Student projects for *Modern JavaScript*

These projects let you practice the skills that you learn as you progress through *Murach’s Modern JavaScript*. They provide a range of difficulty levels. In the project names, the first number specifies the chapter that you should complete before starting the project. For example, you should complete chapter 3 before starting project 3-1 or 3-2.

[General Guidelines 2](#_Toc161670150)

[Project 2-1 Fahrenheit to Celsius 3](#_Toc161670151)

[Project 2-2 Tip Calculator 4](#_Toc161670152)

[Project 2-3 Travel Time Calculator 5](#_Toc161670153)

[Project 3-1 Fahrenheit to Celsius 6](#_Toc161670154)

[Project 3-2 Convert number grades to letter grades 7](#_Toc161670155)

[Project 3-3 Sum of Numbers 8](#_Toc161670156)

[Project 3-4 Fizz Buzz 9](#_Toc161670157)

[Project 4-1 Use a Sales array 10](#_Toc161670158)

[Project 4-2 Contact Viewer 11](#_Toc161670159)

[Project 4-3 Pig Latin Translator 13](#_Toc161670160)

[Project 5-1 Test Scores 14](#_Toc161670161)

[Project 5-2 Change Calculator 15](#_Toc161670162)

[Project 5-3 Password Generator 16](#_Toc161670163)

[Project 6-1 Temperature Converter 17](#_Toc161670164)

[Project 6-2 Test Scores 18](#_Toc161670165)

[Project 6-3 FAQs 19](#_Toc161670166)

[Project 6-4: Tuxedo Cat Coffee 20](#_Toc161670167)

[Project 8-1 Countdown 21](#_Toc161670168)

[Project 8-2 Calendar 22](#_Toc161670169)

[Project 9-1 Contact Info with HTML validation 23](#_Toc161670170)

[Project 9-2 Contact Info with custom validation 24](#_Toc161670171)

[Project 10-1 Website login 25](#_Toc161670172)

[Project 10-2 Contact Info 26](#_Toc161670173)

[Project 11-1 Contact Info 27](#_Toc161670174)

[Project 11-2 Student Scores 28](#_Toc161670175)

[Project 12-1 Change Calculator 29](#_Toc161670176)

[Project 12-2 Burger Town 30](#_Toc161670177)

[Project 13-1 Contact Info 31](#_Toc161670178)

[Project 13-2 Craps dice game 32](#_Toc161670179)

[Project 14-1 To-Do List 33](#_Toc161670180)

[Project 14-2 Mars Rover Photos 34](#_Toc161670181)

[Project 15-1 Create and run a server script with parameters 35](#_Toc161670182)

[Project 15-2 Run a server to host an app 36](#_Toc161670183)

[Project 16-1 Tic Tac Toe 37](#_Toc161670184)

[Project 16-2 Connect Four 38](#_Toc161670185)

# General Guidelines

Naming

* When creating the filenames for your programs, please use the convention specified by your instructor. Otherwise, store the files in a folder named first\_last\_app where first is your first name, last is your last name, and app is the name of the app.
* When creating names for variables and functions, please use the guidelines and recommendations specified by your instructor. Otherwise, use the guidelines and recommendations specified in Murach’s Modern Javascript.

User interfaces

* You should think of the user interfaces that are shown for the projects as starting points. If you can improve on them, especially to make them more user-friendly, by all means do so.

Specifications

* You should think of the specifications that are given for the projects as starting points. If you have the time to enhance the programs by improving on the starting specifications, by all means do so.

Files supplied by your instructor

* Some of the projects require starting HTML, CSS, or other files. These files are identified in the specifications for the projects, and your instructor should make these starting files available to you.

# Project 2-1 Fahrenheit to Celsius

Create an app that converts Fahrenheit temperatures to Celsius temperatures by using the prompt() method and the alert() method. The prompt dialog should look like this:

Screenshot:

Page title: This page says

Label: Enter Fahrenheit temperature
Textbox: shows 38 being entered

Buttons: OK, Cancel

After you display the results, the alert dialog should look like this:

Screenshot:

Page title: This page says

Label: 38.0 F = 3.3 C

Button: OK

Specifications

* Get input from the user using a prompt. Assume that the user will enter valid input.
* To convert Fahrenheit to Celsius, first subtract 32 from the Fahrenheit temperature. Then, multiply that result by 5/9.
* Display the result of the calculation in a dialog.

# Project 2-2 Tip Calculator

Create an app that calculates a tip amount and total given the cost of a meal and tip percent. Then, display the information as shown below.

Screenshot:

Page title: This page says

Label: Meal cost: $25.00
Tip percent: 20%
Tip amount: $5.00
Total cost: $30.00

Button: OK

Specifications

* The app should get the meal cost and a tip percent from the user.
* The app should display the meal cost provided by the user, the tip percent provided by the user, the calculated tip amount, and the calculated total cost as shown above.

