

# Experimental design and analysis

Experiment programming

<https://www.lri.fr/~appert/eval/>

# Project: hypotheses to test

Pick two visual variables of your choice (e.g., color, size, shape, shadow, etc.). Let's call them  $VV_1$  and  $VV_2$ .

Research hypotheses:

$H_1$ :  $VV_1$  is preattentive

$H_2$ :  $VV_2$  is preattentive

$H_3$ :  $VV_1$  and  $VV_2$  combined are less preattentive than  $VV_1$  or  $VV_2$  in isolation

# Operationalization

We refine what *preattentive* means:

A visual variable is preattentive when the **visual search time** for the only object that differs from a collection because of this visual variable **is not affected by the number of objects in the collection.**

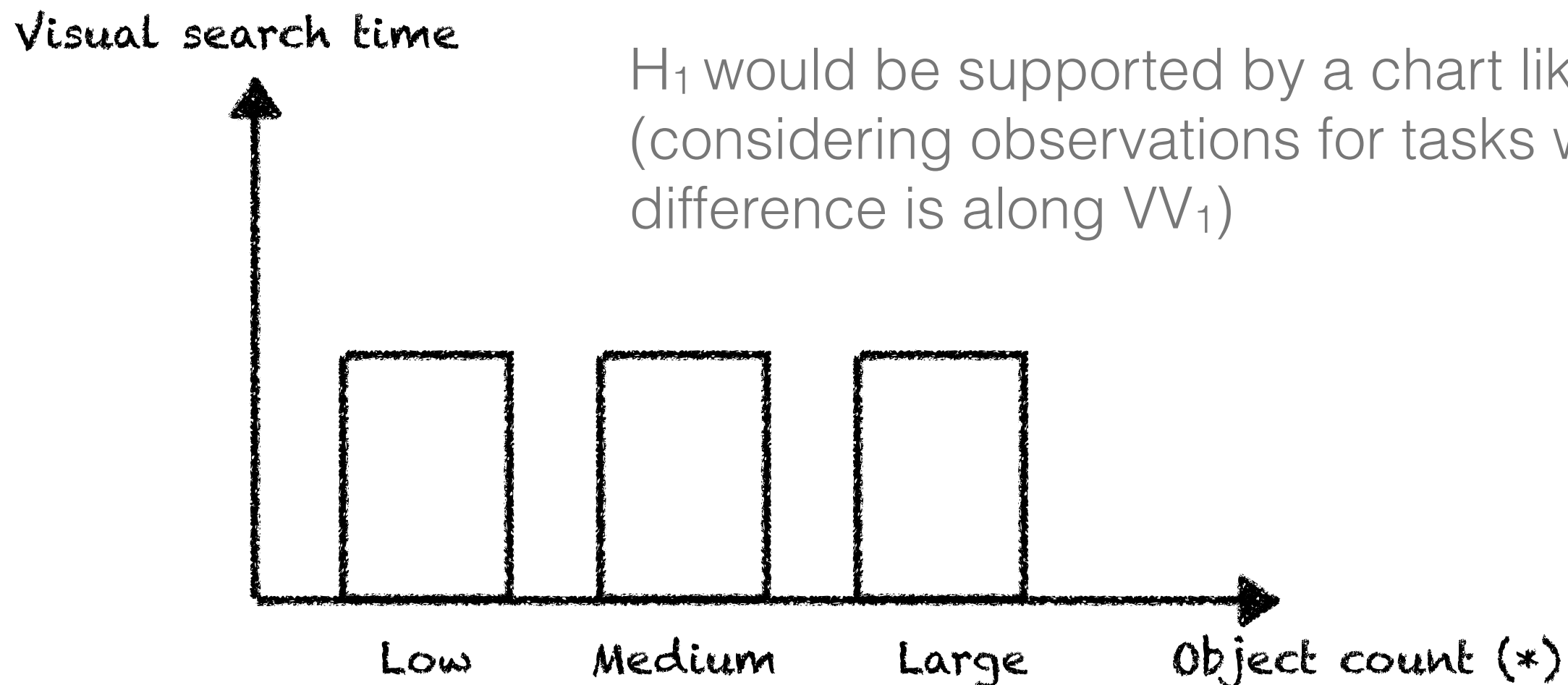
# Operationalization

We identify factors and measures

Sketching the charts you would like to report helps a lot

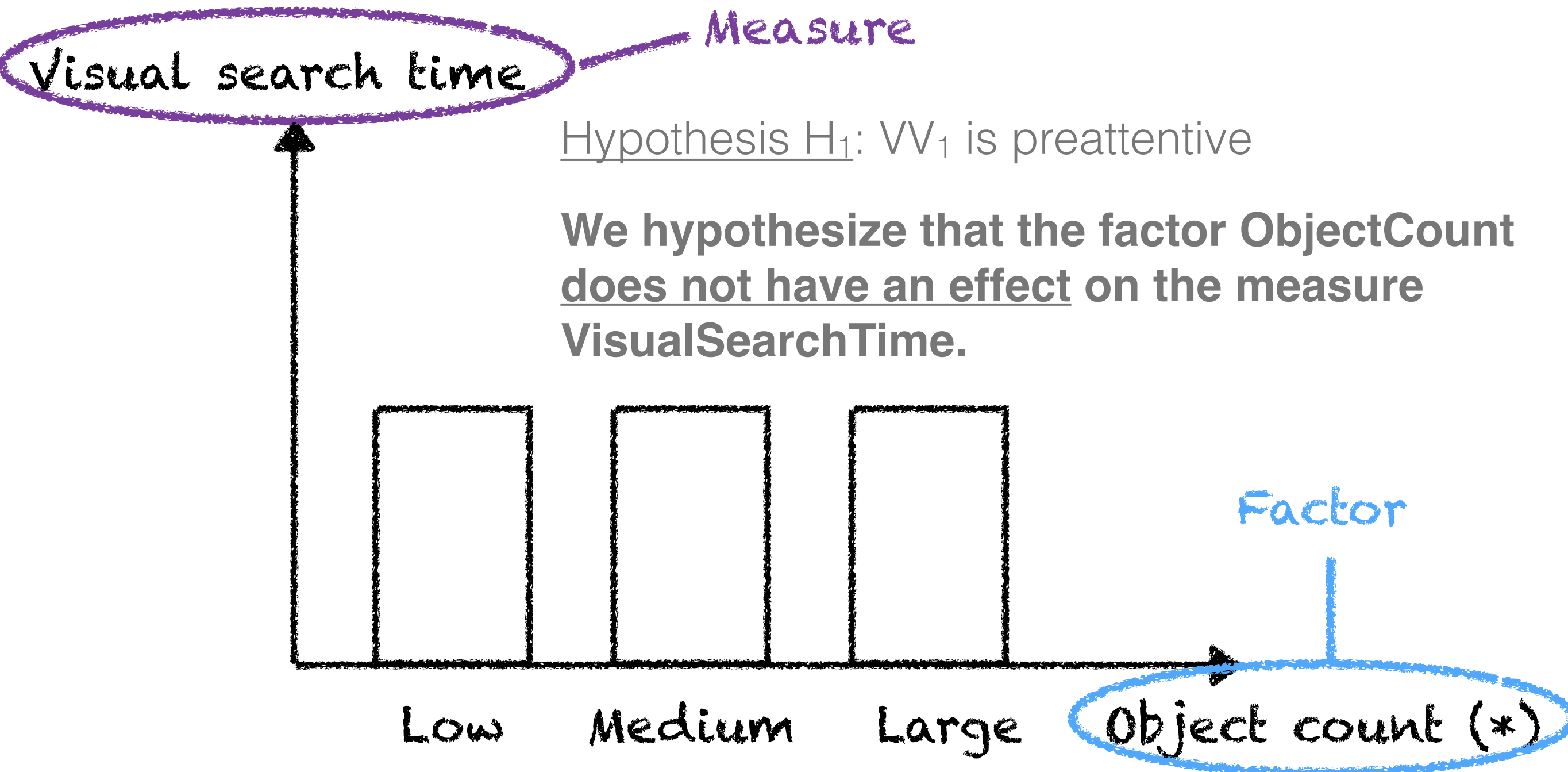
Hypothesis  $H_1$ :  $VV_1$  is preattentive

$H_1$  would be supported by a chart like this one  
(considering observations for tasks where the  
difference is along  $VV_1$ )



