Powercoders Bootcamp - Exercises

Monday 23rd September, 2024

Introduction

!	mandatory
(*)	basic
(**)	intermediate
(***)	advanced

Contents

1 CLI

For this exercises, you need to be using a Unix shell like Bash or ZSH. If you are on Linux or macOS, you don't have to do anything special. If you are on Windows, you need to make sure you are not running cmd.exe or PowerShell; you can use Windows Subsystem for Linux or a Linux virtual machine to use Unix-style command-line tools.

To make sure you're running an appropriate shell, you can try the command `echo $SHELL`. If it says something like `/bin/bash` or `/bin/zsh`, that means you're running the right program.

1.1 Basic Exercises

! Ex. 1.1.1 (*) - Create directories and navigate

- Navigate to your home directory (∼).
- Create a new directory and move into it using a single command.
- Create a sub-directory and then move back to the parent directory.

! Ex. 1.1.2 (*) - Create, move, copy and remove files

- Create a directory.
- Create two files within the directory above.
- Rename one of them and copy the other to a new location.
- Remove a file and then a directory.

! Ex. 1.1.3 (*)

- Create a folder called `cli`.
- Make that folder your current working directory.
- Create two files: `file1.txt`, `file2.txt`.
- Copy `file1.txt` and call the copy `file3.txt`.
- Create a directory called `folderOne`.
- Move `file1.txt` into `folderOne`.
- List the contents of `folderOne` without going into it.
- Rename `file1.txt` to `myfile.txt`.
- Remove the directory `folderOne`, including the file inside.

! Ex. 1.1.4 (*)

- Create a new directory called `powercoders` under `/tmp`.
- Look up the `touch` program. The man program is your friend.
- Use `touch` to create a new file called `bootcamp` (no extension needed) in the `powercoders` folder.
- Write the following into that file, one line at a time:

```
#!/bin/sh
curl --head --silent https://powercoders.org

```

1.2 Intermediate Exercises

Ex. 1.2.1 (**) - Execute a file

- Try to execute the file (i.e. type the path to the script: `./bootcamp`) into your shell and press enter. Understand why it does not work by consulting the output of `ls` (hint: look at the permission bits of the file).
- Run the command by explicitly starting the `sh` interpreter, and giving it the file `bootcamp` as the first argument, i.e. `sh bootcamp`. Why does this work, while `./bootcamp` did not? Look up the `chmod` program (e.g. use `man chmod`).

1.3 Advanced Exercises

Ex. 1.3.1 (***) - Use chmod

Use `chmod` to make it possible to run the command `./bootcamp` rather than having to type `sh bootcamp`. How does your shell know that the file is supposed to be interpreted using `sh`?

Ex. 1.3.2 (***) - Pipe and redirection

Run the `bootcamp` executable file and use the pipe (|) and the redirection operator (`>`) to extract the "last modified" date, writing it into a file called `last-modified.txt`. (You might want to look at `man grep`)

2 GIT Exercises

2.1 Basic Exercises

Ex. 2.1.1 (*) - Familiarize with git

If you don't have any past experience with Git, either try reading the first couple chapters of Pro Git or go through a tutorial like Learn Git Branching.

! Ex. 2.1.2 (*) - Create your own repo, start to commit

In this exercise, you will be creating a repository and adding some project folders and files. You will be working on creating a repository for your coming weeks of school phase (7 weeks). As you read through these steps, think about when would be a good time to commit.
As you execute the steps, you get to decide when to commit.

- Create a new folder named `poco` on your desktop and make it a repository.
- Add for each week of school a folder inside the repository.
- Add to each folder a file called `README.md`.
- Edit each file and add the current week number to it as content.
- Look at your history with `git log` as you add more commits.

You might want to use some hints:

- At the beginning, do not be inside of your current project folder when you make a new folder. Make sure you are on your desktop.
- Make sure you are inside of the folder when you make it a repository.
- Remember to commit. When would be a good time to commit?
- Don't forget to run `git status` regularly so that you can see what is happening at each stage.

