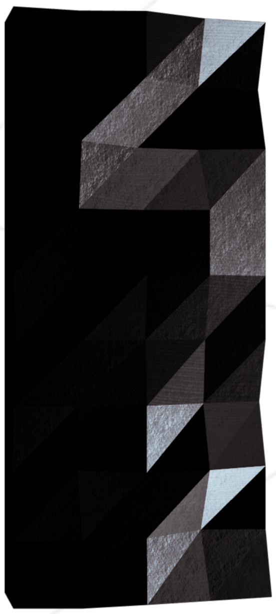


AP Computer Science Principles - Data Project

Data is Beautiful

Kai kai@42.us.org

Summary: Make your point by laying the facts out visually.

Contents

Chapter I

Introduction

I.1 Concepts

This assignment will proceed as a guided tutorial with discussion questions.

Consider three main topics in connection with this exercise:

- **Processing data**: Data is often stored in text files with simple formatting such as comma separated values (CSV) or javascript object notation (JSON). You can write code to loop through the lines of these files and extract the data.

  A CSV is like a spreadsheet, but instead of being visually seperated in boxes, each item is seperated by a comma. You can export spreadsheets from Google Sheets or Microsoft Excel as a CSV.

- **Application Program Interface (API)**: A way to send information back and forth from internet-connected databases.

- **Evaluating Hypotheses**: You must learn how to think scientifically about what conclusions you can be drawn from your data, and to understand when the data is insufficient to prove a hypothesis.

- **Displaying Data**: Have you ever stumbled upon an infographic on social media that made you pay more attention to an issue or change your mind about something you thought you knew? Artistic displays of data can be a more powerful way to explain your ideas than text or images alone. At the same time, if you are viewing someone else's data chart, you should be careful to fact check their sources and think logically about their conclusions.

I.2 Hypothesis

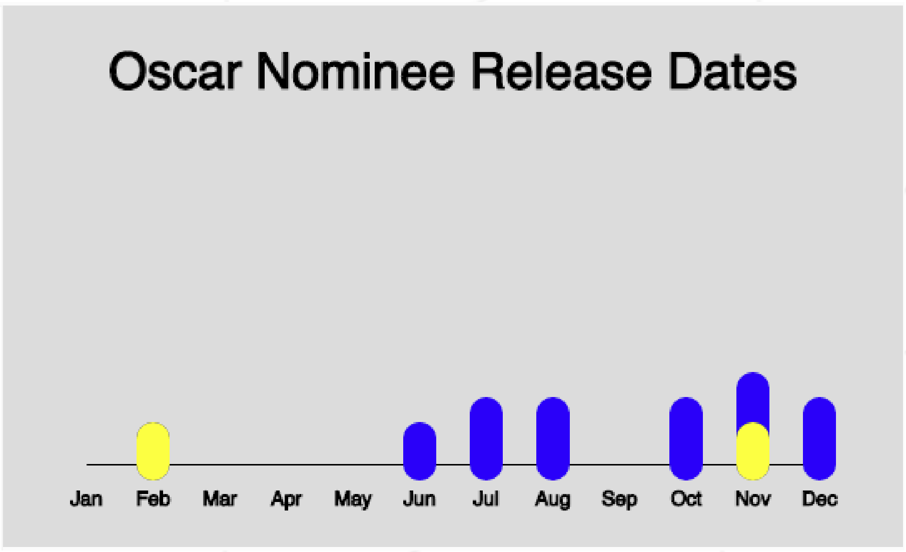
I will walk you through the process of collecting and visualizing data around a specific topic: Can a Release Date Predict an Oscar Winner?

In the article linked, author Zach Kram argues that which month a movie is released influences how likely it is to win an Oscar award.

Take some time to read the article. What do you think of his graphs and hypothesis? Do his visual data representations make the point of the article easy for you to understand?

In the tutorial below we will attempt to build a better visualization for his article. We will combine data from an API for The Movie Database (TMDB) and a CSV provided by Datahub.io. The visualization will show a bar chart counting how many Oscar nominees were released in each month of the year. Oscar winners will be graphed on the same axis in a different color than the other nominees.

As the basic learning experience we will graph the release date distribution for just one year. If you like this project and decide to go further, we can add a slider that allows the viewer to interact by changing the year displayed.



Chapter II

Learn how to Handle Data

Let's start by practicing two new coding techniques: Reading information from a CSV, and requesting information from an API.

We will combine the data from both sources together and store them in an object in our code.

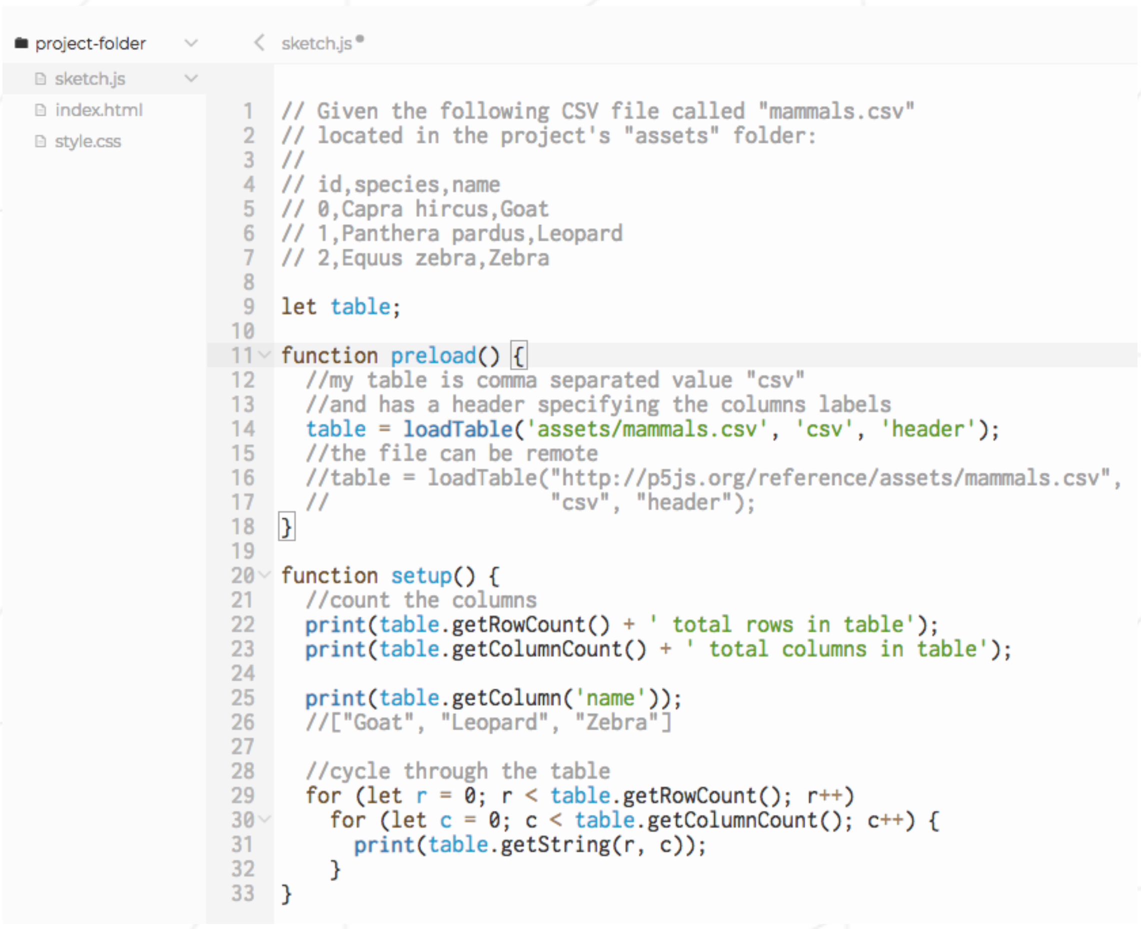
II.1 Read from a CSV

`Comma Seperated Values (CSV):` serves the same purpose as a spreadsheet, but it's displayed in plain text instead of a grid. Each "cell" is seperated by a comma. Each row is on a new line.

Look in the p5js reference for the secion on "IO", short for Input/Output. This will show you several ways to add data into your project (and, if you wanted to, how to export results).

We are going to use the loadtable() function to read a CSV file. Let's first make sure you are comfortable adding a file to the project and reading its contents.

Go ahead and open the reference page for loadTable(). Look at the example code; in fact, let's copy and paste the entire example onto a new blank sketch to see how it works.



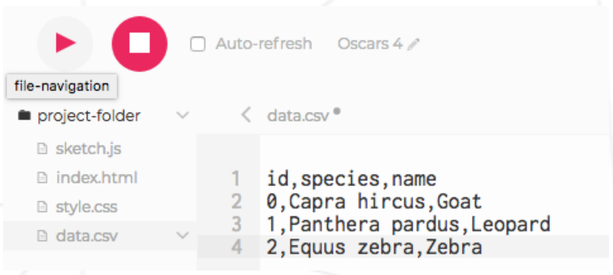
```
1  // Given the following CSV file called "mammals.csv"
2  // located in the project's "assets" folder:
3  //
4  // id,species,name
5  // 0,Capra hircus,Goat
6  // 1,Panthera pardus,Leopard
7  // 2,Equus zebra,Zebra
8
9  let table;
10
11 function preload() {
12   //my table is comma separated value "csv"
13   //and has a header specifying the columns labels
14   table = loadTable('assets/mammals.csv', 'csv', 'header');
15   //the file can be remote
16   //table = loadTable("http://p5js.org/reference/assets/mammals.csv",
17   //                  "csv", "header");
18 }
19
20 function setup() {
21   //count the columns
22   print(table.getRowCount() + ' total rows in table');
23   print(table.getColumnCount() + ' total columns in table');
24
25   print(table.getColumn('name'));
26   //["Goat", "Leopard", "Zebra"]
27
28   //cycle through the table
29   for (let r = 0; r < table.getRowCount(); r++)
30     for (let c = 0; c < table.getColumnCount(); c++) {
31       print(table.getString(r, c));
32     }
33 }
```

If you run the sketch at this point, it will do nothing except say "Loading..." forever. Why is that?

