Hands-on tutorial for high-throughput web-based human psychophysics

Visual Object Recognition

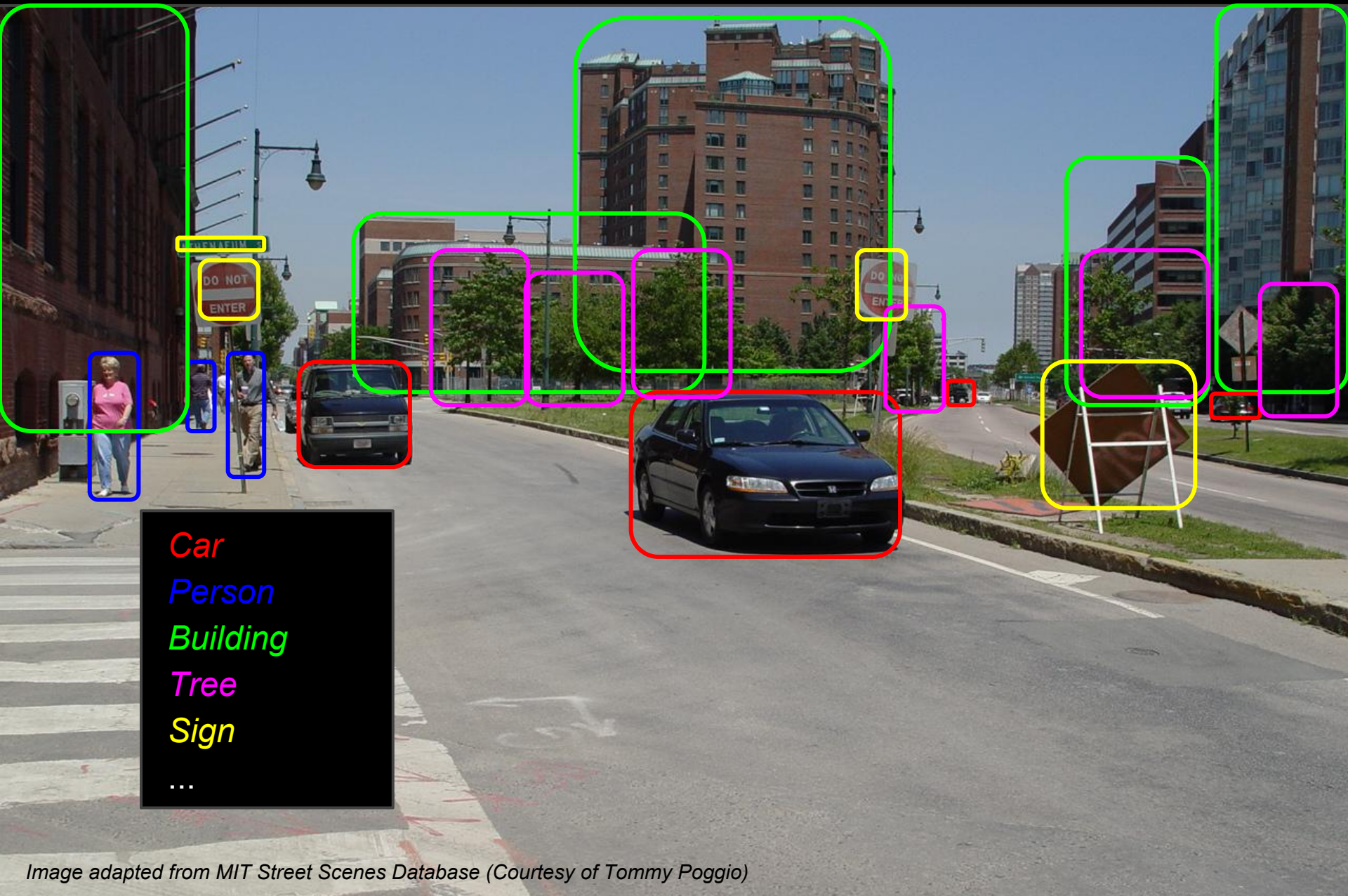
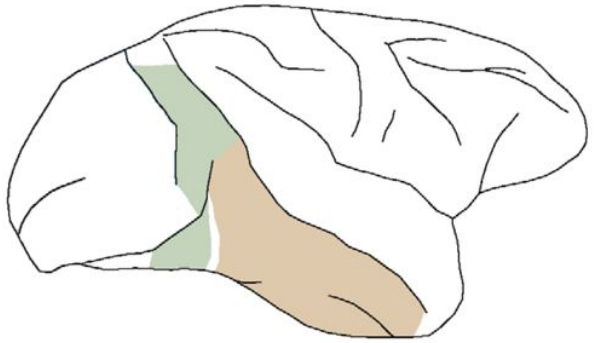


