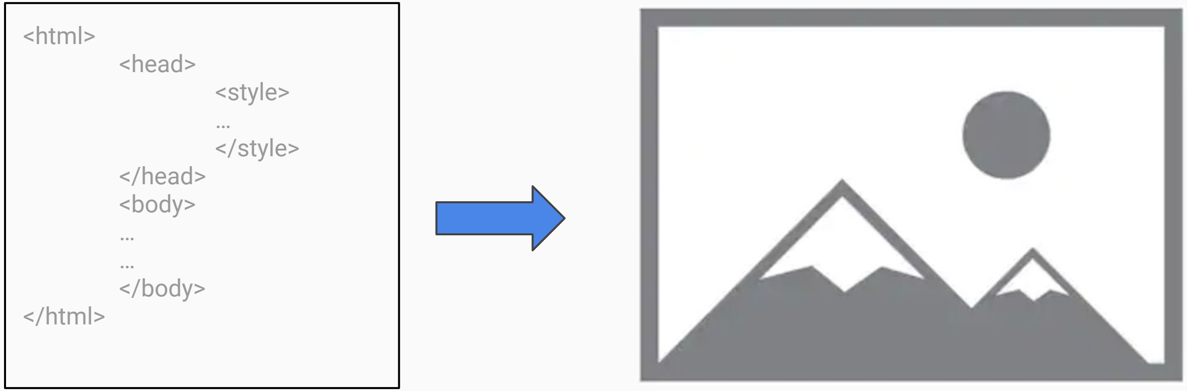
Toy Web Browser

A simple model to demonstrate how WEB browser renders HTML.

Objective of Toy Browser

Turn an server responded HTML string into an image.



```
<html>
          <head>
                    <style>
                    …
                    </style>
          </head>
          <body>
          …
          …
          </body>
</html>
```

State Machine

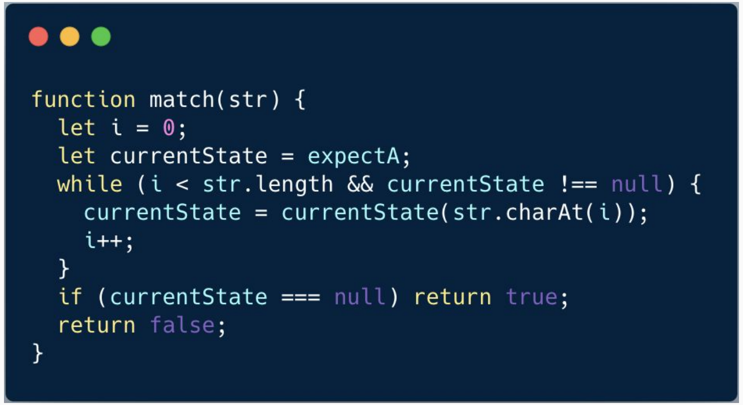
Example - String searching

Let’s say we want to find the occurences of [string B] in [string A].
A: **ababcdABCD**
B: **ABCD**

Let’s do it with a **state machine**.

Example - String searching

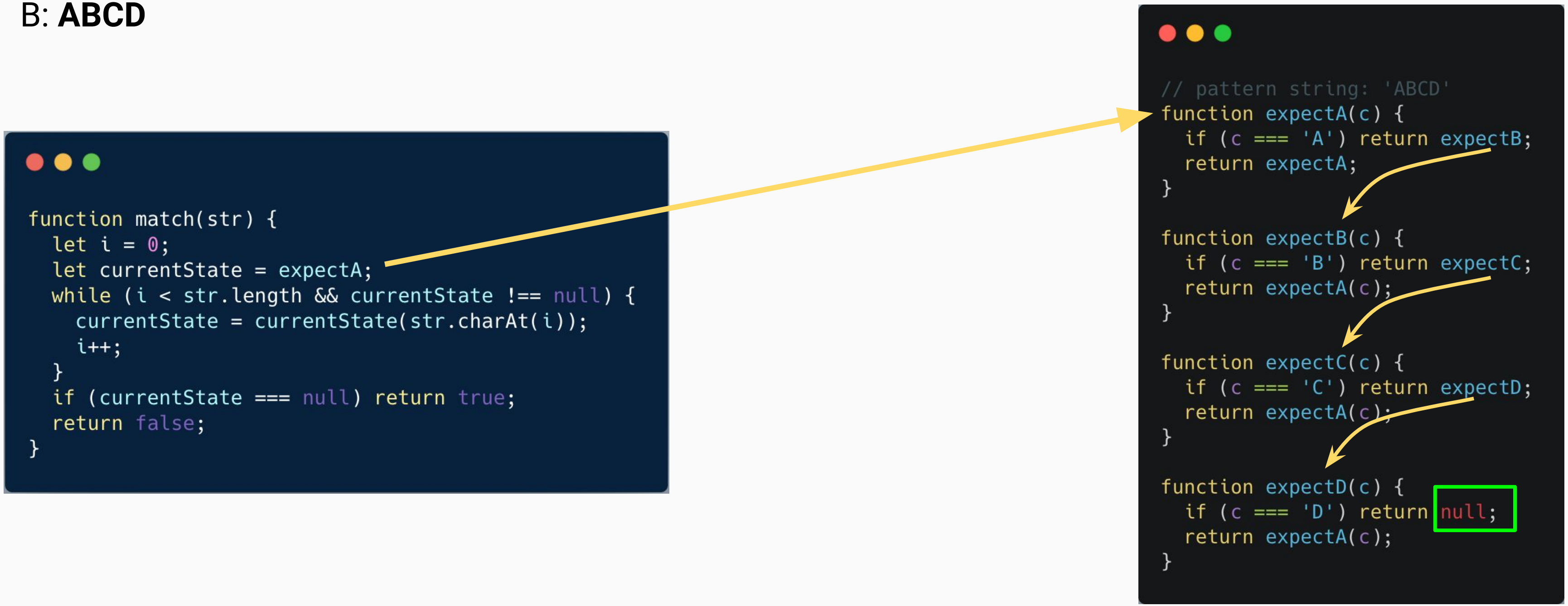
Find the occurences of [string B] in [string A].
A: **ababcdABCD**
B: **ABCD**



```
function match(str) {
  let i = 0;
  let currentState = expectA;
  while (i < str.length && currentState !== null) {
    currentState = currentState(str.charAt(i));
    i++;
  }
  if (currentState === null) return true;
  return false;
}
```

Example - String searching

Find the occurences of [string B] in [string A].
A: **ababcdABCD**
B: **ABCD**



```
function match(str) {
  let i = 0;
  let currentState = expectA;
  while (i < str.length && currentState !== null) {
    currentState = currentState(str.charAt(i));
    i++;
  }
  if (currentState === null) return true;
  return false;
}
```

```
// pattern string: 'ABCD'
function expectA(c) {
  if (c === 'A') return expectB;
  return expectA;
}

function expectB(c) {
  if (c === 'B') return expectC;
  return expectA(c);
}

function expectC(c) {
  if (c === 'C') return expectD;
  return expectA(c);
}

function expectD(c) {
  if (c === 'D') return null;
  return expectA(c);
}
```

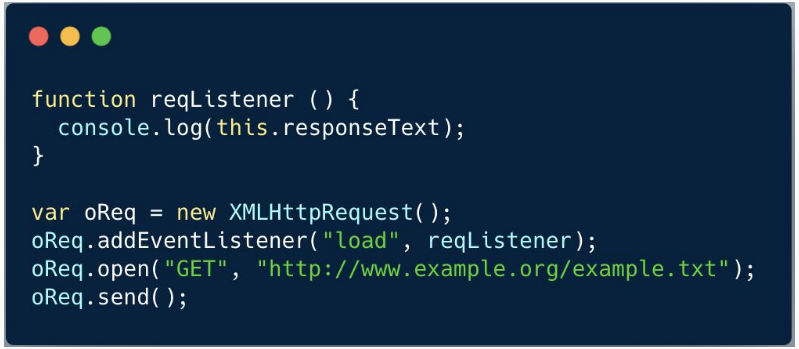
Let’s start to create our Toy Browser

1. Parse HTTP response
2. HTML tokenization
3. CSS computing
4. Layout
5. Render

Parse HTTP response

Modern browser

Provided API like XMLHttpRequest, which can be called to send HTTP request.
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