Notice the name of the CSV that it is trying to load on line 14. We need to create a CSV file in our project folder, and put the correct name of the CSV into that function.

Do you remember loading an image in APCSP part 05? Adding a CSV works just like that:

1. Find the arrow below the "play" button and to the left of "sketch.js". Click on it to expand the sidebar.
2. Click on the down arrow next to project-folder and choose "add file".
3. Call the file data.csv and press enter. Close the file explorer.
4. Copy the example data from lines 4-7 of the code and paste it into the data.csv file. Erase the comment marks.



```
id,species,name
0,Capra hircus,Goat
1,Panthera pardus,Leopard
2,Equus zebra,Zebra
```

Now, go back to the code in sketch.js and change "assets/mammals.csv" to "data.csv". Run the code again! Now you should get some console output from the results of the code in the setup() function. Read the code until you understand which line of code is producing which line(s) of output.

II.1.1 Graded Questions pt 1

Write down the answers to these questions:

1. Why is loadTable() called inside the preload() function? (Hint: Check the reference documentation for preload()!)
2. What code would you use if you wanted to print out only the 2nd item in the 2nd row of the CSV?

II.2 Query an API

`Application Program Interface (API)`: A set of commands used to request information from a database server.

`Server`: A computer whose purpose is to stay on and connected to the network so you can request information from it.

Save or erase your sketch so far and start with a new one. We will practice making an API call before combining the parts.

II.2.1 Register for API Key

The first thing you'll want to do is register for your own API key. This is like a password that you use to identify yourself to the API.

1. Make an account at https://www.themoviedb.org/. You'll need to go into your email and confirm your account.

2. (HELP!) - If you don't have access to your email, you can partner with a classmate and try sharing their API key.

3. Log into your TMDB account and click on your first initial in the top right corner; select Settings.

4. Click on the "API" section. Then, click the link under "Request an API Key". Choose the Developer option and agree to the terms of use.

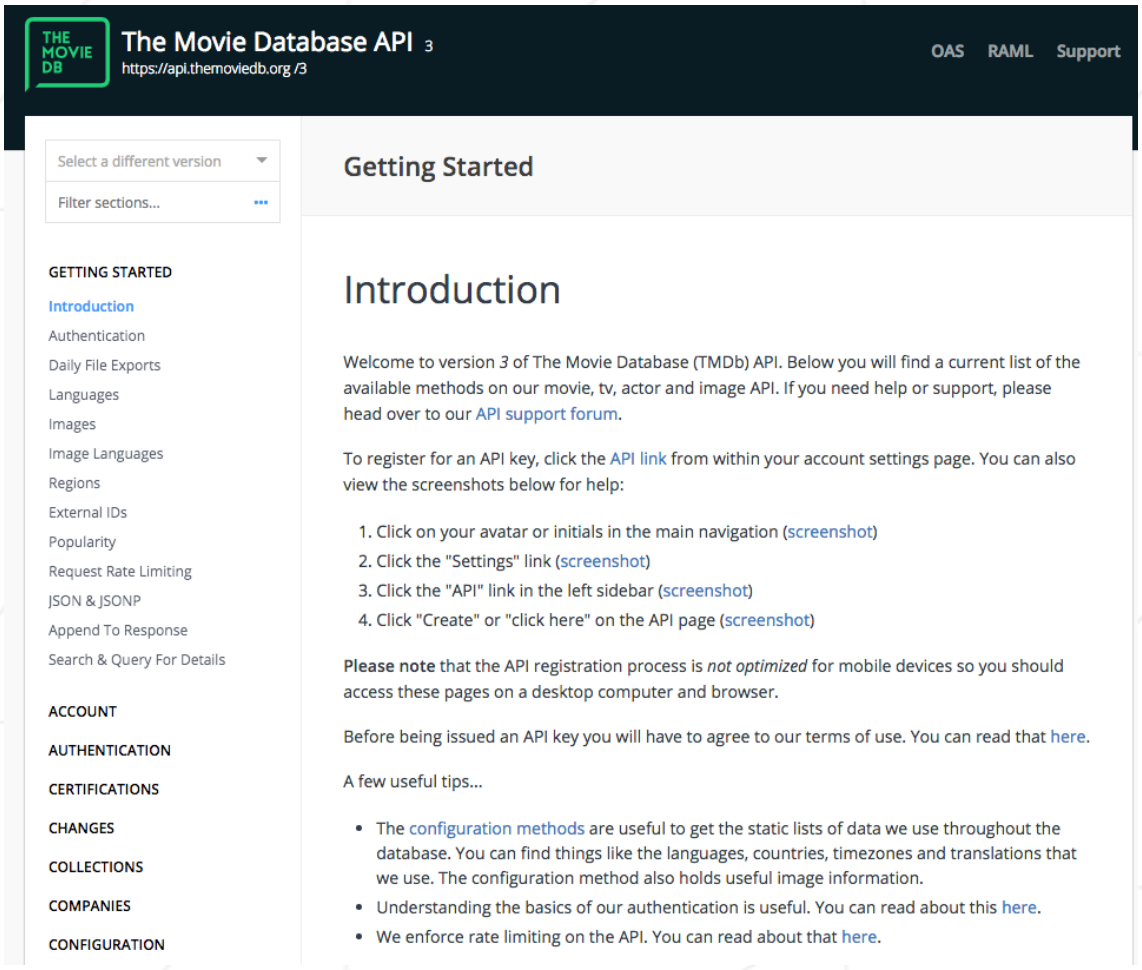
5. Fill out the form: Type of Use = Education. Application Name = APCSP Data Project. Application URL = codeforfun.org or paste a link to your p5js sketch. Application Summary = "Graphing release dates of Oscar nominees". Fill out your name and email. You can use the address of the school if you prefer: 6600 Dumbarton Circle, Fremont, CA 94555.

6. Now you have an API key! You have two of them, in fact. We will be using the Version 3 access key, so copy the shorter one (v3 auth).

7. Paste this API key as a comment at the top of your blank sketch. You'll need it later.

II.2.2 Learn from the Documentation

I'll take you on a brief tour of the steps I used to figure out how to use this API from p5js code. First, consult the API Documentation at developers.themoviedb.org.



Getting Started

Introduction

Welcome to version *3* of The Movie Database (TMDb) API. Below you will find a current list of the available methods on our movie, tv, actor and image API. If you need help or support, please head over to our API support forum.

To register for an API key, click the API link from within your account settings page. You can also view the screenshots below for help:

1. Click on your avatar or initials in the main navigation (screenshot)
2. Click the "Settings" link (screenshot)
3. Click the "API" link in the left sidebar (screenshot)
4. Click "Create" or "click here" on the API page (screenshot)

Please note that the API registration process is *not optimized* for mobile devices so you should access these pages on a desktop computer and browser.

Before being issued an API key you will have to agree to our terms of use. You can read that here.

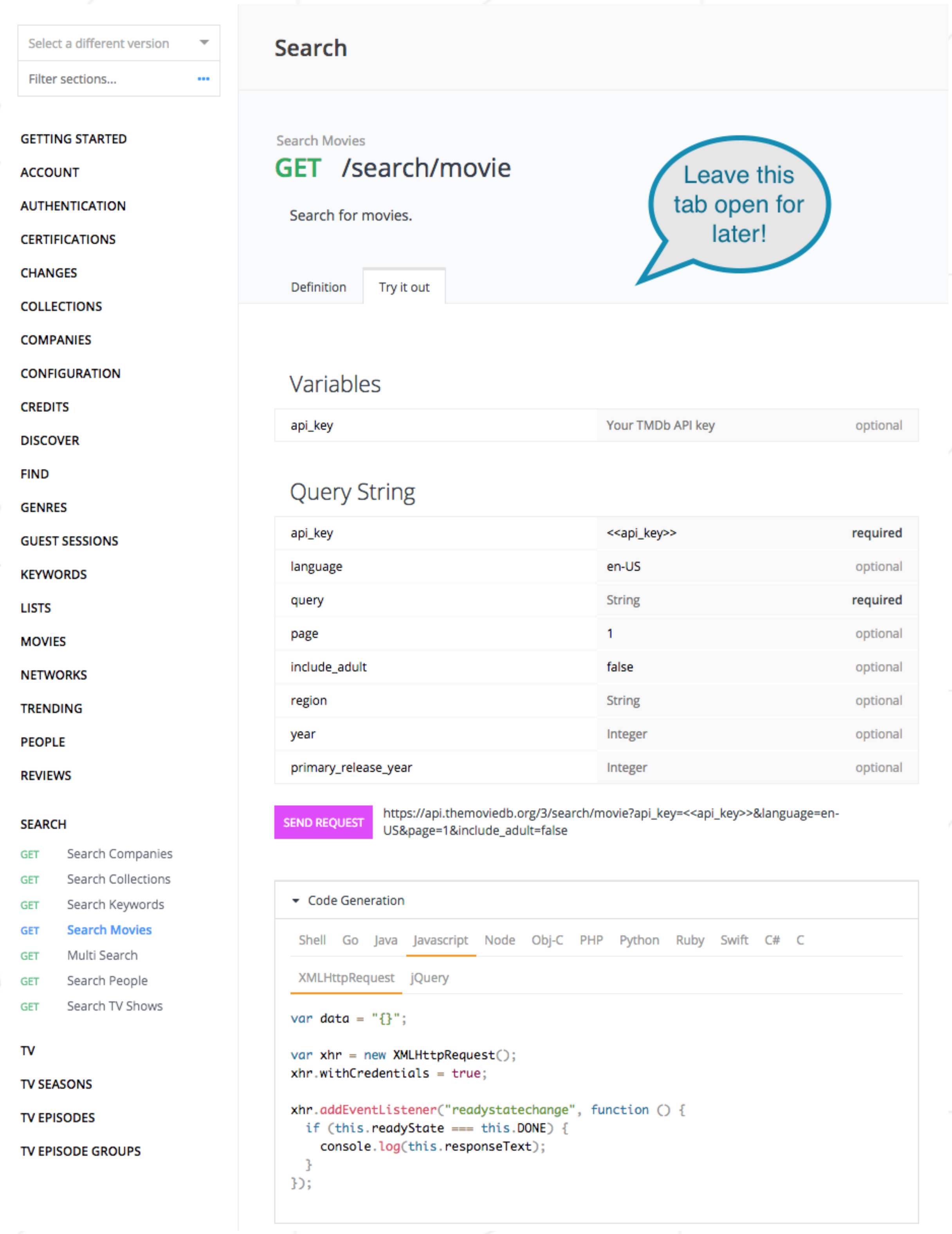
A few useful tips...

