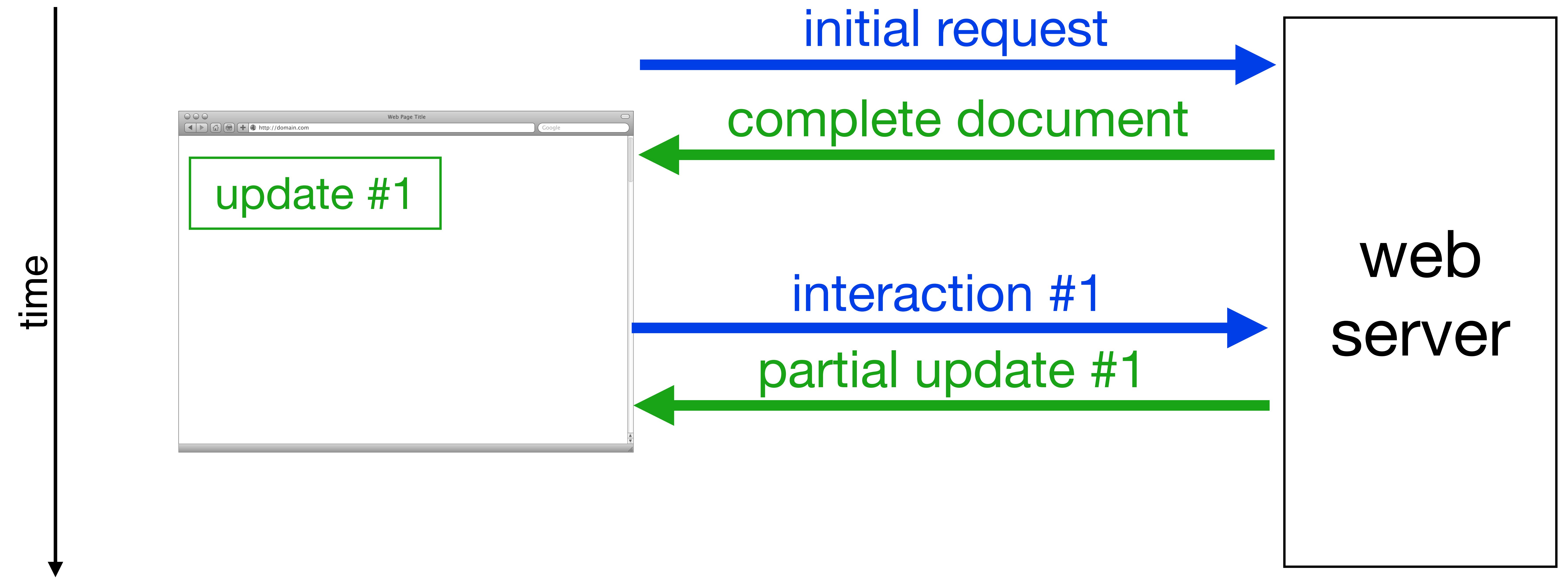
Web Programming

AJAX, Fetch and await

Leander Jehl | University of Stavanger

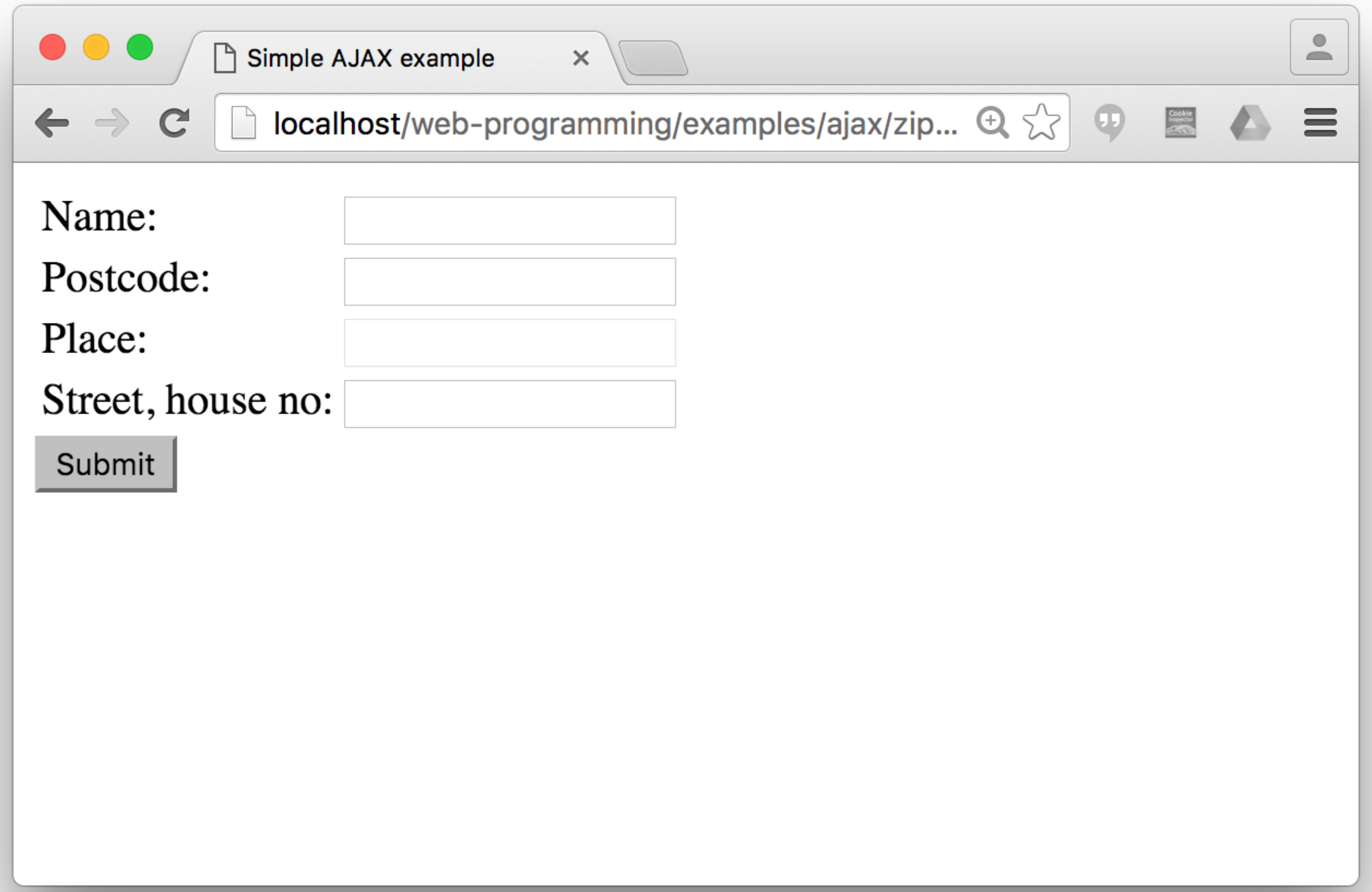
RECAP AJAX using XMLHTTPRequest



Example walkthrough

https://github.com/dat310-spring20/course-info/tree/master/
examples/async/zipcode

