Fetch

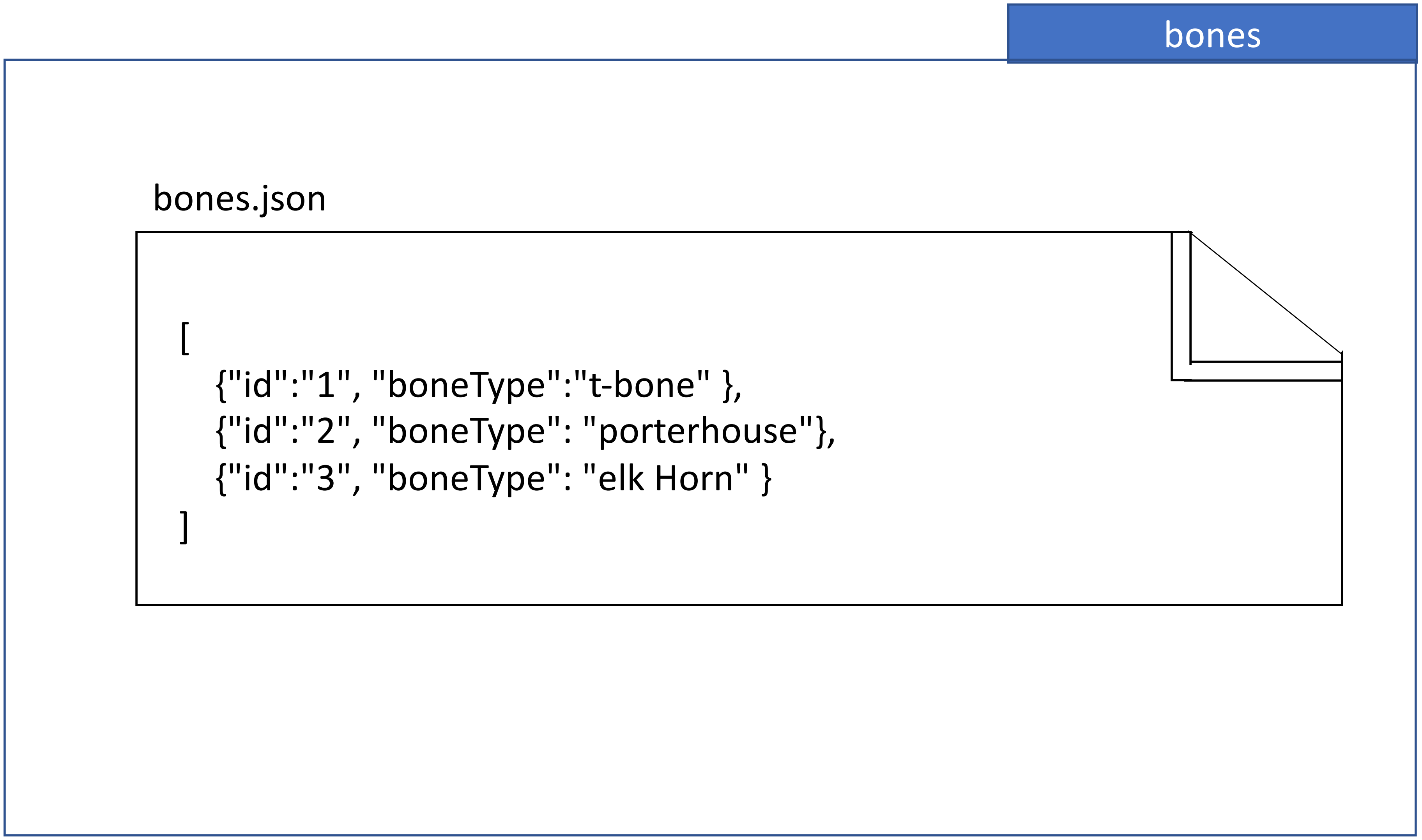
Fetch

- The Fetch API is a modern interface that allows for http/http(s) asynchronous requests from web browsers.
- Older methods of http/http(s) requests used the XMLHttpRequest (XHR) object.
  - Fetch can perform all tasks that the XHR object can.
  - Fetches uses cleaner syntax as well.
- Fetch supports promises.
- Fetches uses cleaner syntax as well.
- Note: Does not support Internet Explorer

Simple Fetch: Fido Retrieves a bone.

- If fido tries to retrieve a bone several outcomes can happen:
    - Fido retrieves the bone (success/resolved)
    - Fido cannot retrieve the bone (failure/rejected)