(\*) Number of objects in the collection

# Operationalization



(\*) Number of objects in the collection

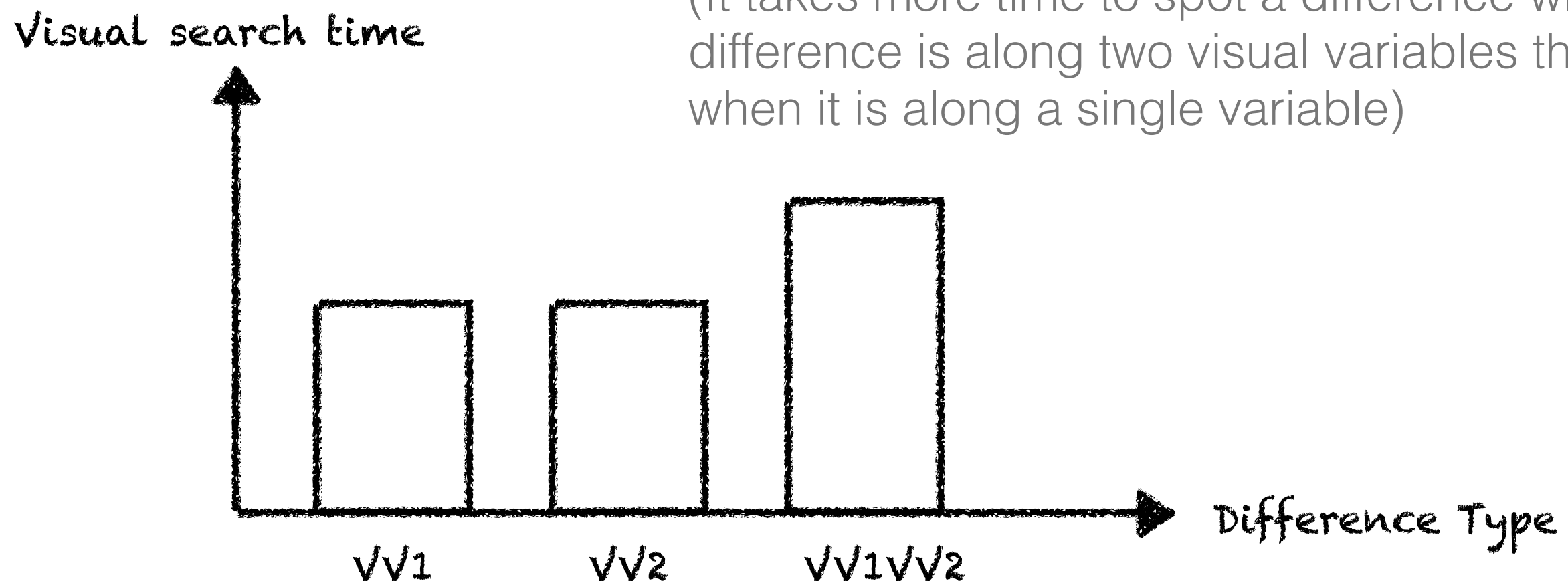
# Operationalization

We identify factors and measures

Sketching the charts you would like to report helps a lot

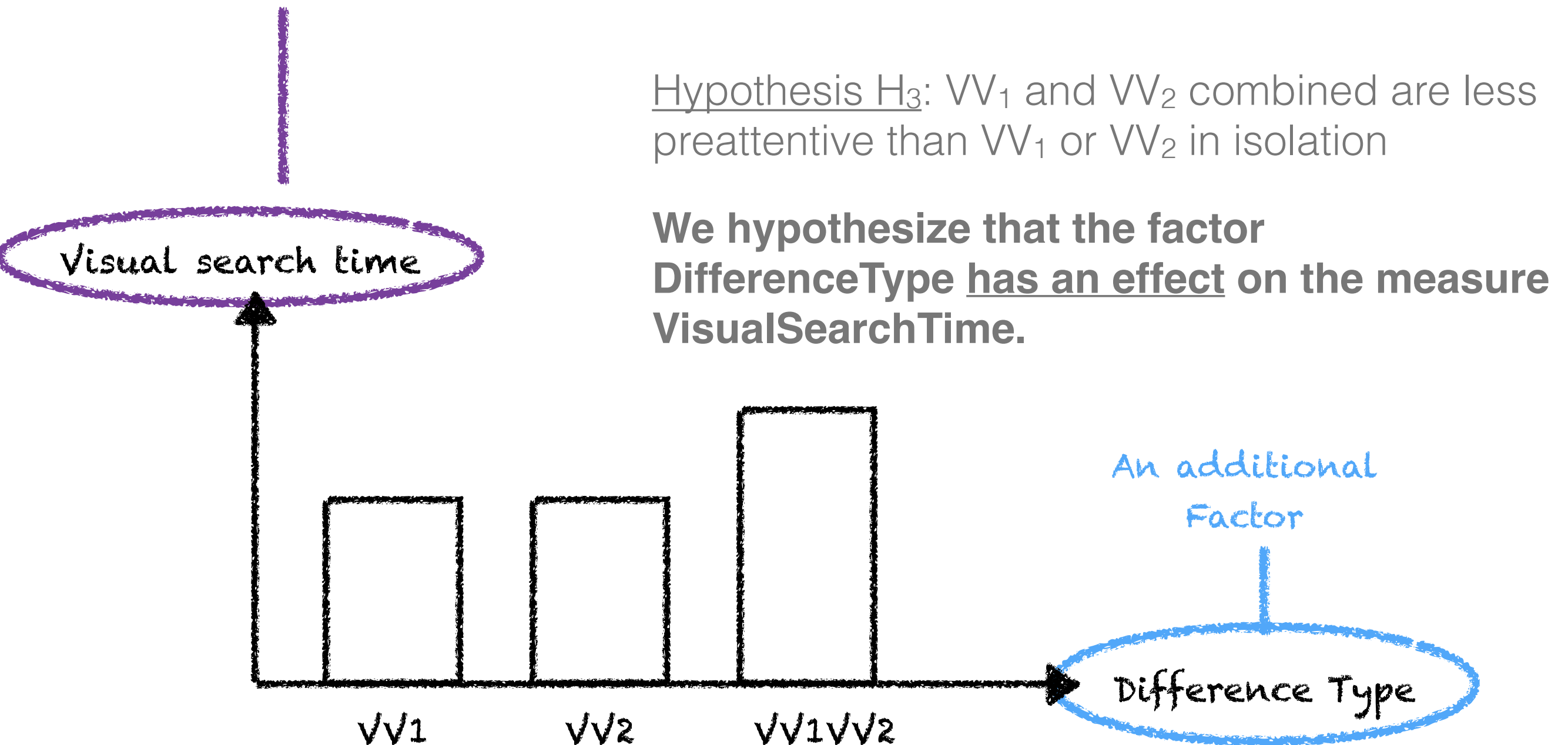
Hypothesis H<sub>3</sub>:  $VV_1$  and  $VV_2$  combined are less preattentive than  $VV_1$  or  $VV_2$  in isolation

(It takes more time to spot a difference when the difference is along two visual variables than when it is along a single variable)



# Operationalization

Measure (which we had already identified for the first two hypotheses)



# Operationalization

Factors:

OC: ObjectCount

{Low, Medium, Large}

DT: Difference Type

{VV1, VV2, VV1VV2}



Task?

Measure:

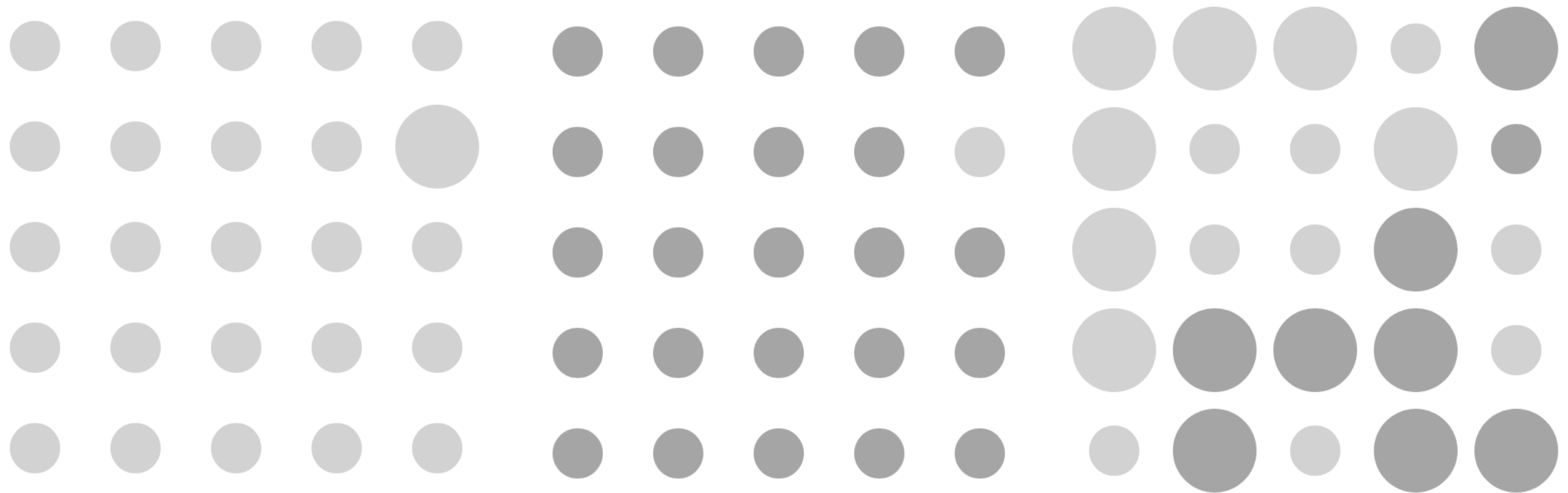
Visual Search Time



# Operationalization - task

Stimulus: present a collection of objects where only one object is different from all the other objects

*Example with  $VV1=Size$  and  $VV2=Color$*



DT=VV1

DT=VV2

DT=VV1VV2

# Operationalization - task

Response: the participant finds the different object

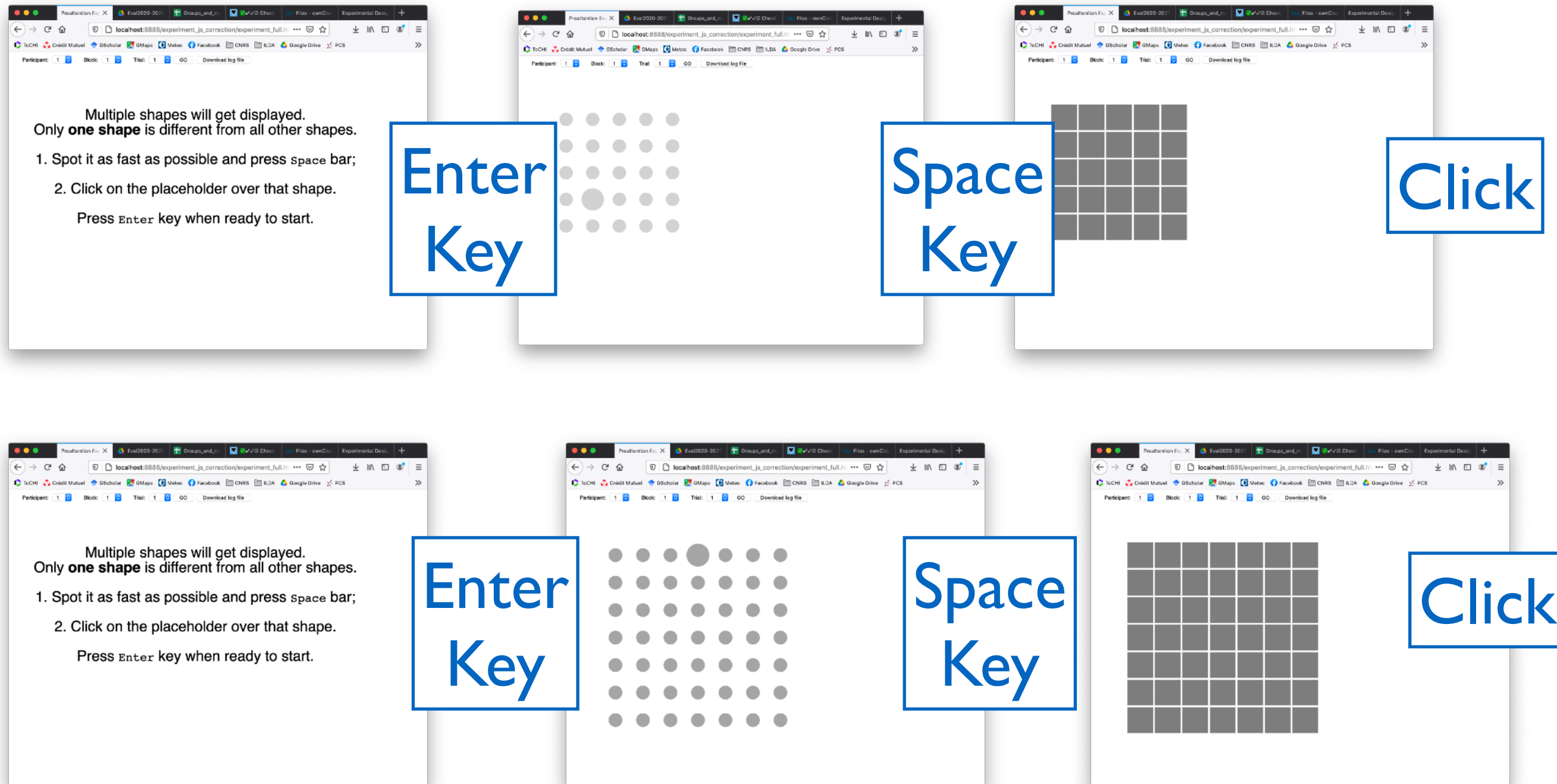
How to measure visual search time?