# Project 2-3 Travel Time Calculator

Create an app that calculates the estimated hours and minutes for a trip and displays them as shown below.

Screenshot:

Page title: This page says

Labels: Distance: 250 miles
Speed: 65 MPG
Travel time: 3 hours, 55 minutes 

Button: OK

Specifications

* The program should get the miles traveled and the speed in miles per hour from the user.
* You can use the Math object to help you calculate the travel time.

# Project 3-1 Fahrenheit to Celsius

Update the Fahrenheit to Celsius app from project 2-1 to validate user entry and to allow the user to do multiple conversions before ending the app.

Screenshot:

Page title: This page says

Label: Enter Fahrenheit temperature
Or enter x to end app

Textbox: shows x being entered

Buttons: OK, Cancel

Screenshot:

Page title: This page says

Labels: You entered 1000
Entry must range from -100 to +212 

Button: OK

Specifications

* Add a loop to the code so the user can do a series of calculations without restarting the app. To end the app, the user must enter x as the temperature as shown in the first dialog above.
* Add data validation so the app won’t do the conversion until the user enters a Fahrenheit temperature between -100 and 212. If the entry is invalid, the app should display a dialog like the second one above.

# Project 3-2 Convert number grades to letter grades

Create an app that converts number grades to letter grades, as shown below.

Screenshot:

Page title: This page says

Label: Grade 92 = A

Button: OK

To derive the letter grade, you should use this table:

A 88-100  
B 80-87  
C 68-79  
D 60-67  
F < 60

Specifications

* The app should prompt the user for grades between 0 and 100 until the user enters x or clicks the Cancel button.
* Every time the user enters a grade, the app should display the corresponding letter grade as shown above.

# Project 3-3 Sum of Numbers

Create an app that sums all of the numbers between 1 and the number entered by the user, as shown below.

Screenshot:

Page title: This page says

Label: The sum of the numbers from 1 through 88 is 3916

Button: OK

Screenshot:

Page title: This page says

Label: Do another sum?
Textbox: shows y being entered

Buttons: OK, Cancel

Specifications

* The app should accept a number between 1 and 100. If the user enters an invalid number or a number outside of this range, the app should display an error.
* The app should sum all of the numbers between 1 and the entered number, display the results, and ask the user if they want to do another sum, as shown in the second dialog above.
* The app should ensure that the user does at least one calculation.

# Project 3-4 Fizz Buzz

Create an app based on the Fizz Buzz word game (also a famous interview question for programmers). The app should print to the console the word ‘Fizz’ for multiples of 3, ‘Buzz’ for multiples of 5, and ‘FizzBuzz’ for multiples of both 3 and 5. The app should do this for numbers 1 through 20 as shown below.

A screenshot of the console in Chrome.

1
2
3 Fizz
4
5 Buzz
6 Fizz
7
8
9 Fizz
10 Buzz
11
12 Fizz
13
14
15 FizzBuzz
16
17
18 Fizz
19
20 Buzz

Specifications

* The app should evaluate numbers 1 through 20.
* If the number is divisible by 3 and 5, print the number followed ‘FizzBuzz’.
* If the number is divisible by 3 only, print the number followed by ‘Fizz’.
* If the number is divisible by 5 only, print the number followed by ‘Buzz’.
* If the number is divisible by neither, print just the number.

# Project 4-1 Use a Sales array

In this project, you’ll start with five arrays that represent sales regions. Each array contains four values that represent the quarterly sales for the region. Then, you’ll summarize the data for the regions, which should look like this:

Screenshot:

Page title: This page says

Labels: Sales by Quarter
Q1: $9965
Q2: $7403
Q3: $9478
Q4: $13061

Annual Sales
$39907 

Button: OK

Specifications

* Use the index.html file in the following folder

projects\ch04\sales\_arrays\

* The data for this app is stored in arrays at the top of the <script> element. Each element in each array represents the sales for one quarter. For example, the first element in the region1 array represents the sales for Q1 for region 1.
* The displayed sales for Q1 should combine the Q1 sales for each of the five regions, the displayed sales for Q2 should combine the Q2 sales for each of the five regions, and so on.

# Project 4-2 Contact Viewer

In this project, you’ll start with an array of strings that represent contacts. Then, you’ll create an app that allows you to view the data for the contacts. The dialogs below show the interface for this app.

