Draft documentation:
A flexible, persistent JavaScript-based soft-timeout for oTree

Faith Bague-Sampson

University of California, Irvine

September 30, 2022

A flexible, JavaScript-based soft-timer for oTree (Chen, Schonger, and Wickens, 2016) that can be per-page-submit or span multiple pages and that does NOT reset on page refresh. Each page/subpart of the experiment can have its own timer settings.

Full code available on GitHub.

1 Overview

1.1 General Features

- Each page can have its own timer or the timer can span multiple pages[1]
- The soft-timer does NOT reset on page refresh
- Each page class or subsection of an experiment can have its own timer
- The timer can be set for hours, minutes, or just seconds; its display adjusts accordingly
- Two different (and entirely optional) user-notification methods
- Primary features are controlled in the `__init__.py`; look-and-feel settings are adjusted via style sheet (included at the page-level in the sample app)

1.2 How it works:

Determine how much time (in seconds) that you would like the timer to count down and whether or not the countdown should be on a per-page/submit basis or span multiple pages/submits. You must also decide on the behavior of the timer once time expires. Should the countdown timer itself look different? Should users receive an on-page message, a pop-up notification, both, or nothing? If users are to receive a message, you must also decide on what the text should say. The

[1]The pages must be of the same template class.

answers to these six questions plus the round number are sent to the JavaScript on the page template using the timed page's `def js_vars(player)` function.

There are three HTML elements that must be incorporated on the page template, and at least one style that must be either locally specified in the template or included in the app's style sheet. These determine how the timer should look and where on the page it should be. The JavaScript code itself must be included on the page template[2], but, if the soft-timer's default behavior is acceptable, the script will not require adjustment of any kind.

2 Page settings in the __init.py__ file:

2.1 JavaScript variables sent to the page template

Necessary JavaScript variables sent to the page template

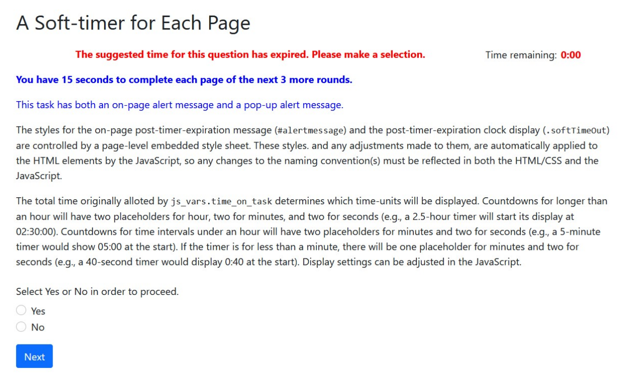
```
class MyPage1(Page):
@staticmethod
# ... other code here as needed ... #

def js_vars(player: Player):
# ... other code here as needed ... #
    return dict(
        # ... other variables sent to the page's scripts here as needed ... #
        round_number = player.round_number,
        time_on_task = TIME_IN_SECONDS,
        time_is_per_page = 'True' or 'False',
        onpage_hasAlert = 'True' or 'False',
        onpage_AlertText = "ON-PAGE_MESSAGE_CONTENT",
        popup_hasAlert = 'True' or 'False',
        popup_AlertText = "POP-UP_MESSAGE_CONTENT",
    )
# ... other code here as needed ... #
```

For each timer, the following seven variables must be included in the `def js_vars(player: Player)` dictionary supplied to the page template (e.g., in `class MyPage(Page)` on your app's `__init__.py`):

- `round_number = player.round_number`
    - This ensures that the per-page-timer functions correctly
- `time_is_per_page =` 'True' or 'False'
    - Does the timer span multiple pages ('False') or does it reset every individual page ('True')?

[2]A separate script content block, placed below the main page content, is recommended for this.

