New Vue.
New Compiler.

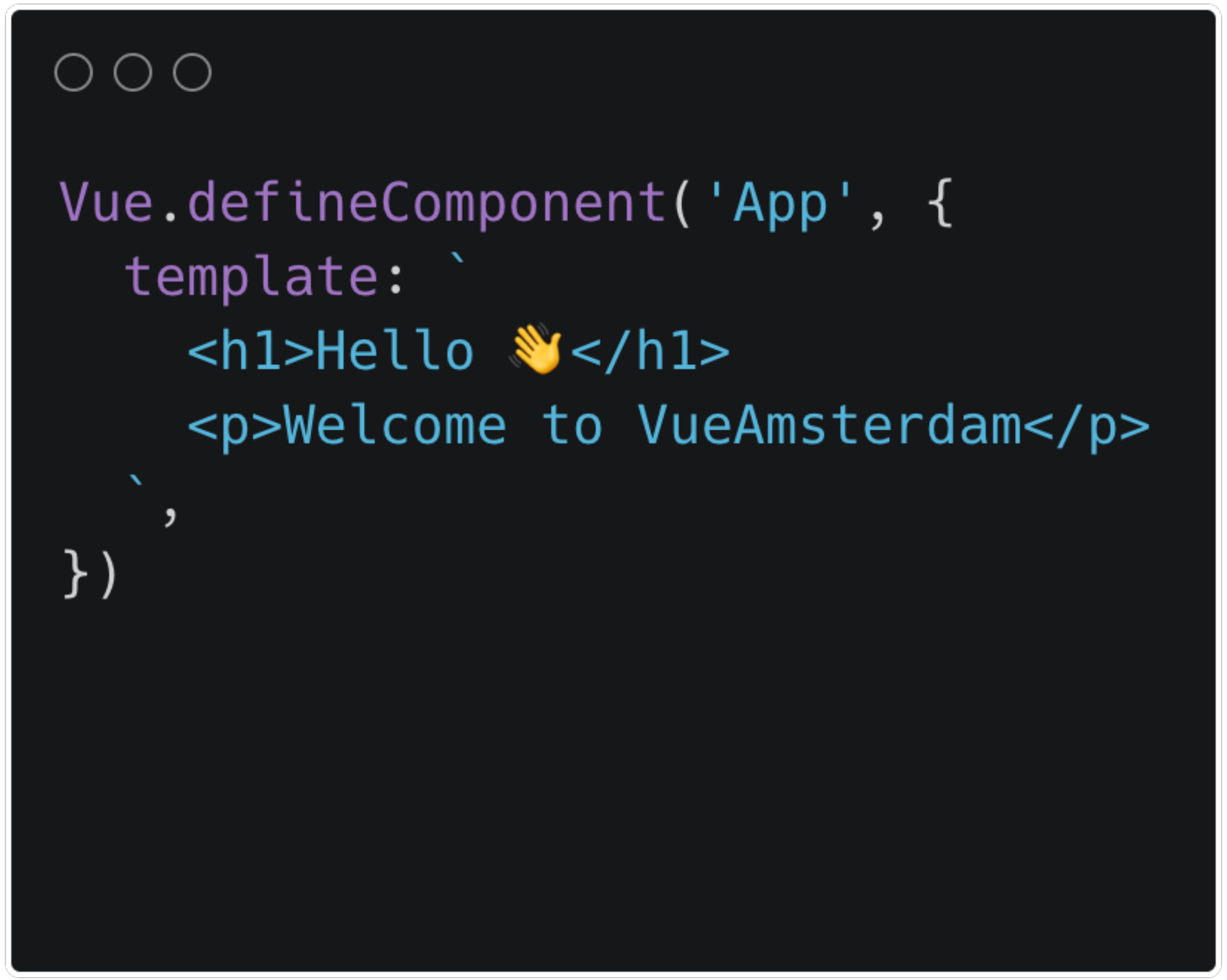
Let’s Unbox

Template Compiler

SFC Compiler

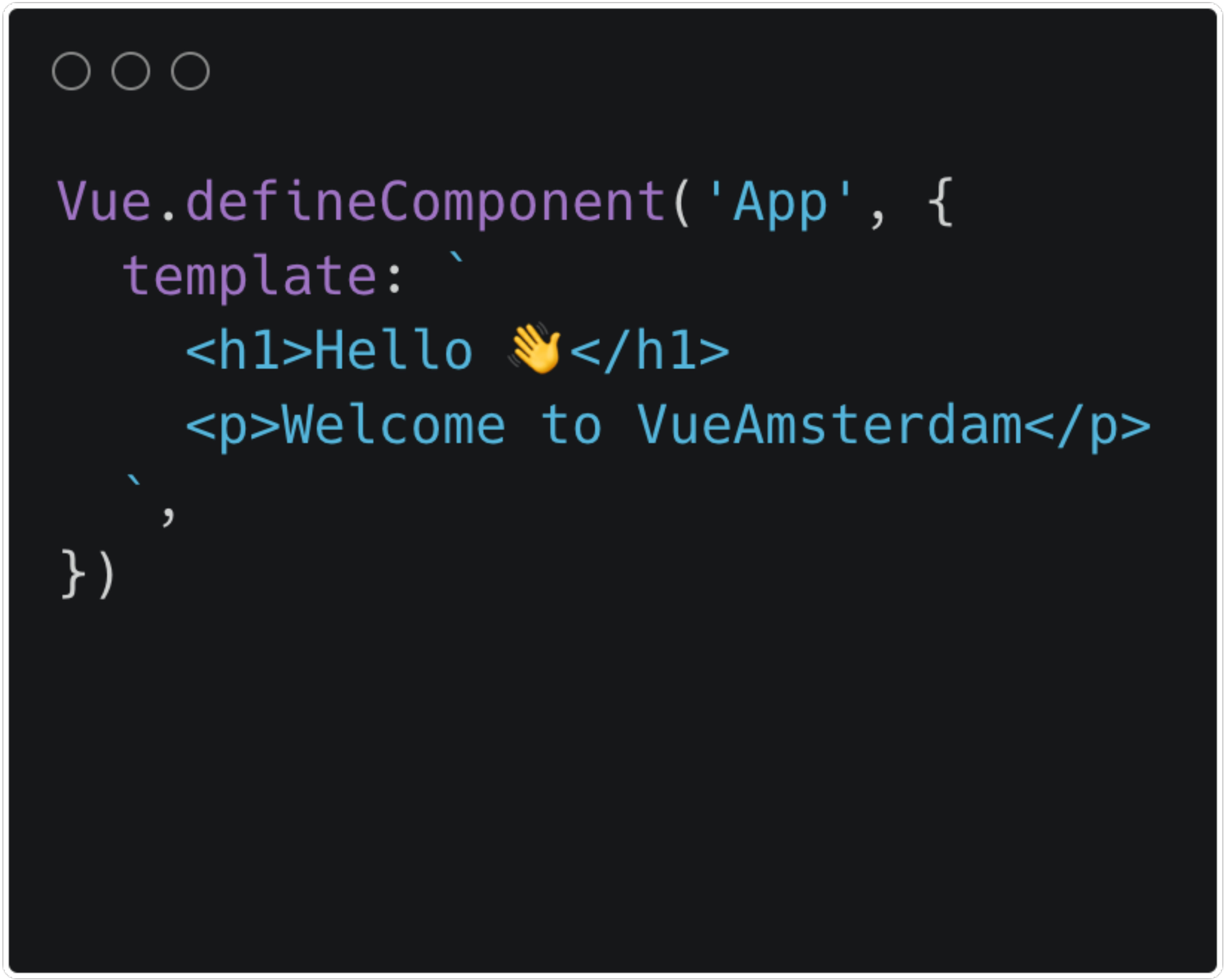
Template Compiler

Template Compiler

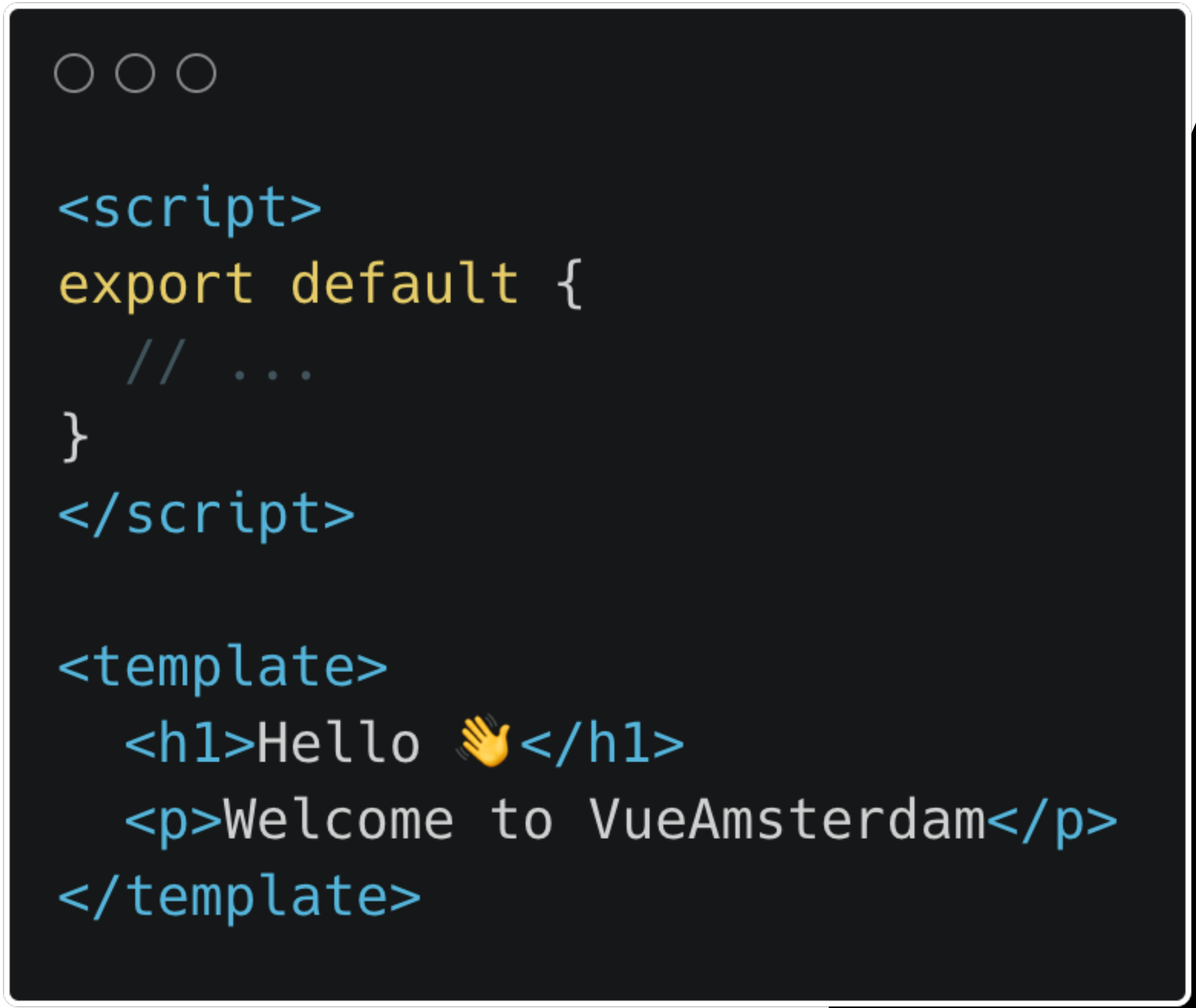


```
Vue.defineComponent('App', {
  template: `
    <h1>Hello 👋</h1>
    <p>Welcome to VueAmsterdam</p>
  `,
})
```

Template Compiler

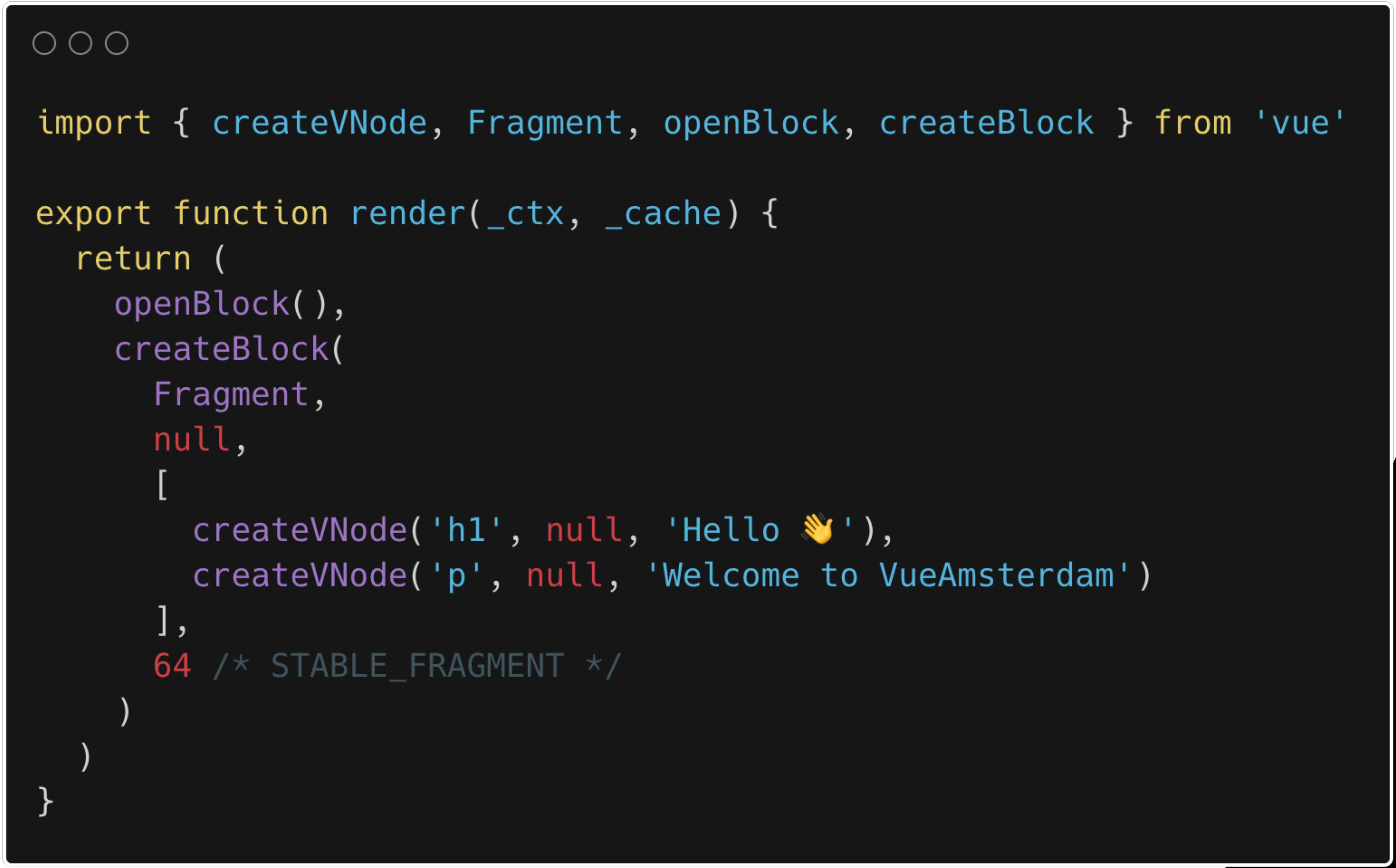


```
Vue.defineComponent('App', {
  template: `
    <h1>Hello 👋</h1>
    <p>Welcome to VueAmsterdam</p>
  `,
})
```



```
<script>
export default {
  // ...
}
</script>

<template>
  <h1>Hello 👋</h1>
  <p>Welcome to VueAmsterdam</p>
</template>
```

Template Compiler



```
import { createVNode, Fragment, openBlock, createBlock } from 'vue'

export function render(_ctx, _cache) {
  return (
    openBlock(),
    createBlock(
      Fragment,
      null,
      [
        createVNode('h1', null, 'Hello 👋'),
        createVNode('p', null, 'Welcome to VueAmsterdam')
      ],
      64 /* STABLE_FRAGMENT */
    )
  )
}
```

Template Compiler

`@vue/compiler-core`

`@vue/compiler-dom`

`@vue/compiler-ssr`

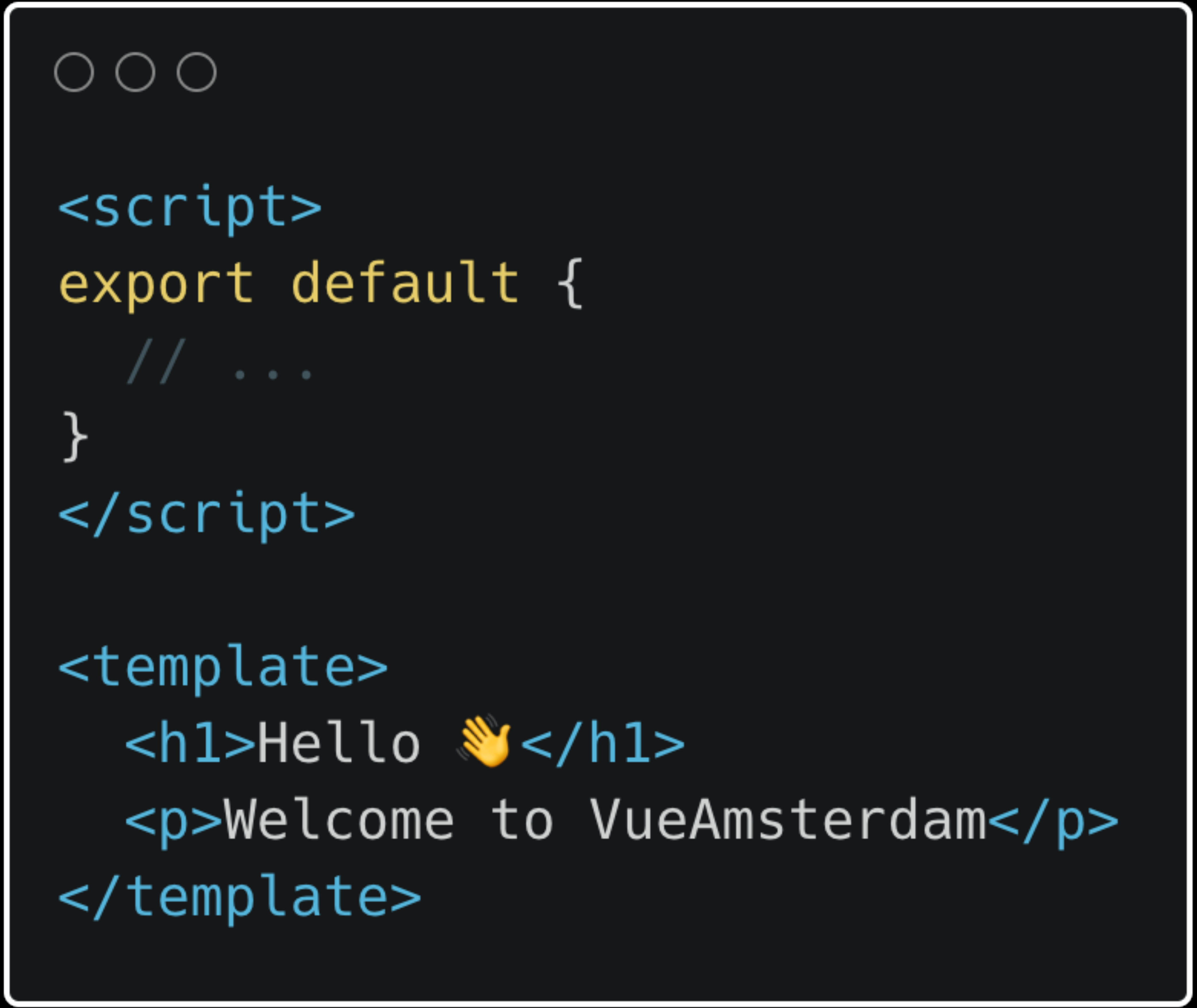
Template Compiler

SFC Compiler

SFC Compiler

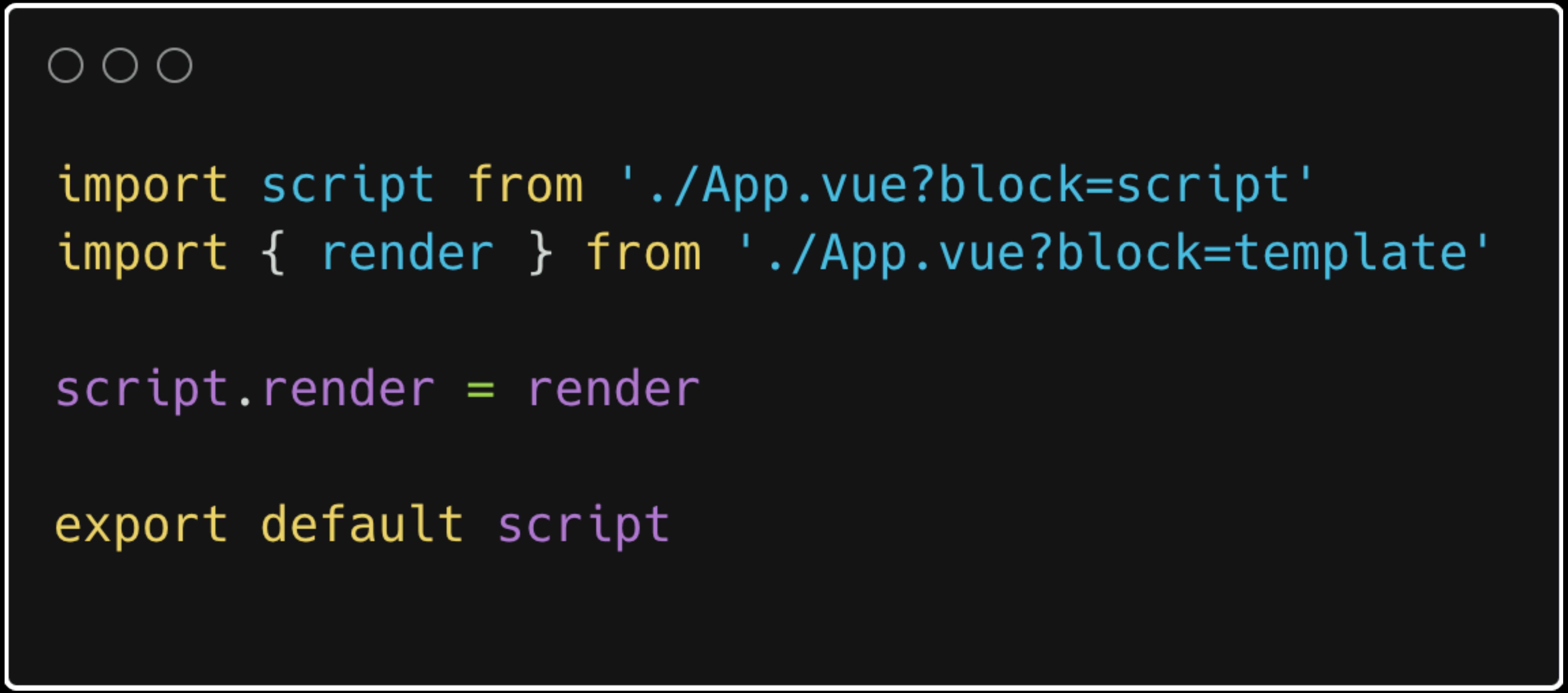
Single File Component Compiler

SFC Compiler



```
<script>
export default {
  // ...
}
</script>

<template>
  <h1>Hello 👋</h1>
  <p>Welcome to VueAmsterdam</p>
</template>
```

SFC Compiler



```
import script from './App.vue?block=script'
import { render } from './App.vue?block=template'

script.render = render

export default script
```

SFC Compiler

`@vue/compiler-sfc`

`vue-loader`

`rollup-pluign-vue`

Template Compiler

SFC Compiler

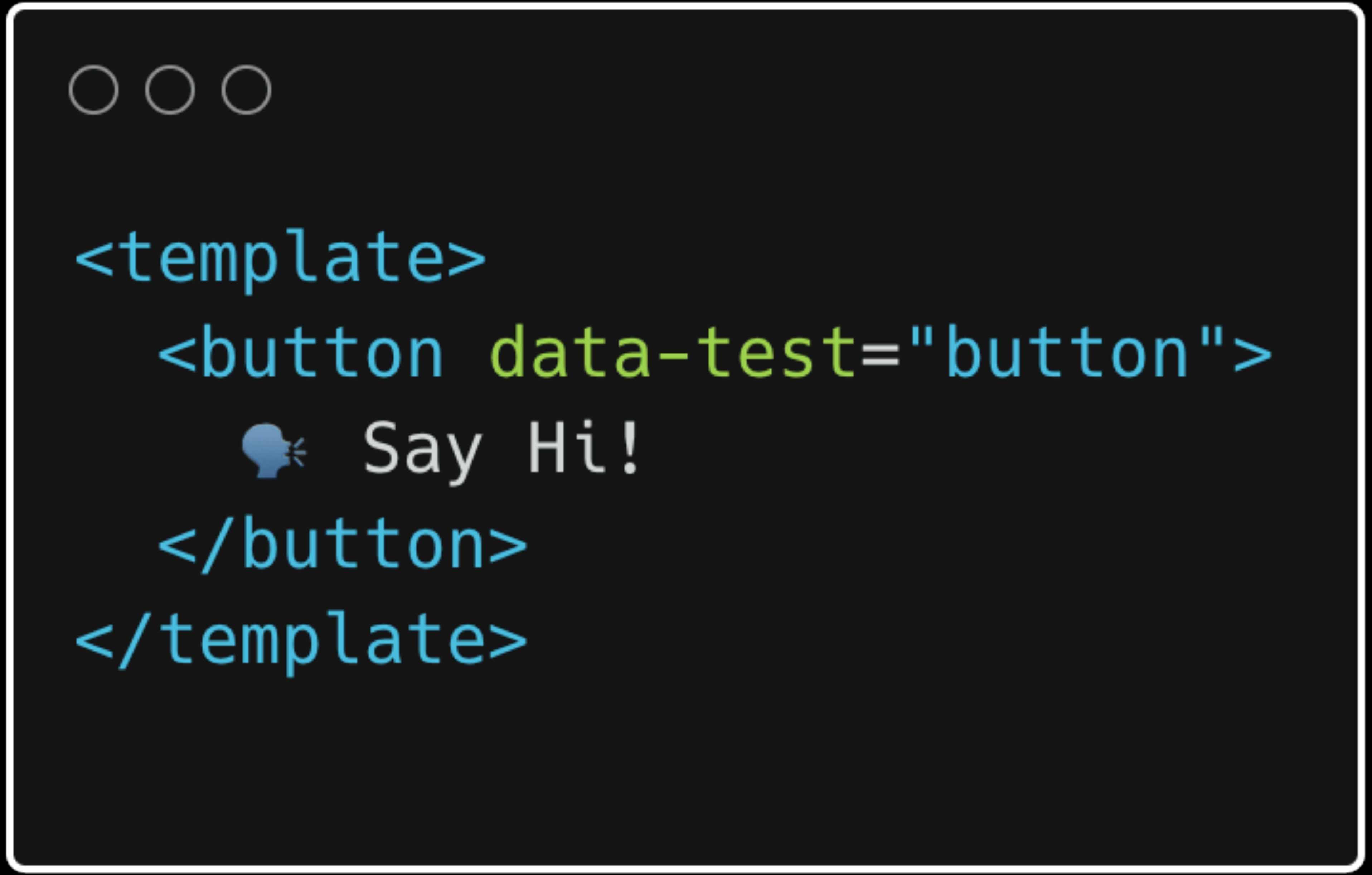
Why should **you** care?

Escape Hatch in Unit Test

```
data-test="*"
```

Escape Hatch in Unit Test

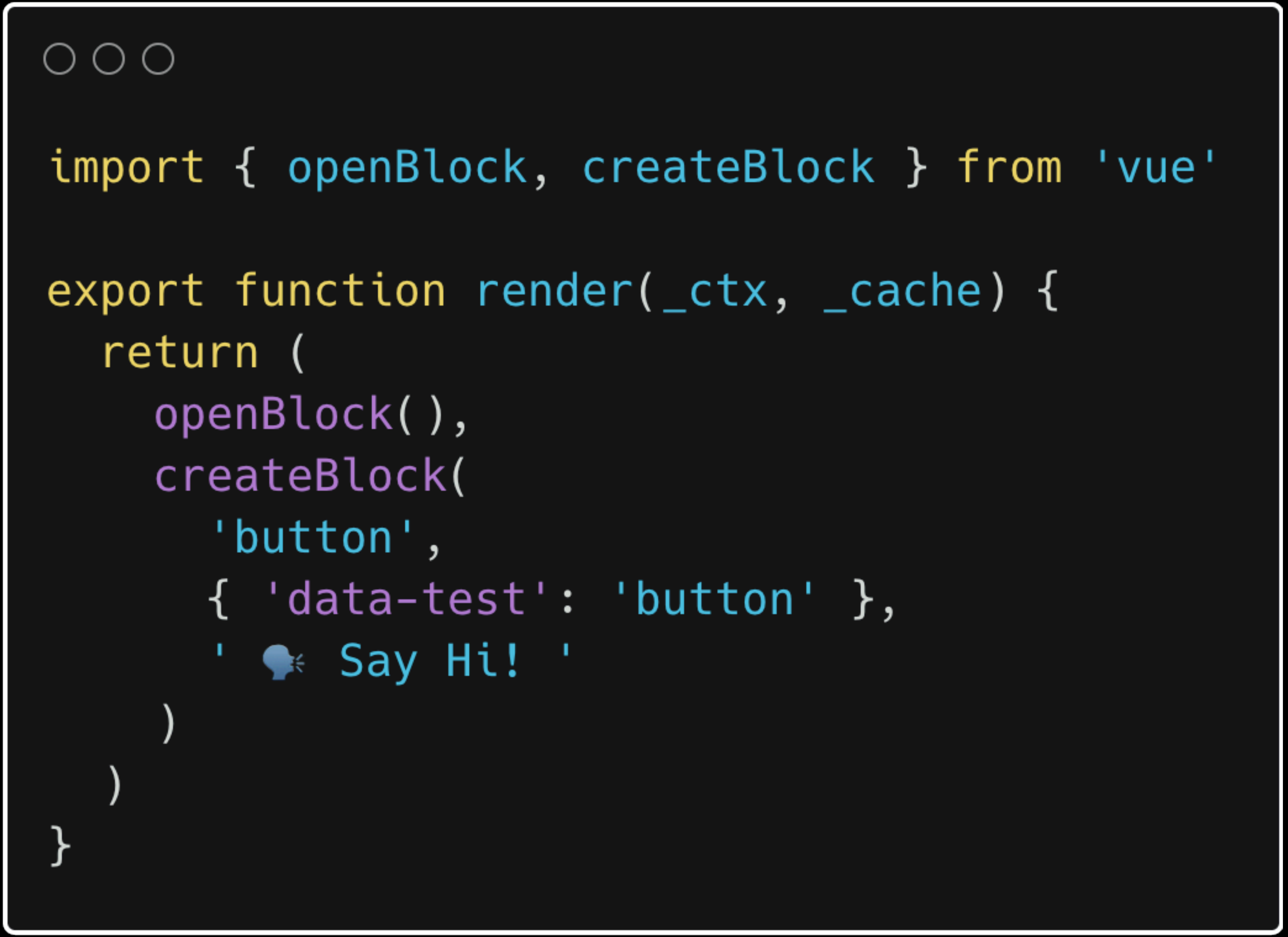
`data-test="*"`



```
<template>
  <button data-test="button">
    🗣 Say Hi!
  </button>
</template>
```

Escape Hatch in Unit Test

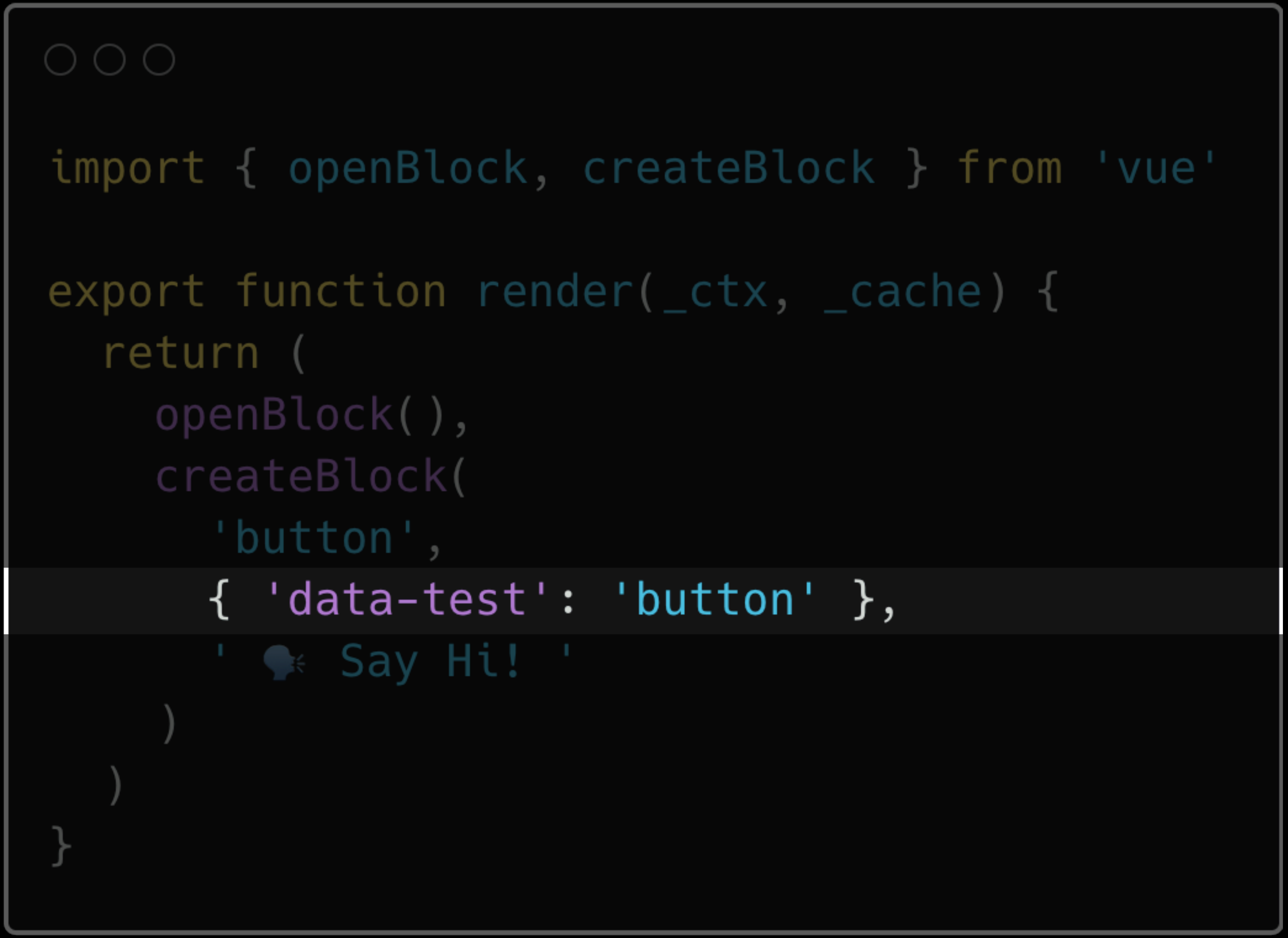
`data-test="*"`



```
import { openBlock, createBlock } from 'vue'

export function render(_ctx, _cache) {
  return (
    openBlock(),
    createBlock(
      'button',
      { 'data-test': 'button' },
      ' 🗣 Say Hi! '
    )
  )
}
```

Escape Hatch in Unit Test

`data-test="*"`



```
import { openBlock, createBlock } from 'vue'

export function render(_ctx, _cache) {
  return (
    openBlock(),
    createBlock(
      'button',
      { 'data-test': 'button' },
      ' 🗣 Say Hi! '
    )
  )
}
```

