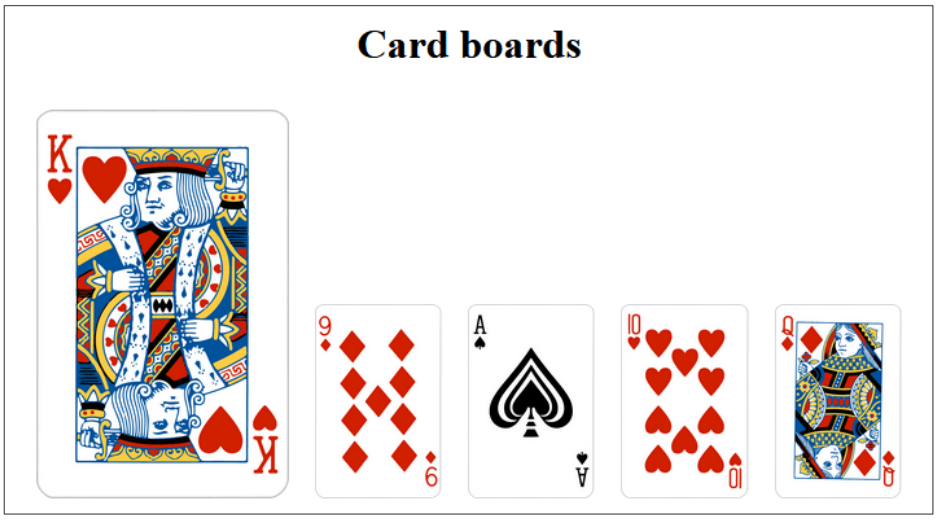
Web Programming

Tutorial 3

To begin this tutorial, please download `tut03-starter.zip` from the course website.

When you finish, zip all your deliveries to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `tut01`, `tut02`, `tut03`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

Activity 1 – Cards



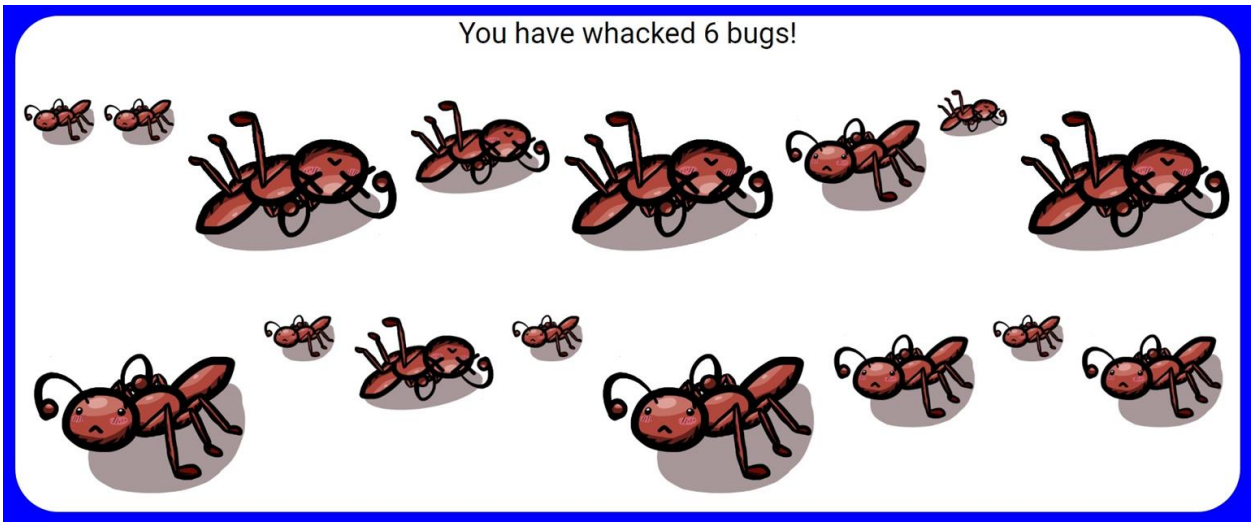
In the `cards` folder, you're given images of playing cards under the `images` folder. Create these 2 files inside the `cards` folder:

- `cards.js`: populate `index.html` with images of at least 5 cards
- `index.html`: html for card board with references to `.js` files

You have to do the following tasks:

1. In `cards.js`, create an array of links for at least 5 cards. You can use the card pictures inside `images` folder.
2. Create the card board (a container for displaying cards) in `index.html`.
3. Loop the array of images to create `img` elements add to the card board.
4. Add suitable CSS for your HTML.
5. When user chooses a card (by clicking), its height is enlarged by 1.5 times.
6. User can click another card, but only 01 card enlarged at a time.

Activity 2 – Whack-a-Bug! game



In the `whack-a-bug` folder under the starter folder. You have to define a function called `whackBug` so that the game works. Specifications of the `whackBug` function:

- Changes the image of the bug that was clicked from `bug.png` to `bug-whacked.png`
- Add the `whacked` class to the image that was clicked
- Increment the whack count by `1` (each bug should only be counted once even if clicked multiple times)
- If all 24 bugs have been whacked, change the text in the `#game p` tag to say "all bugs have been whacked".

→ **Delivery**: modified `whack.js`

Activity 3 – Simple blog

The HTML, CSS, and part of the JS for a simple blog has been provided for you in the `blog` folder under the starter folder. You are tasked with writing the `addEntry` function described below.

Specifications of the `addEntry` function:

- An article should be appended to the `#posts` container. Inside should be a third level heading followed by a paragraph. The article should have the class `.post` added to it.

- The third level heading text content should be the text `"Date: "` followed by the date submitted. The paragraph's text content should be the text `"Entry: "` followed by the entry submitted.
    - **Hint:** to grab the text from a form element such as `input` or `textarea`, use `.value` (the attribute named `value` of the DOM element object)
- If a user double clicks on any of the blog posts, the post that was double-clicked should be removed from the page.
- The content in `#date` and `#entry` should be cleared after adding the blog post.
- If the number of blog posts ever hits `3`, the “add entry!” button should be disabled.
    - **Hint:** use the `qsa` helper function to find the number of blog posts.

→ **Delivery**: modified `blog.js`

Activity 4 – 3, 2, 1, Go!

Create two HTML pages containing JavaScript code that counts down from 3 in the console. Every second for four seconds, print "3", then "2", then "1", and finally "Go!".

- There are several ways to solve this
- Create two solutions for this activity, one that uses `setTimeout` and one that uses `setInterval.` Name them `setTimeout.html` and `setInterval.html` respectively.

→ **Delivery**: `setTimeout.html` and `setInterval.html`

Activity 5 – Stopwatch

Build a `stopwatch.html` page which logs the number of seconds passed to the console. In the following specification, **you are to replace “CT” with the current time counter in any output statements.** The page should adhere to the following behaviors:

- When the `#stopwatch` button is clicked**:**
    - If the timer is not running, the stopwatch should begin counting up seconds, logging “**CT** seconds” to the console each second.
    - Otherwise, you should pause the timer interval. If the button is clicked again, the interval should be started again, continuing with the **CT** (see example output).

- When the `#reset` button is clicked, the stopwatch's **CT** should be reset to 0, whether or not it is running (continuing the timer as usual with the next output being "1 seconds" if it was running).

(*) Hint: our solution uses two module global variables, one for the timer id, one for the current second count.

→ **Delivery**: `stopWatch.html` and possibly a `.js` file