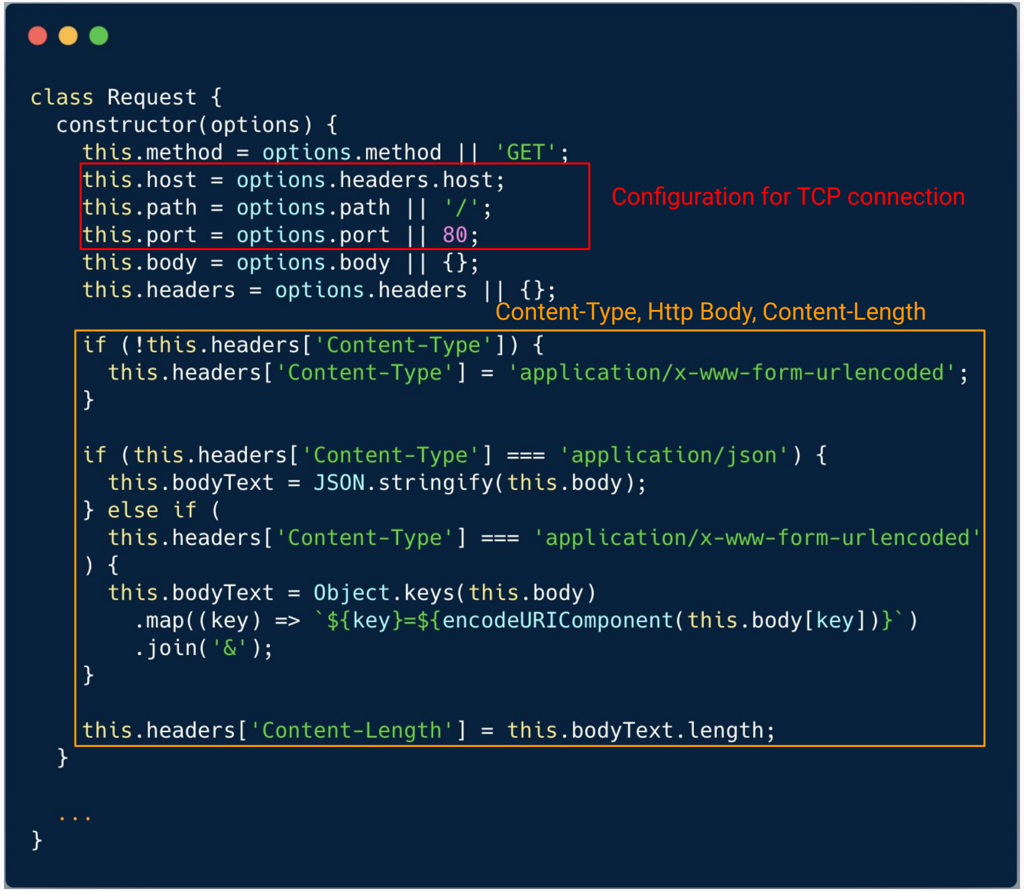


```
function reqListener () {
  console.log(this.responseText);
}

var oReq = new XMLHttpRequest();
oReq.addEventListener("load", reqListener);
oReq.open("GET", "http://www.example.org/example.txt");
oReq.send();
```

Toy browser

Let create a Request Object to send HTTP request.

HTTP Request - Constructor

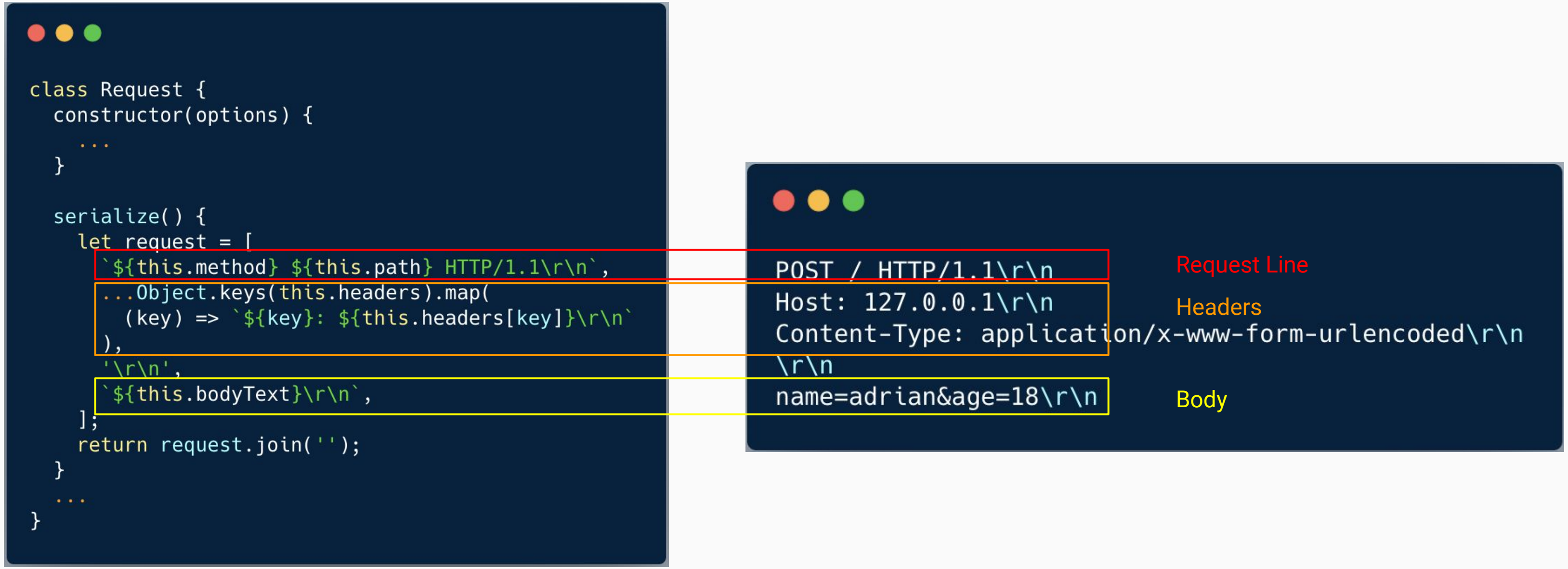


```
class Request {
  constructor(options) {
    this.method = options.method || 'GET';
    this.host = options.headers.host;
    this.path = options.path || '/';
    this.port = options.port || 80;
    this.body = options.body || {};
    this.headers = options.headers || {};

    if (!this.headers['Content-Type']) {
      this.headers['Content-Type'] = 'application/x-www-form-urlencoded';
    }

    if (this.headers['Content-Type'] === 'application/json') {
      this.bodyText = JSON.stringify(this.body);
    } else if (
      this.headers['Content-Type'] === 'application/x-www-form-urlencoded'
    ) {
      this.bodyText = Object.keys(this.body)
        .map((key) => `${key}=${encodeURIComponent(this.body[key])}`)
        .join('&');
    }

    this.headers['Content-Length'] = this.bodyText.length;
  }

  ...
}
```

Configuration for TCP connection

Content-Type, Http Body, Content-Length

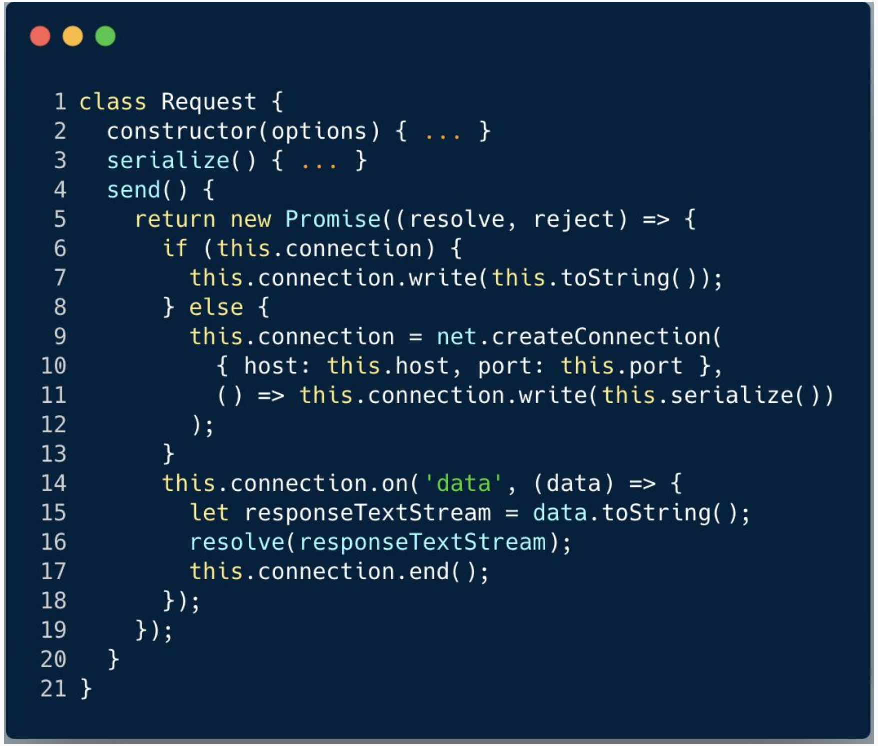
HTTP Request - Serialize( )



```
class Request {
  constructor(options) {
    ...
  }

  serialize() {
    let request = [
      `${this.method} ${this.path} HTTP/1.1\r\n`,
      ...Object.keys(this.headers).map(
        (key) => `${key}: ${this.headers[key]}\r\n`
      ),
      '\r\n',
      `${this.bodyText}\r\n`,
    ];
    return request.join('');
  }
  ...
}
```

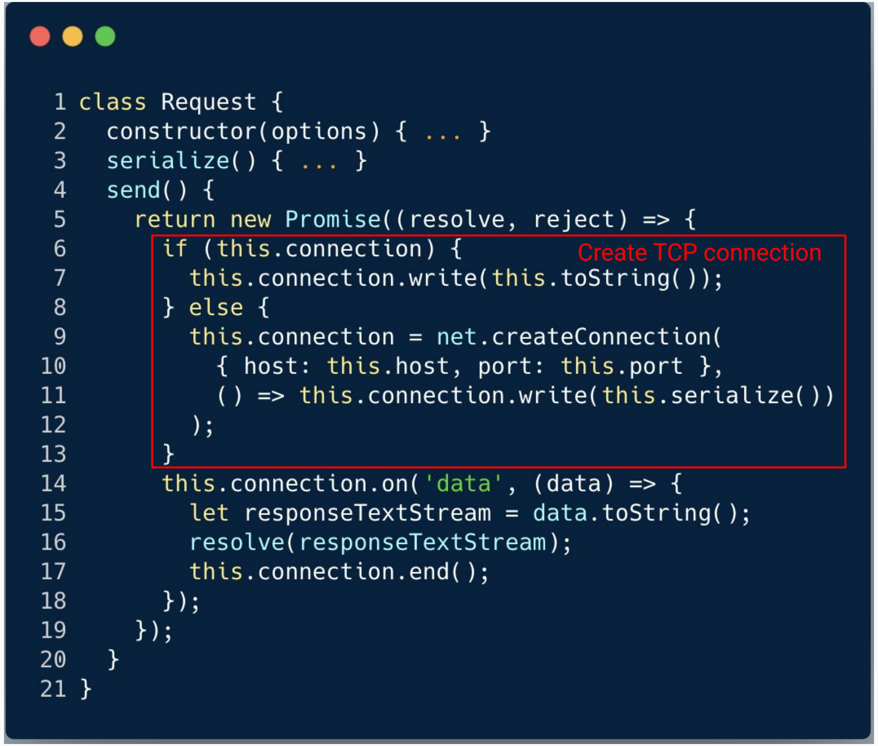
```
POST / HTTP/1.1\r\n
Host: 127.0.0.1\r\n
Content-Type: application/x-www-form-urlencoded\r\n
\r\n
name=adrian&age=18\r\n
```