Escape Hatch in Unit Test

`data-test="*"`

Template Compiler

Escape Hatch in Unit Test

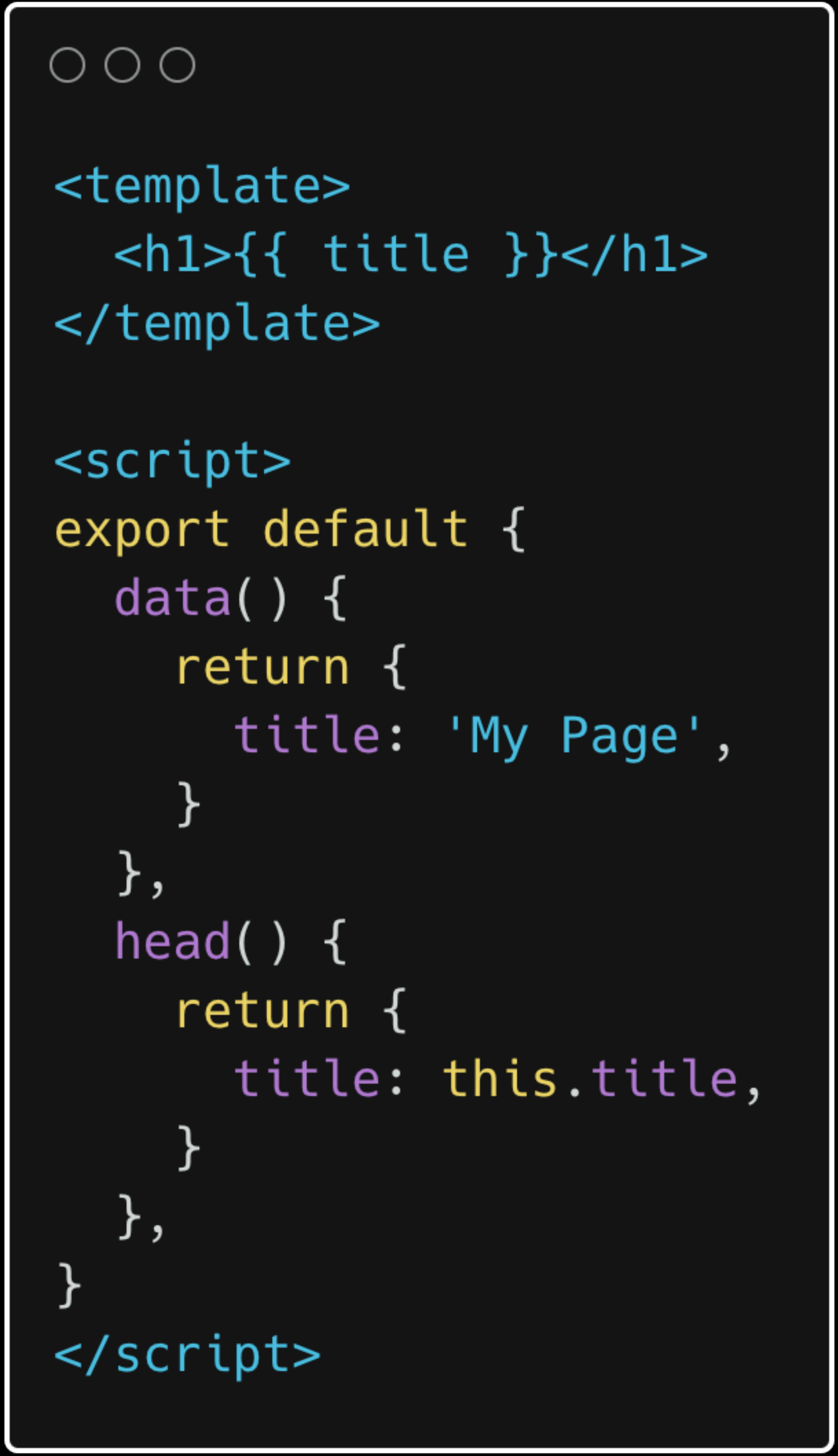
```
data-test="*"
```

Head Meta in SSR

```
<title>My Page</title>
```

Head Meta in SSR

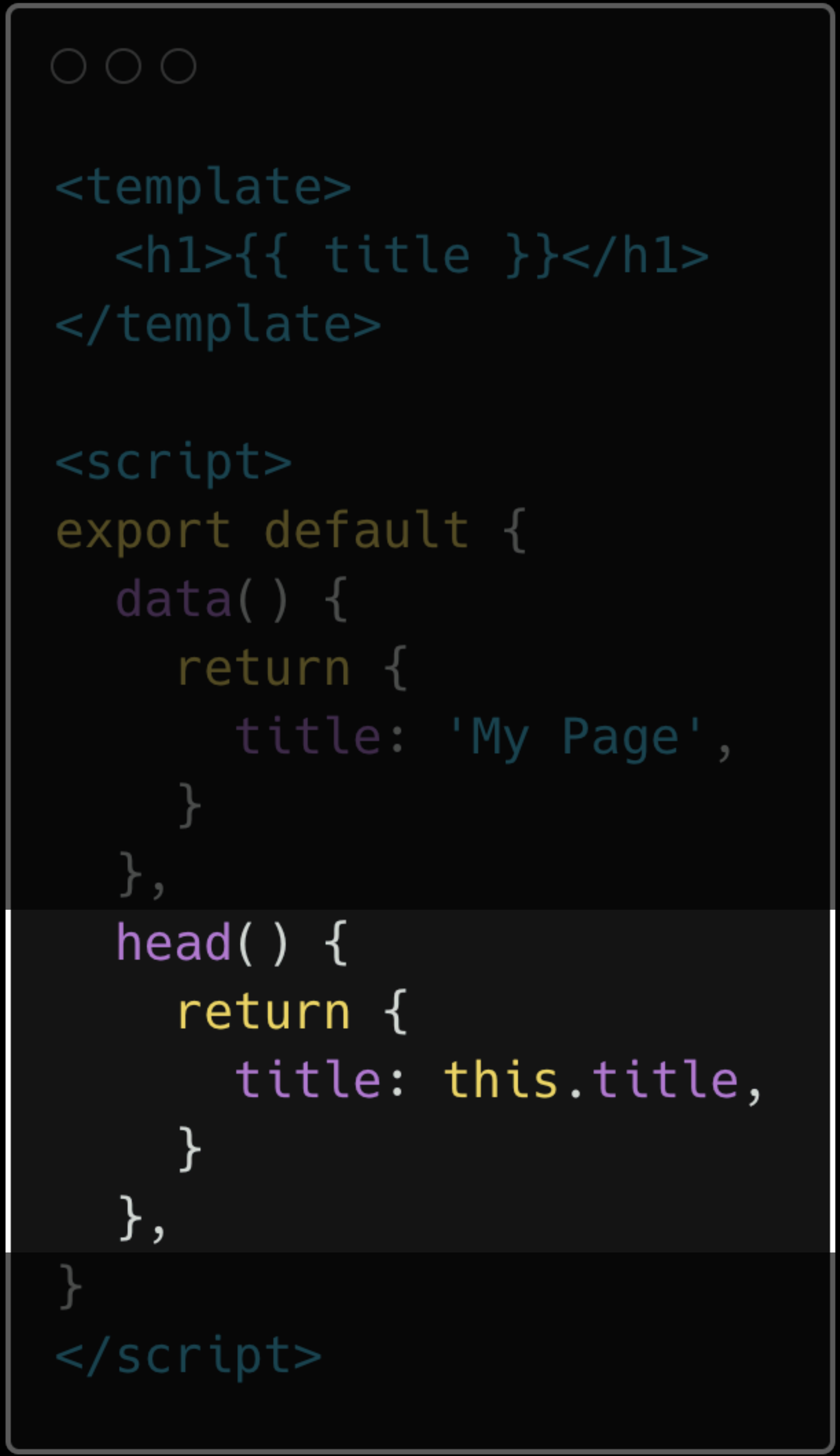
```
<title>My Page</title>
```



```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
  head() {
    return {
      title: this.title,
    }
  },
}
</script>
```

Head Meta in SSR

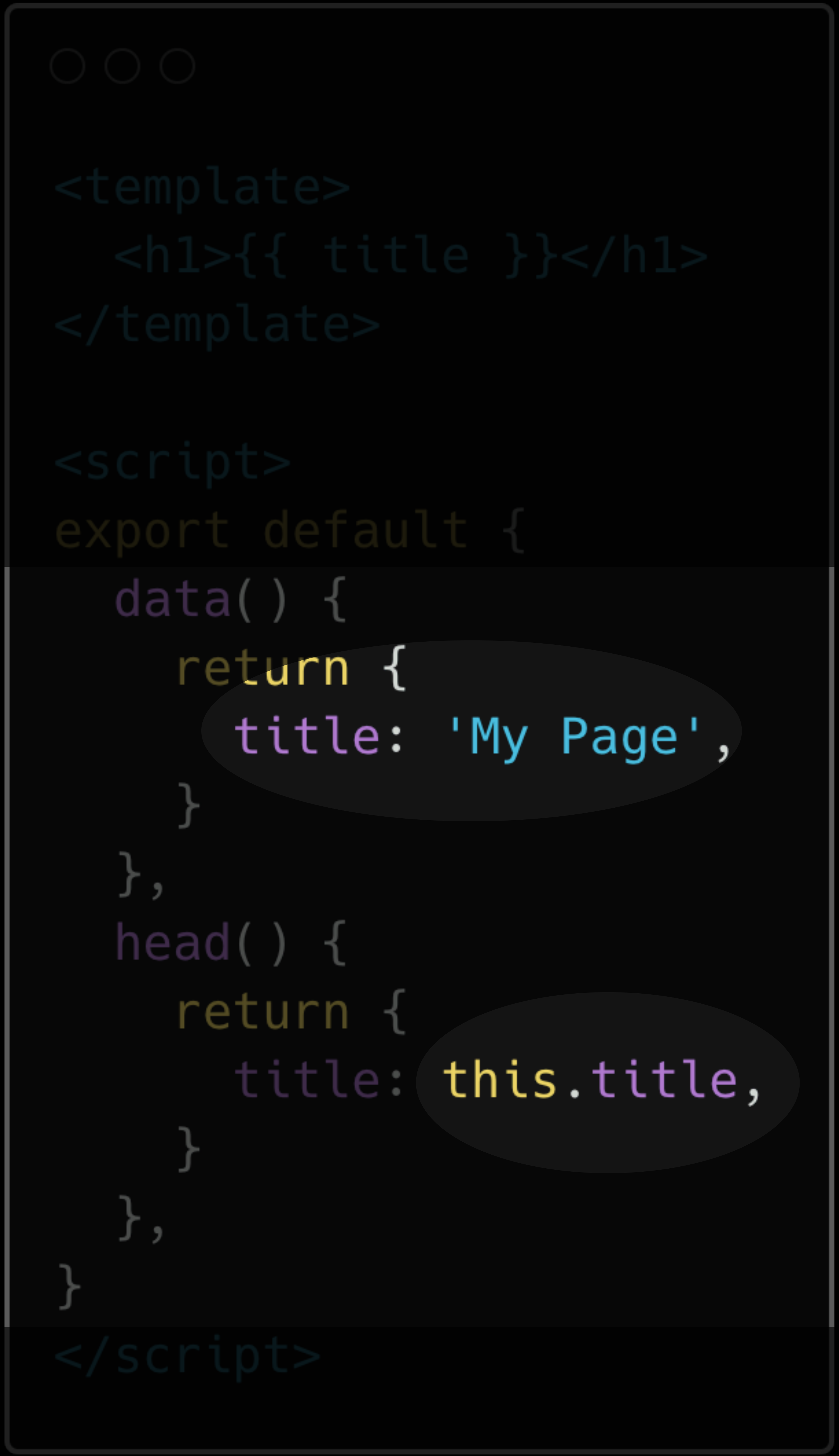
`<title>My Page</title>`



```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
  head() {
    return {
      title: this.title,
    }
  },
}
</script>
```

Head Meta in SSR

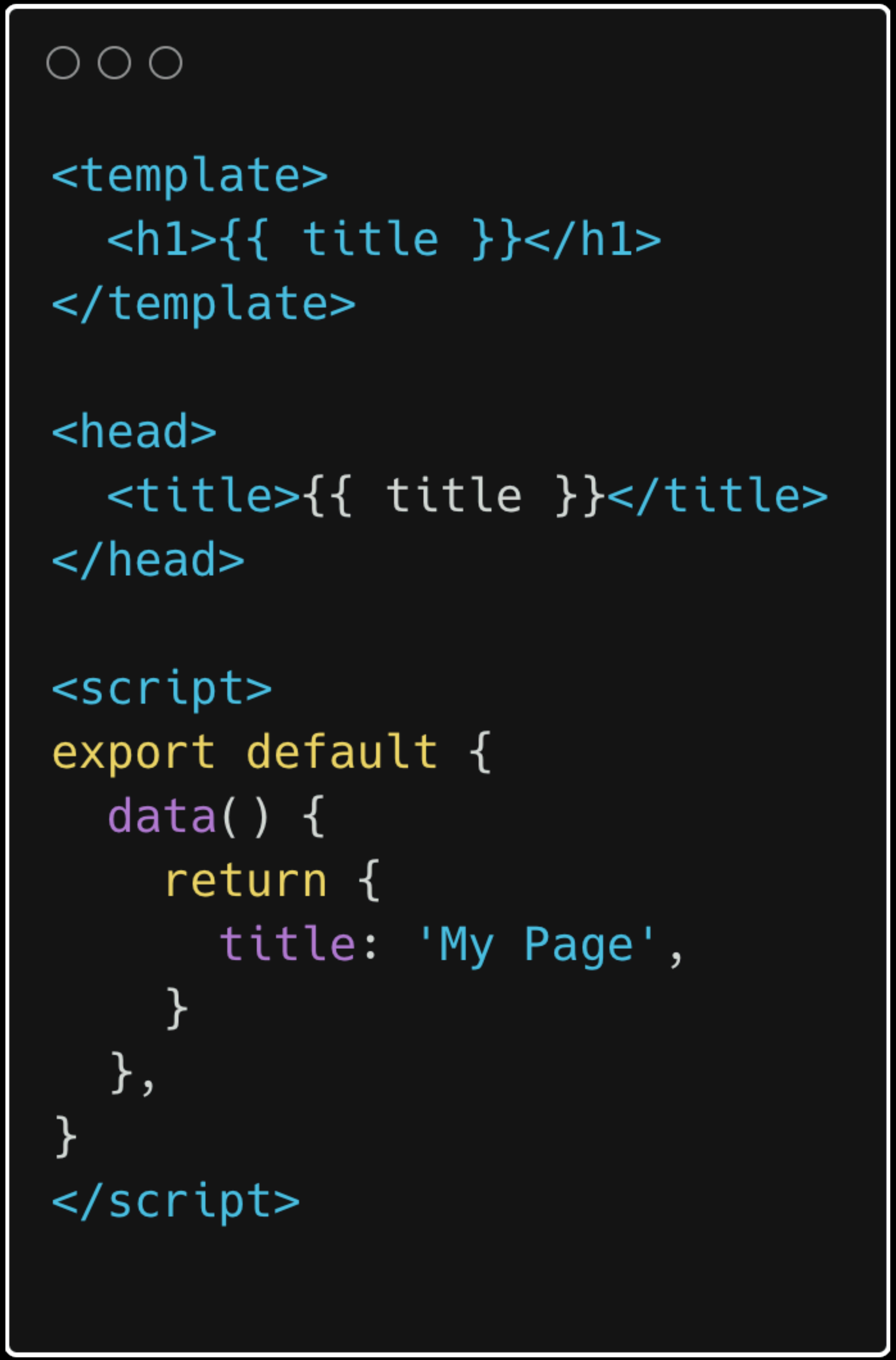
`<title>My Page</title>`



```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
  head() {
    return {
      title: this.title,
    }
  },
}
</script>
```

Head Meta in SSR

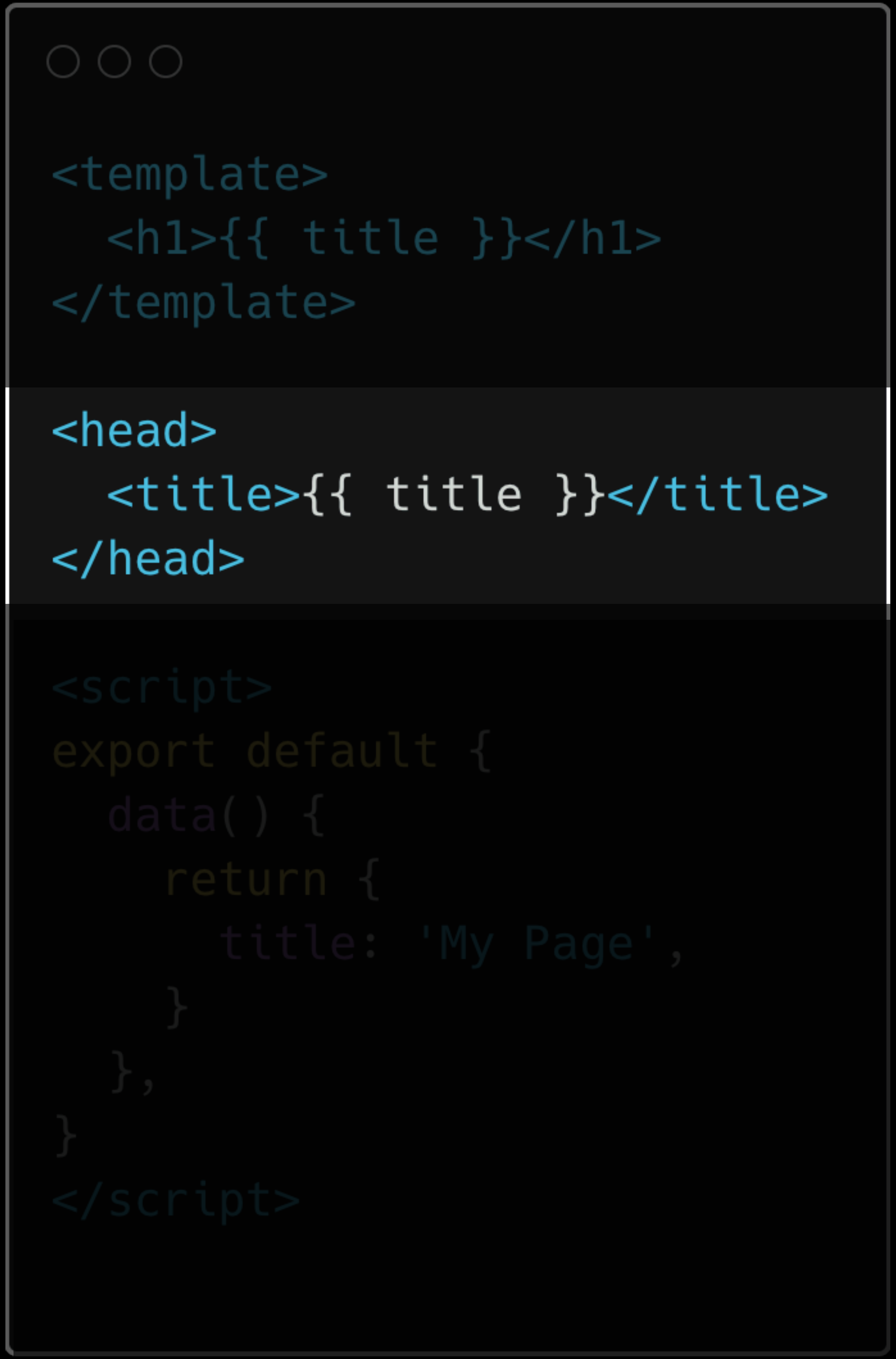
`<title>My Page</title>`



```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title>
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

Head Meta in SSR

`<title>My Page</title>`



```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title>
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

Head Meta in SSR

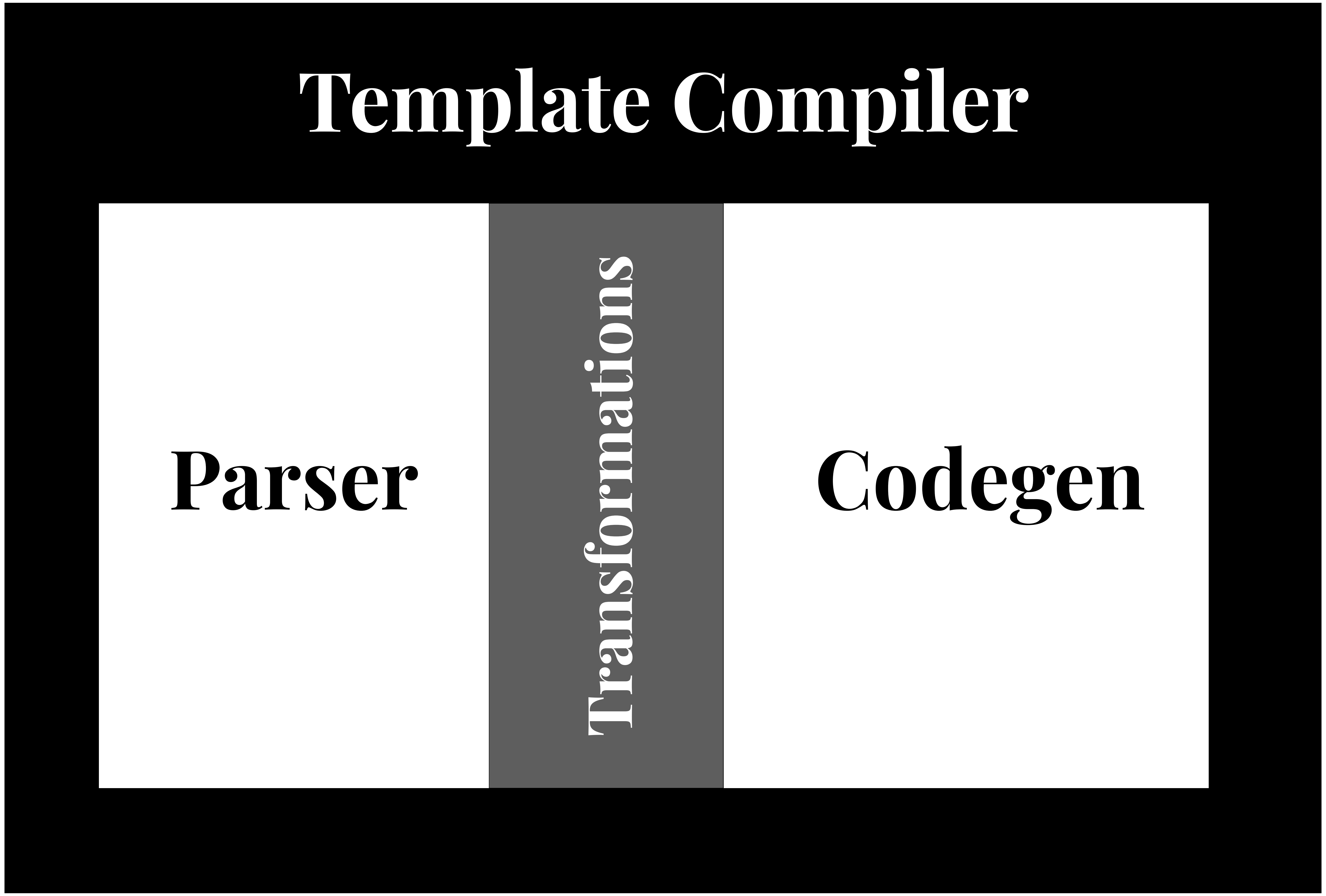
```
<title>My Page</title>
```

SFC Compiler

Head Meta in SSR

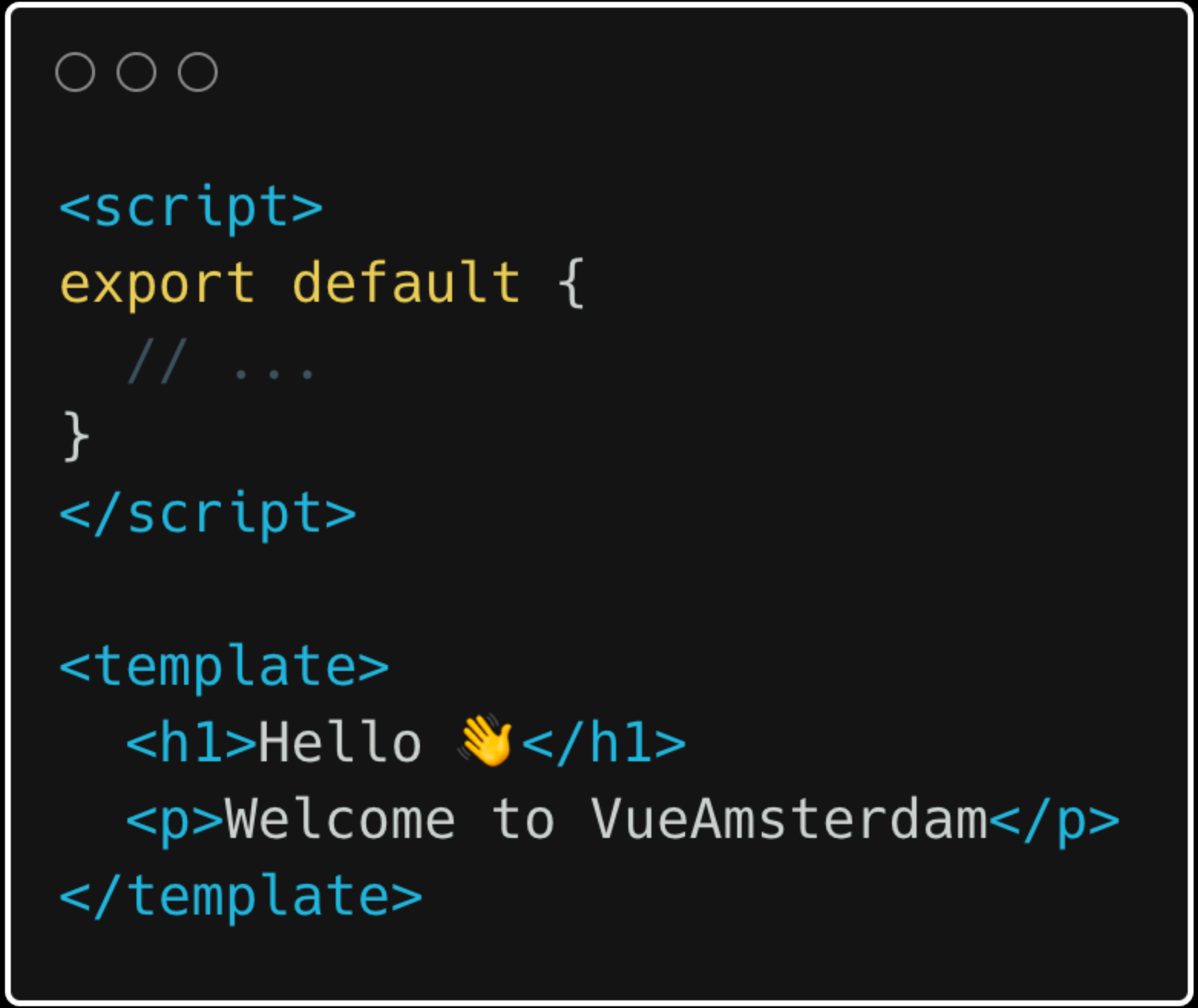
```
<title>My Page</title>
```

Template Compiler


Template Compiler
Parser
Transformations
Codegen

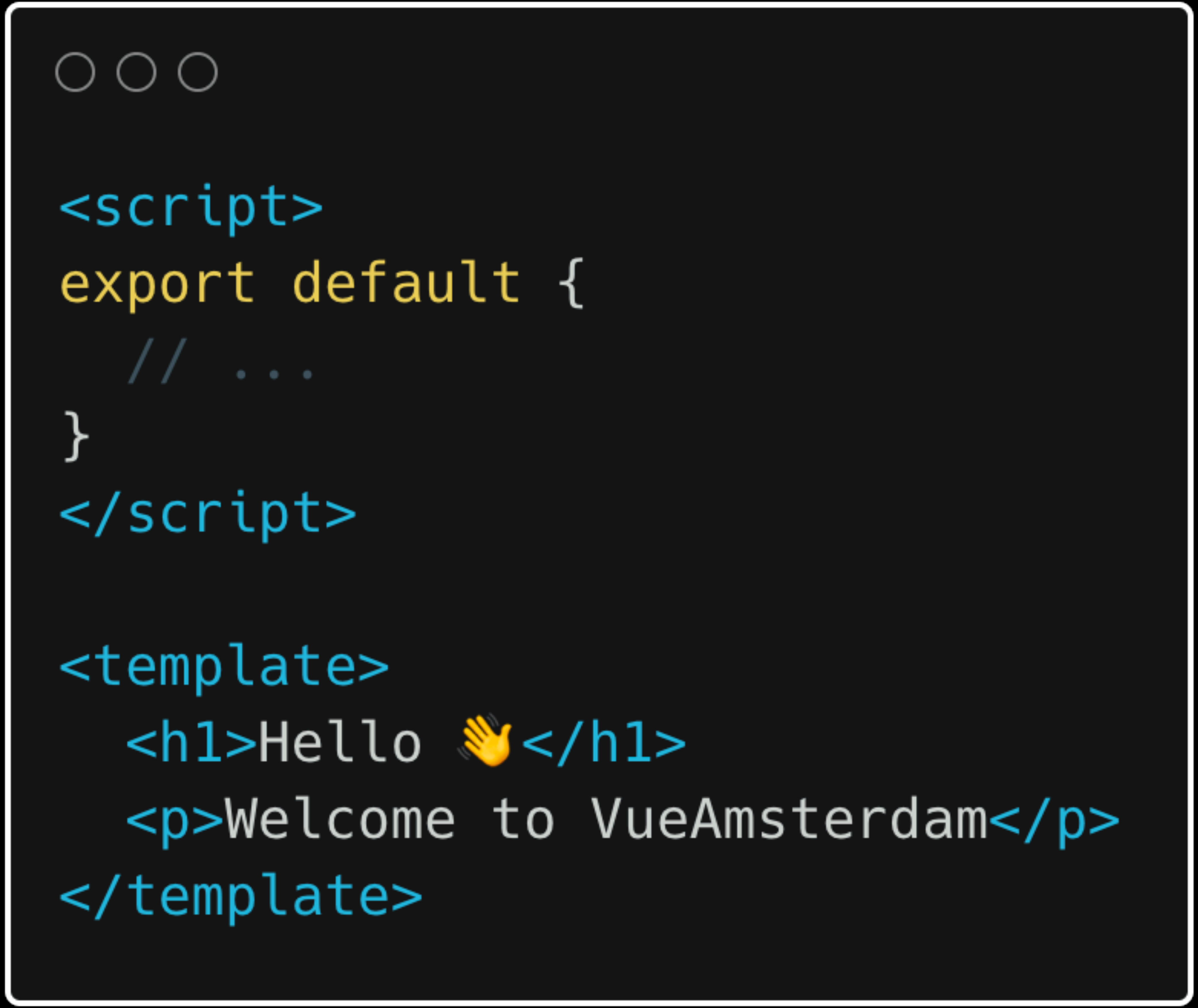
Parser

Parser



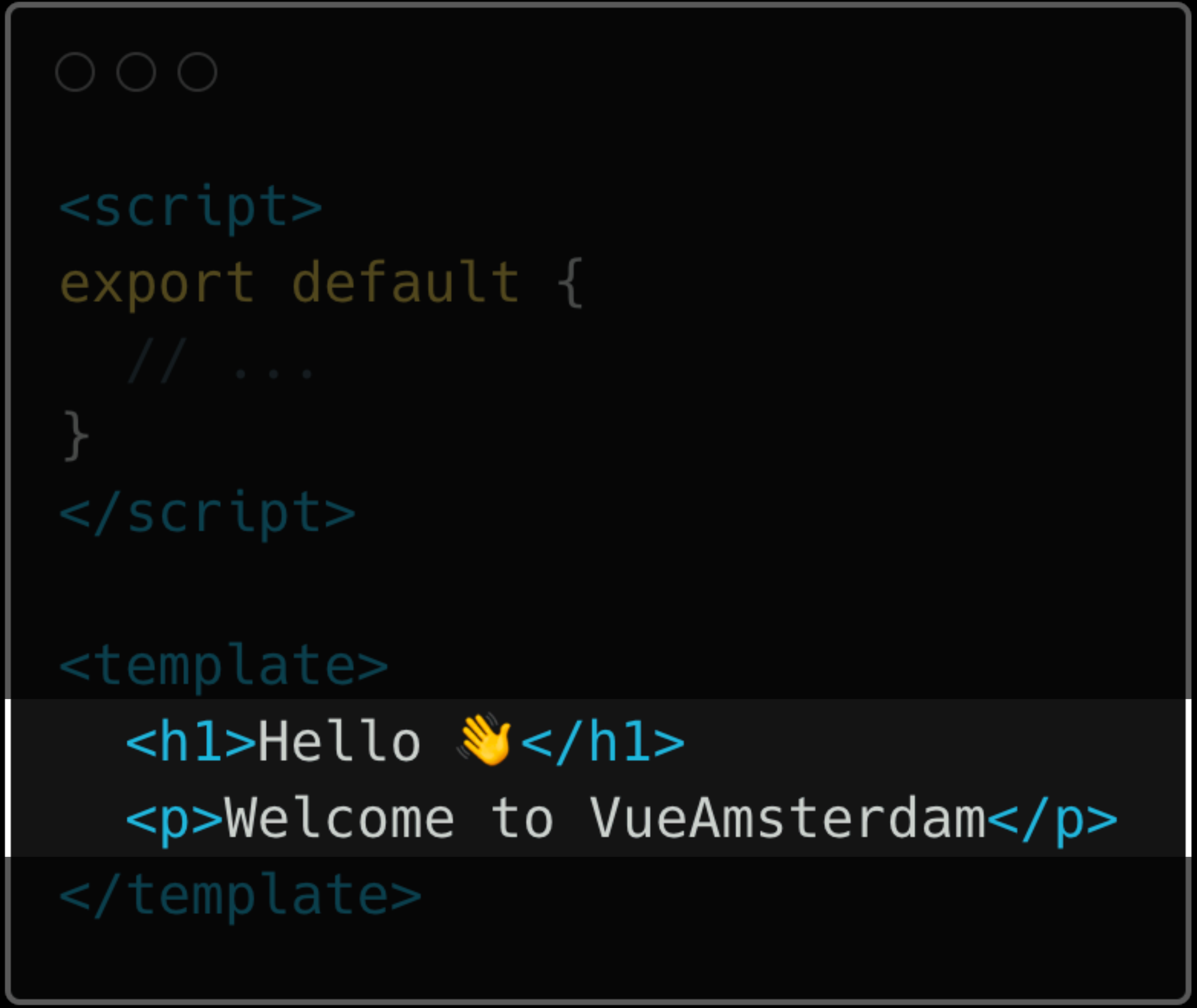
```
<script>
export default {
  // ...
}
</script>

<template>
  <h1>Hello 👋</h1>
  <p>Welcome to VueAmsterdam</p>
</template>
```

Parser



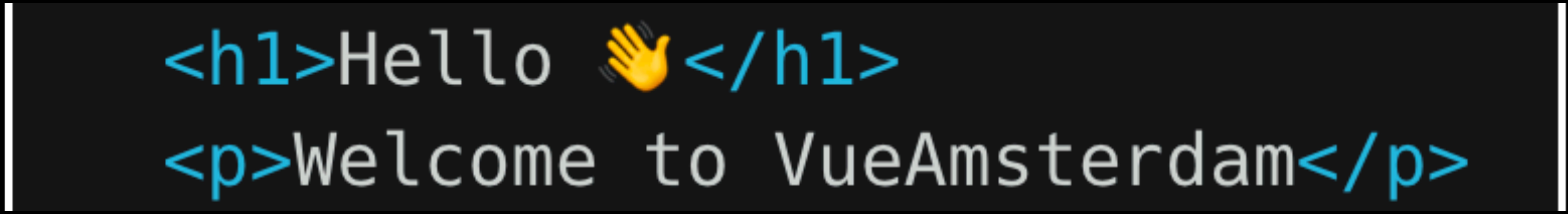
```
<script>
export default {
  // ...
}
</script>

<template>
  <h1>Hello 👋</h1>
  <p>Welcome to VueAmsterdam</p>
</template>
```

