Strict Mode

Introduction

- Strict mode is a restricted variant of JS, i.e. it has different (stricter) rules

- Three categories of difference:

1. Errors in place of silent failures
2. Fixes enabling optimisation
3. Prohibited use of keywords of the future

- Q. Why? A. Primarily to make your code more robust

Invocation

- At the script level: `'use strict';`
- At the function level, e.g.:

```
function myStrictFunction() {
  'use strict';
  …
}
```

- Don't concatenate strict and non-strict scripts

Errors

- Accidental global, e.g. `x = 1;`
- Assignment to a non-writeable property
- Deletion of a non-configurable property
- Property name/parameter duplication
- Octal literals
- Properties on primitives

Fixes

- `with` is a syntax error
- `eval` does not introduce new variables into the surrounding scope
- Deletion of plain names is a syntax error
- `arguments` is not aliased; e.g. `arguments[0]` is not an alias for the first parameter
- Use of `caller` and `callee` is a TypeError

Keywords of the Future

- The following may not be used as identifiers:
  - implements
  - interface
  - package
  - private
  - protected
  - public
  - static
  - yield

Summary

- Strict mode is a restricted variant of JS
- Enable at the script/function level via `'use strict';`
- Some silent failures in regular JS are errors in strict mode, e.g. property/parameter name duplication
- Strict mode fixes enable optimisations, e.g. eval doesn't introduce variables into the surrounding scope
- Strict mode prohibits use of keywords of the future, e.g. implements and interface