This example demonstrates Scope Hoisting in combination with Code Splitting.

This is the dependency graph for the example: (solid lines express sync imports, dashed lines async imports)

All modules except `cjs` are EcmaScript modules. `cjs` is a CommonJs module.

The interesting thing here is that putting all modules in single scope won't work, because of multiple reasons:

- Modules `lazy`, `c`, `d` and `cjs` need to be in a separate chunk
- Module `shared` is accessed by two chunks (different scopes)
- Module `cjs` is a CommonJs module

webpack therefore uses a approach called **"Partial Scope Hoisting"** or "Module concatenation", which chooses the largest possible subsets of ES modules which can be scope hoisted and combines them with the default webpack primitives.

While module concatenation identifiers in modules are renamed to avoid conflicts and internal imports are simplified. External imports and exports from the root module use the existing ESM constructs.

example.js

```
_{{example.js}}_
```

lazy.js

```
_{{lazy.js}}_
```

a.js

```
_{{node_modules/a.js}}_
```

b.js

```
_{{node_modules/b.js}}_
```

c.js

```
_{{node_modules/c.js}}_
```

d.js

```
_{{node_modules/d.js}}_
```

cjs.js

```
_{{node_modules/cjs.js}}_
```

shared.js

```
_{{node_modules/shared.js}}_
```

shared2.js

```
_{{node_modules/shared2.js}}_
```

webpack.config.js

```
_{{webpack.config.js}}_
```

dist/output.js

```
_{{dist/output.js}}_
```

dist/872.output.js

```
_{{dist/872.output.js}}_
```

Minimized

```
_{{production:dist/872.output.js}}_
```

Info

Unoptimized

```
_{{stdout}}_
```

Production mode

```
_{{production:stdout}}_
```