HTTP Request - send( )



```
class Request {
  constructor(options) { ... }
  serialize() { ... }
  send() {
    return new Promise((resolve, reject) => {
      if (this.connection) {
        this.connection.write(this.toString());
      } else {
        this.connection = net.createConnection(
          { host: this.host, port: this.port },
          () => this.connection.write(this.serialize())
        );
      }
      this.connection.on('data', (data) => {
        let responseTextStream = data.toString();
        resolve(responseTextStream);
        this.connection.end();
      });
    });
  }
}
```

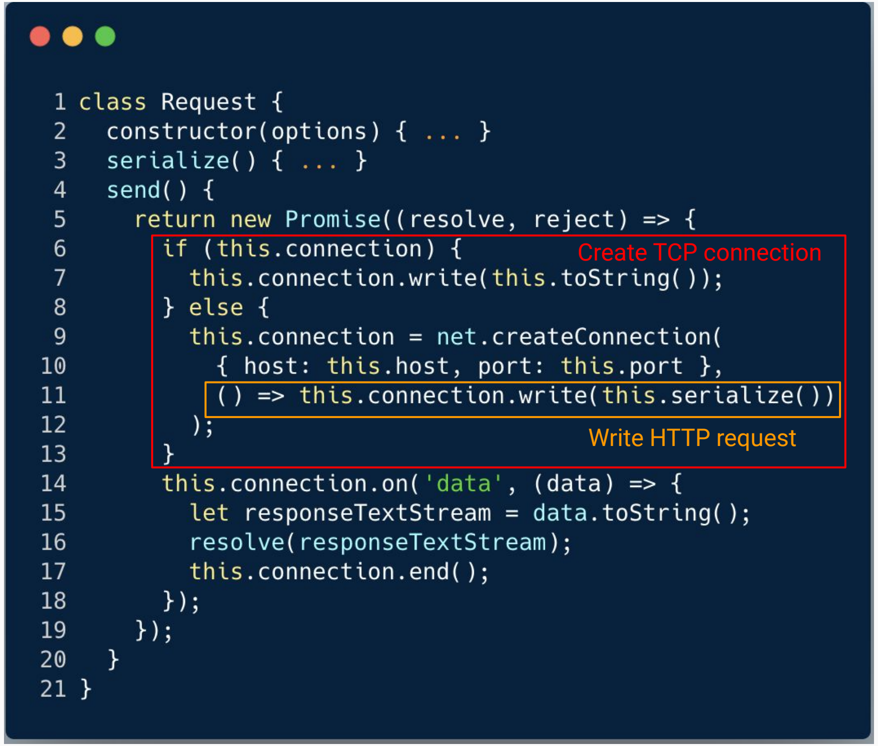
HTTP Request - send( )



```
class Request {
  constructor(options) { ... }
  serialize() { ... }
  send() {
    return new Promise((resolve, reject) => {
      if (this.connection) {
        this.connection.write(this.toString());
      } else {
        this.connection = net.createConnection(
          { host: this.host, port: this.port },
          () => this.connection.write(this.serialize())
        );
      }
      this.connection.on('data', (data) => {
        let responseTextStream = data.toString();
        resolve(responseTextStream);
        this.connection.end();
      });
    });
  }
}
```

Create TCP connection

HTTP Request - send( )



```
class Request {
  constructor(options) { ... }
  serialize() { ... }
  send() {
    return new Promise((resolve, reject) => {
      if (this.connection) {
        this.connection.write(this.toString());
      } else {
        this.connection = net.createConnection(
          { host: this.host, port: this.port },
          () => this.connection.write(this.serialize())
        );
      }
      this.connection.on('data', (data) => {
        let responseTextStream = data.toString();
        resolve(responseTextStream);
        this.connection.end();
      });
    });
  }
}
```

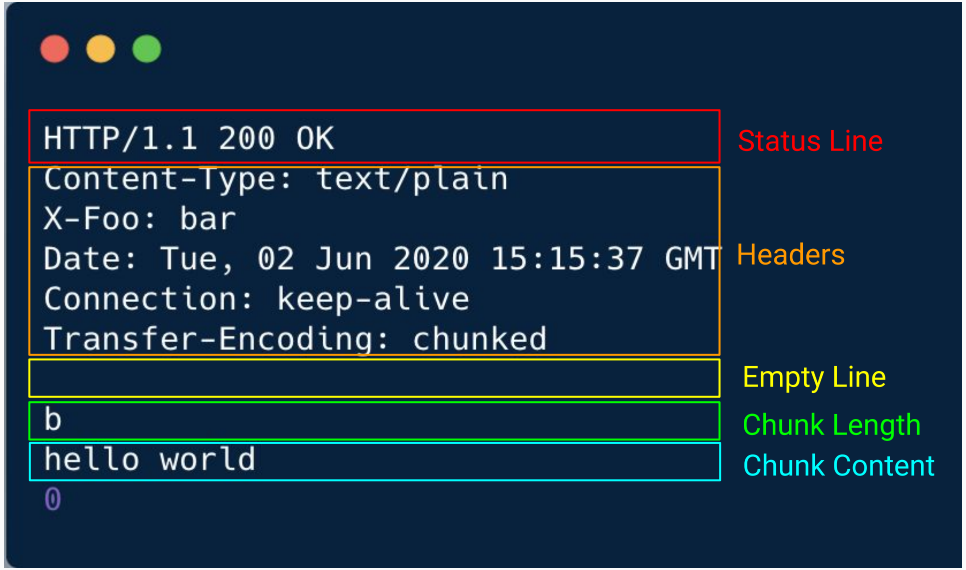
Create TCP connection

Write HTTP request

DEMO

Git branch: 1.1-parse-http-response

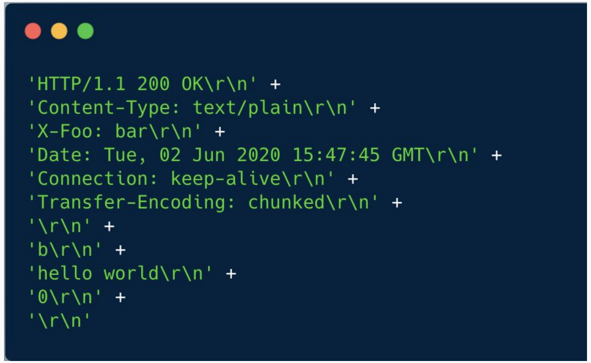
Object to Send HTTP Request - Response Text Stream



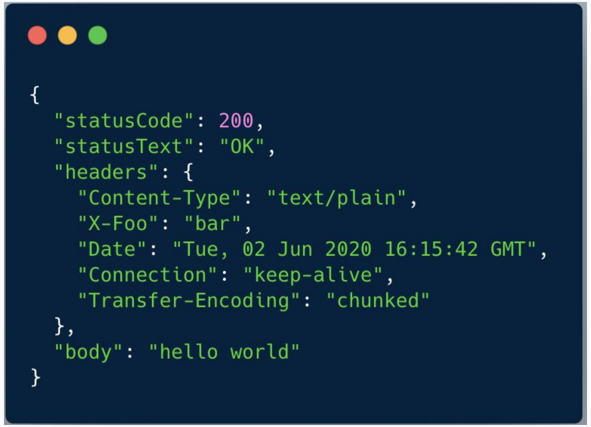
```
HTTP/1.1 200 OK
Content-Type: text/plain
X-Foo: bar
Date: Tue, 02 Jun 2020 15:15:37 GMT
Connection: keep-alive
Transfer-Encoding: chunked

b
hello world
0
```

Response parser

Just text makes less meaning.