bones.json

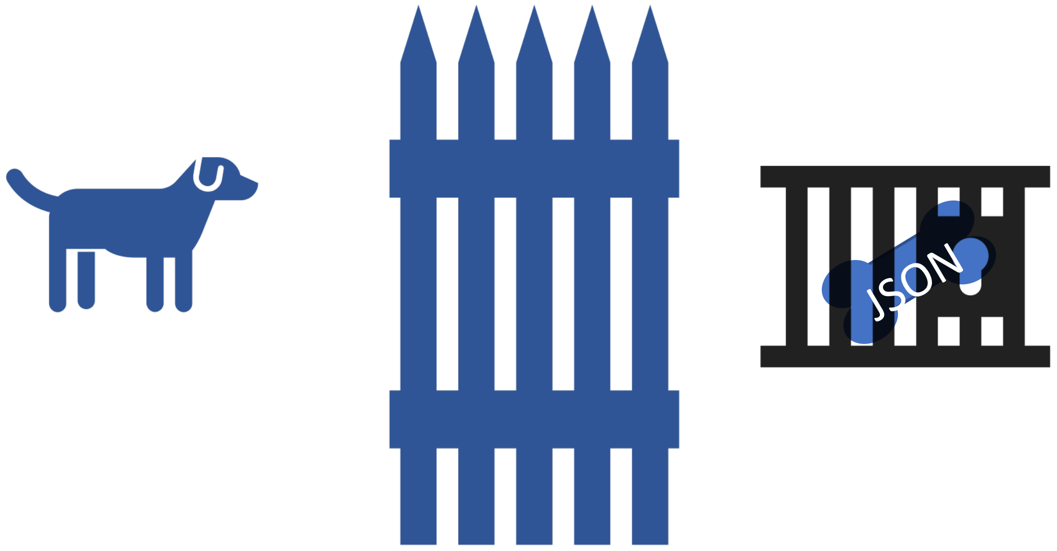


```json
[
   {"id":"1", "boneType":"t-bone" },
   {"id":"2", "boneType": "porterhouse"},
   {"id":"3", "boneType": "elk Horn" }
]
```

Fidos Bone Quest: Locally or Remotely

Fido Gets a bone
Located locally

Fido Gets a bone
Located in the neighbor's yard



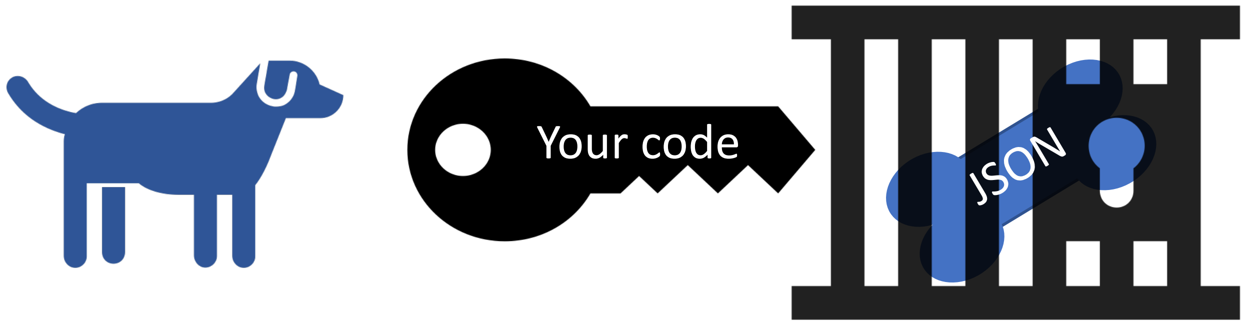
Fidos Bone Quest: Locally

Fido Gets a bone
Located in his yard

Basic challenges Local and Remote Data:

- Find way to fetch the bone.
- Find a way to get at the JSON content of the bone.

Fido’s Bone is not readily accessible: He wants the JSON content of the bone



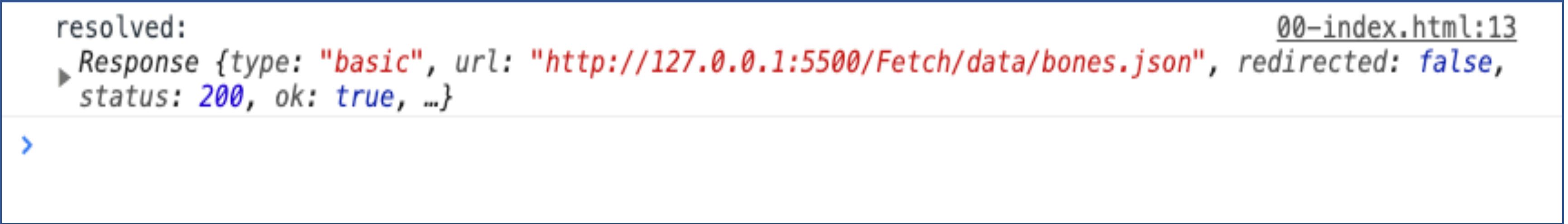
When we use “fetch” we also need to find a way to get the content out of the response (res) object.

fetch: Locally

JavaScript

```
fetch('../data/bones.json’)
   .then((res)=>{
    console.log('resolved: ', res)
 })
```

