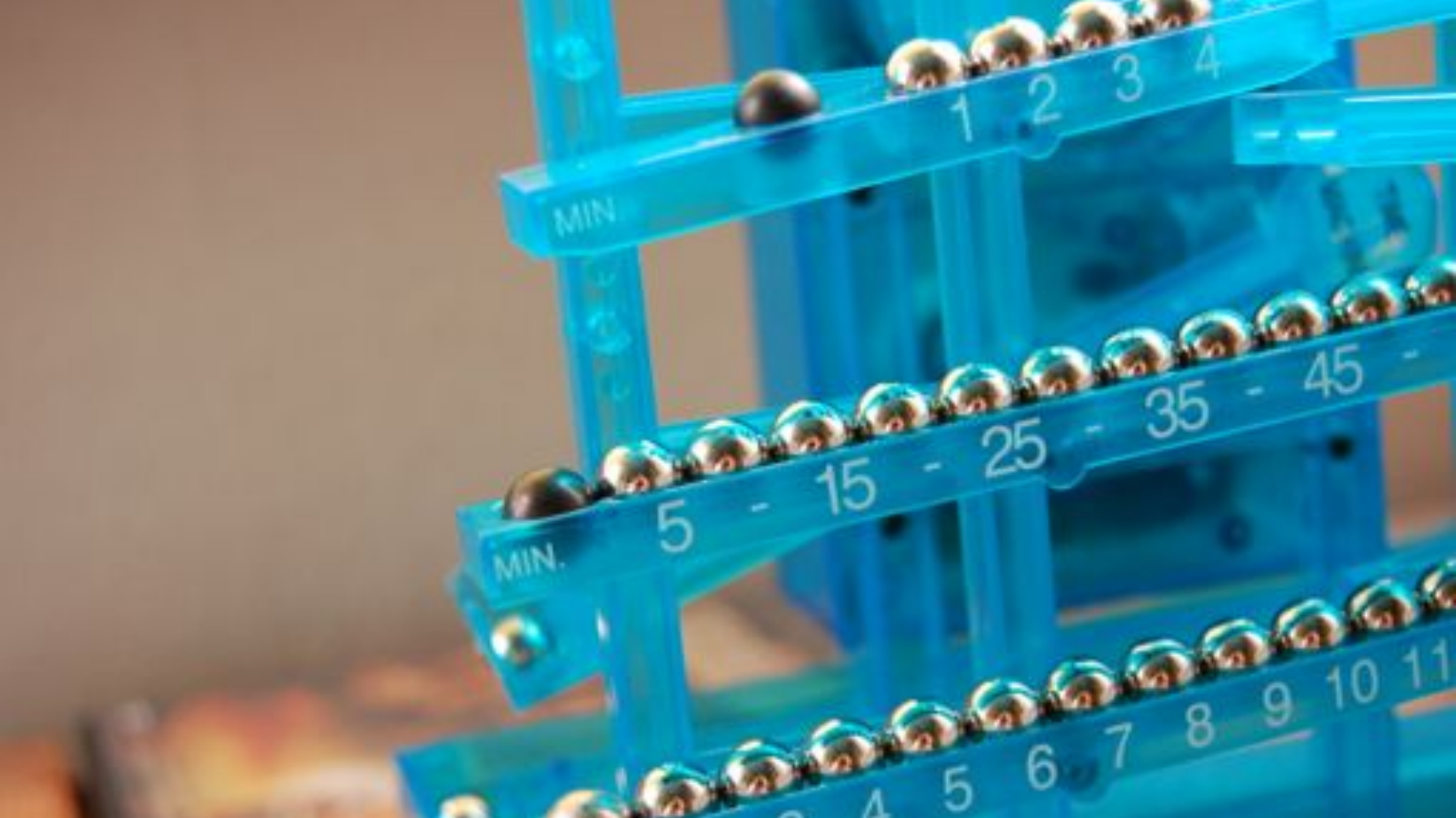
```
Object.keys(ColourCell.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'age',
    'setAge',
    'nextAge' ]
```