Screenshot:

Page title: This page says

Label: COMMAND MENU
list - List all contacts
get # - Get contact with the specified number
exit - Exit program

Textbox: shows "exit" being entered

Buttons: OK, Cancel

User entered ‘list’

Screenshot:

Page title: This page says

Label: 1 - Scott
2 - Joel
3 - Mile

Button: OK

User entered ‘get 1’

Screenshot:

Page title: This page says

Label: Contact info for Scott
Email: scott@murach.com
Phone: 1-599-555-5555

Button: OK

User entered ‘get 15’

Screenshot:

Page title: This page says

Label: No data for #15

Button: OK

## Project 4-2 Contact Viewer (continued)

User entered ‘show 1’

Screenshot:

Page title: This page says

Label: Invalid command

Button: OK

Specifications

* Use the contact.js file in the following folder

projects\ch04\contact\_viewer\

* The data for this app is stored in an array at the top of the script. Each element in the array is a string that stores values for a contact separated by pipe characters (**|**).
* The app should display a menu to show the available commands.
* The app should accept commands until the user enters the exit command or clicks Cancel.
* The list command should display all users in the contacts array.
* The get command should display the contact info for the specified contact, or a message that the contact isn’t found.
* Any other command should display an ‘invalid command’ message.

# Project 4-3 Pig Latin Translator

Create an app that takes input from the user and translates it to Pig Latin.

Screenshot:

Page title: This page says

Label: Enter text:
Textbox: shows "Frankincense and Myrrh!" being entered

Buttons: OK, Cancel

Screenshot:

Page title: This page says

Label: Pig Latin Translator

Entered text: frankincense and myrrh
Pig Latin: ankincensefray andway yrrhmay

Button: OK

Screenshot:

Page title: This page says

Label: Continue? (y/n):

Textbox: shows "y" being entered

Buttons: OK, Cancel

Specifications

* The app should accept user input until the user stops entering “y” or clicks Cancel when asked if they want to continue.
* If the user input is an empty string, the app should prompt the user again.
* Convert the user input to lowercase and remove punctuation before translating.
* Assume that words are separated from each other by a single space.
* If the word starts with a vowel, just add *way* to the end of the word.
* If the word starts with a consonant, move all of the consonants that appear before the first vowel to the end of the word, then add *ay* to the end of the word.
* If a word starts with the letter *y*, the *y* should be treated as a consonant. If the *y* appears anywhere else in the word, it should be treated as a vowel.

# Project 5-1 Test Scores

Create an app that accepts a comma-separated list of test scores and displays the list of scores, the low score, the high score, and the average score rounded to 1 decimal, as shown below.

Screenshot:

Page title: This page says

Label: Enter test scores separated by commas.

Textbox: shows "89, 98, 73, 87, 96" being entered

Buttons: OK, Cancel

Screenshot:

Page title: This page says

Label: Scores: 89, 98, 73, 87, 96
Low score = 73
High score = 98
Average score = 88.6

Button: OK

Specifications

* If the user inputs an empty string or clicks Cancel, the app should display a message that no scores were entered.
* Assume that the user enters valid numbers separated by commas.
* Convert the user entry to an array of scores.
* Use the Math.min() and Math.max() methods to get the low and high scores. (Hint: use the spread operator to pass an array to these methods.)
* Use functions to organize your code.

# Project 5-2 Change Calculator

Create an app that accepts a number of cents and displays the minimum number of quarters, dimes, nickels, and pennies that make up the number of cents, as shown below.

Screenshot:

(In big, blue letters) Change Calculator

Label: Enter number of cents (0-99):
Textbox: shows "67" being entered

Buttons: Calculate, Clear

Label: Quarters:
Textbox: shows "2"

Label: Dimes:
Textbox: shows "1"

Label: Nickels:
Textbox: shows "1"

Label: Pennies:
Textbox: shows "2"

Specifications

* Use the files in the following folder:

projects\ch05\change\_calculator\

* The app should validate the user input to ensure it falls within the proper range (0-99).
* The app should always return the minimum number of coins needed to make the change.
* The Clear button should clear both the input and the generated change amounts.

# Project 5-3 Password Generator

Create an app that generates strong passwords of the length entered by the user, as shown below.

Screenshot:

(In big, blue letters) Generate a strong password

Label: Number of characters:
Textbox: shows "10" being entered

Label: Password:
Textbox: shows "HWU!x@2hBn"

Buttons: Get Password, Clear

Specifications

* Use the files in this folder:

projects\ch05\password\

* The app should validate the user input to ensure it’s a valid number greater than zero.
* The app should select random characters from a predefined alphabet of characters that includes numbers, uppercase letters, lowercase letters, and symbols.
* The Clear button should clear both the input and the generated password.

# Project 6-1 Temperature Converter

Create an app that uses radio buttons to determine whether the conversion is from Fahrenheit to Celsius or vice versa. The interface should look like the screens below.