Example

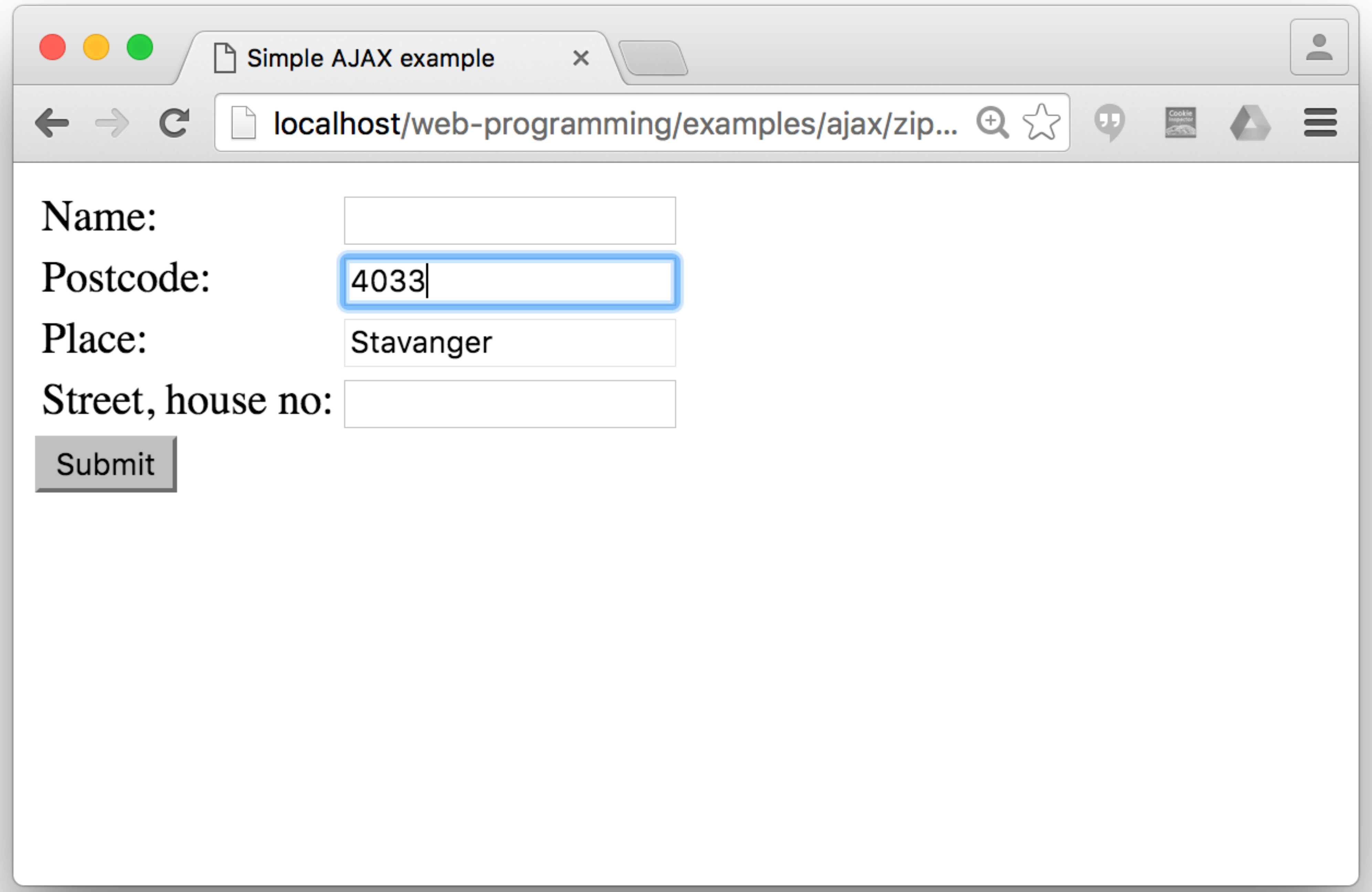


Simple AJAX example

localhost/web-programming/examples/ajax/zip...

Name:	
Postcode:	
Place:	
Street, house no:	

Submit



Simple AJAX example

localhost/web-programming/examples/ajax/zip...

Name:	
Postcode:	4033
Place:	Stavanger
Street, house no:	

Submit

Flask app

- Run the flask application.
  - Opening zipcode.html directly will not work

app.py

```
@app.route("/")
def index():
    return app.send_static_file("zipcode.html")
```

AJAX request with callback

- Make asynchronous call

zipcode.js

```
function getPlace(postcode) {
    var xhr = new XMLHttpRequest();
    /* register an embedded function as the handler */
    xhr.onreadystatechange = function () {
        /* readyState = 4 means that the response has been completed
         * status = 200 indicates that the request was successfully completed */
        if (xhr.readyState == 4 && xhr.status == 200) {
            var result = xhr.responseText;
            document.getElementById("place").value = result;
        }
    };
    /* send the request using GET */
    xhr.open("GET", "/getplace?postcode=" + postcode, true);
    xhr.send(null);
}
```

Callback function

Send request

Restructured GET

- **`ajaxGET`** function contains no application logic
- **`success`** function called with reply text.

zipcode.js

```
function ajaxGET(uri, success){
    var xhr = new XMLHttpRequest();
    /* register an embedded function as the handler */
    xhr.onreadystatechange = function () {
        /* readyState = 4 means that the response has been completed
         * status = 200 indicates that the request was successfully completed */
        if (xhr.readyState == 4 && xhr.status == 200) {
            var result = xhr.responseText;
            success(result);  Callback function
        }
    };
    /* send the request using GET */                Send request
    xhr.open("GET", uri, true);
    xhr.send(null);
}
```

Restructured GET

zipcode.js

```
/* update place in form. Used as success funtion */
function updatePlace(place){
    document.getElementById("place").value = result;
}

/* get place from postcode */
function getPlace(postcode) {

    let uri = "/getplace?postcode=" + postcode;

    ajaxGET(uri,updatePlace);
}
```