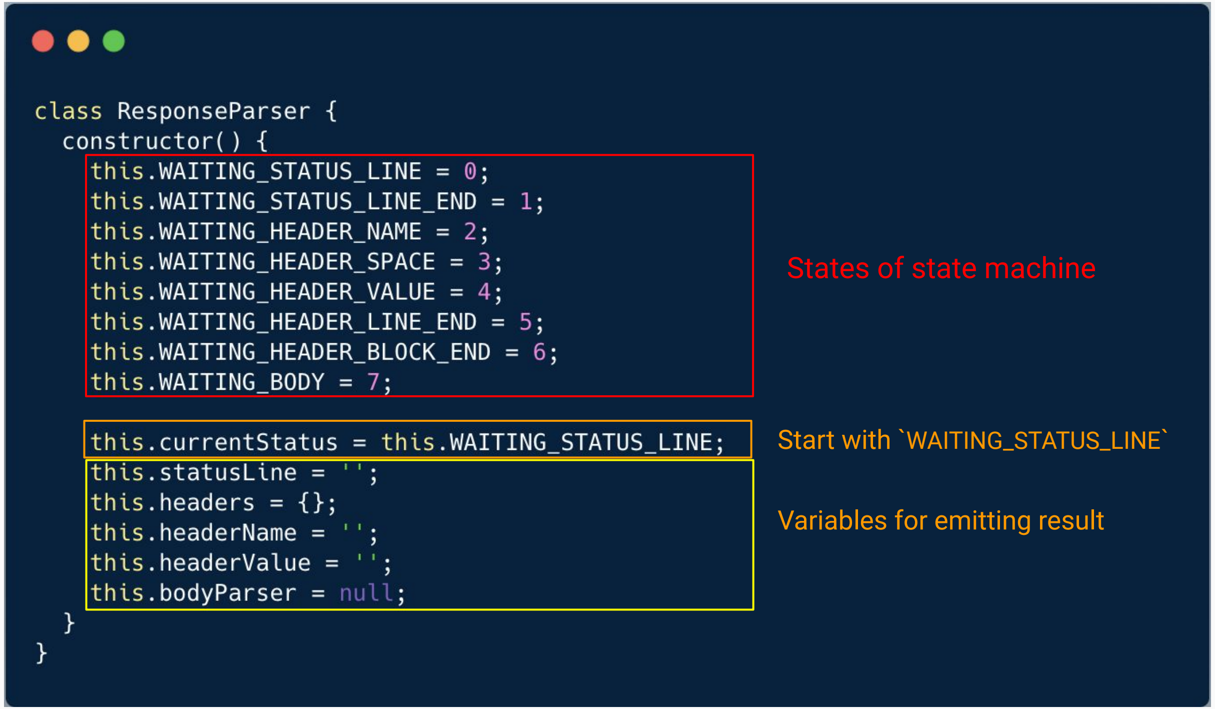


```
'HTTP/1.1 200 OK\r\n' +
'Content-Type: text/plain\r\n' +
'X-Foo: bar\r\n' +
'Date: Tue, 02 Jun 2020 15:47:45 GMT\r\n' +
'Connection: keep-alive\r\n' +
'Transfer-Encoding: chunked\r\n' +
'\r\n' +
'b\r\n' +
'hello world\r\n' +
'0\r\n' +
'\r\n'
```



```
{
  "statusCode": 200,
  "statusText": "OK",
  "headers": {
    "Content-Type": "text/plain",
    "X-Foo": "bar",
    "Date": "Tue, 02 Jun 2020 16:15:42 GMT",
    "Connection": "keep-alive",
    "Transfer-Encoding": "chunked"
  },
  "body": "hello world"
}
```

Object to Send HTTP Request - Parse Response



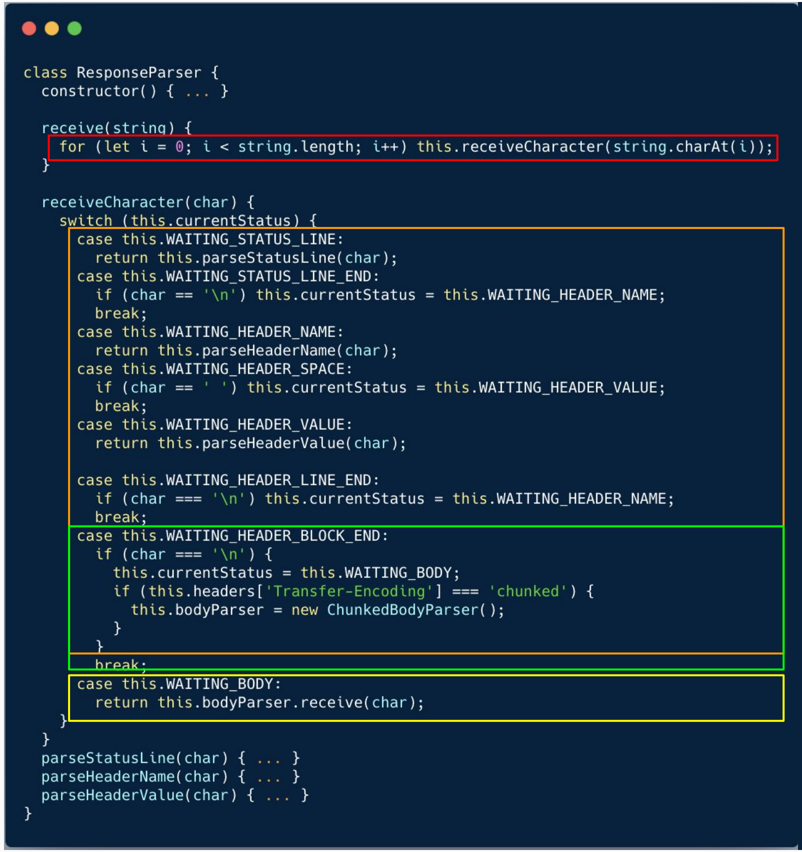
```
class ResponseParser {
  constructor() {
    this.WAITING_STATUS_LINE = 0;
    this.WAITING_STATUS_LINE_END = 1;
    this.WAITING_HEADER_NAME = 2;
    this.WAITING_HEADER_SPACE = 3;
    this.WAITING_HEADER_VALUE = 4;
    this.WAITING_HEADER_LINE_END = 5;
    this.WAITING_HEADER_BLOCK_END = 6;
    this.WAITING_BODY = 7;

    this.currentStatus = this.WAITING_STATUS_LINE;
    this.statusLine = '';
    this.headers = {};
    this.headerName = '';
    this.headerValue = '';
    this.bodyParser = null;
  }
}
```

States of state machine

Start with `WAITING_STATUS_LINE`

Variables for emitting result

Object to Send HTTP Request - Parse Response



```
class ResponseParser {
  constructor() { ... }

  receive(string) {
    for (let i = 0; i < string.length; i++) this.receiveCharacter(string.charAt(i));
  }

  receiveCharacter(char) {
    switch (this.currentStatus) {
      case this.WAITING_STATUS_LINE:
        return this.parseStatusLine(char);
      case this.WAITING_STATUS_LINE_END:
        if (char == '\n') this.currentStatus = this.WAITING_HEADER_NAME;
        break;
      case this.WAITING_HEADER_NAME:
        return this.parseHeaderName(char);
      case this.WAITING_HEADER_SPACE:
        if (char == ' ') this.currentStatus = this.WAITING_HEADER_VALUE;
        break;
      case this.WAITING_HEADER_VALUE:
        return this.parseHeaderValue(char);

      case this.WAITING_HEADER_LINE_END:
        if (char === '\n') this.currentStatus = this.WAITING_HEADER_NAME;
        break;
      case this.WAITING_HEADER_BLOCK_END:
        if (char === '\n') {
          this.currentStatus = this.WAITING_BODY;
          if (this.headers['Transfer-Encoding'] === 'chunked') {
            this.bodyParser = new ChunkedBodyParser();
          }
        }
        break;
      case this.WAITING_BODY:
        return this.bodyParser.receive(char);
    }
  }
  parseStatusLine(char) { ... }
  parseHeaderName(char) { ... }
  parseHeaderValue(char) { ... }
}
```