Fahrenheit to Celsius:

Screenshot:

(In big, blue letters) Convert temperatures

Radio buttons: Fahrenheit to Celsius, Celsius to Fahrenheit
(First radio button is selected)

Label: Enter F degrees:
Textbox: shows "100" being entered

Label: Degrees Celsius:
Textbox: displays "38"

Button: Convert

Celsius to Fahrenheit:

Screenshot:

(In big, blue letters) Convert temperatures

Radio buttons: Fahrenheit to Celsius, Celsius to Fahrenheit
(Second radio button is selected)

Label: Enter C degrees:
Textbox: shows "100" being entered

Label: Degrees Fahrenheit:
Textbox: displays "212"

Button: Convert

Specifications

* Use the files in this folder to develop this app:

projects\ch06\convert\_temps\

* To convert Fahrenheit to Celsius, first subtract 32 from the Fahrenheit temperature. Then, multiply that result by 5/9.
* To convert from Celsius to Fahrenheit, multiply the Celsius temperature by 9/5 and add 32.
* When the user clicks a radio button, the prompts should change accordingly.
* If the user enters invalid data, the app should display an error message below the Convert button.

# Project 6-2 Test Scores

Create an app that uses an array for the test scores and adds nodes to the DOM to display the Results and the Scores. The interface after adding names and scores should look like this:

Screenshot:

(In big, blue letters) Test Scores

Label: Name:
Textbox: empty

Label: Score
Textbox: empty

Buttons: Add, Clear

Labels:

Results
Low score = Judy with a score of 77
High score = Anne with a score of 99
Average score = 91

Scores
Ben    88
Joel    98
Judy    77
Anne   99

If the user enters invalid data, it should look like this:

Screenshot:

(In big, blue letters) Test Scores

Label: Name:
Textbox: empty
Red letters next to textbox: Please enter a name.

Label: Score
Textbox: empty
Red letters next to textbox: Score must be between 0 and 100.
Buttons: Add, Clear

Specifications

* Use the files in this folder to develop this app:

projects\ch06\test\_scores\

* The app should allow the user to enter scores for multiple users one at a time with the Add button.
* When a score is added, the Results section should display the updated low score, high score, and average score, and the Scores section should display all of the entered scores.
* The Clear button should clear both the input and the scores and results.

# Project 6-3 FAQs

Modify the FAQs app so only one answer can be displayed at a time. In other words, when the user clicks on a heading to display the answer, the other answers must be hidden.

Screenshot:

JavaScript FAQs
+ What is JavaScript?
- What is jQuery?
   jQuery is a library of the JavaScript functions that you're most likely to need as you develop websites.
+ Why is jQuery so popular?

Specifications

* Use the files in this folder to modify this app:

projects\ch06\faqs\

* When one element expands, any open element should automatically close.

# Project 6-4: Tuxedo Cat Coffee

Create an app that allows the user to place an order at a fictitious coffee shop named Tuxedo Cat Coffee.

Screenshot:

Label: Menu
Click an item to add it to your order

Six photos of menu items in a 3x2 grid. The mouse is over the picture in the bottom right, which displays "Scone $2.95" over the image.

Label: Your Order
Textbox with a scroller: $3.45 - Cappuccino
$2.95 - Scone

Label: Total: $6.40

Buttons: Place Order, Clear Order

Specifications

* Use the files in this folder to develop this app:

projects\ch06\cafe\

* When the mouse hovers over one of the images in the menu, another image should be displayed with the description and price of the item. The id attribute of each <img> element identifies the image to be displayed when it’s rolled over.
* When the user clicks on an image, that item should be added to the order list, and the order list and order total should be updated and displayed.
* If the user clicks the Place Order button, the checkout.html page should be displayed.
* If the user clicks the Clear Order button, all of the items should be removed from the order list and the total should be cleared.

# Project 8-1 Countdown

Modify the Countdown app from chapter 8 so that events in the future display the days, hours, minutes, and seconds until the event, updated every second. An event in the future should look like this:

Screenshot:

(In big, blue letters) Countdown To...

Label: Event Name:
Textbox: shows "tax day" entered

Label: Event Date:
Textbox: shows "4/15/2024" entered

Button: Countdown!

(in bold, red letters) 54 day(s), 6 hours, 24 minutes, 45 seconds until tax day (Mon Apr 15 2024)

Specifications

* Use the files in this folder to develop this app:

projects\ch08\countdown\

* To calculate the days, hours, minutes, and seconds, first get the number of seconds between the current date and the event date. Then, use the following values:
* Days = seconds divided by 86400
* Hours = seconds divided by 3600
* Minutes = seconds divided by 60
* Clicking the Countdown button should cancel any previous interval timers.

# Project 8-2 Calendar

Create an app that displays a calendar for the current month:

Screenshot:

(In big, blue letters) February 2024

A picture of a calendar for February 2024. The box for day 19 is colored in light blue.

**Note:** To build this calendar, you’re going to need the getDay() method of a Date object. This method returns the number of the day of the week (0 for Sunday, 1 for Monday, etc.).