Only visual search: stop timer at ~~pointing~~ (no!), stop timer at key press (yes!)

Make sure the participant spots the right object: 2-step task with first key press then click on placeholders

Avoid an "animation effect" by using placeholders that are clearly different from all objects in the collection

# Experiment storyboard



# Operationalization - task

Make sure that you give **all the necessary details** to ensure replicability of your experiment by others

Participants must identify the object with a unique appearance as quickly as possible. They press 'Enter' when ready, and a scene with multiple shapes appears, starting the timer. As soon as they spot the distinct object, they press 'Space', stopping the timer. All objects are thus turned into generic square placeholders at their original locations... etc.

# Make the design formal with TouchStone 2 (20')

Two constraints:

You need at least 30 measures per condition overall  
to run inferential statistics

You have access to 6 participants

# TouchStone 2

PreattentionExperiment ■

DT DifferenceType

- Size
- Color
- ColorSize

Latin square of 1 replication(s) not serial

OC ObjectCount

- Low
- Medium
- High

Latin square of 6 replication(s) serial

Suitable for a multiple of 3 Participant(s)  
I plan to recruit 6 Participant(s)  
Order effect coverage 100%

Average duration per trial 2 sec  
Delay after each trial 2 sec  
Delay after each block 10 sec  
Each session takes 00:03:46 per participant

# Design description

Here again, make sure that you give **all the necessary details** to ensure replicability of your experiment.

The experiment tests two factors (DT and OC) according to a within-subject design.

Trials are blocked by DifferenceType (DT), and then by ObjectCount (OC). Presentation order of DT blocks and OC sub-blocks are counterbalanced using a LatinSquare. Each sub-block contains 6 replications of the task in the corresponding DTxOC condition.

In total, we collect:

6 participants x 3 DT x 3 OC x 6 replications = 324 trials

# Experiment programming



# Experiment programming I/O

experiment design (TouchStone csv output)

```
DesignName,ParticipantID,TrialID,Block1,Block2,DT,OC  
PreattentionExperiment,1,1,1,1,Color,Medium  
PreattentionExperiment,1,2,1,1,Color,Medium  
PreattentionExperiment,1,3,1,1,Color,Medium  
...  
PreattentionExperiment,6,268,3,3,Color,Medium  
PreattentionExperiment,6,269,3,3,Color,Medium  
PreattentionExperiment,6,270,3,3,Color,Medium
```

experiment\_touchstone2.csv



experiment program



log file (csv file for your statistical analyses)

```
DesignName,ParticipantID,TrialID,Block1,Block2,DT,OC,visualSearchTime,ErrorCount  
PreattentionExperiment,1,1,1,1,Color,Medium,1632,0  
PreattentionExperiment,1,2,1,1,Color,Medium,1552,1  
PreattentionExperiment,1,3,1,1,Color,Medium,2030,0  
...
```

# Getting started

Download JavaScript code skeleton on class website

## Experimental Design and Analysis

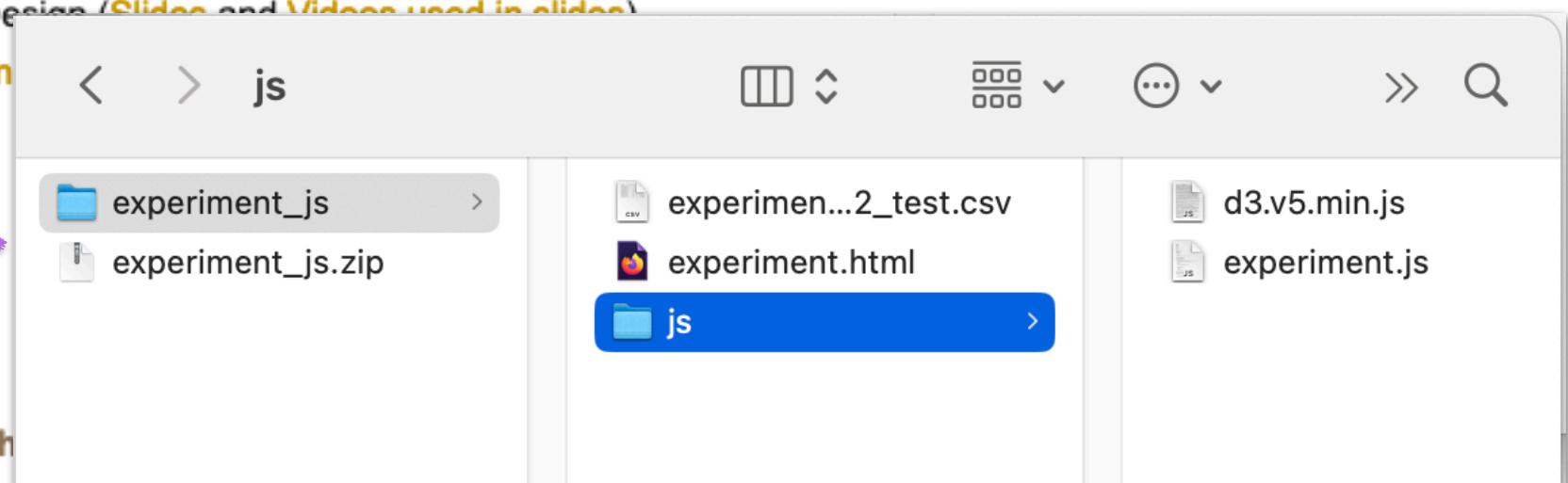
Caroline Appert

### Class material

- Class 1: Introduction to Experimental Design (**Slides** and **Videos used in slides**)
- Class 2: Statistical analyses for Laboratory Experiments (**Slides**)
- Class 3: Hands-on approach to Experimental Design (**Slides** and **Videos used in slides**)
  - Touchstone 2: **online tool**, and **introduction**
- Class 4: Experiment Programming
  - Project start code: **Download**

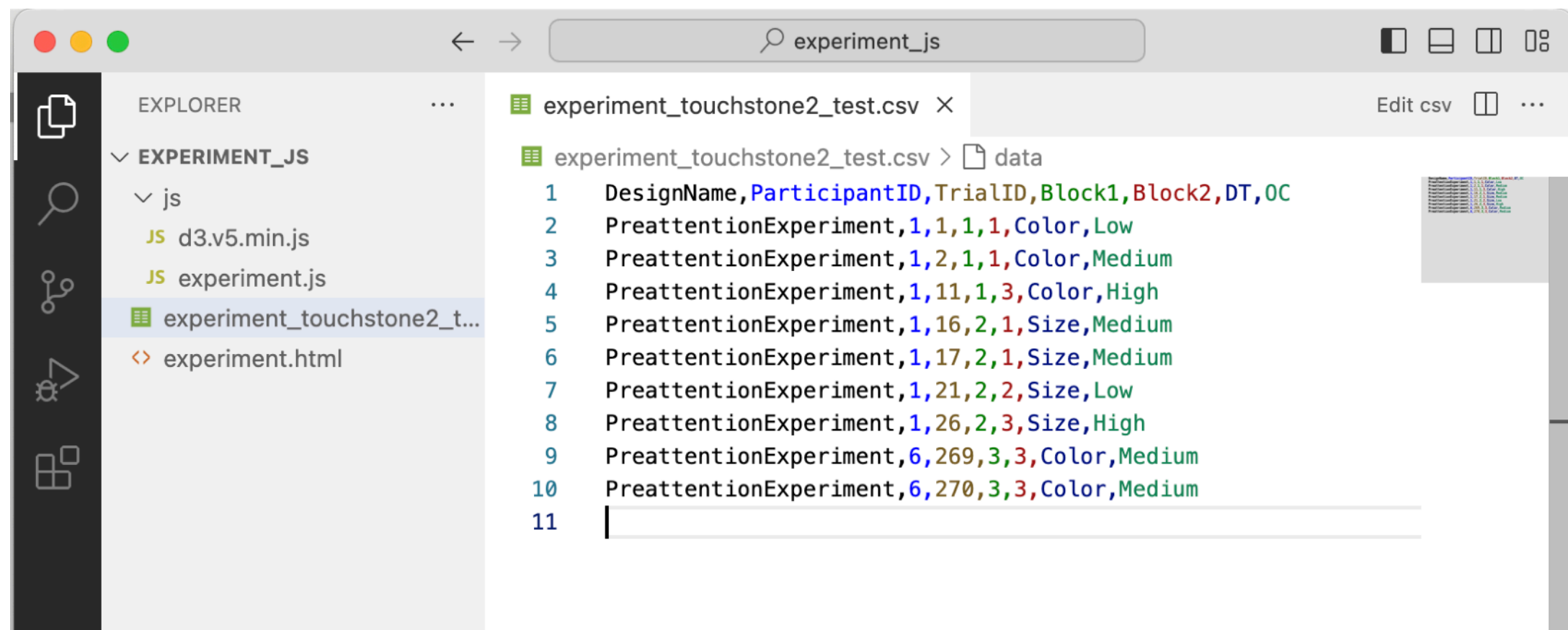
### Project

IMPORTANT: Register your group in **this spreadsheet**



# Getting started

`experiment_touchstone2_test.csv` is just an excerpt from a TouchStone2 design file. You have to replace it with your own design.



