The JavaScript Language (Part 4)

Copyright © 2024 by

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - A subset of JavaScript…
  - That is appropriate for COS 333…
  - Through example programs

Agenda

- **Arrays**
- Associative arrays
- Asynchronous processing: callbacks

Arrays

- See **arrays.js**

```
$ node arrays.js
[ 'Ruth', 'Gehrig', 'Jeter' ]
3
------------------------------------------------
[ 'Ruth', 'Gehrig', 'Jeter' ]
3
------------------------------------------------
[ 'Ruth', 'Mantle', 'Jeter' ]
3
------------------------------------------------
[ 'Ruth', 'Mantle', 'Jeter', 'Berra' ]
4
------------------------------------------------
[ 'Ruth', 'Mantle', 'Jeter' ]
3
------------------------------------------------
Ruth
Mantle
Jeter
------------------------------------------------
Ruth
Mantle
Jeter
------------------------------------------------
$
```

arrays.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // arrays.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7:
 8: function main() {
 9:    // Creating an array
10:    let a = ['Ruth', 'Gehrig', 'Jeter'];
11:    console.log(a);  // [ 'Ruth', 'Gehrig', 'Jeter' ]
12:    console.log(a.length);  // 3
13:    console.log('-----------------------------------------------');
14:
15:    // Accessing an element
16:    let player = a[1];  // 'Gehrig'
17:    console.log(a);  // [ 'Ruth', 'Gehrig', 'Jeter' ]
18:    console.log(a.length);  // 3
19:    console.log('-----------------------------------------------');
20:
21:    // Changing an element
22:    a[1] = 'Mantle';
23:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter' ]
24:    console.log(a.length);  // 3
25:    console.log('-----------------------------------------------');
26:
27:    // Adding an element to the end
28:    a.push('Berra');
29:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter', 'Berra' ]
30:    console.log(a.length);  // 4
31:    console.log('-----------------------------------------------');
32:
33:    // Removing an element from the end
34:    let element = a.pop();  // 'Berra'
35:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter' ]
36:    console.log(a.length);  // 3
37:    console.log('-----------------------------------------------');
38:
39:    // Iterating over an array (version 1)
40:    for (let i = 0; i < a.length; i++)
41:       console.log(a[i]);
42:       // Ruth
43:       // Mantle
44:       // Jeter
45:    console.log('-----------------------------------------------');
46:
47:    // Iterating over an array (version 2)
48:    for (let element of a)
49:       console.log(element);
50:       // Ruth
51:       // Mantle
52:       // Jeter
53:    console.log('-----------------------------------------------');
54: }
55:
56: if (require.main === module)
57:    main();
```

linesort.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // linesort.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7: const fs = require('fs');
 8:
 9: //-------------------------------------------------------------------------
10:
11: function splitIntoLines(str) {
12:    let lines = [];
13:    let line = '';
14:    for (let c of str) {
15:       if (c === '\n') { lines.push(line); line = ''; }
16:       else line += c;
17:    }
18:    if (line !== '') lines.push(line);
19:    return lines;
20: }
21:
22: //-------------------------------------------------------------------------
23:
24: function main() {
25:
26:    if (process.argv.length !== 3) {
27:       process.stderr.write('Usage: ' +
28:          process.argv[0] + ' ' + process.argv[1] + ' infile\n');
29:       process.exit(1);
30:    }
31:
32:    let fileName = process.argv[2];
33:
34:    try {
35:        let data = fs.readFileSync(fileName, 'UTF-8');
36:        let lines = splitIntoLines(data);
37:        lines.sort();
38:        for (let line of lines)
39:           process.stdout.write(line + '\n');
40:    }
41:    catch (err) {
42:        process.stderr.write(err + '\n');
43:    }
44: }
45:
46: if (require.main === module)
47:    main();
```

Arrays

- An **array** is:
  - An object...
  - That delegates to `Array.prototype`...
  - That has a `length` property...
  - That is maintained automatically…
  - Such that its value is always one greater than the largest integer index (or 0 if the array is empty)