- `time_on_task` = duration of the timer (in seconds)
  - For a 30-second timer, `time_on_task = 30`
  - For a 9-minute timer, `time_on_task = 540` (i.e., 9 minutes $\times$ 60 seconds per minute = 540 seconds)
  - For an hour and 15 minute timer, `time_on_task = 4500` (i.e., 75 minutes $\times$ 60 seconds per minute = 4500 seconds)
- `onpage_hasAlert` = 'True' or 'False'
  - Determines if subjects see a persistent on-page message after time expires
  - If such a message is desired, set to 'True', otherwise set to 'False'
  - Message visibility resets with the timer
    - With a per-page timer, the message disappears when the next page loads
    - If the timer spans multiple pages/submits, the message will remain visible on any pages remaining in the timed task
- `onpage_AlertText` = "Text of the message you want shown"
  - The text contents of the persistent on-page message that subjects receive after time expires (if `onpage_hasAlert = 'True'`)
  - The text will not be shown if `onpage_hasAlert = 'False'`
  - Even when not using an on-page message, a value for this variable should still be sent to the page template. In that case, use `onpage_AlertText = ""`
- `popup_hasAlert` = 'True' or 'False'
  - Determines if subjects receive a one-time pop-up alert notifying them the instant that the timeout happens
  - If such a message is desired, set to 'True', otherwise set to 'False'
  - Once acknowledged (the subject clicks "OK"), the pop-up message will not reappear until the timer is reset
    - With a per-page timer, the timer resets with each page submit, so the pop-up notification can be triggered on every single page
    - With a multiple-page-spanning timer, the pop-up can be triggered but once no matter how many pages remain to be completed after time has expired
- `popup_AlertText` = "Text of the message you want shown"
  - The text contents of the one-time pop-up alert that subjects receive after time expires (if `popup_hasAlert = 'True'`)
  - The text will not be shown if `popup_hasAlert = 'False'`
  - Even when not using a pop-up notification, a value for this variable should still be sent to the page template. In that case, use `popup_AlertText = ""`



A Soft-timer for Each Page

The suggested time for this question has expired. Please make a selection.   Time remaining: **0:00**

You have 15 seconds to complete each page of the next 3 more rounds.

This task has both an on-page alert message and a pop-up alert message.

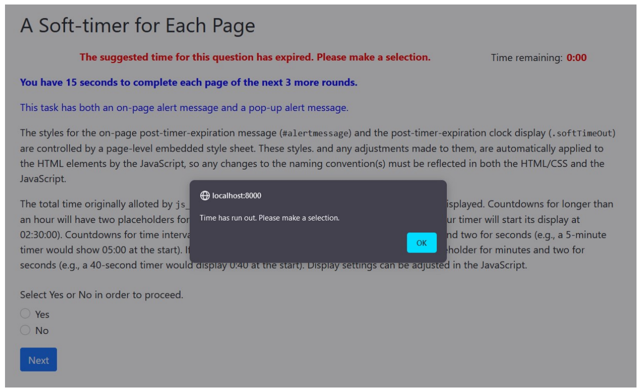
The styles for the on-page post-timer-expiration message (`#alertmessage`) and the post-timer-expiration clock display (`.softTimeOut`) are controlled by a page-level embedded style sheet. These styles. and any adjustments made to them, are automatically applied to the HTML elements by the JavaScript, so any changes to the naming convention(s) must be reflected in both the HTML/CSS and the JavaScript.

The total time originally alloted by `js_vars.time_on_task` determines which time-units will be displayed. Countdowns for longer than an hour will have two placeholders for hour, two for minutes, and two for seconds (e.g., a 2.5-hour timer will start its display at 02:30:00). Countdowns for time intervals under an hour will have two placeholders for minutes and two for seconds (e.g., a 5-minute timer would show 05:00 at the start). If the timer is for less than a minute, there will be one placeholder for minutes and two for seconds (e.g., a 40-second timer would display 0:40 at the start). Display settings can be adjusted in the JavaScript.

Select Yes or No in order to proceed.

- ○ Yes
- ○ No

(a) On-page expiration message



localhost:8000

Time has run out. Please make a selection.