- The configuration methods are useful to get the static lists of data we use throughout the database. You can find things like the languages, countries, timezones and translations that we use. The configuration method also holds useful image information.
- Understanding the basics of our authentication is useful. You can read about this here.
- We enforce rate limiting on the API. You can read about that here.

The links on the left are mostly **endpoints**, indicating the type of information that the API has to offer. We are going to use the endpoint "Search".

Click on "SEARCH" in the sidebar and then, under Search, click "Search Moves". On this page the documentation will show you what parameters you can use, and what the response will look like.

Switch to the "Try it out" tab and expand the "Code Generation" section. Choose Javascript. Nice of them - they already wrote some code for us to use!


Select a different version
Filter sections...
Leave this tab open for later!
Definition
Try it out
SEND REQUEST
Code Generation
Shell Go Java Javascript Node Obj-C PHP Python Ruby Swift C# C
XMLHttpRequest jQuery

GETTING STARTED
ACCOUNT
AUTHENTICATION
CERTIFICATIONS
CHANGES
COLLECTIONS
COMPANIES
CONFIGURATION
CREDITS
DISCOVER
FIND
GENRES
GUEST SESSIONS
KEYWORDS
LISTS
MOVIES
NETWORKS
TRENDING
PEOPLE
REVIEWS

SEARCH
GET Search Companies
GET Search Collections
GET Search Keywords
GET Search Movies
GET Multi Search
GET Search People
GET Search TV Shows

TV
TV SEASONS
TV EPISODES
TV EPISODE GROUPS

Search

Search Movies

GET /search/movie

Search for movies.

Variables

api_key	Your TMDb API key	optional

Query String

api_key	<<api_key>>	required
language	en-US	optional
query	String	required
page	1	optional
include_adult	false	optional
region	String	optional
year	Integer	optional
primary_release_year	Integer	optional

https://api.themoviedb.org/3/search/movie?api_key=<<api_key>>&language=en-US&page=1&include_adult=false

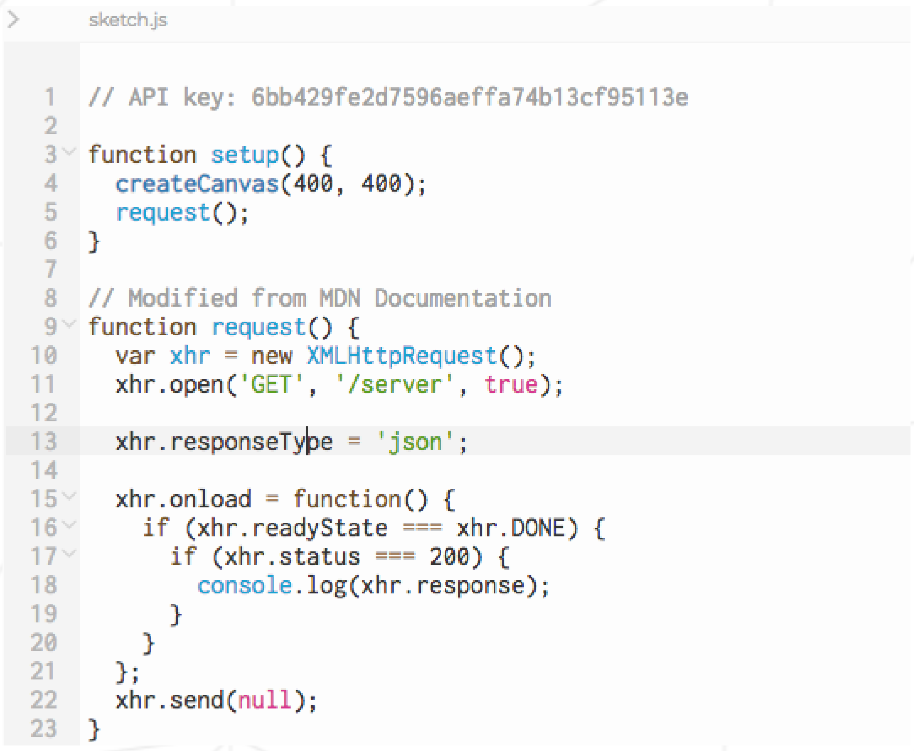
```
var data = "{}";

var xhr = new XMLHttpRequest();
xhr.withCredentials = true;

xhr.addEventListener("readystatechange", function () {
  if (this.readyState === this.DONE) {
    console.log(this.responseText);
  }
});
```

NOW, let's fastforward and say that when I tried using their example code I got a bunch of errors. I tried learning more about the function it suggested that we use, XML-HttpRequest. What worked for me was to go to the official Javascript documentation (Mozilla) and use one of their code examples for XMLHttpRequest(). You may also want to skim the main page of documentation on XMLHttpRequest, but don't worry about trying to understand all of it.

Copy the code example for XMLHttpRequest. Inside your blank p5js sketch, create a new function called request() and paste that code inside of it. Make two edits:

1. Change xhr.responseType to 'json' instead of 'text'
2. Delete the line where it prints out xhr.responseText.

Then, add a call to request() from setup().



```
// API key: 6bb429fe2d7596aeffa74b13cf95113e

function setup() {
  createCanvas(400, 400);
  request();
}

// Modified from MDN Documentation
function request() {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', '/server', true);

  xhr.responseType = 'json';

  xhr.onload = function() {
    if (xhr.readyState === xhr.DONE) {
      if (xhr.status === 200) {
        console.log(xhr.response);
      }
    }
  };
  xhr.send(null);
}
```

We are going to edit the "/server" parameter and replace it with a long link describing the web address of the database, plus the endpoint that we want to pull from, the parameters we will use, and our API key.

Go back to the tab where you are logged into TMDB, or click here. Make sure you are in the "Try it out" tab. Find your API key from earlier and paste it in the "Variables" section where it says "Your TMDb API key".

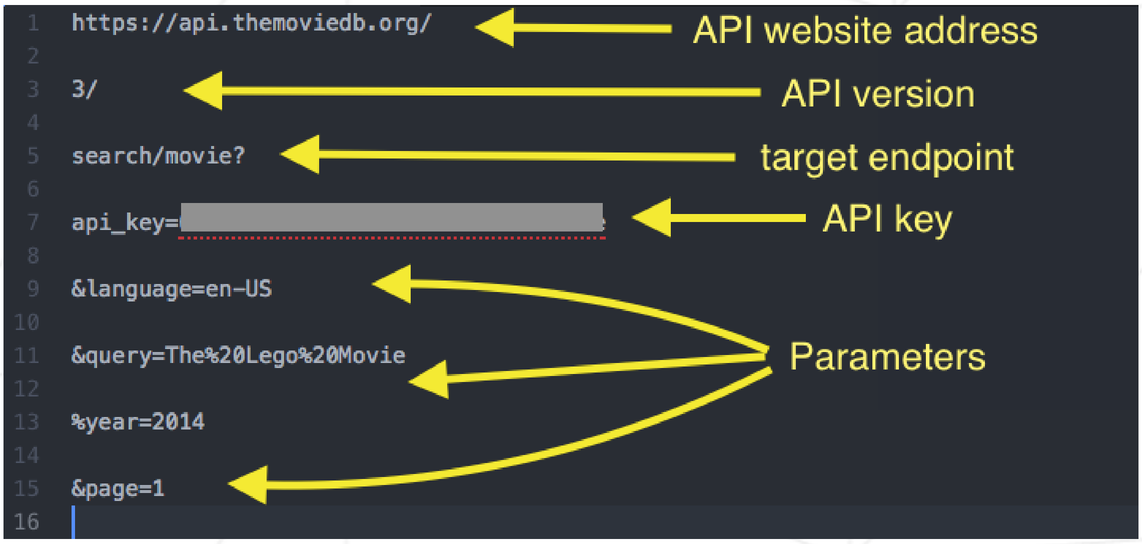
For the "query" parameter, type the name of a movie that you like.

Click "Send Request" and look at the results. You might have one or more results. In JSON format (which is basically a lot of arrays and objects packed together), the results tell you information such as: its popularity score, original language, plot summary, and... release date!

If you see multiple movies listed in the results, try filling in the "year" parameter for the actual year of the movie you were looking for. Run the query again. Did that help reduce the number of hits?

OK, now copy the long link next to the "SEND REQUEST" button, starting with https://api.themoviedb.org/3/search/movie?....