Arrays

- See **linesort.js**

```
$ cat file1
to be
or not to be
that is
the question
$ node linesort.js file1
or not to be
that is
the question
to be
$
```

arrays.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // arrays.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7:
 8: function main() {
 9:    // Creating an array
10:    let a = ['Ruth', 'Gehrig', 'Jeter'];
11:    console.log(a);  // [ 'Ruth', 'Gehrig', 'Jeter' ]
12:    console.log(a.length);  // 3
13:    console.log('-----------------------------------------------');
14:
15:    // Accessing an element
16:    let player = a[1];  // 'Gehrig'
17:    console.log(a);  // [ 'Ruth', 'Gehrig', 'Jeter' ]
18:    console.log(a.length);  // 3
19:    console.log('-----------------------------------------------');
20:
21:    // Changing an element
22:    a[1] = 'Mantle';
23:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter' ]
24:    console.log(a.length);  // 3
25:    console.log('-----------------------------------------------');
26:
27:    // Adding an element to the end
28:    a.push('Berra');
29:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter', 'Berra' ]
30:    console.log(a.length);  // 4
31:    console.log('-----------------------------------------------');
32:
33:    // Removing an element from the end
34:    let element = a.pop();  // 'Berra'
35:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter' ]
36:    console.log(a.length);  // 3
37:    console.log('-----------------------------------------------');
38:
39:    // Iterating over an array (version 1)
40:    for (let i = 0; i < a.length; i++)
41:       console.log(a[i]);
42:       // Ruth
43:       // Mantle
44:       // Jeter
45:    console.log('-----------------------------------------------');
46:
47:    // Iterating over an array (version 2)
48:    for (let element of a)
49:       console.log(element);
50:       // Ruth
51:       // Mantle
52:       // Jeter
53:    console.log('-----------------------------------------------');
54: }
55:
56: if (require.main === module)
57:    main();
```

linesort.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // linesort.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7: const fs = require('fs');
 8:
 9: //-------------------------------------------------------------------------
10:
11: function splitIntoLines(str) {
12:    let lines = [];
13:    let line = '';
14:    for (let c of str) {
15:       if (c === '\n') { lines.push(line); line = ''; }
16:       else line += c;
17:    }
18:    if (line !== '') lines.push(line);
19:    return lines;
20: }
21:
22: //-------------------------------------------------------------------------
23:
24: function main() {
25:
26:    if (process.argv.length !== 3) {
27:       process.stderr.write('Usage: ' +
28:          process.argv[0] + ' ' + process.argv[1] + ' infile\n');
29:       process.exit(1);
30:    }
31:
32:    let fileName = process.argv[2];
33:
34:    try {
35:        let data = fs.readFileSync(fileName, 'UTF-8');
36:        let lines = splitIntoLines(data);
37:        lines.sort();
38:        for (let line of lines)
39:           process.stdout.write(line + '\n');
40:    }
41:    catch (err) {
42:        process.stderr.write(err + '\n');
43:    }
44: }
45:
46: if (require.main === module)
47:    main();
```

Agenda

- Arrays
- **Associative arrays**
- Asynchronous processing: callbacks

Associative Arrays

- See **assocarrays1.js**

```
$ node assocarrays1.js
{ Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
{ Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
{ Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
{ Ruth: 'P', Gehrig: '1B', Jeter: 'SS', Maris: 'RF' }
-----------------------------------------
{ Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
Ruth: P
Gehrig: 1B
Jeter: SS
$
```

assocarrays1.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // assocarrays1.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7:
 8: function main() {
 9:    // Creating an associative array of 3 key-value bindings
10:    let aa = {'Ruth': 'RF', 'Gehrig': '1B', 'Jeter': 'SS'};
11:    console.log(aa);
12:       // { Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
13:    console.log('-----------------------------------------------');
14:
15:    // Accessing a value for a given key
16:    let position = aa['Gehrig'];     // '1B'
17:    console.log(aa);
18:       // { Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
19:    console.log('-----------------------------------------------');
20:
21:    // Changing a value for a given key
22:    aa['Ruth'] = 'P';
23:    console.log(aa);
24:       // { Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
25:    console.log('-----------------------------------------------');
26:
27:    // Adding a binding
28:    aa['Maris'] = 'RF';
29:    console.log(aa);
30:       // { Ruth: 'P', Gehrig: '1B', Jeter: 'SS', Maris: 'RF' }
31:    console.log('-----------------------------------------------');
32:
33:    // Deleting a binding
34:    delete aa['Maris'];
35:    console.log(aa);
36:       // { Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
37:    console.log('-----------------------------------------------');
38:
39:    // Iterating over bindings
40:    for (let key in aa)
41:       console.log(key + ': ' + aa[key]);
42:       // Ruth: P
43:       // Gehrig: 1B
44:       // Jeter: SS
45: }
46:
47: if (require.main === module)
48:    main();
```

assocarrays2.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // assocarrays2.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7:
 8: function main() {
 9:    // Creating an associative array of 3 key-value bindings
10:    let aa = {Ruth: 'RF', Gehrig: '1B', Jeter: 'SS'};
11:    console.log(aa);
12:       // { Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
13:    console.log('-----------------------------------------------');
14:
15:    // Accessing a value for a given key
16:    let position = aa.Gehrig;     // '1B'
17:    console.log(aa);
18:       // { Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
19:    console.log('-----------------------------------------------');
20:
21:    // Changing a value for a given key
22:    aa.Ruth = 'P';
23:    console.log(aa);
24:       // { Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
25:    console.log('-----------------------------------------------');
26:
27:    // Adding a binding
28:    aa.Maris = 'RF';
29:    console.log(aa);
30:       // { Ruth: 'P', Gehrig: '1B', Jeter: 'SS', Maris: 'RF' }
31:    console.log('-----------------------------------------------');
32:
33:    // Deleting a binding
34:    delete aa.Maris;
35:    console.log(aa);
36:       // { Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
37:    console.log('-----------------------------------------------');
38:
39:    // Iterating over bindings
40:    for (let key in aa)
41:       console.log(key + ': ' + aa[key]);
42:       // Ruth: P
43:       // Gehrig: 1B
44:       // Jeter: SS
45: }
46:
47: if (require.main === module)
48:    main();
```