Parser



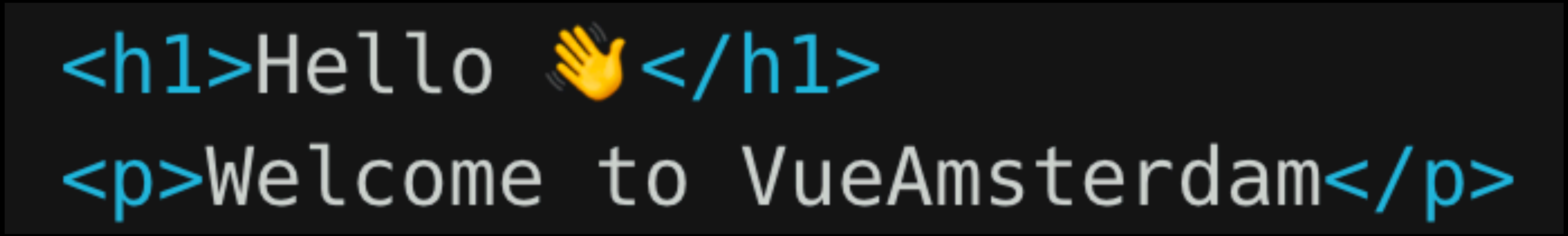
```
<script>
export default {
  // ...
}
</script>

<template>
  <h1>Hello 👋</h1>
  <p>Welcome to VueAmsterdam</p>
</template>
```

Parser



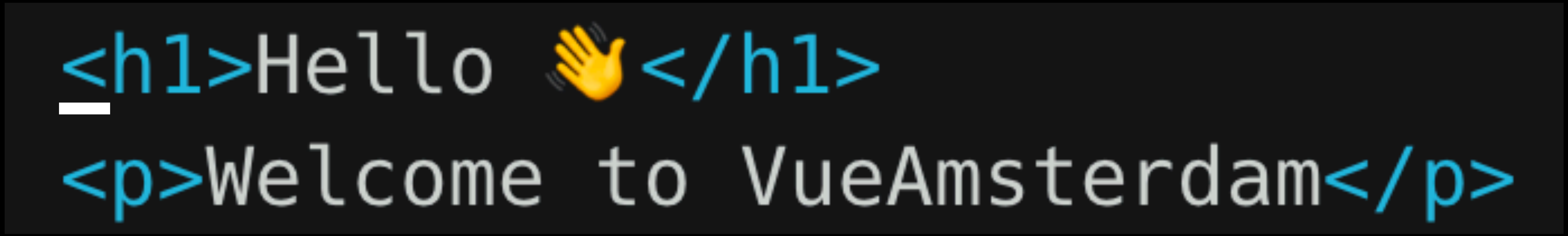
```html
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser



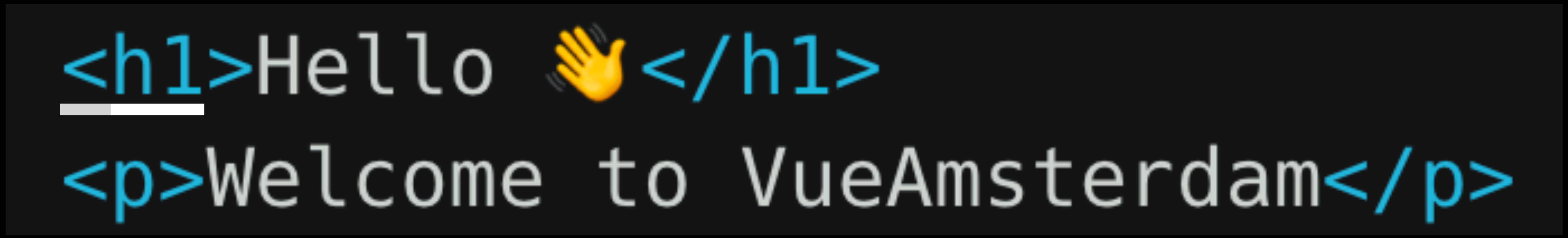
```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser



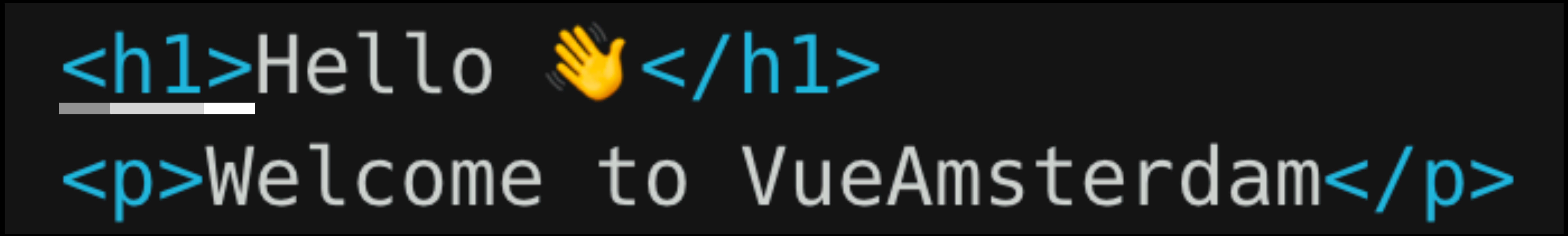
```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser



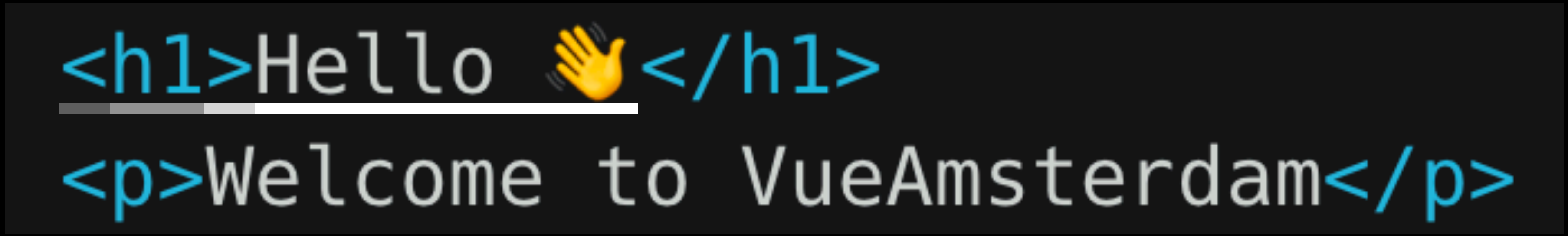
```html
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser



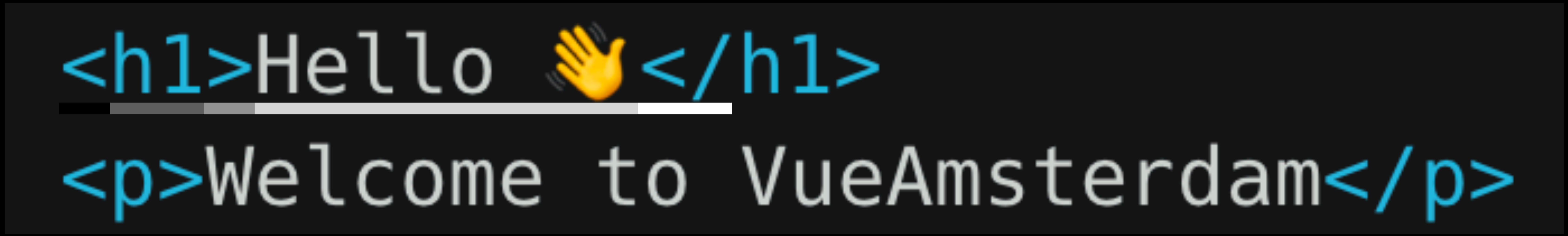
```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser



```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser



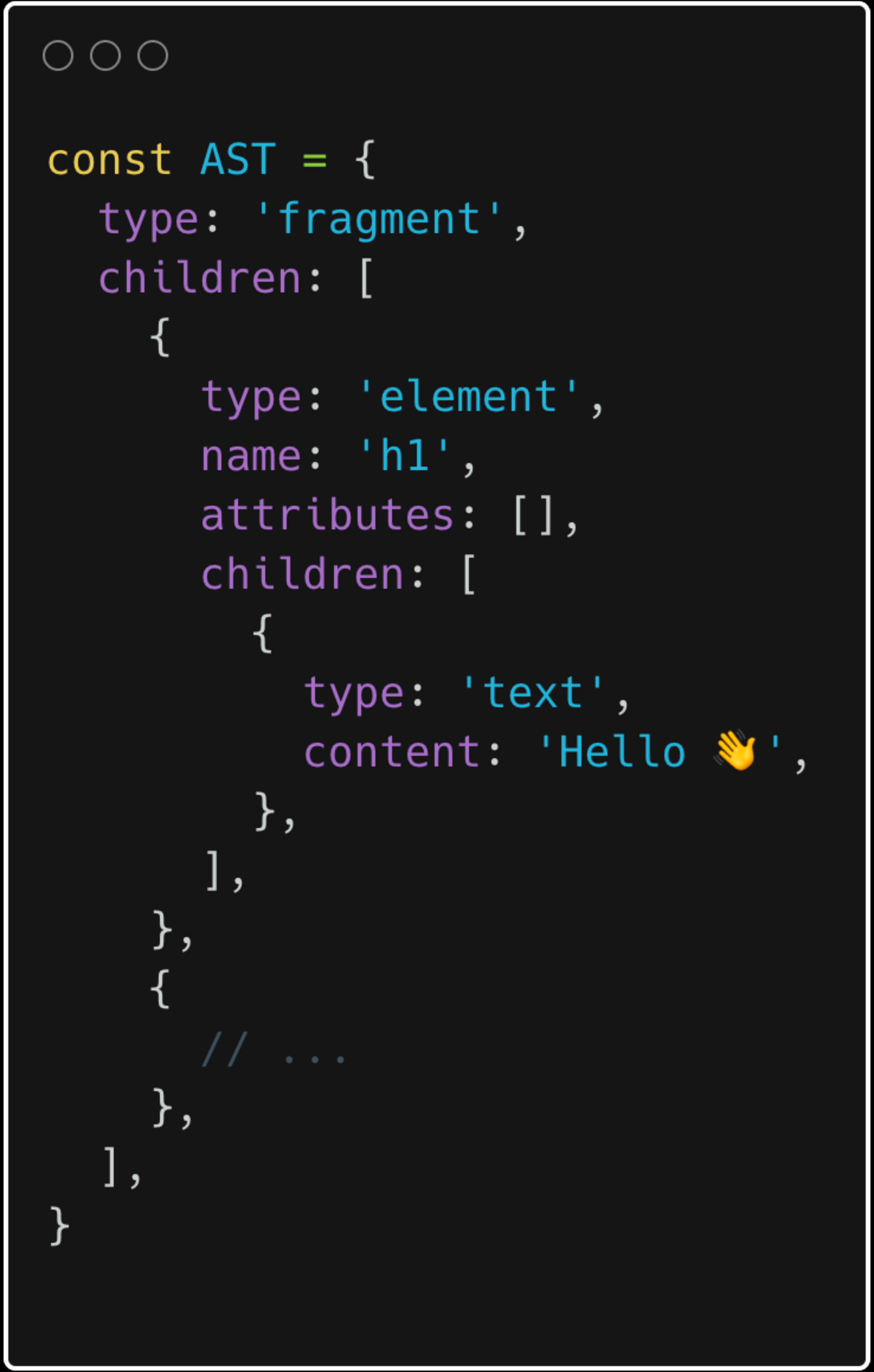
```html
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser

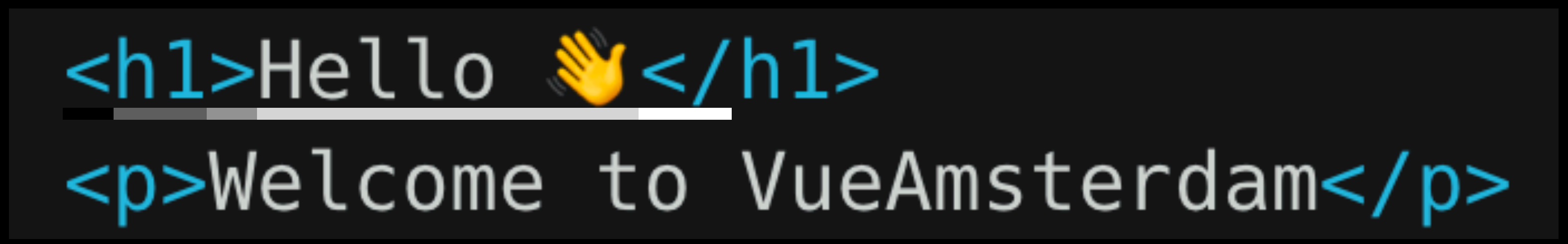
```
const AST = {
  type: 'fragment',
  children: [
    {
      type: 'element',
      name: 'h1',
      attributes: [],
      children: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
    {
      // ...
    },
  ],
}
```

```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser

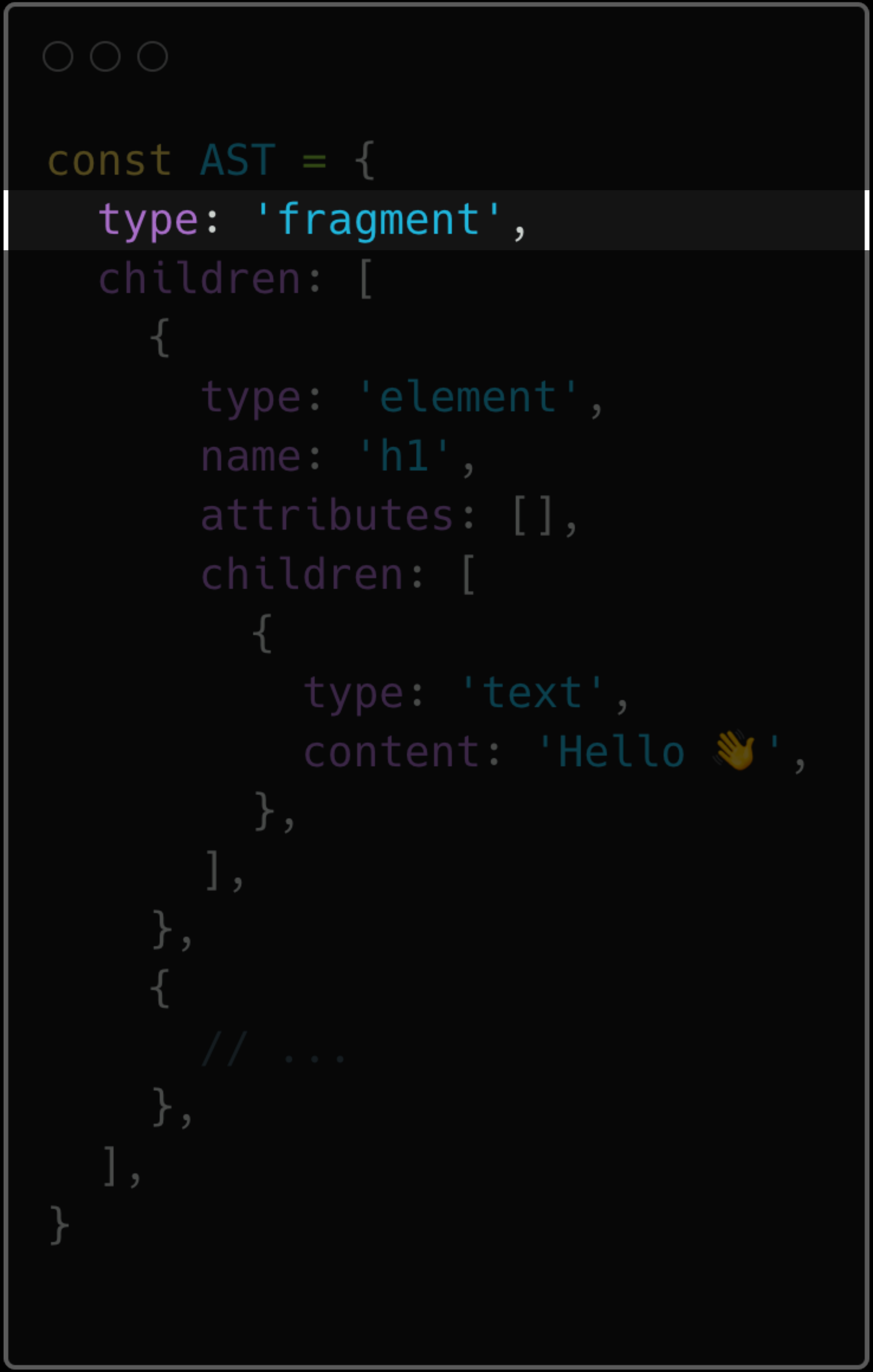


```
const AST = {
  type: 'fragment',
  children: [
    {
      type: 'element',
      name: 'h1',
      attributes: [],
      children: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
    {
      // ...
    },
  ],
}
```



```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser



```
const AST = {
  type: 'fragment',
  children: [
    {
      type: 'element',
      name: 'h1',
      attributes: [],
      children: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
    {
      // ...
    },
  ],
}
```

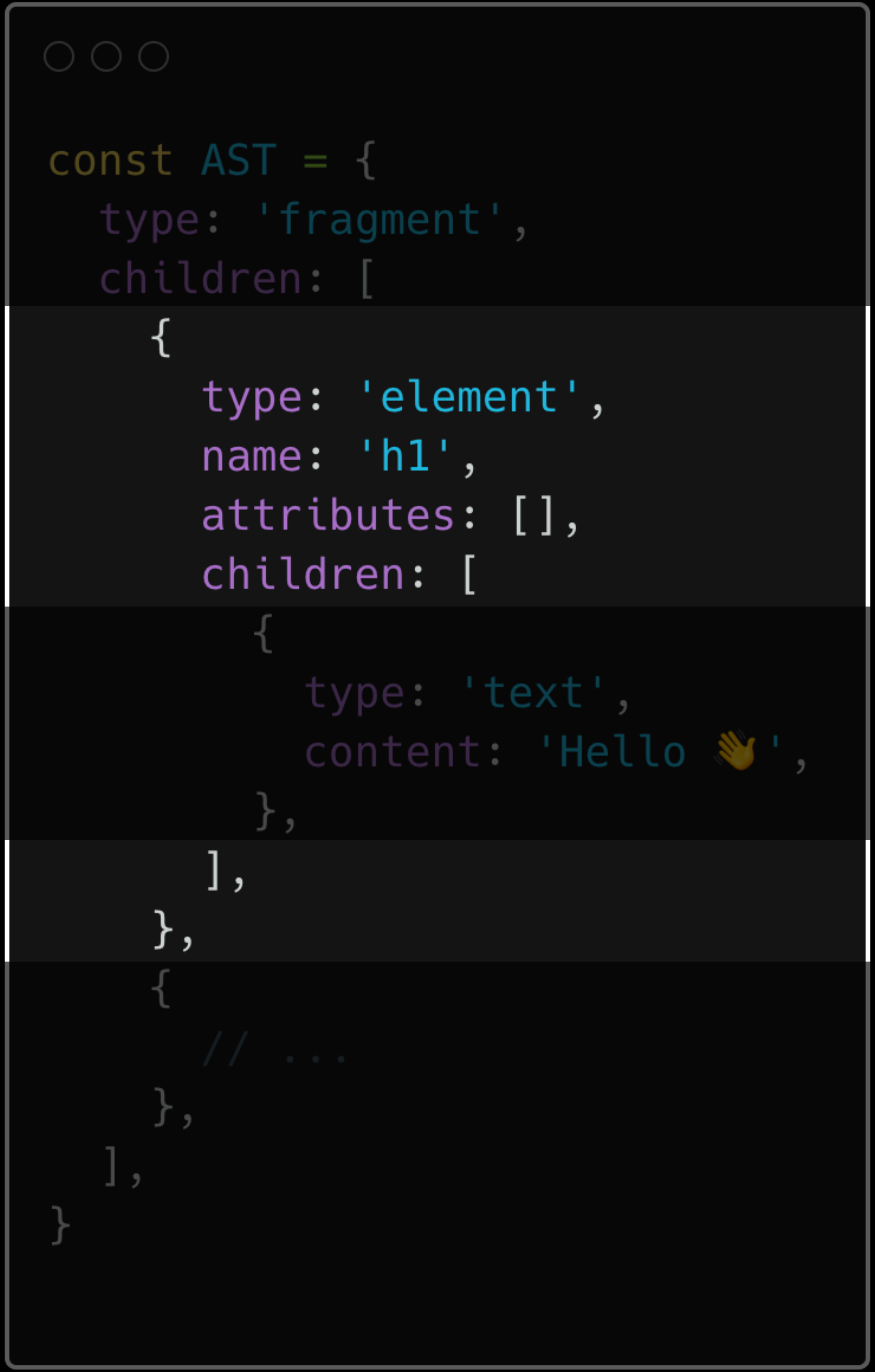
```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser

```
const AST = {
  type: 'fragment',
  children: [
    {
      type: 'element',
      name: 'h1',
      attributes: [],
      children: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
    {
      // ...
    },
  ],
}
```

```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

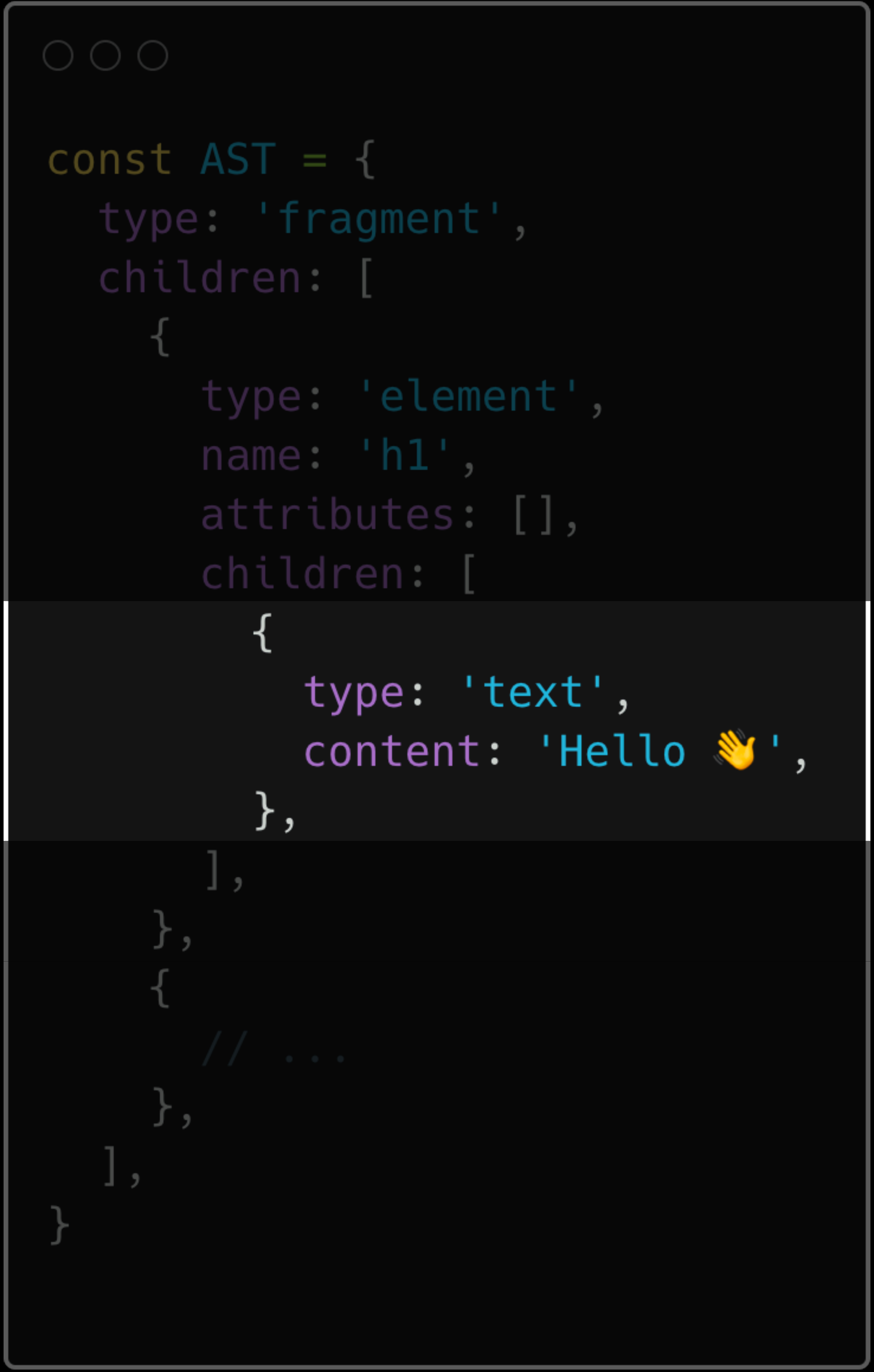
Parser



```
const AST = {
  type: 'fragment',
  children: [
    {
      type: 'element',
      name: 'h1',
      attributes: [],
      children: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
    {
      // ...
    },
  ],
}
```

```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

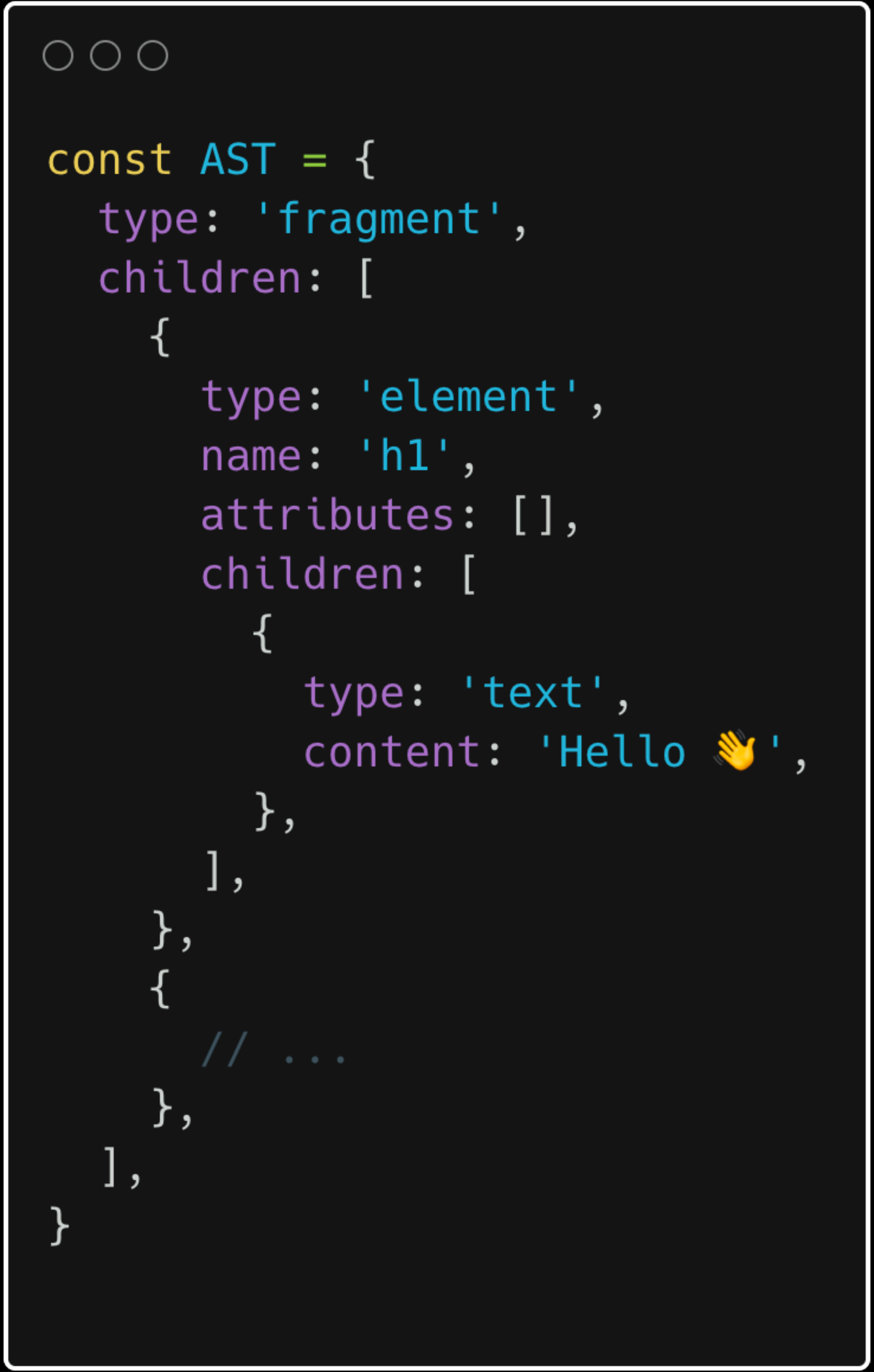
Parser



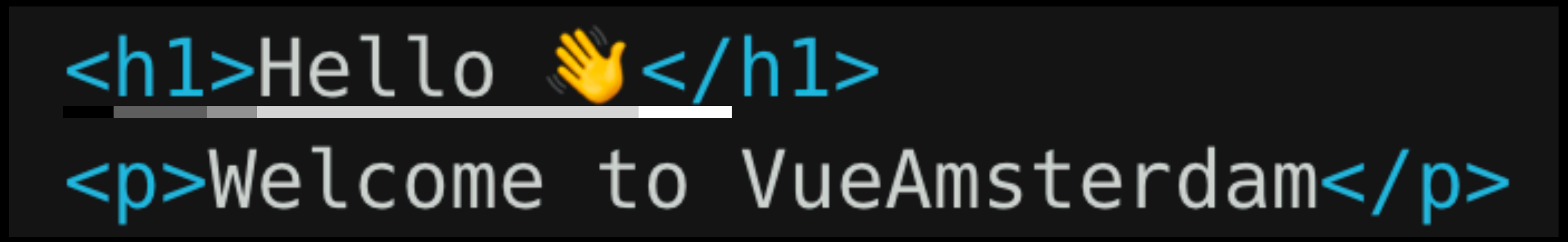
```
const AST = {
  type: 'fragment',
  children: [
    {
      type: 'element',
      name: 'h1',
      attributes: [],
      children: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
    {
      // ...
    },
  ],
}
```

```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser

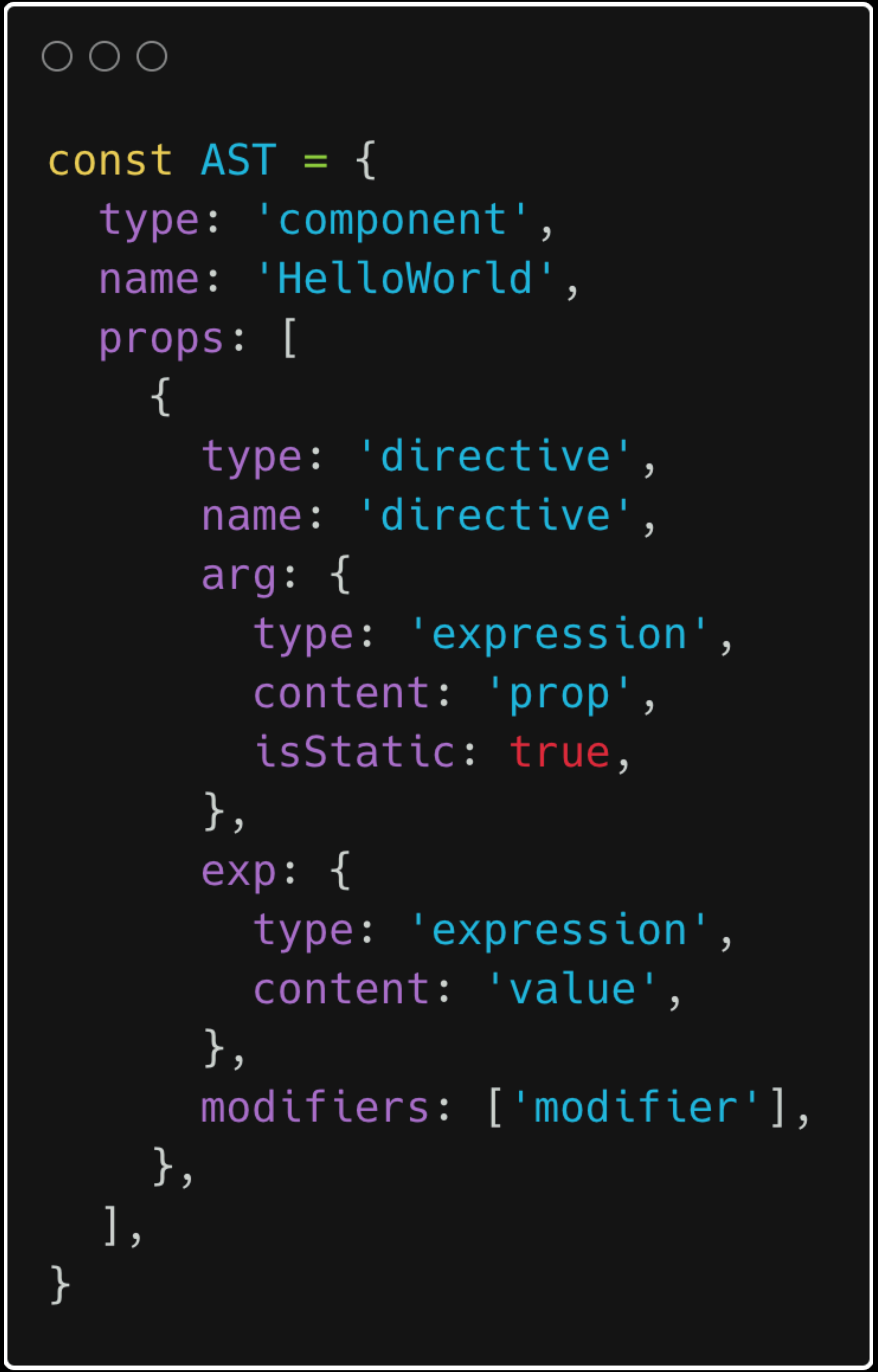


```
const AST = {
  type: 'fragment',
  children: [
    {
      type: 'element',
      name: 'h1',
      attributes: [],
      children: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
    {
      // ...
    },
  ],
}
```

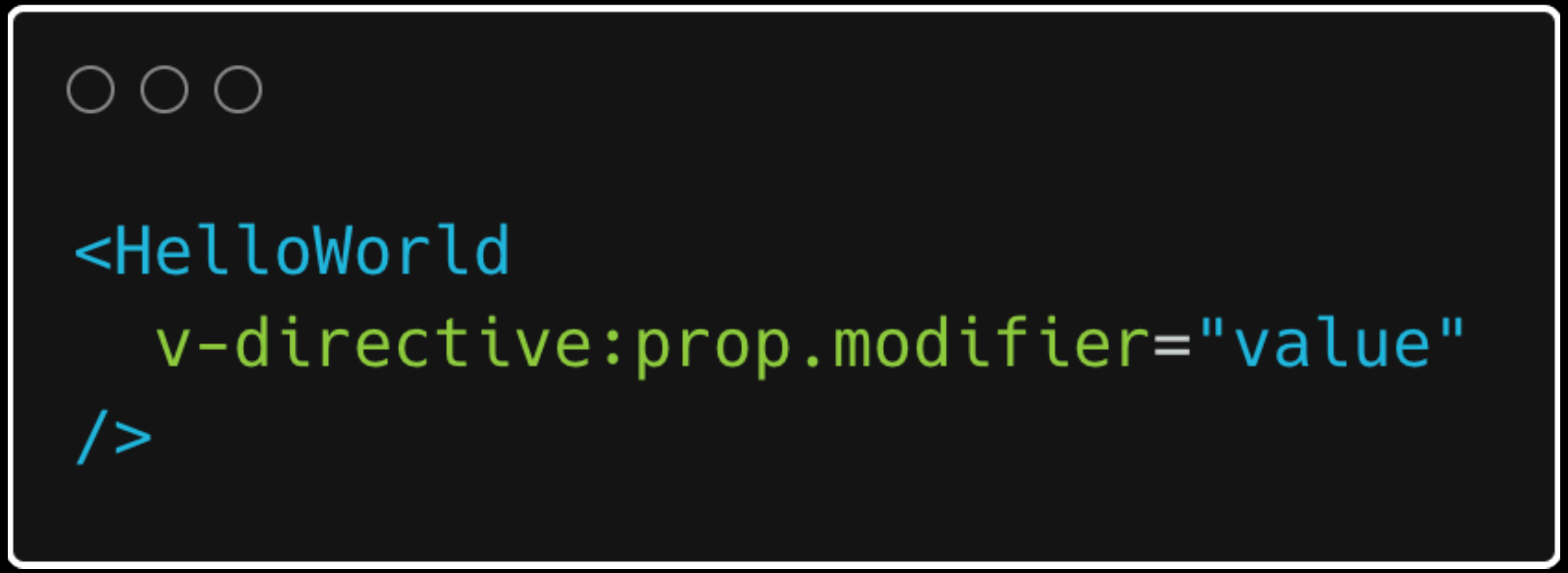