(b) Pop-up expiration notification

Figure 1: **Time expiration messages.** These two screenshots show the visual and behavioral differences between an on-page message and a pop-up notification. User notifications about expired time are not mutually exclusive and can be used in conjunction if desired, as in Figure 1b.

2.2 Examples

A 5-minute soft-timer spanning multiple pages that uses both an on-page expiration message and a pop-up alert

```
class MyPage(Page):
@staticmethod
# ... other code here as needed ... #

def js_vars(player):
# ... other code here as needed ... #
    return dict(
        # ... other variables sent to the page's scripts here as needed ... #
        round_number = player.round_number,
        time_on_task = 300,
        time_is_per_page = 'False',
        onpage_hasAlert = 'True'
        onpage_AlertText = "Time has expired. Please make your selection and proceed.",
        popup_hasAlert = 'True',
        popup_AlertText = "The suggested time has expired.",
    )
# ... other code here as needed ... #
```

A 10-second soft-timer for each page that uses an on-page expiration message but no pop-up alert

```
class MyPage(Page):
@staticmethod
# ... other code here as needed ... #

def js_vars(player):
# ... other code here as needed ... #
    return dict(
        # ... other variables sent to the page’s scripts here as needed ... #
        round_number = player.round_number,
        time_on_task = 10,
        time_is_per_page = ‘True’,
        onpage_hasAlert = ‘True’,
        onpage_AlertText = "Hurry up! Decide already!",
        popup_hasAlert = ‘False’,
        popup_AlertText = "",
    )
# ... other code here as needed ... #
```

3 Code on the `MyPage.html` template

What must be on the `MyPage.html` template for the timer to function properly:

```
{{ block styles }}
    <style>
        .softTimeOut { /* Sets styles for the expired countdown clock */
            color: red ;
            font-weight: bold ;
        }

        #alertmessage { /* Sets styles for the alert message on the page */
            color: red ;
            font-weight: bold ;
            text-align:center;
        }

        /* #CountdownClock {
            /* styles for the running countdown timer are set here
        } */
    </style>
{{ endblock }}

{{ block title }}
    MyPage Title Here
{{ endblock }}

{{ block content }}
    <div style="float: right;">
        <table style="border: none;" width="200px">
            <tr>
                <td style="border: none;" width="120px">Time remaining:</td>
                <td style="border: none;" width="80px"><span id="CountdownClock"></span></td>
            </tr>
        </table>
    </div>
    <span style="display:none;" id="timer"></span> <!-- NECESSARY for "ticking" to function -->

    <p id="alertmessage"></p> <!-- Optional on-page message once time expires -->

    <!-- Rest of the content for the page -->
    {{ formfields }}
    {{ next_button }}
{{ endblock }}

{{ block scripts }}
    <script>
        JAVASCRIPT_SOFT-TIMER_CODE_HERE
    </script>
{{ endblock }}
```

Styles automatically applied by the JavaScript upon the expiration of the timer.

HTML added to page template

This span tag is responsible for displaying the countdown timer.

The countdown timer does NOT work without this *hidden* tag.

Optional on-page timer-expiration message.

The JavaScript code is added after the block content in its own script block.

3.1 Template Styles

```
<style>
    /* other styles here */

    .softTimeOut {  /* NECESSARY to include: expired countdown clock */
        /* set styles here */
    }

    #alertmessage {  /* entirely OPTIONAL: on-page expiration message */
        /* set styles here */
    }

    /* other styles here */
</style>
```

The styles for the post-timer-expiration clock display (`.softTimeOut`) and the on-page post-timer-expiration message (`#alertmessage`) can be controlled by a linked or page-level embedded style sheet. Styles for `#alertmessage` are entirely optional (i.e., they can be removed from the style sheet) even if the element itself is not. Further recall that the on-page message only appears if `onpage_hasAlert = 'True'`, as this element is otherwise invisible. The style dictated by the `softTimeOut` class, however, is always applied by the JavaScript the instant time expires. Because the JavaScript is programmed to append this class name specifically, any changes to its name in the style sheet must also be reflected in the JavaScript code. If no visual distinction between a running timer or an expired timer is desired, it is best to include placeholder code for the `softTimeOut` class by having the class "inherit" its parent-element styles. The countdown timer can be visually set apart from other elements on the page by creating styles for `#CountdownClock`.