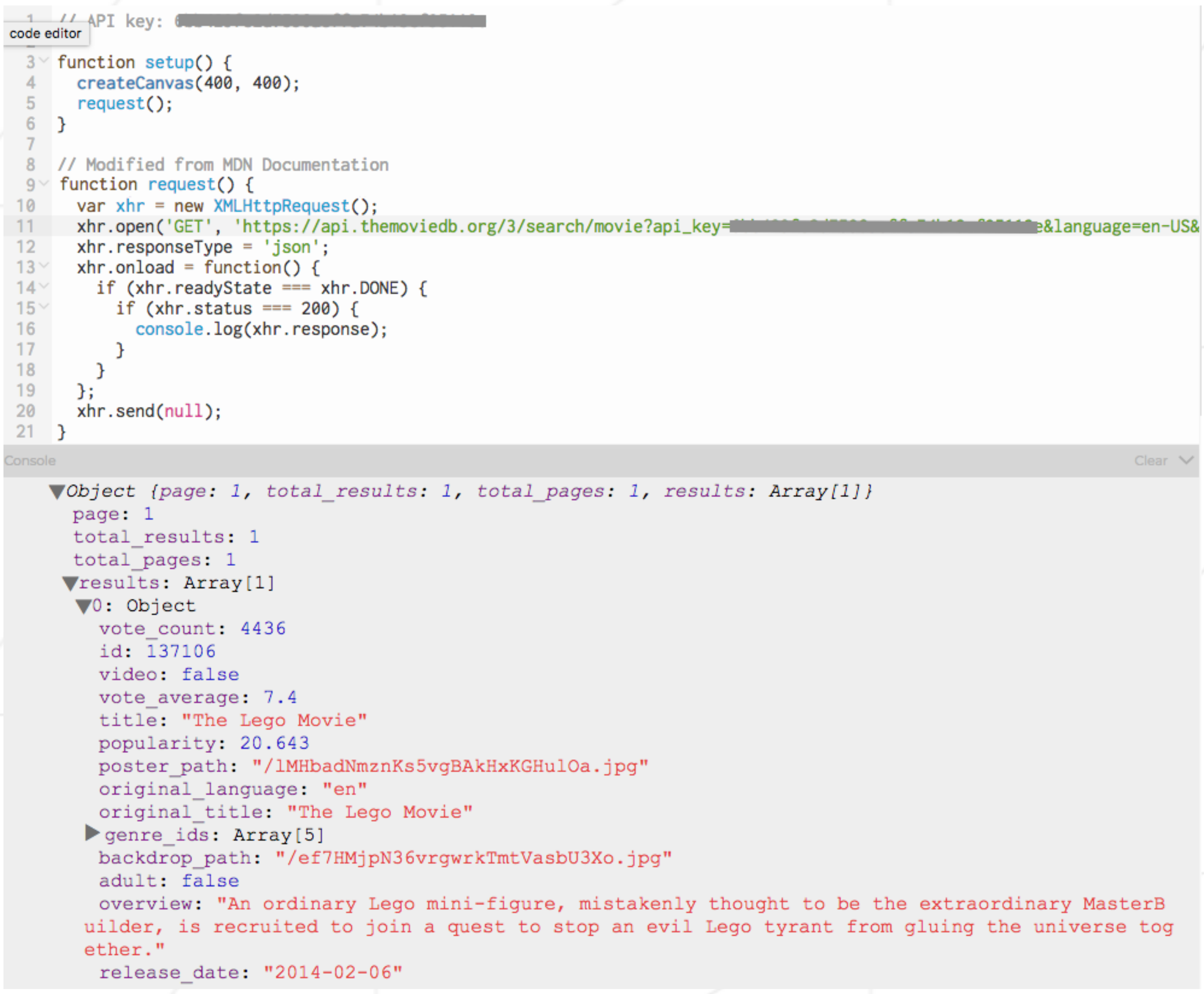
Notice how this link is made up of different logical parts.



```
https://api.themoviedb.org/

3/

search/movie?

api_key=[redacted]

&language=en-US

&query=The%20Lego%20Movie

%year=2014

&page=1

```

Back in your p5js editor, paste that link inside xhr.open() to replace '/server'. Make sure you wrap the link in quotation marks.

Place a call to request() inside of setup(). Then, run the code and see what it outputs!



```
// API key: [redacted]

function setup() {
  createCanvas(400, 400);
  request();
}

// Modified from MDN Documentation
function request() {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', 'https://api.themoviedb.org/3/search/movie?api_key=[redacted]e&language=en-US&
  xhr.responseType = 'json';
  xhr.onload = function() {
    if (xhr.readyState === xhr.DONE) {
      if (xhr.status === 200) {
        console.log(xhr.response);
      }
    }
  };
  xhr.send(null);
}
```

```
▼Object {page: 1, total_results: 1, total_pages: 1, results: Array[1]}
   page: 1
   total_results: 1
   total_pages: 1
  ▼results: Array[1]
   ▼0: Object
     vote_count: 4436
     id: 137106
     video: false
     vote_average: 7.4
     title: "The Lego Movie"
     popularity: 20.643
     poster_path: "/lMHbadNmznKs5vgBAkHxKGHulOa.jpg"
     original_language: "en"
     original_title: "The Lego Movie"
    ▶genre_ids: Array[5]
     backdrop_path: "/ef7HMjpN36vrgwrkTmtVasbU3Xo.jpg"
     adult: false
     overview: "An ordinary Lego mini-figure, mistakenly thought to be the extraordinary MasterB
   uilder, is recruited to join a quest to stop an evil Lego tyrant from gluing the universe tog
   ether."
     release_date: "2014-02-06"
```

II.2.3 Graded Questions pt 2

1. Which variable in your API query code contains the main results of your query? What data type is that variable?

2. If you wanted to add another parameter to the query string without using the TMDB API docs, how would you do it? Say you have this query string and you want to also specify that "director" should be "Miyazaki".

```
https://api.themoviedb.org/3/search/multi?api_key=
6bb429redactedredactedcf95113e&genre=anime&language
=en-US&page=1
```

II.3 Put them together

II.3.1 Loop through CSV Again

OK, it's time to revisit your CSV reading abilities, use them on a CSV that contains movie info, and search for the release date of that movie in the API. Still with me? Let's go. Here is a CSV with the nominees for two main categories in 2018.

```
year,category,winner,entity
2018,BEST PICTURE,False,Black Panther
2018,BEST PICTURE,False,BlacKkKlansman
2018,BEST PICTURE,False,Bohemian Rhapsody
2018,BEST PICTURE,False,Vice
2018,BEST PICTURE,False,The Favourite
2018,BEST PICTURE,False,Green Book
2018,BEST PICTURE,False,Roma
2018,BEST PICTURE,False,A Star Is Born
2018,DIRECTING,False,Cold War
2018,DIRECTING,False,Roma
2018,DIRECTING,False,Vice
2018,DIRECTING,False,BlacKkKlansman
2018,DIRECTING,False,The Favourite
```

I have put it in the same format as the historical data, and assigned "False" to the "winner?" column for all. Go ahead and pick your favorite movie in each category and change that value to "True". Since we don't know the winners for this year we'll just make it up. Later, if you are interested enough to expand this project, we'll graph all the historical truth.

You can keep working on your sketch with the API code. Copy and paste this CSV data into a file on your project, like we did with the mammals example. Set up the loadTable() command using the preLoad() function that we discussed before. Then pull up the loadTable() reference page to remember which commands you can use to navigate the CSV data.

Write some code that does the following:

- Loops through each line of the CSV (yes, you might want to check that example again). (Hint, there's no need for a *double* for-loop.)
- On each line, it grabs the year and title and stores them as two variables.
- Ideally this would be in a new function called read_csv() that is called from the setup() function.
- Also, create a global variable for your apikey and store it there.

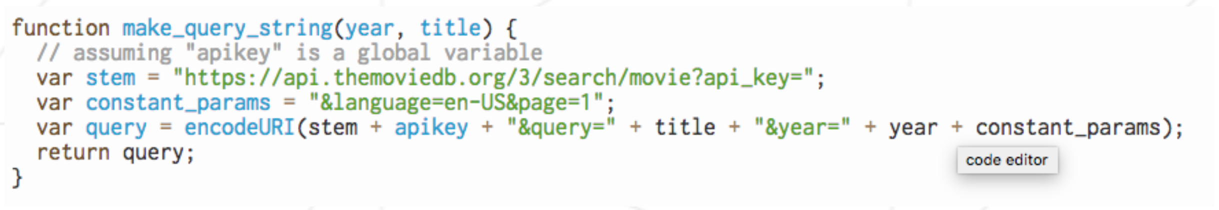
II.3.2 Build the Query String

I'll help you with this next part. The goal is to ask the TMDB database for the release date (not just the year, but the day and month) of each movie on the Oscar nominee list.

To do this, we need to generate a new query string for each nominee. We'll do that by adding (concatenating) the constant parts of the string together with the variable parts of the string.

You have used string concatenation before... in print statements, for debugging! This time we are going to save the result of "string addition" in a variable, instead of printing it out.

Oh, and as standard procedure you need to use the Javascript function encodeURI() to replace all the spaces in the request with percent signs.