Browser Console

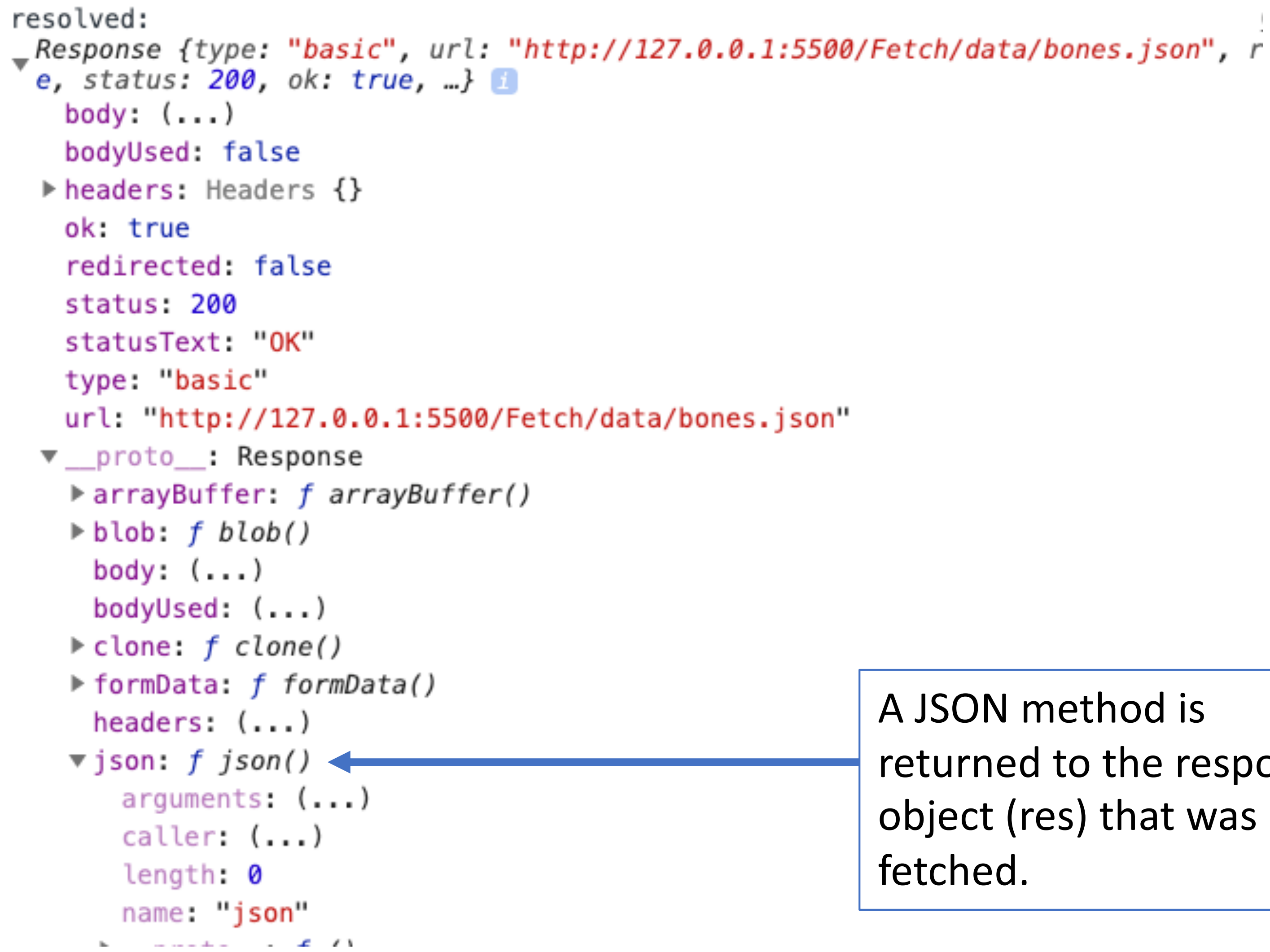


```
resolved:                                                               00-index.html:13
▸ Response {type: "basic", url: "http://127.0.0.1:5500/Fetch/data/bones.json", redirected: false,
  status: 200, ok: true, …}
>
```

- The fetch API makes a request to GET the data and the data was obtained from the file and outputted to the browser by a response object (res).
- The problem is that the data is not in a readable format for us to examine, however we do get some meaningful information such as the “status” code, which can be used to do error handling.
- Fetch/01-SimpleFetch/00-index.html

fetch: Locally

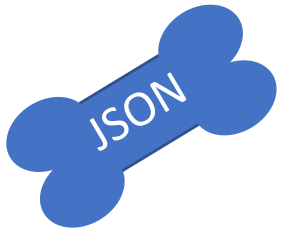
Browser Output Expanded



```
resolved:
▼Response {type: "basic", url: "http://127.0.0.1:5500/Fetch/data/bones.json", r
 e, status: 200, ok: true, …} ℹ
   body: (...)
   bodyUsed: false
 ▶ headers: Headers {}
   ok: true
   redirected: false
   status: 200
   statusText: "OK"
   type: "basic"
   url: "http://127.0.0.1:5500/Fetch/data/bones.json"
 ▼ __proto__: Response
   ▶ arrayBuffer: ƒ arrayBuffer()
   ▶ blob: ƒ blob()
     body: (...)
     bodyUsed: (...)
   ▶ clone: ƒ clone()
   ▶ formData: ƒ formData()
     headers: (...)
   ▼ json: ƒ json()
       arguments: (...)
       caller: (...)
       length: 0
       name: "json"
```

A JSON method is returned to the response object (res) that was fetched.