Example of placeholder code for the `softTimeOut` class:

```
<style>
    /* other styles here */

    .softTimeOut {  /* NECESSARY to include: expired countdown clock */
        color: inherit;
        font-weight: inherit;
    }

    /* other styles here */
</style>
```

3.2 Template HTML

The HTML elements required for full countdown functionality

```
<div style="float: right;">
    <table style="border: none;" width="200px">
        <tr>
            <td style="border: none;" width="120px">Time remaining:</td>
           <td style="border: none;" width="80px"><span id="CountdownClock"></span></td>
        </tr>
    </table>
</div>
<span style="display:none;" id="timer"></span> <!-- NECESSARY for "ticking" to function -->

<p id="alertmessage"></p> <!-- Optional on-page message once time expires -->

<!-- Rest of the content for the page -->
```

Note on the color-coding:

The red-colored elements MUST be present, but the blue-colored tags to which they are attributes is discretionary.

The three HTML elements controlled by the JavaScript are the red-colored tags with the `id`'s `"timer"`, `"CountdownClock"`, and `"alertmessage"`. Note that the display of the countdown timer is handled separately from the ticking of the interval process in the JavaScript code. The `"timer"` tag – without which this script CANNOT function as it is responsible for tracking the passage of time[3] – should be set to `style="display:none;"`.

The `"CountdownClock"` tag displays the human-readable countdown timer; this is the timer that the subjects actually see. It is recommended that `<span id="CountdownClock"></span>` is placed in a separate, fixed-width HTML tag not shared with any other text or elements (in the sample code, "Time remaining" is in its own `<td>` tag). Any contents in a shared tag will flash upon page refresh, and any non-fixed-width parent element will shift its position on the page because its width changes in the milliseconds between the JavaScript/server calls for the time remaining and the browser visually rendering that time on the page. This behavior can be quite distracting.

The tag `"alertmessage"` is entirely optional (see the red text in Figure 1a for an example). Even if included on your page template, `onpage_hasAlert` must be set to `'True'` in the class settings for this page in order for text to appear once time expires. Once visible [upon timeout], the message will not disappear until the timer is reset (this could be accomplished by clicking "Next" if the timer is per-page or in several pages when the subject reaches the next subsection of the experiment if the timer spans multiple pages). The text content for this element is set with the `onpage_AlertText` variable, which is also located in `js_vars()` function for this page class. Do consider how the page content might shift with the appearance of the timeout message, and design your page accordingly. Some experiments might benefit from

[3]Technically, timers for 59 seconds or fewer can function without this element; however, to ensure full flexibility for longer timeout intervals, it is recommended that it is always included no matter the length of the time interval.

assigning the message to elements with an absolute position and height – even when invisible.

4 Code-chunk-by-code-chunk breakdown of the JavaScript

For those who might want to make adjustments to the timer, the following is an explanation of what the JavaScript code is doing line-by-line, starting from the top of the code.

Note: in this section, custom functions will be red-colored (including the parentheses), user-defined variables/input-values will be blue-colored, and all other code – including built-in JavaScript functions – will be black.

4.1 Declaration of variables that will be used in multiple functions

Constants and variables declared at the top for use in any subsequent function

```
const popup_AlertText = js_vars.popup_AlertText;
var start_time, countAmt, interval;
var tickingTimer = document.getElementById("CountdownClock");
var timeOverMsg = document.getElementById("alertmessage");
```

- `const popup_AlertText = js_vars.popup_AlertText;` – Sets the value of the text for the pop-up notification[4].
- `var start_time, countAmt, interval;` – Declares the existence of these variables for use across multiple functions. These three variables are used to track the countdown of time.
- `var tickingTimer = document.getElementById("CountdownClock");` – Finds the element `id="CountdownClock"` in the HTML code on the page, assigns it as the variable `tickingTimer`. This allows the script to update that element on the page.
- `var timeOverMsg = document.getElementById("alertmessage");` – Finds the element `id="alertmessage"` in the HTML code on the page, assigns it as the variable `timeOverMsg`. This allows the script to update that element on the page.