The screenshot shows a web browser window with the address bar displaying `experiment_js`. The left sidebar contains an 'EXPLORER' panel with a tree view showing the following structure:

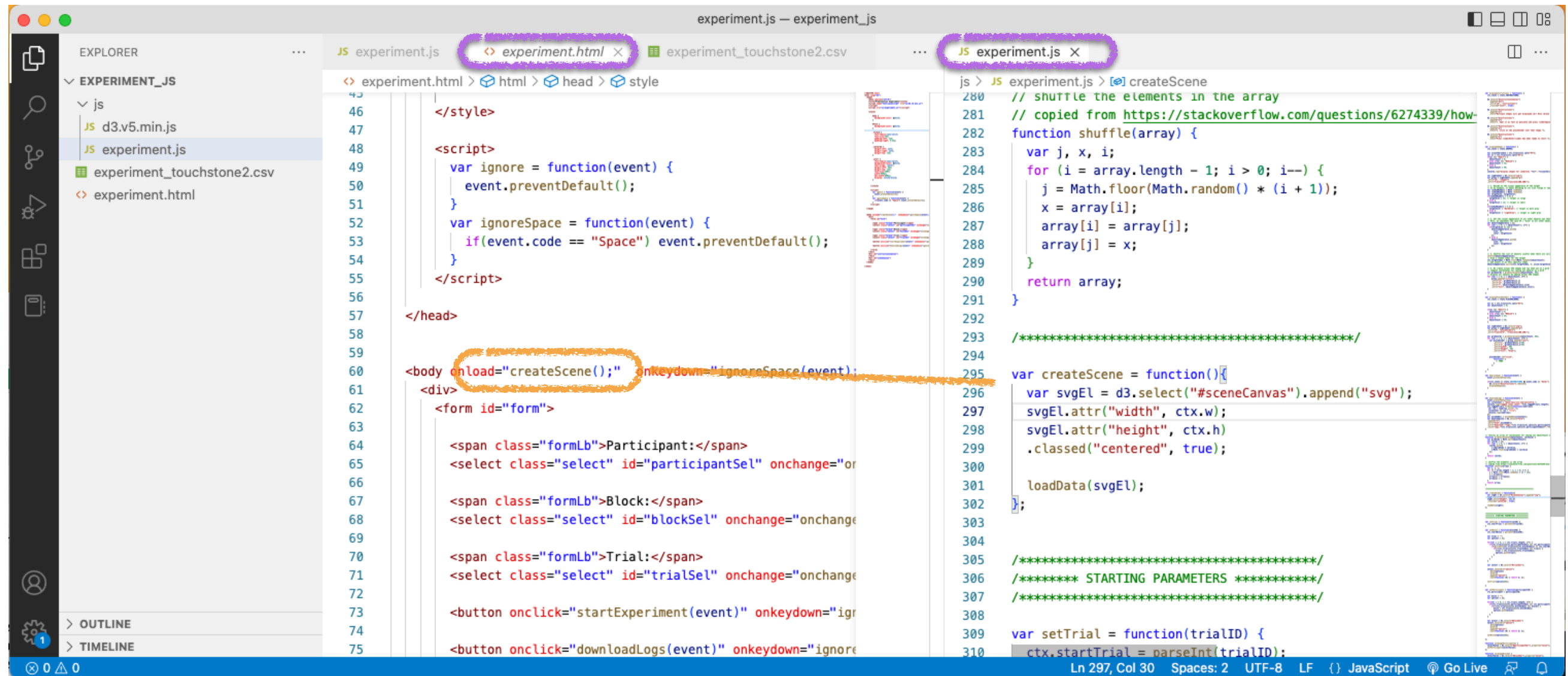
- EXPERIMENT\_JS
  - js
    - d3.v5.min.js
    - experiment.js
    - experiment\_touchstone2\_t...
    - experiment.html

The main content area displays the file `experiment_touchstone2_test.csv` with a tab labeled 'Edit csv'. The file content is as follows:

```
experiment_touchstone2_test.csv > data
1  DesignName,ParticipantID,TrialID,Block1,Block2,DT,0C
2  PreattentionExperiment,1,1,1,1,Color,Low
3  PreattentionExperiment,1,2,1,1,Color,Medium
4  PreattentionExperiment,1,11,1,3,Color,High
5  PreattentionExperiment,1,16,2,1,Size,Medium
6  PreattentionExperiment,1,17,2,1,Size,Medium
7  PreattentionExperiment,1,21,2,2,Size,Low
8  PreattentionExperiment,1,26,2,3,Size,High
9  PreattentionExperiment,6,269,3,3,Color,Medium
10 PreattentionExperiment,6,270,3,3,Color,Medium
11
```

Two main files: `experiment.html` and `experiment.js`

We will modify `experiment.js` only



The screenshot shows a code editor with two files open: `experiment.html` and `experiment.js`. The `experiment.html` file is on the left, and the `experiment.js` file is on the right. The `experiment.html` file contains HTML code for a form with three select boxes for 'Participant', 'Block', and 'Trial', and two buttons for 'startExperiment' and 'downloadLogs'. The `experiment.js` file contains JavaScript code for a D3.js visualization, including a `shuffle` function, a `createScene` function, and a `setTrial` function. The `createScene` function is highlighted with an orange box in the `experiment.html` file, and the `createScene` function is also highlighted with an orange box in the `experiment.js` file. The status bar at the bottom indicates the current position is Line 297, Column 30, with 2 spaces, UTF-8 encoding, and LF line endings.

```
46 </style>
47
48 <script>
49   var ignore = function(event) {
50     event.preventDefault();
51   }
52   var ignoreSpace = function(event) {
53     if(event.code == "Space") event.preventDefault();
54   }
55 </script>
56
57 </head>
58
59 <body onload="createScene();" onkeydown="ignoreSpace(event);">
60 <div>
61   <form id="form">
62     <span class="formLb">Participant:</span>
63     <select class="select" id="participantSel" onchange="onchange">
64       <span class="formLb">Block:</span>
65       <select class="select" id="blockSel" onchange="onchange">
66       <span class="formLb">Trial:</span>
67       <select class="select" id="trialSel" onchange="onchange">
68       <button onclick="startExperiment(event)" onkeydown="ignoreSpace(event)">Start Experiment</button>
69       <button onclick="downloadLogs(event)" onkeydown="ignoreSpace(event)">Download Logs</button>
70     </div>
71   </div>
72 </body>
73
74
75
```

```
280 // shuffle the elements in the array
281 // copied from https://stackoverflow.com/questions/6274339/how-to-shuffle-an-array
282 function shuffle(array) {
283   var j, x, i;
284   for (i = array.length - 1; i > 0; i--) {
285     j = Math.floor(Math.random() * (i + 1));
286     x = array[i];
287     array[i] = array[j];
288     array[j] = x;
289   }
290   return array;
291 }
292
293 /*****
294
295 var createScene = function() {
296   var svgEl = d3.select("#sceneCanvas").append("svg");
297   svgEl.attr("width", ctx.w);
298   svgEl.attr("height", ctx.h);
299   .classed("centered", true);
300
301   loadData(svgEl);
302 };
303
304 /*****
305 /***** STARTING PARAMETERS *****/
306 /*****
307
308 var setTrial = function(trialID) {
309   ctx.startTrial = parseInt(trialID);
310 }
```

At page loading time, function `createScene` is called.  
`createScene` then calls `loadData`.

loadData needs to access the CSV design file output by TouchStone, which is a local file on your system

Use a (local) Web server to serve local files with HTTP

### Option 1

Launch an HTTP server in the TD's directory

```
> cd experiment_js/
```

```
> python -m http.server 8888 (python 3) or > python -m SimpleHTTPServer 8888 (python 2)
```