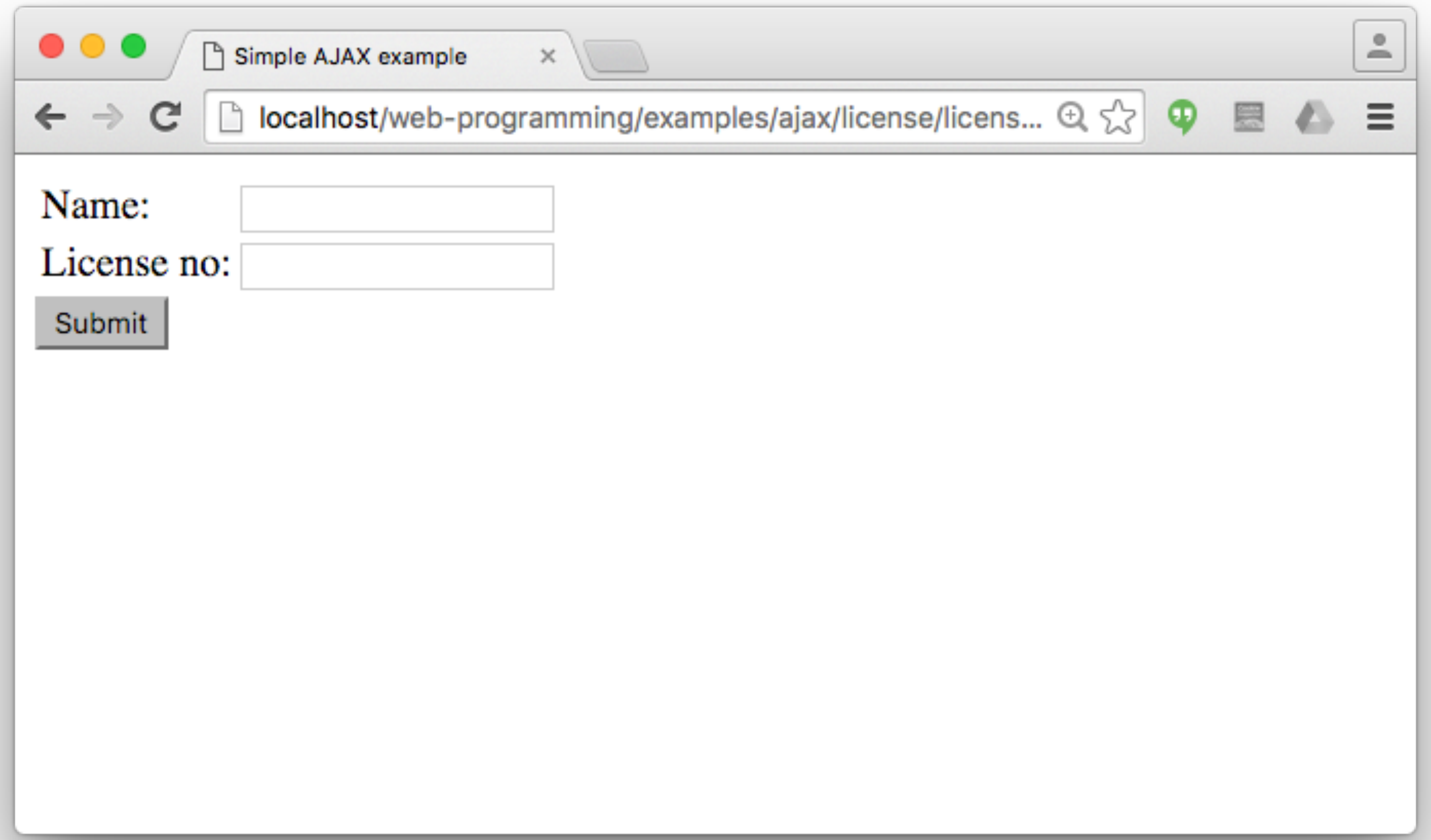
Callback function

Encode parameters in URI

Example walkthrough #2

https://github.com/dat310-spring20/course-info/tree/master/
