Overview

Miscellaneous left-over JavaScript topics.

- Classes (needed for React components).
- Monkey-patching.
- Destructuring.

Inheritance

- We could implement classical inheritance using a pattern like `Child.prototype = new Parent()`. Hence `Child` will inherit properties from `Parent`.
- Note that we use `new Parent()`, rather than simply `Parent` as we do not want assignments to `Child.prototype` to affect `Parent`.
- Problematic in that we need to apply this pattern. Could wrap within a function `inherit()`, but still messy (see Crockford).
- Also, classical inheritance is generally problematic.

JavaScript Classes

- Added in es6 to make programmers coming in from other languages more comfortable.
- Create a new class using a class declaration.
- Create a new class using a class expression.
- Inheritance using `extends`.
- Static methods.
- Can extend builtin classes.
- Very thin layer around prototype-based inheritance. See this for tradeoffs.

Shapes Example

```
class Shape {
  constructor(x, y) {
    this.x = x; this.y = y;
  }
  static distance(s1, s2) {
    const xDiff = s1.x - s2.x;
    const yDiff = s1.y - s2.y;
    return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
  }
}
```

Shapes Example Continued

```
class Rect extends Shape {
  constructor(x, y, w, h) {
    super(x, y);
    this.width = w; this.height = h;
  }
  area() { return this.width*this.height; }
}

class Circle extends Shape {
  constructor(x, y, r) {
    super(x, y);
    this.radius = r;
  }
  area() { return Math.PI*this.radius*this.radius; }
}
```

Shapes Example Driver and Log

```
const shapes = [
  new Rect(3, 4, 5, 6),
  new Circle(0, 0, 1),
];

shapes.forEach((s) => console.log(s.x, s.y, s.area()));

console.log(Shape.distance(shapes[0], shapes[1]));
```

```
$ ./shapes.js
3 4 30
0 0 3.141592653589793
5
$
```

Class Constants

- Cannot define `const` within a `class`; following results in a syntax error:

```
class C {
  static const constant = 42;
}
```

- Use following pattern:

```
const C = 42;

class C {
  static get constant() { return C; }
}

console.log(C.constant);
```

Monkey Patching to Add a New Function

Built-in types can be changed at runtime: **monkey-patching**.

```
> '  abcd  '.trim()
'abcd'
> '  abcd  '.ltrim()  //trim only on left
TypeError: "  abcd  ".ltrim is not a function
> String.prototype.ltrim =
... String.prototype.ltrim || //do not change
... function() { return this.replace(/^\s+/, ''); }
[Function]
> '  abcd  '.ltrim()
'abcd  '
>
```

Monkey Patching to Modify an Existing Function

```
> const oldFn = String.prototype.replace
undefined
> String.prototype.replace = function(a1, a2) {
    const v = oldFn.call(this, a1, a2);
    console.log(`${this}.replace(${a1}, ${a2})=>${v}`);
    return v;
  }
[Function]
> ' aabcaca'.replace(/aa+/, 'x')
 aabcaca.replace(/aa+/, x)=> xbcaca
' xbcaca'
> ' aabcaca'.replace(/a/g, (x, i) => String(i))
 aabcaca.replace(/a/g, (x, i) => String(i))=> 12bc5c7
' 12bc5c7'
>
```

Array Destructuring Examples

```
> let [a, b] = [22, 42]
undefined
> [a, b]
[ 22, 42 ]
> [a, b] = [b, a] //exchange without temporary
[ 42, 22 ]
> [a, b]
[ 42, 22 ]
> [a, , , b] = [1, 2, 3, 4] //ignored values
[ 1, 2, 3, 4 ]
> [a, b]
[ 1, 4 ]
```

Array Destructuring Examples Continued

```
> let [ x, y = 99] = [42] //default value of 99 for y
undefined
> [x, y]
[ 42, 99 ]
> let [ x1, y1 = 99] = [42, 22] //default not used
undefined
> [x1, y1]
[ 42, 22 ]
>
```

Array Destructuring Examples Continued

```
[a, ...b] = [1, 2, 3, 4] //rest parameters
[ 1, 2, 3, 4 ]
> [a, b]
[ 1, [ 2, 3, 4 ] ]
> [a, b] = [a, ...b] //spreading b
[ 1, 2, 3, 4 ]
> [a, b]
[ 1, 2 ]
> [, a, b] = 'abc3-123'.match(/(\w+).(\d+)/)
[ 'abc3-123', 'abc3', '123', index: 0, input:
'abc3-123' ]
> [a, b]
[ 'abc3', '123' ]
>
```

Object Destructuring Examples

```
let { p, q } = { p: 22, q: 42 }
undefined
> [p, q]
[ 22, 42 ]
{ p, ...rest } = { a: 22, p: 42, b: 33 } //rest params
{ a: 22, p: 42, b: 33 }
> [p, rest]
[ 42, { a: 22, b: 33 } ]
> {p = 33, q = 42 } = { q: 99, a: 44 } //default value
{ q: 99, a: 44 }
> [p, q]
[ 33, 99 ]
//var names different from property names
> { a: p, b: q } = { p: 1, a: 2, q: 3, b: 4 }
{ p: 1, a: 2, q: 3, b: 4 }
> [p, q]
[ 2, 4 ]
```

Combining Object and Array Destructuring

```
> { a: [p, ...q], b: c } = {a: [1, 2, 3], b: 42}
{ a: [ 1, 2, 3 ], b: 42 }
> [p, q, c]
[ 1, [ 2, 3 ], 42 ]
> [ { a, ...b}, c] = [ {a: 2, b: 9, x: 22}, { a: 1}]
[ { a: 2, b: 9, x: 22 }, { a: 1 } ]
> [a, b, c]
[ 2, { b: 9, x: 22 }, { a: 1 } ]
>
```

Function Parameters Destructuring

```
> function f({a, b}) { console.log(a, b); }
undefined
> f({x: 22, b: 2, a: 99, y:2})
99 2
undefined
>
```