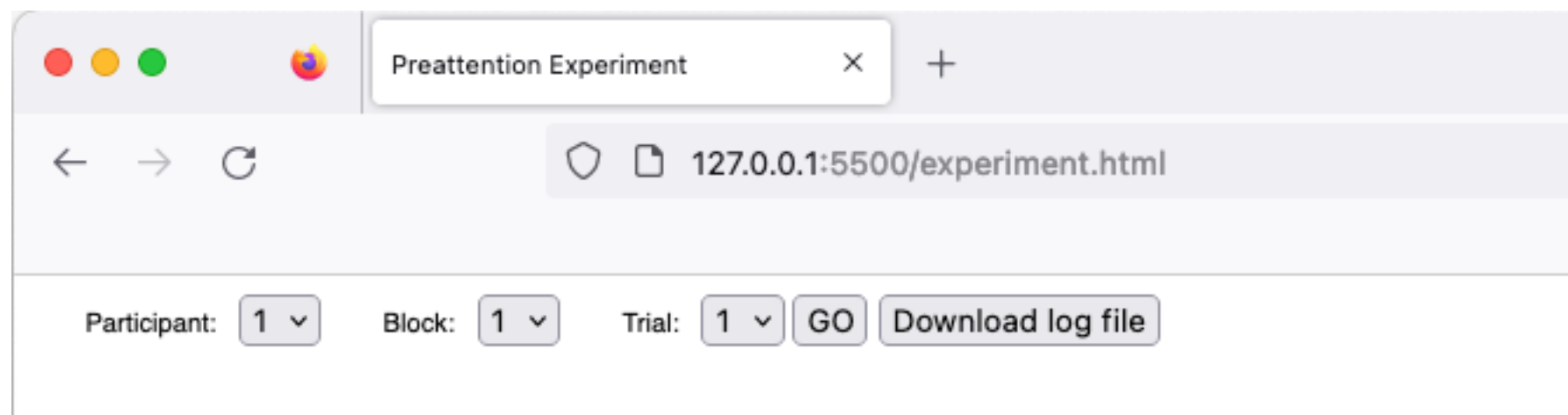
Access the page from your browser

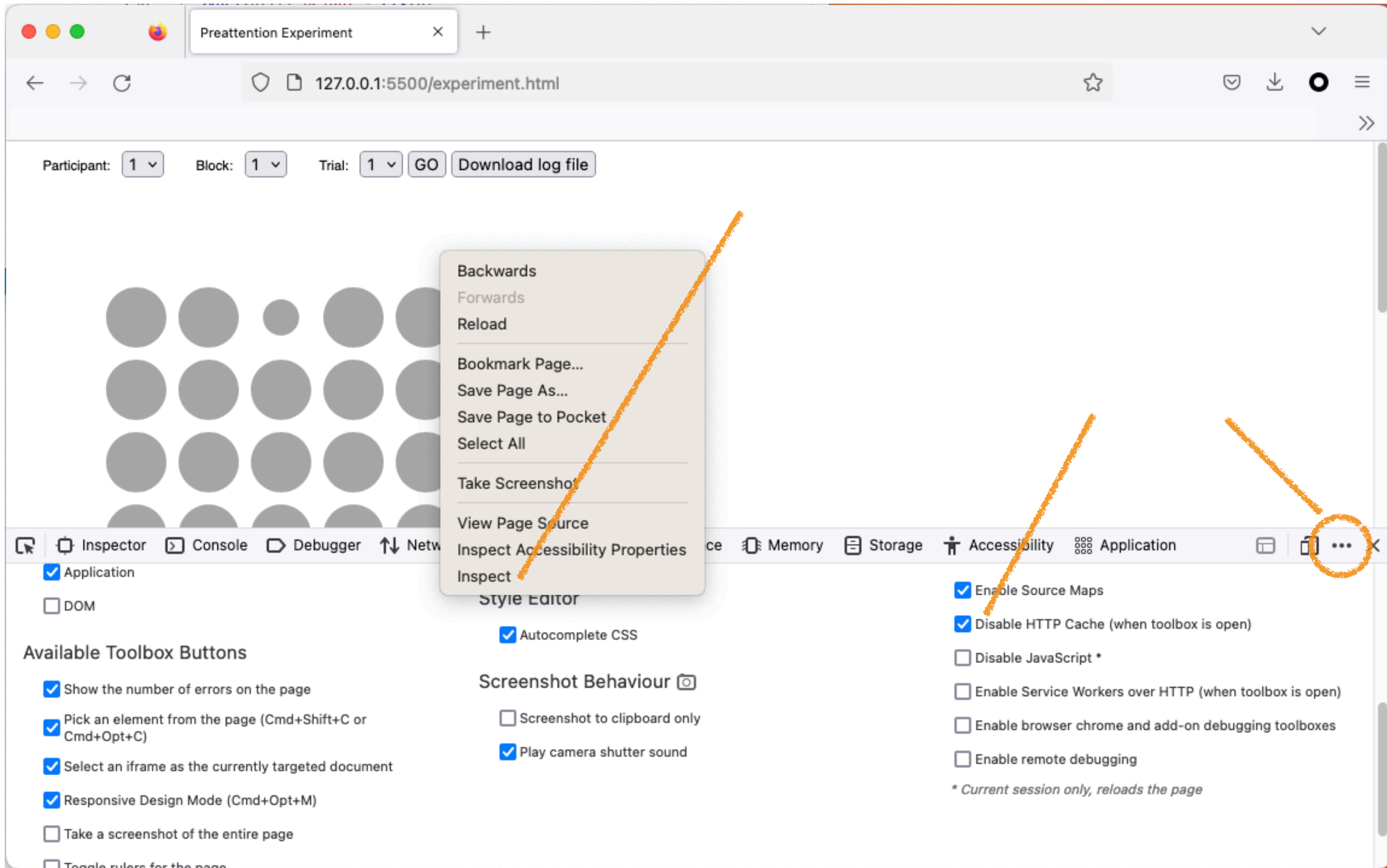
<http://localhost:8888/experiment.html>

### Option 2

Use a plugin for your code editor

e.g., Live Server for Visual Studio Code, or atom-live-server for Atom, etc.





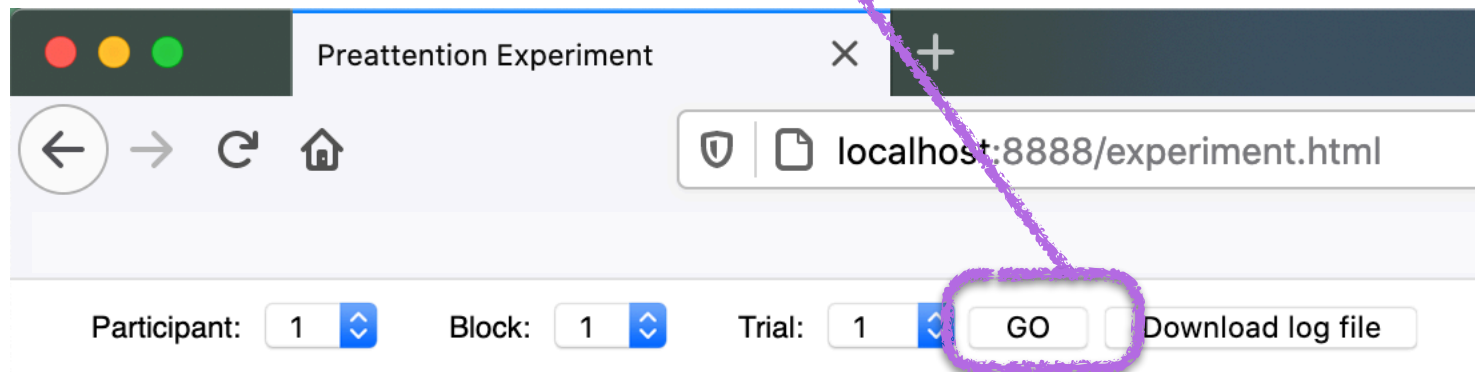


loadData function reads experiment\_touchstone2\_test.csv and turns it into an array of trials.

```
js > JS experiment.js > [loadData] > then() callback
34
35 /*****
36 /***** LOAD CSV DESIGN FILE *****/
37 /*****
38
39 var loadData = function(svgEl){
40 // d3.csv parses a csv file...
41 d3.csv("experiment_touchstone2_test.csv").then(function(data){
42 // ... and turns it into a 2-dimensional array where each line is an array indexed by the column header
43 // for example, data[2]["OC"] returns the value of OC in the 3rd line
44 ctx.trials = data;
45 // all trials for the whole experiment are stored in global variable ctx.trials
46
47 var participant = "";
48 var options = [];
49
50 for(var i = 0; i < ctx.trials.length; i++) {
51 if(!(ctx.trials[i][ctx.participantIndex] === participant)) {
52 participant = ctx.trials[i][ctx.participantIndex];
53 options.push(participant);
54 }
55 }
56
57 var select = d3.select("#participantSel")
58 select.selectAll("option")
59 .data(options)
60 .enter()
61 .append("option")
62 .text(function (d) { return d; });
63
64 setParticipant(options[0]);
65
66 }).catch(function(error){console.log(error)});
67
68
69
```

Ln 44, Col 23 Spaces: 2 UTF-8 LF {} JavaScript Port : 5500

# Clicking button GO calls function `startExperiment`



```
var startExperiment = function(event) {  
  ...  
  
  // start first trial  
  console.log("start experiment at "+ctx.cpt);  
  nextTrial();  
}
```

`ctx.cpt` is now the index just before the first trial to run in the trial table `ctx.trials`.

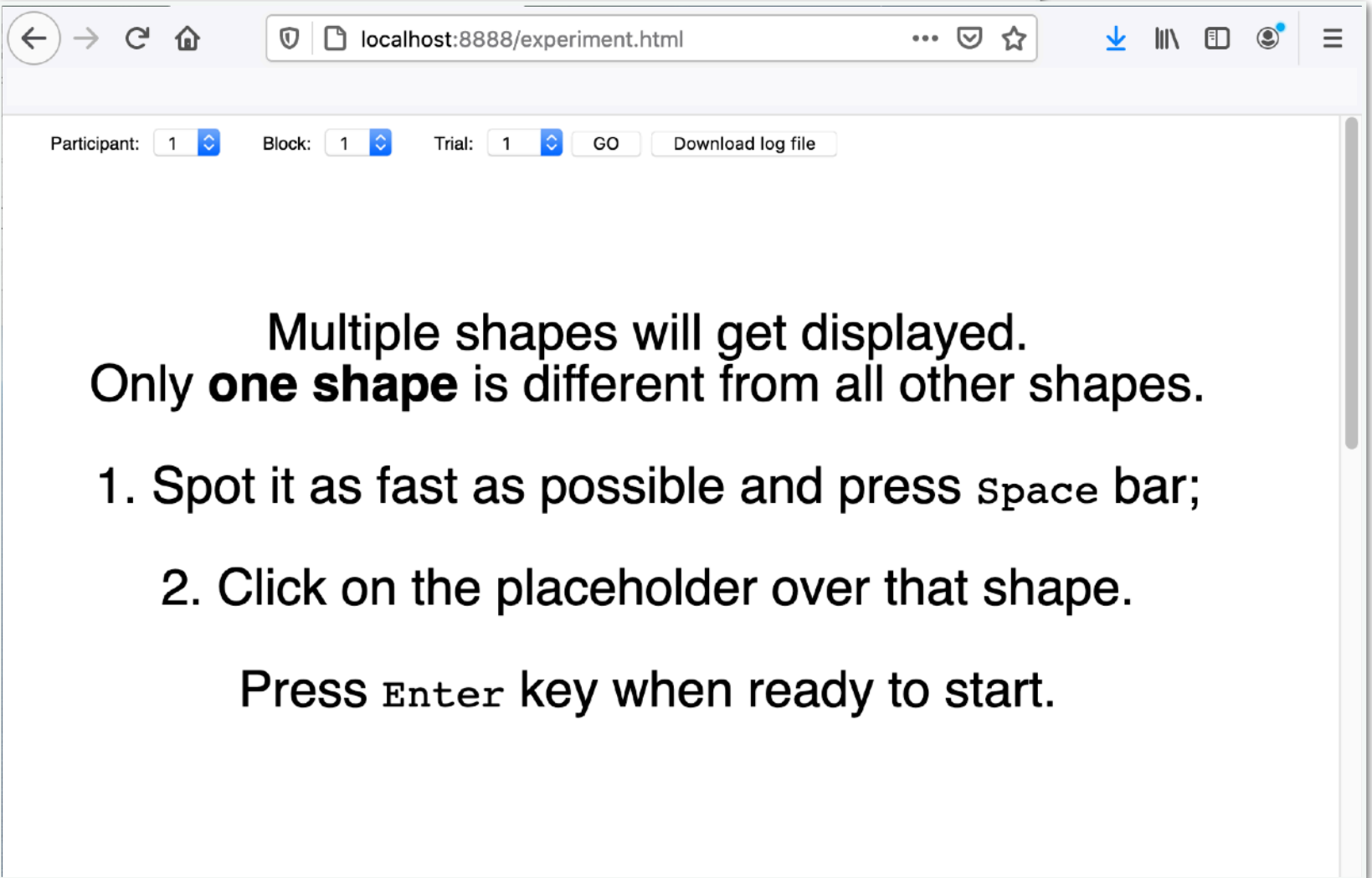
`nextTrial` function is called.



```
var nextTrial = function() {  
  ctx.cpt++;  
  displayInstructions();  
}
```

nextTrial calls displayInstructions

```
var displayInstructions = function() {  
  ctx.state = state.INSTRUCTIONS;  
  
  d3.select("#instructionsCanvas")  
    .append("div")  
    .attr("id", "instructions")  
    .classed("instr", true);  
  
  d3.select("#instructions")  
    .append("p")  
    .html("Multiple shapes will get displayed.<br>");  
  
  d3.select("#instructions")  
    .append("p")  
    .html("1. Spot it as fast as possible and press space bar;");  
  
  d3.select("#instructions")  
    .append("p")  
    .html("2. Click on the placeholder over that shape.");  
  
  d3.select("#instructions")  
    .append("p")  
    .html("Press <code>Enter</code> key when ready to start.");  
}
```



The screenshot shows a web browser window with the address bar displaying 'localhost:8888/experiment.html'. The page content includes a header with controls for 'Participant: 1', 'Block: 1', and 'Trial: 1', each with a dropdown arrow, followed by a 'GO' button and a 'Download log file' button. The main content area contains the following text:

Multiple shapes will get displayed.  
Only **one shape** is different from all other shapes.