4.2 Function that adds a 0 to any single-digit number

The `addZero()` function adds a placeholder "0" to the 10's position of any single-digit positive number

```
function addZero(obj) {
    if (obj < 10) {obj = "0" + obj}
    return obj;
}
```

[4]For whatever reason – message me if you know why and how I can fix it – the pop-up notification will not function unless the text is either specified in-line or as a constant before the conditional statement in the function. Declaring the value here allows experimenters to set the text without editing the JavaScript.

The function `addZero()` takes the number `obj`, tests if its value is less than 10, and, if this is true, adds a "0" to it. The function then returns this transformed number, `obj`, as a *string*. Note that negative numbers are by definition less than 10, so the original `obj` sent to `addZero()` must be non-negative. Only numbers less than 10 are returned as strings; the input is returned as-is if the condition fails.

For example, the following code takes the number 8 as an input and outputs a new variable, `newValue`, that is the string `'08'`:

```
const myValue = 8;
let newValue = addZero(myValue);
```

Whereas with this code, because $23 > 10$, no 0 is added to the number when it is returned. `newValue = 23`:

```
const myValue = 23;
let newValue = addZero(myValue);
```

4.3 Function translates a quantity of seconds into its constituent hours, minutes, and seconds

The `makeitBaseSixty()` function creates time-units for display

```
function makeitBaseSixty(remainingTime) {
    var hoursLeft = Math.floor(remainingTime/3600);
    var minutesLeft = Math.floor((remainingTime - hoursLeft*3600)/60);
    var secondsLeft = remainingTime -  Math.floor(remainingTime/60)*60;
    let h = addZero(hoursLeft);
    let m = addZero(minutesLeft);
    let s = addZero(secondsLeft);
    return {h, m, s};
}
```

The function `makeitBaseSixty()` is called in the fourth line of the `tick()` function[5]. `makeitBaseSixty()` takes a certain number of seconds, `remainingTime`, and converts this value the time-units hours, minutes, and seconds. Note that the built-in JavaScript function `Math.floor()` rounds its argument down to the nearest whole number, thus eliminating any decimal values.

- `var hoursLeft = Math.floor(remainingTime/3600);` – Returns the number of hours in any quantity of seconds (there are 3600 seconds in one hour).
- `var minutesLeft = Math.floor((remainingTime - hoursLeft*3600)/60);` – Returns the number of minutes under 60 remaining (after the number of hours have been accounted for) in any quantity of seconds.

[5]Where that line is: `let displayClock = makeitBaseSixty(timeLeftOver);`

- `var secondsLeft = remainingTime - Math.floor(remainingTime/60)*60;` – Returns the number of seconds under 60 remaining (after the number of both hours and minutes have been accounted for) in any quantity of seconds.
- `let h = addZero(hoursLeft);` – If `hoursLeft` < 10, returns hours as a two-digit text string with 0 as the first number (e.g., if there is 1 hour left, `addZero(hoursLeft);` returns `'01'` as a text string.), otherwise returns the number of hours as a number.
- `let m = addZero(minutesLeft);` – If `minutesLeft` < 10, returns minutes as a two-digit text string with 0 as the first number, otherwise returns the number of minutes as a number.
- `let s = addZero(secondsLeft);` – If `secondsLeft` < 10, returns seconds as a two-digit text string with 0 as the first number, otherwise returns the number of seconds as a number.
- `return {h, m, s};` – Returns the time-unit quantity of hours, `h`, minutes, `m`, and seconds, `s`, remaining in the countdown as a JavaScript object that can be referenced by `VARIABLE_NAME.js_object` (i.e., the original code assigned the variable `displayClock` to this JavaScript object. Therefore, to access the number of hours, `h`, one would use `displayClock.h`.)