changes to "cell" ripple through

Universe and View


MIN.
1 2 3 4
MIN.
5 - 15 - 25 - 35 - 45
4 5 6 7 8 9 10 11

direct

Duck Typing

```
ColourCell.prototype.alive =
  function alive () {
    return this._age > 0;
  };
```

```
ColourCell.prototype.setAlive =
  function setAlive (alive) {
    if (alive) {
      this.setAge(this.age() + 1);
    }
    else this.setAge(0);
    return this;
  };
```

```
ColourCell.prototype.nextAlive =
  StandardCell.prototype.nextAlive;
```

```
Object.keys(ColourCell.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'age',
    'setAge',
    'nextAge',
    'alive',
    'setAlive',
    'nextAlive' ]
```

there is

Another Way

```
function AsStandard (colour) {
  this.it = colour;
}

var quacksLikeAStandardDuck =
  new AsStandard(aColourCell);
```

```
AsStandard.prototype.neighbours =
  function neighbours () {
    return this.it.neighbours();
  };

AsStandard.prototype.setNeighbours =
  function setNeighbours (neighbours) {
    this.it.setNeighbours(neighbours);
    return this;
  };
```

```
AsStandard.prototype.alive =
  function alive () {
    return this.it.age() > 0;
  };
```

```
AsStandard.prototype.setAlive =
  function setAlive (alive) {
    if (alive) {
      this.it.setAge(this.it.age() + 1);
    }
    else this.it.setAge(0);
    return this;
  };
```

```
AsStandard.prototype.nextAlive =
  function nextAlive () {
    return this.it.nextAge() > 0;
  }
```

```
Object.keys(AsStandard.prototype)
// =>
  [ 'setNeighbours',
    'alive',
    'setAlive',
    'nextAlive' ]
```


we can go in the
Other Direction

```
function AsColour (standard) {
  this.it = standard;
}

var quacksLikeAColouredDuck =
  new AsColour(aStandardCell);
```

```
AsColour.prototype.neighbours =
  function neighbours () {
    return this.it.neighbours();
  };

AsColour.prototype.setNeighbours =
  function setNeighbours (neighbours) {
    this.it.setNeighbours(neighbours);
    return this;
  };
```

```
AsColour.prototype.age =
  function age () {
    return this.it.alive()
           ? 1
           : 0;
  };
```

```
AsColour.prototype.setAge =
  function setAge (age) {
    this.it.setAlive(age > 0);
    return this;
  };
```

```
AsColour.prototype.nextAge =
  function nextAge () {
    return this.it.nextAlive()
           ? 1
           : 0;
  }
```