Pass each character `receiveCharacter`

Consume char based on state

chunked
gzip
compress
deflate

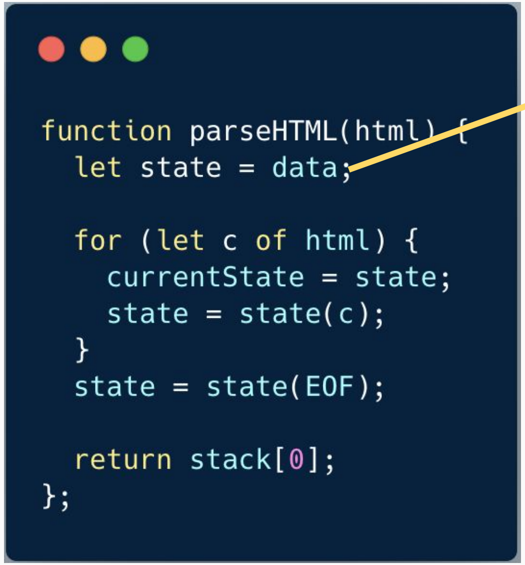
Body parser

DEMO

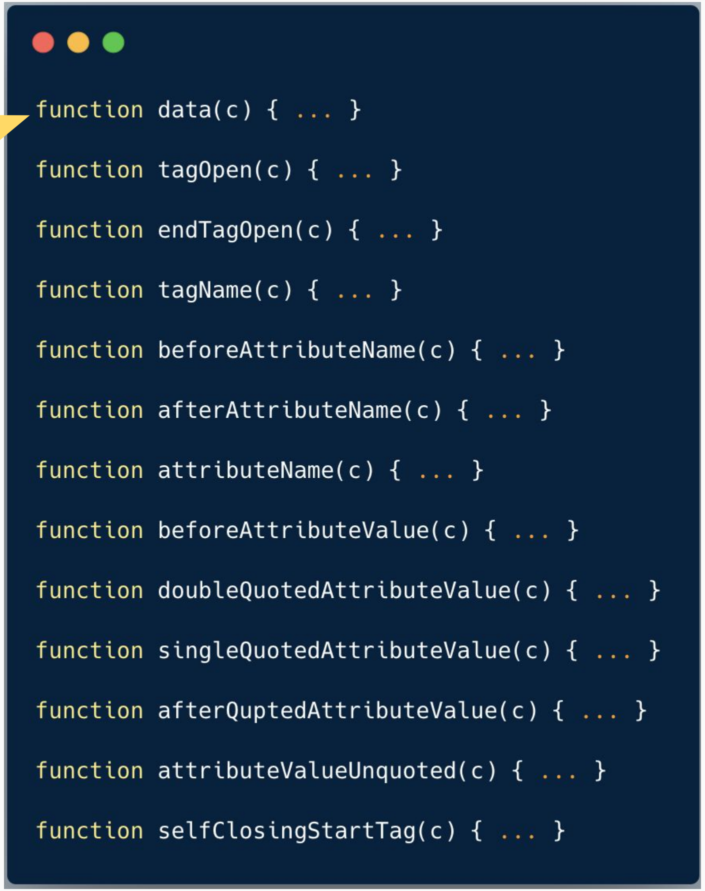
Git branch: 1-parse-http-response

1. Parse HTTP response
2. HTML tokenization
3. CSS computing
4. Layout
5. Render

HTML tokenization



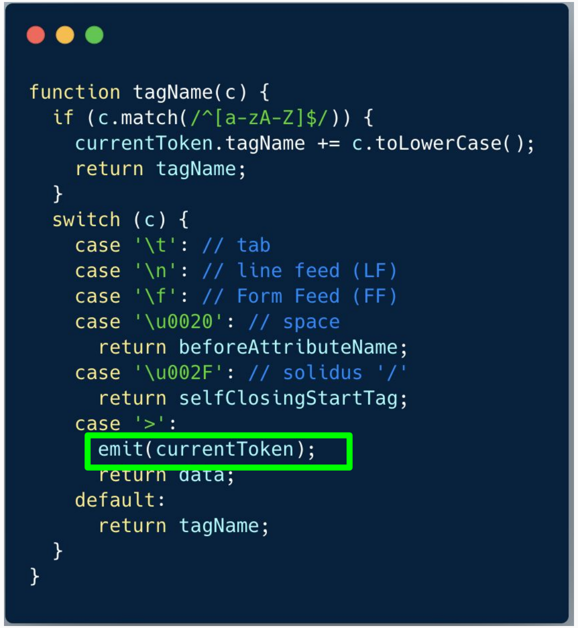
```
function parseHTML(html) {
  let state = data;

  for (let c of html) {
    currentState = state;
    state = state(c);
  }
  state = state(EOF);

  return stack[0];
};
```



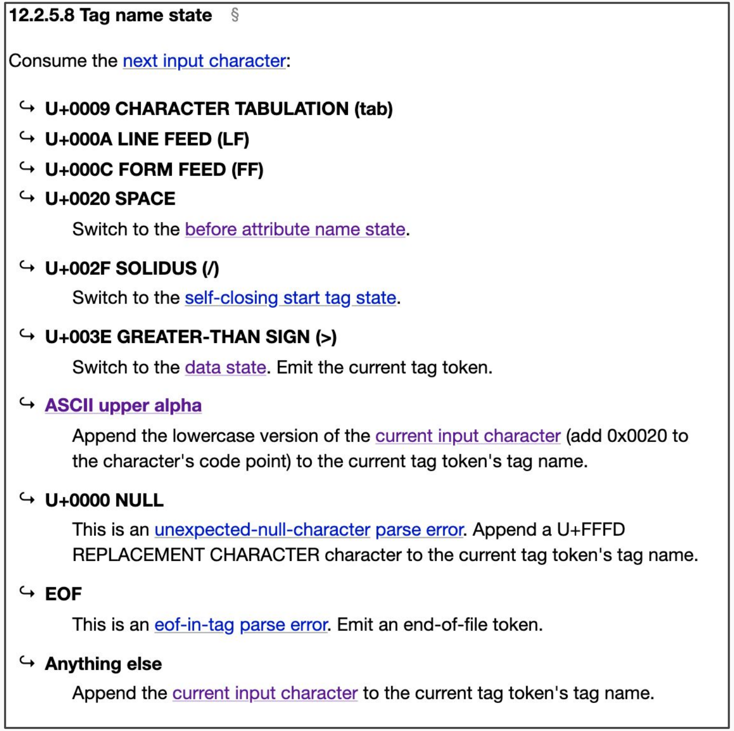
```
function data(c) { ... }

function tagOpen(c) { ... }

function endTagOpen(c) { ... }

function tagName(c) { ... }

function beforeAttributeName(c) { ... }

function afterAttributeName(c) { ... }

function attributeName(c) { ... }

function beforeAttributeValue(c) { ... }

function doubleQuotedAttributeValue(c) { ... }

function singleQuotedAttributeValue(c) { ... }

function afterQuptedAttributeValue(c) { ... }

function attributeValueUnquoted(c) { ... }

function selfClosingStartTag(c) { ... }
```

How does the state function work?

HTML tokenization



```
function tagName(c) {
  if (c.match(/^[a-zA-Z]$/)) {
    currentToken.tagName += c.toLowerCase();
    return tagName;
  }
  switch (c) {
    case '\t': // tab
    case '\n': // line feed (LF)
    case '\f': // Form Feed (FF)
    case '\u0020': // space
      return beforeAttributeName;
    case '\u002F': // solidus '/'
      return selfClosingStartTag;
    case '>':
      emit(currentToken);
      return data;
    default:
      return tagName;
  }
}
```



12.2.5.8 Tag name state §

Consume the next input character:

↪ **U+0009 CHARACTER TABULATION (tab)**

↪ **U+000A LINE FEED (LF)**

↪ **U+000C FORM FEED (FF)**

↪ **U+0020 SPACE**

Switch to the before attribute name state.

↪ **U+002F SOLIDUS (/)**

Switch to the self-closing start tag state.

↪ **U+003E GREATER-THAN SIGN (>)**

Switch to the data state. Emit the current tag token.

↪ **ASCII upper alpha**

Append the lowercase version of the current input character (add 0x0020 to the character's code point) to the current tag token's tag name.

↪ **U+0000 NULL**

This is an unexpected-null-character parse error. Append a U+FFFD REPLACEMENT CHARACTER character to the current tag token's tag name.

↪ **EOF**

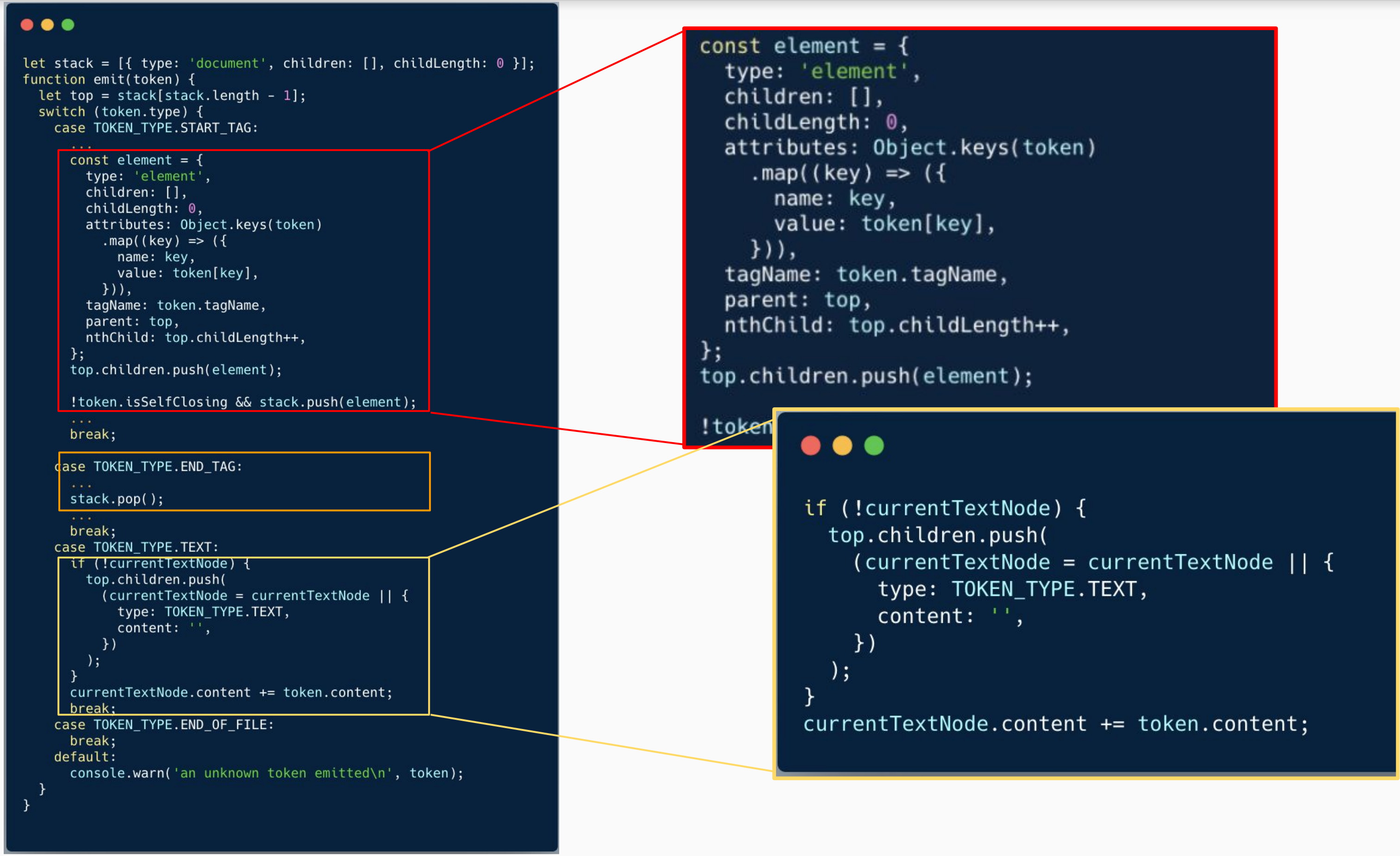
This is an eof-in-tag parse error. Emit an end-of-file token.