fetch: Locally



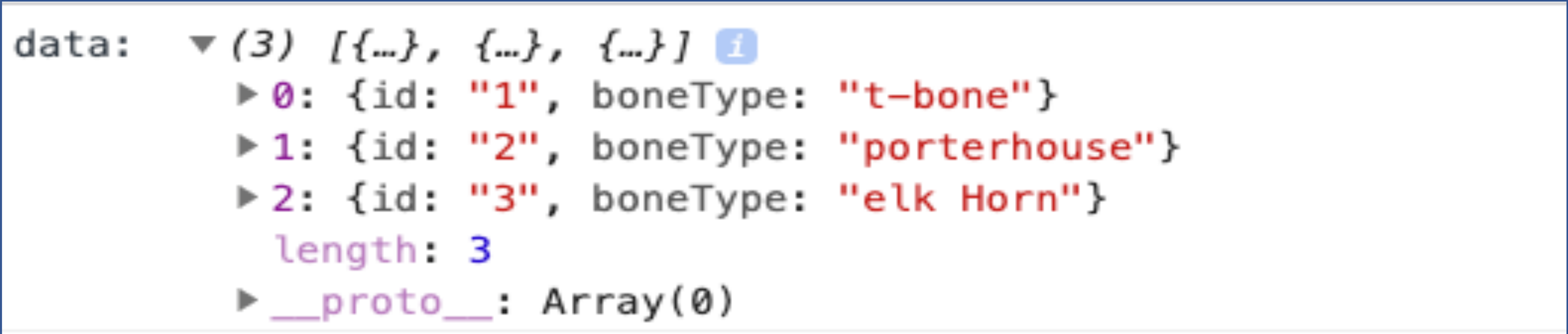
JavaScript

```
fetch('../data/bones.json’)
  .then((res)=>{
 // console.log('resolved: ', res)
 //res.json returns a promise which will
 //be the data object in JSON format
   return res.json()
  }).then(data => {
      console.log('data: ', data)
  })
```

Three Steps

1. Fetch the data
2. Return the response using the JSON() method
3. Do something with the data.

Browser Console



```
data:    ▼(3) [{…}, {…}, {…}]
           ▶0: {id: "1", boneType: "t-bone"}
           ▶1: {id: "2", boneType: "porterhouse"}
           ▶2: {id: "3", boneType: "elk Horn"}
            length: 3
           ▶__proto__: Array(0)
```

Fetch/01-SimpleFetch/01-index.html

fetch: Locally + Error Handling

JavaScript

```
fetch('../12313data/bones.json’)
  .then((res) => {
  return res.json()
  }).then(data => {
  console.log('data: ', data)
  //Error handling in a fetch only gets triggered if a network connection
 // can not be established or maintained
}).catch((err) => {
console.log('rejected', err)
})
```

Note: Fetch views errors as network connection errors and not 400, 500 status codes.

Fetch/01-SimpleFetch/02-index.html

fetch Locally: + async/await: Code

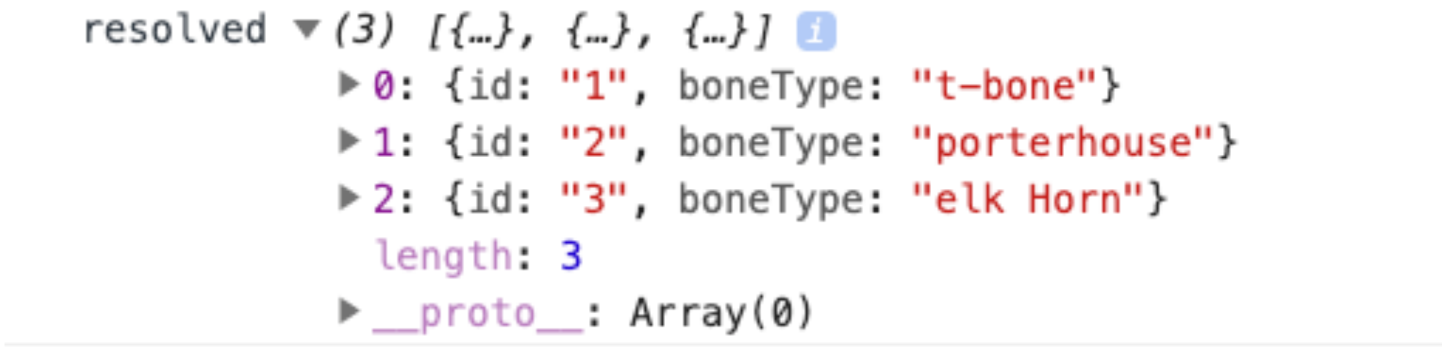
JavaScript

```
// async functions always returns a promise
const getLocalBones = async () =>{
//the await keyword "stalls" the JS assignment
// until the data is returned (promise resolves or is rejected)
    const res = await fetch('../data/bones.json’)
    const data = await res.json();
    return data;
}
// After the function resolves the .then is run
let retrivedBones = getLocalBones()
.then(data => console.log('resolved',data));
```

Using async and await cleans the code so we do not need to chain on functionality using .then within the async function.
This becomes more powerful for multiple fetches.