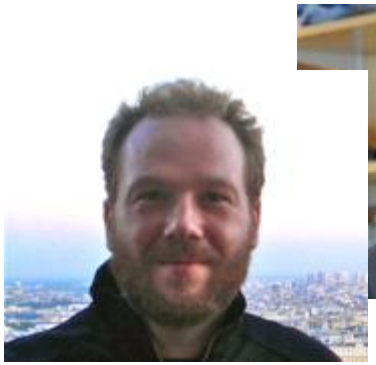
Image adapted from MIT Street Scenes Database (Courtesy of Tommy Poggio)

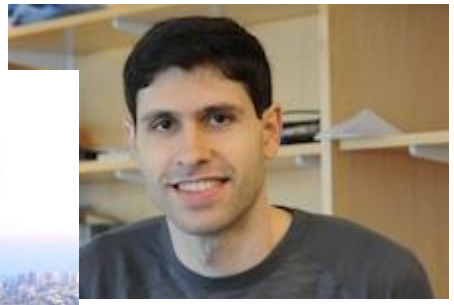
Different ways to study VOR

$S \rightarrow N \rightarrow \boldsymbol{\Psi}$



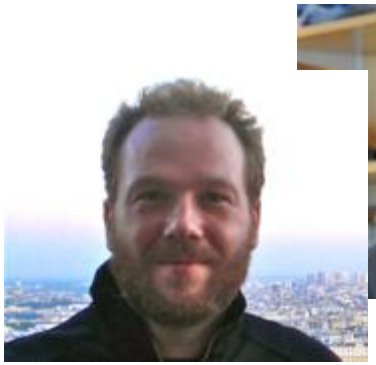
Different ways to study VOR

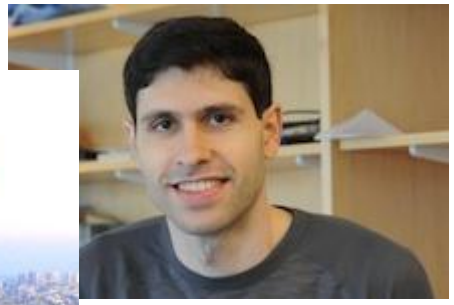




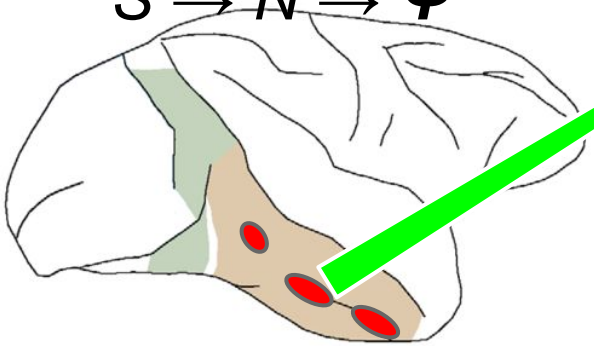