examples/async/license

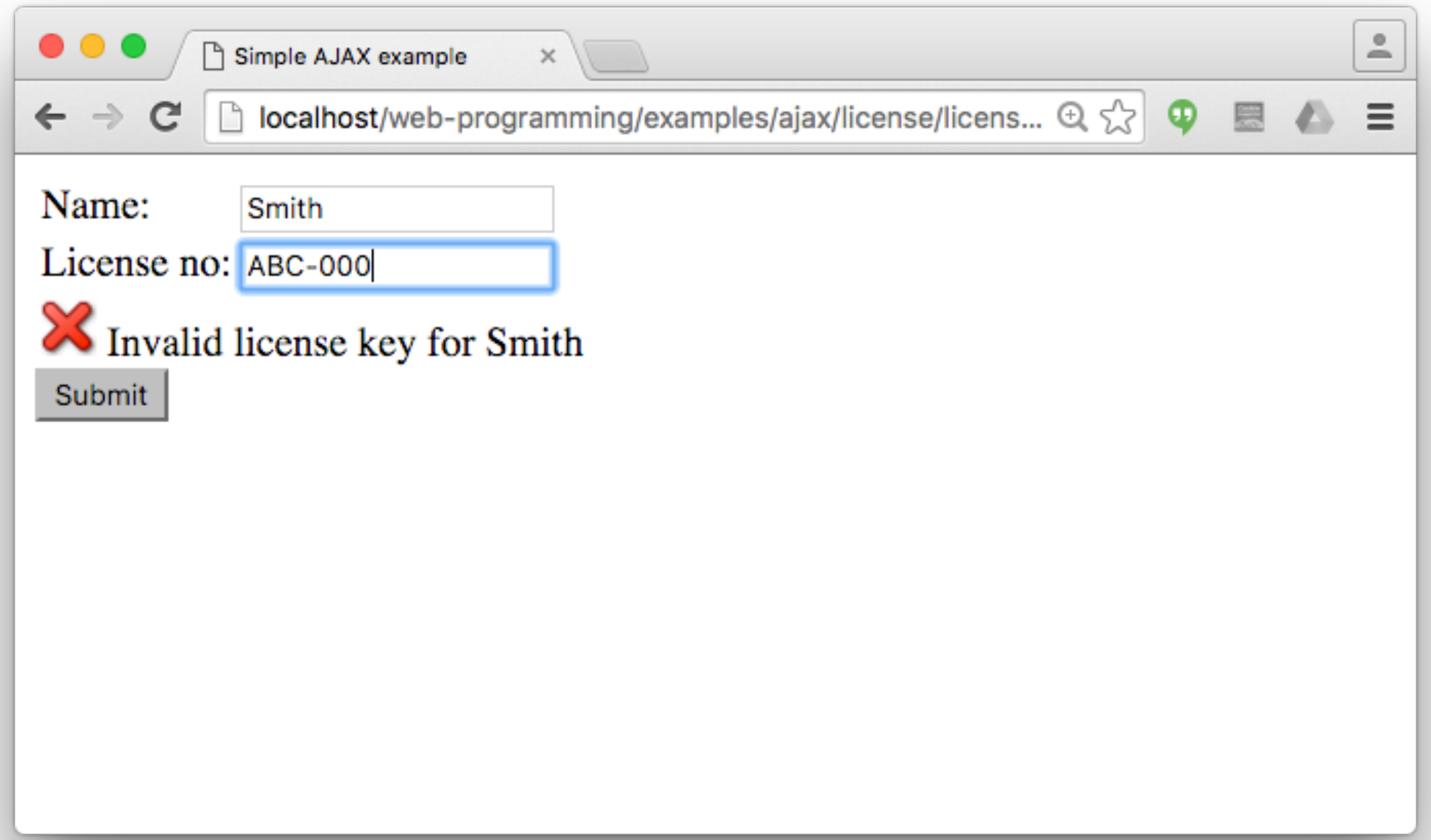
Example #2


Simple AJAX example

localhost/web-programming/examples/ajax/license/licens...