Fetch/01-SimpleFetch/03-index.html

fetch Locally + async/await: Results



```
resolved ▼(3) [{…}, {…}, {…}]
           ▶0: {id: "1", boneType: "t-bone"}
           ▶1: {id: "2", boneType: "porterhouse"}
           ▶2: {id: "3", boneType: "elk Horn"}
            length: 3
           ▶__proto__: Array(0)
```

Check if the async is blocking code

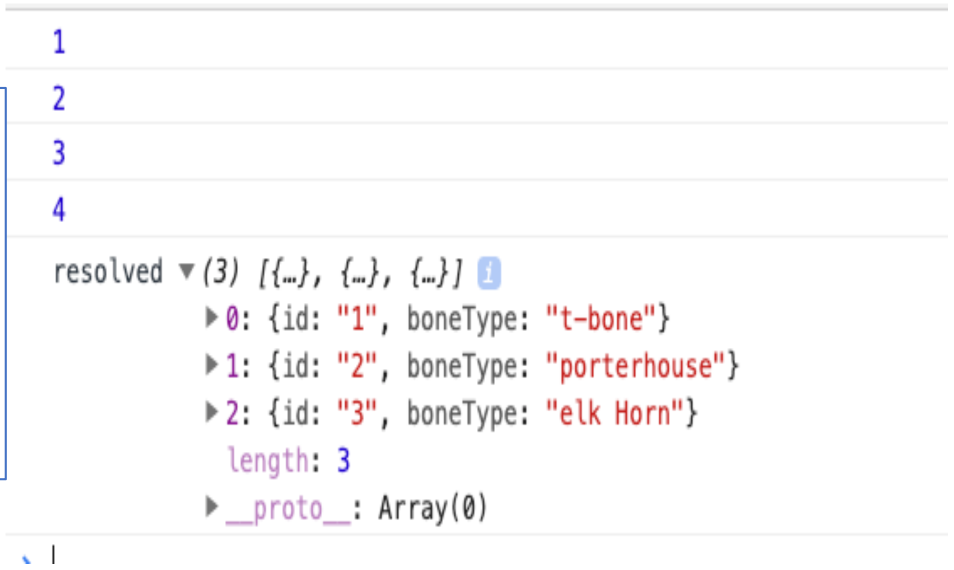
```
// async functions always returns a promise
const getLocalBones = async () => {
//the await keyword "stalls" the JS assignment
// until the data is returned (promise resolves or is rejected)
const res = await fetch('../data/bones.json')
const data = await res.json();

return data;
}
console.log(1);
console.log(2);
let retrivedBones = getLocalBones()
.then(data => console.log('resolved', data));
console.log(3);
console.log(4);
```

What do you expect to be printed to the browser console?

Fetch/01-SimpleFetch/03b-index.html

Check if the async is blocking code: Browser

```
console.log(1);
console.log(2);
let retrivedBones = getLocalBones()
.then(data => console.log('resolved', data));
console.log(3);
console.log(4);
```



```
1
2
3
4
resolved ▼(3) [{…}, {…}, {…}]
    ▶0: {id: "1", boneType: "t-bone"}
    ▶1: {id: "2", boneType: "porterhouse"}
    ▶2: {id: "3", boneType: "elk Horn"}
     length: 3
    ▶__proto__: Array(0)
```

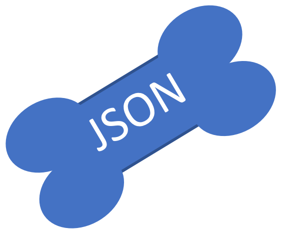
The async and await function is non-blocking code, as it took longer to finish than the regular console.log statements

Error Handling async/await

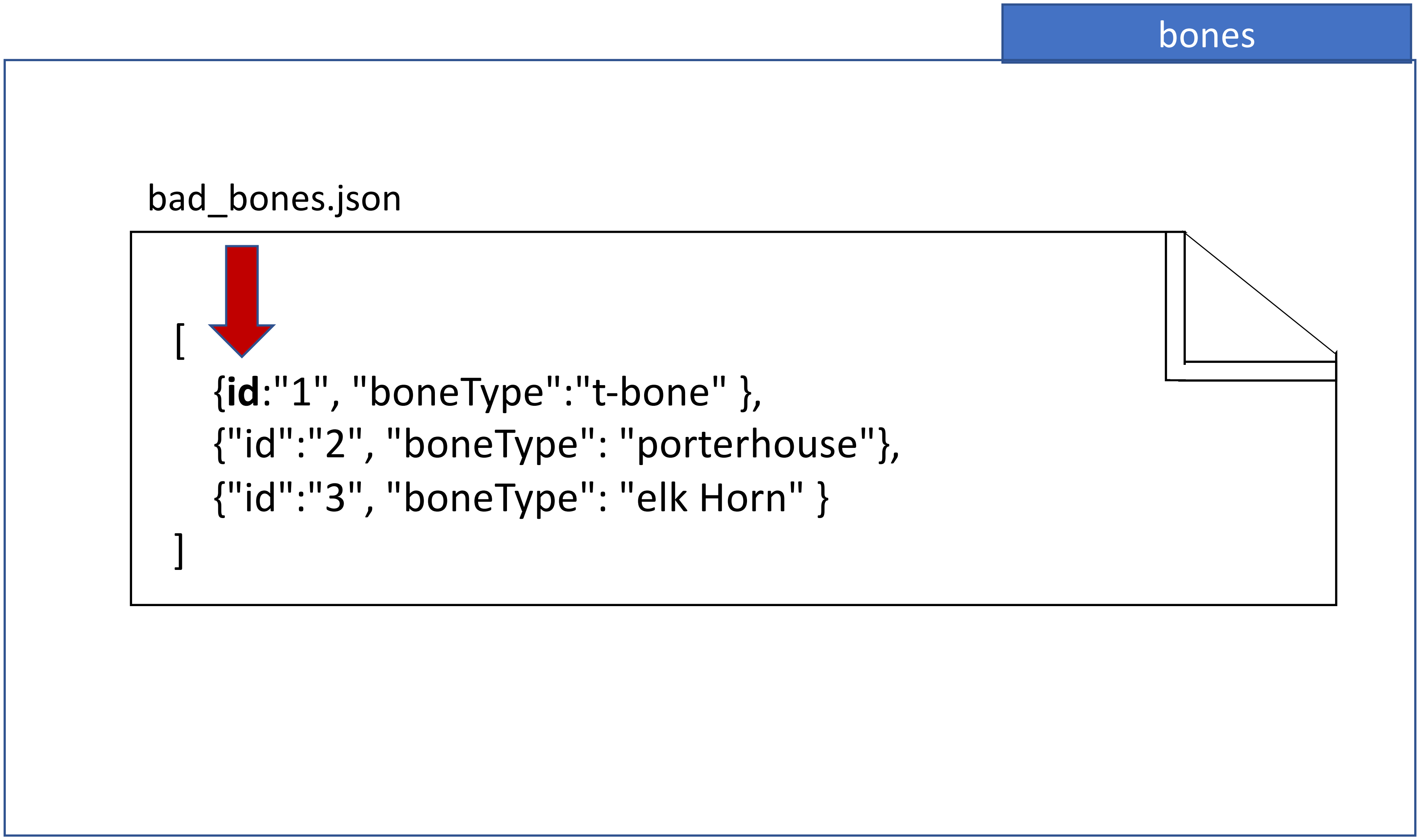
```
const getLocalBones = async () => {
  //fetch ill-formed JSON
  const res = await fetch('../data/bad_bones.json’)
  const data = await res.json();
  return data;
}
let retrivedBones = getLocalBones()
.then(data => console.log('resolved', data))
// Catch the error this Error detects if the JSON can be read
// Note: No error will result if the resource you are fetching does
// exist
.catch((err) => console.log('rejected', err.message))
```