Specifications

* Use the files in this folder to develop this app:

projects\ch08\calendar\

* The code should produce the calendar for the current month.
* Add <tr> and <td> elements to the HTML file to implement the calendar cells.
* Assign the <td> element for today’s date to the “today” CSS class so it displays with a different background color.

# Project 9-1 Contact Info with HTML validation

Create a Contact Info app that uses HTML data validation to validate the user’s entries. This app should use HTML only, no JavaScript.

Screenshot:

(In big, blue letters) My Contact Info

Label: Name:
Textbox: shows "Grace" entered

Label: E-Mail:
Textbox: shows "grace@hotmail.com" entered

Label: Mobile phone:
Textbox: shows "555-1234" entered, and a popup notification that says "Please match the requested format. nnn-nnn-nnnn"

Label: Zip Code
Textbox is obscured by the popup.

Label: Date of Birth:
Textbox is obscured by the popup, but uses a default date picker.

Buttons: Save, Reset

Specifications

* Use the files in this folder to start your program:

projects\ch09\contact\_info\_1.0\

* Make all five fields required.
* Be sure to use the correct input types for each field.
* Use the following patterns to verify the mobile phone and zip code:
* Phone: nnn-nnn-nnnn
* Zip code: nnnnn

Use both placeholder and title attributes to guide the user.

* The app should set the focus on the first field when the page loads.
* The Save button should validate the form and navigate to the confirm.html page if and only if all fields are valid.
* The Reset button should reset the form to the original blank (name, e-mail, date of birth) or placeholder (phone, zip) values.

# Project 9-2 Contact Info with custom validation

Modify the Contact Info app from project 9-1 to use JavaScript to improve the data validation.

Screenshot:

(In big, blue letters) My Contact Info

Label: Name:
Textbox: empty, followed by red letters that say "Please enter your name."

Label: E-Mail:
Textbox: empty, followed by red letters that say "Please enter an email or phone."

Label: Mobile phone:
Textbox: empty

Label: Zip Code
Textbox: empty, followed by red letters that say "Please enter a 5 digit ZIP."

Label: Date of Birth:
Textbox shows a date of 03/01/2027, followed by red letters that say "DOB must be in the past."

Buttons: Save, Reset

Specifications

* Use the files in this folder to get started:

projects\ch09\contact\_info\_2.0\

* All the specifications from project 9-1 apply, except that *both* email and phone number are no longer required. Instead, the user may enter *either* an email or a phone number, so long as they enter at least one.
* All five fields should have custom error messages that display next to the field.
* The date of birth entered by the user must be a date in the past.
* The Save button should validate the form and navigate to the confirm.html page if and only if all fields are valid.
* If the Save button is clicked and the form is invalid, display all appropriate error messages and do not navigate to the confirm.html page.
* The Reset button should clear all fields and error messages.

Important information about Date objects

* In Chrome, an <input> element with a type attribute of “date” returns a date string in universal format, such as “2024-04-15”. However, as you learned in chapter 8, passing such a string to the Date() constructor produces a UTC date and time rather than local. To get a local date and time, you can append “T00:00:00” to the date string before passing it to the Date() constructor.
* For two Date objects to be equal, they must have the same time as well as the same date. If you only wish to compare dates, you can set the hours, minutes, seconds, and milliseconds of each Date object to 0 before comparing them for equality.

# Project 10-1 Website login

Create an app that allows you to log in and log out. The interface looks like this initially:

Screenshot:

(In big, blue letters) My website

Label: User name:
Textbox shows "Anne" entered

Button: Log In

And the interface looks like this after a user has logged in:

Screenshot:

(In big, blue letters) My website

Label: Welcome, Anne!

Button: Log Out

Specifications

* Use the files in this folder to help write this app:

projects\ch10\login\

* When a user clicks the Log In button, the app should store the username in local storage and then display a welcome message.
* If the field is blank, the Log In button should not do anything.
* When you close the browser and reopen it, the welcome message should be displayed without the user having to log in again.
* When the user clicks the Log Out button, the username should be removed from local storage and the log in page should be displayed again.

# Project 10-2 Contact Info

Modify the Contact Info app from project 9-2 to save its data in session storage and retrieve and display that data on the confirmation page.

Screenshot:

(In big, blue letters) My Contact Info

Label: Name:
Textbox: empty

Label: E-Mail:
Textbox: empty

Label: Mobile phone:
Textbox: empty

Label: Zip Code
Textbox: empty

Label: Date of Birth:
Textbox empty but with a date picker

Buttons: Save, Reset

When you save valid data, the confirmation page should get the data from session storage and display it like this:

Screenshot:

(In big, blue letters) My Contact Info

Label: Name: Grace
E-Mail: grace@hotmail.com
Mobile phone: 555-123-4567
ZIP Code: 12345
Date of Birth: Tue Dec 09 1986

Link: Back to Contact Info

Specifications

* Use the files in this folder to help you develop the app:

projects\ch10\contact\_info\

* The app should validate the input for each field and not submit the form unless all fields have been correctly filled out.
* The app should use session storage to store the data entered in the form. When the user returns to the Contact Info page via the link, they should not have to retype the data. It should be pre-populated in the fields for them.
* **Note:** In Chrome, when you set the value of an <input> element with a type attribute of “date”, the value must be in universal format (yyyy-MM-dd).