Ex. 2.1.3 (*) - Explore git stash

Clone a repository from GitHub, or use a local repository you have created, and modify one of its existing files.

- What happens when you do `git stash`?
- What do you see when running `git log --all --oneline`?
- Run `git stash pop` to undo what you did with `git stash`.
- In what scenario might this be useful?

! Ex. 2.1.4 (*) - Navigate history - GitHub needed

Clone this repository from GitHub.

- Explore the version history by visualizing it as a graph.
- Who was the last person to modify `README.md`? (Hint: use git log with an argument).
- What was the commit message associated with the last modification to this specific file: `soupsmushroom-soup.md`? (Hint: use git blame and git show).

2.2 Intermediate Exercises

! **Ex. 2.2.1 (**) - Create commits and branches** ca. 20 min

- Fork the recipe-book repository (you might want to look at this.)
- First create a new branch and then add a recipe to the branch and commit the change.
- In a new commit, modify the recipe you just added.
- Switch to the main branch and modify a recipe there.
- Compare the branch that you created with the main branch. Can you find an easy way to see the differences? (hint: use `git diff main feature`)

Ex. 2.2.2 (**) - Play with .gitconfig

Like many command line tools, Git provides a configuration file (or dotfile) called `.gitconfig`.

- Create an alias in `.gitconfig` so that when you run `git graph`, you get the output of `git log --all --graph --decorate --oneline`.

You can do this by directly editing the `.gitconfig` file, or you can use the `git config` command to add the alias. Information about git aliases can be found here.

2.3 Advanced Exercises

Ex. 2.3.1 (***) - Explore .gitignore

You can define global ignore patterns in `.gitignore_global` after running
`git config --global core.excludesfile ~/.gitignore_global`.
Do this, and set up your global gitignore file to ignore OS-specific or editor-specific temporary files, like `.DS_Store`.

Ex. 2.3.2 (***) - Make your PR

Let's work again on this repository, find a typo or some other improvement you can make, add a recipe of your own, and submit a pull request on GitHub (you may want to look at this).

Ex. 2.3.3 (***) - Remove data from a repository

One common mistake when learning Git is to commit large files that should not be managed by Git or adding sensitive information. Try adding a file to a repository, making some commits and then deleting that file from history (you may want to look at this).

3 Programming 101 Exercises

Solve the following problems. If you are stuck, ask one of the IT trainer or one of your peers.

a) Understand: Visualize the why?how?what?
b) Plan: What are the constraints, the inputs, the steps?
c) Divide: Break bigger problems into smaller ones.
d) Conquer: Write pseudocode. (Text and/or Flowchart)

Introductory Riddle - How to cross the river?

A farmer with a fox, a goose and a sack of corn needs to cross a river. The farmer has a rowboat, but there is room for only the farmer and one of his three items. Unfortunately, both the fox and the goose are hungry. The fox cannot be left alone with the goose, or the fox will eat the goose. The goose cannot be left alone with the corn, or the goose will eat the corn.

How does the farmer get everything across the river?

Introductory Riddle - Darkness phobia

One family wants to get through a tunnel. Dad can make it in 1 minute, mama in 2 minutes, son in 4 and daughter in 5 minutes. Unfortunately, not more than two persons can go through the narrow tunnel at one time, moving at the speed of the slower one.

Can they all make it to the other side if they have a torch that lasts only 12 minutes and they are afraid of the dark?

3.1 Basic Exercises

Ex. 3.1.1 (*) - Print greeting

Create a program that prompts for your name and prints a greeting using your name.

Ex. 3.1.2 (*) - Add numbers ca. 30 min

Allow the user to input numbers, add them and print the result.

Ex. 3.1.3 (*) - Which number is bigger? ca. 15 min

Declare 2 variables, both numbers. Compare which number is greater Log the output, e.g. "The greater number of 5 and 10 is 10." Add an output for the else statement, e.g. "The smaller number of 5 and 10 is 5."