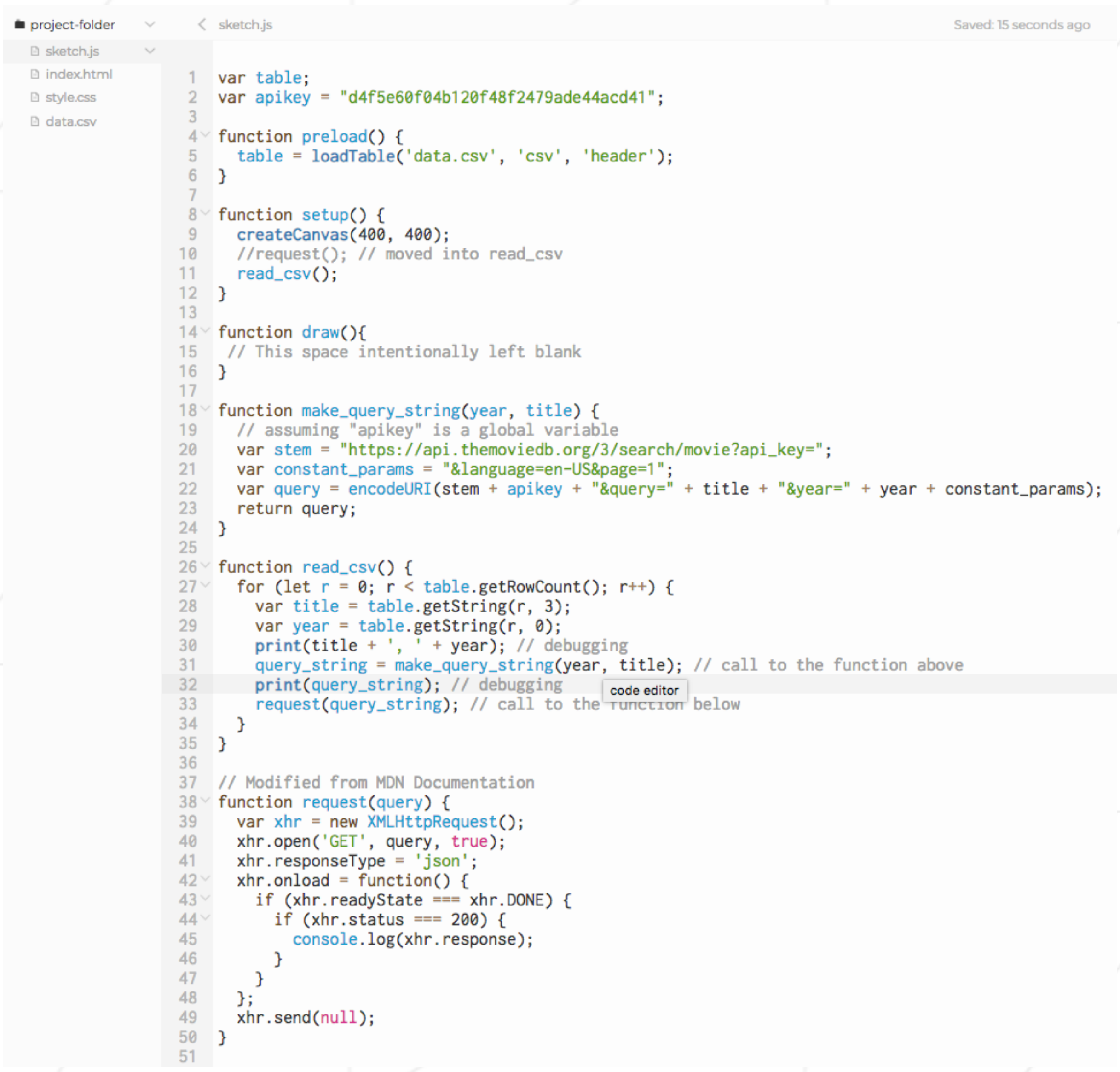
```
function make_query_string(year, title) {
  // assuming "apikey" is a global variable
  var stem = "https://api.themoviedb.org/3/search/movie?api_key=";
  var constant_params = "&language=en-US&page=1";
  var query = encodeURI(stem + apikey + "&query=" + title + "&year=" + year + constant_params);
  return query;
}
```

II.3.3 Use Abstraction on the Request function

Modify the request() function so that it takes in the query string as a parameter and uses that inside of xhr.open().

Modify read_csv() so that for each row, it generates a query string, and then calls request(query).

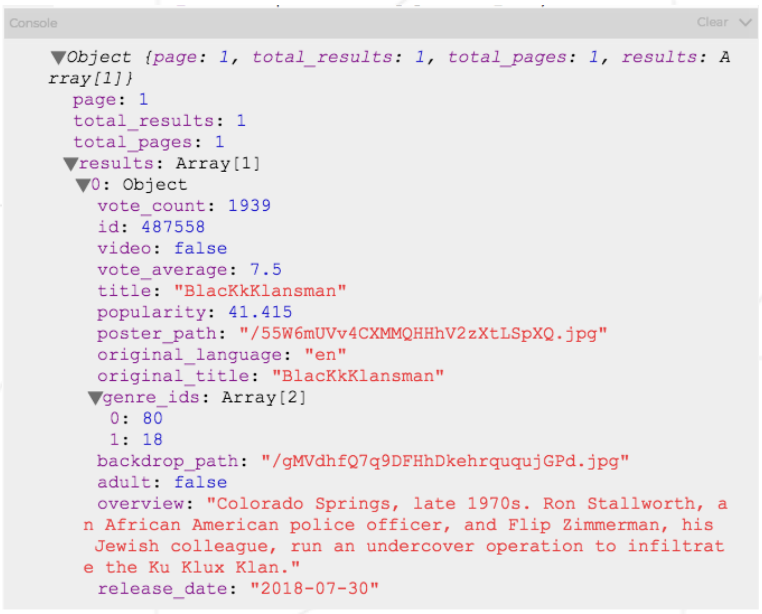
I'll let you look at my code if you need to catch up.



```
var table;
var apikey = "d4f5e60f04b120f48f2479ade44acd41";

function preload() {
  table = loadTable('data.csv', 'csv', 'header');
}

function setup() {
  createCanvas(400, 400);
  //request(); // moved into read_csv
  read_csv();
}

function draw(){
 // This space intentionally left blank
}

function make_query_string(year, title) {
  // assuming "apikey" is a global variable
  var stem = "https://api.themoviedb.org/3/search/movie?api_key=";
  var constant_params = "&language=en-US&page=1";
  var query = encodeURI(stem + apikey + "&query=" + title + "&year=" + year + constant_params);
  return query;
}

function read_csv() {
  for (let r = 0; r < table.getRowCount(); r++) {
    var title = table.getString(r, 3);
    var year = table.getString(r, 0);
    print(title + ', ' + year); // debugging
    query_string = make_query_string(year, title); // call to the function above
    print(query_string); // debugging
    request(query_string); // call to the function below
  }
}

// Modified from MDN Documentation
function request(query) {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', query, true);
  xhr.responseType = 'json';
  xhr.onload = function() {
    if (xhr.readyState === xhr.DONE) {
      if (xhr.status === 200) {
        console.log(xhr.response);
      }
    }
  };
  xhr.send(null);
}

```

II.4 Navigating the JSON Result

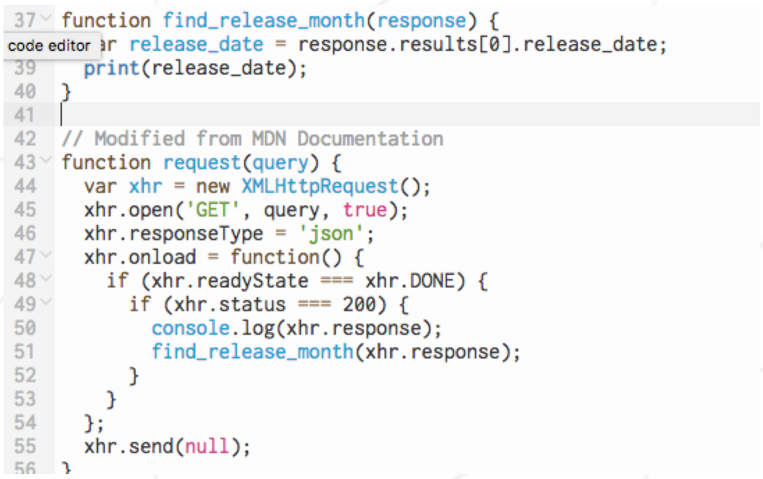
In the console, click the arrow next to one of the Object printouts to examine its contents.



```
▼Object {page: 1, total_results: 1, total_pages: 1, results: A
rray[1]}
   page: 1
   total_results: 1
   total_pages: 1
  ▼results: Array[1]
   ▼0: Object
     vote_count: 1939
     id: 487558
     video: false
     vote_average: 7.5
     title: "BlacKkKlansman"
     popularity: 41.415
     poster_path: "/55W6mUVv4CXMMQHHhV2zXtLSpXQ.jpg"
     original_language: "en"
     original_title: "BlacKkKlansman"
    ▼genre_ids: Array[2]
      0: 80
      1: 18
     backdrop_path: "/gMVdhfQ7q9DFHhDkehrququjGPd.jpg"
     adult: false
     overview: "Colorado Springs, late 1970s. Ron Stallworth, a
    n African American police officer, and Flip Zimmerman, his
     Jewish colleague, run an undercover operation to infiltrat
    e the Ku Klux Klan."
     release_date: "2018-07-30"
```

This is basically an **object** with many **variables** including "page" and "results". "results" is an array of objects. It could contain more than one movie, but it just has one in this example. Each movie in the result array is an object that has variables such as "vote_count", "id", and "video".

We want to pull out contents the "release_date" variable (at the bottom of the image). It's the first time we have worked with this type of data, so here's some code that will do it. Add the find_release_month function to your sketch and call it from inside the request() function, right after printing out the response.



```
function find_release_month(response) {
  ar release_date = response.results[0].release_date;
  print(release_date);
}

// Modified from MDN Documentation
function request(query) {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', query, true);
  xhr.responseType = 'json';
  xhr.onload = function() {
    if (xhr.readyState === xhr.DONE) {
      if (xhr.status === 200) {
        console.log(xhr.response);
        find_release_month(xhr.response);
      }
    }
  };
  xhr.send(null);
}
```

This code uses dot notation to reach inside of an object, and bracket notation to reach inside of an array. Look at what's happening:

- "response" is an object
- "response.results" is an array of objects

- "response.results[0]" is the first object in that array
- "response.results[0].release_date" is the variable labelled release_date on that object.

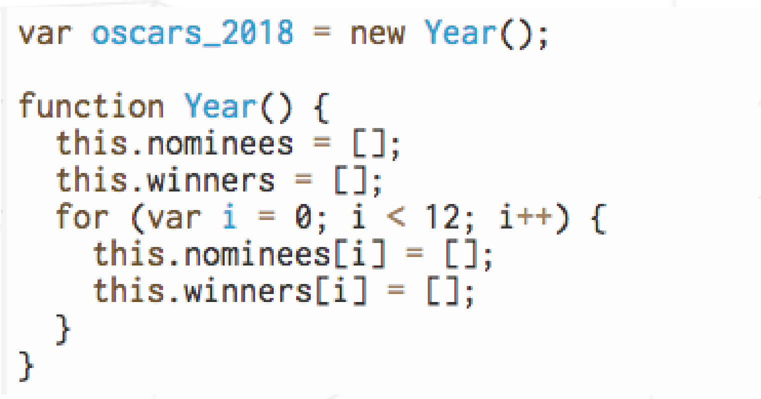
The result is in the format "YYYY-MM-DD", and we'll grab the month from it later.

II.5 Store Results

II.5.1 Plan the Data Structure

Similar to this Visualization of the results rolling two dice, I'm planning to draw a bar chart simply consisting of vertical lines along an axis.