A pseudo-code demonstration of how the function works:

```
remainingTime = 3853;
function makeitBaseSixty(3853) {
    var hoursLeft = Math.floor(3853/3600) = Math.floor(1.070277) = 1;
    var minutesLeft = Math.floor((3853 - hoursLeft*3600)/60) = Math.floor((3853 - 1*3600)/60) .....
                = Math.floor((3853 - 3600)/60) = Math.floor(253/60) = Math.floor(4.2166) = 4;
    var secondsLeft = remainingTime - Math.floor(remainingTime/60)*60 = 3853 - Math.floor(3853/60)*60 .....
                = 3853 - Math.floor(64.2166)*60 = 3853 - 64*60 = 3853 - 3840 = 13;
    let h = addZero(hoursLeft) = addZero(1) = "01";
    let m = addZero(minutesLeft) = addZero(4) = "04";
    let s = addZero(secondsLeft) = addZero(13) = 13;
    return {h, m, s} = {"01", "04", 13};
}
```

4.4 Support function for `tick()` and `startTimer()`

The `now()` function gets the current time in terms of milliseconds since 01/01/1970

```
function now() {
    return ((new Date()).getTime());
}
```

The function `now()` is called in both the `startTimer()` and the `tick()` functions. `now()` returns the output from the built-in JavaScript method, `getTime()`, which reports the number of milliseconds that have past since the ECMAScript epoch (January 1, 1970). Specifically, `(new Date())` tells `getTime()` to compute the number of milliseconds starting from the current time. For more information on this built-in method, please look through JavaScript documentation on the matter.

4.5 Function that runs the countdown

```
function tick() {
    var elapsed = now() - start_time;
    var cnt = countAmt - elapsed;
    let timeLeftOver = Math.round(cnt / 1000);
    let displayClock = makeitBaseSixty(timeLeftOver);
    sessionStorage.setItem("timeLeftOver", timeLeftOver);
    if (cnt > 0) {
        if (js_vars.time_on_task > 3599) {
            tickingTimer.innerHTML = displayClock.h + ":" + displayClock.m + ":" + displayClock.s;
        } else if (js_vars.time_on_task < 3600 && js_vars.time_on_task > 59) {
            tickingTimer.innerHTML = displayClock.m + ":" + displayClock.s;
        } else {
            tickingTimer.innerHTML = 0 + ":" + displayClock.s;
        }
    } else {
        if (js_vars.time_on_task > 3599) {
            tickingTimer.innerHTML = "00:00:00";
        } else if (js_vars.time_on_task < 3600 && js_vars.time_on_task > 59) {
            tickingTimer.innerHTML = "00:00";
        } else {tickingTimer.innerHTML = "0:00";}
        tickingTimer.classList.add('softTimeOut');
        if (js_vars.onpage_hasAlert == 'True') {
            timeOverMsg.innerHTML = js_vars.onpage_AlertText; // on-page message
        }
        if (js_vars.popup_hasAlert == 'True') {
            alert(popup_AlertText); // popup message
        }
        clearInterval(interval);
    }
    sessionStorage.setItem("clockLeftOver", tickingTimer.innerHTML);
}
```

The function `tick()` is called every second[6] from the `startTimer()` function, which contains JavaScript's `setInterval()` method. This function is responsible for the counting down action of the timer itself! Therefore, it is recommended that, unless you know what you are doing with JavaScript or have much time to dedicate to the task, the only code altered is

[6]Where it is called in the ninth line: `interval = setInterval(tick, 1000);`

that determining the output of the `.innerHTML`.