Name:	
License no:	

Submit

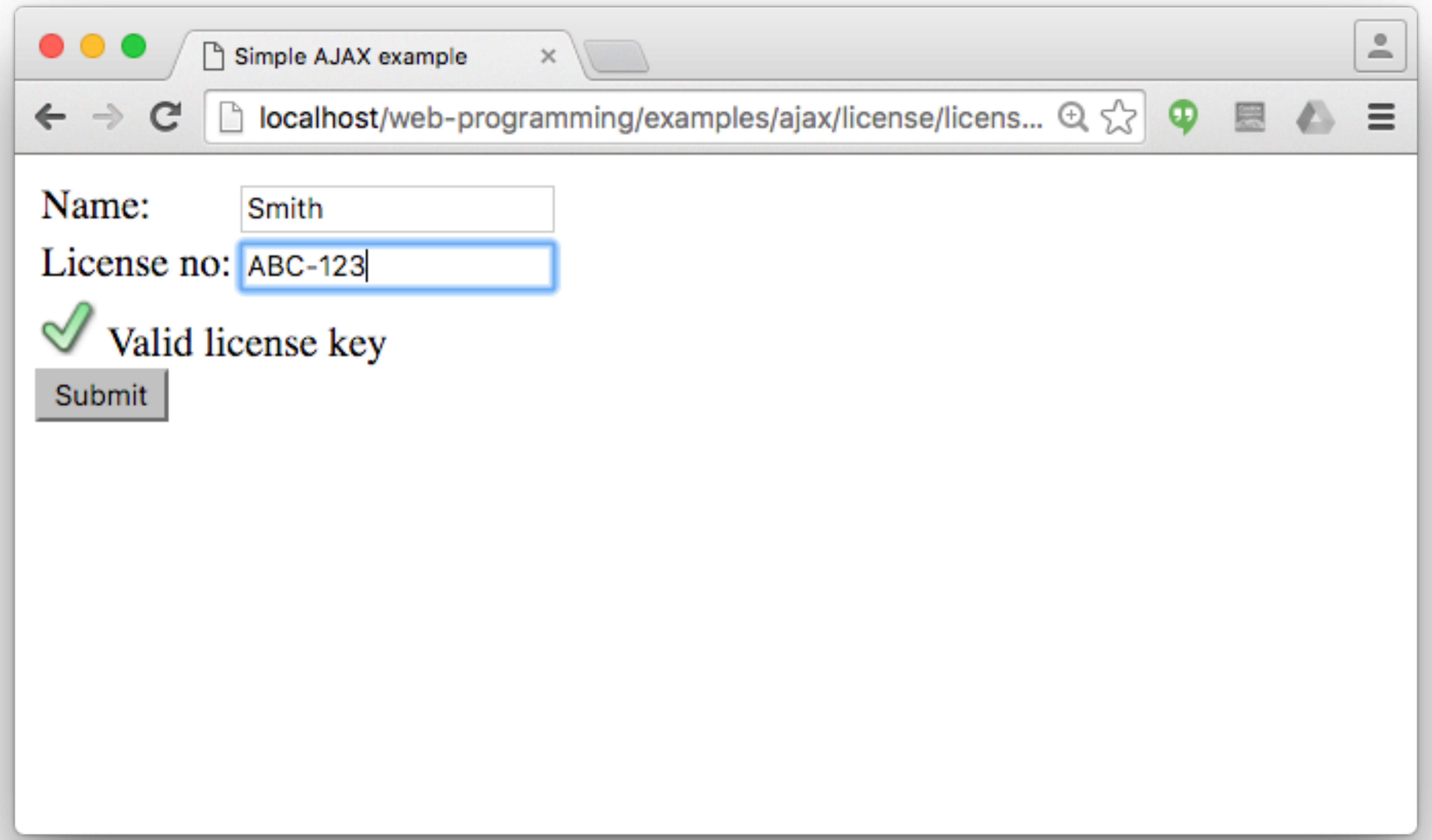

Simple AJAX example

localhost/web-programming/examples/ajax/license/licens...

Name:	Smith
License no:	ABC-000

Invalid license key for Smith

Submit


Simple AJAX example

localhost/web-programming/examples/ajax/license/licens...