1. Spot it as fast as possible and press space bar;
2. Click on the placeholder over that shape.

Press Enter key when ready to start.

```

var displayInstructions = function() {
  ctx.state = state.INSTRUCTIONS;

  d3.select("#instructionsCanvas")
    .append("div")
    .attr("id", "instructions")
    .classed("instr", true);

  d3.select("#instructions")
    .append("p")
    .html("Multiple shapes will get displayed.<br> Only <b>one

  d3.select("#instructions")
    .append("p")
    .html("1. Spot it as fast as possible and press <code>Space

  d3.select("#instructions")
    .append("p")
    .html("2. Click on the placeho

  d3.select("#instructions")
    .append("p")
    .html("Press <code>Enter</code>
}

```

experiment.js

We use library d3 to make DOM selections and manipulations easy. For example,

```

d3.select("#instructionsCanvas")
  .append("div")
  .attr("id", "instructions")
  .classed("instr", true);

```

selects element whose id is **instructionsCanvas**, in the HTML document, adds a new **div** child to this element, sets id of this new **div** to **instructions** and adds CSS class **instr** to it.

```

<body onload="createScene();" onkeydown="ignoreSpace(event);" onkeyup="keyListener(event);">
  <div>
    <form id="form">

      <span class="formLb">Participant:</span>
      <select class="select" id="participantSel" onchange="onChangeParticipant()"></select>

      <span class="formLb">Block:</span>
      <select class="select" id="blockSel" onchange="onChangeBlock()"></select>

      <span class="formLb">Trial:</span>
      <select class="select" id="trialSel" onchange="onChangeTrial()"></select>

      <button onclick="startExperiment(event)" onkeydown="ignore(event);" onkeyup="ignore(event);">GO</button>

      <button onclick="downloadLogs(event)" onkeydown="ignore(event)" onkeyup="ignore(event)">Download log file</button>

    </form>
  </div>
  <div id="instructionsCanvas">
  </div>
  <div id="sceneCanvas">
  </div>
</body>

```

experiment.html

# Keyboard events

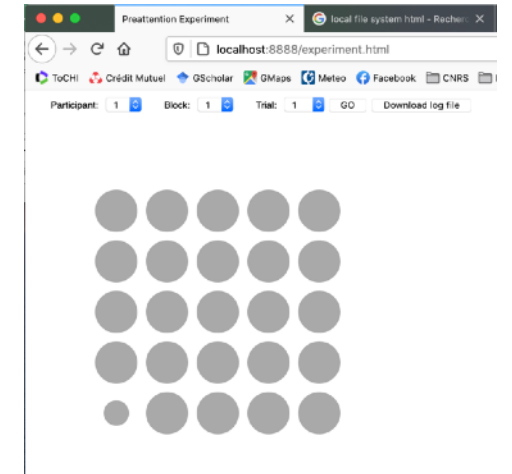
Function `keyListener` gets called when a key is pressed

```
var keyListener = function(event) {  
  event.preventDefault();  
  
  if(ctx.state == state.INSTRUCTIONS && event.code == "Enter") {  
    d3.select("#instructions").remove();  
    displayShapes();  
  }  
}
```



# Display grid of shapes

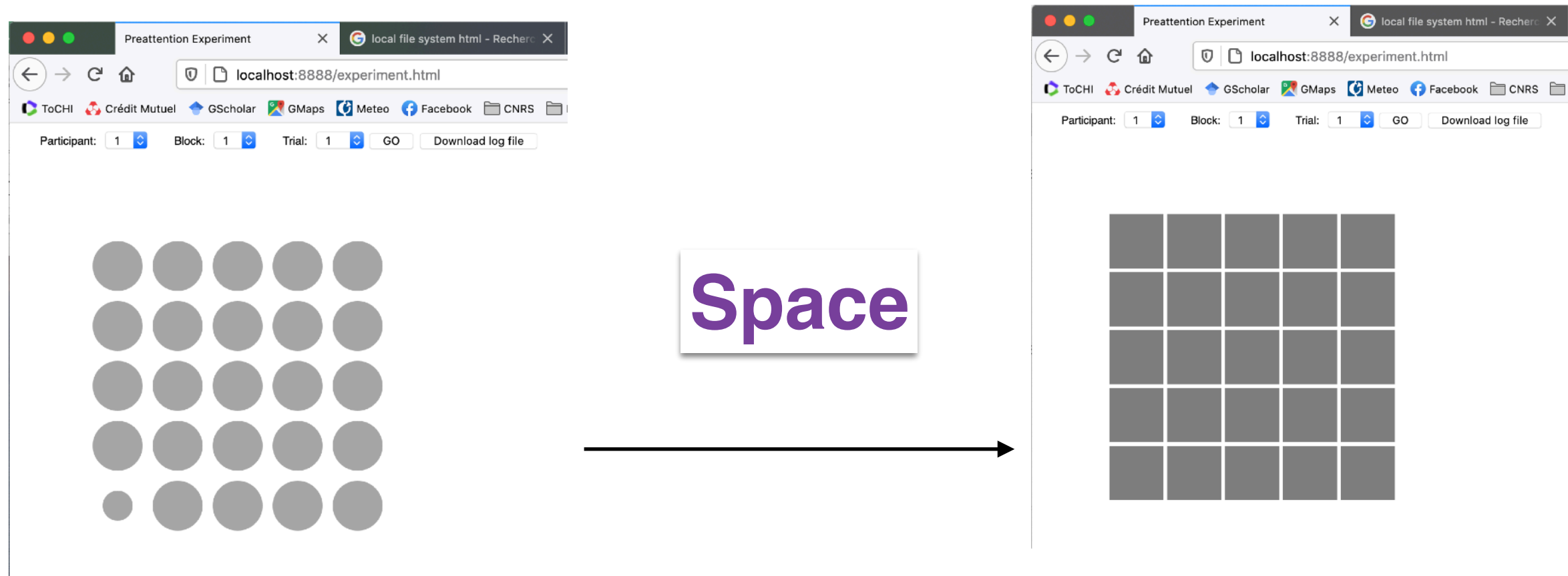
Function `displayShapes` does the job



```
var displayShapes = function() {  
  ctx.state = state.SHAPES;  
  
  var differenceType = ctx.trials[ctx.cpt]["DT"];  
  var objectCount = ctx.trials[ctx.cpt]["OC"];  
  if(objectCount === "Low") {  
    objectCount = 9;  
  } else if(objectCount === "Medium") {  
    objectCount = 25;  
  } else {  
    objectCount = 49;  
  }  
  console.log("display shapes for condition "+objectCount+", "+differenceType);  
  
  var svgElement = d3.select("svg");  
  var group = svgElement.append("g")  
    .attr("id", "shapes")  
    .attr("transform", "translate(100,100)");
```

# TODO step 1-a

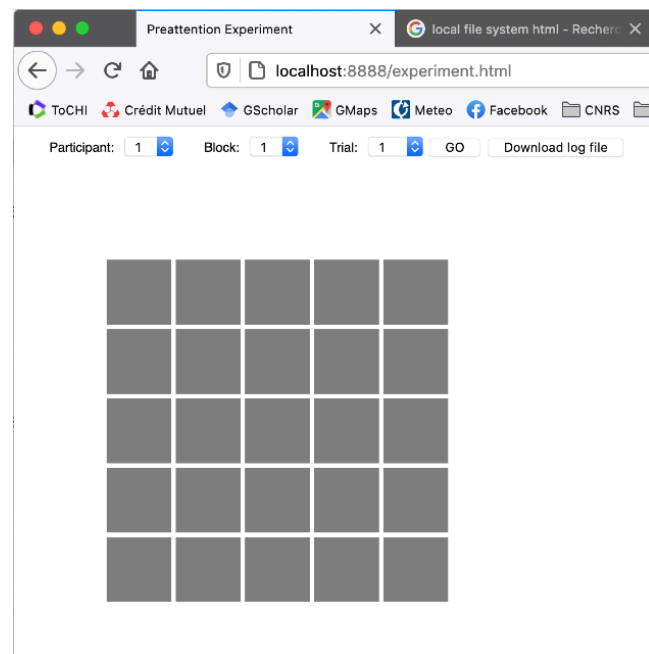
Update function `keyListener` to remove shapes and display placeholders instead when participant presses `Space` bar in state `state.SHAPES`.



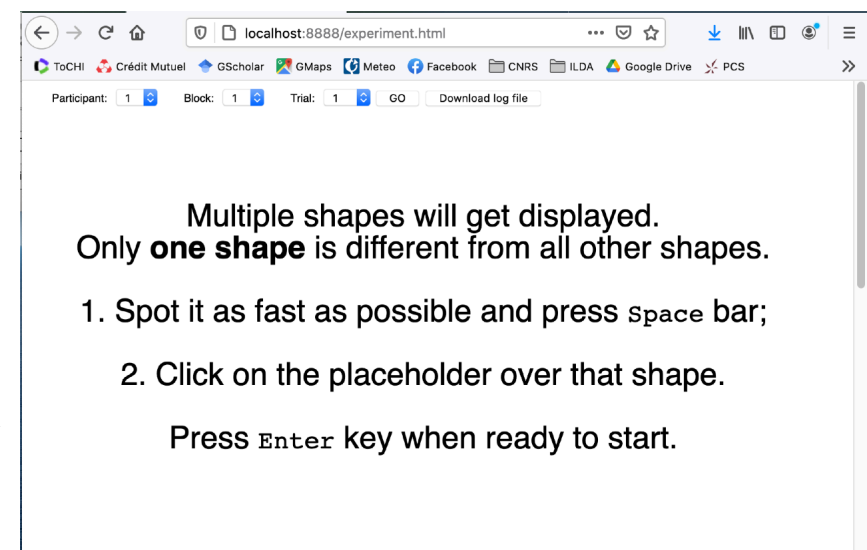
Function `displayPlaceholders` is provided

# TODO step 1-b

Update function `displayPlaceholders` to remove placeholders and progress to next trial (call `nextTrial`) when participant clicks a placeholder. **Ignore errors for now, progress to next trial in all cases.**



Click



```
var displayPlaceholders = function() {  
  ...  
  placeholder.on("click",  
    function() {  
      // TODO  
    }  
  );  
}
```

# Scene of objects

Let's take a closer look at function `displayShapes`

```
var displayShapes = function() {  
  ...  
  
  // 1. Decide on the visual appearance of the target  
  // In my example, it means deciding on its size (large or small) and its color (light or dark)  
  var randomNumber1 = Math.random();  
  var randomNumber2 = Math.random();  
  var targetSize, targetColor;  
  if(randomNumber1 > 0.5) {  
    targetSize = 25; // target is large  
  } else {  
    targetSize = 15; // target is small  
  }  
  if(randomNumber2 > 0.5) {  
    targetColor = "DarkGray"; // target is dark gray  
  } else {  
    targetColor = "LightGray"; // target is light gray  
  }  
}
```

We decide on the target's appearance





























# Scene of objects

Let's take a closer look at function `displayShapes`

```
var displayShapes = function() {  
  ...  
  
  // 2. Set the visual appearance of all other objects now that the target appearance is  
  // decided  
  // Here, we implement the case DT = "Size" so all other objects are large (resp. small)  
  if target is small (resp. large) but have the same color as target.  
  var objectsAppearance = [];  
  for (var i = 0; i < objectCount-1; i++) {  
    if(targetSize == 25) {  
      objectsAppearance.push({  
        size: 15,  
        color: targetColor  
      });  
    } else {  
      objectsAppearance.push({  
        size: 25,  
        color: targetColor  
      });  
    }  
  }  
}
```

Target	Other objects					
						...
						...
						...
						...