Ex. 3.1.4 (*) - Subtract numbers ca. 45 min

Allow the user to input numbers, subtract the smaller from the bigger number and print the result.

Ex. 3.1.5 (*) - Combine ca. 60 min

Combine Ex. 3.1.2 and 3.1.4 into one program.
The program should ask the user if they want to add or subtract two numbers. Then the program should ask for the two numbers and print the result.

3.2 Intermediate Exercises

! **Ex. 3.2.1 (**) - FizzBuzz**

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Ex. 3.2.2 (**) ca. 45 min

Write a program that asks the user for a date. Find the day of the year for the given date. For example, January 1st would be 1, and December 31st is 365.

a) Do not include leap years.
b) Include leap years. Check the rules for leap years on the internet, and implement them in your program.

3.3 Advanced Exercises

Ex. 3.3.1 (***) - Sorting algorithms

Write a sort algorithm yourself. (e.g. bubble sort)

Ex. 3.3.2 (***) - Greatest Common Divisor ca. 45 min

Write a program that calculates the greatest common divisor (GCD) of two (positive) numbers.[1]

Ex. 3.3.3 (***) - Binary Search Algorithm

Write a binary search algorithm.

Ex. 3.3.4 (***) - Caesar Cipher

Write a program that encrypts a message using the Caesar Cipher[2]. The program should ask the user for a message and a “key”. The program should then print the encrypted message.

[1] https://en.wikipedia.org/wiki/Greatest_common_divisor
[2] https://en.wikipedia.org/wiki/Caesar_cipher

4 JavaScript Exercises

4.1 Basic Exercises

Ex. 4.1.1 (*) - Play with data types, operations and comparisons

Guess what answers you would get if you ran this in the JavaScript Console of your browser. Once you have an answer to the questions check the answers by copying them and running it in the console yourself line by line

a) Evaluate the below:
   - `5 + "34"`
   - `5 - "4"`
   - `10 % 5`
   - `5 % 10`
   - `"Java" + "Script"`
   - `" " + " "`
   - `" " + 0`
   - `true + true`
   - `true + false`
   - `false + true`
   - `false - true`
   - `3 - 4`
   - `"Bob" - "bill"`

b) Evaluate the below comparisons:
   - `5 >= 1`
   - `0 === 1`
   - `4 <= 1`
   - `1 != 1`
   - `"A" > "B"`
   - `"B" < "C"`
   - `"a" > "A"`
   - `"b" < "A"`
   - `true === false`
   - `true != true`

c) Make the string: `"Hi There! It's "sunny" out"` by using the + sign

Ex. 4.1.2 (*) - Evaluate

Evaluate what answers you would get if you ran this in the JavaScript Console in Google Chrome. Answer the questions then check them by copying them and running it in the console yourself line by line

a) Add two variables `firstName` and `lastName`, so that they equal your name.
b) Create a variable that holds the answer of `firstName` + `" "` + `lastName`.
c) Evaluate this question: what is `a + b` in the following code?

```
var a = 34;
var b = 21;
var c;
a = 2;
a + b

```

d) What is c equal to?

! Ex. 4.1.3 (*) - Hello World!

Write a program that prints a `Hello, World!` message in the console of a web browser.

! Ex. 4.1.4 (*) - Greetings to you

Write a program that

- asks for the user's name
- and then prints a greeting message in the console of a web browser.

! Ex. 4.1.5 (*) - Even or Odd

Write a program that asks for the user to input a number:

- If the number is even, print "The number is even."
- If the number is odd, print "The number is odd."
- If it is not a number, print "This is not a number."

! Ex. 4.1.6 (*) - Age Calculator

Want to find out how old you'll be? Calculate it!

- Use `prompt()` and `alert()`.
- Ask user for birth year.
- Ask user for a future year.
- Calculate the 2 possible ages for the given year.
- Output them to the screen/console like so: "I will be either NN or NN in YYYY", substituting the values.

For example, if you were born in 1988, then in 2026 you'll be either 37 or 38, depending on what month it is in 2026.

Additional Task: Ask for precise dates.