↪ **Anything else**

Append the current input character to the current tag token's tag name.

https://html.spec.whatwg.org/multipage/parsing.html#tag-name-state

HTML tokenization



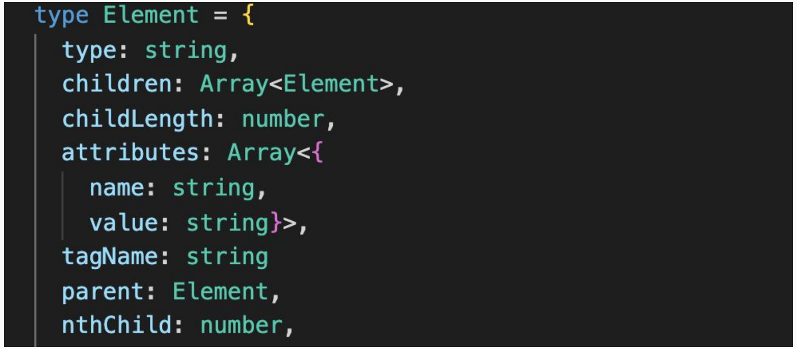
```
let stack = [{ type: 'document', children: [], childLength: 0 }];
function emit(token) {
  let top = stack[stack.length - 1];
  switch (token.type) {
    case TOKEN_TYPE.START_TAG:
      ...
      const element = {
        type: 'element',
        children: [],
        childLength: 0,
        attributes: Object.keys(token)
          .map((key) => ({
            name: key,
            value: token[key],
          })),
        tagName: token.tagName,
        parent: top,
        nthChild: top.childLength++,
      };
      top.children.push(element);

      !token.isSelfClosing && stack.push(element);
      ...
      break;

    case TOKEN_TYPE.END_TAG:
      ...
      stack.pop();
      ...
      break;
    case TOKEN_TYPE.TEXT:
      if (!currentTextNode) {
        top.children.push(
          (currentTextNode = currentTextNode || {
            type: TOKEN_TYPE.TEXT,
            content: '',
          })
        );
      }
      currentTextNode.content += token.content;
      break;
    case TOKEN_TYPE.END_OF_FILE:
      break;
    default:
      console.warn('an unknown token emitted\n', token);
  }
}
```

```
const element = {
  type: 'element',
  children: [],
  childLength: 0,
  attributes: Object.keys(token)
    .map((key) => ({
      name: key,
      value: token[key],
    })),
  tagName: token.tagName,
  parent: top,
  nthChild: top.childLength++,
};
top.children.push(element);

!token
```

```
if (!currentTextNode) {
  top.children.push(
    (currentTextNode = currentTextNode || {
      type: TOKEN_TYPE.TEXT,
      content: '',
    })
  );
}
currentTextNode.content += token.content;
```

Output

An Object of DOM tree



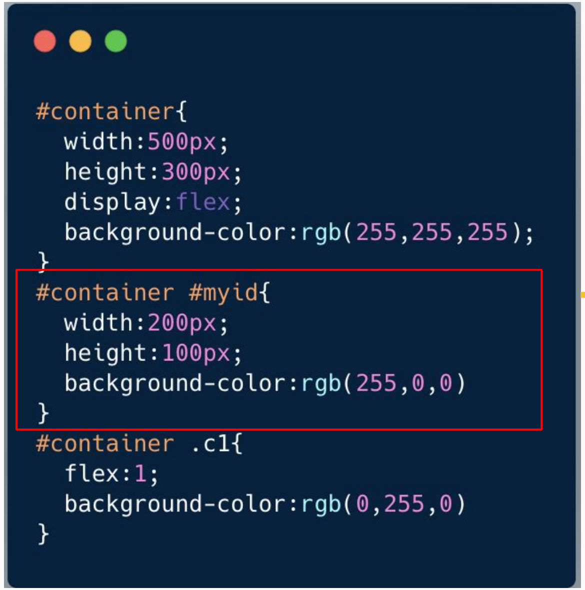
```
type Element = {
  type: string,
  children: Array<Element>,
  childLength: number,
  attributes: Array<{
    name: string,
    value: string}>,
  tagName: string
  parent: Element,
  nthChild: number,
```

DEMO

Git branch: 2-html-tokenization

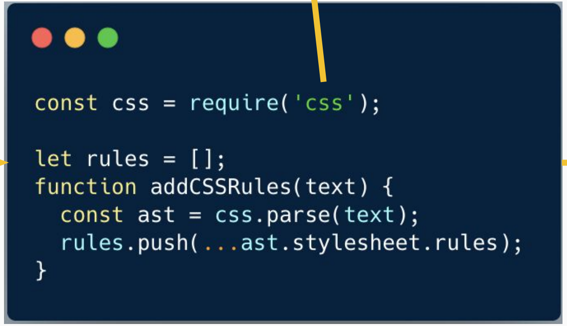
1. Parse HTTP response
2. HTML tokenization
3. CSS computing
4. Layout
5. Render

CSS computing

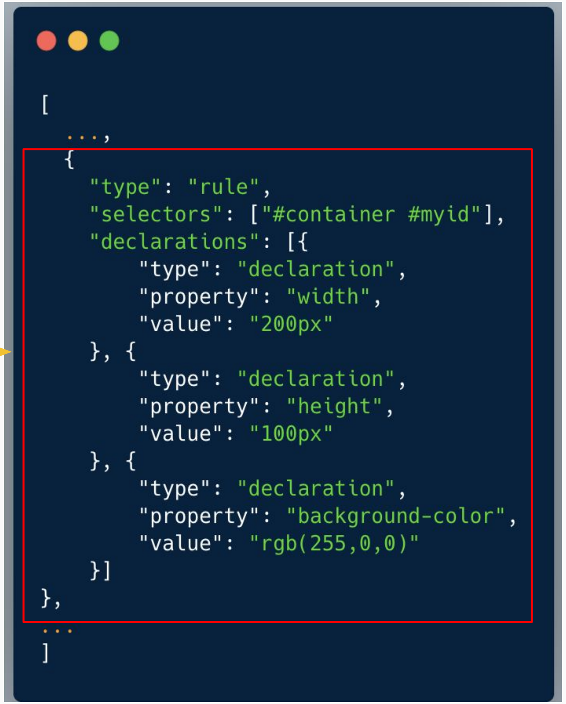


```
#container{
  width:500px;
  height:300px;
  display:flex;
  background-color:rgb(255,255,255);
}
#container #myid{
  width:200px;
  height:100px;
  background-color:rgb(255,0,0)
}
#container .c1{
  flex:1;
  background-color:rgb(0,255,0)
}
```

https://en.wikipedia.org/wiki/LR_parser

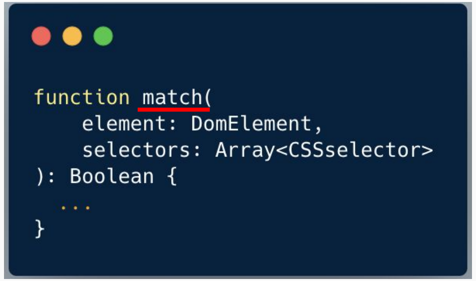


```
const css = require('css');

let rules = [];
function addCSSRules(text) {
  const ast = css.parse(text);
  rules.push(...ast.stylesheet.rules);
}
```



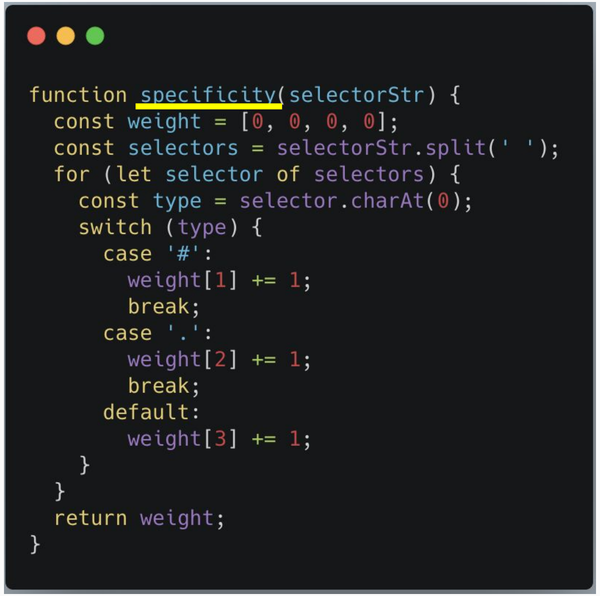
```
[
 ...,
 {
  "type": "rule",
  "selectors": ["#container #myid"],
  "declarations": [{
      "type": "declaration",
      "property": "width",
      "value": "200px"
  }, {
      "type": "declaration",
      "property": "height",
      "value": "100px"
  }, {
      "type": "declaration",
      "property": "background-color",
      "value": "rgb(255,0,0)"
  }]
},
...
]
```

Match CSS with DOM

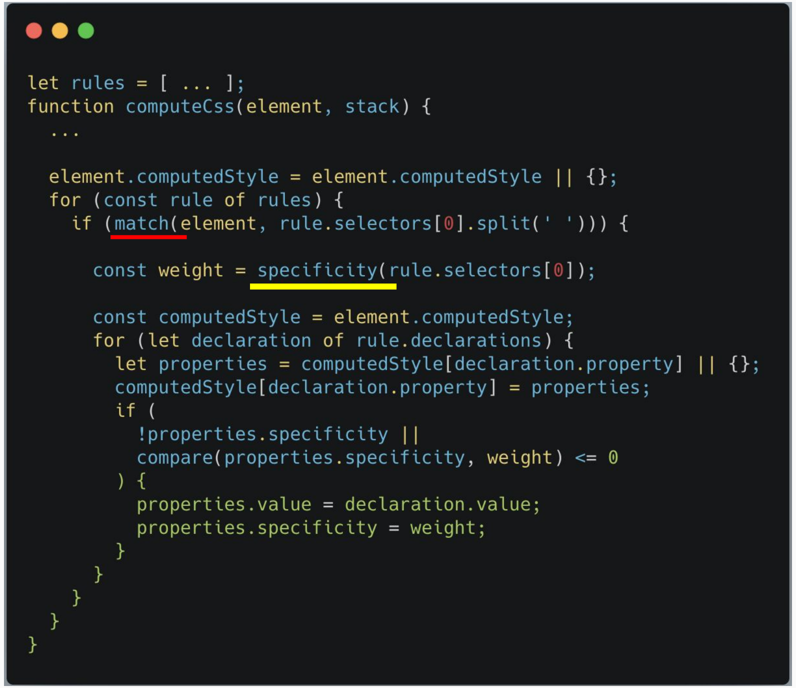


```
function match(
    element: DomElement,
    selectors: Array<CSSselector>
): Boolean {
  ...
}
```