Associative Arrays

- See **assocarrays2.js**

```
$ node assocarrays2.js
{ Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
{ Ruth: 'RF', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
{ Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
{ Ruth: 'P', Gehrig: '1B', Jeter: 'SS', Maris: 'RF' }
-----------------------------------------
{ Ruth: 'P', Gehrig: '1B', Jeter: 'SS' }
-----------------------------------------
Ruth: P
Gehrig: 1B
Jeter: SS
$
```

Uses `object.prop` instead of `assocarray[key]`

Associative Arrays

- The ***member access operator***
  - `object.property`
  - `property` must be a simple identifier
- The ***computed member access operator***
  - `object[property]`
  - `property` can be an arbitrary expression

Associative Arrays

To create and use an **associative array**:

```
aa = {'Ruth': 'RF', 'Gehrig': '1B', …};
…
… aa['Ruth'] …          // Computed member access operator
… aa['Ru' + 'th'] …     // Computed member access operator
… aa.Ruth …             // Member access operator
```

To create and use an **object**:

```
aa = {Ruth: 'RF', Gehrig: '1B', …};
…
… aa.Ruth …             // Member access operator
… aa['Ruth'] …          // Computed member access operator
… aa['Ru' + 'th'] …     // Computed member access operator
```

Associative Arrays

- See **concord.js**

```
$ cat file1
to be
or not to be
that is
the question
$ node concord.js file1
to: 2
be: 2
or: 1
not: 1
that: 1
is: 1
the: 1
question: 1
$
```

concord.js (Page 1 of 1)

```
//-----------------------------------------------------------------------
// concord.js
// Author: Bob Dondero
//-----------------------------------------------------------------------

'use strict';

const fs = require('fs');

//-----------------------------------------------------------------------

function createConcordance(data) {
   let lowercaseData = data.toLowerCase();
   let words = lowercaseData.match(/[a-z]+/g);
   if (words === null)
      words = [];

   let concordance = {};
   for (let word of words)
      if (word in concordance)
         concordance[word]++;
      else
         concordance[word] = 1;
   return concordance;
}

//-----------------------------------------------------------------------

function writeConcordance(concordance) {
   for (let word in concordance)
      process.stdout.write(word + ': ' + concordance[word] + '\n');
}

//-----------------------------------------------------------------------

function main() {
   if (process.argv.length !== 3) {
      process.stderr.write('Usage: ' + process.argv[0] + ' ' +
         process.argv[1] + ' infile\n');
      process.exit(1);
   }

   let fileName = process.argv[2];

   try {
      let data = fs.readFileSync(fileName, 'UTF-8');
      let concordance = createConcordance(data);
      writeConcordance(concordance);
   }
   catch (err) {
       process.stderr.write(err + '\n');
   }
}

if (require.main === module)
   main();
```