```
<h1>Hello 👋</h1>
<p>Welcome to VueAmsterdam</p>
```

Parser

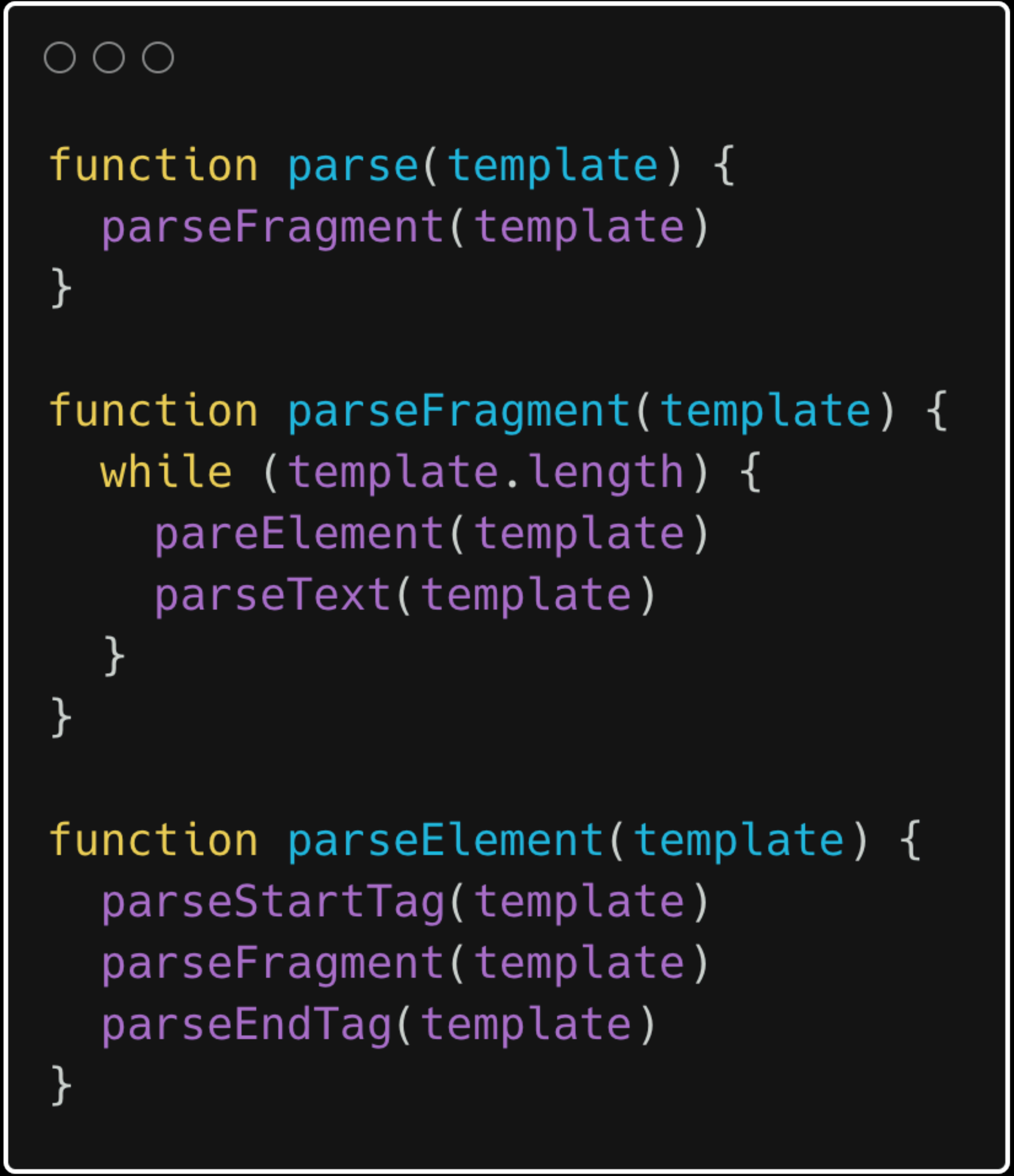


```
const AST = {
  type: 'component',
  name: 'HelloWorld',
  props: [
    {
      type: 'directive',
      name: 'directive',
      arg: {
        type: 'expression',
        content: 'prop',
        isStatic: true,
      },
      exp: {
        type: 'expression',
        content: 'value',
      },
      modifiers: ['modifier'],
    },
  ],
}
```



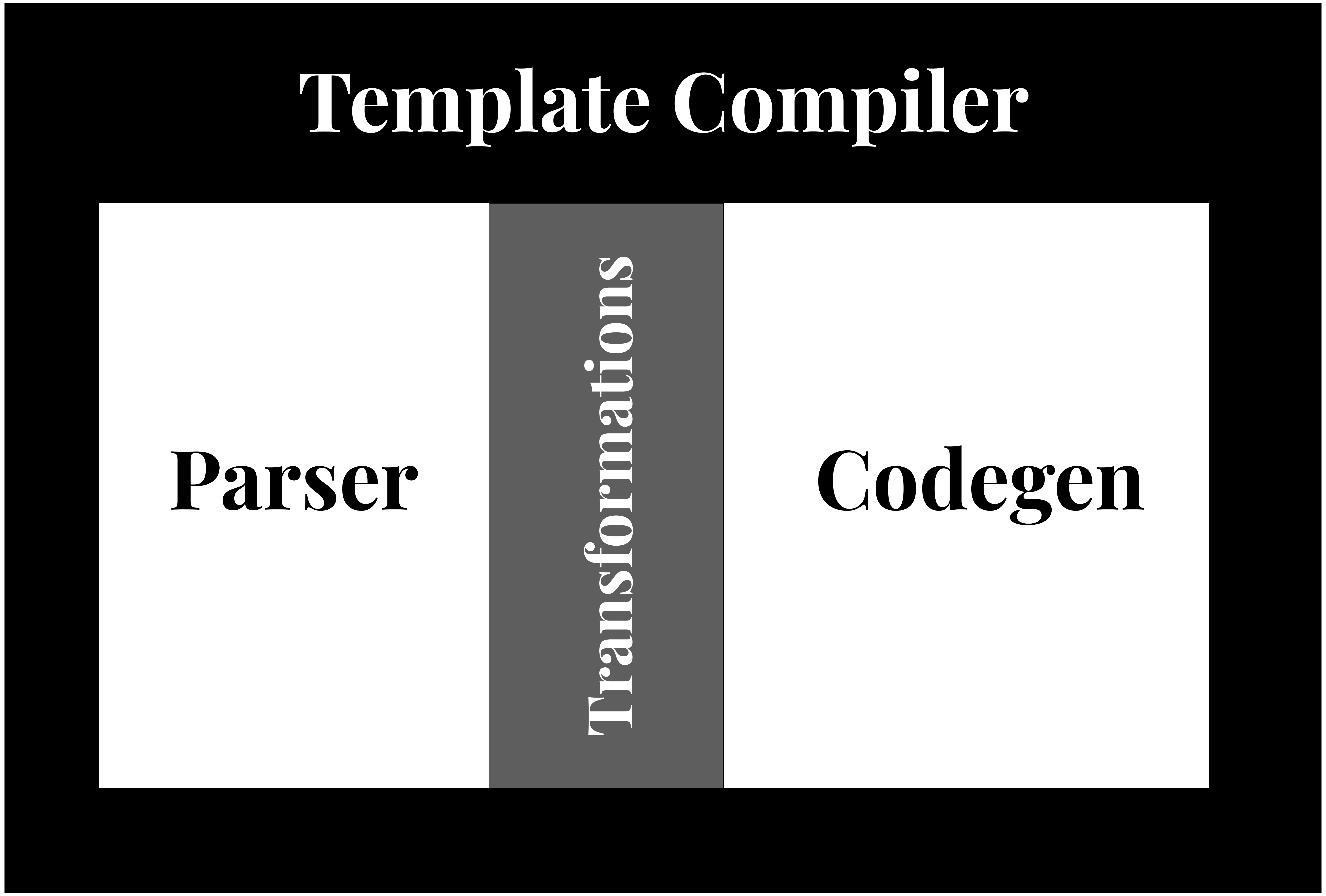
```
<HelloWorld
  v-directive:prop.modifier="value"
/>
```

Parser



```
function parse(template) {
  parseFragment(template)
}

function parseFragment(template) {
  while (template.length) {
    pareElement(template)
    parseText(template)
  }
}

function parseElement(template) {
  parseStartTag(template)
  parseFragment(template)
  parseEndTag(template)
}
```

Parser

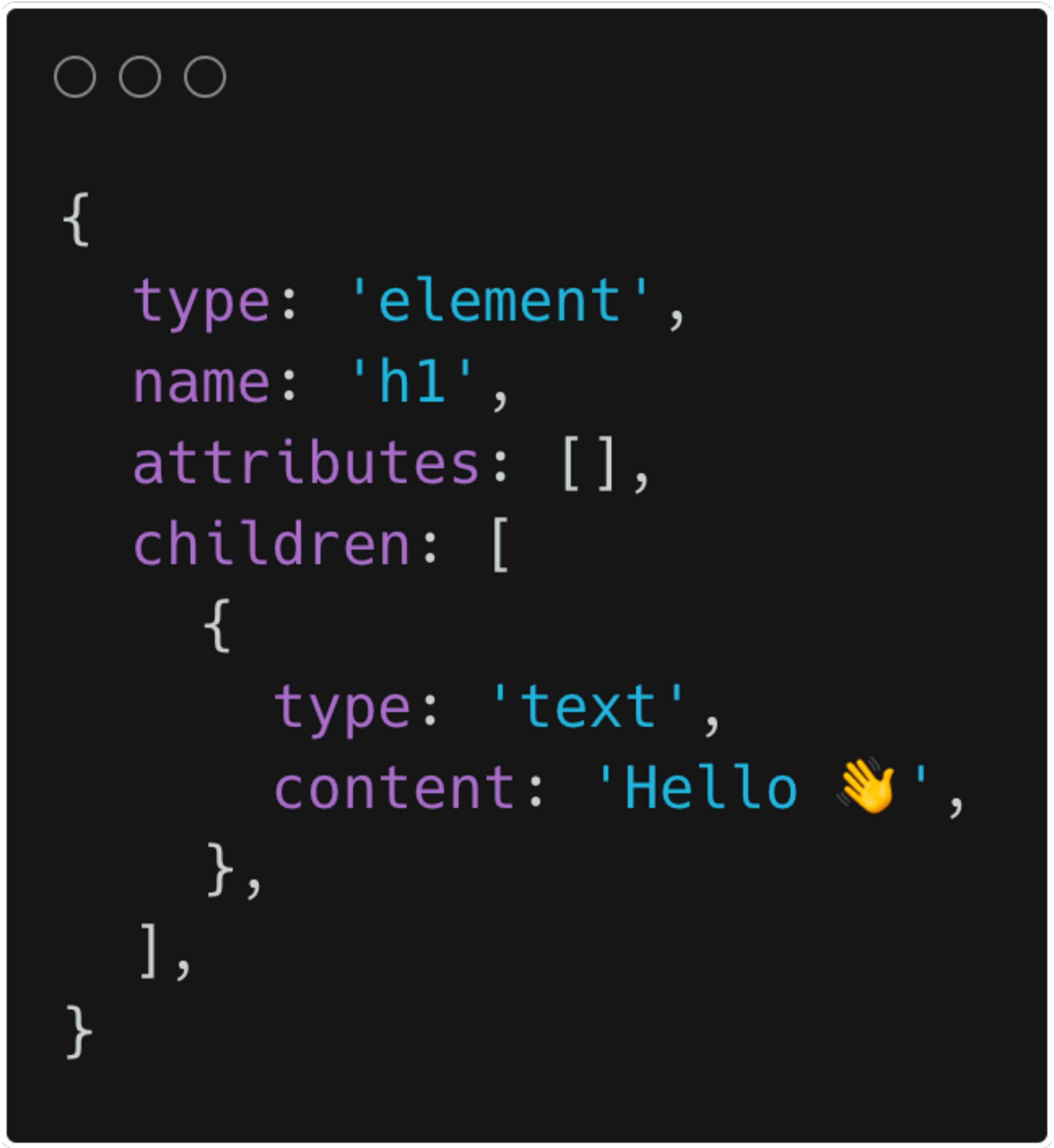

Template Compiler
Parser
Transformations
Codegen

Transformations

Transformations

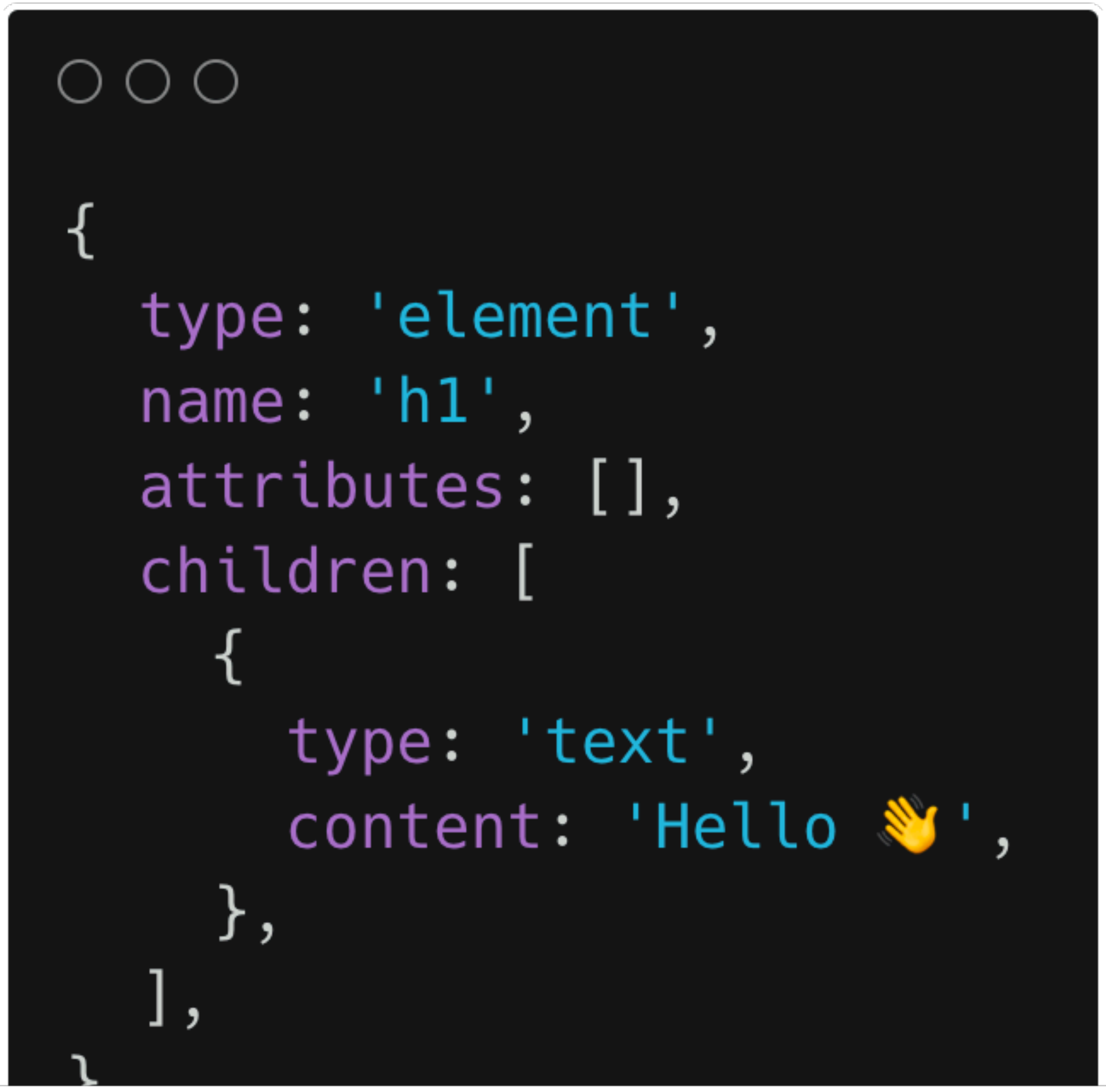
```
codegenAST  = f(parseAST)
```

Transformations

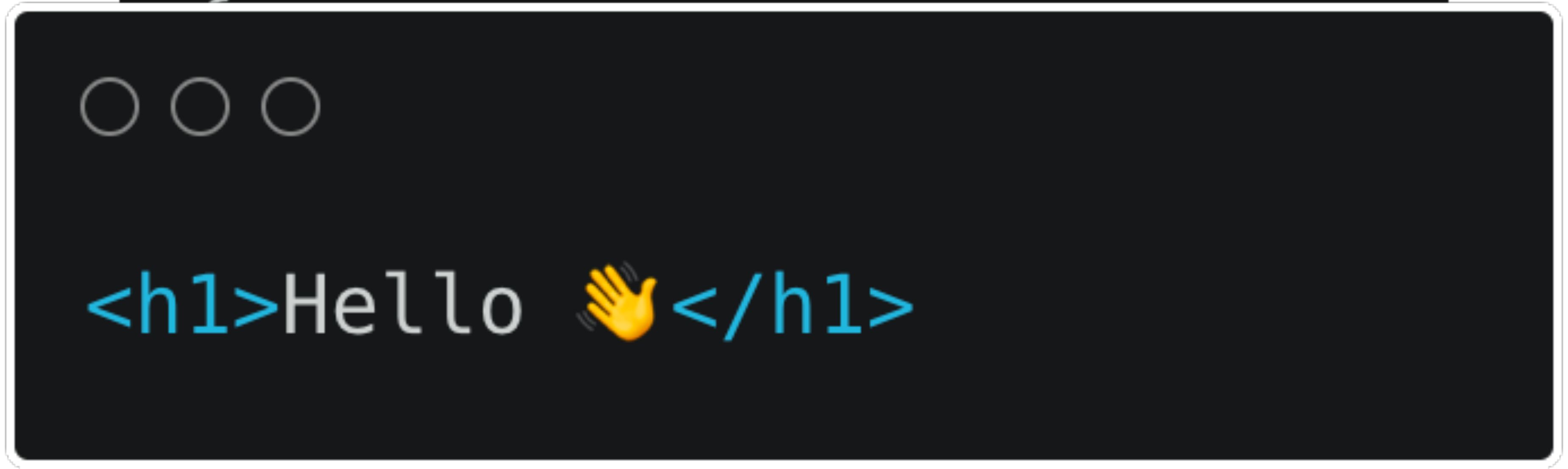


```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
}
```

Transformations

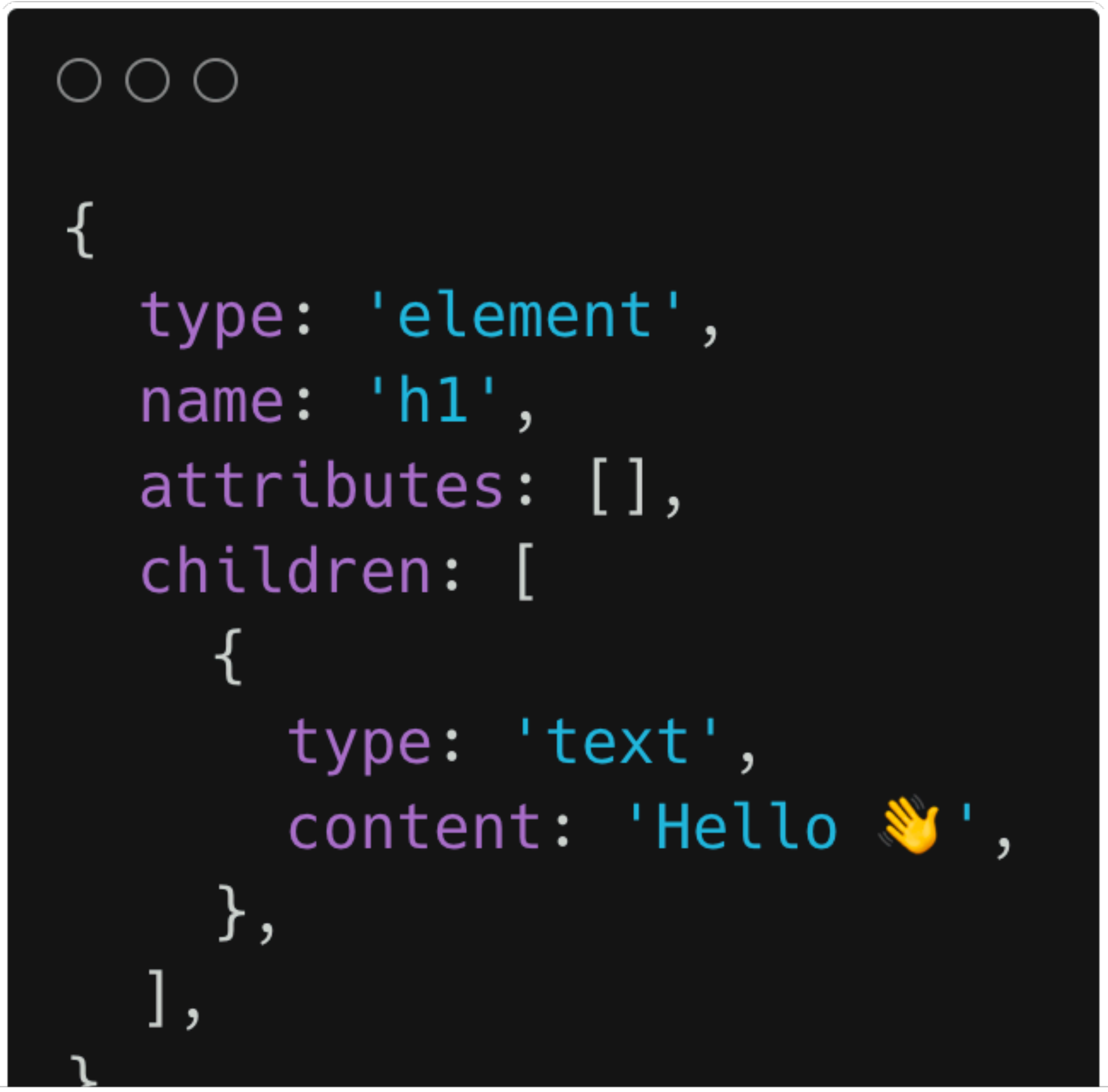


```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
}
```

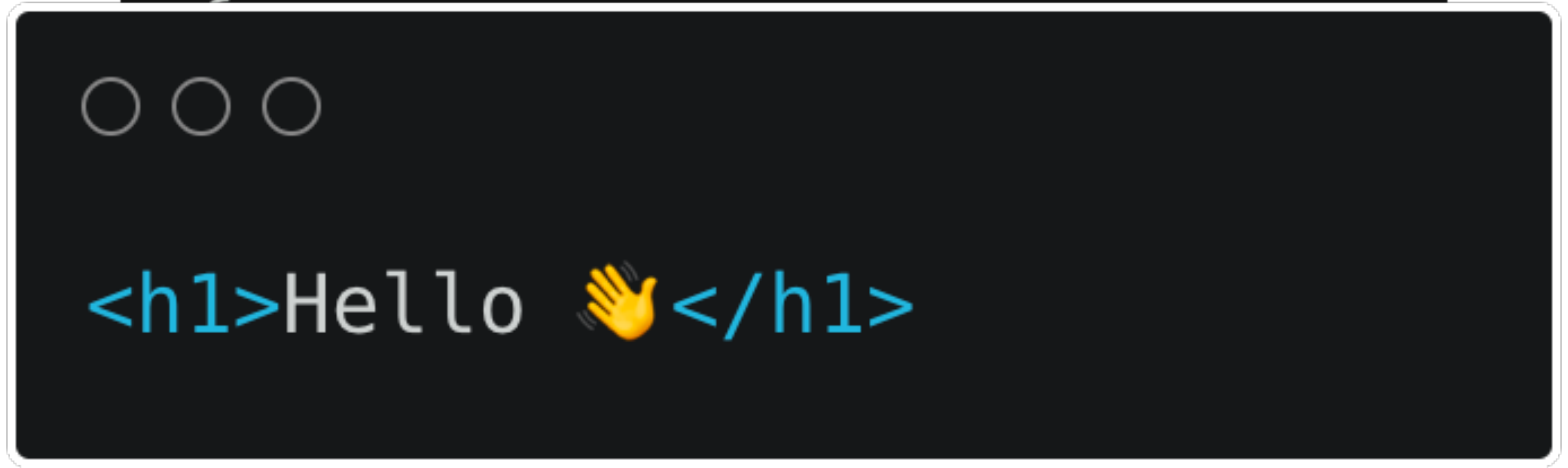


```
<h1>Hello 👋</h1>
```

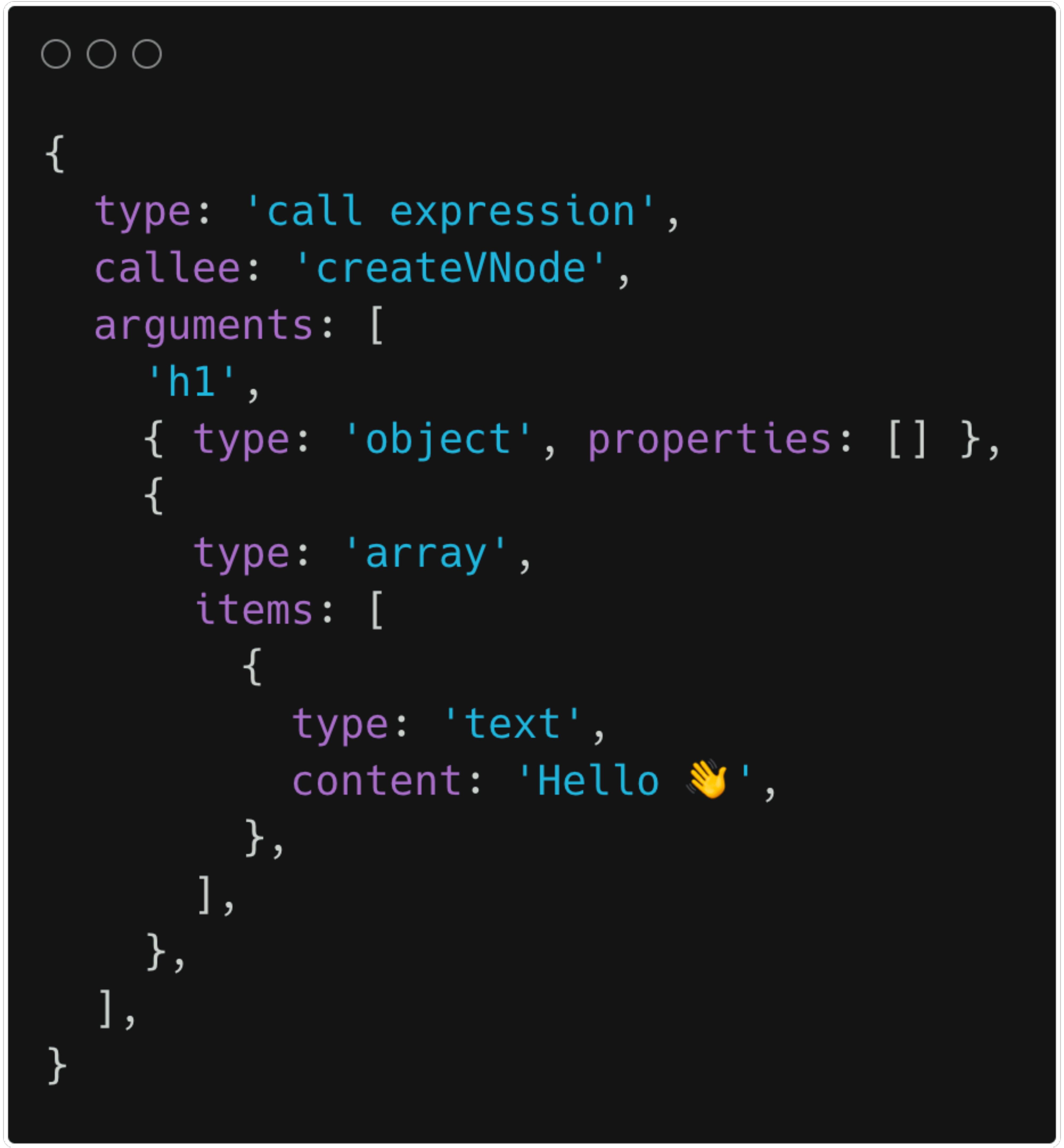
Transformations



```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
}
```

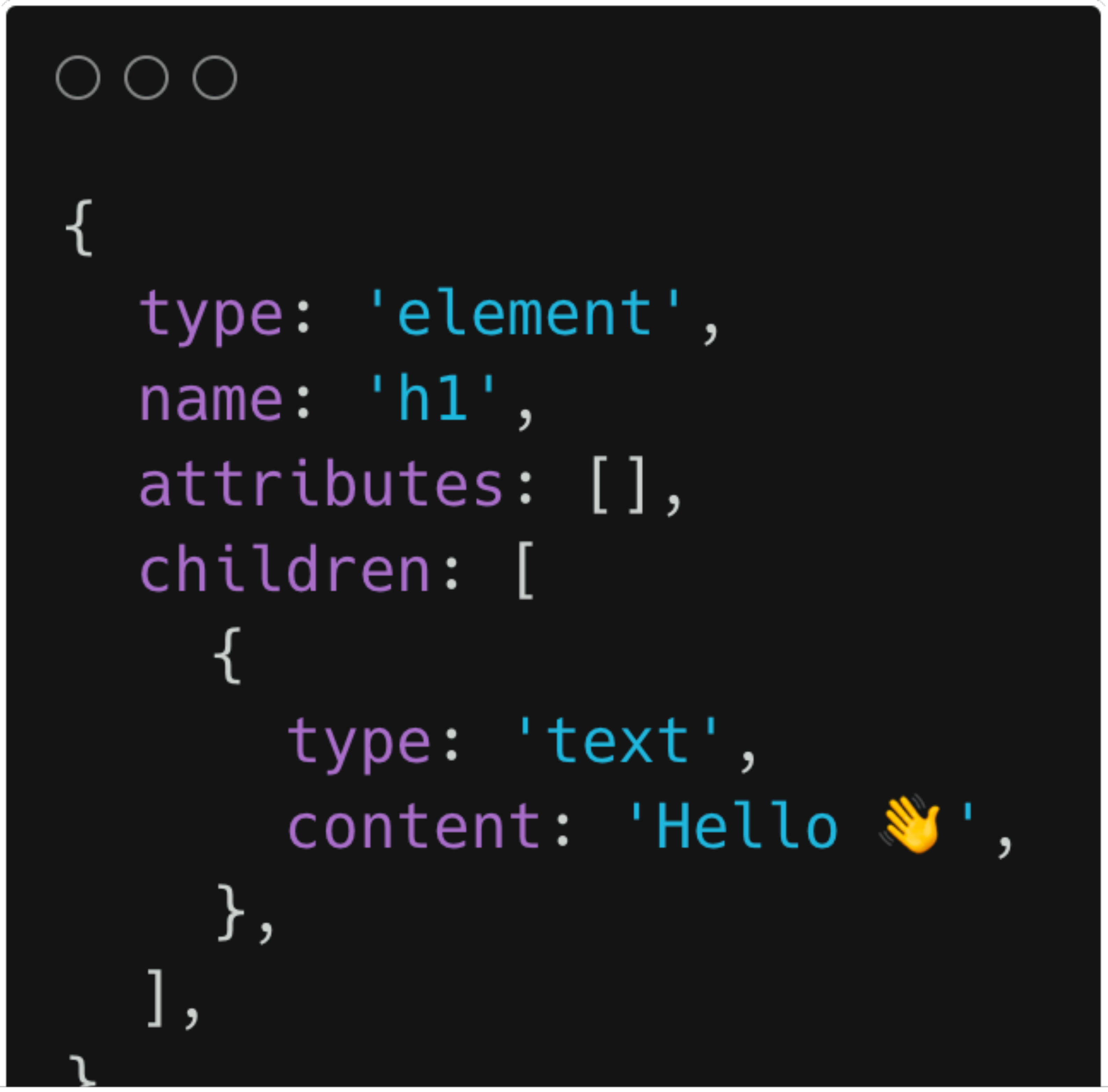