# Project 11-1 Contact Info

Modify the Contact Info app from project 10-2 so it uses an array.

Screenshot:

(In big, blue letters) My Contact Info

Label: Name:
Textbox: empty, followed by red letters that say "Please enter your name."

Label: E-Mail:
Textbox: empty, followed by red letters that say "Please enter an email or phone."

Label: Mobile phone:
Textbox: empty

Label: Zip Code
Textbox: empty, followed by red letters that say "Please enter a 5 digit ZIP."

Label: Date of Birth:
Textbox shows a date of 03/01/2027, followed by red letters that say "DOB must be in the past."

Buttons: Save, Reset

Screenshot:

(In big, blue letters) My Contact Info

Label: Name: Grace
E-Mail: grace@hotmail.com
Mobile phone: 555-123-4567
ZIP Code: 12345
Date of Birth: Tue Dec 09 1986

Link: Back to Contact Info

Specifications

* Use the files in this folder to help you write the program:

projects\ch11\contact\_info\

* The app should take an email, phone number, zip code, and date of birth as input.
* The app should validate all input.
* The app should continue to store user input in web storage, but it should store and retrieve an array of data rather than individual values.

# Project 11-2 Student Scores

Create an app that tracks student’s scores; tallies the average of the entered scores; lets the user sorts the entered students by first name, last name, or score; and lets the user filter the entered students by score.

Screenshot:

(In big, blue letters) Student Scores

Label: First Name
Textbox: shows "Lucy" entered

Label: Last Name:
Textbox: shows "Van Pelt" entered

Label: Score:
Textbox: shows 91 entered

Buttons: Add, Clear Scores

Text area: Brown, Charlie: 75
Patty, Peppermint: 86
Van Pelt, Linus: 98

Label: Average Score: 86.3

Label: Sort by:
Drop-down menu displays "Last Name"

Label: Filter by:
Drop-down menu is being clicked, showing that it contains "Show All, 90+, 80+, 70+, 60+" as options.


Specifications

* Use the files in this folder to help you write this app:

projects\ch11\scores\

* The app should take a first name, last name, and score as input. The app should use an array of arrays to store this data.
* By default, the app should display all entered students sorted by last name.
* The average score should be for the students currently being displayed. That is, if the drop-down filter is “80+”, the displayed average score should average only scores of 80 and above.
* When either of the drop-downs change, the students being displayed and the average score should be recalculated and displayed immediately based on the value of both drop-downs.
* The app should implement event handlers for the click event of the Add Score and Clear Scores buttons, and for the change event of the Sort by and Filter by drop-downs.

# Project 12-1 Change Calculator

Modify the Change Calculator app from project 5-2 to use an object literal.

Screenshot:

(In big, blue letters) Change Calculator

Label: Enter number of cents (0-99):
Textbox: shows "67" being entered

Buttons: Calculate, Clear

Label: Quarters:
Textbox: shows "2"

Label: Dimes:
Textbox: shows "1"

Label: Nickels:
Textbox: shows "1"

Label: Pennies:
Textbox: shows "2"

Specifications

* Use the files in this folder to help you develop the app:

projects\ch12\change\

* Change the code to use an object literal with methods to validate that the entered number of cents is between 0 and 99 and to calculate the number of coins needed.
* The number of quarters, dimes, nickels, and pennies should be stored as properties in the object literal.

# Project 12-2 Burger Town

Create a Burger Town app that uses classes to implement a menu system for a burger restaurant. When you’re done, the app will work as shown below.

Screenshot:

(In big, blue letters) Burger Town Menu

() is use to denote a radio button and [] for a check box in this description

On the left, in a column:

Label: Burger
Type () Regular () Cheese
Size () Single () Double

Toppings
[] Tomatoes [] Lettuce [] Pickles
[] Onions [] Mustard [] Ketchup

Label: Drink
Type () Water () Tea () Soda
Size () Small () Medium () Large

Label: Fries
Type () Regular () Curly
Size () Small () Medium () Large


Buttons: Add Order, Clear Order

On the right, in a column:

Label: Order

single cheeseburger - $6.00
* tomatoes
* lettuce
double burger - $7.00
* lettuce
* pickles
* mustard
* ketchup

large water $0.00
large soda $3.75
medium fries - $3.00
large curly fries - $3.75

Total: $23.50

Specifications

* Use the files in this folder to develop the app:

projects\ch12\burger\_town\

* Use a class to represent each menu item.
* Use a class to represent the order.
* The Add Order button should add any selected items to the order and update the display.
* The Click Order button should remove all items from the order and update the display.
* If a burger is ordered with only a type specified, use a default size of single. If a burger is ordered with only a size, use a default type of regular.
* If a burger is ordered with only toppings specified (no type or size), do not add it to the order.
* For drinks, the default type and size should be water and small. For fries, the default type and size are regular and small.

