This example illustrates a very simple case of Code Splitting with `require.ensure`.

- `a` and `b` are required normally via CommonJS
- `c` is made available(,but doesn't get execute) through the `require.ensure` array.
  - webpack will load it on demand
- `b` and `d` are required via CommonJs in the `require.ensure` callback
  - webpack detects that these are in the on-demand-callback and
  - will load them on demand
  - webpack's optimizer can optimize `b` away
    - as it is already available through the parent chunks

You can see that webpack outputs two files/chunks:

- `output.js` is the entry chunk and contains
  - the module system
  - chunk loading logic
  - the entry point `example.js`
  - module `a`
  - module `b`
- `1.output.js` is an additional chunk (on-demand loaded) and contains
  - module `c`
  - module `d`

You can see that chunks are loaded via JSONP. The additional chunks are pretty small and minimize well.

## example.js

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

## dist/output.js

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

## dist/796.output.js

```
_{{dist/796.output.js}}_
```

Minimized

```
_{{production:dist/796.output.js}}_
```

# Info

## Unoptimized

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

## Production mode

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