https://gist.github.com/jzhang026/a802cb6b8b62267cb080c7d8bf787c89



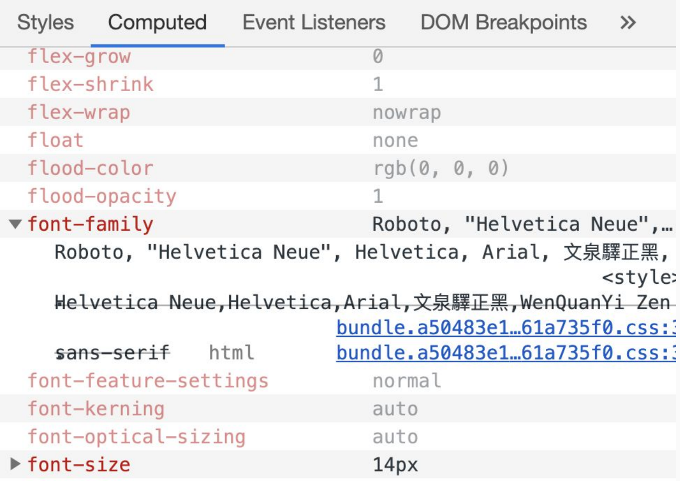
```
function specificity(selectorStr) {
  const weight = [0, 0, 0, 0];
  const selectors = selectorStr.split(' ');
  for (let selector of selectors) {
    const type = selector.charAt(0);
    switch (type) {
      case '#':
        weight[1] += 1;
        break;
      case '.':
        weight[2] += 1;
        break;
      default:
        weight[3] += 1;
    }
  }
  return weight;
}
```



```
let rules = [ ... ];
function computeCss(element, stack) {
  ...

  element.computedStyle = element.computedStyle || {};
  for (const rule of rules) {
    if (match(element, rule.selectors[0].split(' '))) {

      const weight = specificity(rule.selectors[0]);

      const computedStyle = element.computedStyle;
      for (let declaration of rule.declarations) {
        let properties = computedStyle[declaration.property] || {};
        computedStyle[declaration.property] = properties;
        if (
          !properties.specificity ||
          compare(properties.specificity, weight) <= 0
        ) {
          properties.value = declaration.value;
          properties.specificity = weight;
        }
      }
    }
  }
}
```

Output

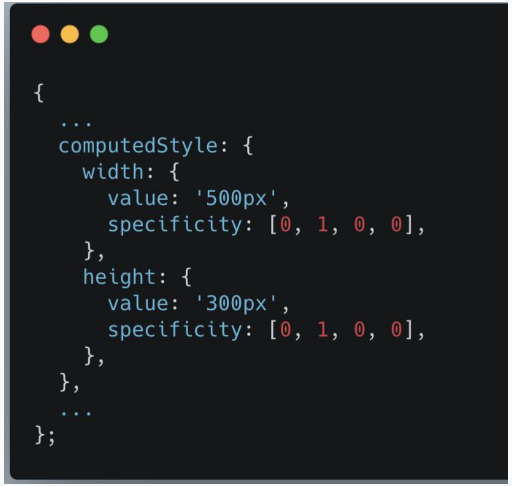
Modern Browser:



```
flex-grow                     0
flex-shrink                   1
flex-wrap                     nowrap
float                         none
flood-color                   rgb(0, 0, 0)
flood-opacity                 1
▼ font-family                 Roboto, "Helvetica Neue",…
    Roboto, "Helvetica Neue", Helvetica, Arial, 文泉驛正黑,
                                                        <style
    ~~Helvetica Neue,Helvetica,Arial,文泉驛正黑,WenQuanYi Zen~~
                              bundle.a50483e1…61a735f0.css:
    ~~sans-serif~~    html        bundle.a50483e1…61a735f0.css:
font-feature-settings         normal
font-kerning                  auto
font-optical-sizing           auto
▶ font-size                   14px
```

Toy Browser

Add a property named `computedStyle` to our DOM element.



```
{
  ...
  computedStyle: {
    width: {
      value: '500px',
      specificity: [0, 1, 0, 0],
    },
    height: {
      value: '300px',
      specificity: [0, 1, 0, 0],
    },
  },
  ...
};
```

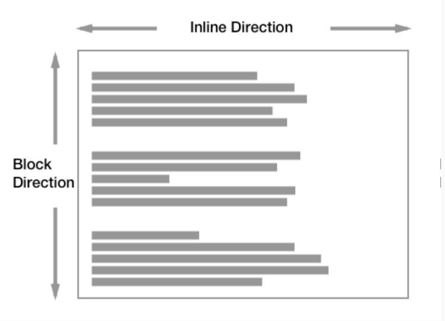
DEMO

Git Branch: 3-css-computing

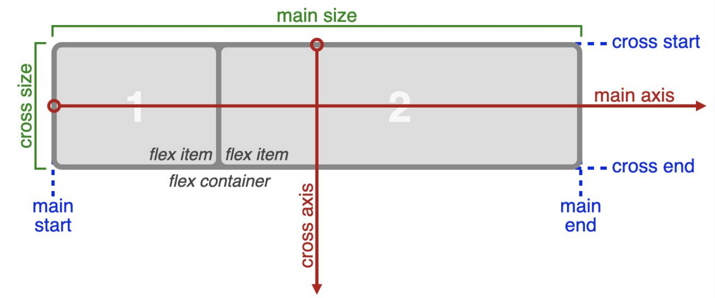
1. Parse HTTP response
2. HTML tokenization
3. CSS computing
4. Layout
5. Render