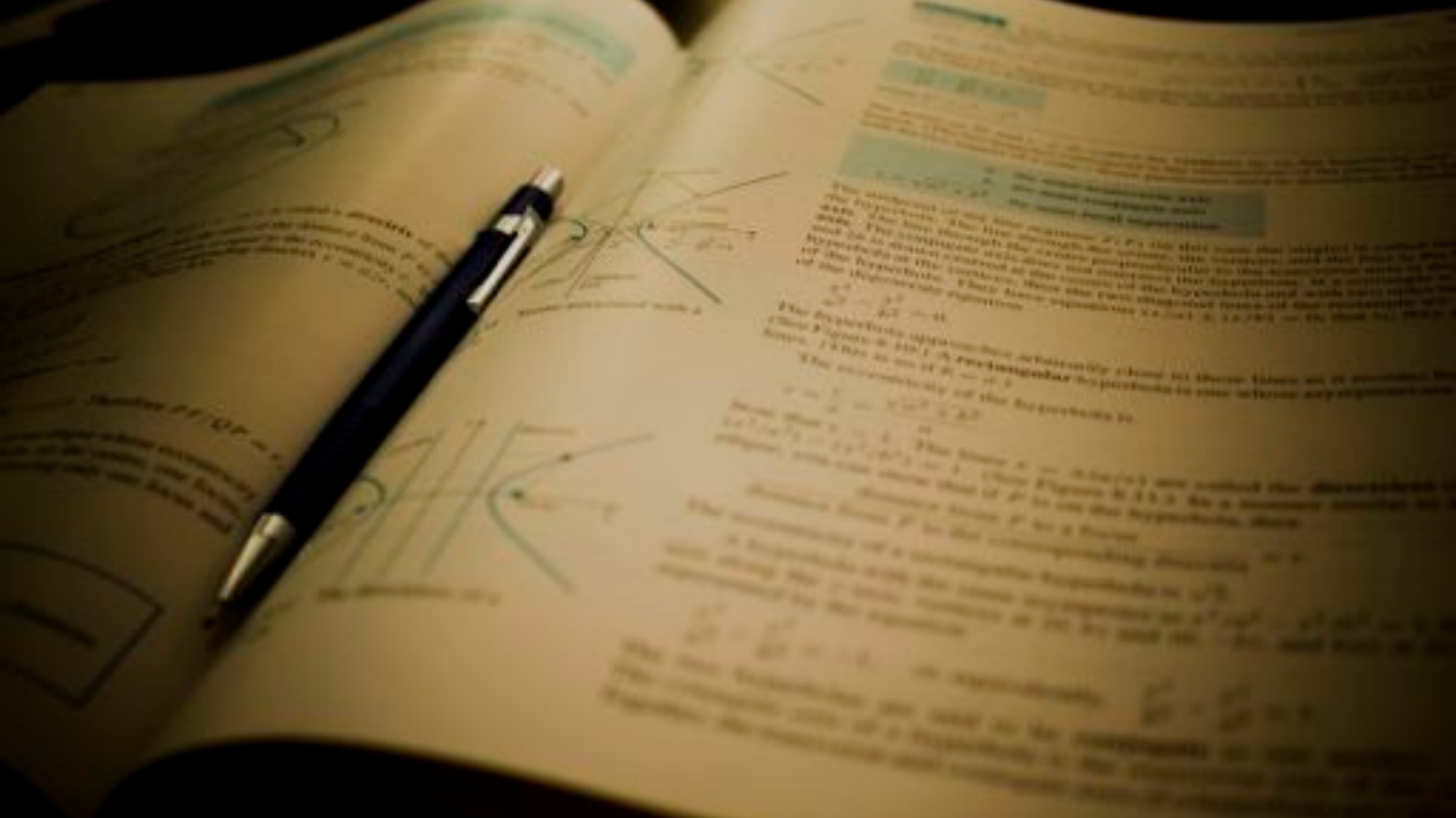
```
Object.keys(AsColour.prototype)
// =>
  [ 'neighbours',
    'setNeighbours',
    'age',
    'setAge',
    'nextAge' ]
```