```
<h1>Hello 👋</h1>
```

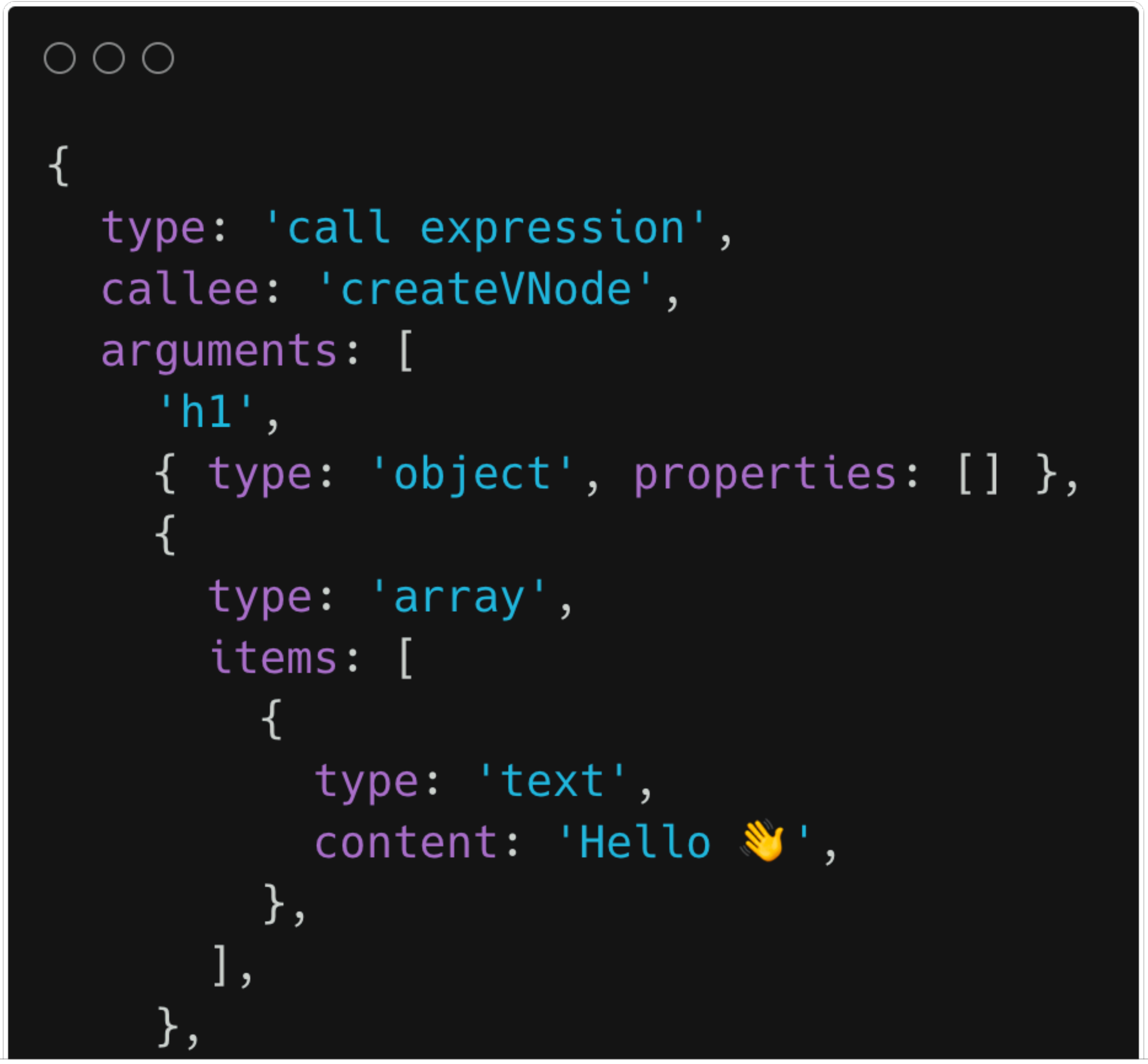


```
{
  type: 'call expression',
  callee: 'createVNode',
  arguments: [
    'h1',
    { type: 'object', properties: [] },
    {
      type: 'array',
      items: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
  ],
}
```

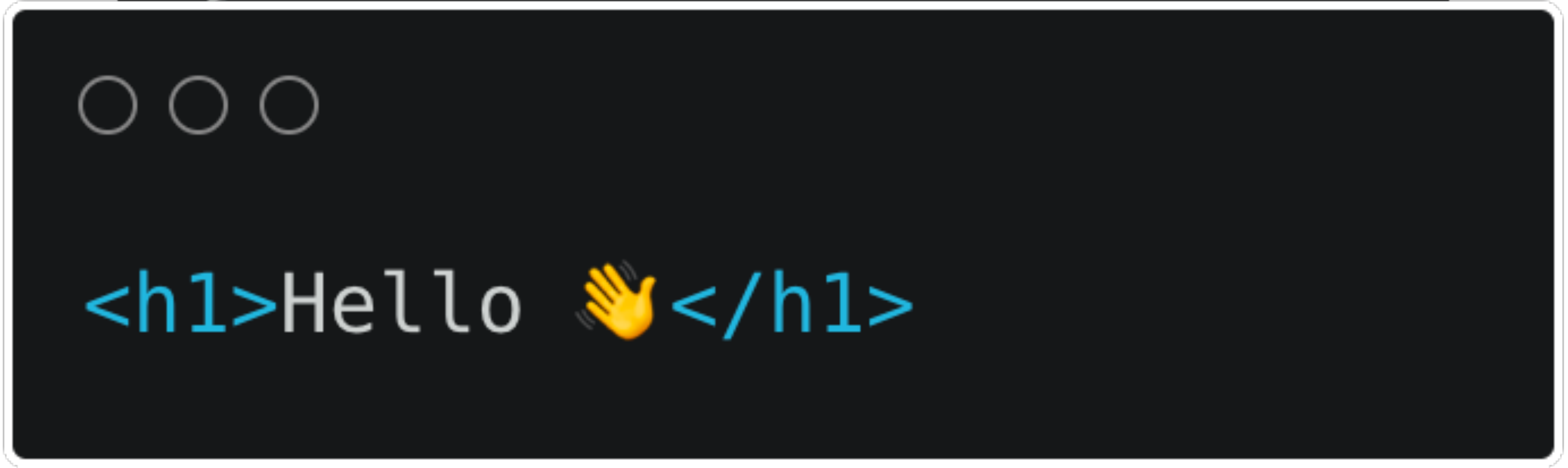
Transformations



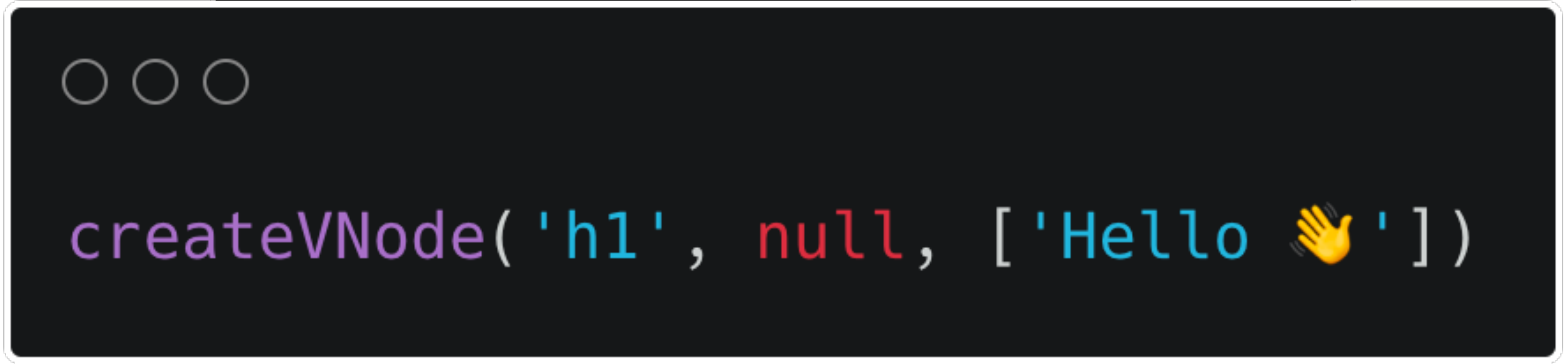
```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
}
```



```
{
  type: 'call expression',
  callee: 'createVNode',
  arguments: [
    'h1',
    { type: 'object', properties: [] },
    {
      type: 'array',
      items: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
```

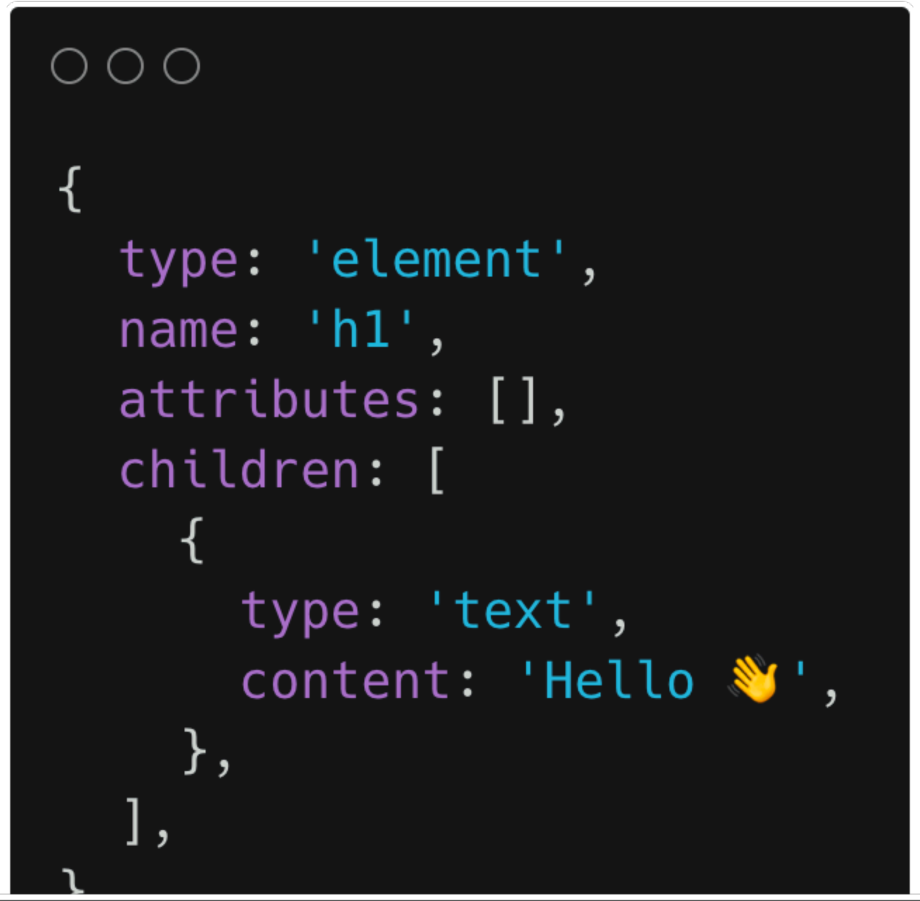


```
<h1>Hello 👋</h1>
```

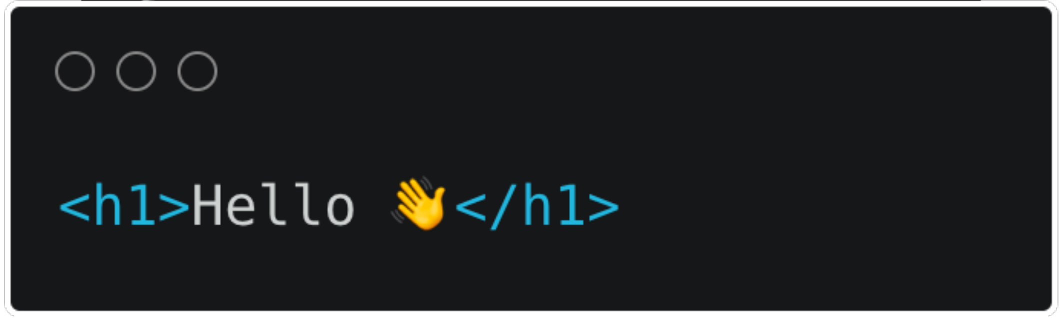


```
createVNode('h1', null, ['Hello 👋'])
```

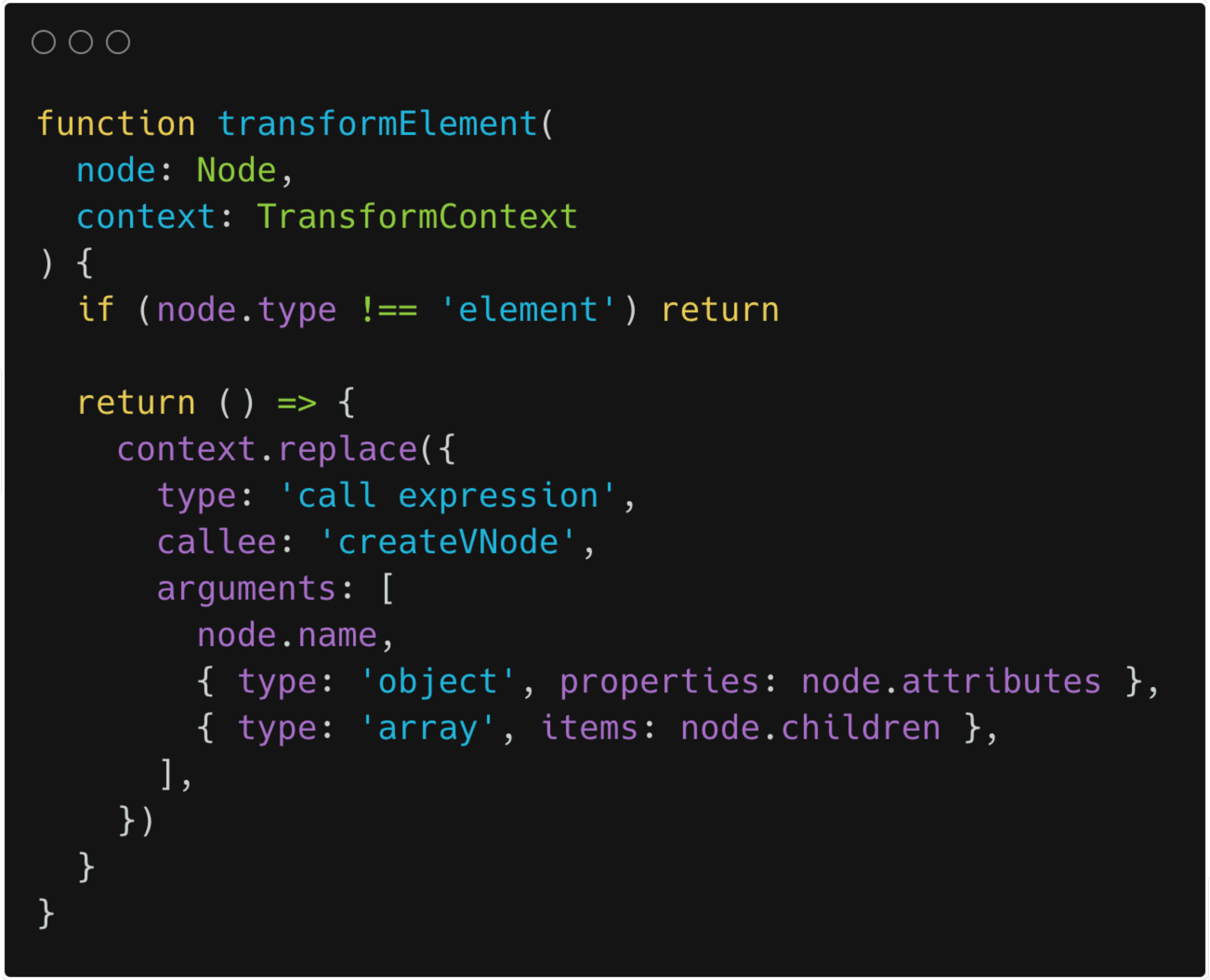
Transformations



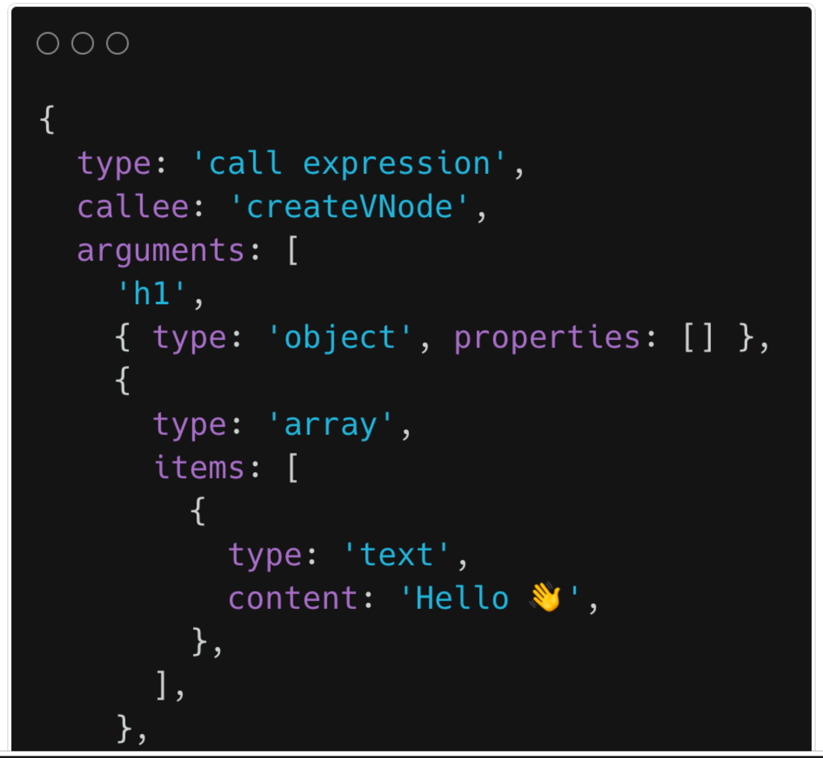
```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
```



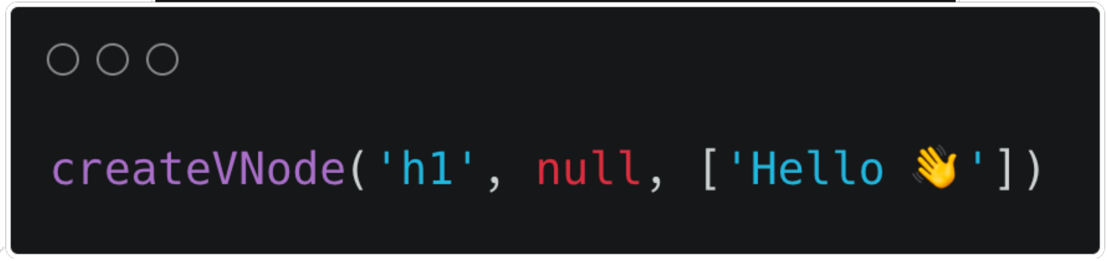
```
<h1>Hello 👋</h1>
```



```
function transformElement(
  node: Node,
  context: TransformContext
) {
  if (node.type !== 'element') return

  return () => {
    context.replace({
      type: 'call expression',
      callee: 'createVNode',
      arguments: [
        node.name,
        { type: 'object', properties: node.attributes },
        { type: 'array', items: node.children },
      ],
    })
  }
}
```

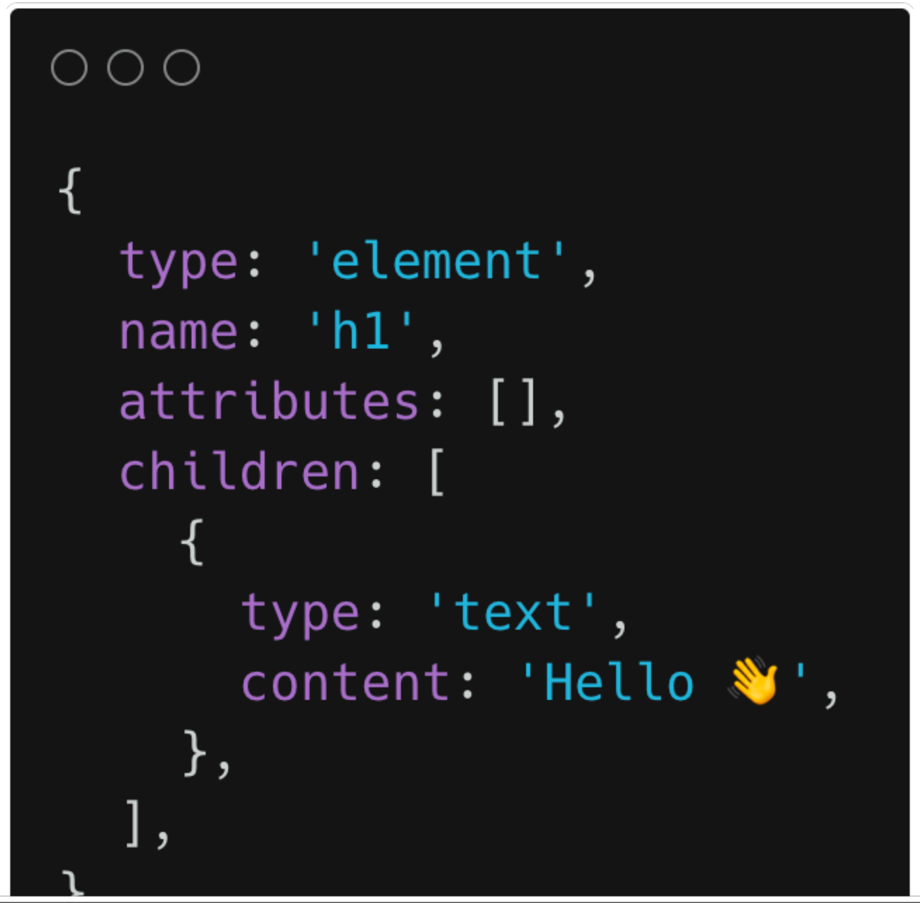


```
{
  type: 'call expression',
  callee: 'createVNode',
  arguments: [
    'h1',
    { type: 'object', properties: [] },
    {
      type: 'array',
      items: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
```

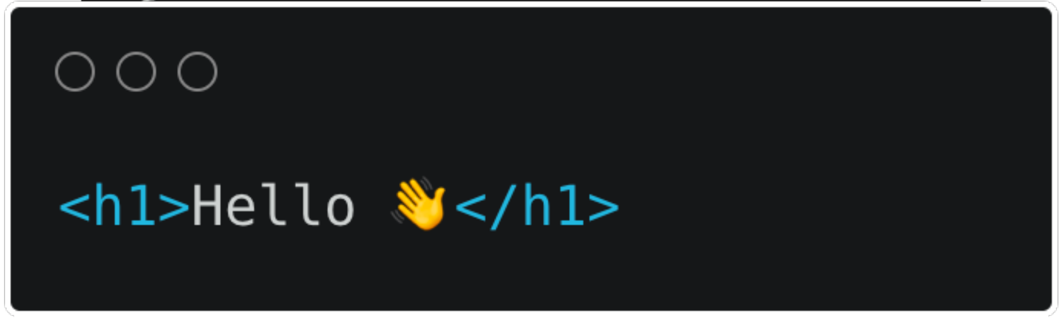


```
createVNode('h1', null, ['Hello 👋'])
```

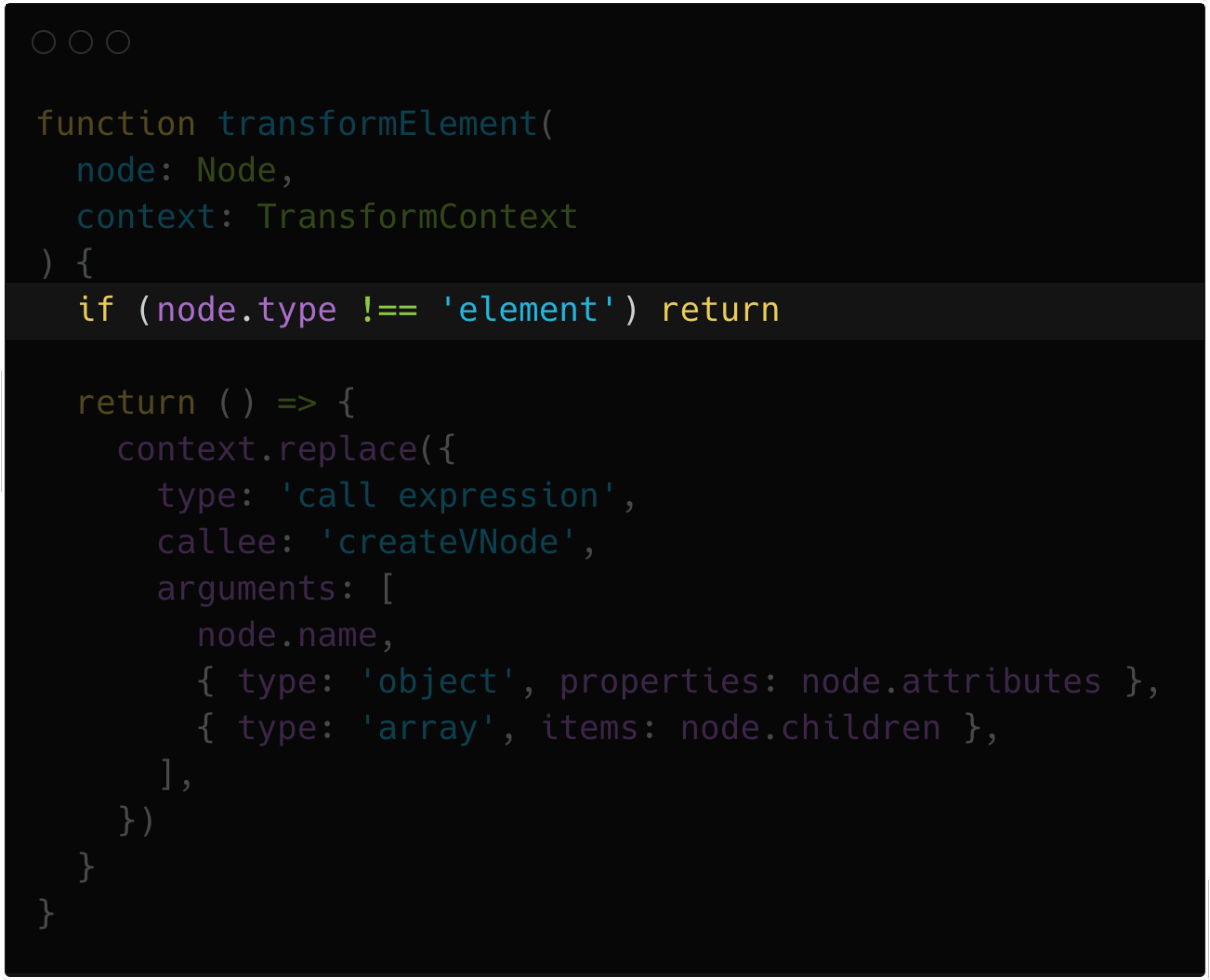
Transformations



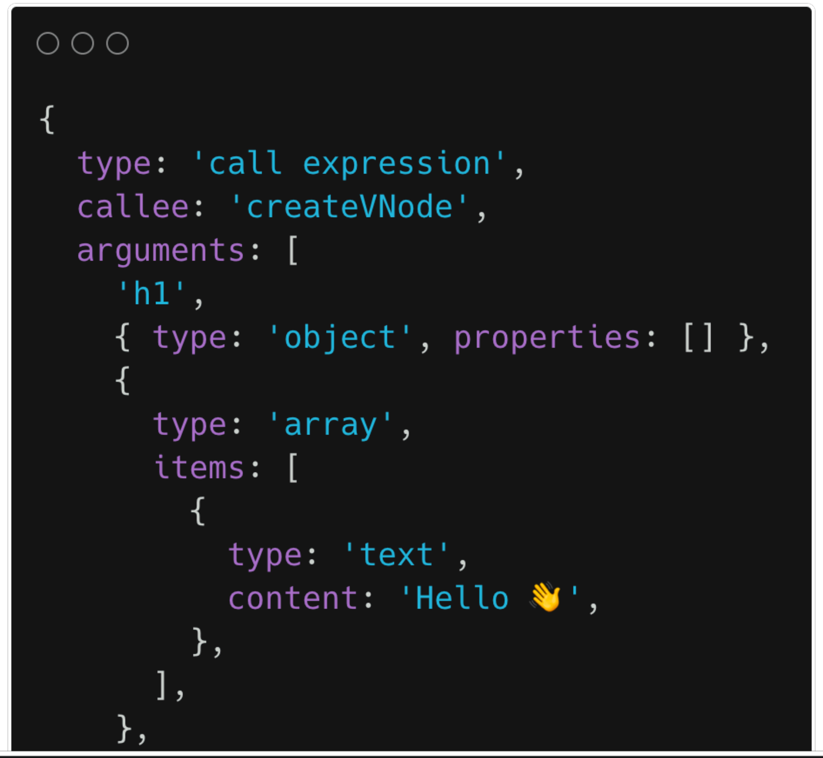
```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
```



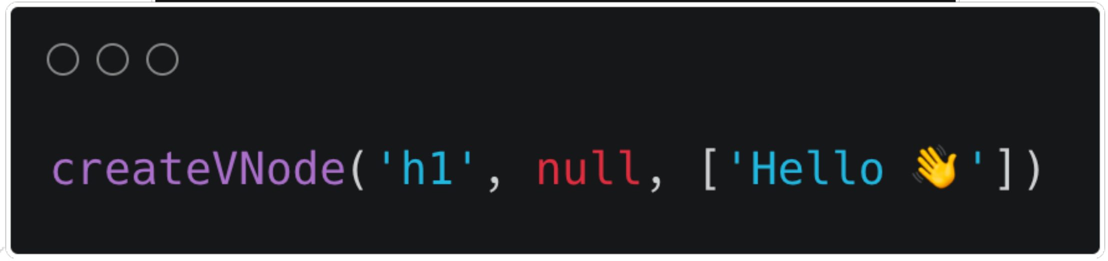
```
<h1>Hello 👋</h1>
```



```
function transformElement(
  node: Node,
  context: TransformContext
) {
  if (node.type !== 'element') return

  return () => {
    context.replace({
      type: 'call expression',
      callee: 'createVNode',
      arguments: [
        node.name,
        { type: 'object', properties: node.attributes },
        { type: 'array', items: node.children },
      ],
    })
  }
}
```



```
{
  type: 'call expression',
  callee: 'createVNode',
  arguments: [
    'h1',
    { type: 'object', properties: [] },
    {
      type: 'array',
      items: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
```



```
createVNode('h1', null, ['Hello 👋'])
```

Transformations

```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
```

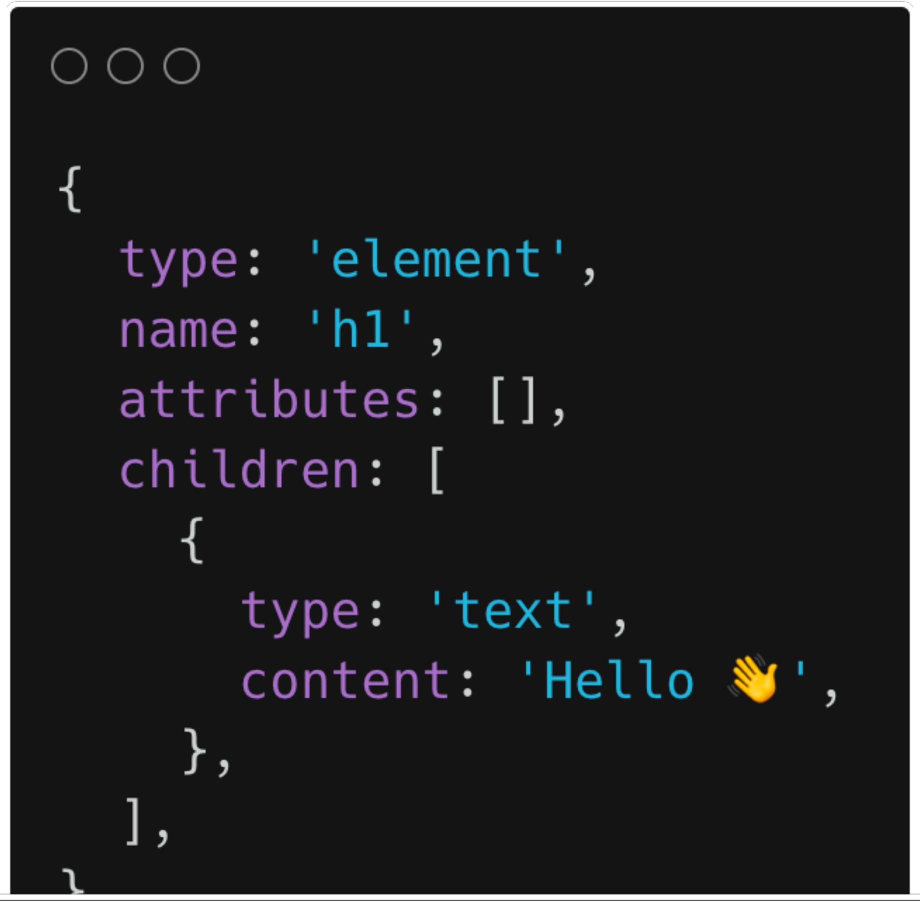
```
<h1>Hello 👋</h1>
```

```
function transformElement(
  node: Node,
  context: TransformContext
) {
  if (node.type !== 'element') return

  return () => {
    context.replace({
      type: 'call expression',
      callee: 'createVNode',
      arguments: [
        node.name,
        { type: 'object', properties: node.attributes },
        { type: 'array', items: node.children },
      ],
    })
  }
}
```

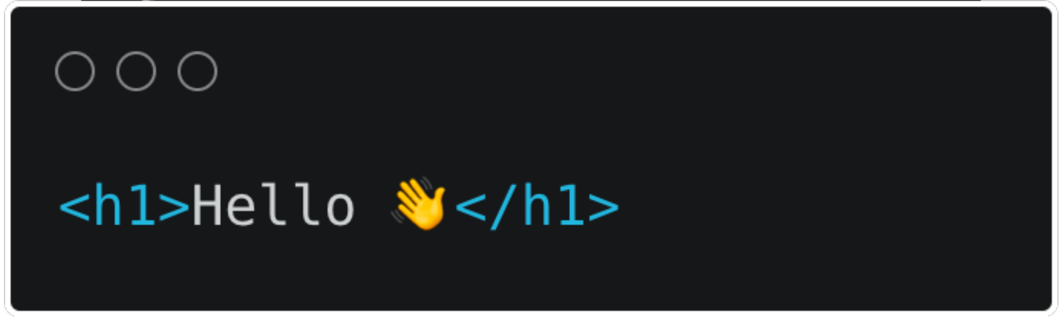
```
{
  type: 'call expression',
  callee: 'createVNode',
  arguments: [
    'h1',
    { type: 'object', properties: [] },
    {
      type: 'array',
      items: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
```

```
createVNode('h1', null, ['Hello 👋'])
```

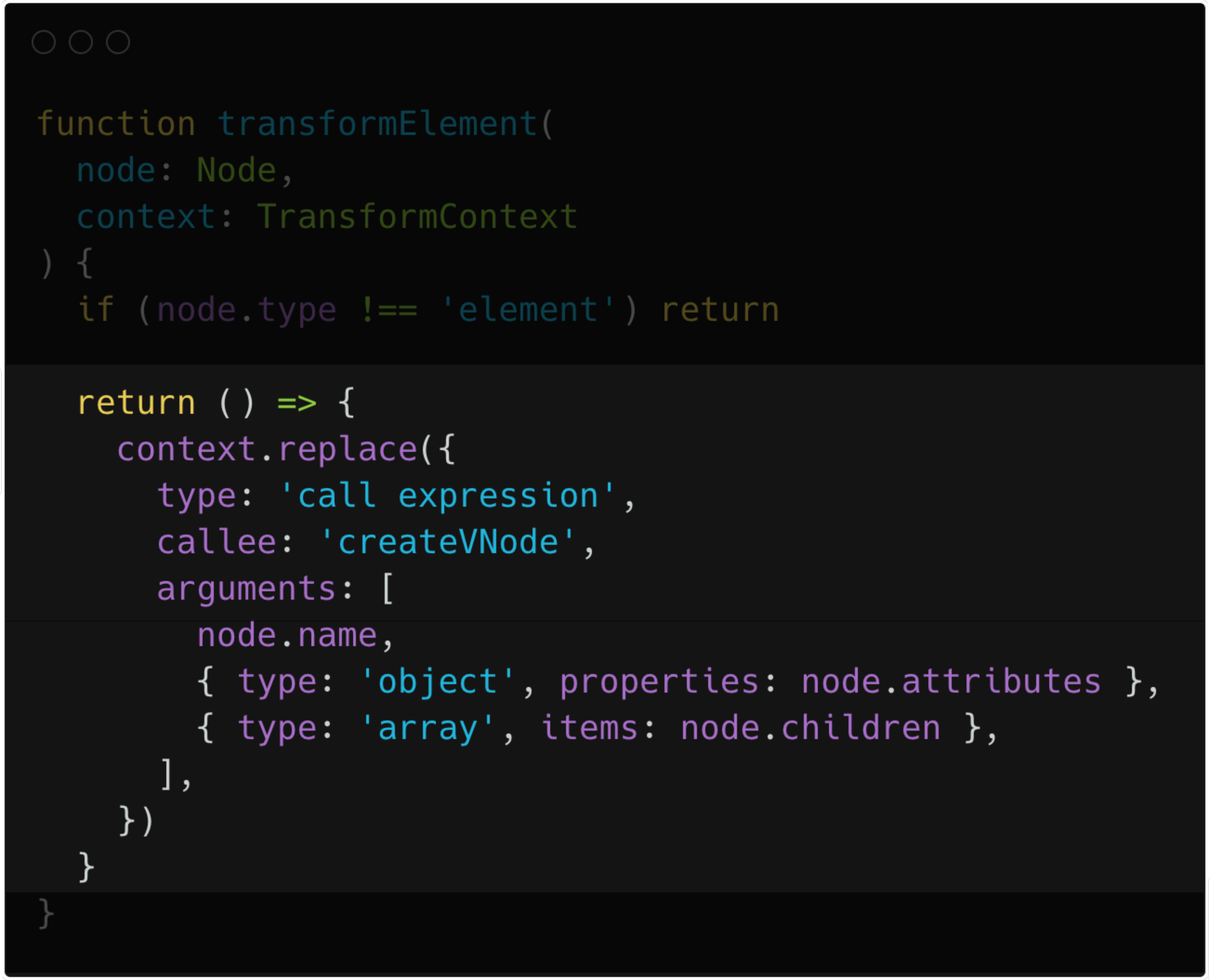
Transformations



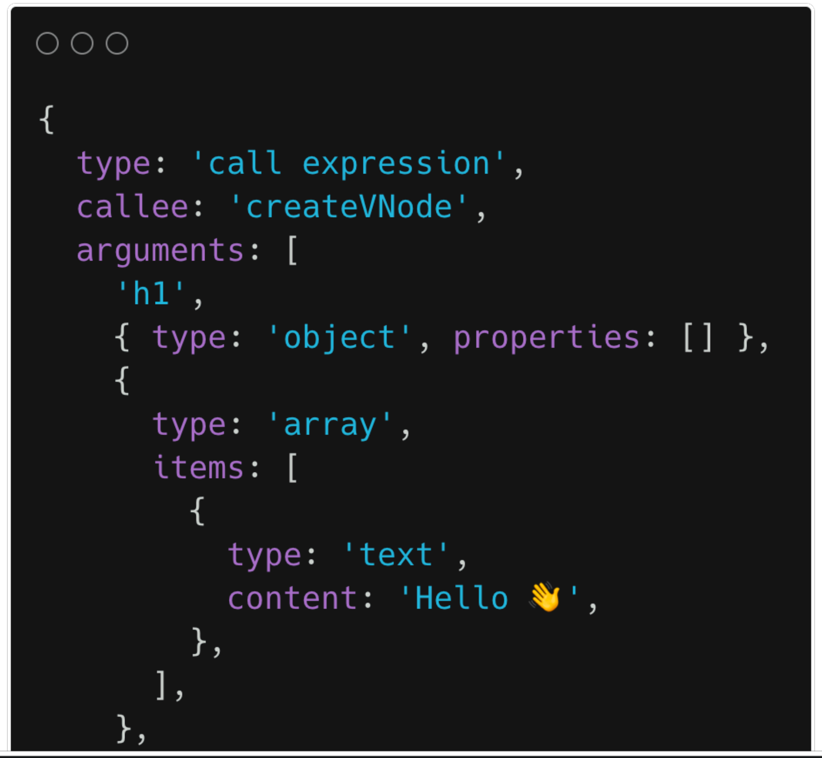
```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
```



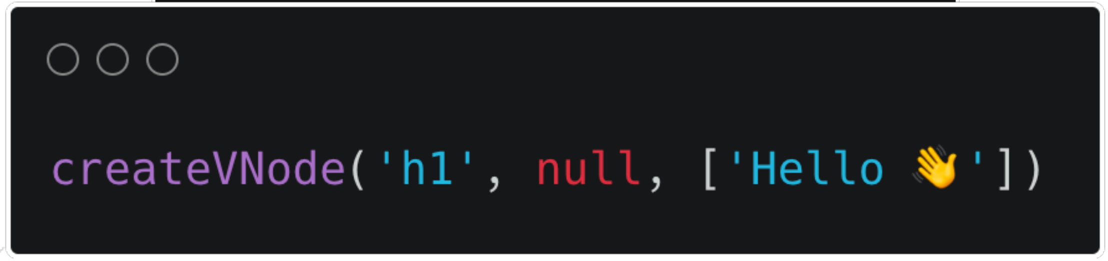
```
<h1>Hello 👋</h1>
```