Layout

Normal Flow

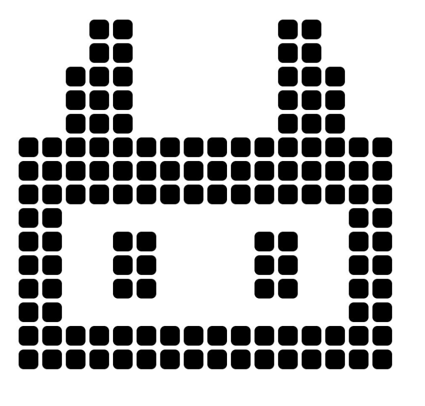


Flex Layout



https://drafts.csswg.org/css-flexbox-1/

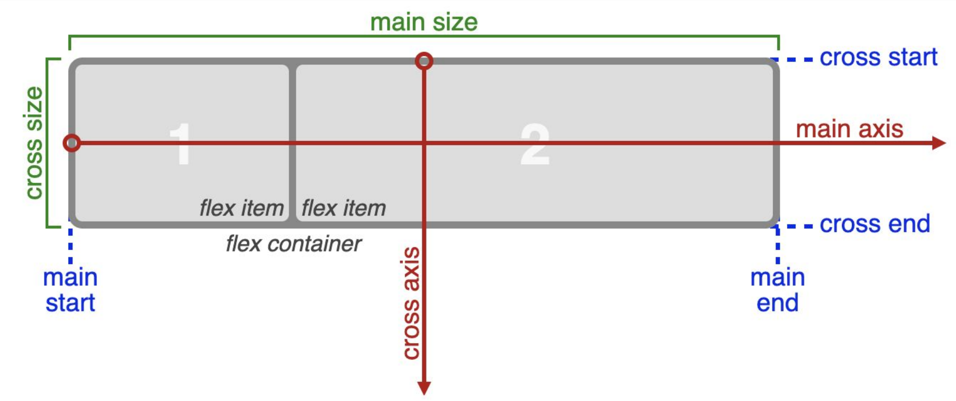
Grid Layout

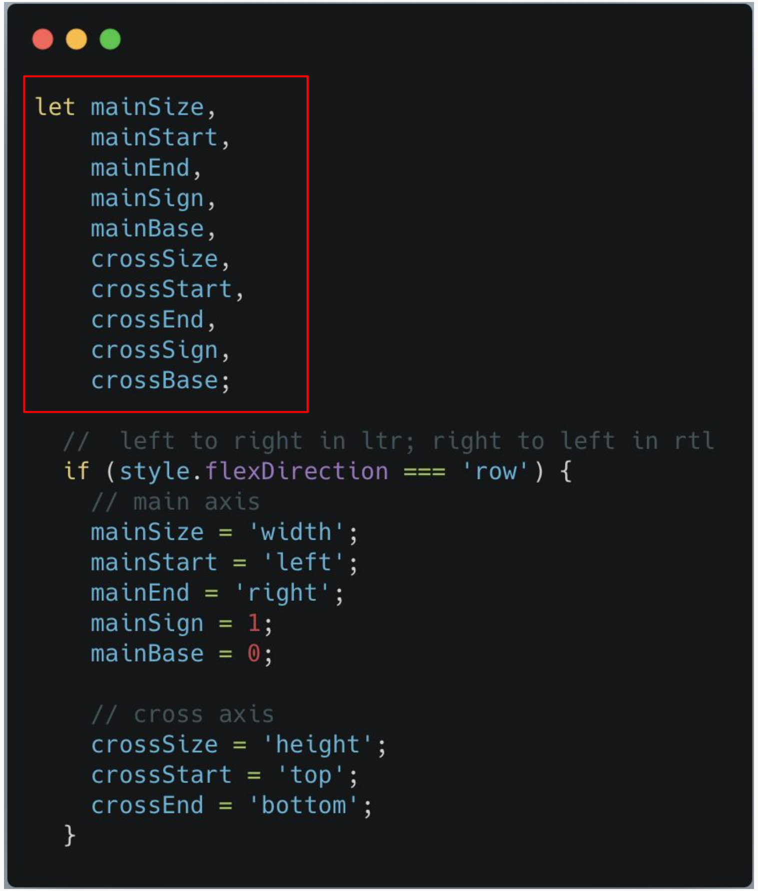


https://codesandbox.io/s/boxocat-5b9rg

https://github.com/Shopee/shopee-react-knowledgeable/issues/207

Flex Layout



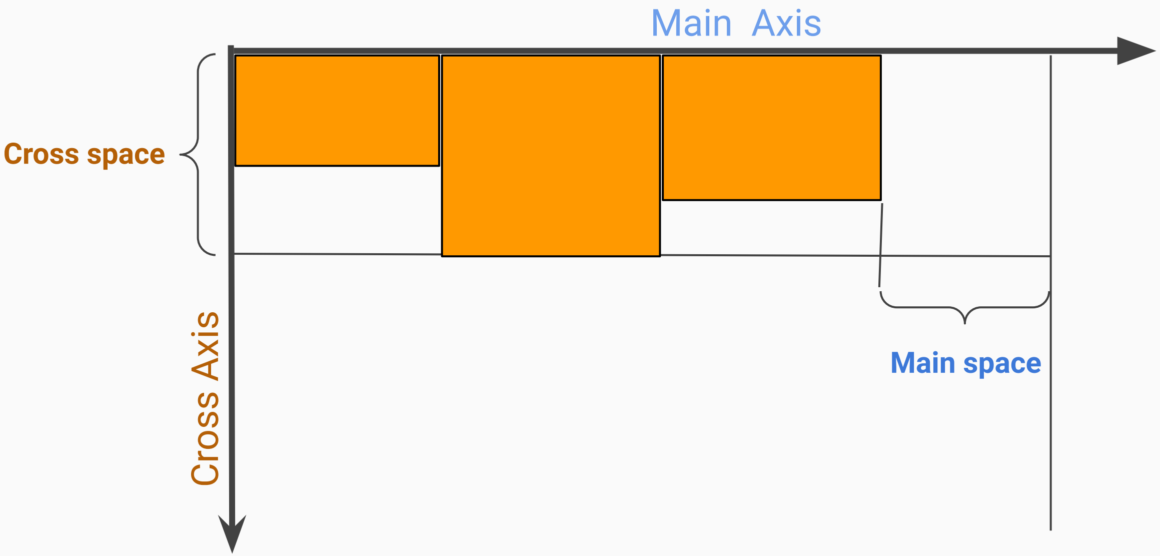


```
let mainSize,
    mainStart,
    mainEnd,
    mainSign,
    mainBase,
    crossSize,
    crossStart,
    crossEnd,
    crossSign,
    crossBase;

  //  left to right in ltr; right to left in rtl
  if (style.flexDirection === 'row') {
    // main axis
    mainSize = 'width';
    mainStart = 'left';
    mainEnd = 'right';
    mainSign = 1;
    mainBase = 0;

    // cross axis
    crossSize = 'height';
    crossStart = 'top';
    crossEnd = 'bottom';
  }
```

Flex Layout - Main axis

Collect all flex items in to rows.
flex items is the children of a flex container

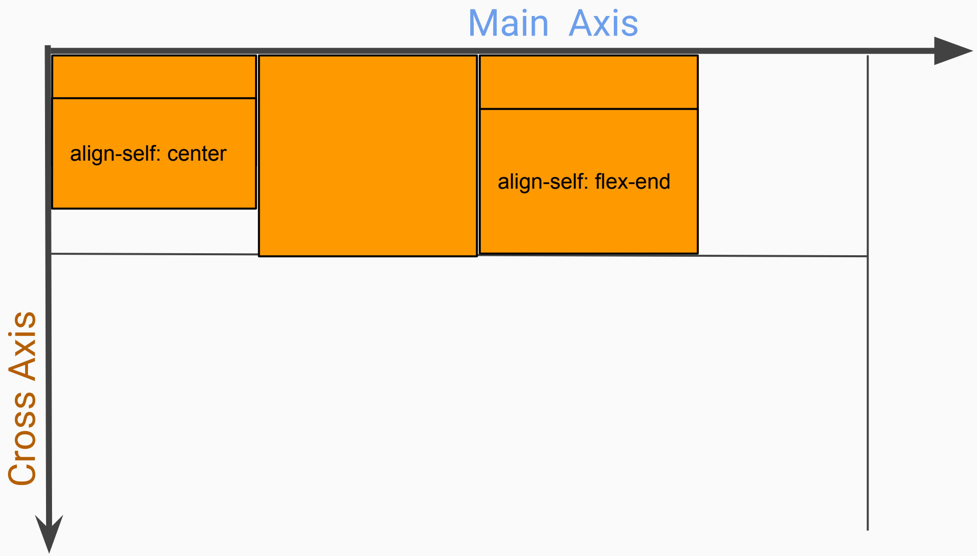
Main space > 0 ?
    Assign the extra space to those flex items
    :
    shrink those children items.



Flex Layout - Cross axis

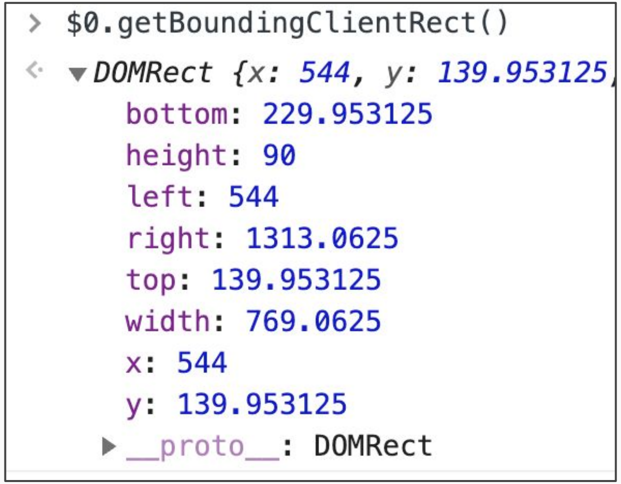
Calculate on cross axis.

- Line height is determined by the highest items in the main axis
- Determine items position based on `align-items` or `align-self`



Output

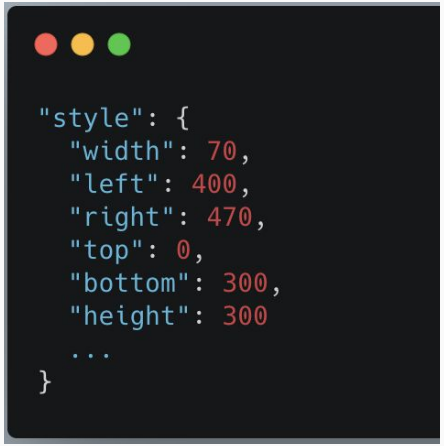
Modern Browser:



```
> $0.getBoundingClientRect()
< ▼DOMRect {x: 544, y: 139.953125,
      bottom: 229.953125
      height: 90
      left: 544
      right: 1313.0625
      top: 139.953125
      width: 769.0625
      x: 544
      y: 139.953125
    ▶ __proto__: DOMRect
```

Toy Browser

Add a property named `style` to our DOM element.



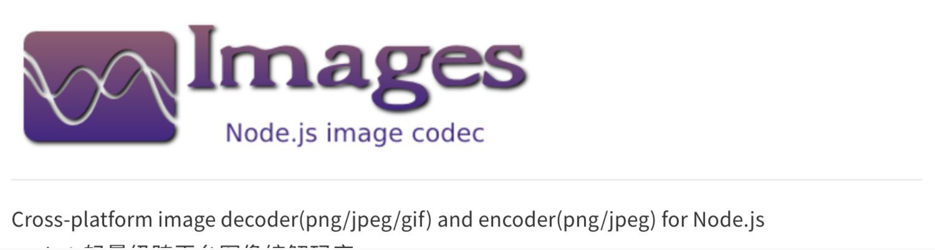
```
"style": {
  "width": 70,
  "left": 400,
  "right": 470,
  "top": 0,
  "bottom": 300,
  "height": 300
  ...
}
```

DEMO

Git Branch: 4-calculate-layout

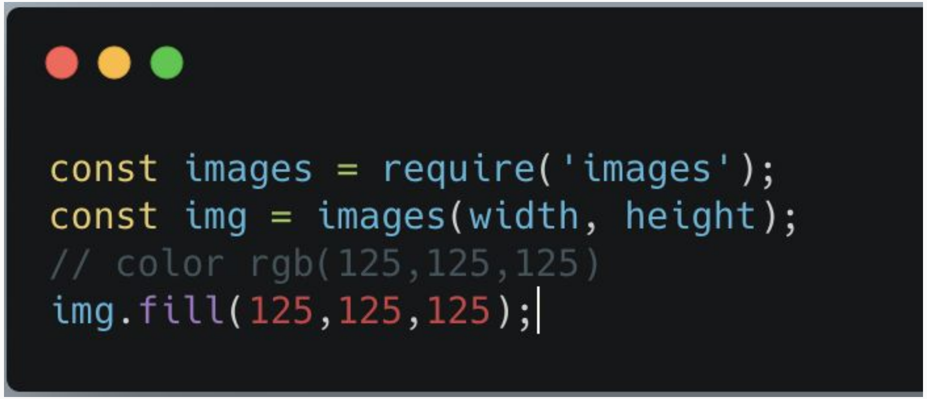
1. Parse HTTP response
2. HTML tokenization
3. CSS computing
4. Layout
5. Render

Render



Cross-platform image decoder(png/jpeg/gif) and encoder(png/jpeg) for Node.js

https://www.npmjs.com/package/images



```
const images = require('images');
const img = images(width, height);
// color rgb(125,125,125)
img.fill(125,125,125);
```

Output

An image described by the HTML string

DEMO

Git Branch: 5-render

Thank You