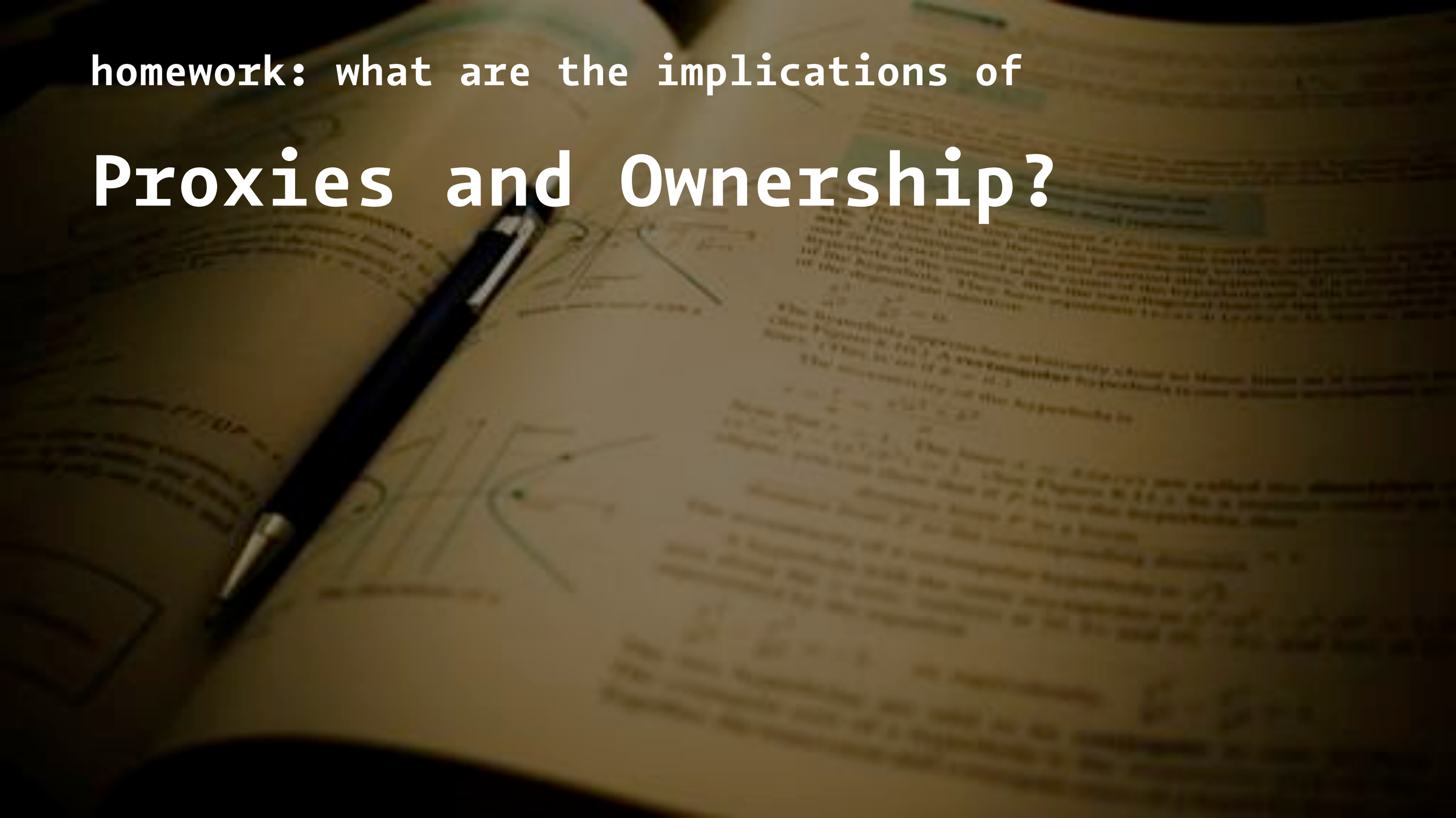
AsStandard and AsColour are

Adaptors

The adapter (sic) pattern is a software design pattern that allows the interface of an existing class to be used from another interface...

It is often used to make existing classes work with others without modifying their source code.