linesortcallback.js (Page 1 of 1)

```
//-----------------------------------------------------------------------
// linesortcallback.js
// Author: Bob Dondero
//-----------------------------------------------------------------------

'use strict';
const fs = require('fs');

//-----------------------------------------------------------------------

function splitIntoLines(str) {
   let lines = [];
   let line = '';
   for (let c of str) {
      if (c === '\n') { lines.push(line); line = ''; }
      else line += c;
   }
   if (line !== '') lines.push(line);
   return lines;
}

//-----------------------------------------------------------------------

function reportError(err) {
   process.stderr.write(err.message + '\n');
}

//-----------------------------------------------------------------------

function writeLines(lines) {
   for (let line of lines)
      process.stdout.write(line + '\n');
}

//-----------------------------------------------------------------------

function sortWriteLines(err, data) {
   if (err)
      reportError(err);
   else {
      let lines = splitIntoLines(data);
      lines.sort();
      writeLines(lines);
   }
}

//-----------------------------------------------------------------------

function main() {
   if (process.argv.length !== 3) {
      process.stderr.write('Usage: ' + process.argv[0] + ' ' +
         process.argv[1] + ' infile\n');
      process.exit(1);
   }
   let fileName = process.argv[2];
   fs.readFile(fileName, 'UTF-8', sortWriteLines);
   process.stderr.write('Doing other work\n');
}

if (require.main === module)
   main();
```

Agenda

- Arrays
- Associative arrays
- **Asynchronous processing: callbacks**

Async Processing: Callbacks

- Recall **linesort.js**

```
…
let data = fs.readFileSync(filename, 'UTF-8');
let concordance = createConcordance(data);
…
```

  - `fs.readFileSync()`
    - Reads all data from the file **synchronously**
      - Execution does not proceed until `fs.readFileSync()` returns

arrays.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // arrays.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7:
 8: function main() {
 9:    // Creating an array
10:    let a = ['Ruth', 'Gehrig', 'Jeter'];
11:    console.log(a);  // [ 'Ruth', 'Gehrig', 'Jeter' ]
12:    console.log(a.length);  // 3
13:    console.log('-----------------------------------------------');
14:
15:    // Accessing an element
16:    let player = a[1];  // 'Gehrig'
17:    console.log(a);  // [ 'Ruth', 'Gehrig', 'Jeter' ]
18:    console.log(a.length);  // 3
19:    console.log('-----------------------------------------------');
20:
21:    // Changing an element
22:    a[1] = 'Mantle';
23:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter' ]
24:    console.log(a.length);  // 3
25:    console.log('-----------------------------------------------');
26:
27:    // Adding an element to the end
28:    a.push('Berra');
29:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter', 'Berra' ]
30:    console.log(a.length);  // 4
31:    console.log('-----------------------------------------------');
32:
33:    // Removing an element from the end
34:    let element = a.pop();  // 'Berra'
35:    console.log(a);  // [ 'Ruth', 'Mantle', 'Jeter' ]
36:    console.log(a.length);  // 3
37:    console.log('-----------------------------------------------');
38:
39:    // Iterating over an array (version 1)
40:    for (let i = 0; i < a.length; i++)
41:       console.log(a[i]);
42:       // Ruth
43:       // Mantle
44:       // Jeter
45:    console.log('-----------------------------------------------');
46:
47:    // Iterating over an array (version 2)
48:    for (let element of a)
49:       console.log(element);
50:       // Ruth
51:       // Mantle
52:       // Jeter
53:    console.log('-----------------------------------------------');
54: }
55:
56: if (require.main === module)
57:    main();
```

linesort.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // linesort.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7: const fs = require('fs');
 8:
 9: //-------------------------------------------------------------------------
10:
11: function splitIntoLines(str) {
12:    let lines = [];
13:    let line = '';
14:    for (let c of str) {
15:       if (c === '\n') { lines.push(line); line = ''; }
16:       else line += c;
17:    }
18:    if (line !== '') lines.push(line);
19:    return lines;
20: }
21:
22: //-------------------------------------------------------------------------
23:
24: function main() {
25:
26:    if (process.argv.length !== 3) {
27:       process.stderr.write('Usage: ' +
28:          process.argv[0] + ' ' + process.argv[1] + ' infile\n');
29:       process.exit(1);
30:    }
31:
32:    let fileName = process.argv[2];
33:
34:    try {
35:        let data = fs.readFileSync(fileName, 'UTF-8');
36:        let lines = splitIntoLines(data);
37:        lines.sort();
38:        for (let line of lines)
39:           process.stdout.write(line + '\n');
40:    }
41:    catch (err) {
42:        process.stderr.write(err + '\n');
43:    }
44: }
45:
46: if (require.main === module)
47:    main();
```