```
function transformElement(
  node: Node,
  context: TransformContext
) {
  if (node.type !== 'element') return

  return () => {
    context.replace({
      type: 'call expression',
      callee: 'createVNode',
      arguments: [
        node.name,
        { type: 'object', properties: node.attributes },
        { type: 'array', items: node.children },
      ],
    })
  }
}
```



```
{
  type: 'call expression',
  callee: 'createVNode',
  arguments: [
    'h1',
    { type: 'object', properties: [] },
    {
      type: 'array',
      items: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
```



```
createVNode('h1', null, ['Hello 👋'])
```

Transformations

```
{
  type: 'element',
  name: 'h1',
  attributes: [],
  children: [
    {
      type: 'text',
      content: 'Hello 👋',
    },
  ],
}
```

```
formElement(

nsformContext

e !== 'element') return

 {
place({
all expression',
'createVNode',
s: [
ame,
: 'object', properties: 
'array', items: node.ch
```

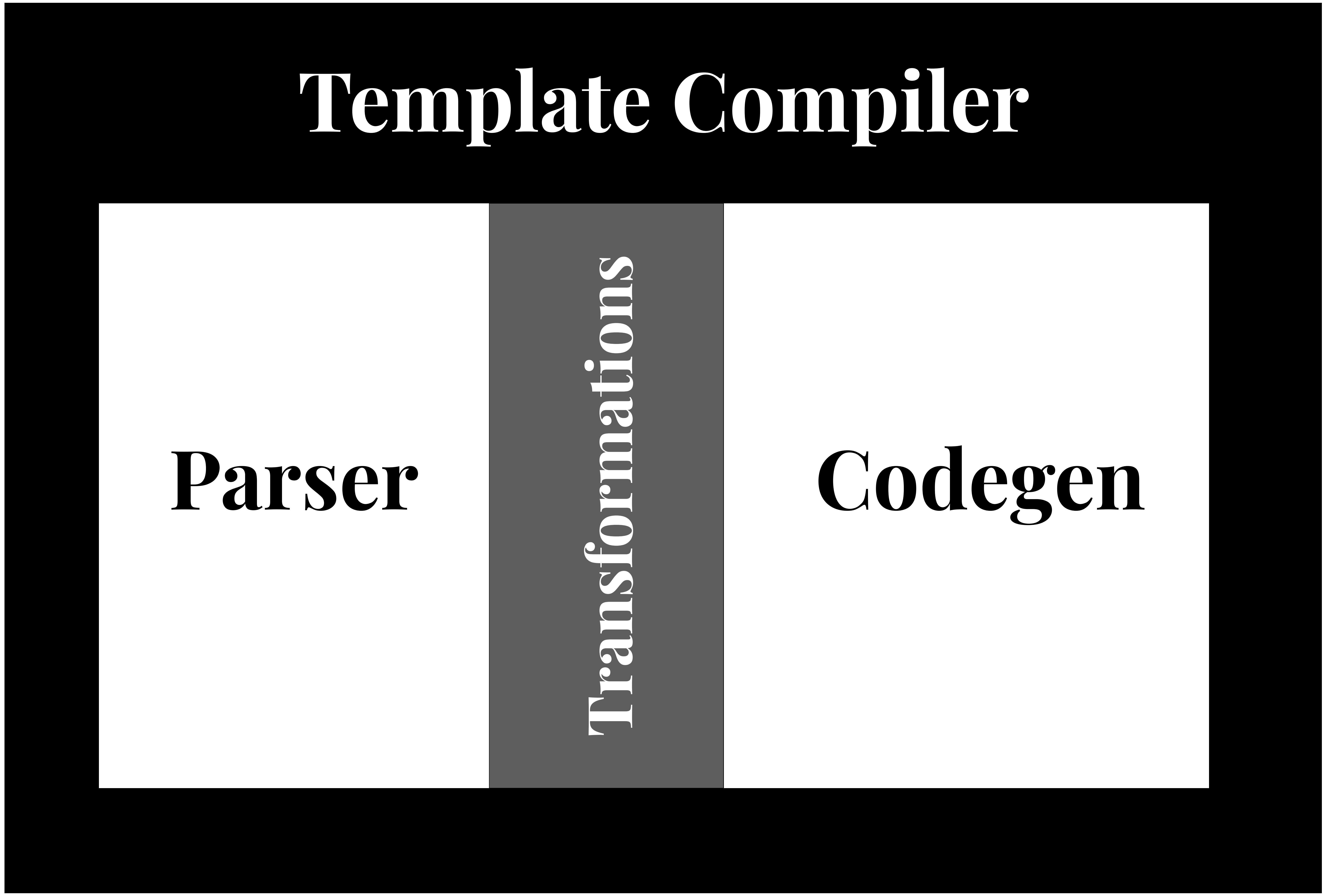
```
{
  type: 'call expression',
  callee: 'createVNode',
  arguments: [
    'h1',
    { type: 'object', properties: [] },
    {
      type: 'array',
      items: [
        {
          type: 'text',
          content: 'Hello 👋',
        },
      ],
    },
```

```
<h1>Hello 👋</h1>
```

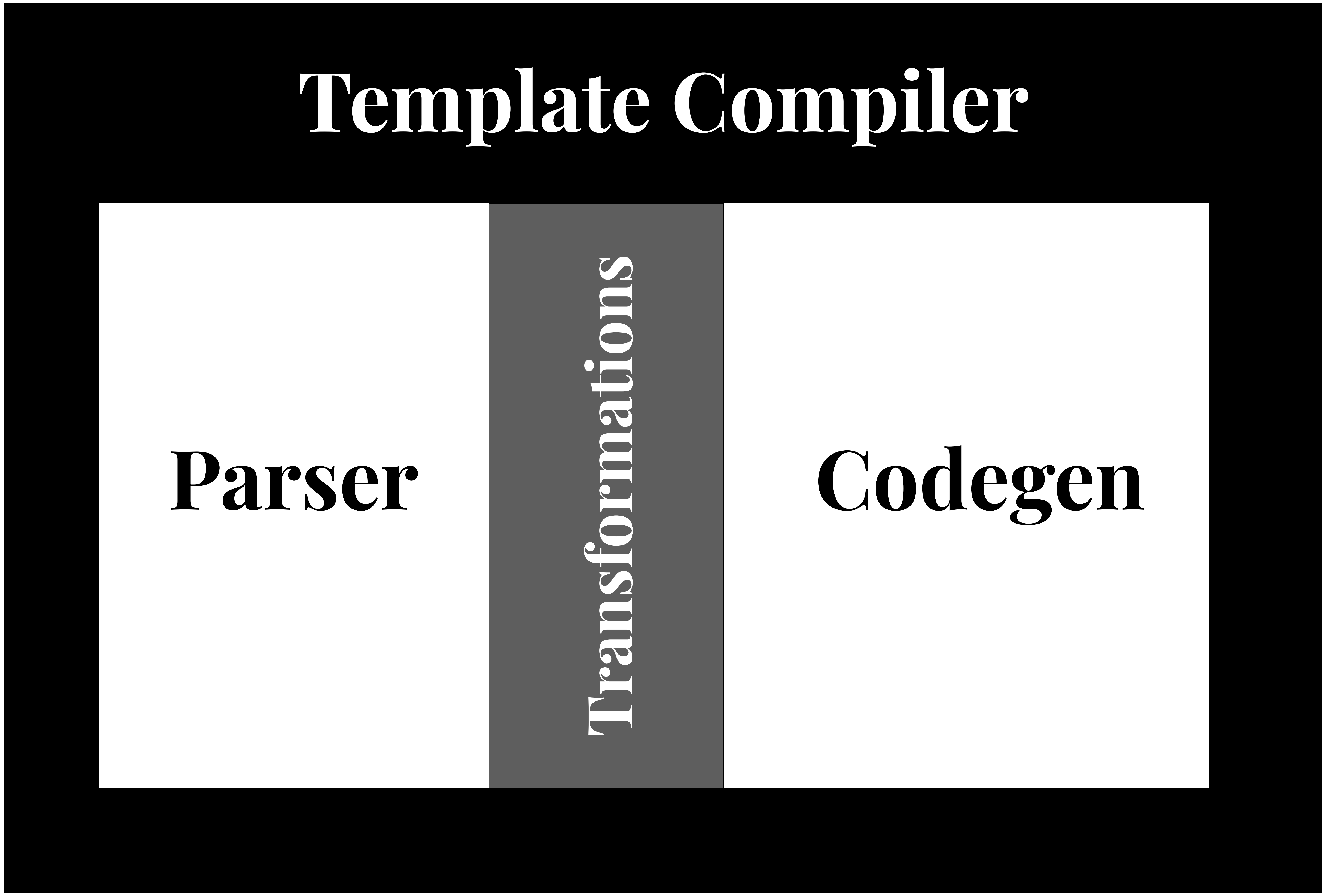
```
createVNode('h1', null, ['Hello 👋'])
```

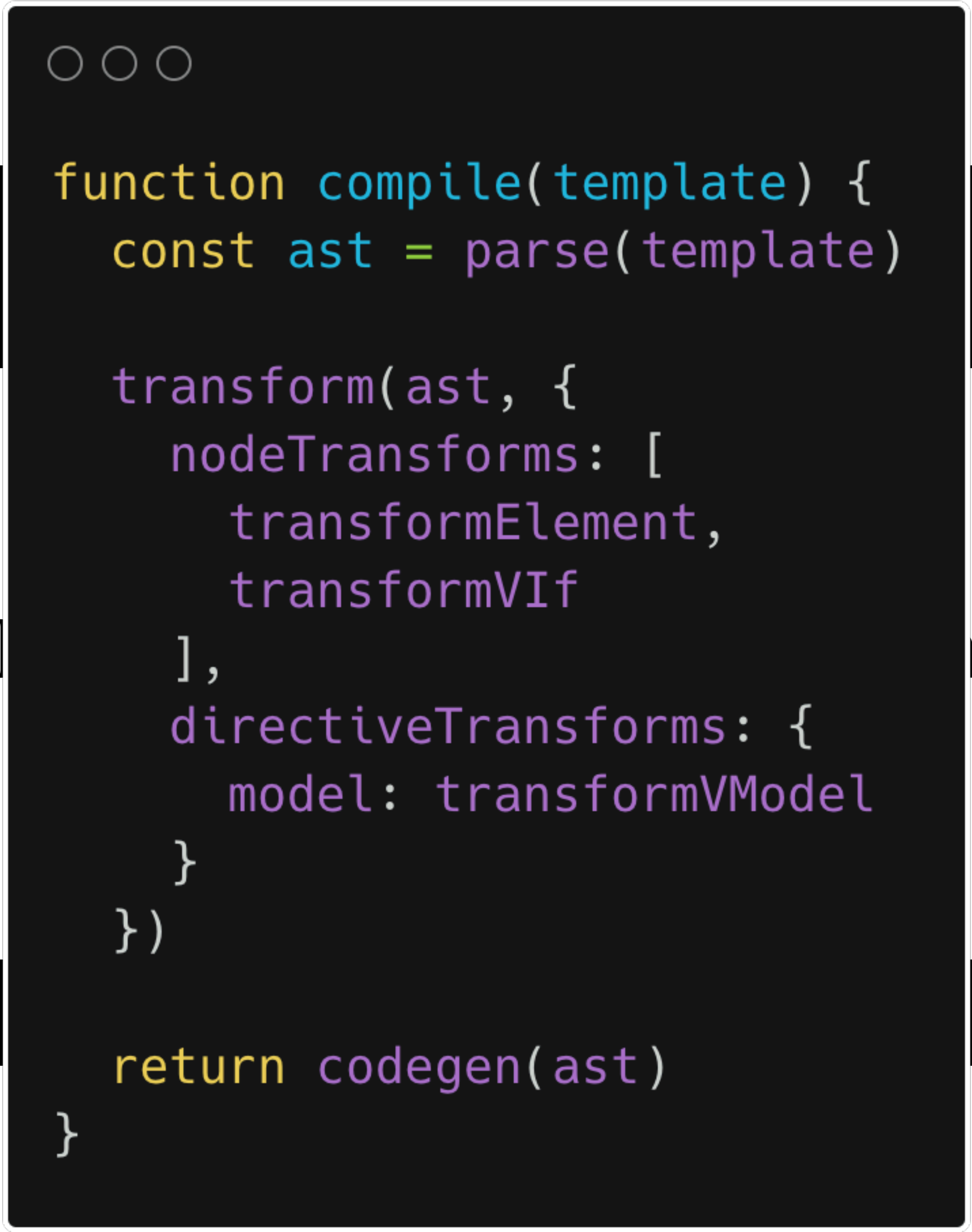
Transformations

Transformations

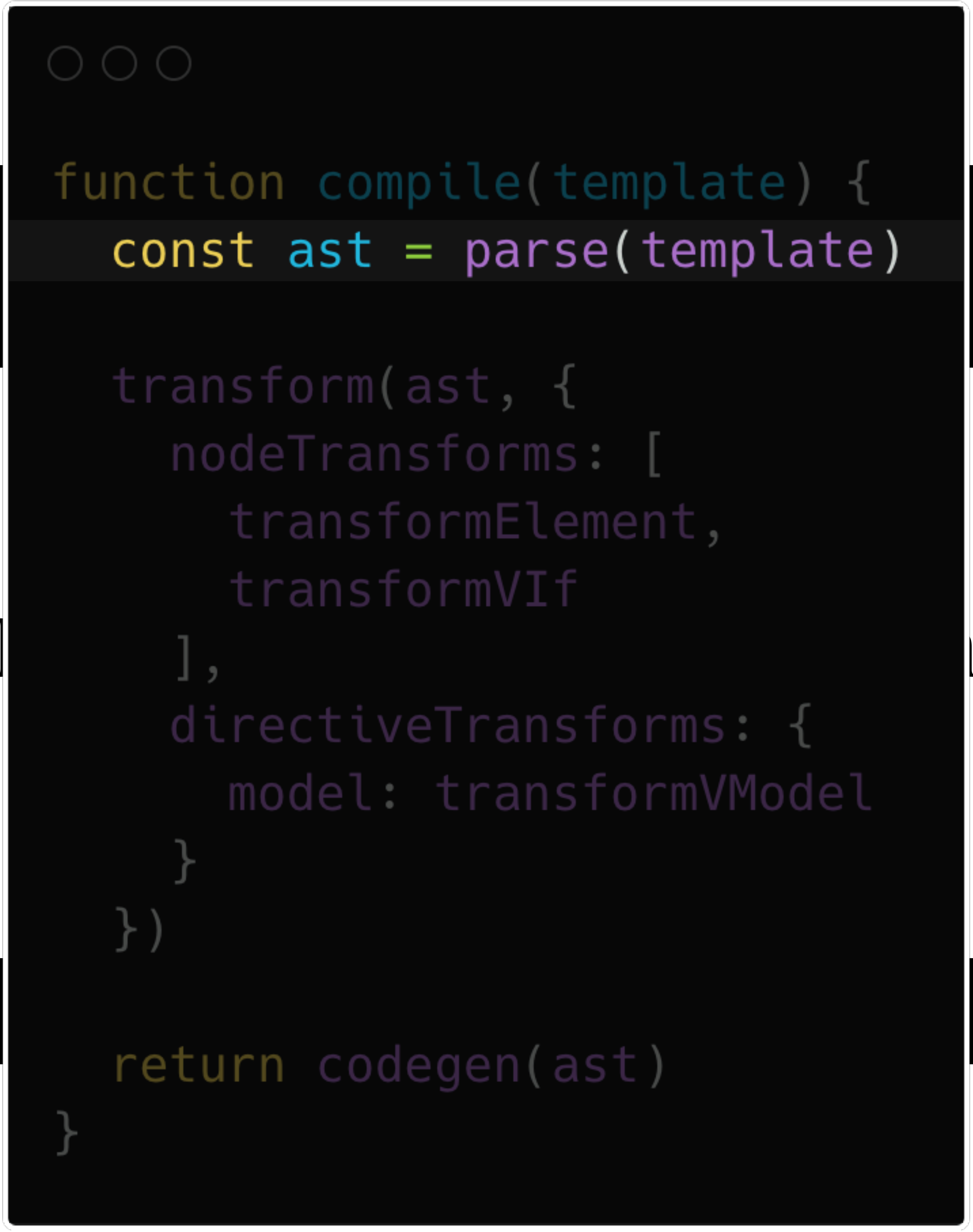

Template Compiler
Parser
Transformations
Codegen

Codegen

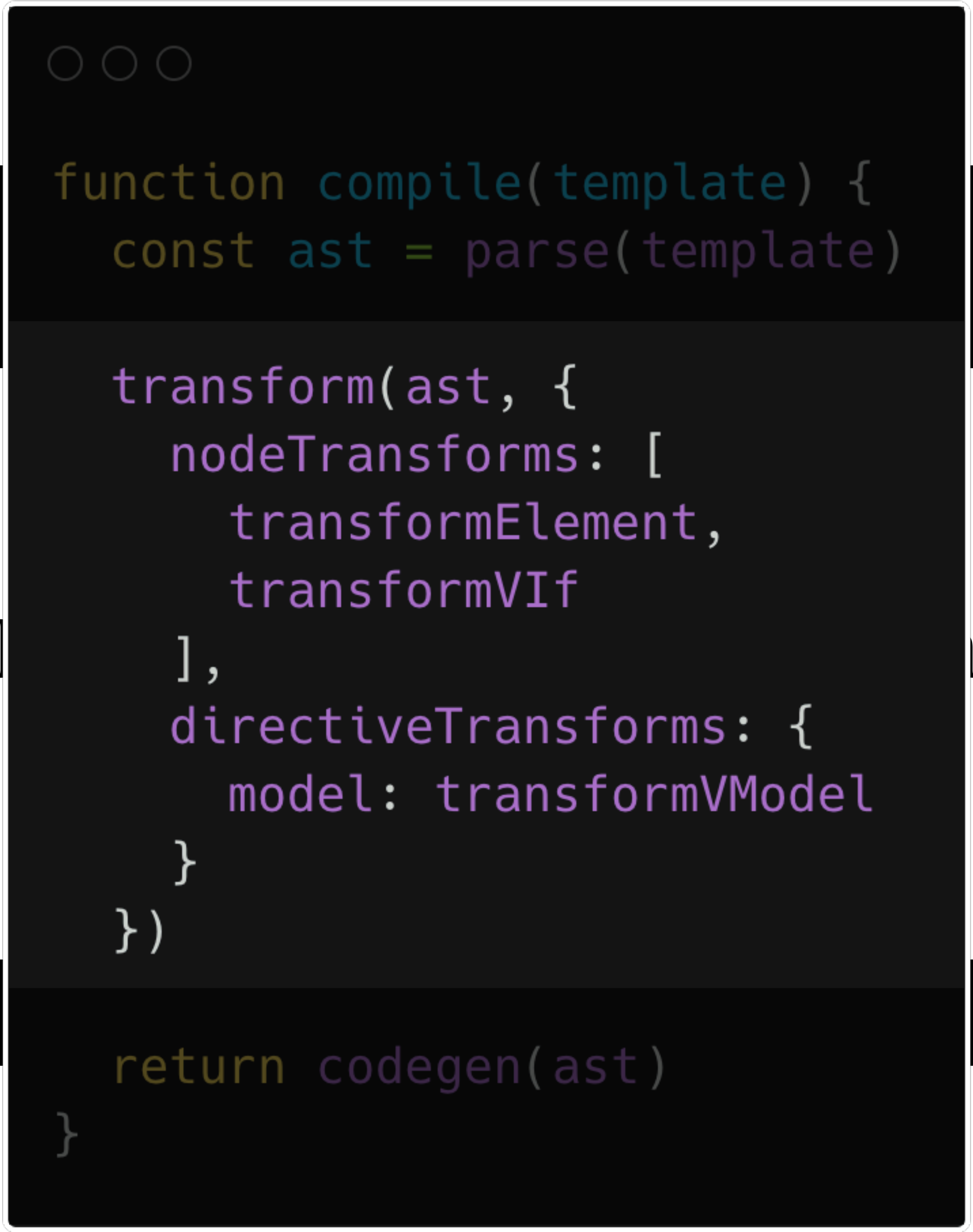

Template Compiler
Parser
Transformations
Codegen



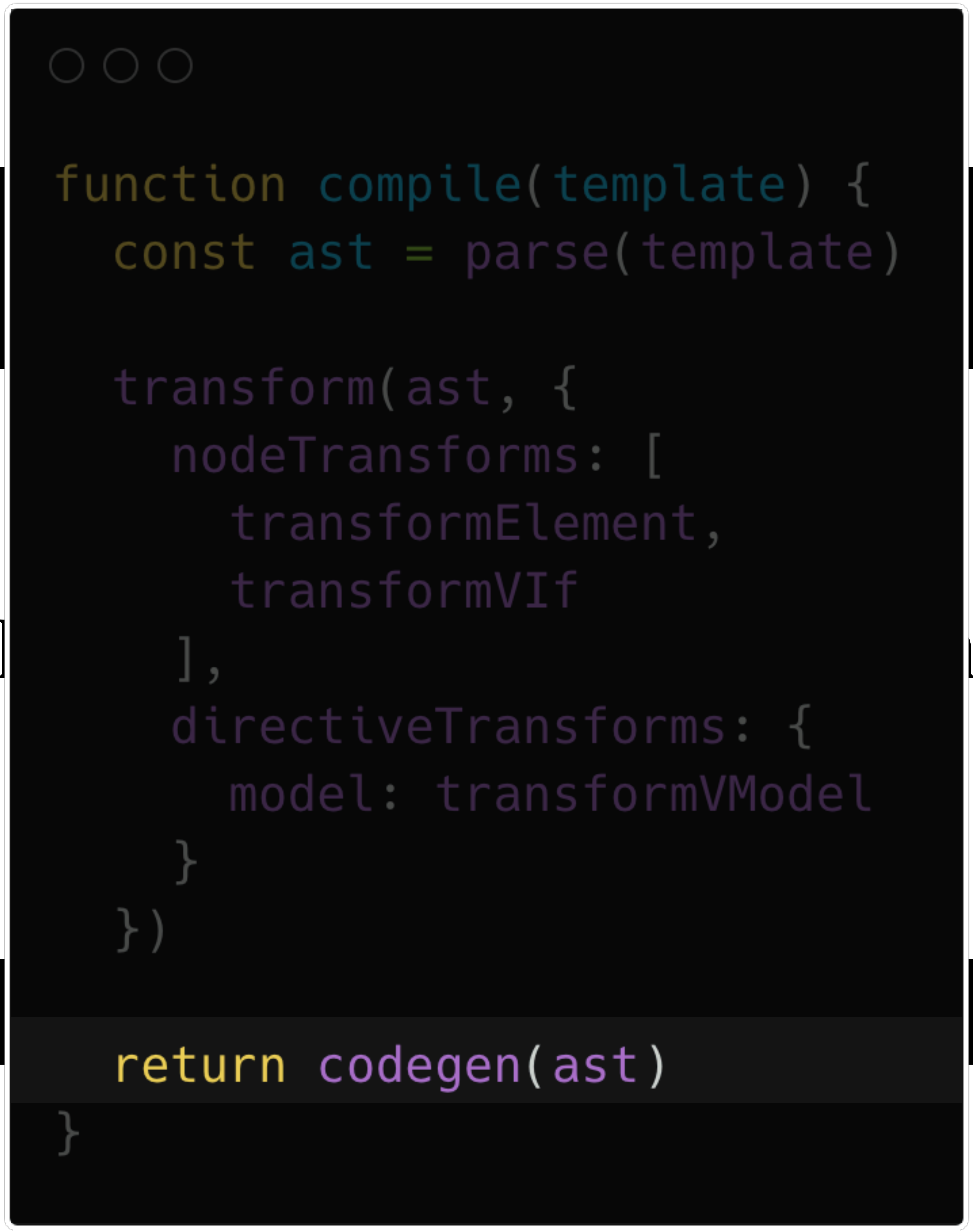
```
function compile(template) {
  const ast = parse(template)

  transform(ast, {
    nodeTransforms: [
      transformElement,
      transformVIf
    ],
    directiveTransforms: {
      model: transformVModel
    }
  })

  return codegen(ast)
}
```



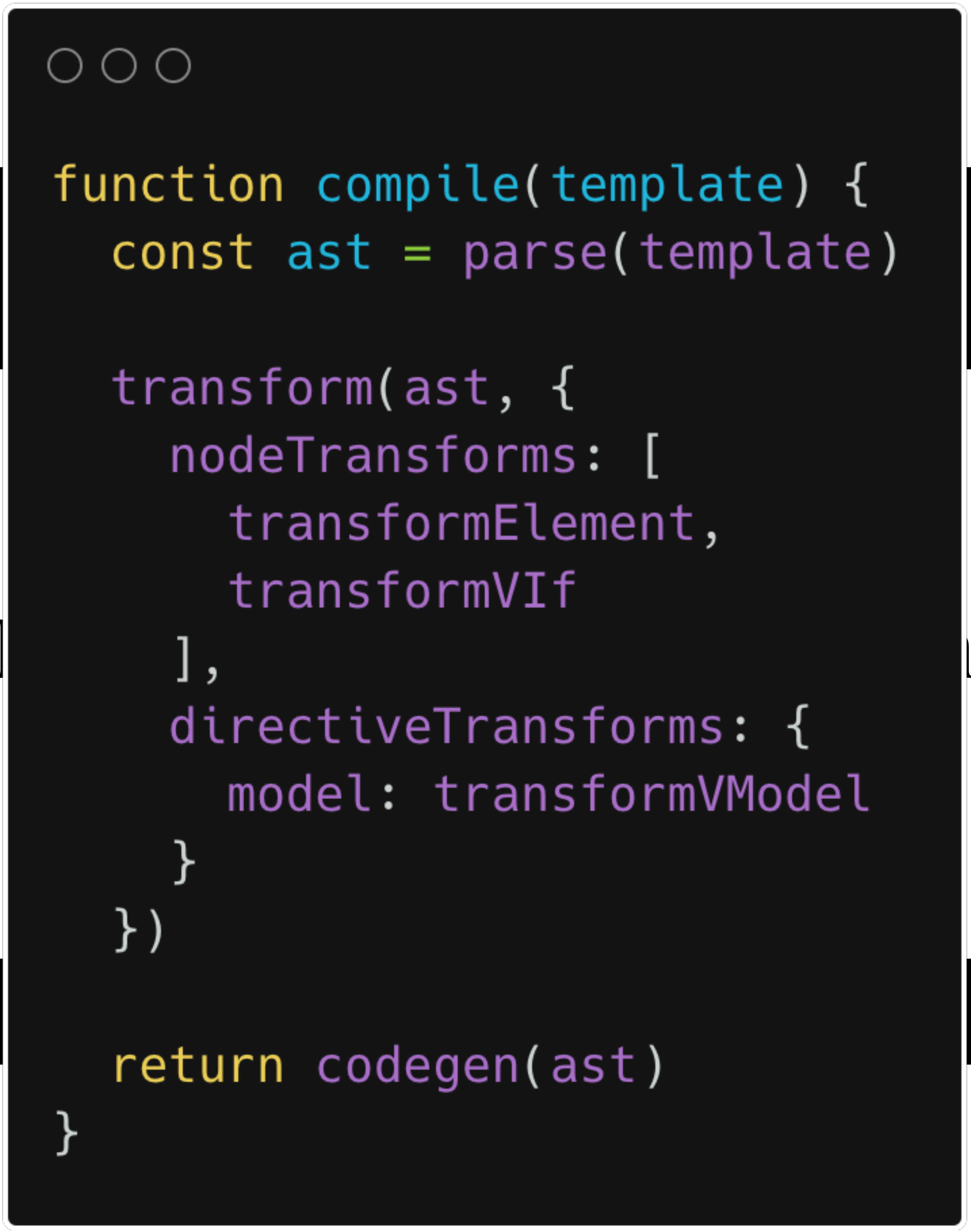
```
function compile(template) {
  const ast = parse(template)

  transform(ast, {
    nodeTransforms: [
      transformElement,
      transformVIf
    ],
    directiveTransforms: {
      model: transformVModel
    }
  })

  return codegen(ast)
}
```



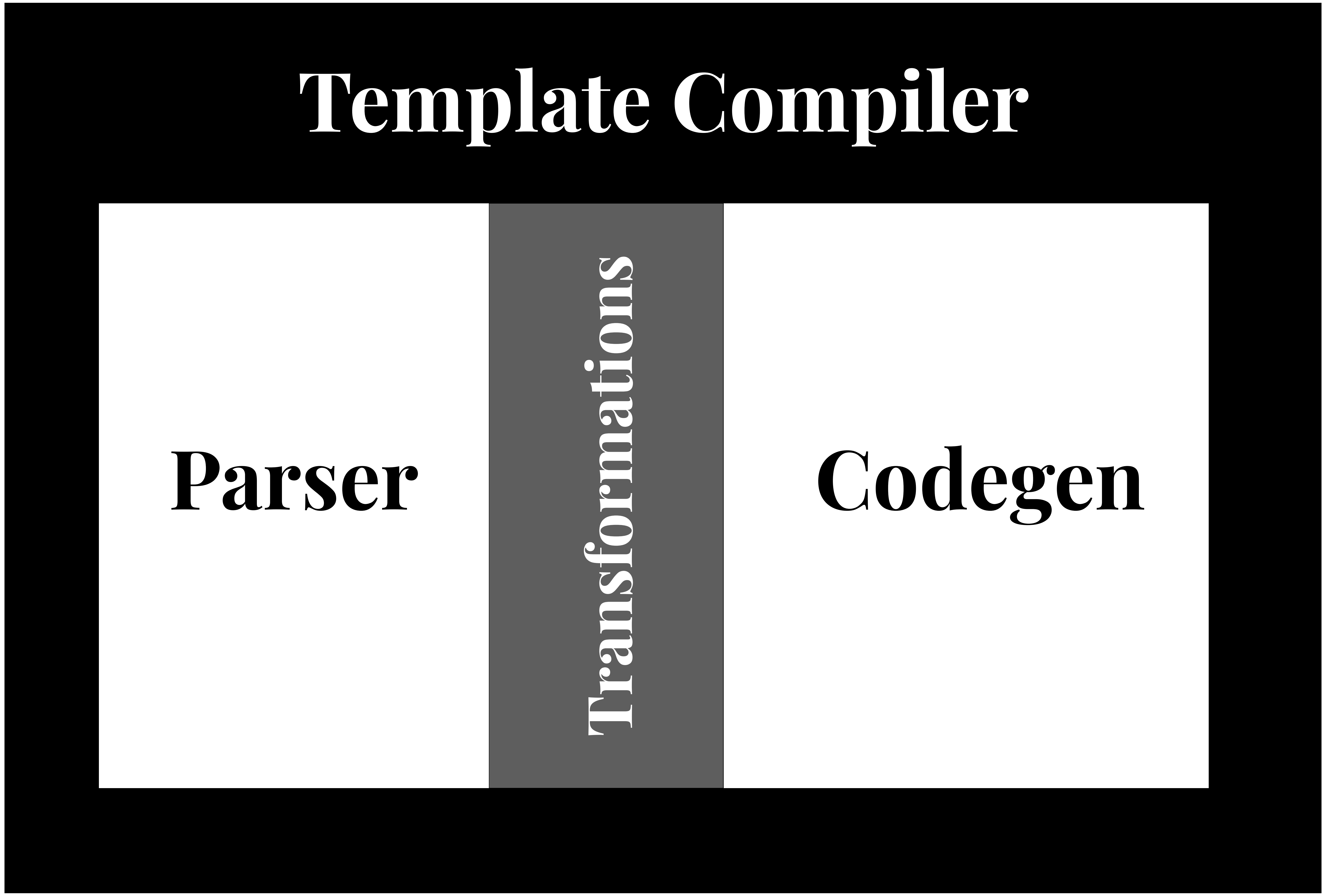
```
function compile(template) {
  const ast = parse(template)

  transform(ast, {
    nodeTransforms: [
      transformElement,
      transformVIf
    ],
    directiveTransforms: {
      model: transformVModel
    }
  })

  return codegen(ast)
}
```



```js
function compile(template) {
  const ast = parse(template)

  transform(ast, {
    nodeTransforms: [
      transformElement,
      transformVIf
    ],
    directiveTransforms: {
      model: transformVModel
    }
  })

  return codegen(ast)
}
```



```javascript
function compile(template) {
  const ast = parse(template)

  transform(ast, {
    nodeTransforms: [
      transformElement,
      transformVIf
    ],
    directiveTransforms: {
      model: transformVModel
    }
  })

  return codegen(ast)
}
```


Template Compiler
Parser
Transformations
Codegen

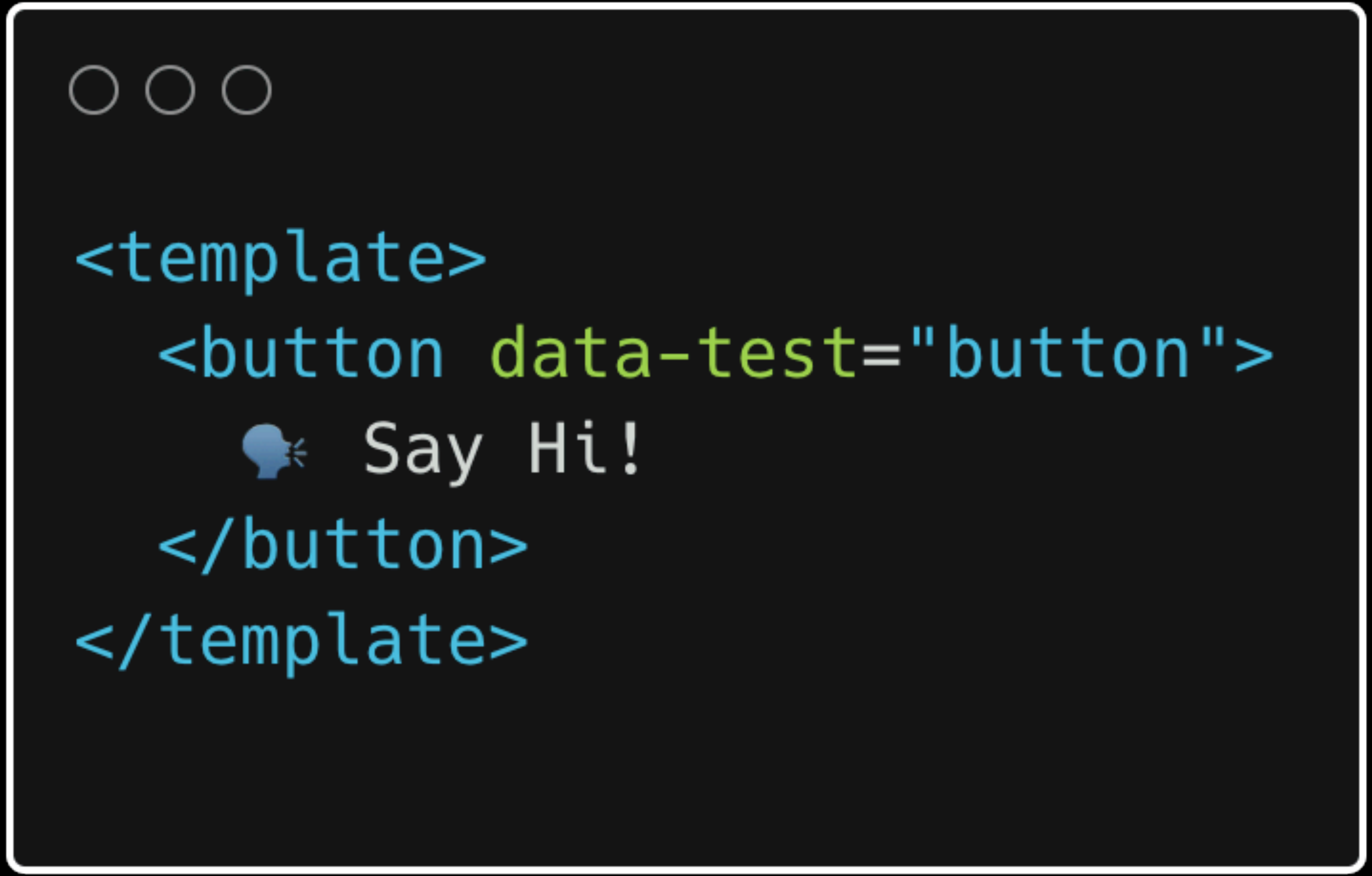
Why should **you** care?

Escape Hatch
in Unit Test

