# Experimental design and analysis 

Experiment programming
https://www.|Ir.fr/~appert/eval/

Caroline Appert

## Project: hypotheses to test

Pick two visual variables of your choice (e.g., color, size, shape, shadow, etc.). Let's call them $\mathrm{VV}_{1}$ and $\mathrm{VV}_{2}$.

Research hypotheses:
$H_{1}: V V_{1}$ is preattentive
$H_{2}: V V_{2}$ is preattentive
$H_{3}: V V 1$ and $V V 2$ combined are less preattentive than $V V 1$ or VV2 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 $\mathrm{H}_{1}: \mathrm{VV}_{1}$ is preattentive

Visual search Eime

(*) 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 $\mathrm{H}_{3}: \mathrm{VV}_{1}$ and $\mathrm{V} \mathrm{V}_{2}$ combined are less preattentive than $\mathrm{VV}_{1}$ or $\mathrm{VV}_{2}$ 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)

Visual search Eime


Operationalization

Visual search kime
Measure (which we had already idenkified for the first kwo hypotheses)


## Operationalization

Factors:
oc: object count
\{Low, Medium, Large\}
VV: Difference in Visual Variables $\left\{V_{1}, V_{2}, V_{1} V^{2}\right\}$

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


## 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



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.


## 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



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 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 Deaian IClidan and Vidano uand in alidon
- Touchstone 2: online tool, and introduction
- Class 4: Experiment Pregrarnming
- Project start co de: Download
$\langle>$ js
experiment_js
experiment_js.zip$\hat{v}$
$\checkmark$
>>


## Project

IMPORTANT: Register your group in this spreadsh

## Getting started

> experiment_touchstone 2 test.csv is just an excerpt from a TouchStone 2 design file. You have to replace it with your own design.


Two main files: experiment.html and experiment.js

## We will modify experiment.js only



## 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.



## 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.<b
    d3.select("\#instructions")
        . append ("p")
        .html("1. Spot it as fast as possible and pr
    d3.select("\#instructions")
        . append ("p")
        .html("2. Click on the placeholder over that
    d3.select("\#instructions")
        . append("p")
        .html("Press <code>Enter</code> key when rea
    

```
    Participant:
    | Block: 1
        0 Trial: 
        60
    Download log file
```

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);
        L
    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>Spac
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.
    d3.select("#instructions")
        .append("p")
        .html("2. Click on the placeho
    d3.select("#instructions")
        .append("p")
        .html("Press <code>Enter</code
        <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();
    }

We use library d3 to make DOM manipulations easy. For example, d3.select("\#instructions").remove(); removes element whose id is instructions

\section*{Display grid of shapes}

\section*{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"];
    1f(objectcount ==0 Lown \(\{\)
        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)");

```

We will add all shapes to a group whose id is shapes so that we can remove all shapes later on with the single following line of code:
d3.select("\#shapes").remove();

\section*{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

\section*{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．
```

\& C)

```

```

OT,

```
```

OT,

```


```

    Paticipant:1目 Block: 1囵Tral: 1 0o Download log flic
    ```
```

    Paticipant:1目 Block: 1囵Tral: 1 0o Download log flic
    ```


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

```

\section*{Scene of objects}

\section*{Let's take a closer look at function displayShapes}
```

var displayShapes = function() {
// 1. Decide on the visual appearance of the target
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
}

```
// In my example, it means deciding on its size (large or small) and its color (light or dark)

We decide on the target's appearance

In order to avoid participants look for a specific object as opposed to look for the different object (i.e., threat to internal validity), we introduce some variation on the target appearance by setting it randomly to one of the four possible object appearances:


\section*{Scene of objects}

\section*{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
});
}
}

```


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

\section*{Scene of objects}

\section*{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

\section*{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)

\section*{TODO step 2-a}

For now, function displayShapes ignores the actual value of DT and simply implements the case DT = "Size"

Adapt the code to your visual variable \(\mathrm{V} \mathrm{V}_{1}\)
(i.e., handle your own case \(\mathrm{DT}=\mathrm{V} \mathrm{V}_{1}\) )

\section*{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

\section*{SVG - Scalable Vector Graphics}

```

<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>
```

\section*{SVG - Scalable Vector Graphics}

\section*{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)"/>

```

\section*{SVG - Scalable Vector Graphics}

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="offOut" in="SourceAlpha" dx="20" dy="20" />
            <feGaussianBlur result="blurOut" in="offOut" stdDeviation="10" />
            <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
            </filter>
    </defs>
    <rect x="10" y="10" width="90" height="90" fill="yellow" stroke="#333"
            filter="url(#ds)" />
</Svg>
```

Detailed SVG documentation:

\section*{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)");
Select first svg element and
insert a g element as a child
whose attr id is shapes
group.append("circle")
.attr("cx", 50)
.attr("cy", 50)
.attr("r", 20)
.attr("fill", "red");

```
<Svg>
```

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

\section*{TODO step 2-b}

Complement the code to make it work for your second visual variable (i.e., handle case \(\mathrm{VV}=\mathrm{VV}_{2}\) )

\section*{TODO step 2-c}

Complement the code to make it work for the combination of your two visual variables (i.e., handle case \(\mathrm{VV}=\mathrm{V} \mathrm{V}_{1} \mathrm{~V} \mathrm{~V}_{2}\) )

This is where shuffling other objects in displayShapes makes sense
example for VV="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

\section*{TODO step 3}

\section*{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","VV","OC","visualSearchTime","ErrorCount"]];
ctx.loggedTrials.push(
    [ "Preattention-experiment",1,1,1,1,"Size","Medium",1582,0]
)
```

\section*{TODO step 3}

\section*{Button download log file calls function downloadLogs which turns ctx.loggedTrials into a csv that you can download}


\section*{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.

\section*{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.
```

\section*{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)