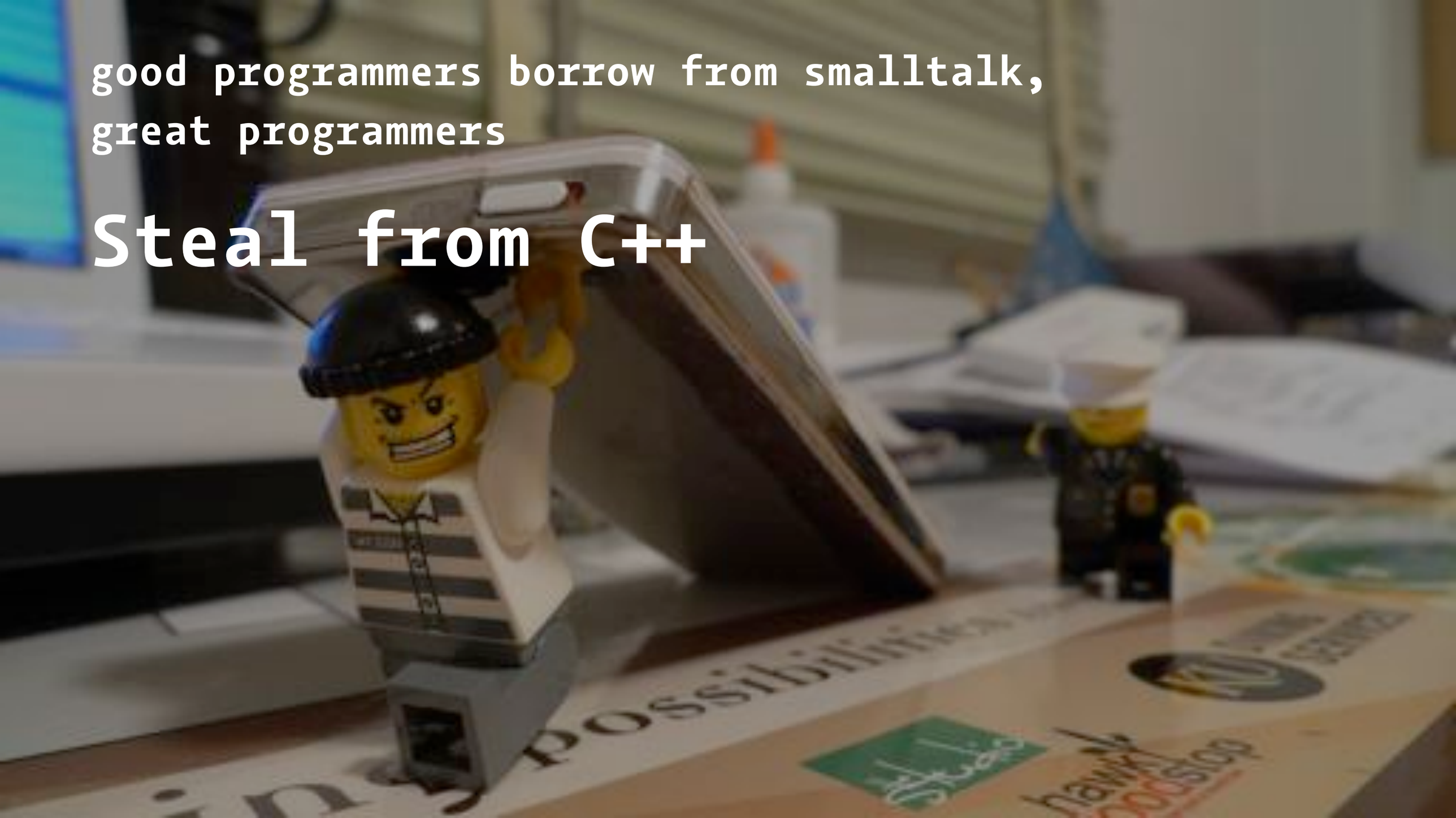




homework: what are the implications of

Proxies and Ownership?


possibilities


good programmers borrow from smalltalk,
great programmers
Steal from C++

```
function colourFromStandard (standard) {
  return new ColourCell()
          .setNeighbours(standard.neighbours())
          .setAge(standard.alive() ? 1 : 0);
}
```

```
function standardFromColour (colour) {
  return new StandardCell()
          .setNeighbours(colour.neighbours())
          .setAlive(colour.age() > 0);
}
```


if you use rails, you'd call these
"Object Migrations"


Hmmm, that's interesting!

What if we could manage change with migrations between versions of classes?


think (thingk). v.. thought, think
—v.i. 1- to use one's mind rationally
to have a certain thing as the subje
one's thoughts thinking about