```
data-test="*"
```

Escape Hatch in Unit Test

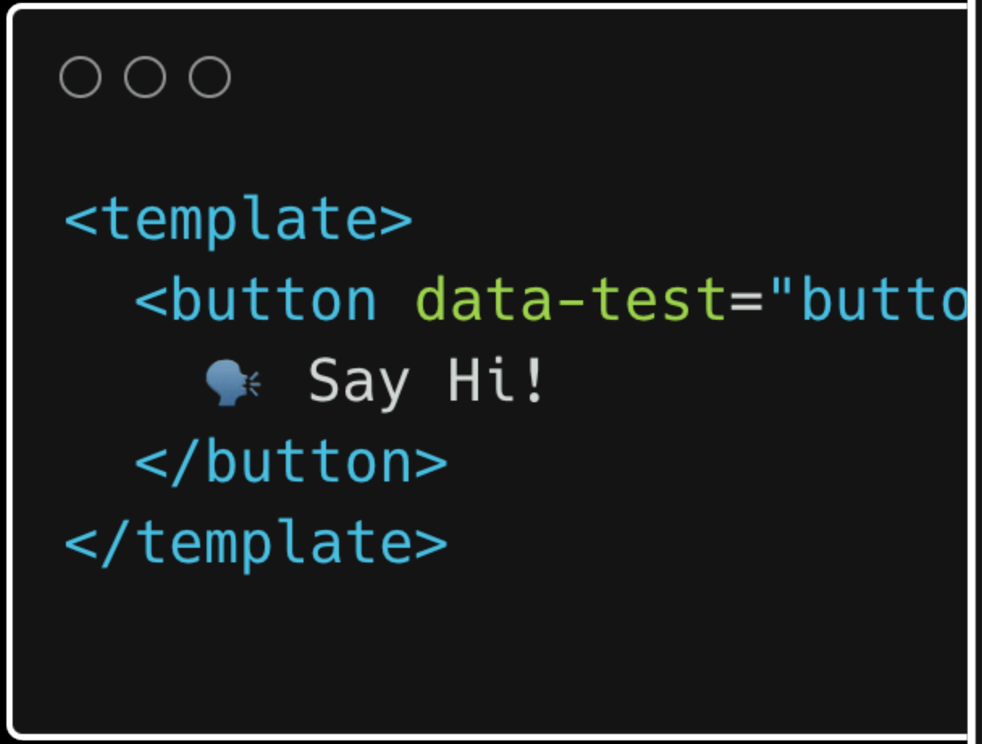
`data-test="*"`



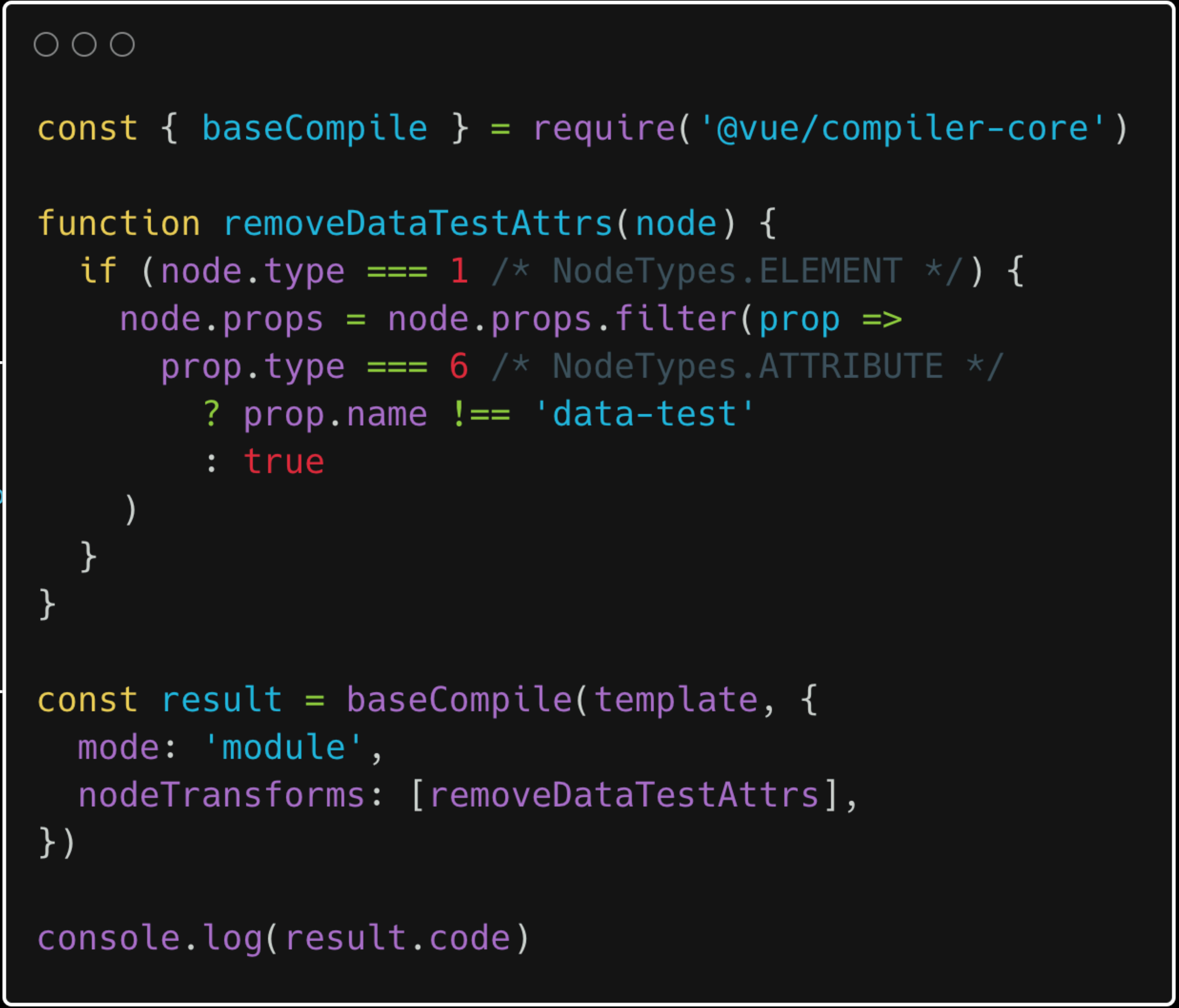
```
<template>
  <button data-test="button">
    🗣 Say Hi!
  </button>
</template>
```

Escape Hatch in Unit Test

data-test="*"



```
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```



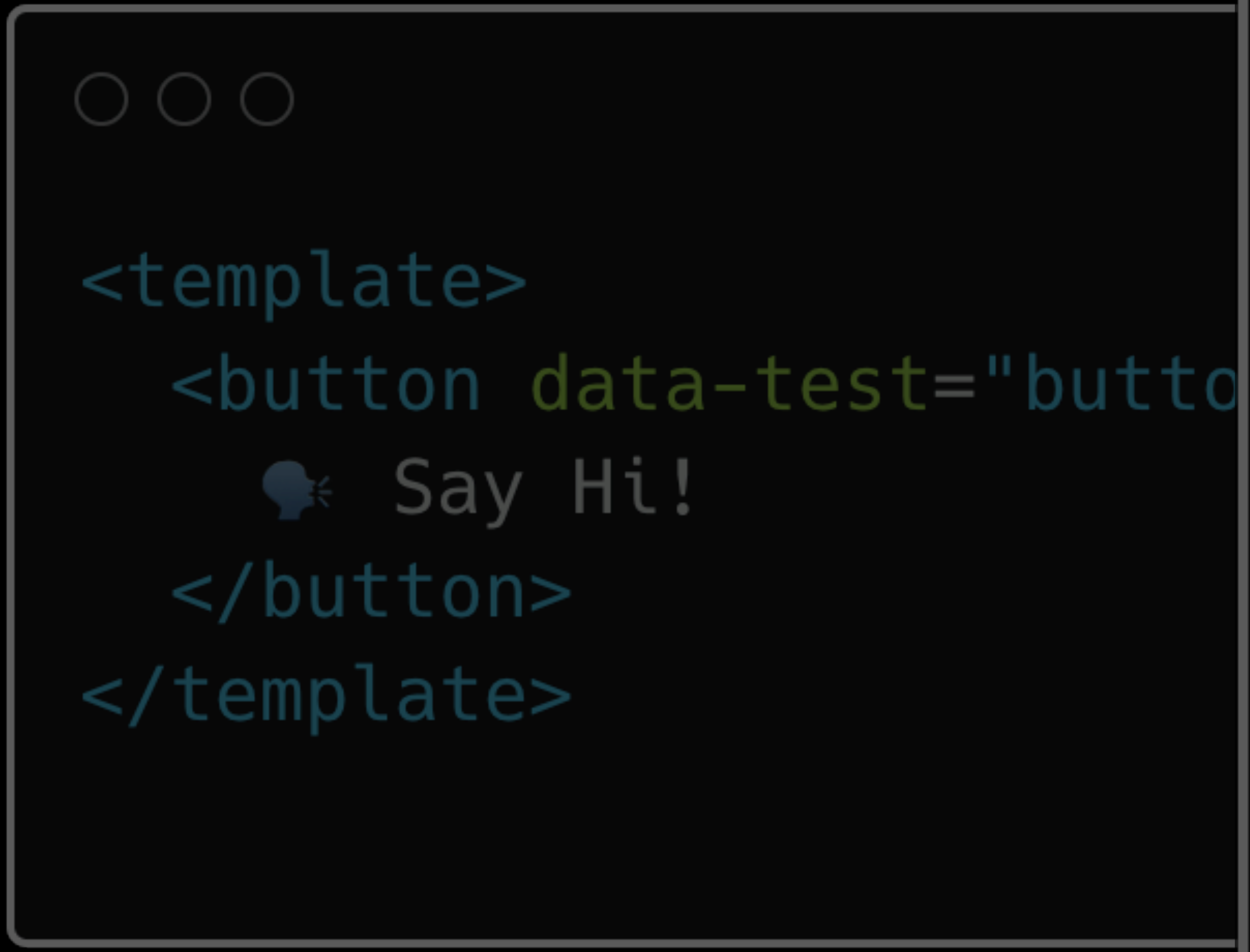
```
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```

```html
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```

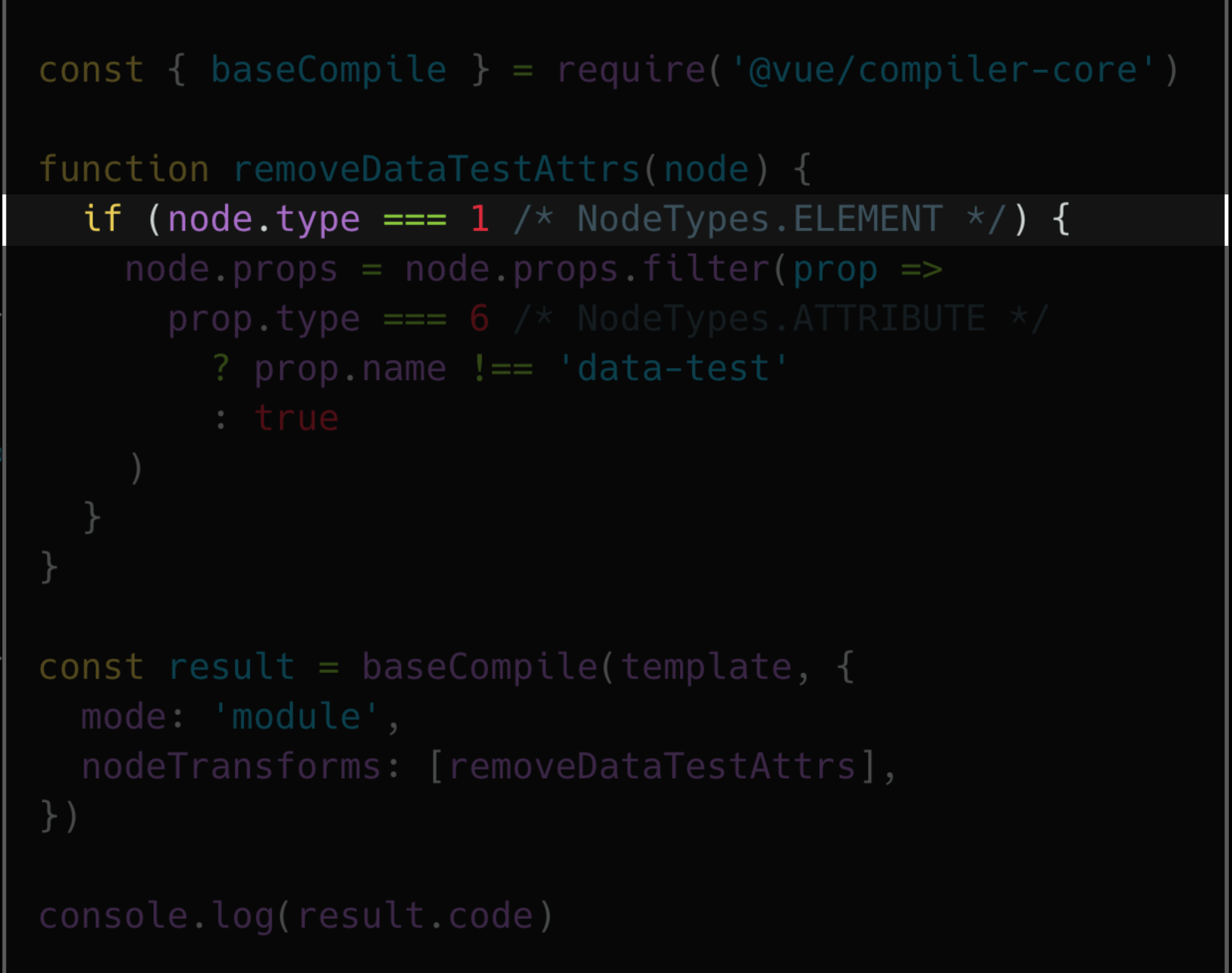
```js
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```

```
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```

```js
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```



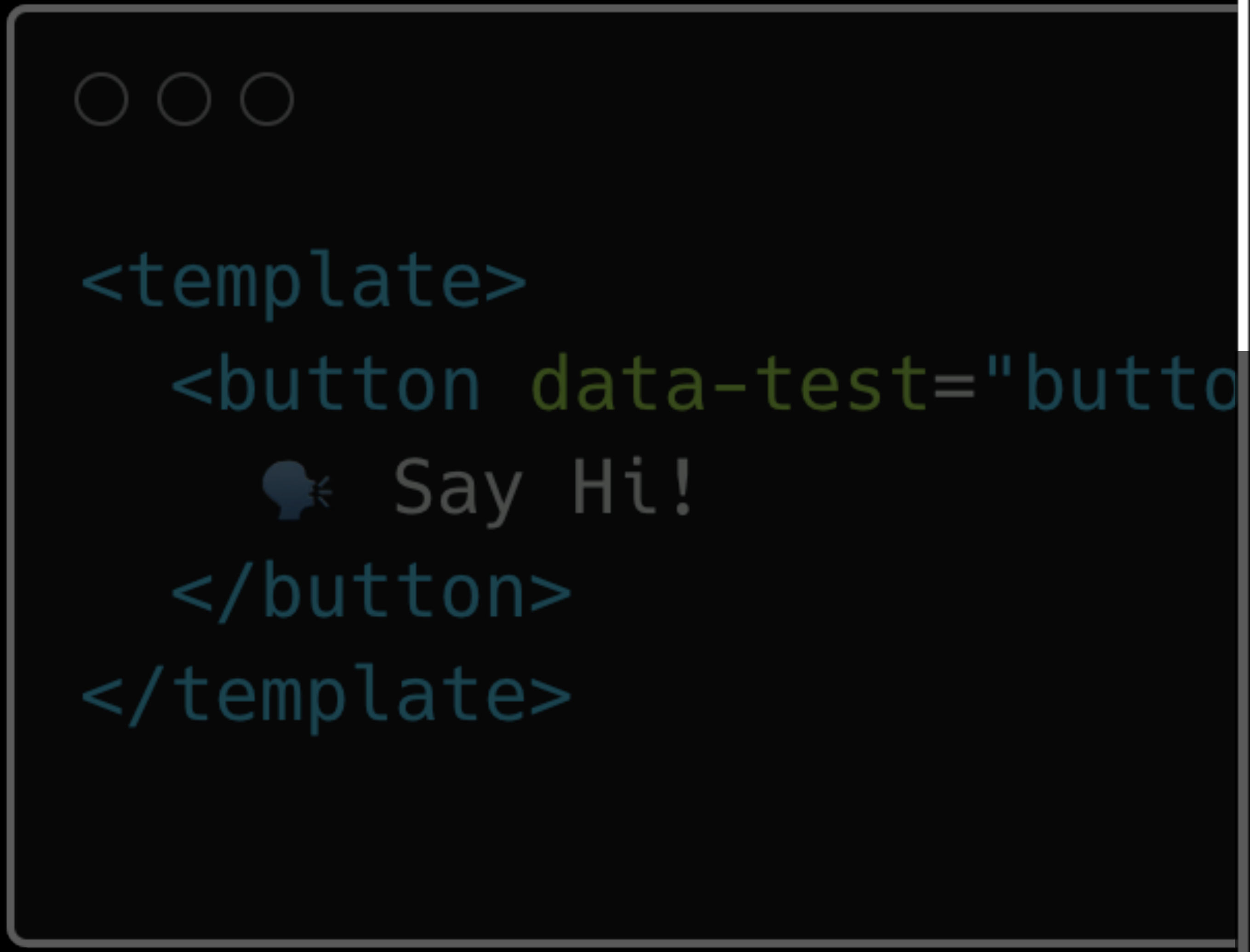
```
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```



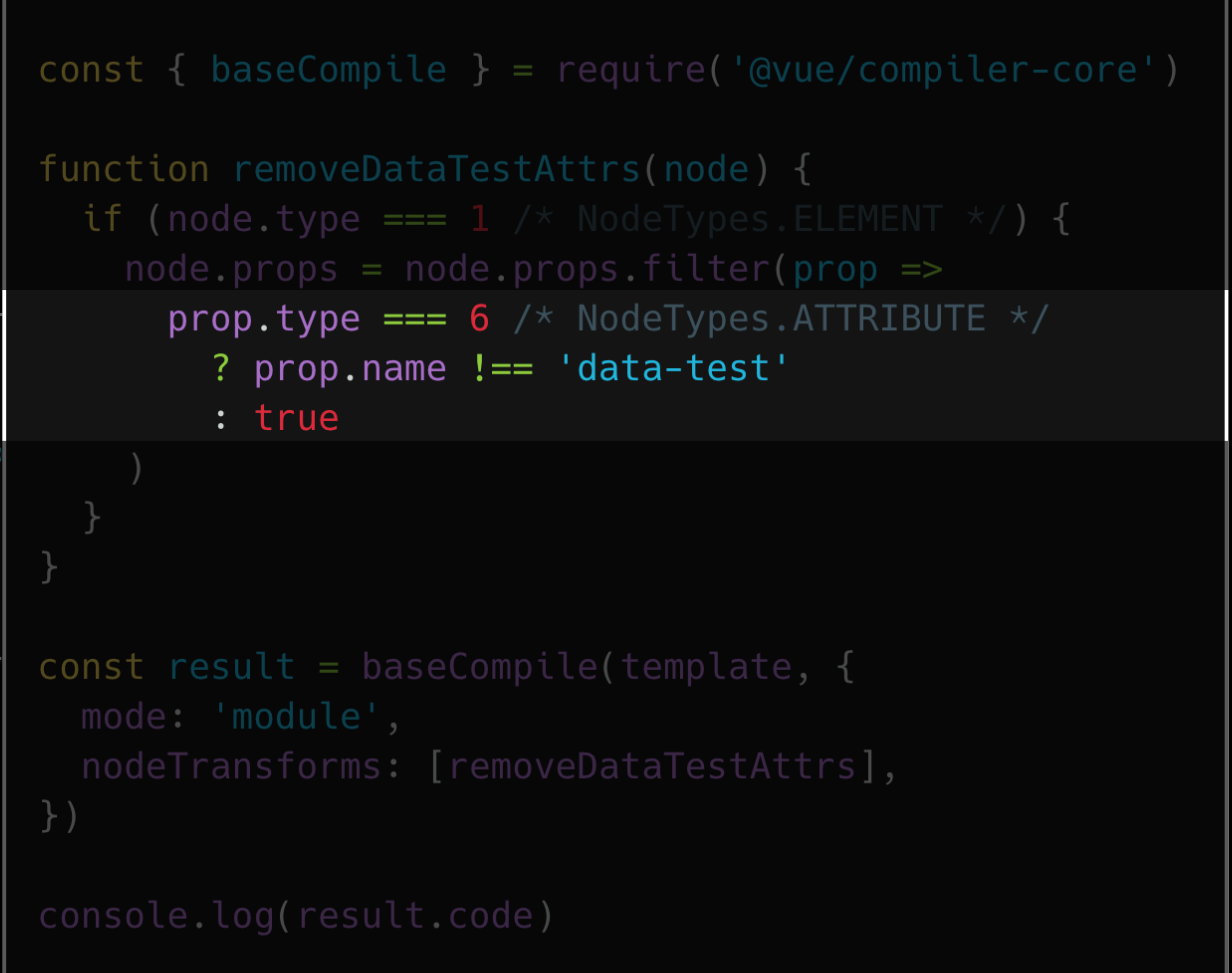
```
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```

```html
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```

```js
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```



```
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```



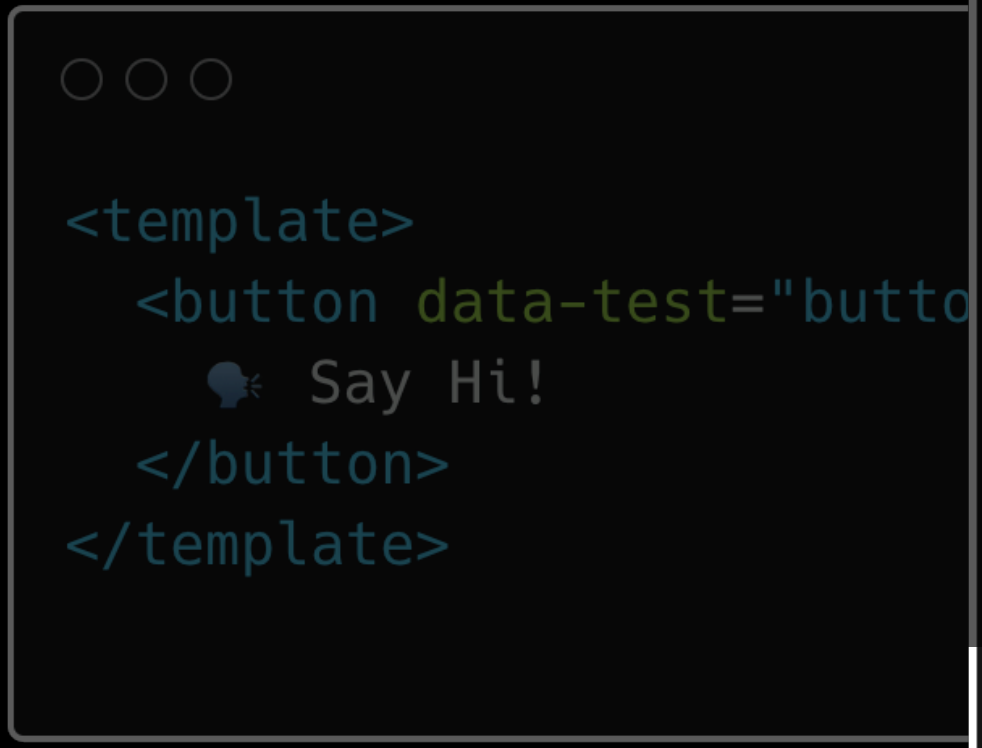
```
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```

```
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```

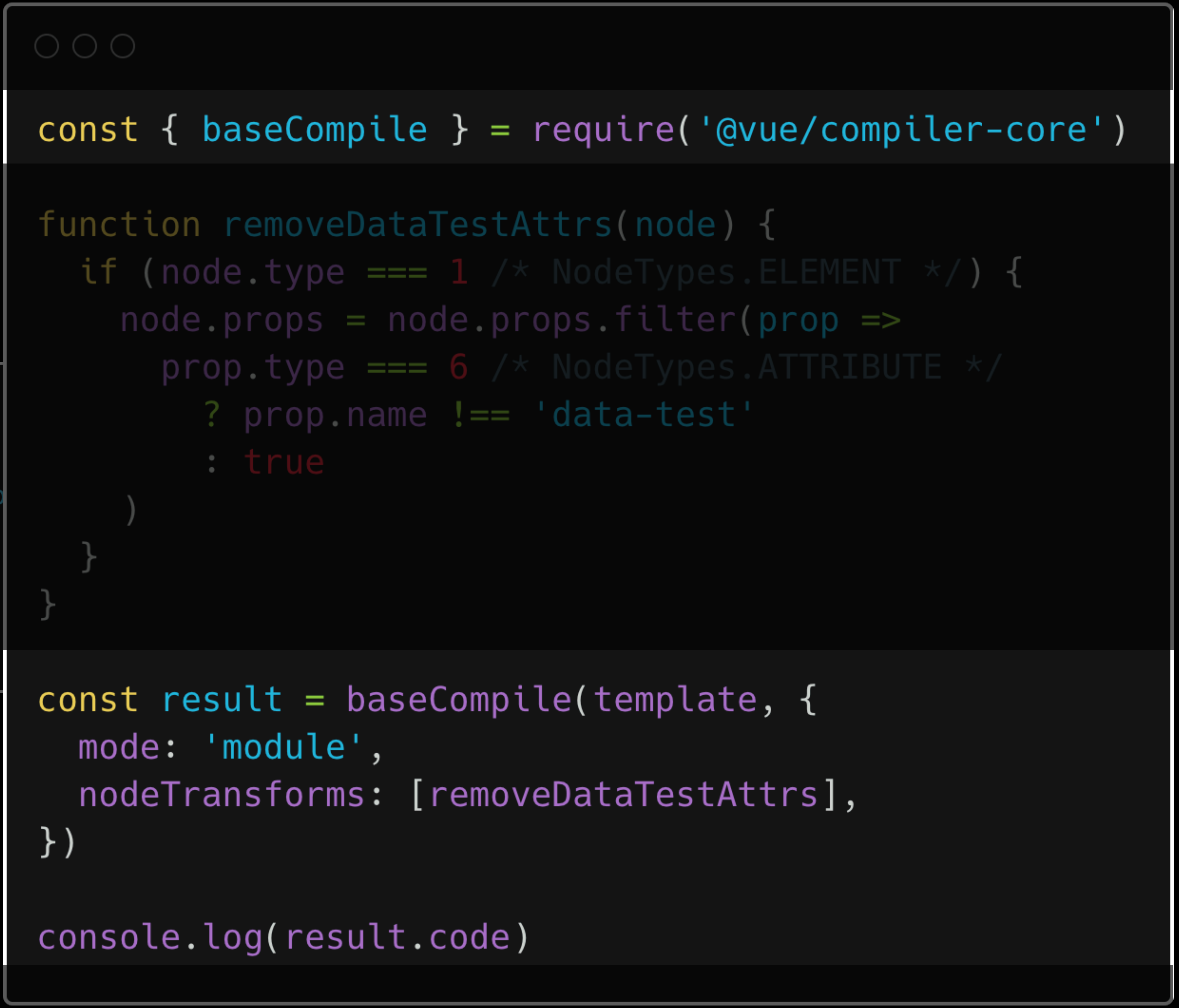
```
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```

Escape Hatch in Unit Test

`data-test="*"`



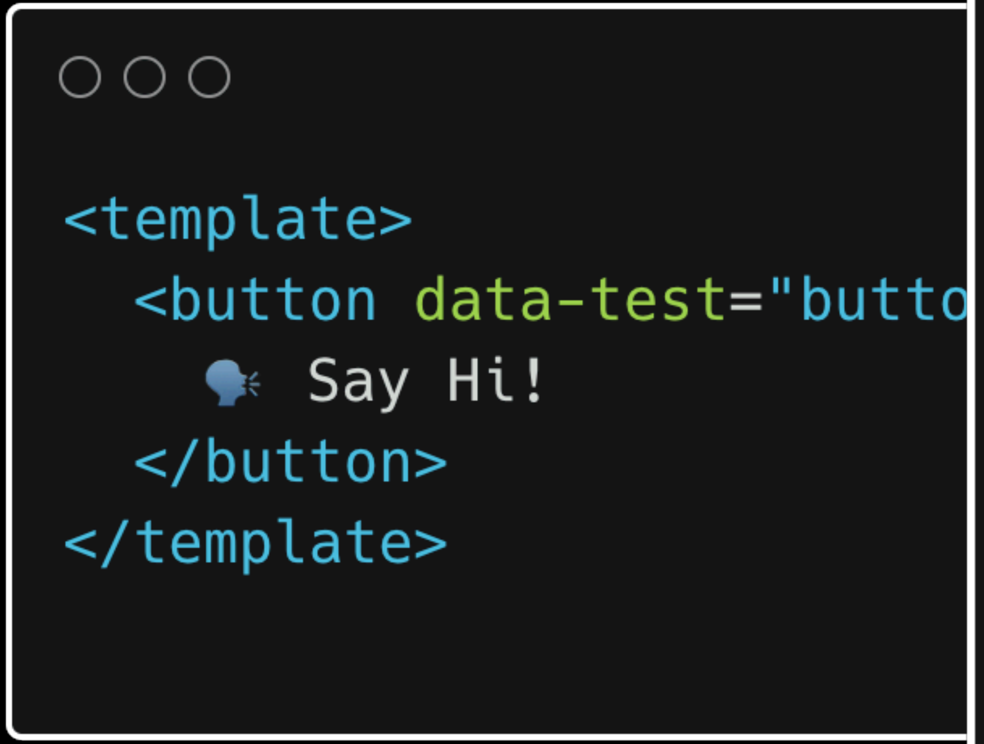
```
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```



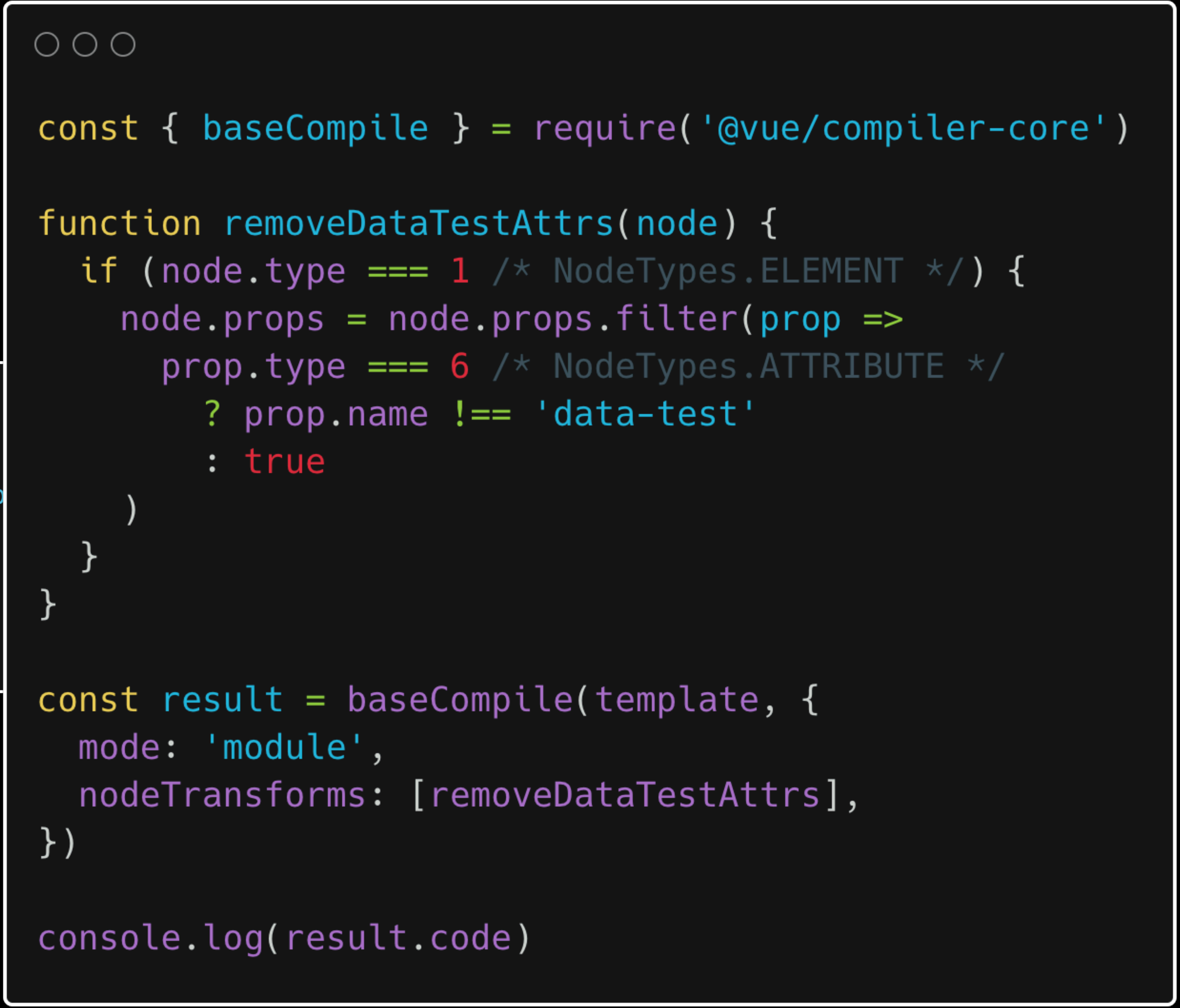
```
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```

Escape Hatch in Unit Test

data-test="*"



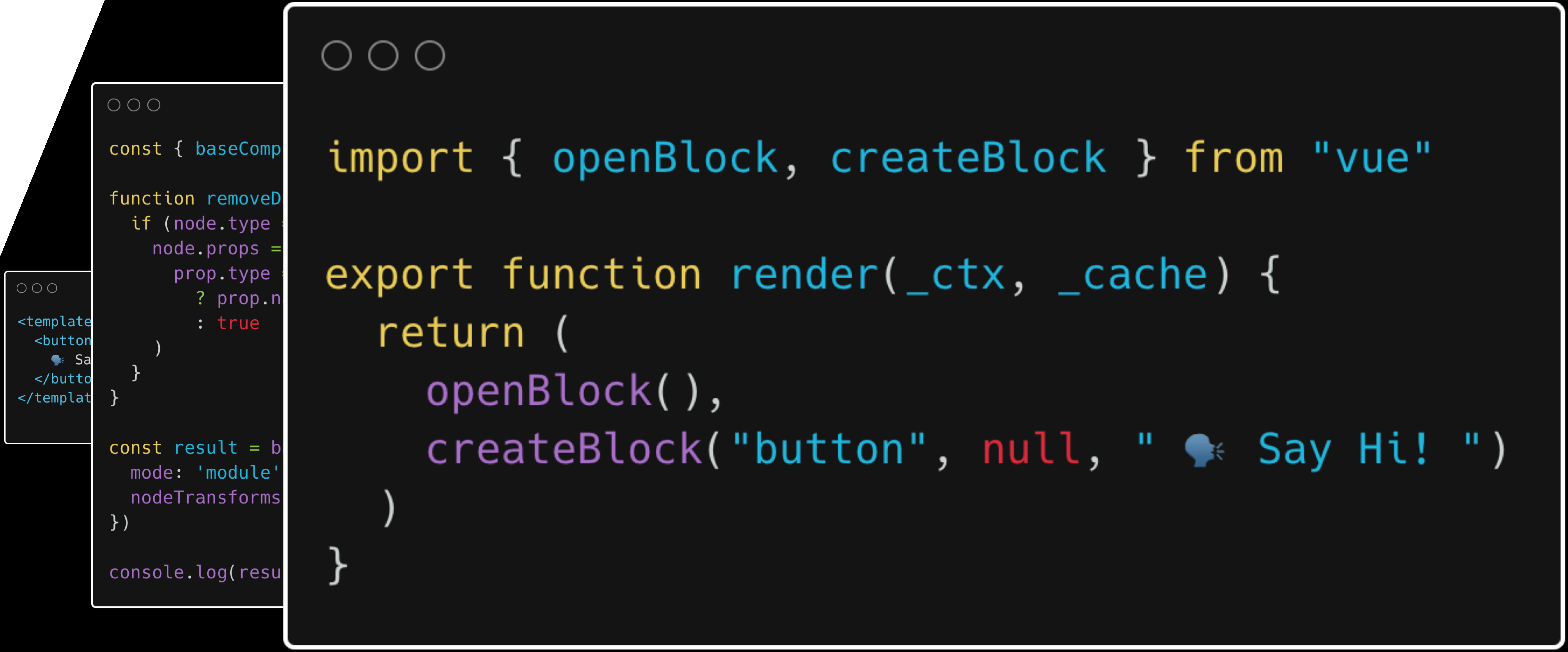
```
<template>
  <button data-test="butto
    🗣 Say Hi!
  </button>
</template>
```