Name:	Smith
License no:	ABC-123

Valid license key

Submit

Example #2

- Request can be POST as well
- It is also possible for the server to send back a HTML snippet
- The client updates part of the page (i.e., the DOM) with the received snippet

Restructured POST function

license.js

```
function ajaxPOST(url, data, success) {
    var xhr = new XMLHttpRequest();
    /* register an embedded function as the handler */
    xhr.onreadystatechange = function () {
        /* readyState = 4 means that the response has been completed
         * status = 200 indicates that the request was successfully completed */
        if (xhr.readyState == 4 && xhr.status == 200) {
            var result = xhr.responseText;
            success(result);
        }
    };
    /* send the request using POST */
    xhr.open("POST", url, true);
    /* To POST data like an HTML form, add an HTTP header */
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    /* variables go in the request body */
    xhr.send(data);
}
```

Restructured POST function

- **`ajaxPOST`** function contains no application logic

license.js

```
function ajaxPOST(url, data, success) {
    var xhr = new XMLHttpRequest();
    /* register an embedded function as the handler */
    xhr.onreadystatechange = function () {
        /* readyState = 4 means that the response has been completed
         * status = 200 indicates that the request was successfully completed */
        if (xhr.readyState == 4 && xhr.status == 200) {
            var result = xhr.responseText;
            success(result);
        }
    };
    /* send the request using POST */
    xhr.open("POST", url, true);
    /* To POST data like an HTML form, add an HTTP header */
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    /* variables go in the request body */
    xhr.send(data);
}
```

Restructured POST

- **`ajaxPOST`** is used

```
function updateLicence(snippet){
    document.getElementById("license_check").innerHTML = snippet;
}
```

Callback function

```
function checkLicense() {
    var name = document.getElementById("name").value;
    var license = document.getElementById("license").value;
    /* send the request if both name and license are filled in */
    if (name.length > 0 && license.length > 0) {

        let data = "name=" + name + "&license=" + license;

        ajaxPOST("/check_license", data, updateLicence);
    }
    else {
        updateLicence("");
    }
}
```

Format data

Process post

- Flask app generates a HTML snippet

app.py

```
@app.route("/check_license", methods=["POST"])
def check_license():
    VALID_LICENSES = {…}
    name = request.form.get("name", None)
    license = request.form.get("license", None)
    # check if name and license match
    if name and license:
        if VALID_LICENSES.get(name, None) == license:
            return "<img src='/static/images/yes.png' /> Valid license key"
        else:
            return "<img src='/static/images/no.png' />
                    Invalid license key for {}".format(name)
    return ""
```

Exercises #1, #1b

github.com/dat310-spring20/course-info/tree/master/
exercises/async

Fetch

- Perform AJAX call
- Returns a promise

```
let promise = fetch("/getplace?postcode=" + postcode);
```

Sends **`GET`** request if no additional arguments are given.

Promises

- Promises have initial state **“`pending`”**
- Returns a promise

```
let promise = fetch("/getplace?postcode=" + postcode);
```

Sends **`GET`** request if no additional arguments are given.

Promises

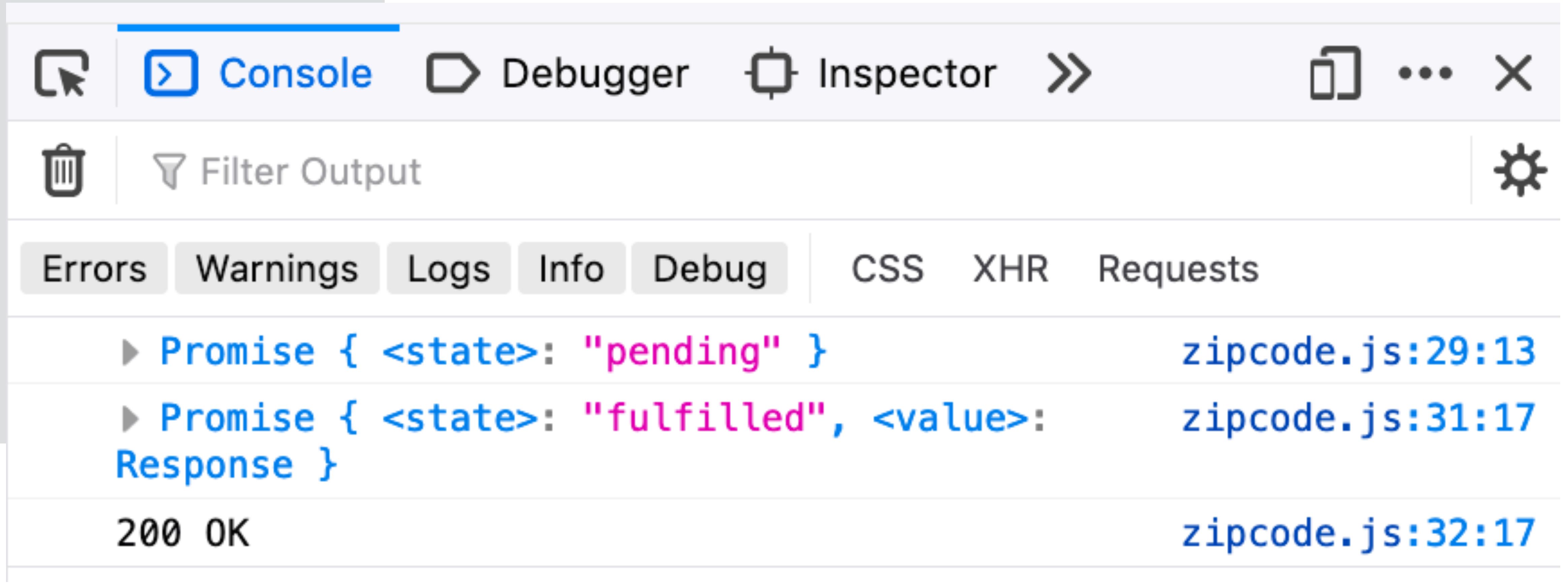
- Promises have initial state **“`pending`”**
- Can register a callback using **`.then(callback)`**

```
let promise = fetch("/getplace?postcode=" + postcode);

promise.then(function(response){ ... });
```