! Ex. 4.1.7 (*) - Make a keyless car

This car will only let you drive if you are over 18. Make it do the following:

- Use `prompt()` and `alert()`.
- Ask a user for their age.
- IF they say they are below 18, respond with: "Sorry, you are too young to drive this car. Powering off"
- IF they say they are 18, respond with: "Congratulations on your first year of driving. Enjoy the ride!"
- IF they say they are over 18, respond with: "Powering On. Enjoy the ride!"

! Ex. 4.1.8 (*) - Print multiplication table

Write a program that prints the multiplication table of the number 7. (Use a loop)

! Ex. 4.1.9 (*) - Play with for loop

Construct for loops that accomplish the following tasks:

- Print the numbers 0 - 20, one number per line.
- Print only the ODD values from 3 - 29, one number per line.
- Print the EVEN numbers 12 down to -14 in descending order, one number per line.
- Print the numbers 50 down to 20 in descending order, but only if the numbers are multiples of 3.

Ex. 4.1.10 (*) - Play with arrays and for loop

Initialize two variables to hold the string `LaunchCode` and the array `[1, 5, 'LC101', 'blue', 42]`, then construct for loops to accomplish the following tasks:

a) Print each element of the array to a new line.
b) Print each character of the string -in reverse order- to a new line.

Ex. 4.1.11 (*) - Play with while loop

Define three variables for the `LaunchCode` shuttle:

- one for the starting fuel level,
- another for the number of astronauts aboard,
- and the third for the altitude the shuttle reaches.

Construct `while` loops to do the following:

a) Prompt the user to enter the starting fuel level. The loop should continue until the user enters a positive value greater than 5000 but less than 30000.
b) Use a second loop to query the user for the number of astronauts (up to a maximum of 7). Validate the entry by having the loop continue until the user enters an integer from 1 - 7.
c) Use a final loop to monitor the fuel status and the altitude of the shuttle. Each iteration, decrease the fuel level by 100 units for each astronaut aboard. Also, increase the altitude by 50 kilometers. (Hint: The loop should end when there is not enough fuel to boost the crew another 50 km, so the fuel level might not reach 0).

Ex. 4.1.12 (*) - Fix the code

The intended behavior is to increment by one but why is this function not working? Can you fix this? (try to solve this on paper first and then debug the code)

```
function incrementItems(arr) {
    for (let i = 0; i < array.length; i++)
        arr[i] === arr[i] + 1
    return array
}
```

Expected behaviour: Input -> Output

```
incrementItems([0, 1, 2, 3]) -> [1, 2, 3, 4]
incrementItems([2, 4, 6, 8]) -> [3, 5, 7, 9]
incrementItems([-1, -2, -3, -4]) -> [0, -1, -2, -3]
```

4.2 Intermediate Exercises

! Ex. 4.2.1 (**) - FizzBuzz

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

! Ex. 4.2.2 (**)

Write a program that prints a multiplication table for numbers up to 10. (use nested loops)

1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

! Ex. 4.2.3 (**) - The world translater ca. 30 min

Write a function named helloWorld that:

- Takes 1 argument, a language code (e.g. "es", "de", "en")
- Returns "Hello, World" for the given language, for at least 3 languages. It should default to returning English.

Call the function for each of the supported languages and log the result to make sure it works.

Ex. 4.2.4 (**) - Retirement calculator ca. 90 min

Create a program that determines how many years you have left until retirement and the year you can retire. It should prompt for your current age and the age you want to retire and display the output as shown in the example below.

- What is your current age? 25
- At what age would you like to retire? 65
- You have 40 years left until you can retire.
- It's 2024, so you can retire in 2064.

Ex. 4.2.5 (**) - Sum and Average

Write a program that asks the user to input five numbers, print the sum and the average of the five numbers.

Ex. 4.2.6 (**) - Decimal/Binary conversion

a) Write a program that asks the user for a number and then prints the binary representation of the number. (If you don't know what a binary representation is, look it up. Computers use binary to represent numbers.)
Note: e.g. 25 in binary is 11001

b) Write another program that does the inverse operation - converting binary to decimal.