30

adaptors

Separate Concerns

adaptors

Isolate Change


57309

adaptors

Decouple Modules

adaptors *can* be used to

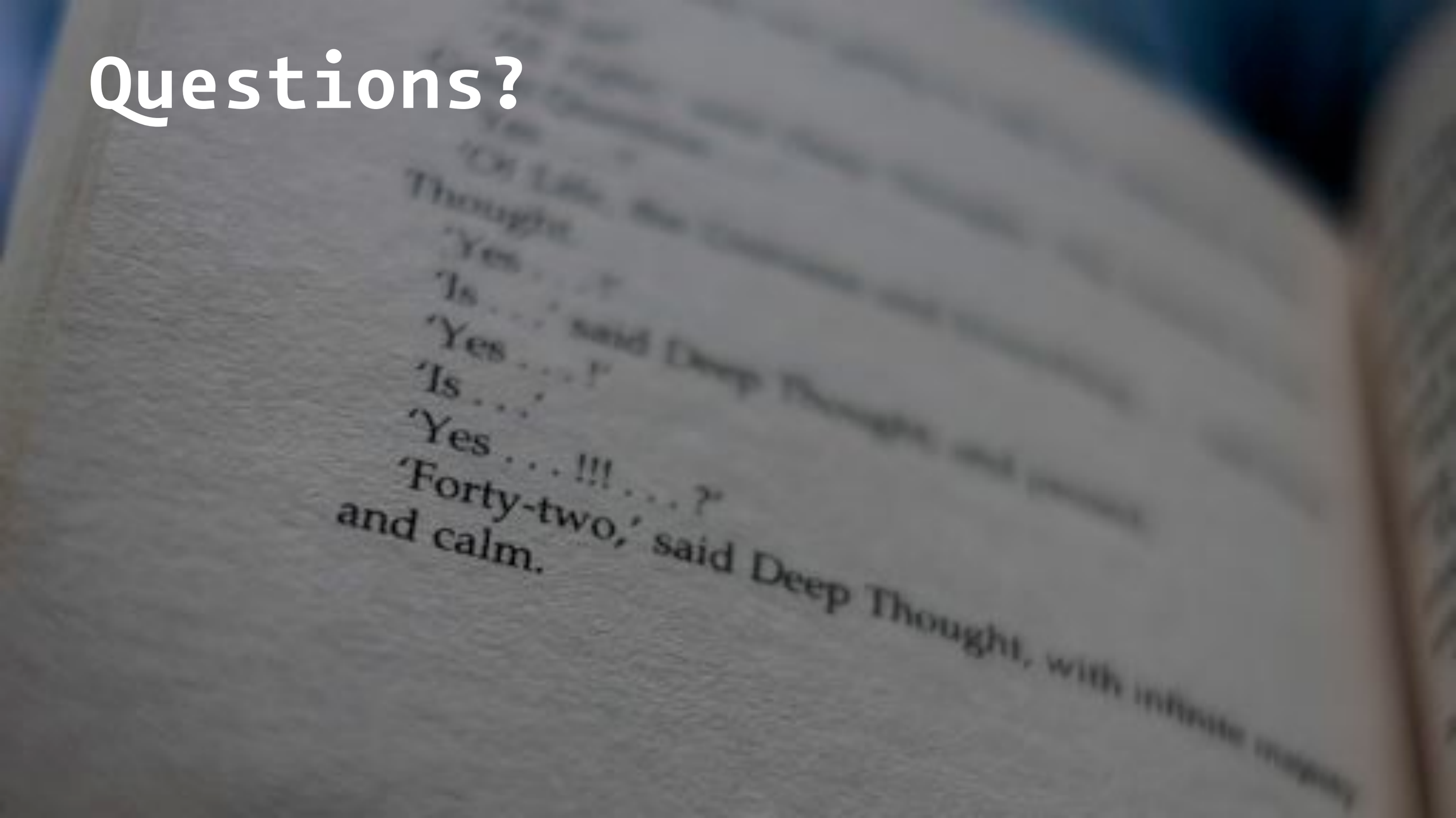
Solve Compatibility Problems

but they also gives us the freedom to

Make Changes, Safely

Tack Så Mycket!

Questions?


'Yes . . . ?'
'Is . . .'
'Yes . . . !!! . . . ?'
'Forty-two,' said Deep Thought, with infinite majesty
and calm.

Reginald Braithwaite

GitHub, Inc.

raganwald.com

@raganwald

Nordic JS
Artipelag, Stockholm,
September 19, 2014