$S \rightarrow N \rightarrow \Psi$

Different ways to study VOR

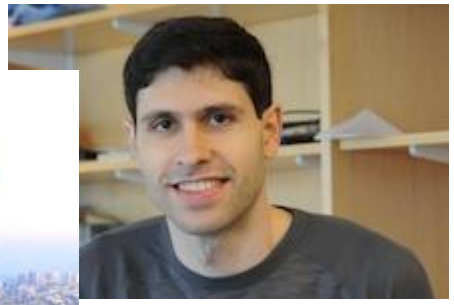




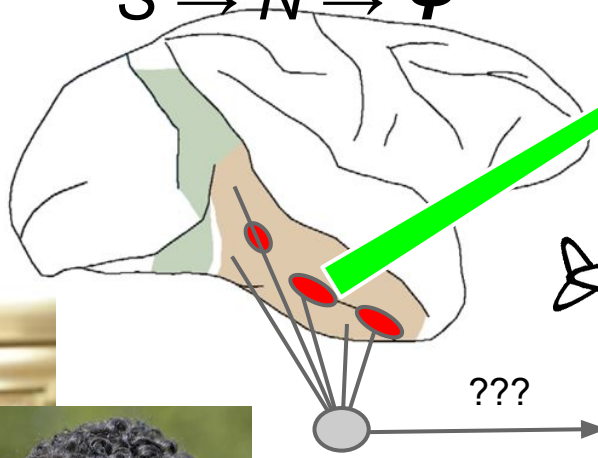
$S \rightarrow N \rightarrow \Psi$



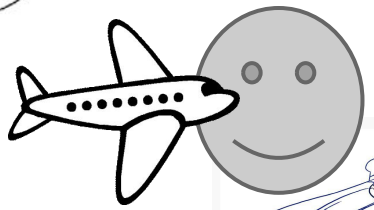
Different ways to study VOR

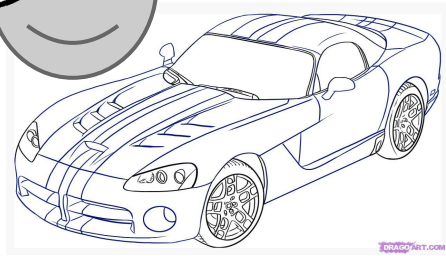


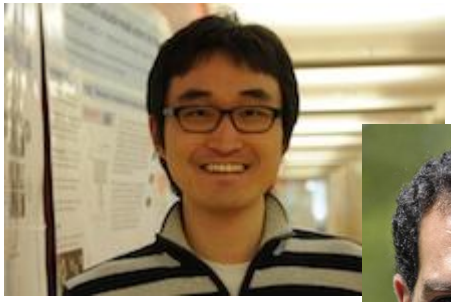
$S \rightarrow N \rightarrow \Psi$



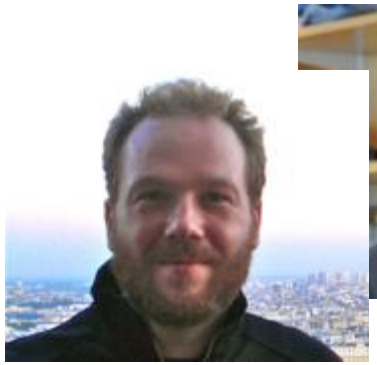
???