Fetch/01-SimpleFetch/04-index.html

bad_bones.json



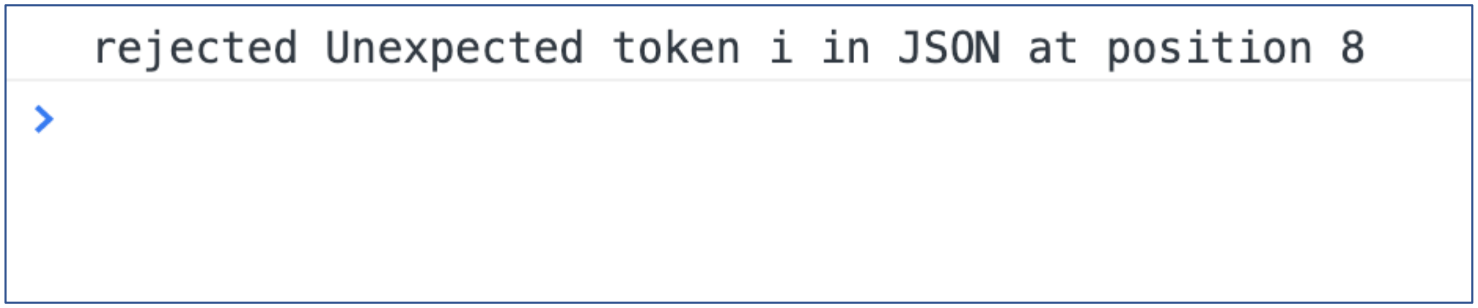
JSON. Is not formatted Properly



bad_bones.json

```
[
   {id:"1", "boneType":"t-bone" },
   {"id":"2", "boneType": "porterhouse"},
   {"id":"3", "boneType": "elk Horn" }
]
```

Error Handling async/await



```
rejected Unexpected token i in JSON at position 8
>
```

The browser shows the rejection and the error for the attempted fetching of the JSON file