We need to store the release dates and movie info into a format that makes it easy to draw a graph. There are lots of valid ways to do this, but after thinking about it a lot, for this class I decided to use the following structure:



```
var oscars_2018 = new Year();

function Year() {
  this.nominees = [];
  this.winners = [];
  for (var i = 0; i < 12; i++) {
    this.nominees[i] = [];
    this.winners[i] = [];
  }
}
```

The data for this year will be stored as one global variable, oscars_2018. That variable will be an object that is built from a constructor function. The constructor function is called Year(). Every Year object has two double arrays: one for all the nominees, and one for just the winners.

When a new Year object is created, the code loops through a counter for each month (1 to 12, or 0 to 11) and creates that many inner arrays in both the "winners" and "nominees" arrays.

As we go through the data I want to identify the release month and whether the movie was a winner or not. Then, I will copy the name of the movie into the corresponding spot in these double arrays.

Check Your Understanding: What information will be stored in oscars_2018.nominees[3]?

II.5.2 Sorting by Month

In order to fill up the data structure, let's add some logic to the find_release_month() function.

I want you to write this code yourself. Here it is in pseudocode:

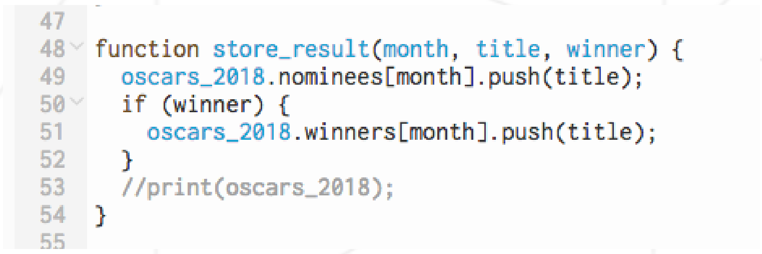
- Split the release date string by the "-" delimiter. (HINT: research how to use Javascript's string.split() function!)

- The second item in the resulting array represents the release month. Use the Javascript function parseInt() to transform it from a string to a number.

- This month is represented in the way humans count, from 1 to 12. Subtract one from the month number so that we can use it with our array of twelve arrays, which are numbered from 0 to 11.

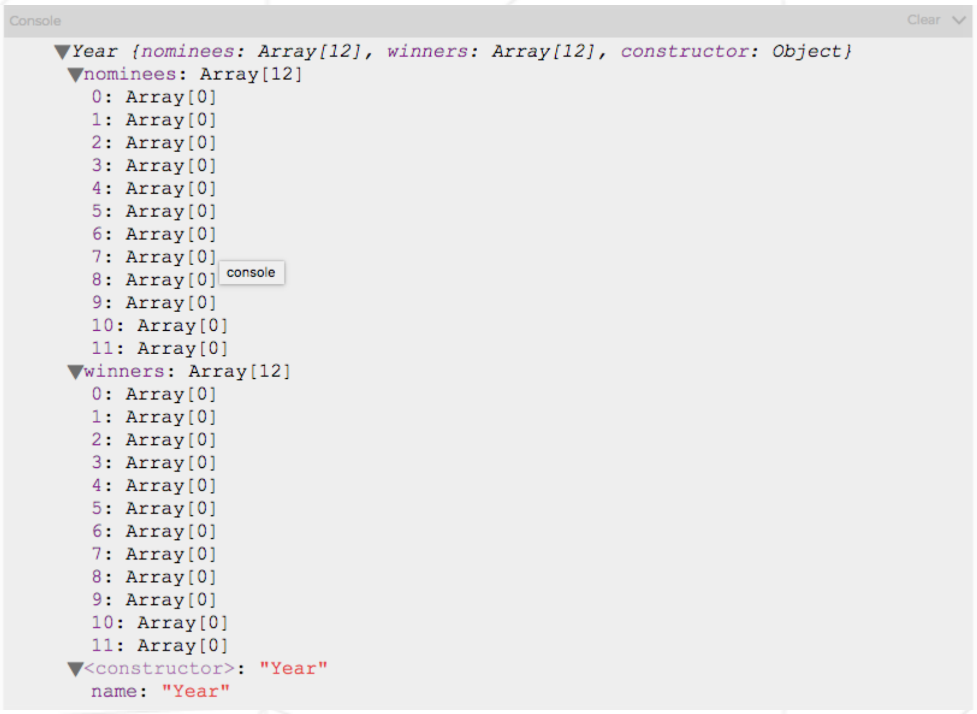
Can you take that psuedocode and turn it into real code? Use your brain. Type it in to the find_release_month() function.

Next I am going to create a new function, called store_result(month, title, winner). Here is the code for it:



```

function store_result(month, title, winner) {
  oscars_2018.nominees[month].push(title);
  if (winner) {
    oscars_2018.winners[month].push(title);
  }
  //print(oscars_2018);
}

```

Copy this function into your sketch and we'll take a minute to understand what it does. One thing that will help is to add the command `print(oscars_2018);` into your setup() function. This will print out a representation of the object, while all of its arrays are still empty:



```
▼Year {nominees: Array[12], winners: Array[12], constructor: Object}
 ▼nominees: Array[12]
   0: Array[0]
   1: Array[0]
   2: Array[0]
   3: Array[0]
   4: Array[0]
   5: Array[0]
   6: Array[0]
   7: Array[0]
   8: Array[0]
   9: Array[0]
   10: Array[0]
   11: Array[0]
 ▼winners: Array[12]
   0: Array[0]
   1: Array[0]
   2: Array[0]
   3: Array[0]
   4: Array[0]
   5: Array[0]
   6: Array[0]
   7: Array[0]
   8: Array[0]
   9: Array[0]
   10: Array[0]
   11: Array[0]
 ▼<constructor>: "Year"
   name: "Year"
```

The store_result function takes in three parameters: month (as a number), title (as a string), and winner (it should be a boolean). All movies will be added to the nominees array. oscars2018.nominees[n] takes us to the 'nth' array, which represents the 'nth' month - January = 0, February = 1, March = 2 and so on.

push() is a basic Javascript function which allows us to add an item into the chosen array.

Then, the "if(winner)" statement checks if the **winner** boolean is **true** and also adds this movie to the winner array if it is.

II.5.3 Keep Track of Who's a Winner

It's time to go back and fix a detail that we ignored so far: we need to extract the "winner" variable from the CSV, and pass it from function to function until it is used to sort films in our data object.

Trace the flow from one function to another in the code so far. What order do the functions run in?

1. preload() runs first (and variables are initiated about the same time, so Year() runs somewhere around here).
2. setup() calls read_csv().
3. read_csv() makes calls to make_query_string() and request().

4. make_query_string() runs and returns a value.

5. request() runs next, and makes a call to find_release_month().

6. store_data() hasn't been called by any code yet.

I suggest that we make the call to store_data() from find_release_month(). But, store_data() needs a parameter for the **winner**! Find_release_month has no such information available at the moment.

It does NOT make sense to store winner as a global variable, because it is different for every movie! We need to keep track of it alongside the other movie-specific information.

Please make the following changes to the code:

- Inside read_csv, add the line "var winner = (table.getString(r, 2) == 'True');". The "==" statement checks if the string from the table is equal to the string True. If that evaluates to True, then the variable "winner" will receive the True boolean. Otherwise, it is a False boolean.

- Modify the request() function definition so it takes two parameters now, query_string and winner. Send winner to request() when you call request().

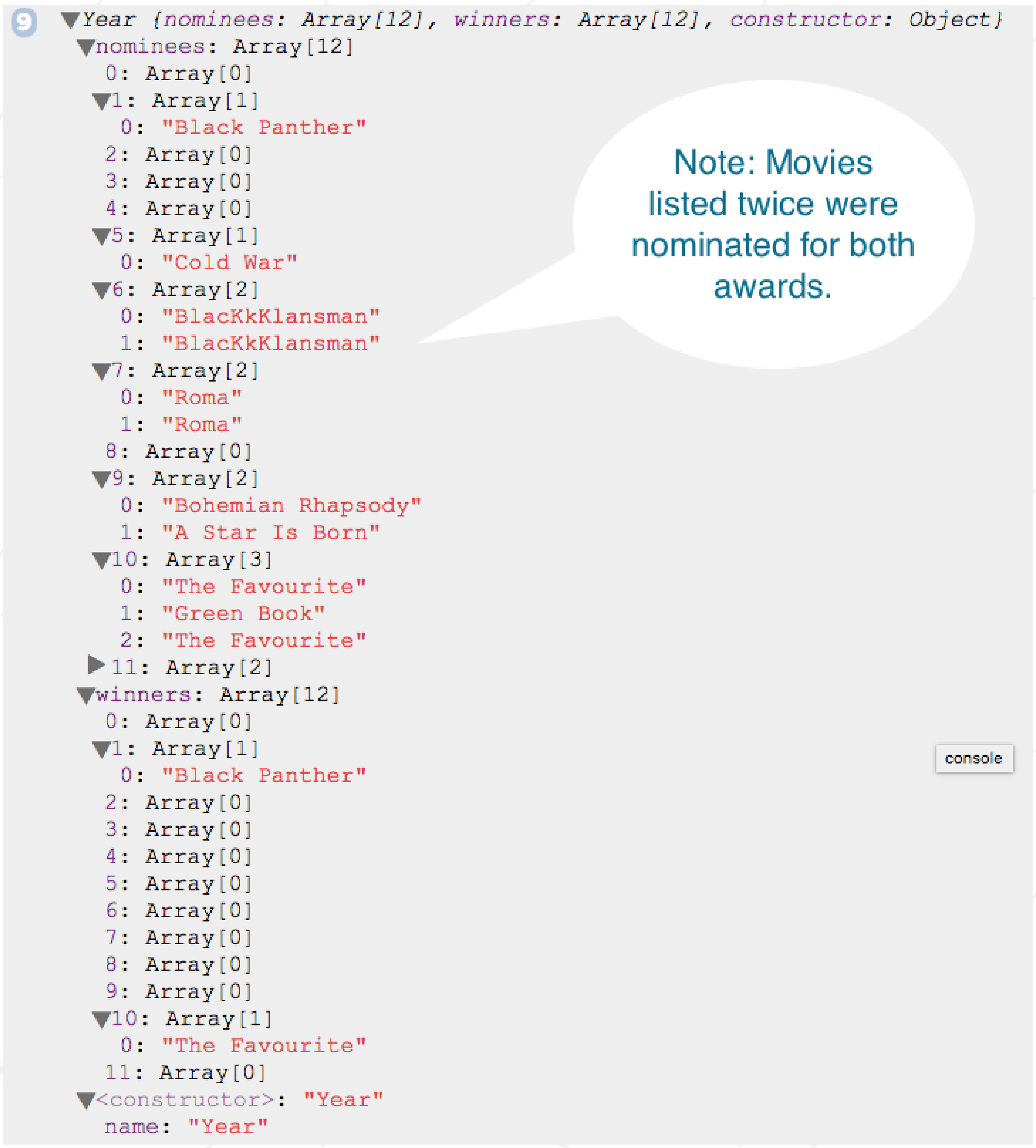
- Modify the find_release_month() function so that it takes two parameters, response and winner. Send winner to find_release_month() when you that function.