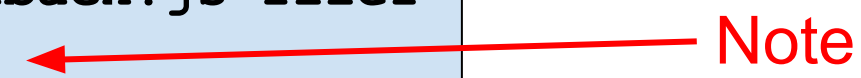
Async Processing: Callbacks

- Recall **linesort.js** (cont.)
  - The more normal approach…
  - `fs.readFile()`
    - Reads all data from the file **asynchronously**
    - Execution proceeds before `fs.readFile()` returns

Async Processing: Callbacks

- See **linesortcallback.js**

```
$ cat file1
to be
or not to be
that is
the question
$ node linesortcallback.js file1
Doing other work
or not to be
that is
the question
to be
$
```



concord.js (Page 1 of 1)

```
//-----------------------------------------------------------------------
// concord.js
// Author: Bob Dondero
//-----------------------------------------------------------------------

'use strict';

const fs = require('fs');

//-----------------------------------------------------------------------

function createConcordance(data) {
   let lowercaseData = data.toLowerCase();
   let words = lowercaseData.match(/[a-z]+/g);
   if (words === null)
      words = [];

   let concordance = {};
   for (let word of words)
      if (word in concordance)
         concordance[word]++;
      else
         concordance[word] = 1;
   return concordance;
}

//-----------------------------------------------------------------------

function writeConcordance(concordance) {
   for (let word in concordance)
      process.stdout.write(word + ': ' + concordance[word] + '\n');
}

//-----------------------------------------------------------------------

function main() {
   if (process.argv.length !== 3) {
      process.stderr.write('Usage: ' + process.argv[0] + ' ' +
         process.argv[1] + ' infile\n');
      process.exit(1);
   }

   let fileName = process.argv[2];

   try {
      let data = fs.readFileSync(fileName, 'UTF-8');
      let concordance = createConcordance(data);
      writeConcordance(concordance);
   }
   catch (err) {
       process.stderr.write(err + '\n');
   }
}

if (require.main === module)
   main();
```

linesortcallback.js (Page 1 of 1)

```
//-----------------------------------------------------------------------
// linesortcallback.js
// Author: Bob Dondero
//-----------------------------------------------------------------------

'use strict';
const fs = require('fs');

//-----------------------------------------------------------------------

function splitIntoLines(str) {
   let lines = [];
   let line = '';
   for (let c of str) {
      if (c === '\n') { lines.push(line); line = ''; }
      else line += c;
   }
   if (line !== '') lines.push(line);
   return lines;
}

//-----------------------------------------------------------------------

function reportError(err) {
   process.stderr.write(err.message + '\n');
}

//-----------------------------------------------------------------------

function writeLines(lines) {
   for (let line of lines)
      process.stdout.write(line + '\n');
}

//-----------------------------------------------------------------------

function sortWriteLines(err, data) {
   if (err)
      reportError(err);
   else {
      let lines = splitIntoLines(data);
      lines.sort();
      writeLines(lines);
   }
}

//-----------------------------------------------------------------------

function main() {
   if (process.argv.length !== 3) {
      process.stderr.write('Usage: ' + process.argv[0] + ' ' +
         process.argv[1] + ' infile\n');
      process.exit(1);
   }
   let fileName = process.argv[2];
   fs.readFile(fileName, 'UTF-8', sortWriteLines);
   process.stderr.write('Doing other work\n');
}

if (require.main === module)
   main();
```

Async Processing: Callbacks

Node.js

JS Engine

```
…
function sortWriteLines(
    err, data) {
    JS statements;
}
…
function main() {
    JS statements;
    fs.readFile(fileName,
        'UTF-8',
        sortWriteLines);
    JS statements;
}
```

JS Event Queue

```
fs.readFile(file, enc,
    callback) {
    C++ statements;
    Queue an event;
}
```

Async Processing: Callbacks

Node.js

JS Engine

```
…
function sortWriteLines(
    err, data) {
    JS statements;
}
…
function main() {
    JS statements;
    fs.readFile(fileName,
        'UTF-8',
        sortWriteLines);
    JS statements;
}
```

JS Event Queue

```
fs.readFile(file, enc,
    callback) {
    C++ statements;
    Queue an event;
}
```

Async Processing: Callbacks

Node.js

JS Engine

```
…
function sortWriteLines(
    err, data) {
    JS statements;
}
…
function main() {
    JS statements;
    fs.readFile(fileName,
        'UTF-8',
        sortWriteLines);
    JS statements;
}
```

JS Event Queue

```
fs.readFile(file, enc,
    callback) {
    C++ statements;
    Queue an event;
}
```

Async Processing: Callbacks

Node.js

JS Engine

```
…
function sortWriteLines(
    err, data) {
    JS statements;
}
…
function main() {
    JS statements;
    fs.readFile(fileName,
        'UTF-8',
        sortWriteLines);
    JS statements;
}
```

