Ecma/TC39/2010/046

Generators

Dave Herman

July 28, 2010

Generator functions

```
function fibonacci() {
    var [prev, curr] = [0, 1];
    for (;;) {
        [prev, curr] = [curr, prev + curr];
        yield curr;
    }
}
```

presence of **yield** indicates generator function

Generator objects

create and suspend function activation

```
var g = fibonacci();
print(g.next()); // 1
print(g.next()); // 2
print(g.next()); // 3
print(g.next()); // 5
print(g.next()); // 8
...
```

resume and run to next **yield**

On-demand iteration

```
function values(obj) {
    for (var key in obj) {
        yield obj[key];
    }
}
```

On-demand iteration, ctd.

```
var g = values(['foo', 'bar', 42]);
print(g.next()); // foo
print(g.next()); // bar
print(g.next()); // 42
print(g.next()); // uncaught: StopIteration
```

On-demand iteration, ctd.

```
for (let x in values(['foo', 'bar', 42])) {
    print(x);
}
// ⇒ foo, bar, 42
```

Generators are coroutines

```
function f() {
    try {
        var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.send(42);
```

Generators are coroutines

```
function f() {
    try {
        var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.send(42);
```

Generators are coroutines

```
function f() {
    try {
        var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.send(42);
```

Generators are coroutines

```
function f() {
    try {
        var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.send(42);
```

gimme an x

Generators are coroutines

```
function f() {
    try {
        var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.send(42);
```

Generators are coroutines

```
function f() {
    try {
      → var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.send(42);
```

gimme an x

Generators are coroutines

```
function f() {
    try {
        var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.send(42);
```

received: 42

gimme an x

Generators are coroutines, ctd.

```
function f() {
    try {
        var x = yield “gimme an x”;
        print(“received:” + x);
    } catch (e) {
        print(“caught: ” + e);
    }
}
g = f();
print(g.next());
g.throw(42);
```

caught: 42

Generators

- Generator functions express lazy streams in direct style.
- Generator functions create generator objects.
- Generator objects are iterable.
- Can resume a **yield** with a result or thrown exception.

Generators vs CPS

```
function onError(e) { ... }

XHR.load("x.txt", function(x) {
    XHR.load("y.txt", function(y) {
        XHR.load("z.txt", function(z) {
            ... x, y, z ...
        }, onError);
    }, onError);
}, onError);
```

Generators vs CPS, ctd.

```
t = new Thread(function() {
    try {
        var x = yield XHR.loadAsync(this, “x.js”);
        var y = yield XHR.loadAsync(this, “y.js”);
        var z = yield XHR.loadAsync(this, “z.js”);
        ...
    } catch (e) { ... }
});

t.start();
```

cooperative, not pre-emptive

Generators vs CPS, ctd.

```
XHR.loadAsync = function(thread, url) {
    function onLoad(data) { thread.resume(data); }
    function onError(err) { thread.throw(err); }
    XHR.load(url, onLoad, onError);
};
```

Generators vs CPS, ctd.

```
function Thread(thunk) {
    this.generator = thunk.call(this);
}
Thread.prototype = {
    start: function() { this.generator.next(); },
    resume: function(x) { this.generator.send(x); },
    throw: function(x) { this.generator.throw(x); }
};
```

Generators for MVC

```
function controller() {
    var name = yield “What’s your name?”;
    if (!(yield “Hi, ” + name + “, wanna play a game?”)) {
        yield “Good-bye.”;
        return;
    }
    ...
}
```

Generators for MVC, ctd.

```
function textView(ctrl) {
    var question, answer;
    try {
        for (;;) {
            question = ctrl.send(answer);
            print(question);
            answer = readLine();
        }
    } catch (e) { print("Game over."); }
}
```

Generators for MVC, ctd.

```
function graphicalView(ctrl, answer = undefined) {
    try {
        var question = ctrl.send(answer);
        console.innerHTML = question;
        input.onSubmit = function() {
            graphicalView(ctrl, input.value);
        }
    } catch (e) { console.innerHTML = “Game over.”; }
}
```

Summary

- Popular: Python, Lua, JavaScript 1.7
- Expressive: concise lazy iteration
- Lightweight: less invasive than continuations
- Relevant: direct style for callback-heavy API’s