# Project 13-1 Contact Info

Modify the Contact Info app from project 11-1 to use modules.

Screenshot:

(In big, blue letters) My Contact Info

Label: Name:
Textbox: empty, followed by red letters that say "Please enter your name."

Label: E-Mail:
Textbox: empty, followed by red letters that say "Please enter an email or phone."

Label: Mobile phone:
Textbox: empty

Label: Zip Code
Textbox: empty, followed by red letters that say "Please enter a 5 digit ZIP."

Label: Date of Birth:
Textbox shows a date of 03/01/2027, followed by red letters that say "DOB must be in the past."

Buttons: Save, Reset

Screenshot:

(In big, blue letters) My Contact Info

Label: Name: Grace
E-Mail: grace@hotmail.com
Mobile phone: 555-123-4567
ZIP Code: 12345
Date of Birth: Tue Dec 09 1986

Link: Back to Contact Info

Specifications

* Use the files in this folder to help you write the program:

projects\ch13\contact\_info\

* The app should take an email, phone number, zip code, and date of birth as input.
* The app should validate all input.
* The app should continue to store user input in session storage, but it should store a custom Contact object rather than an array or individual values.
* Store the code for the object in modules.
* One module should contain code for a Dob class (date of birth) which inherits Date.
* One module should contain code for a Contact class which validates and stores the contact information. It should import the Dob class.

# Project 13-2 Craps dice game

Create an app that lets the user play street craps, which is a simplified version of the craps game played in casinos. The app should use modules to do its work.

Screenshot:

(In big, blue letters) Street craps

Label: Rules:
* If you roll 7 or 11 on the first (come out) roll, you win.
* If you roll 2, 3, or 12 on the come out roll, you lose.
* Any other number on the come out roll sets the point.
* If you roll the point before you roll 7, you win.
* If you roll 7 before you roll the point, you lose.

Buttons: Roll (greyed out), New Game

Label: Current roll: 2  The point:   0

(in red letters) You rolled 2 on the come out roll - you lose.

Specifications

* Use the files in the following folder:

projects\ch13\craps\

* Use a module to implement a Die class and a Dice class.
* Use a module to implement the rules below. It should include properties and functions to roll two dice and set an appropriate message based on the total rolled.
* When the app starts, it should enable the Roll button and disable the New Game button.
* The Roll button should simulate rolling two 6-sided dice. The total of the two dice should display in the Current roll label.
* The first roll is called the “come out” roll. If the user rolls 7 or 11 on the come out roll, they win. If they roll 2, 3, or 12 on the come out roll, they lose.
* If the user doesn’t win or lose on the come out roll, the number they roll should be assigned as the point value. This value should display in the point label.
* If the user doesn’t win or lose on the come out roll, they should continue until they roll the point or a 7. If they roll the point, they win. If they roll a 7, they lose.
* When the user wins or loses, the app should display the result in red, disable the Roll button, and enable the New Game button.
* The New Game button should start a new game, set the Current roll and point labels to zero, clear any previous message, enable the Roll button, and disable the New Game button.

# Project 14-1 To-Do List

Create an app that uses data from the JSON Placeholder API to display a list of to-do items for a specified user. After selecting a user, the app should look like this:

Screenshot:

(In big, blue letters) ToDo List

Label: Select a User
Drop-down menu displays "Leanne Graham"

Two columns, one headed "ToDo Item" and one "Completed". The ToDo column contains latin phrases. The Complete column contains either "true" or "false" for each item.


Specifications

* Use the files in this folder to help you develop the app:

projects\ch14\todo\_list\

* You can use the following links to access a list of users and the to-do items for a user (you’ll need to supply a user id number between 1 and 10):

https://jsonplaceholder.typicode.com/users

https://jsonplaceholder.typicode.com/todos/?userId=#

* When the app first loads, it should display the to-do items for the first user in the <select> element.
* When a new user is selected, the app should display the to-do items for that user (and only that user).