JS Event Queue

```
fs.readFile(file, enc,
    callback) {
    C++ statements;
    Queue an event;
}
```

Async Processing: Callbacks

Node.js

JS Engine

```
…
function sortWriteLines(
    err, data) {
    JS statements;
}
…
function main() {
    JS statements;
    fs.readFile(fileName,
        'UTF-8',
        sortWriteLines);
    JS statements;
}
```

JS Event Queue

```
fs.readFile(file, enc,
    callback) {
    C++ statements;
    Queue an event;
}
```

Async Processing: Callbacks

Node.js

JS Engine

```
…
function sortWriteLines(
    err, data) {
    JS statements;
}
…
function main() {
    JS statements;
    fs.readFile(fileName,
        'UTF-8',
        sortWriteLines);
    JS statements;
}
```

JS Event Queue

Event:

sortWriteLines(err, data)

```
fs.readFile(file, enc,
    callback) {
    C++ statements;
    Queue an event;
}
```

Async Processing: Callbacks

Node.js

JS Engine

```
…
function sortWriteLines(
    err, data) {
    JS statements;
}
…
function main() {
    JS statements;
    fs.readFile(fileName,
        'UTF-8',
        sortWriteLines);
    JS statements;
}
```

JS Event Queue

```
fs.readFile(file, enc,
    callback) {
    C++ statements;
    Queue an event;
}
```

Async Processing: Callbacks

Summary of the cycle:

<table>
<tr><td colspan="2">(1) JS Engine handles event</td></tr>
<tr><td colspan="2">(2) JS Engine calls <code>slowfn()</code>, giving it <code>callback()</code></td></tr>
<tr><td>(3) JS Engine continues handling event</td><td>(3) Node.js executes <code>slowfn()</code></td></tr>
<tr><td>(4) JS Engine, when finished handling event, examines JS Event Queue</td><td>(4) Node.js, when finished executing <code>slowfn()</code>, adds event to JS Event Queue to call <code>callback()</code></td></tr>
<tr><td colspan="2">(5) JS Engine removes event from JS Event Queue</td></tr>
<tr><td colspan="2">(6) Go to step (1)</td></tr>
</table>

Summary

- **Python** is:
  - Multithreaded
  - Preemptive
    - OS can context switch at any time
    - Must beware of race conditions

Summary

- **JavaScript** is:
  - Event driven (not multithreaded)
  - Not preemptive
    - Handles each event to completion without interruption
      - So (esp in browsers) event handlers must consume little time
    - Delegates slow tasks to container (browser or node.js)
      - Race conditions can occur only in the sense that the container finishes handling slow tasks at unpredictable times

Summary

- We have covered:
  - Arrays
  - Associative arrays
  - Asynchronous processing
    - Function callbacks

Summary

- JavaScript language summary
  - C/Java-like syntax
  - Many versions
    - Transpilers used routinely
  - Dynamically typed
  - “Never fail” design philosophy

Summary

- JavaScript language summary (cont.)
  - Unusual object model
    - Delegation to prototypes
    - Objects are associative arrays and vice versa
    - ES6 syntax is **much** different from pre-ES6
  - Event driven, not multi-threaded
  - Asynchronous computation is the norm

Commentary

- JavaScript is:
  - Difficult to learn
  - Difficult to use
  - Unavoidable in web applications
  - Worth learning

Summary

- We have covered:
  - A subset of JavaScript…
  - That is appropriate for COS 333…
  - Through example programs
- See also:
  - **Appendix 1**: Asynchronous processing: promises
  - **Appendix 2**: Asynchronous processing: await

Appendix 1: Asynchronous Processing: Promises

Async Processing: Promises

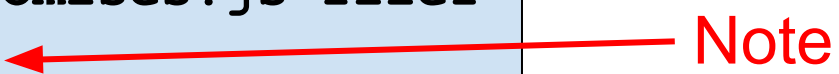
- **Problem**:
  - Programs using (many) callbacks can be difficult to understand