- Also, add a line to find_release_month() which extracts the title from the response object and saves it as a variable called "title". This one line of code will be very similar to the line of code that extracts the release date.

- Add a call to store_result(month, title, winner) at the end of find_release_month().

- Run the code and see if it works! If you are stuck, try debugging with print statements. Verify that it works properly by printing out the oscars_2018 variable at the end of store_result().

`There is an advanced topic hidden in the code here:` **`asynchronous Javascript.`** `Learn about it more when you have time. What it means for us today is that we cannot return a value from the request() function and jump back into the read_csv() algorithm. request() takes a relatively long time to run each time, and the other code in the program will not wait for it before moving on. This could result in the "response" variable being undefined. That's one reason I am telling you to write the functions in a specific way.`



```
▼Year {nominees: Array[12], winners: Array[12], constructor: Object}
  ▼nominees: Array[12]
     0: Array[0]
    ▼1: Array[1]
       0: "Black Panther"
     2: Array[0]
     3: Array[0]
     4: Array[0]
    ▼5: Array[1]
       0: "Cold War"
    ▼6: Array[2]
       0: "BlacKkKlansman"
       1: "BlacKkKlansman"
    ▼7: Array[2]
       0: "Roma"
       1: "Roma"
     8: Array[0]
    ▼9: Array[2]
       0: "Bohemian Rhapsody"
       1: "A Star Is Born"
    ▼10: Array[3]
       0: "The Favourite"
       1: "Green Book"
       2: "The Favourite"
    ▶11: Array[2]
  ▼winners: Array[12]
     0: Array[0]
    ▼1: Array[1]
       0: "Black Panther"
     2: Array[0]
     3: Array[0]
     4: Array[0]
     5: Array[0]
     6: Array[0]
     7: Array[0]
     8: Array[0]
     9: Array[0]
    ▼10: Array[1]
       0: "The Favourite"
     11: Array[0]
  ▼<constructor>: "Year"
     name: "Year"
```

Above is what the Oscars 2018 object will look like if you are successful so far!

Chapter III

Setting up a graph

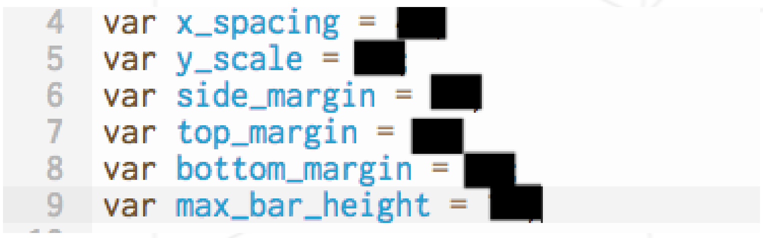
Similar to this Visualization of the results rolling two dice, we can draw a simple bar chart using just lines and the coordinate system. If you want to improve this graph to use rectangles for the bars, do it as a bonus at the end.

III.1 Draw an X-axis and Title

Plan out the size and shape of the graph: We will place 12 vertical bars, one for each month. Let's say they will be spaced 40 pixels apart with 50 pixels padding on each side of the graph. That means the x axis should start at x = ? and end at x = ?. Can you fill in the question marks?

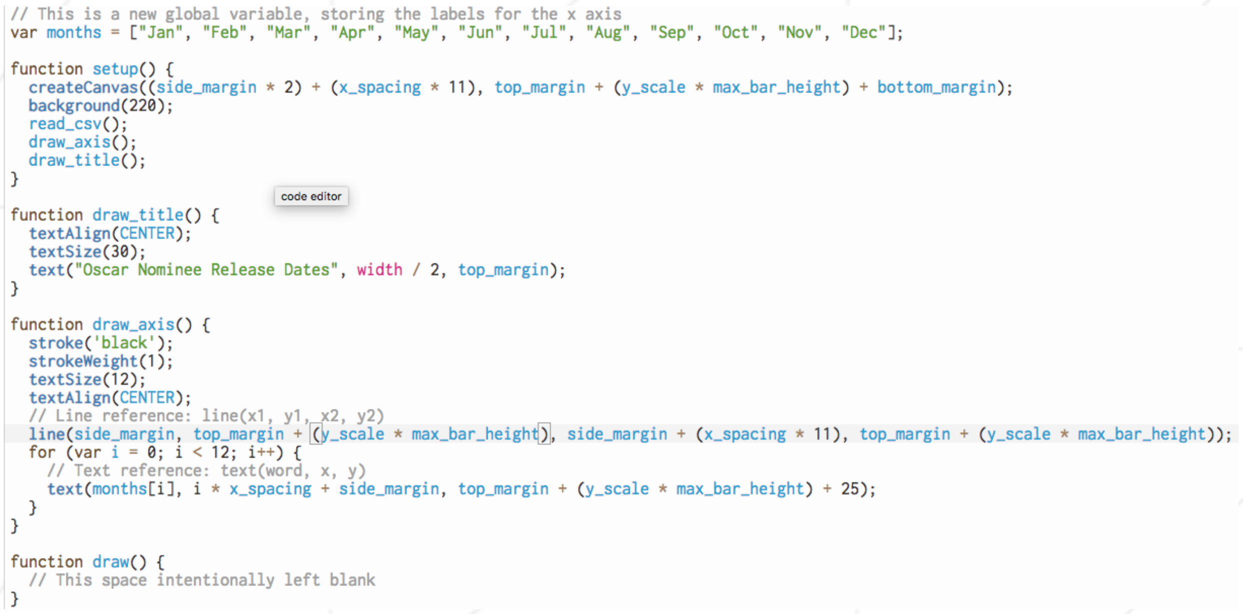
Vertically, how tall will the bars be? Right now we have data for Best Picture and Best Director. Let's say we will show them together without seperating them in the chart. Most categories always have 5 nominees, and the Best Picture can have 10 nominees. So we will be graphing 15 movies for each year if we use just these 2 categoreis. We should make space for a bar that is 15 units tall just in case, by some concidence, all nominees were released in the same month. Let's say that each vertical unit is <y_scale> pixels tall and there is padding of <top_margin> pixels on top for the title.

Create global variables that store your choices for the following values: spacing between bars, how many vertical pixels are used per movie counted, margin on each side of the graph, top margin pixels for the title, and bottom margin pixels for the labels.



```
var x_spacing = [redacted]
var y_scale = [redacted]
var side_margin = [redacted]
var top_margin = [redacted]
var bottom_margin = [redacted]
var max_bar_height = [redacted]
```

Then you can use the following code to create a canvas of the corresponding size and an x axis for labels. Notice that setup() is responsible both for triggering read_csv and for setting up the graph.



```
// This is a new global variable, storing the labels for the x axis
var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"];

function setup() {
  createCanvas((side_margin * 2) + (x_spacing * 11), top_margin + (y_scale * max_bar_height) + bottom_margin);
  background(220);
  read_csv();
  draw_axis();
  draw_title();
}

function draw_title() {
  textAlign(CENTER);
  textSize(30);
  text("Oscar Nominee Release Dates", width / 2, top_margin);
}

function draw_axis() {
  stroke('black');
  strokeWeight(1);
  textSize(12);
  textAlign(CENTER);
  // Line reference: line(x1, y1, x2, y2)
  line(side_margin, top_margin + (y_scale * max_bar_height), side_margin + (x_spacing * 11), top_margin + (y_scale * max_bar_height));
  for (var i = 0; i < 12; i++) {
    // Text reference: text(word, x, y)
    text(months[i], i * x_spacing + side_margin, top_margin + (y_scale * max_bar_height) + 25);
  }
}

function draw() {
  // This space intentionally left blank
}
```