We generate the list of other objects depending on the target's appearance



# Scene of objects

Let's take a closer look at function `displayShapes`

```
var displayShapes = function() {  
  ...  
  
  // 3. Shuffle the list of objects (useful when there are variations regarding both visual  
variable) and add the target at a specific index  
  shuffle(objectsAppearance);  
  // draw a random index for the target  
  ctx.targetIndex = Math.floor(Math.random()*objectCount);  
  // and insert it at this specific index  
  objectsAppearance.splice(ctx.targetIndex, 0, {size:targetSize, color:targetColor});  
}
```

We shuffle (\*) the list of other objects and then insert the target at a specific index

(\*) explanation for shuffling later on

# Scene of objects

Let's take a closer look at function `displayShapes`

```
var displayShapes = function() {  
  ...  
  
  // 4. We create actual SVG shapes and lay them out as a grid  
  // compute coordinates for laying out objects as a grid  
  var gridCoords = gridCoordinates(objectCount, 60);  
  // display all objects by adding actual SVG shapes  
  for (var i = 0; i < objectCount; i++) {  
    group.append("circle")  
      .attr("cx", gridCoords[i].x)  
      .attr("cy", gridCoords[i].y)  
      .attr("r", objectsAppearance[i].size)  
      .attr("fill", objectsAppearance[i].color);  
  }  
}
```

We actually display shapes as a SVG shapes laid out as a grid.

We use d3 library to manipulate the DOM structure (add elements and set their attributes' values)

# TODO step 2-a

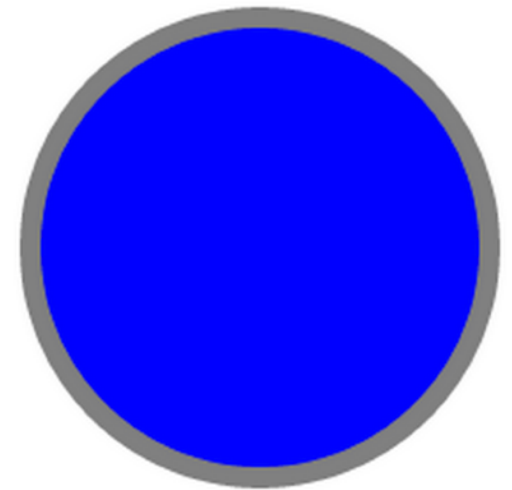
For now, function **displayShapes** ignores the actual value of DT and simply implements the case  $DT = \text{"Size"}$

Adapt the code to your visual variable  $VV_1$   
(i.e., handle your own case  $DT = VV_1$ )

# SVG and visual variables

I used circles with Size and Color visual variables, but SVG provides you with different types of graphical shape and various graphical attributes

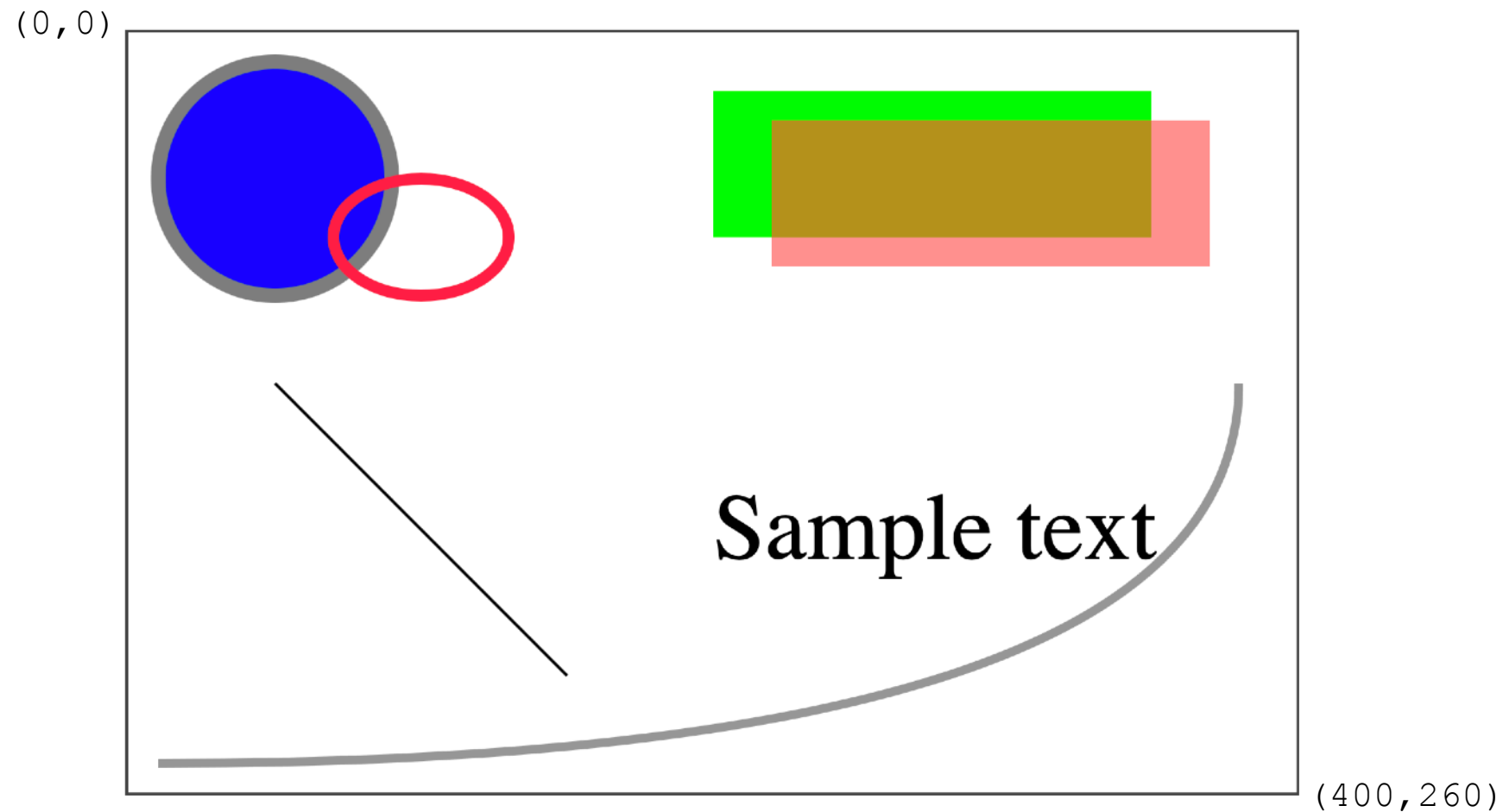
```
<svg width="100" height="100">  
  <circle cx="50" cy="50" r="22" fill="blue" stroke="gray" stroke-width="4"/>  
</svg>
```



- SVG code can be included directly in HTML documents
- Shapes: `rect`, `circle`, `ellipse`, `line`, `text`, `path`
- Styling: `fill`, `stroke`, `stroke-width`, `opacity`, `font-family`, `font-size`
- or use CSS rules
- Transparency can be controlled with `opacity` or `rgba(r,g,b,a)` color tuples

# SVG - Scalable Vector Graphics

source: Emmanuel Pietriga

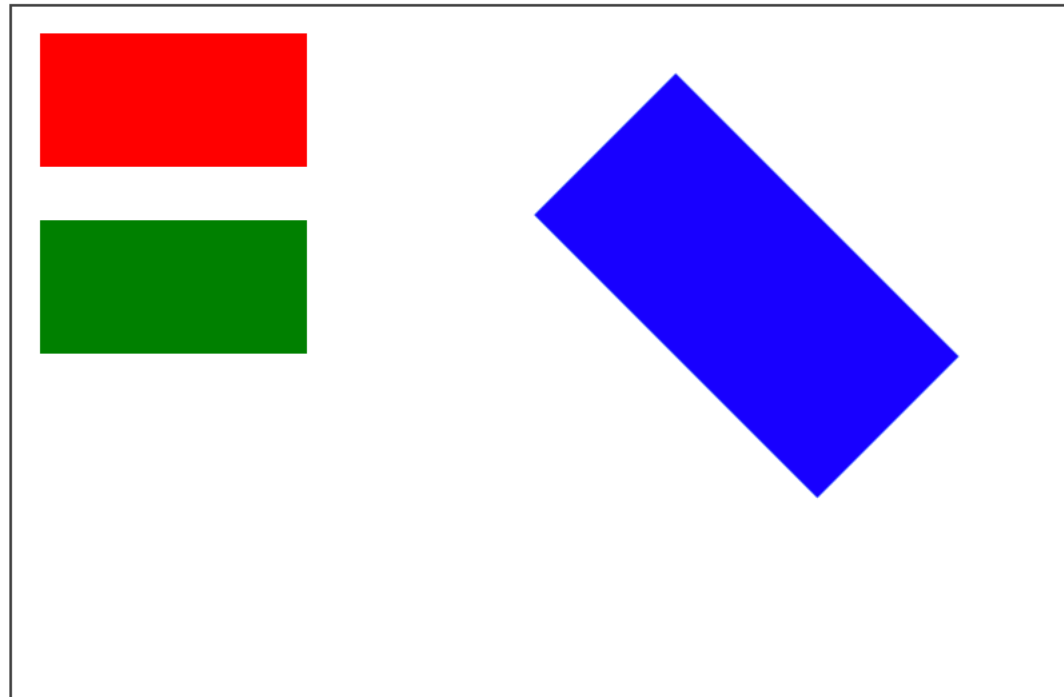


```
<svg width="400" height="260">
  <!-- blue circle with a 5px-gray border-->
  <circle cx="50" cy="50" r="40" fill="blue" stroke="gray" stroke-width="5"/>
  <!-- ellipse with a 4px redish border and no fill color-->
  <ellipse cx="100" cy="70" rx="30" ry="20" fill="none" stroke="#FF2244" stroke-width="4"/>
  <!-- two rectangles partially overlapping, the one above (which us red) is semi-transparent-->
  <rect x="200" y="20" width="150" height="50" fill="#0F0"/>
  <rect x="220" y="30" width="150" height="50" fill="#F00" opacity=".5"/>
  <!-- simple black line -->
  <line x1="50" y1="120" x2="150" y2="220" stroke="black"/>
  <!-- simple text element -->
  <text x="200" y="180">Sample text</text>
  <!-- a quadratic bézier curve -->
  <path fill="none" stroke="#999" stroke-width="3" d="M10,250 Q380,250 380,120" />
</svg>
```