- **Solution…**

Async Processing: Promises

- See **linesortpromises.js**

```
$ cat file1
to be
or not to be
that is
the question
$ node linesortpromises.js file1
Doing other work
or not to be
that is
the question
to be
$
```



linesortpromises.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // linesortpromises.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7: const fs = require('fs');
 8:
 9: //-------------------------------------------------------------------------
10: function splitIntoLines(str) {
11:    let lines = [];
12:    let line = '';
13:    for (let c of str) {
14:       if (c === '\n') { lines.push(line); line = ''; }
15:       else line += c;
16:    }
17:    if (line !== '') lines.push(line);
18:    return lines;
19: }
20:
21: //-------------------------------------------------------------------------
22: function reportError(err) {
23:    process.stderr.write(err.message + '\n');
24: }
25:
26: //-------------------------------------------------------------------------
27: function writeLines(lines) {
28:    for (let line of lines)
29:       process.stdout.write(line + '\n');
30: }
31:
32: //-------------------------------------------------------------------------
33: function sortLines(data) {
34:    let lines = splitIntoLines(data);
35:    lines.sort();
36:    return lines;
37: }
38:
39: //-------------------------------------------------------------------------
40: function main(argv) {
41:    if (argv.length !== 3) {
42:       process.stderr.write(
43:          'Usage: ' + process.argv[0] + ' ' + process.argv[1] +
44:          ' infile\n');
45:       process.exit(1);
46:    }
47:    let fileName = process.argv[2];
48:
49:    // let promise1 = fs.promises.readFile(fileName, 'UTF-8');
50:    // let promise2 = promise1.then(sortLines);
51:    // let promise3 = promise2.then(writeLines);
52:    // promise3.catch(reportError);
53:
54:    fs.promises.readFile(fileName, 'UTF-8')
55:       .then(sortLines)
56:       .then(writeLines)
57:       .catch(reportError);
58:
59:    process.stderr.write('Doing other work\n');
60: }
61:
62: if (require.main === module)
63:    main(process.argv);
```

linesortawait.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // linesortawait.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7: const fs = require('fs');
 8:
 9: //-------------------------------------------------------------------------
10: function splitIntoLines(str) {
11:    let lines = [];
12:    let line = '';
13:    for (let c of str) {
14:       if (c === '\n') { lines.push(line); line = ''; }
15:       else line += c;
16:    }
17:    if (line !== '') lines.push(line);
18:    return lines;
19: }
20:
21: //-------------------------------------------------------------------------
22: function reportError(err) {
23:    process.stderr.write(err.message + '\n');
24: }
25:
26: //-------------------------------------------------------------------------
27: function writeLines(lines) {
28:    for (let line of lines)
29:       process.stdout.write(line + '\n');
30: }
31:
32: //-------------------------------------------------------------------------
33: function sortLines(data) {
34:    let lines = splitIntoLines(data);
35:    lines.sort();
36:    return lines;
37: }
38:
39: //-------------------------------------------------------------------------
40: async function handleFile(fileName) {
41:    try {
42:       let data = await fs.promises.readFile(fileName, 'UTF-8');
43:       let lines = sortLines(data);
44:       writeLines(lines);
45:    }
46:    catch (err) {
47:       reportError(err);
48:    }
49: }
50:
51: //-------------------------------------------------------------------------
52: function main() {
53:    if (process.argv.length !== 3) {
54:       process.stderr.write('Usage: ' + process.argv[0] + ' ' +
55:          process.argv[1] + ' infile\n');
56:       process.exit(1);
57:    }
58:    let fileName = process.argv[2];
59:    handleFile(fileName);
60:    process.stderr.write('Doing other work\n');
61: }
62:
63: if (require.main === module)
64:    main();
```

Async Processing: Promises

- See **linesortpromises.js** (cont.)

```
let promise1 = fs.promises.readFile(fileName, 'UTF-8')
// Let promise1 represent err and data, the future
// result of reading from fileName

let promise2 = promise1.then(sortLines)
// Let promise2 represent lines, the future result of
// calling sortLines(data)

let promise3 = promise2.then(writeLines)
// Let promise3 represent null, the future result
// of calling writeLines(lines)

promise3.catch(reportError);
// If err is not null, then call reportError(err)
```

Async Processing: Promises

- See **linesortpromises.js** (cont.)

```
fs.promises.readFile(
    fileName, 'UTF-8')
    .then(sortLines)
    .then(writeLines)
    .catch(reportError);
```

Dear Node.js:

Call `fs.promises.readFile` asynchronously, passing it `fileName`.

Then, after `readFile` is finished, call `sortLines`, passing it the value returned by `readFile` (i.e., `data`).

Then, after `sortLines` is finished, call `writeLines`, passing it the value returned by `sortLines` (i.e., `lines`).

If `readFile`, `sortLines`, or `writeLines` throws an exception, then call `reportError`, passing it the exception object.