# Project 14-2 Mars Rover Photos

Modify an app that uses NASA’s Mars Rover Photos API to allow you to view photos from any of the rovers sent to Mars by NASA. After selecting a rover, a date, a camera, and \clicking the “View” button, the app should look like this:

Screenshot:

(In big, blue letters) Mars Rover Photos

Label: Select a Rover
Drop-down menu shows "Curiosity" selected

Label: Status
immediately below: active

Label: # of Photos
immediately below: 695670

Label: Landing Date
immediately below: 2012-08-06

Label: Max Date
immediately below: 2024-02-19

Label: Select a Date:
Three drop-down menus for selecting a year, month (by number), and day

Label: Select a Camera:
a drop-down menu showing "Front Hazard Avoidance Camera"

Button: View photos

Two photos displayed

Specifications

* Use the files in this folder to help you develop the app:

projects\ch14\mars\_rover\

* To get a list of all the Mars rovers, use this URL:

https://api.nasa.gov/mars-photos/api/v1/rovers ?api\_key=DEMO\_KEY&page=1

(For best results, sign up for and use your own API key rather than DEMO\_KEY)

* To get the photos for a specific rover and date, update the URL like this:

https://api.nasa.gov/mars-photos/api/v1/rovers/Curiosity/photos  
?api\_key=DEMO\_KEY&page=1&earth\_date=2024-2-19

* To get the photos for a specific camera, update the URL like this:

https://api.nasa.gov/mars-photos/api/v1/rovers/Curiosity/photos  
?api\_key=DEMO\_KEY&page=1&earth\_date=2024-2-19&camera=FHAZ

* When the app loads, it should populate the “Select a Rover” drop-down.
* When the user selects a rover, the app should use data for the selected rover to display the status, # of photos, landing date, and max date and to populate the Select a Date and Select a Camera drop-downs.
* When the user clicks the View photos button, photos should display as shown above.

# Project 15-1 Create and run a server script with parameters

Write the JavaScript for a server-side script named test\_scores that calculates the average of the test scores that are passed to it. You should be able to pass one or more test scores to this script.

> cd /projects/ch15

> node test\_scores 89 100 92 93

All scores: 89, 100, 92, 93

Average score: 94

Specifications

* Use the index.js file in this directory to code the script:

projects\ch15\test\_scores\

* The script should take one or more test scores separated by spaces as an argument.

# Project 15-2 Run a server to host an app

Start a web server with node.js so you can run the Countdown app from project 8-1:

Screenshot:

(In big, blue letters) Countdown To...

Label: Event Name:
Textbox: shows "Christmas" entered

Label: Event Date:
Textbox shows "12/25/2024" entered

Button: Countdown!

(in red letters) 287 day(s), 13 hours, 46 minutes, 8 seconds until Christmas (Wed Dec 25 2024)

Specifications

* If necessary, install Node.js on your system.
* Navigate to the files in this directory and start the http server:

projects\ch15\countdown\

* Start the http server and navigate to the hosted app.
* Test the app to make sure that everything is working correctly.

# Project 16-1 Tic Tac Toe

Screenshot:

(In big, blue letters)  Tic Tac Toe

Label: X wins!
Button: New Game

A 3 by 3 grid of squares. It is marked
X O X
O X (blank)
O (blank) XCreate a Tic Tac Toe game. This game expands on the Tic Tac Toe app presented in the book by allowing two players to play a game of Tic Tac Toe.

Screenshot:

(In big, blue letters)  Tic Tac Toe

Label: O's turn
Button: New Game

A 3 by 3 grid of squares. The top row is marked X O XSpecifications

* Use the files in this folder to help you develop the app:

projects\ch16\tictactoe\

* When a user clicks in a square, a mark for the corresponding user should be placed in the center of that square. Use the letters X and O for the marks.
* When a user clicks in a square that already has a mark, nothing should happen.
* After a mark is placed, the turn should switch to the other user automatically.
* Every time a mark is placed, the game should check if there is a winner.
* The game ends when there’s a winner or when all the squares have a mark and there’s no winner. The users should be notified of the result when the game ends.
* After a game ends, nothing should happen when a user clicks in a square until a new game starts.
* The New Game button should reset the game so users can play again.

# Project 16-2 Connect Four

Create a version of the popular game Connect Four as shown below. In this game, users take turns placing pieces in a vertical grid until one user gets four pieces in a row. When a user places a piece, it falls to the lowest empty position on the board in that column. For example, the yellow piece that is about to be placed in the third column will fall down until it rests above the red piece below it.

Screenshot:

(In big, blue letters) Connect Four!

Label: Yellow's Turn

Button: New Game

A yellow circle representing yellow's next piece to place is in the center above a blue board.

The blue board has a 6 x 7 grid of white circles on blue. In the bottom left, a red piece is placed. Next to it is a yellow piece, and then a red piece at the bottom of the third column.

Specifications

* Use the files in this directory to help you develop this app:

projects\ch16\connect\_four\

* Clicking on the piece at the top of the board should create a transparent piece which the user can drag and drop in the desired column.
* When the user drops the piece, the piece should move to the correct location on the board. You do not need to animate it “falling.”
* When a user gets four of their pieces in a row, the game should notify the users who won. If there are no winners and no more spaces to make a move, it’s a draw.