```
const { baseCompile } = require('@vue/compiler-core')

function removeDataTestAttrs(node) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    node.props = node.props.filter(prop =>
      prop.type === 6 /* NodeTypes.ATTRIBUTE */
        ? prop.name !== 'data-test'
        : true
    )
  }
}

const result = baseCompile(template, {
  mode: 'module',
  nodeTransforms: [removeDataTestAttrs],
})

console.log(result.code)
```

Escape Hatch in Unit Test

`data-test="*"`



```
<template
  <button
    🗣 Sa
  </butto
</templat
```

```
const { baseComp

function removeD
  if (node.type
    node.props =
      prop.type
        ? prop.n
        : true
    )
  }
}

const result = b
  mode: 'module'
  nodeTransforms
})

console.log(resu
```

```
import { openBlock, createBlock } from "vue"

export function render(_ctx, _cache) {
  return (
    openBlock(),
    createBlock("button", null, " 🗣 Say Hi! ")
  )
}
```

Escape Hatch in Unit Test

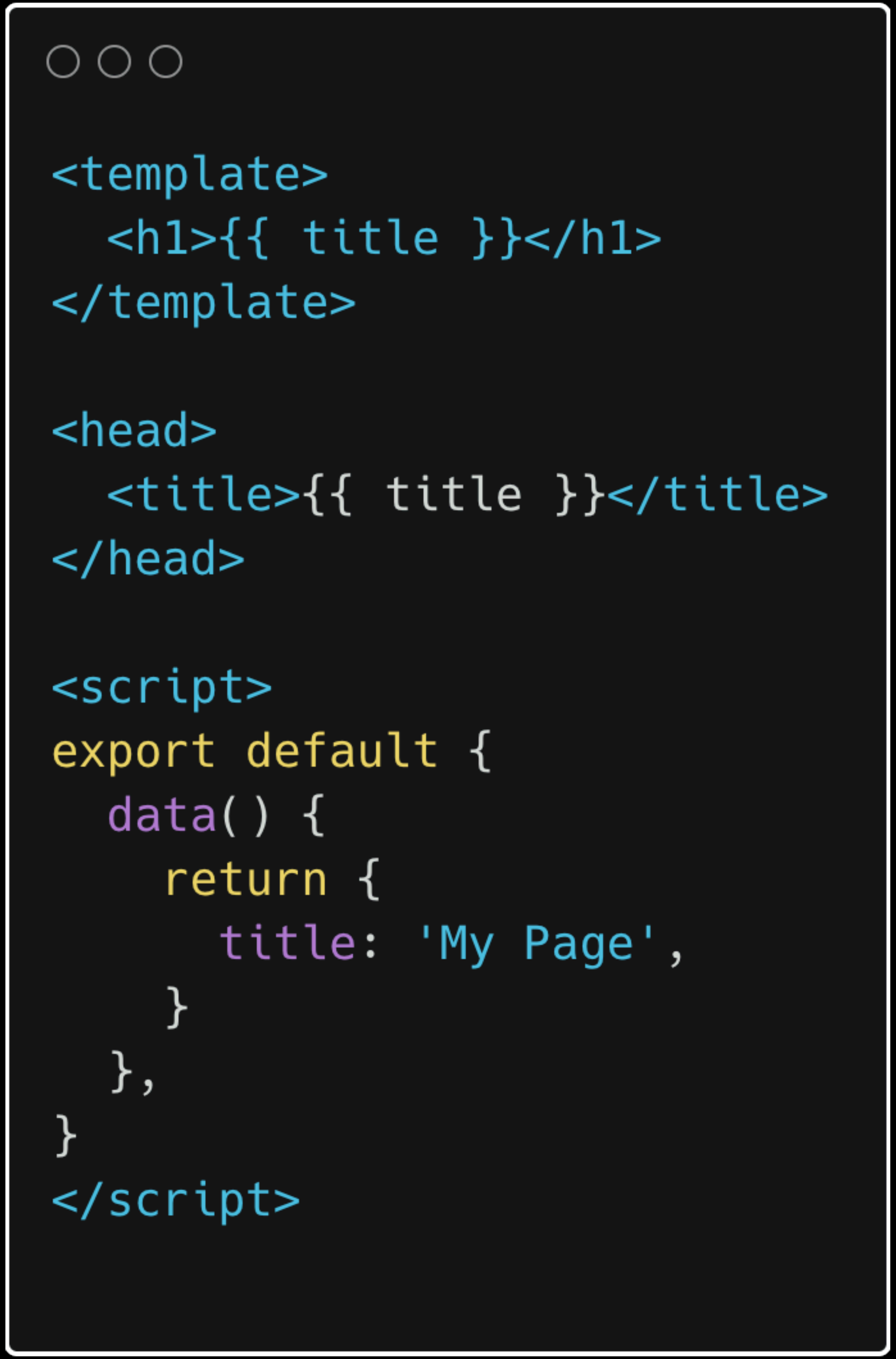
```
data-test="*"
```

Head Meta in SSR

```
<title>My Page</title>
```

Head Meta in SSR

`<title>My Page</title>`



```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title>
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

Head Meta in SSR

`<title>My Page</title>`

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title>
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

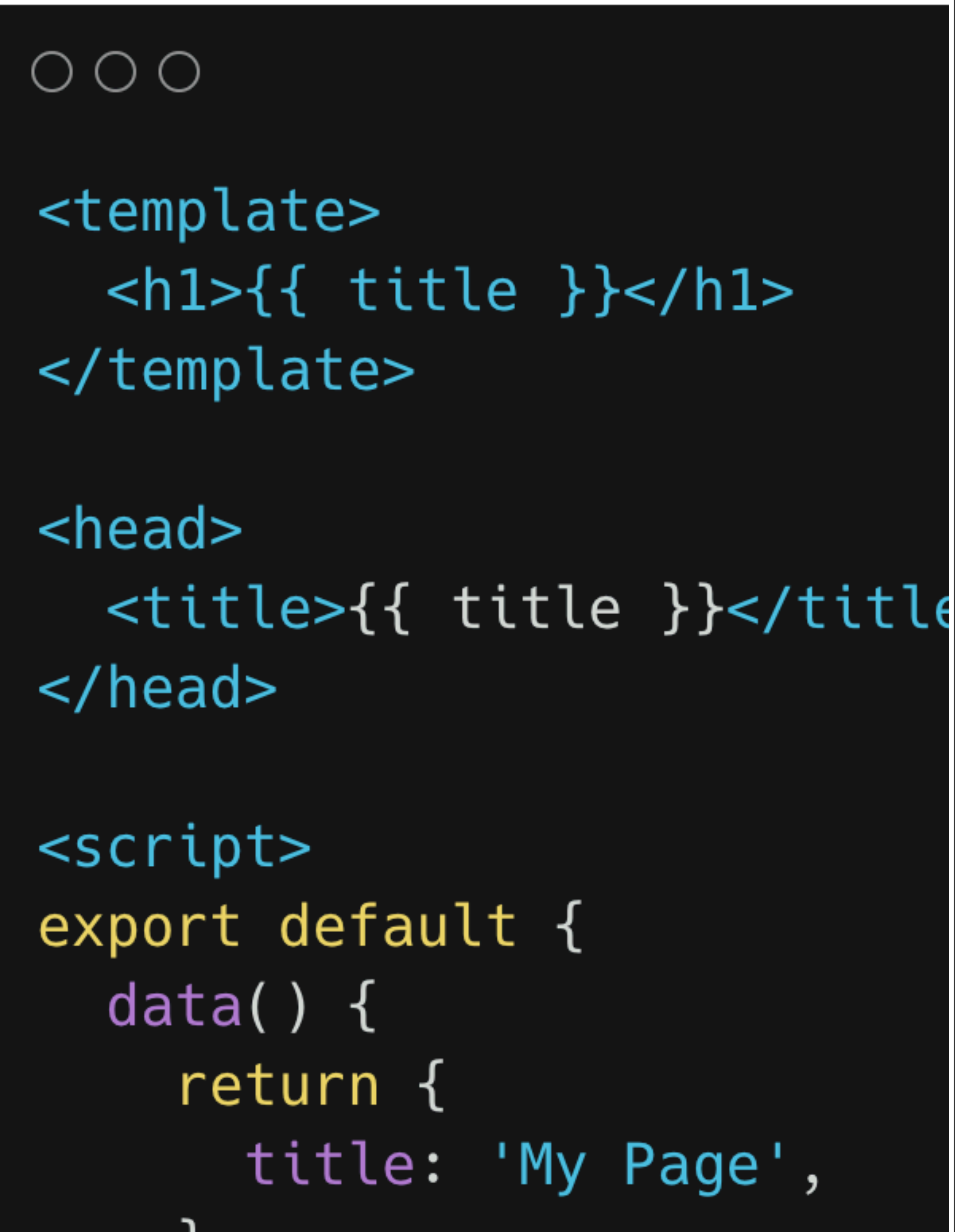
```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
  head() {
    return {
      title: this.title,
    }
  },
}
</script>
```

Head Meta in SSR

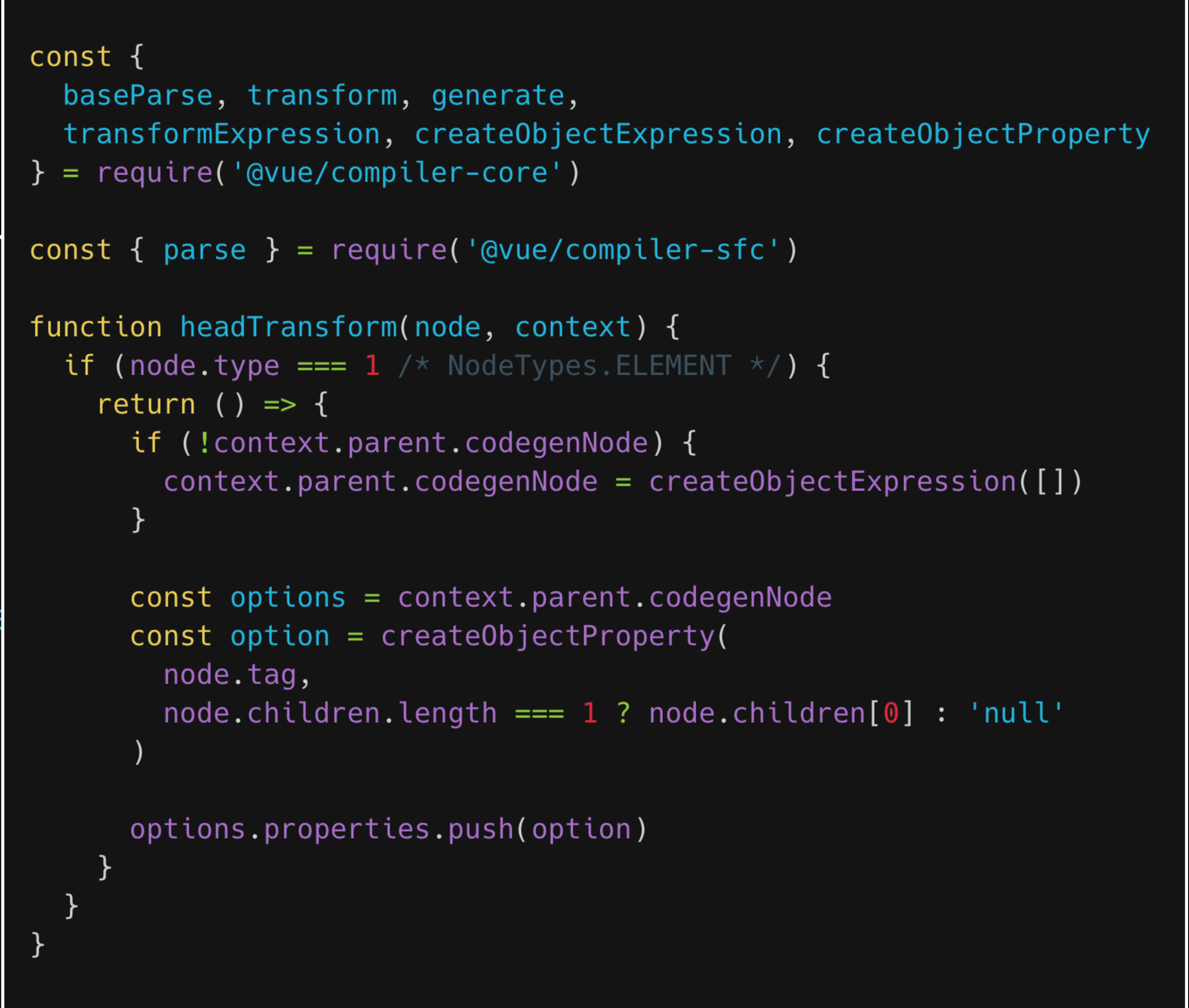
`<title>My Page</title>`

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

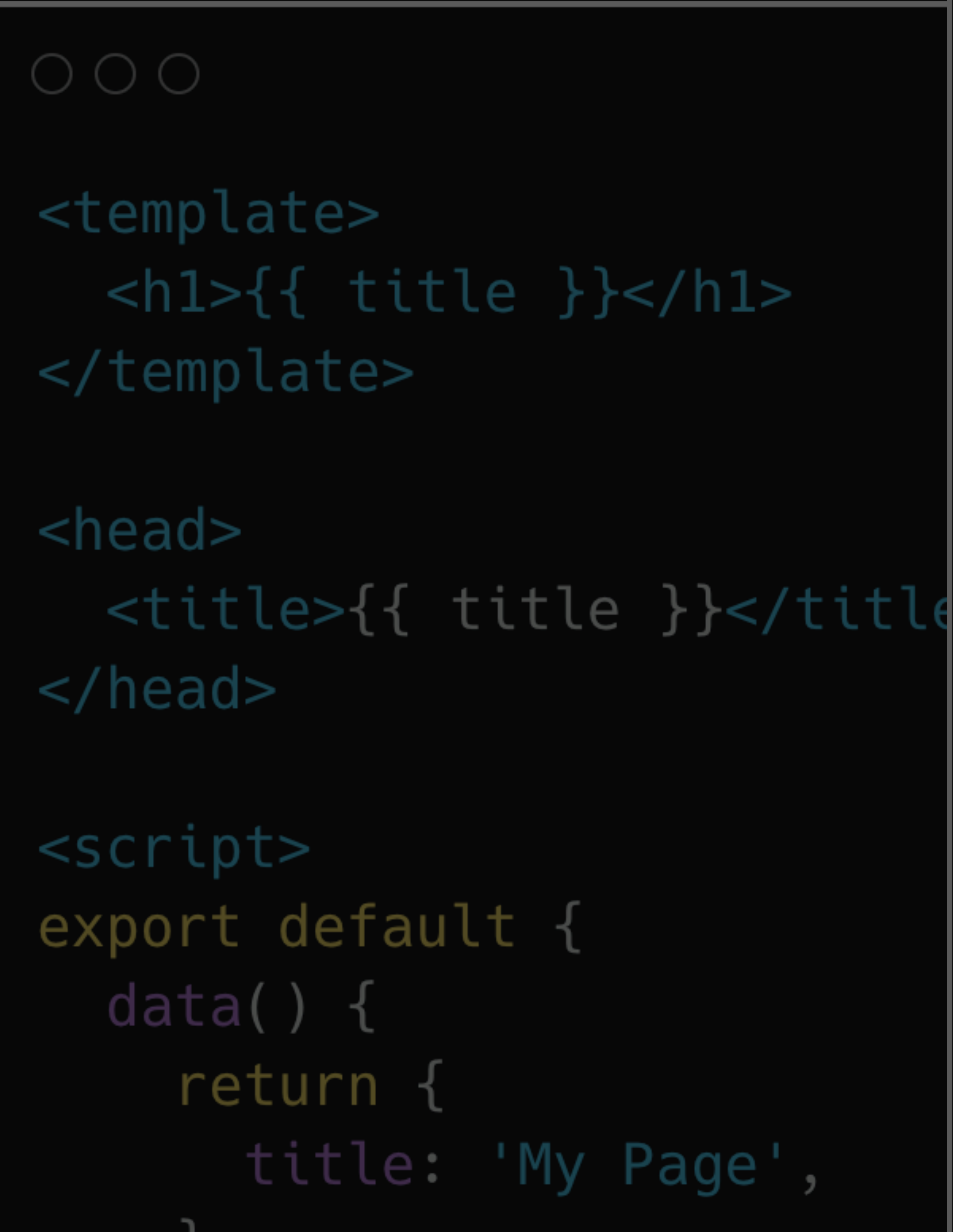
```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```



```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
```



```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```



```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

```js
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')
const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

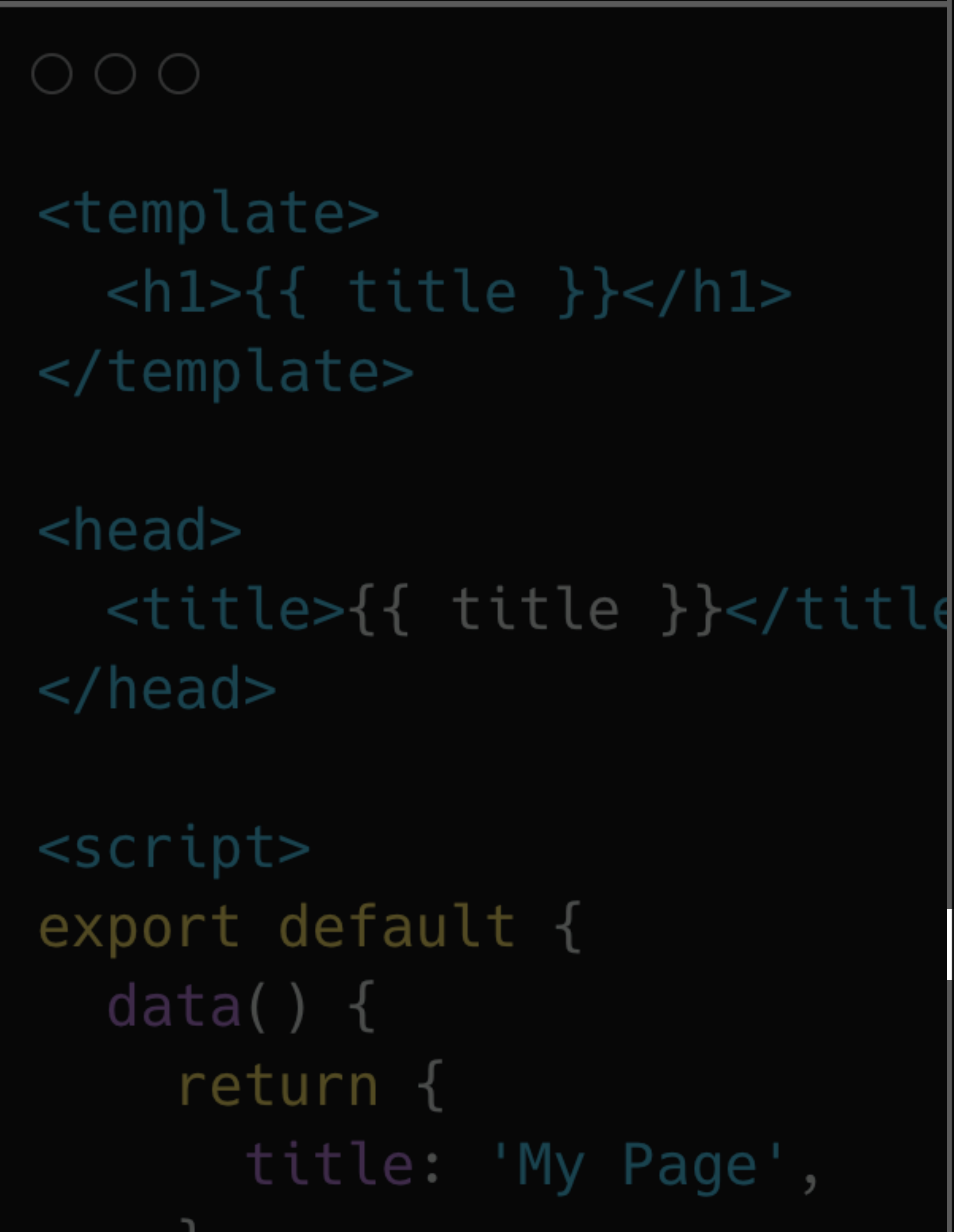
```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

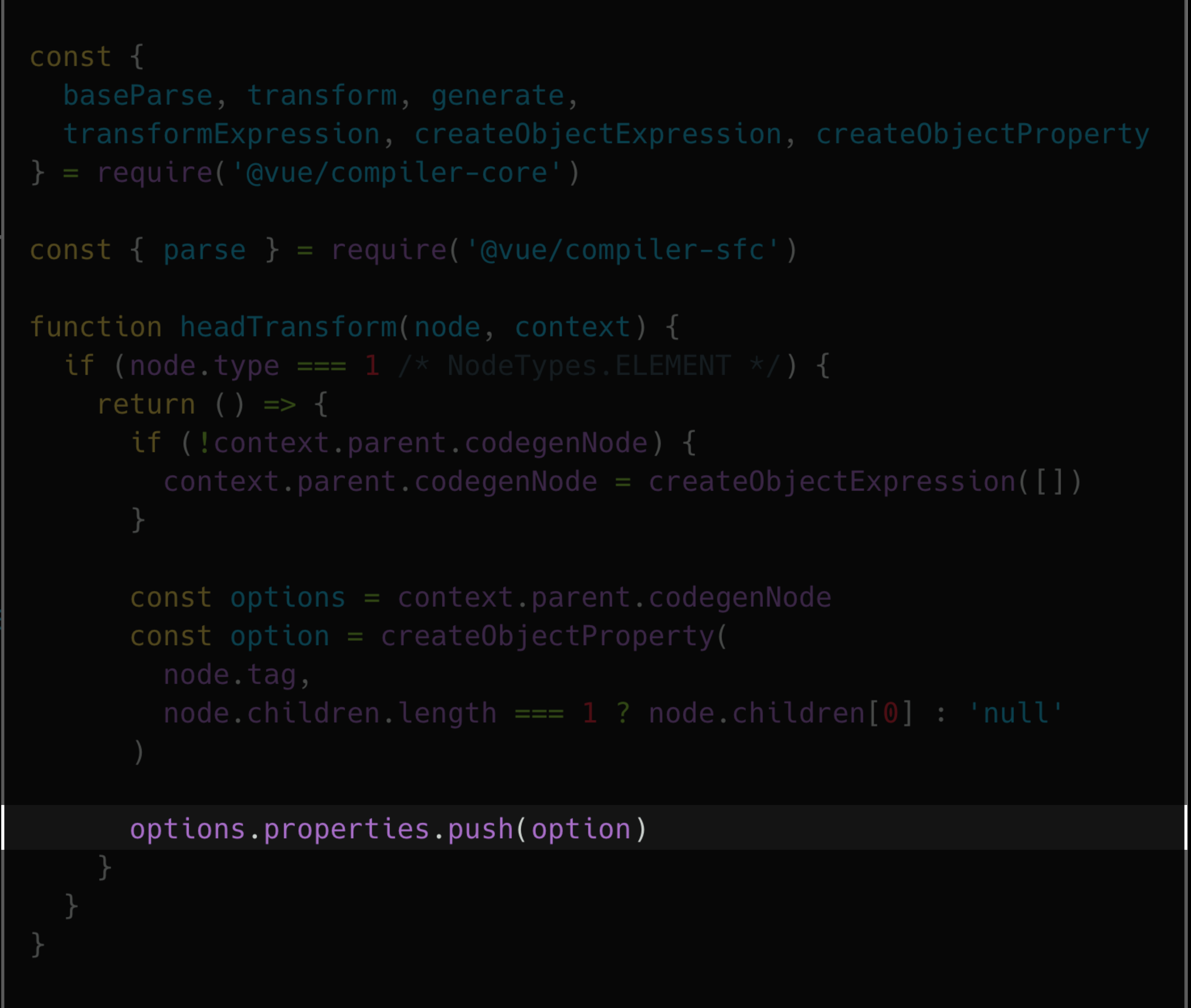
```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```



```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
```



```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}
```

```
<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

```
      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

```html
<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

```js
      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

```
<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

```
      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

```html
<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

```js
      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

```
<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

```
      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

Head Meta in SSR

`<title>My Page</title>`

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

Head Meta in SSR

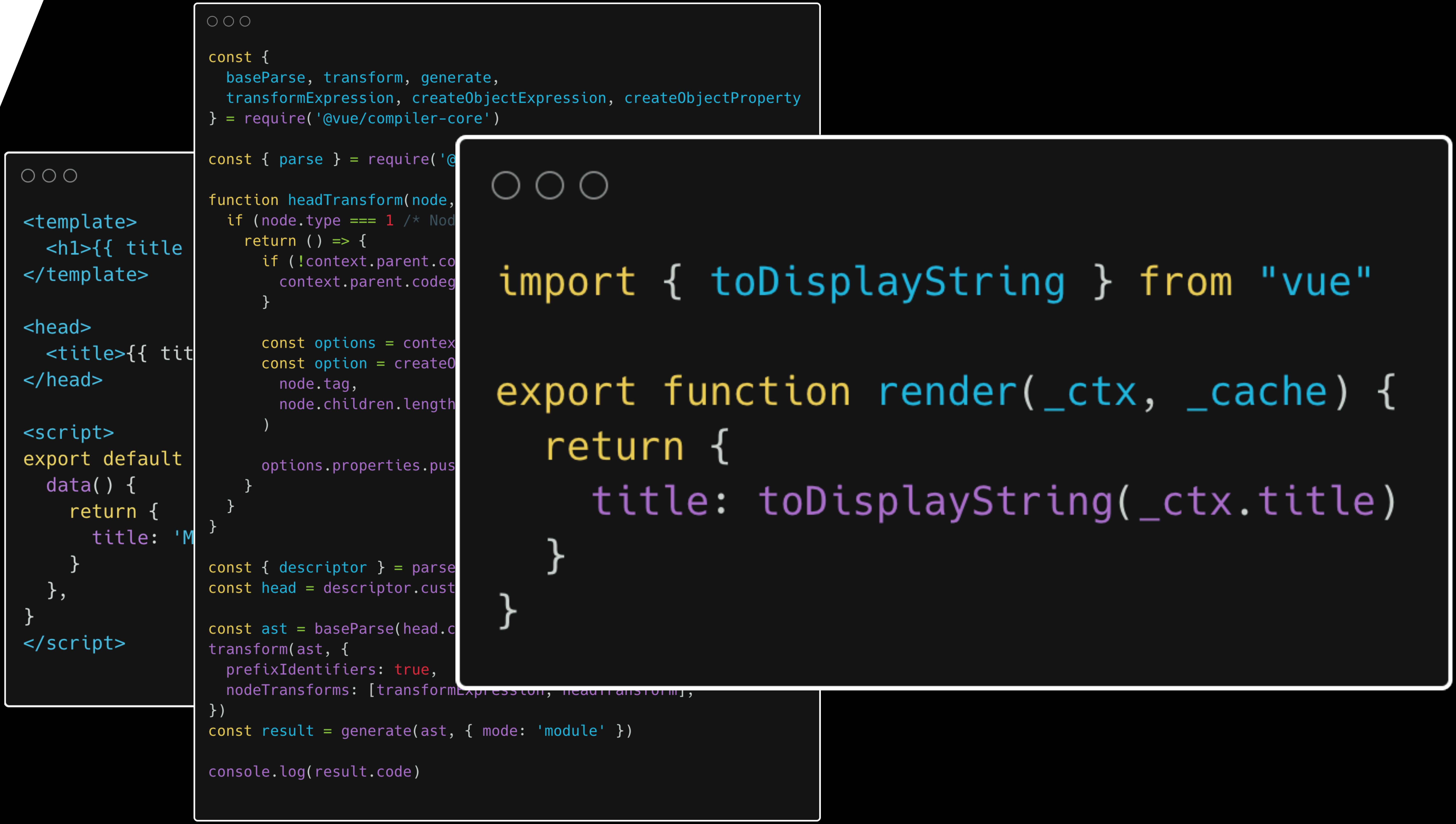
`<title>My Page</title>`

```
<template>
  <h1>{{ title }}</h1>
</template>

<head>
  <title>{{ title }}</title
</head>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
}
</script>
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@vue/compiler-sfc')

function headTransform(node, context) {
  if (node.type === 1 /* NodeTypes.ELEMENT */) {
    return () => {
      if (!context.parent.codegenNode) {
        context.parent.codegenNode = createObjectExpression([])
      }

      const options = context.parent.codegenNode
      const option = createObjectProperty(
        node.tag,
        node.children.length === 1 ? node.children[0] : 'null'
      )

      options.properties.push(option)
    }
  }
}

const { descriptor } = parse(SFC)
const head = descriptor.customBlocks[0]

const ast = baseParse(head.content)
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

Head Meta in SSR

`<title>My Page</title>`



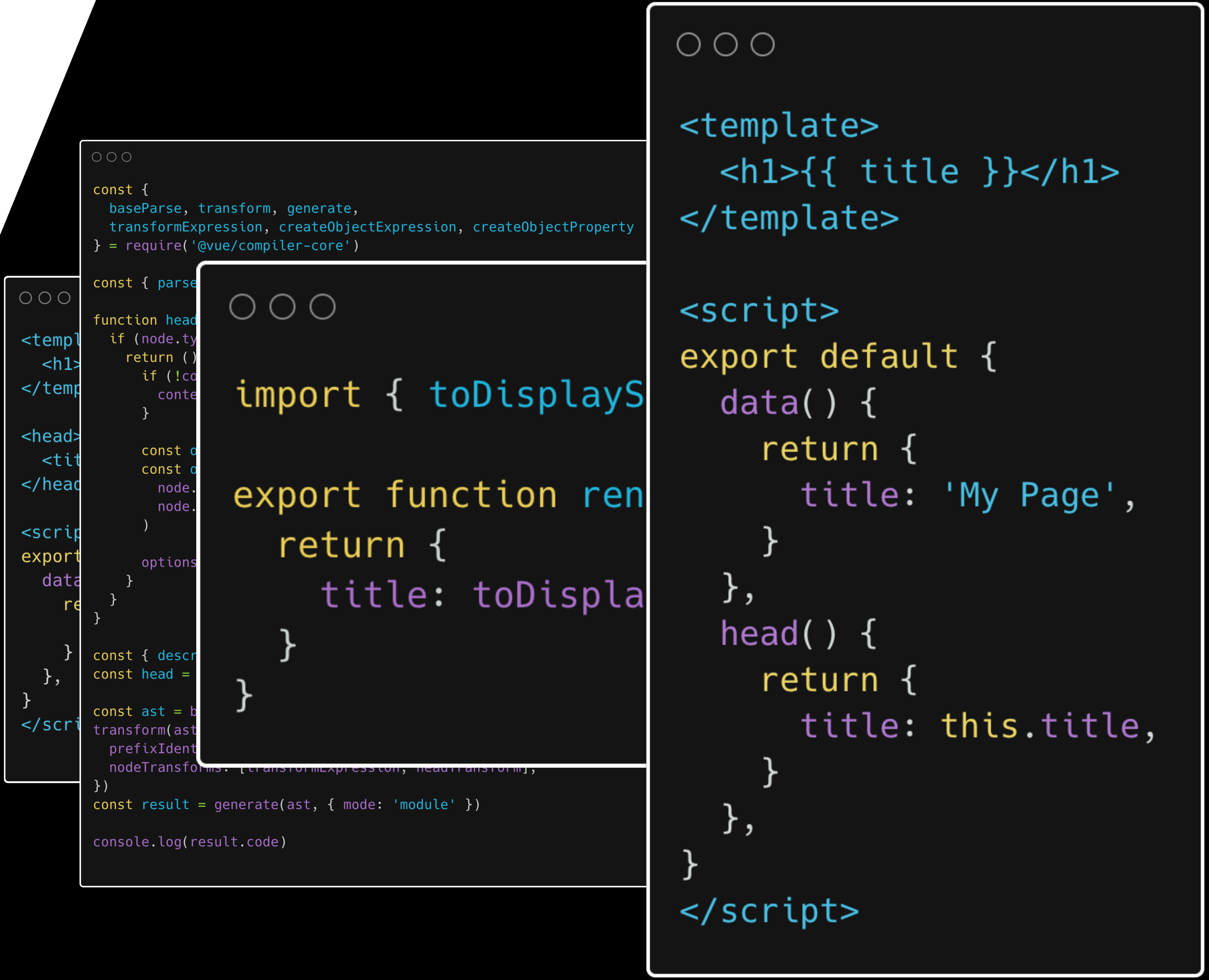
```
<template>
  <h1>{{ title
</template>

<head>
  <title>{{ tit
</head>

<script>
export default
  data() {
    return {
      title: 'M
    }
  },
}
</script>
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse } = require('@

function headTransform(node,
  if (node.type === 1 /* Nod
    return () => {
      if (!context.parent.co
        context.parent.codeg
      }

      const options = contex
      const option = createO
        node.tag,
        node.children.length
      )

      options.properties.pus
    }
  }
}

const { descriptor } = parse
const head = descriptor.cust

const ast = baseParse(head.c
transform(ast, {
  prefixIdentifiers: true,
  nodeTransforms: [transformExpression, headTransform],
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

```
import { toDisplayString } from "vue"

export function render(_ctx, _cache) {
  return {
    title: toDisplayString(_ctx.title)
  }
}
```

Head Meta in SSR

`<title>My Page</title>`



```
<templ
  <h1>
</temp

<head>
  <tit
</head

<scrip
export
  data
    re

    }
  },
}
</scri
```

```
const {
  baseParse, transform, generate,
  transformExpression, createObjectExpression, createObjectProperty
} = require('@vue/compiler-core')

const { parse

function head
  if (node.ty
    return ()
      if (!co
        conte
      }

      const c
      const c
        node.
        node.
      )

      options
    }
  }
}

const { descr
const head =

const ast = b
transform(ast
  prefixIdent
  nodeTransfo
})
const result = generate(ast, { mode: 'module' })

console.log(result.code)
```

```
import { toDisplayS

export function ren
  return {
    title: toDispla
  }
}
```

```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
export default {
  data() {
    return {
      title: 'My Page',
    }
  },
  head() {
    return {
      title: this.title,
    }
  },
}
</script>
```

Head Meta in SSR

```
<title>My Page</title>
```

How does the compiler work?

How does the compiler work?

How you can customise it?

@znck0