Route Instances

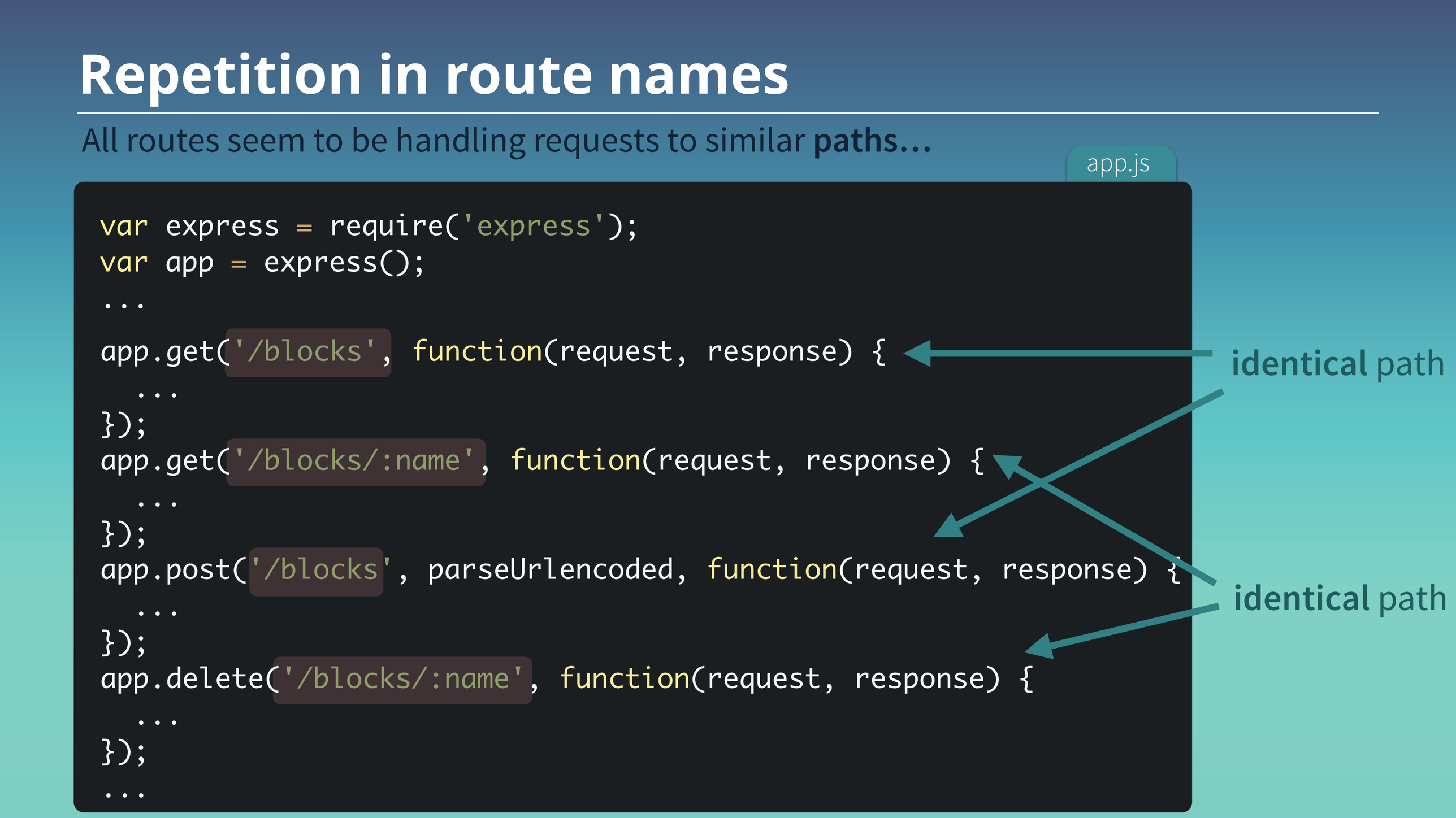
Level 5 - Part I

Repetition in route names

All routes seem to be handling requests to similar **paths...**

app.js

```
var express = require('express');
var app = express();
...
app.get('/blocks', function(request, response) {
  ...
});
app.get('/blocks/:name', function(request, response) {
  ...
});
app.post('/blocks', parseUrlencoded, function(request, response) {
  ...
});
app.delete('/blocks/:name', function(request, response) {
  ...
});
...
```



Replacing repetition with a route instance

Using **app.route** is a recommended approach for avoiding **duplicate** route names

app.js

```
var express = require('express');
var app = express();
...
var blocksRoute = app.route('/blocks')



...
app.listen(3000);
```

returns route object which handles all requests to the **/blocks** path

Routes that act on /blocks

No **path** argument is needed for route handlers belonging to the same route instance

app.js

```
var express = require('express');
var app = express();
...

var blocksRoute = app.route('/blocks')
blocksRoute.get(function(request, response) {
  ...
});
blocksRoute.post(parseUrlencoded, function(request, response) {
  ...
});

...
app.listen(3000);
```

used to be

```
app.get('/blocks'...
```

used to be

```
app.post('/blocks'...
```

Removing intermediate variables

There's unnecessary repetition of the **blocksRoute** variable

app.js

```
var express = require('express');
var app = express();
...

var blocksRoute = app.route('/blocks')
blocksRoute.get(function(request, response) {
  ...
});
blocksRoute.post(parseUrlencoded, function(request, response) {
  ...
});


...
app.listen(3000);
```

Chaining function calls on route

Chaining functions can eliminate **intermediate variables** and help our code stay more **readable**. This is a pattern commonly found in Express applications.

app.js

```
var express = require('express');
var app = express();
...
app.route('/blocks')
  .get(function(request, response) {
    ...
  });
  .post(parseUrlencoded, function(request, response) {
    ...
  });
...
```

no semi-colon at the end of the line

chaining means calling functions on the return value of previous functions

lines starting with **dot** indicate function calls on the object returned from the **previous line**

Dynamic route instances

The **app.route** function accepts the same url argument format as before

app.js

```
var express = require('express');
var app = express();
...

app.route('/blocks')
   .get(function(request, response) {
      ...
   });
   .post(parseUrlencoded, function(request, response) {
      ...
   });
app.route('/blocks/:name')
```

returns route object which handles all requests to the **/blocks/:name** path

Routes that act on /blocks/:name

Our route handlers for **/blocks/:name** reference **blocks** fetched by their **name**

app.js

```
var express = require('express');
var app = express();
...

app.route('/blocks')
...

app.route('/blocks/:name')
  .get(function(request, response) {
    ...
  })
  .delete(function(request, response) {
    ...
  });

app.listen(3000);
```

used to be

```
app.get('/blocks/:name/'...
```

```
app.delete('/blocks/:name'...
```

used to be