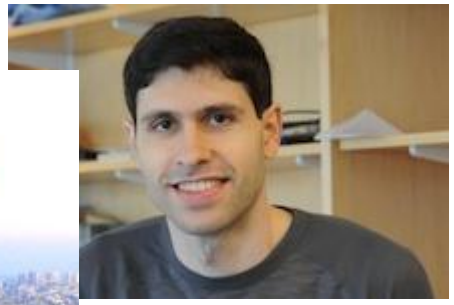




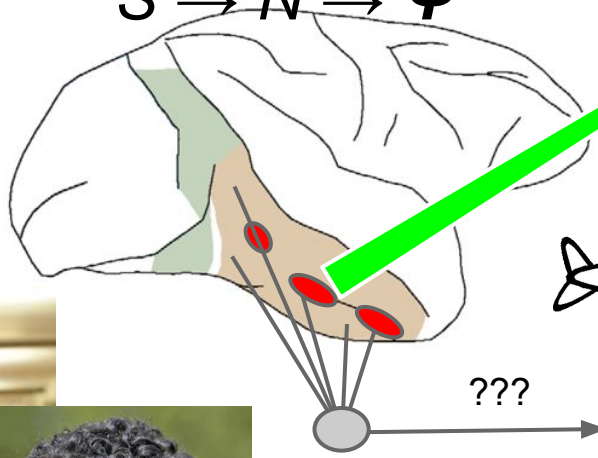


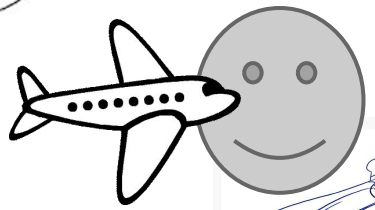
Different ways to study VOR

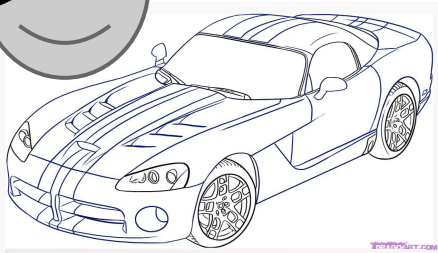


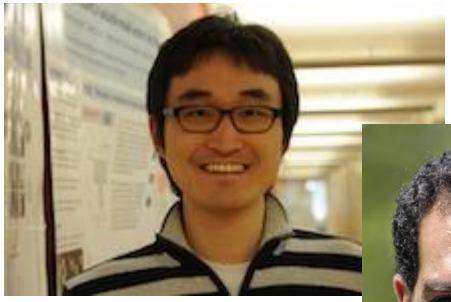


$S \rightarrow N \rightarrow \Psi$

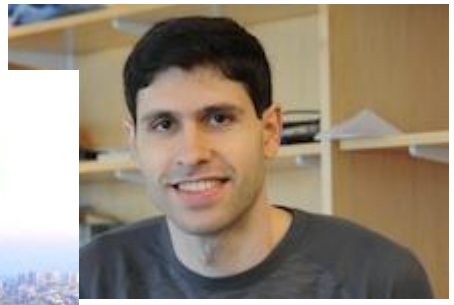


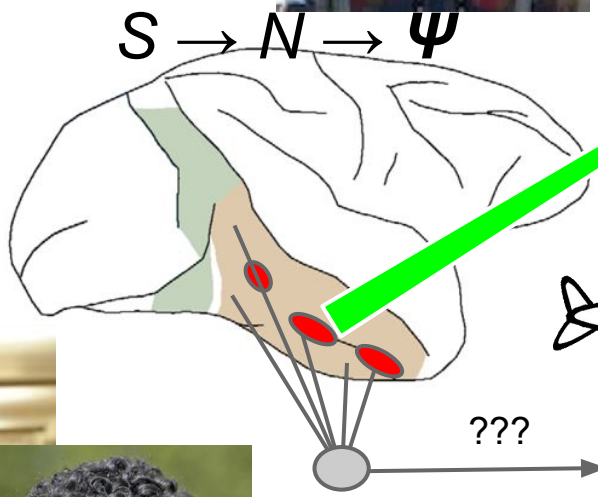


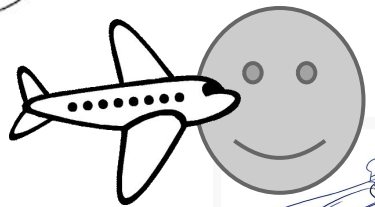
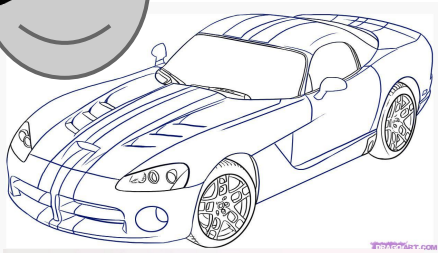
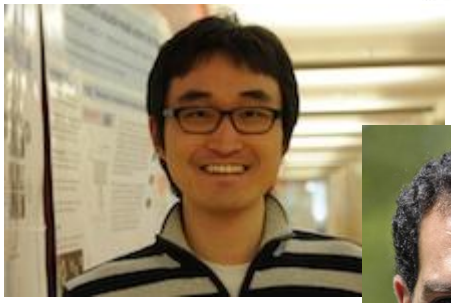