# SVG - Scalable Vector Graphics

*source: Emmanuel Pietriga*

## Affine Transforms

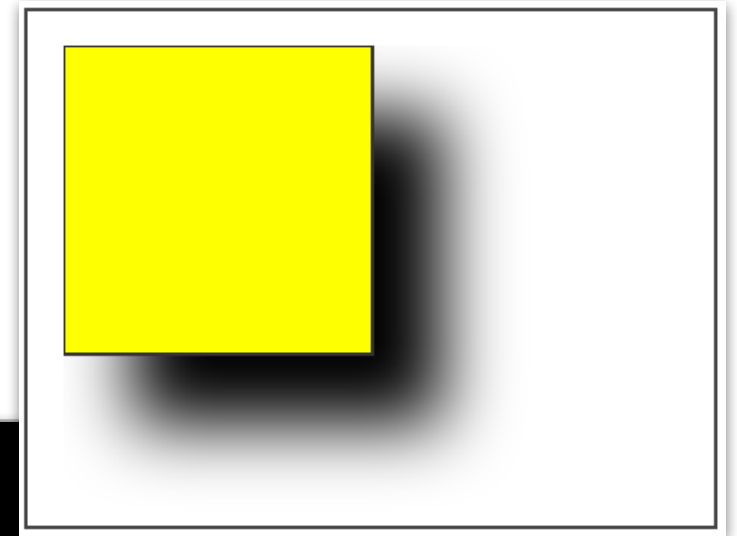


```
<rect x="10" y="10" width="100" height="50" style="fill:red"/>  
<rect x="0" y="0" width="100" height="50" style="fill:green"  
      transform="translate(10,80)" />  
<rect x="0" y="0" width="100" height="50" style="fill:blue"  
      transform="translate(10,150) scale(1.5) rotate(45 180 150)"/>
```

# SVG - Scalable Vector Graphics

*source: Emmanuel Pietriga*

Many more possibilities, including, *e.g.*, filters:



```
<svg width="200" height="150">
  <defs>
    <filter id="ds" x="0" y="0" width="200%" height="200%">
      <feOffset result="off0ut" in="SourceAlpha" dx="20" dy="20" />
      <feGaussianBlur result="blur0ut" in="off0ut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blur0ut" mode="normal" />
    </filter>
  </defs>
  <rect x="10" y="10" width="90" height="90" fill="yellow" stroke="#333"
    filter="url(#ds)" />
</svg>
```


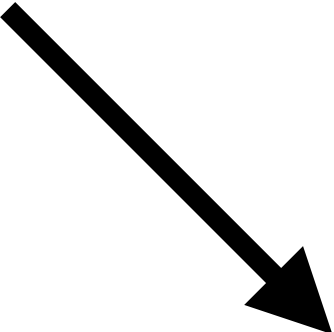
Detailed SVG documentation:

<https://developer.mozilla.org/en-US/docs/Web/SVG/Element>

# Manipulating the DOM of a web page with d3

```
var svgElement = d3.select("svg");  
var group = svgElement.append("g")  
  .attr("id", "shapes")  
  .attr("transform", "translate(100,100)");
```

```
group.append("circle")  
  .attr("cx", 50)  
  .attr("cy", 50)  
  .attr("r", 20)  
  .attr("fill", "red");
```



```
<svg>  
  <g id="shapes" transform="translate(100,100)">  
    <circle cx=50 cy=50 r=20 fill="red" />  
  </g>  
  ...  
</svg>
```



# TODO step 2-b

Complement the code to make it work for your second visual variable (i.e., handle case  $DT = VV_2$ )

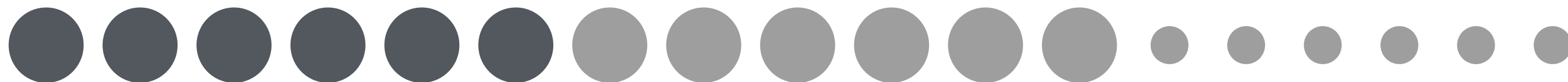
# TODO step 2-c

Complement the code to make it work for the combination of your two visual variables (i.e., handle case  $DT = VV_1VV_2$ )

This is where shuffling other objects in **displayShapes** makes sense

*example for  $DT = \text{"ColorSize"}$  with a target* ●

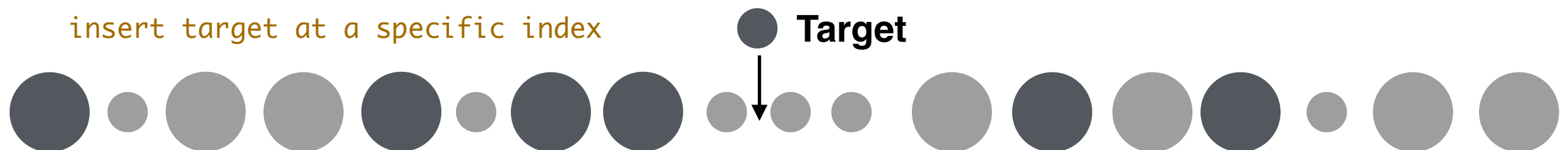
**Other objects (generated by series of objects that have the three possible apperances)**



Shuffle other objects



insert target at a specific index



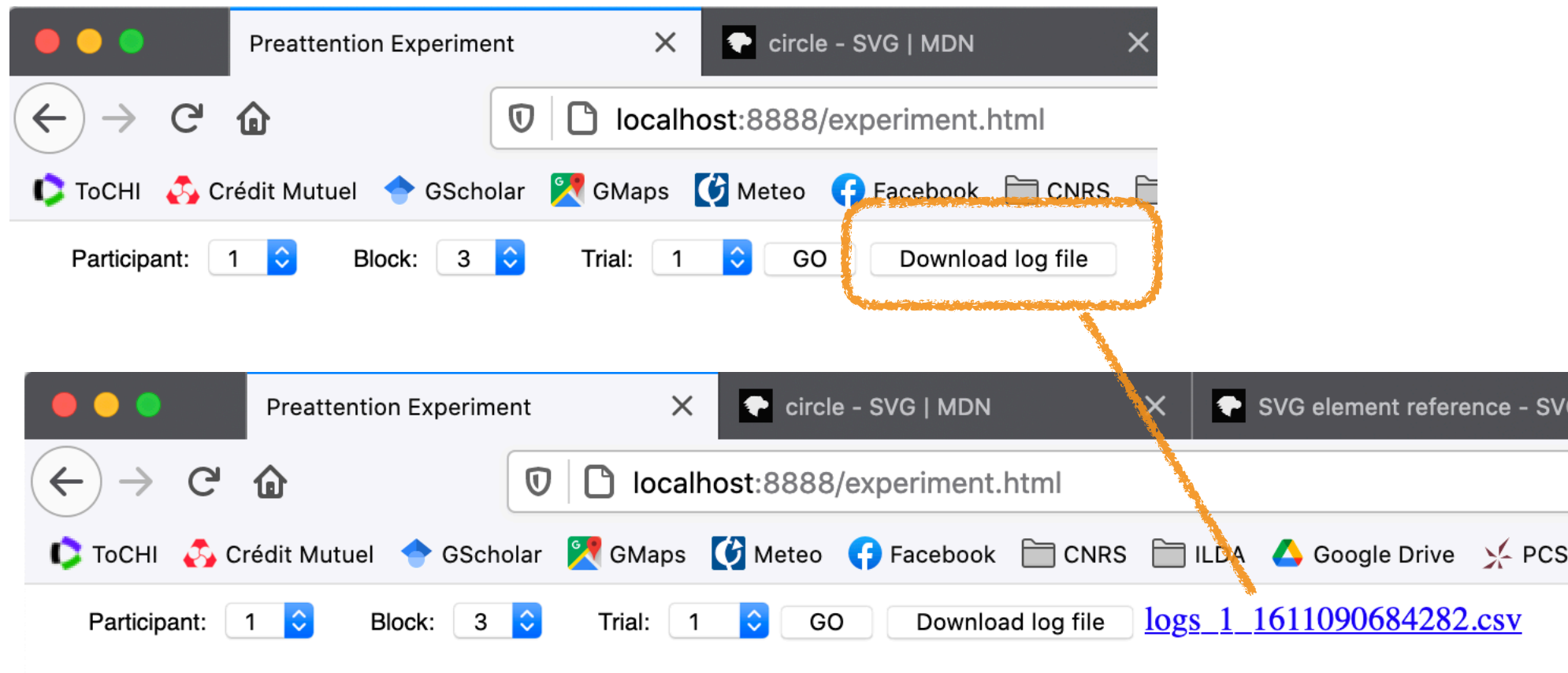
# TODO step 3

Log measures by adding an array with values each time a trial ends at the end of the `ctx.loggedTrials` array which contains one line for each trial that has been run until now.

```
ctx.loggedTrials = [  
  ["DesignName", "ParticipantID", "TrialID", "Block1", "Block2", "DT", "OC", "visualSearchTime", "ErrorCount"]];  
  ...  
ctx.loggedTrials.push(  
  ["Preattention-experiment", 1, 1, 1, 1, "Size", "Medium", 1582, 0]  
)
```

# TODO step 3

Button download log file calls function `downloadLogs` which turns `ctx.loggedTrials` into a csv that you can download




# TODO step 3-a

a) Log measure visualSearchTime, the function `Date.now()` can be useful for handling the timer. It returns the current time in ms.

# TODO step 3-b

b) Log measure ErrorCount: In case of error (wrong element clicked), just count an error but do not log anything. Restart a trial in the same condition (restart the timer as well...). We want to have one correct completion time measure for each experimental condition.

```
DesignName,ParticipantID,TrialID,Block1,Block2,DT,OC,visualSearchTime,ErrorCount  
...  
PreattentionExperiment,1,2,1,1,Color,Medium,1582,2  
...
```



The successful trial in this condition took 1582 ms. It was preceded by two incorrect selections.

# TODO step 4

Make sure that your program stops when all trials for this participant are completed (i.e., when `ctx.trials[ctx.cpt]["ParticipantID"]` is no longer the same value)