Ex. 4.2.7 (**)

a) Write a program that asks the user for a password and checks if it is valid. The password must
   - at least be 8 characters long
   - include 1 special symbol
   - include 1 number
   - not include a space or either

b) Write a program that generates a password according to the requirements above. Use your validator written in a).

4.3 Advanced Exercises

Ex. 4.3.1 (***) - Sorting Algos

Write a sort algorithm yourself. (e.g. bubble sort)

Ex. 4.3.2 (***) - Greatest Common Divisor ca. 45 min

Write a program that calculates the greatest common divisor (GCD) of two (positive) numbers.[3]

Ex. 4.3.3 (***) - Fibonacci

Write a program that prints the first 20 numbers of the Fibonacci sequence. https://en.wikipedia.org/wiki/Fibonacci_sequence

Ex. 4.3.4 (***) - Caesar Cipher

Write a program that encrypts a message using the Caesar Cipher[4]. The program should ask the user for a message and a "key". The program should then print the encrypted message.

[3]https://en.wikipedia.org/wiki/Greatest_common_divisor
[4]https://en.wikipedia.org/wiki/Caesar_cipher

5 HTML Exercises

Validate all your exercises using the W3C HTML validator.[5] Keep your code readable and maintainable.[6]

5.1 Basic Exercises

Ex. 5.1.1 (*)

Go to 2 websites of your choice and write the markup of the homepage.

- Which tags do you use? Why?
- Check your buddy's markup and discuss the choice of tags.

Done? Now, check the source code of the website.

- What do you see?
- Why?

Ex. 5.1.2 (*)

Go to the website digitec.ch and try to navigate to the menu and inside the menu with voice over either on your phone or laptop. (Google "Voiceover" for apple devices and "Narrator" for windows devices)

- Can you use the navigation to go to a subsite?
- Why? Why not?

Try futurecoders.ch next

Ex. 5.1.3 (*)

Create a website to show off your new skills. Core HTML tags you should be able to include on your page(s):

- A *nav* with a few links that either navigate to other pages or act as anchor tags.
- A *heading* to signal something important or declare a new section/paragraph.
- A couple *paragraphs* describing the amazing things you want to share. Remember to use *emphasis* on key words!
- A *list* of key things to know about something in your paragraph. Why not do an ordered list and unordered list?
- An *image* of something relevant.
- A short contact *form*
- A *table* to display additional contact data
- A *footer* at the bottom of the page declaring your copyright on such a wonderful write-up, and make sure you use the copyright *symbol*

Ex. 5.1.4 (*)

Draft a schematic outline of a newspaper page in HTML. (somewhat like the Washington Post)

[5] https://validator.w3.org/#validate_by_input
[6] https://www.smashingmagazine.com/2013/02/using-white-space-for-readability-in-html-and-css/

5.2 Intermediate Exercises

Ex. 5.2.1 (**)

a) Create a form with validation and 5 required form fields, each another input type: email, phone, number, color, date.
b) Look at the UI and validation of each field in at least 2 different browsers, e.g. Google Chrome and Mozilla Firefox. What do you notice? Describe in a few sentences differences as well as similarities.

! **Ex. 5.2.2 (**)**

Create the semantic HTML of this page: https://bootcamp.powercoders.org/img/exercise-form.png

! **Ex. 5.2.3 (**) - Sample shop HTML**

Over the coming weeks we'll try to build a sample web shop application. See an outline under:

- https://bootcamp.powercoders.org/img/sample_shop_homepage.png
- https://bootcamp.powercoders.org/img/sample_shop_video.png
- https://bootcamp.powercoders.org/img/sample_shop_checkout.png

In this first part, we'll start with the HTML. Create the HTML for the homepage of the sample shop.