Different ways to study VOR

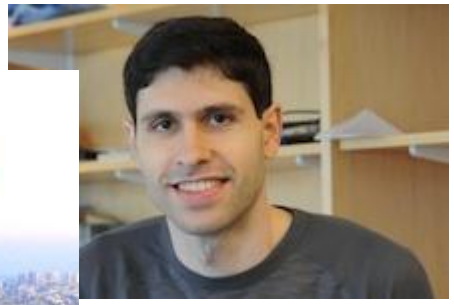



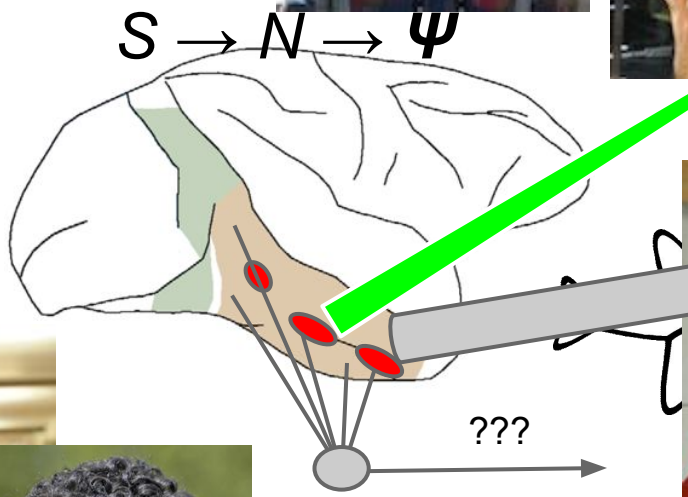



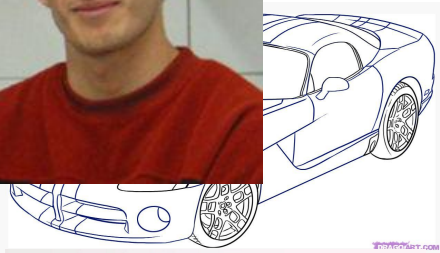




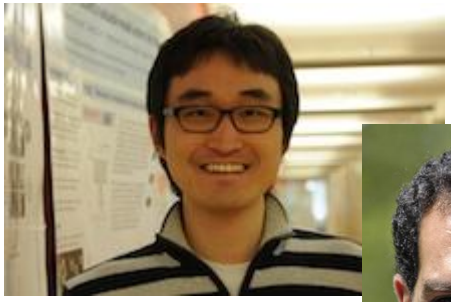


Different ways to study VOR



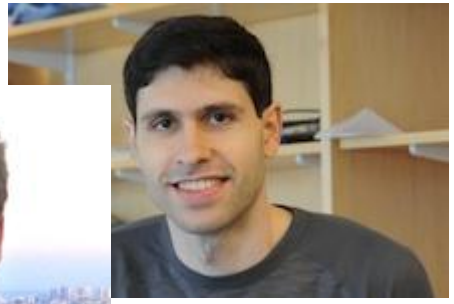