fetch: Remotely

Fido wants to Get a bone
Located in the neighbor's yard

fetch: Remotely

Fido Gets a bone
Located in the neighbor's yard

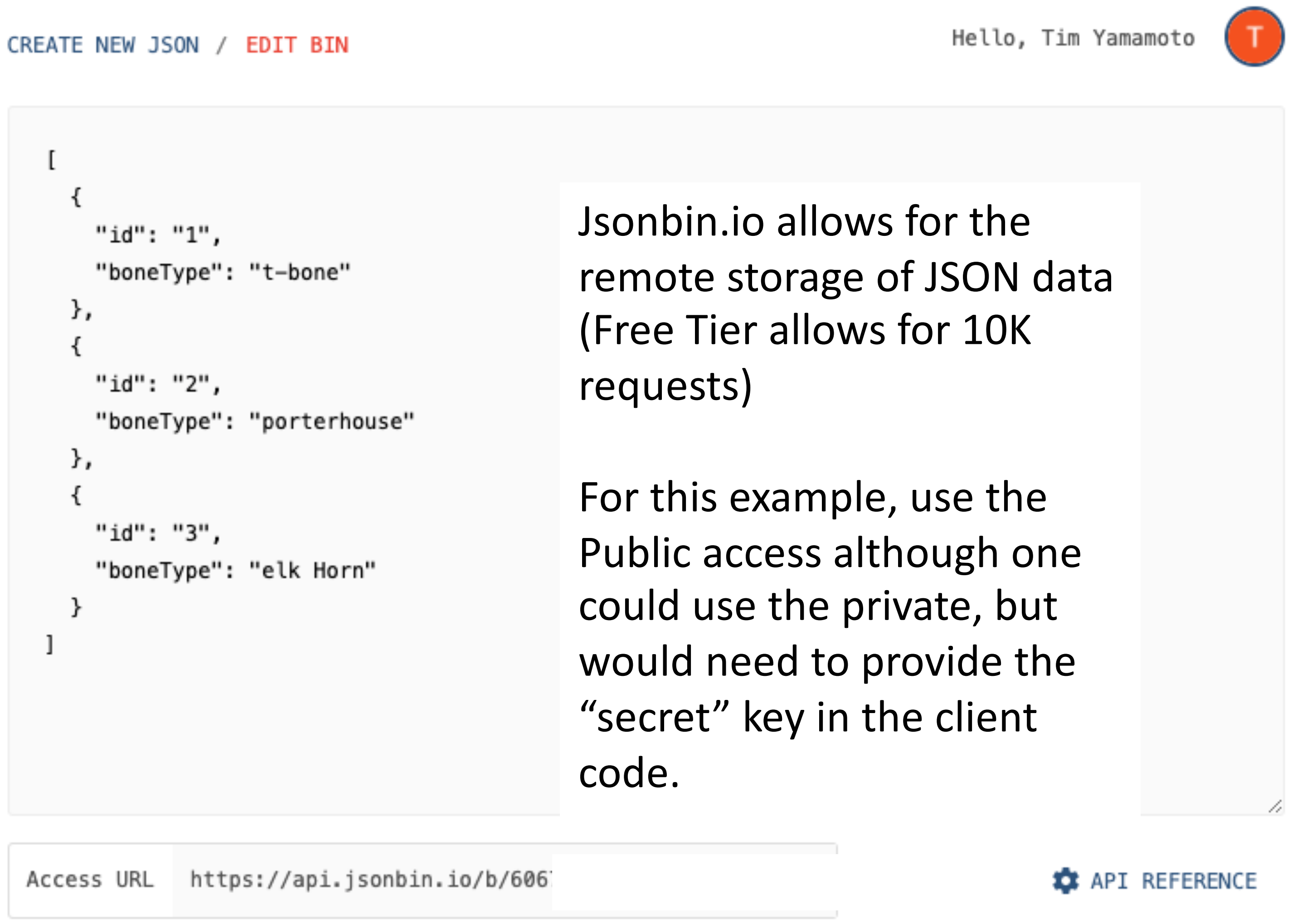
Basic challenges Local and Remote Data:

- Find way to fetch the bone.
- Find a way to get at the JSON content of the bone.

Additional challenges for Remote Retrieval of data:

- Find a way to get over the neighbor's yard.

fetch: Remote: jsonbin.io



```
[
  {
    "id": "1",
    "boneType": "t-bone"
  },
  {
    "id": "2",
    "boneType": "porterhouse"
  },
  {
    "id": "3",
    "boneType": "elk Horn"
  }
]
```

Access URL https://api.jsonbin.io/b/606

Jsonbin.io allows for the remote storage of JSON data (Free Tier allows for 10K requests)

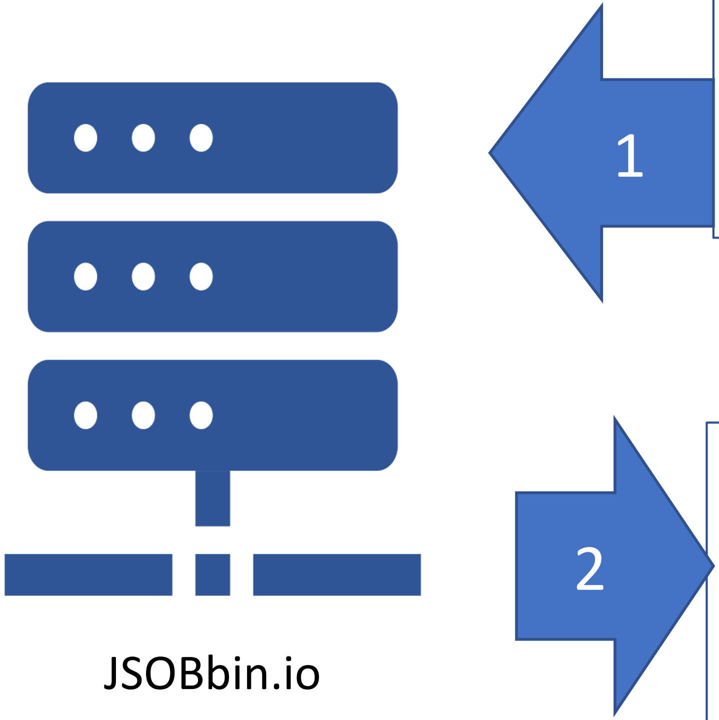
For this example, use the Public access although one could use the private, but would need to provide the "secret" key in the client code.