Promises

- Promises have initial state **“`pending`”**
- Callback is invoked when promise is **“`fulfilled`”**

```
let promise = fetch("/getplace?postcode=" + postcode);
console.log(promise);

promise.then(function(response){
        console.log(promise);
        console.log(response.status+ ' ' +
                    response.statusText);
    });
```



```
▸ Promise { <state>: "pending" }                          zipcode.js:29:13
▸ Promise { <state>: "fulfilled", <value>:                 zipcode.js:31:17
Response }
200 OK                                                     zipcode.js:32:17
```

Fetch response

- Access response text using **`response.text()`**
- **`response.text()`** returns another promise

```
let promise = fetch("/getplace?postcode=" + postcode);
let promise2 = promise.then(function(response){ return response.text(); })
```

`promise2` is the promise returned by **`response.text()`**

```
promise2.then(function(result){ updatePlace(result); });
```

`result` is the actual response text

Chaining promises

- Can omit declaring promises

```
// declared promises
let promise = fetch("/getplace?postcode=" + postcode);
let promise2 = promise.then(function(response){ return response.text(); });
promise2.then(function(result){ updatePlace(result); });
```

```
// without declaring promises
fetch(“/getplace?postcode=" + postcode)
    .then(function(response){ return response.text(); })
    .then(function(result){ updatePlace(result); });
```

Checking status

- Status of the response should always be checked

```
fetch(“/getplace?postcode=" + postcode)
    .then(function(response){
        // check if status code is success
        if (response.status == 200){
            return response.text();
        }
        // else return a default result
        return '';
    })
    .then(function(result){
        updatePlace(result); });
```

Fetch POST

- Fetch takes as second argument, an object

```
let data = "name=" + name + "&license=" + license;
fetch("/check_license",{
            method: "POST",
            headers: {
                "Content-Type": "application/x-www-form-urlencoded",
            },
            body: data,
        })
```

- Response is handled as with GET request.

```
    .then(function(response){
         // check if status code is success
         if (response.status == 200){
             return response.text();
         }
         // else return a default result
         return '';
    })
```

Async / await

- A different way to write promises and callbacks

```
let response = await fetch("/getplac?postcode=" + postcode);
```

Await waits until the promise is fulfilled.
Then response is assigned.

Async / await

with promises and then

```
fetch(“/getplace")
    .then(function(response){
        // check if status code is success
        if (response.status == 200){
             return response.text();
        }
        // else return a default result
        return '';
    })
    .then(function(result){
        updatePlace(result); });
```

with await

```
let response = await fetch("/getplace");
if (response.status == 200){
    let result = await response.text()
    updatePlace(result);
}
```