6 CSS Exercises

6.1 Basic Exercises

! **Ex. 6.1.1 (*)**

Look at the source code and try to find out how the paragraph in each exercise will look like.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body p { color: rgb(0, 128, 0); } /*green*/
      html p { color: rgb(128, 0, 128); } /*purple*/
    </style>
  </head>
  <body>
    <p>Here is a paragraph</p>
  </body>
</html>

```

! **Ex. 6.1.2 (*)**

Look at the source code and try to find out how the paragraph in each exercise will look like.

```
<!DOCTYPE html>
<html>
    <head>
    <style>
        p {
        color: rgb(255, 255, 255);
        background-color: rgb(0, 0, 0);
        }
        p.info {
        background-color: rgba(0, 0, 255, 0.5);
        }
    </style>
    </head>
    <body>
    <p class="info">Here is a paragraph</p>
    </body>
</html>

```

! **Ex. 6.1.3 (*)**

Look at the source code and try to find out how the paragraph in each exercise will look like.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: rgb(255, 255, 255);
        background-color: rgb(0, 0, 0);
        text-decoration: none;
        border: 1px solid rgb(0, 0, 0);
      }
      p.info {
        background-color: rgba(0, 0, 255, 0.5);
        text-decoration: underline;
        padding: 1em;
      }
      p#tip {
        color: black;
        background-color: rgba(0, 255, 0, 0.5);
      }
    </style>
  </head>
  <body>
    <p id="tip" class="info">Here is a paragraph</p>
  </body>
</html>

```

6.2 Intermediate Exercises

Ex. 6.2.1 (**)

Choose three fonts from Google Fonts

- One sans-serif, for headings
- One serif, for almost everything else
- One monospace

Add some content, so you can see the 3 different fonts in action.

Ex. 6.2.2 (**) - CSS GAME

Play the following game https://flukeout.github.io

Ex. 6.2.3 (**) - Sticky footer

sticky footer pattern is one where the footer of your page "sticks" to the bottom of the viewport in cases where the content is shorter than the viewport height. Create a basic website layout with a sticky footer at the bottom.

Ex. 6.2.4 (**) - Responsive layout in grid and flex

Create a responsive layout using once grid and once flex. The to implement the following layout: https://bootcamp.powercoders.org/img/mobile-first-design.png

Ex. 6.2.5 (**)

Build a website like this one: https://bootcamp.powercoders.org/examples/columns/responsive-flex.html

a) With Flex
b) With Grid
c) With Multi-column

Ex. 6.2.6 (**) - Flexbox Froggy Game

https://flexboxfroggy.com/

Ex. 6.2.7 (**) - CSS Grid Garden

https://cssgridgarden.com/#en

Ex. 6.2.8 (**) - Form Styling

Remember the form you created in the HTML exercises?
https://bootcamp.powercoders.org/img/exercise-form.png

Now style it with CSS and make it responsive. (https://bootcamp.powercoders.org/examples/form/)

! Ex. 6.2.9 (**) - Sample Shop Styling

Style the sample shop homepage you created in the HTML exercises.

6.3 Advanced Exercises

Ex. 6.3.1 (***) - Slider

Build a webpage like this one: https://bootcamp.powercoders.org/examples/slider/

Ex. 6.3.2 (***) - Navigation

Implement a navigation (including hamburger menu for mobile version) in CSS like this one: https://bootcamp.powercoders.org/examples/nav/

7 Javascript DOM Exercises

7.1 Basic Exercises

Ex. 7.1.1 (*) - Manipulate the DOM

a) Create an HTML file with at least 3 elements: h1, p and a Assign a new variable for each of these 3 elements.
b) For the h1 variable write a loop with 2 iterations, always adding
c) the number of the iteration to the content of the tag (after the existing content).
d) For the p variable write a loop with 4 iterations, always adding the number of the iteration to the content of the tag (before the existing content).
e) For the a variable write a loop with 7 iterations, always replacing the content of the tag with the number of the iteration.