fetch: Remotely

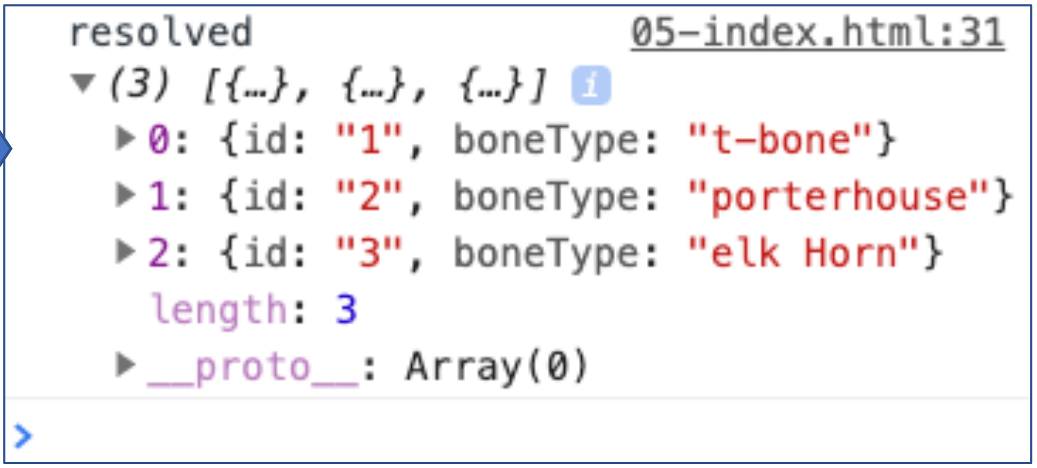
```
const remoteDataURL = "your-unique-URL"

const getRemoteBones = async () => {
//Change the URL, Add Optional Arguments:
const res = await fetch(remoteDataURL, {
   //Optional arguments
    method: 'GET’,
    headers: {
      'Accept': 'application/json. text/plain, */*’,
      "Content-Type": "application/json",
    },
  }
)
const data = await res.json();
return data;
}

let retrivedBones = getRemoteBones()
.then(data => console.log('resolved', data))
.catch((err) => console.log('rejected', err.message))
```

1. Supply remote URL and secret key if required.
2. Supply optional arguments (“Pre-flight”).

Fetch/01-SimpleFetch/05-index.html

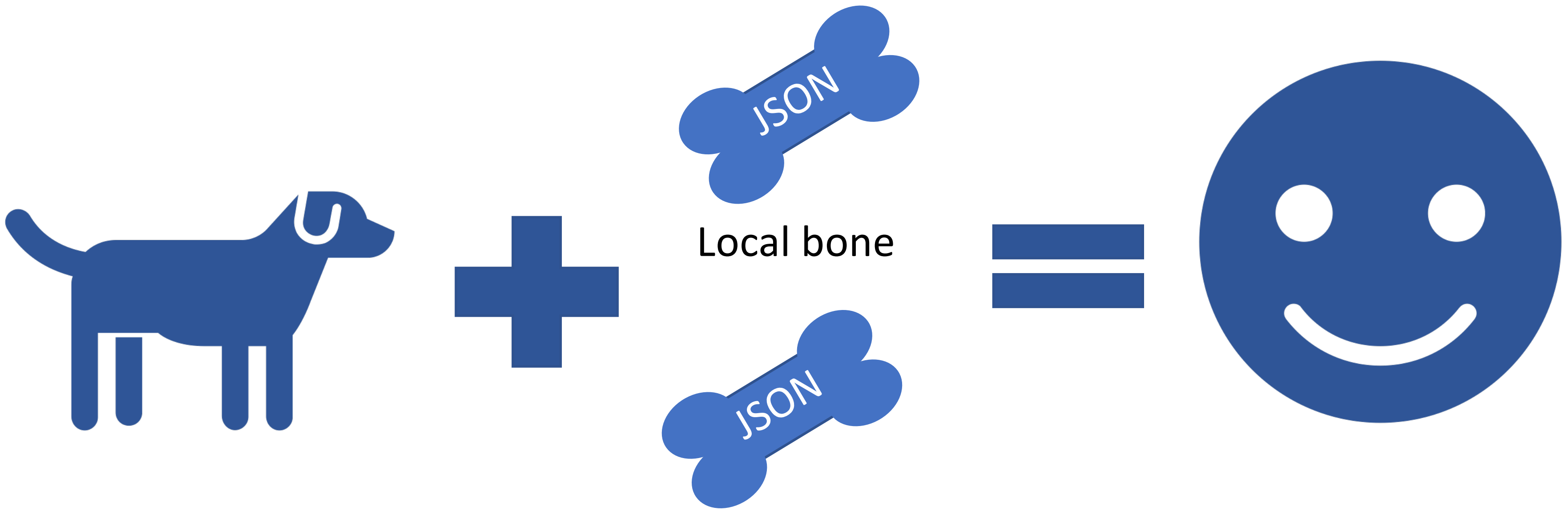


```
fetch(remoteDataURL, {
  //Optional arguments
   method: 'GET',
   headers: {
     'Accept': 'application/json. text/plain, */*',
     "Content-Type": "application/json",
   },
```



```
  resolved                                   05-index.html:31
  ▼(3) [{…}, {…}, {…}]
    ▶0: {id: "1", boneType: "t-bone"}
    ▶1: {id: "2", boneType: "porterhouse"}
    ▶2: {id: "3", boneType: "elk Horn"}
     length: 3
    ▶__proto__: Array(0)
>
```

Fido is Happy now



Remote bone