Async Processing: Promises

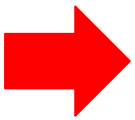
- Promises commentary
  - Difficult to understand the implementation
  - (Usually) easy to use

Appendix 2:
Asynchronous Processing: await

Async Processing: await

- **Await and async**

```
function f(...) {
    initial code
    f1(args)
        .then(f2)
        .then(f3)
        .catch(f4);
    final code
}
```



```
async function helper(args) {
    try {
        let data1 = await f1(args);
        let data2 = f2(data1);
        f3(data2);
    }
    catch (ex) {
        f4(ex);
    }
}

function f(...) {
    initial code
    helper(args);
    final code
}
```

Async Processing: await

- See **linesortawait.js**

```
$ cat file1
to be
or not to be
that is
the question
$ node linesortawait.js file1
Doing other work
or not to be
that is
the question
to be
$
```

linesortpromises.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // linesortpromises.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7: const fs = require('fs');
 8:
 9: //-------------------------------------------------------------------------
10: function splitIntoLines(str) {
11:    let lines = [];
12:    let line = '';
13:    for (let c of str) {
14:       if (c === '\n') { lines.push(line); line = ''; }
15:       else line += c;
16:    }
17:    if (line !== '') lines.push(line);
18:    return lines;
19: }
20:
21: //-------------------------------------------------------------------------
22: function reportError(err) {
23:    process.stderr.write(err.message + '\n');
24: }
25:
26: //-------------------------------------------------------------------------
27: function writeLines(lines) {
28:    for (let line of lines)
29:       process.stdout.write(line + '\n');
30: }
31:
32: //-------------------------------------------------------------------------
33: function sortLines(data) {
34:    let lines = splitIntoLines(data);
35:    lines.sort();
36:    return lines;
37: }
38:
39: //-------------------------------------------------------------------------
40: function main(argv) {
41:    if (argv.length !== 3) {
42:       process.stderr.write(
43:          'Usage: ' + process.argv[0] + ' ' + process.argv[1] +
44:          ' infile\n');
45:       process.exit(1);
46:    }
47:    let fileName = process.argv[2];
48:
49:    // let promise1 = fs.promises.readFile(fileName, 'UTF-8');
50:    // let promise2 = promise1.then(sortLines);
51:    // let promise3 = promise2.then(writeLines);
52:    // promise3.catch(reportError);
53:
54:    fs.promises.readFile(fileName, 'UTF-8')
55:       .then(sortLines)
56:       .then(writeLines)
57:       .catch(reportError);
58:
59:    process.stderr.write('Doing other work\n');
60: }
61:
62: if (require.main === module)
63:    main(process.argv);
```

linesortawait.js (Page 1 of 1)

```
 1: //-------------------------------------------------------------------------
 2: // linesortawait.js
 3: // Author: Bob Dondero
 4: //-------------------------------------------------------------------------
 5:
 6: 'use strict';
 7: const fs = require('fs');
 8:
 9: //-------------------------------------------------------------------------
10: function splitIntoLines(str) {
11:    let lines = [];
12:    let line = '';
13:    for (let c of str) {
14:       if (c === '\n') { lines.push(line); line = ''; }
15:       else line += c;
16:    }
17:    if (line !== '') lines.push(line);
18:    return lines;
19: }
20:
21: //-------------------------------------------------------------------------
22: function reportError(err) {
23:    process.stderr.write(err.message + '\n');
24: }
25:
26: //-------------------------------------------------------------------------
27: function writeLines(lines) {
28:    for (let line of lines)
29:       process.stdout.write(line + '\n');
30: }
31:
32: //-------------------------------------------------------------------------
33: function sortLines(data) {
34:    let lines = splitIntoLines(data);
35:    lines.sort();
36:    return lines;
37: }
38:
39: //-------------------------------------------------------------------------
40: async function handleFile(fileName) {
41:    try {
42:       let data = await fs.promises.readFile(fileName, 'UTF-8');
43:       let lines = sortLines(data);
44:       writeLines(lines);
45:    }
46:    catch (err) {
47:       reportError(err);
48:    }
49: }
50:
51: //-------------------------------------------------------------------------
52: function main() {
53:    if (process.argv.length !== 3) {
54:       process.stderr.write('Usage: ' + process.argv[0] + ' ' +
55:          process.argv[1] + ' infile\n');
56:       process.exit(1);
57:    }
58:    let fileName = process.argv[2];
59:    handleFile(fileName);
60:    process.stderr.write('Doing other work\n');
61: }
62:
63: if (require.main === module)
64:    main();
```