7.2 Intermediate Exercises

! Ex. 7.2.1 (**) - Book List

Create a webpage with an h1 of "My Book List". Add a script tag to the bottom of the page, where all your JS will go. Copy this array of books:

```
// Define your book array here
var books = [
    {
        title: 'The␣Design␣of␣EveryDay␣Things',
        author: 'Don␣Norman',
        alreadyRead: false
    }, {
        title: 'The␣Most␣Human␣Human',
        author: 'Brian␣Christian',
        alreadyRead: true
    }
];
```

Iterate through the array of books. For each book, create a p element with the book title and author and append it to the page.

- Use a `ul` and li to display the books.
- Add an `img` to each book that links to a URL of the book cover.
- Change the style of the book depending on whether you have read it or not.

Ex. 7.2.2 (**) - Color picker

Create color picker for a web page. Use multiple sliders for each single RGB value. Adapt the background accordingly and output the RGB value to the DOM.

Additional Task: Add number input fields which update the sliders

Additional Task: Add a HSV mode

Ex. 7.2.3 (**) - tip calculator

a) Change the code of your tip calculator to use the DOM
   - add input fields
   - add event listeners
   - add the result to the DOM

b) Add a currency converter. Use this free currency API: https://currencylayer.com.
   - add 2 additional buttons to HTML: convert to USD / convert to EUR
   - add event listeners and call the API to convert your CHF values to the chosen currency
   - add the result to the DOM

Ex. 7.2.4 (**) - move box

- Make a container with a box inside.
- Make the box move with your scroll wheel.
- Find out more about the scroll wheel event (https://www.w3schools.com/jsref/obj_wheelevent.asp)

Ex. 7.2.5 (**) - Minesweeper Game

Build the game Minesweeper. The complete guided instructions here including a template can be found: bootcamp.powercoders.org//minesweeper/minesweeper-instructions.html

Ex. 7.2.6 (**) - TO-DO List

Create a web application implementing a TO-DO list.

Figure out: What does it need? Which data structure? Where would you use functions?

Some ideas:

- user input of a new to-do.
- list all to-dos in an alphabetically order in the DOM. Or by date? what are the constraints? e.g. check if to-do already exists in your list.
- if you get stuck with the code, try to put in comments as pseudo-code first or instead.

what abou
the detail
instructio
from the
slides?

Additional Task: Store the to-dos in the local storage.

Ex. 7.2.7 (**) - Countdown Timer

Build a web application that implements a countdown timer.

- Define several pre-set timers
- Add input to have custom countdown
- Show countdown in the middle, counting down
- Bonus: Take current time and check when the countdown is done

Style like this image.

Ex. 7.2.8 (**) - Bookmark App

Create a webpage that allows you to add bookmarks to other websites, including the following features:

- Add a new bookmark (including name and URL)
- Delete a bookmark
- Visit a bookmarked website

Example layout: https://bootcamp.powercoders.org/img/powercoders-exercise-bookmark-websites-2.pdf

Additional Task: Store the bookmarks in the local storage.

7.3 Advanced Exercises

! Ex. 7.3.1 (***) - Sample Shop

We now need to add the javascript to our shop.

What do we need?

- Accounts
- Products
- Cart
- Orders

! Ex. 7.3.2 (***) - Interactive class page

Build a page for displaying all participants of your class.

Here are the constraints:

- Use CV photos of all the people in the class (available on Google Drive)
- Create content dynamically for each person in the class (e.g. name, age, eye color, hobby, etc.) using concepts learned in class, such as objects, loops and DOM manipulation
- Create at least two interactions for each profile (e.g. change image on mouse over & display more information on click)
- Create a profile classification system according to criteria you define (e.g. age, hobby, etc.)

8 Javascript API Exercises

8.1 Basic Exercises

Ex. 8.1.1 (*) - Aare temperature

Make a web page display the current temperature of the Aare river in Bern. Use the following API: https://aareguru.existenz.ch/v2018/today?city=bern

8.2 Intermediate Exercises

Ex. 8.2.1 (**) - TO-DO List 2.0

Update your TO-DO List to use the following API: https://jsonplaceholder.typicode.com

Ex. 8.2.2 (***) - User Authentication

Build a simple user authentication system. Use the following API: https://reqres.in