- First, note that because `tick()` is empty of argument means that the variable(s) upon which it relies must be pre-declared beforehand (i.e., that's why '`var start_time, countAmt, interval;`' is at the top of the script and outside of any specific function).
- `var elapsed = now() - start_time;` – Measures the time elapsed since `start_time` was recorded in the `startTimer()` function.
- `var cnt = countAmt - elapsed;` – `cnt` is the time left over in the countdown; this is the variable actually "ticking" down. `countAmt`, the starting time in milliseconds, is assigned in the `startTimer()` function.
- `let timeLeftOver = Math.round(cnt / 1000);` – `cnt` is measured in milliseconds, therefore is must be converted back into whole seconds (there are 1000 milliseconds in one second).
- `let displayClock = makeitBaseSixty(timeLeftOver);` – `displayClock` is a JavaScript object created by sending the number of seconds remaining in the countdown, `timeLeftOver`, to the function `makeitBaseSixty()`, which converts seconds into the time units of hours, minutes, and seconds: `displayClock` = `{h, m, s}`. In general, such objects are accessed in this manner: `parent_var_name.child_name_1`. Therefore, the number of minutes remaining is accessed with `displayClock.m`.
- `sessionStorage.setItem("timeLeftOver", timeLeftOver);` – This stores the current value of the remaining seconds in the timer in the browser's `sessionStorage`, where `"timeLeftOver"` is the name of the stored object, and `timeLeftOver` is the variable being stored under that name. The value updates every second[7].
- `if (cnt > 0) { \\yes-code\\ } else { \\no-code\\ }` – This if-statement checks whether or not there is any time remaining; if there is, the `yes-code` is executed, otherwise the `no-code` is executed. Within both the `yes-code` and `no-code` blocks are the further conditional statements:
  - `if (js_vars.time_on_task > 3599) { \\yes-code\\ }` – This checks if the starting time was more than an hour in length.
  - `else if (js_vars.time_on_task < 3600 && js_vars.time_on_task > 59) { \\yes-code\\ }` –

[7]To see what is being recorded in the browser, press `Ctrl+Shift+i`, then navigate to the "Storage" tab/section of the inspector, and find "Session Storage."

4.6 Function that begins the countdown

```
function startTimer(secs) {
    clearInterval(interval);
    if (js_vars.time_on_task > 59) {
        document.getElementById("timer").innerHTML = secs; // NECESSARY for "ticking" to function
    } else {
        document.getElementById("CountdownClock").innerHTML = 0 + ":" + addZero(secs);
    }
    countAmt = secs * 1000;
    start_time = now();
    interval = setInterval(tick, 1000);
}
```

4.7 Function that determines the start time

```
document.addEventListener('DOMContentLoaded', (event) => {
        if (!sessionStorage.getItem('roundNumber')) {
            sessionStorage.setItem('roundNumber', js_vars.round_number);
            sessionStorage.setItem('timeLeftOver', js_vars.time_on_task);
            sessionStorage.removeItem('clockLeftOver');
            startTimer(js_vars.time_on_task);
        } else if (js_vars.time_is_per_page == 'True' && .....
                ..... sessionStorage.getItem('roundNumber') < js_vars.round_number) {
            sessionStorage.setItem('roundNumber', js_vars.round_number);
            sessionStorage.setItem('timeLeftOver', js_vars.time_on_task);
            sessionStorage.removeItem('clockLeftOver');
            startTimer(js_vars.time_on_task);
        } else {
            let newTime = sessionStorage.getItem('timeLeftOver');
            let oldClock = sessionStorage.getItem('clockLeftOver');
            if (newTime <= 0) {
                if (js_vars.time_on_task > 3599) {
                    tickingTimer.innerHTML = "00:00:00";
                } else if (js_vars.time_on_task < 3600 && js_vars.time_on_task > 59) {
                    tickingTimer.innerHTML = "00:00";
                } else {tickingTimer.innerHTML = "0:00";}
                tickingTimer.classList.add('softTimeOut');
                if (js_vars.onpage_hasAlert == 'True') {
                    timeOverMsg.innerHTML = js_vars.onpage_AlertText;
                }
            } else {
                tickingTimer.innerHTML = oldClock;
                startTimer(newTime);
            }
        }
    });
```

References

Chen, Daniel L., Martin Schonger, and Chris Wickens. 2016. "OTree—an Open-Source Platform for Laboratory, Online, and Field Experiments." *Journal of Behavioral and Experimental Finance* 9:88–97.