Async / await

- A function that contains **`await`** must be marked as **`async`**

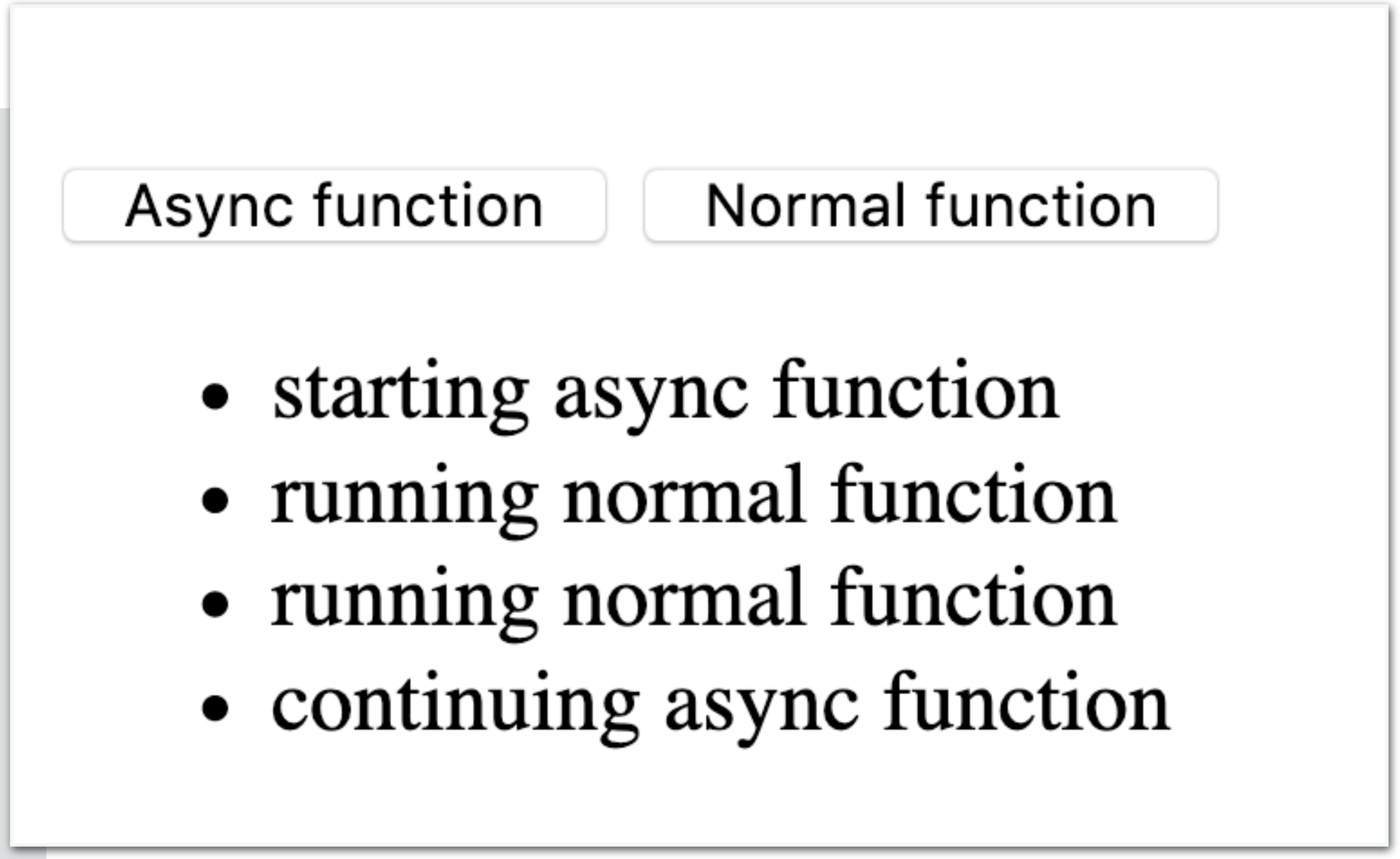
```
async function getPlace(postcode) {
    let uri = "/getplace?postcode=" + postcode;

    let response = await fetch(uri);
    if (response.status == 200){
        let result = await response.text()
        updatePlace(result);
    }
}
```

Example

examples/async/async-fetch

- An **`async`** stops when hitting an awaits and continues later.
  - Other event handlers can run in between

```
async function asyncFunction(){
    appendMessage("starting async function");
    await fetch("/delay");
    appendMessage("continuing async function");
}

function normalFunction(){
    appendMessage("running normal function");
}

function appendMessage(msg){
    let li = document.createElement('li');
    li.innerText = msg;
    document.getElementById("messages").appendChild(li);
}
```



- starting async function
- running normal function
- running normal function
- continuing async function

Exercises #2

github.com/dat310-spring20/course-info/tree/master/
exercises/async

JSON

- JavaScript Object Notation
- Lightweight data-interchange format
- Language independent
- Two structures
  - Collection of name-value pairs (object)
    - a.k.a. record, struct, dictionary, hash table, associative array
  - Ordered list of values (array)
    - a.k.a. vector, list

JSON

- Values can be
  - string (in between "…")
  - number
  - object
  - array
  - boolean (true/false)
  - null

Example JSON

```
{
 "name":"John Smith",
 "age":32,
 "married":true,
 "interests":[1,2,3],
 "other":{
         "city":"Stavanger",
         "postcode":4041
          }
}
```

JSON with Python

examples/ajax/json/json_python.py

- **`json`** is a standard module
- **`json.dumps(data)`**
  - returns JSON representation of the data
- **`json.loads(json_value)`**
  - decodes a JSON value
- **`json.dumps()`** and **`json.loads()`** work with strings
- **`json.dump()`** and **`json.load()`** work with file streams

JSON with JavaScript

examples/ajax/json/json_js.html

- `JSON.stringify(value)`
  - returns JSON representation of a value (encode)
- `JSON.parse(json)`
  - parses a JSON value into a JavaScript object (decode)

Example

examples/async/student

- Send JSON to server:

student.js

```
let student = { name: name, student_no: number};
fetch("/addStudent", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(student),
       });
```

app.py

```
@app.route("/addStudent", methods=["POST"])
def addStudents():
    student = request.get_json()
    if student.get(“name", "") != "":
        STUDENTS.append(student)
```

Example

examples/async/student

- Send JSON to browser:

student.js

```
async function getStudents(){
    let response = await fetch("/students");
    if (response.status == 200){
        let students = await response.json();
        showStudents(students)
    }
}
```

app.py

```
@app.route("/students", methods=["GET"])
def getStudents():
    sleep(1)
    return json.dumps(STUDENTS)
```

Exercises #3

github.com/dat310-spring20/course-info/tree/master/
exercises/async

References

- Mozilla Fetch reference
  https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- Mozilla Async/Await
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function