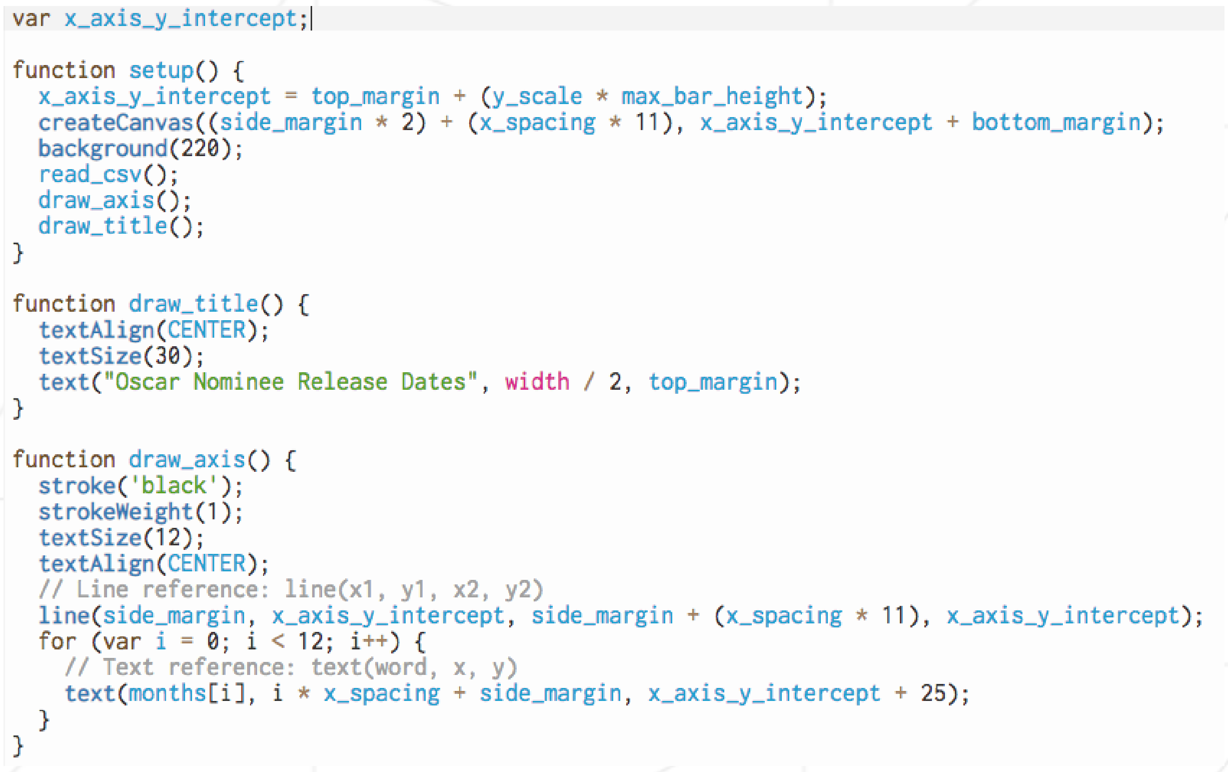
III.1.1 An Opportunity for Abstraction

Phew, that was a lot of code to type or copy over! Did you notice some parts that are repetitive? As I was writing those equations I found myself calculating the **y coordinate** of the **x axis** multiple times. You can clean up the code by storing that value in another global variable.

1. Add another global variable, var x_axis_y_intercept, but don't assign a value to it.

2. At the beginning of setup(), set that variable equal to the top_margin plus y_scale times max_bar_height. We are counting down from the top and that is the maximum bar height, in pixels, that could occur.

3. Replace every place you see top_margin + (y_scale * max_bar_height) with x_axis_y_intercept. It's used in creating the canvas and drawing the axis line.



```
var x_axis_y_intercept;

function setup() {
  x_axis_y_intercept = top_margin + (y_scale * max_bar_height);
  createCanvas((side_margin * 2) + (x_spacing * 11), x_axis_y_intercept + bottom_margin);
  background(220);
  read_csv();
  draw_axis();
  draw_title();
}

function draw_title() {
  textAlign(CENTER);
  textSize(30);
  text("Oscar Nominee Release Dates", width / 2, top_margin);
}

function draw_axis() {
  stroke('black');
  strokeWeight(1);
  textSize(12);
  textAlign(CENTER);
  // Line reference: line(x1, y1, x2, y2)
  line(side_margin, x_axis_y_intercept, side_margin + (x_spacing * 11), x_axis_y_intercept);
  for (var i = 0; i < 12; i++) {
    // Text reference: text(word, x, y)
    text(months[i], i * x_spacing + side_margin, x_axis_y_intercept + 25);
  }
}
```

III.2 Draw bars with the example data

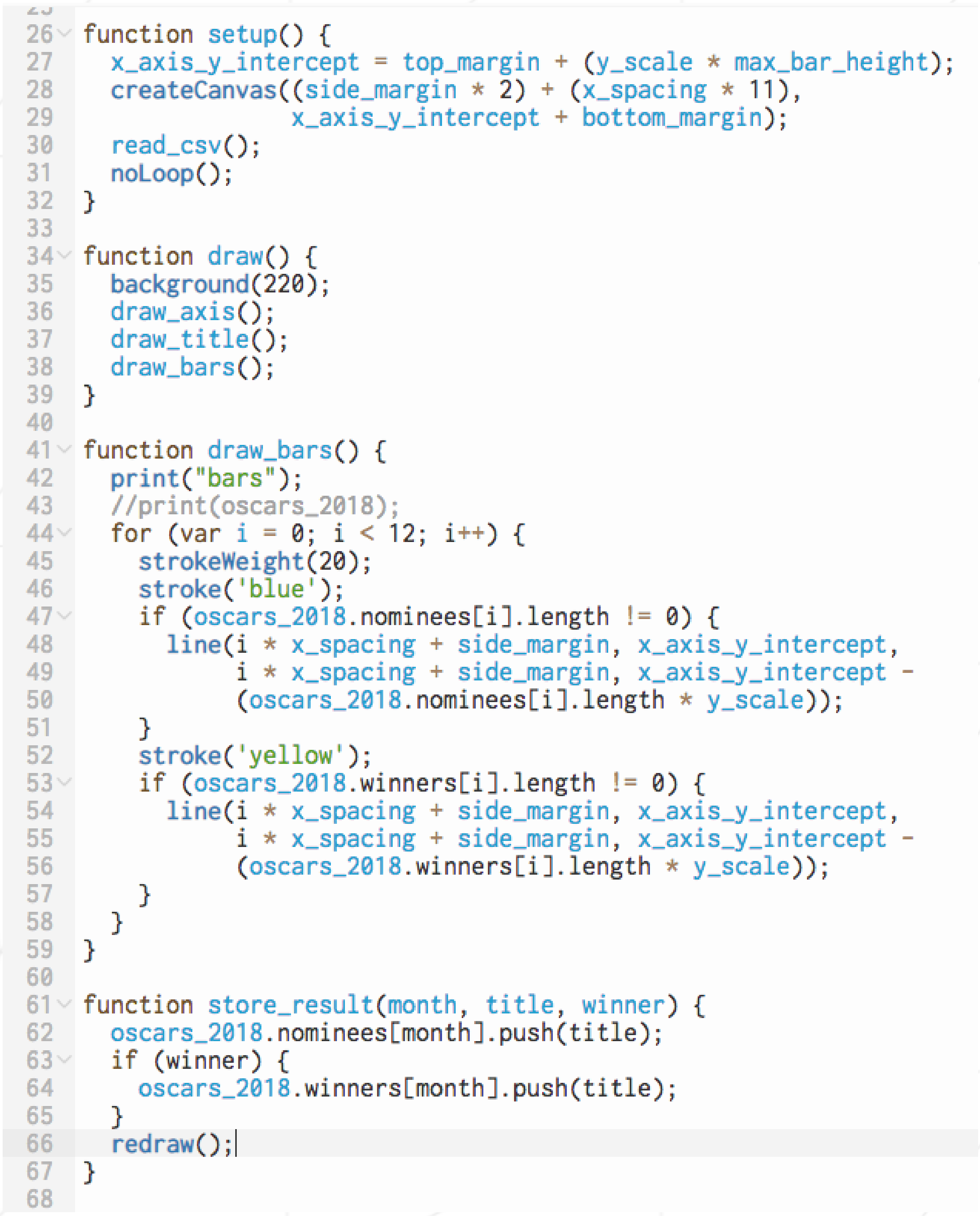
Now we'll create a function to draw the bars of the bar chart, and test it with the example data.

In pseudocode here is what I am going to do:

- Loop 12 times so I follow the same algorithm for each month:
- Check if there were any nominees released in the current month, and if there were, draw a blue line from 0 to the number of nominees.
- Check if there were any winners released in the current month, and if there were, draw a yellow line from 0 to the number of winners, on top of the previous blue line. This works because we will include winners in the count of nominees, storing the titles of winning films in both arrays.

You can type the following code to complete this part. Notice that the hard part is figuring out the Y coordinates of the line. Since 0 is at the top, we subtract the length of the bar from the Y coordinate of the X axis.

Another hard part is pausing the draw() loop and calling it at a specific time when you want the graph to be redrawn. Notice that I include noLoop() in setup (this will prevent the graph from drawing before the data is available), and I call redraw() every time that store_result updates the data. I moved the background, axis and title drawing steps from setup() to draw().



```
function setup() {
  x_axis_y_intercept = top_margin + (y_scale * max_bar_height);
  createCanvas((side_margin * 2) + (x_spacing * 11),
               x_axis_y_intercept + bottom_margin);
  read_csv();
  noLoop();
}

function draw() {
  background(220);
  draw_axis();
  draw_title();
  draw_bars();
}

function draw_bars() {
  print("bars");
  //print(oscars_2018);
  for (var i = 0; i < 12; i++) {
    strokeWeight(20);
    stroke('blue');
    if (oscars_2018.nominees[i].length != 0) {
      line(i * x_spacing + side_margin, x_axis_y_intercept,
           i * x_spacing + side_margin, x_axis_y_intercept -
           (oscars_2018.nominees[i].length * y_scale));
    }
    stroke('yellow');
    if (oscars_2018.winners[i].length != 0) {
      line(i * x_spacing + side_margin, x_axis_y_intercept,
           i * x_spacing + side_margin, x_axis_y_intercept -
           (oscars_2018.winners[i].length * y_scale));
    }
  }
}

function store_result(month, title, winner) {
  oscars_2018.nominees[month].push(title);
  if (winner) {
    oscars_2018.winners[month].push(title);
  }
  redraw();
}
```

Good Luck! Debug by thinking critically if you had any errors while typing it over! Feel free to ask your neighbor for help. Remember, the functions can be defined in the code in any order and it will work the same, so feel free cut and paste them into the positions that help you understand your code.

Chapter IV

Bonus: Multi-Year Slider

The bonus chapter of this assignment is a challenge:

Take the full CSV from datahub.io that I linked at the beginning of the PDF.

Creae a whole array of Year objects so that you can store multiple years worth of data.

Loop through the whole CSV and pull the information you are interested in.

Add a slider using the p5.dom library. In the draw function, check the value of slider.value() in order to decide which year's data to draw.