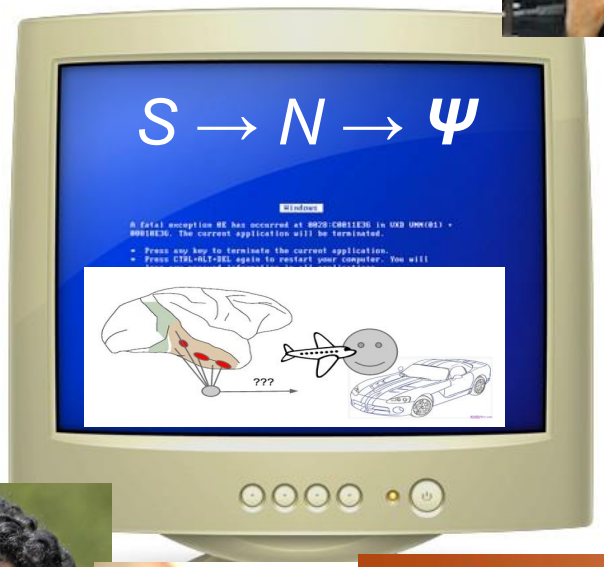


Different ways to study VOR

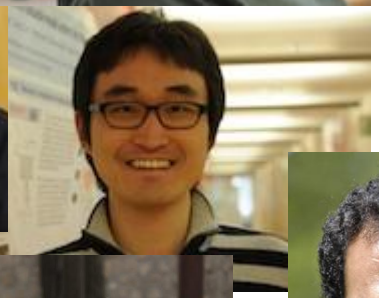


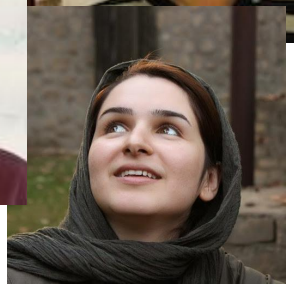






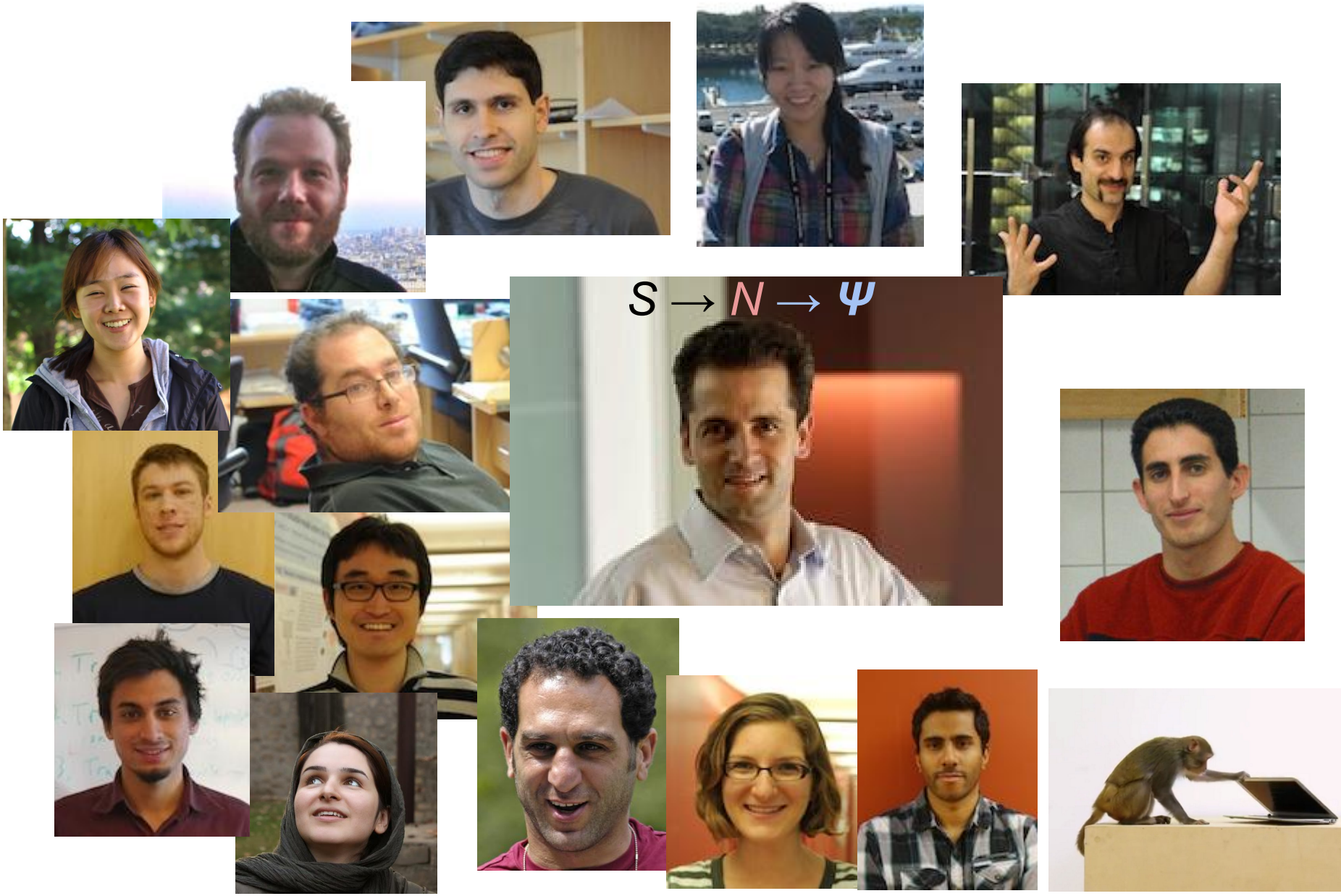








Different ways to study VOR

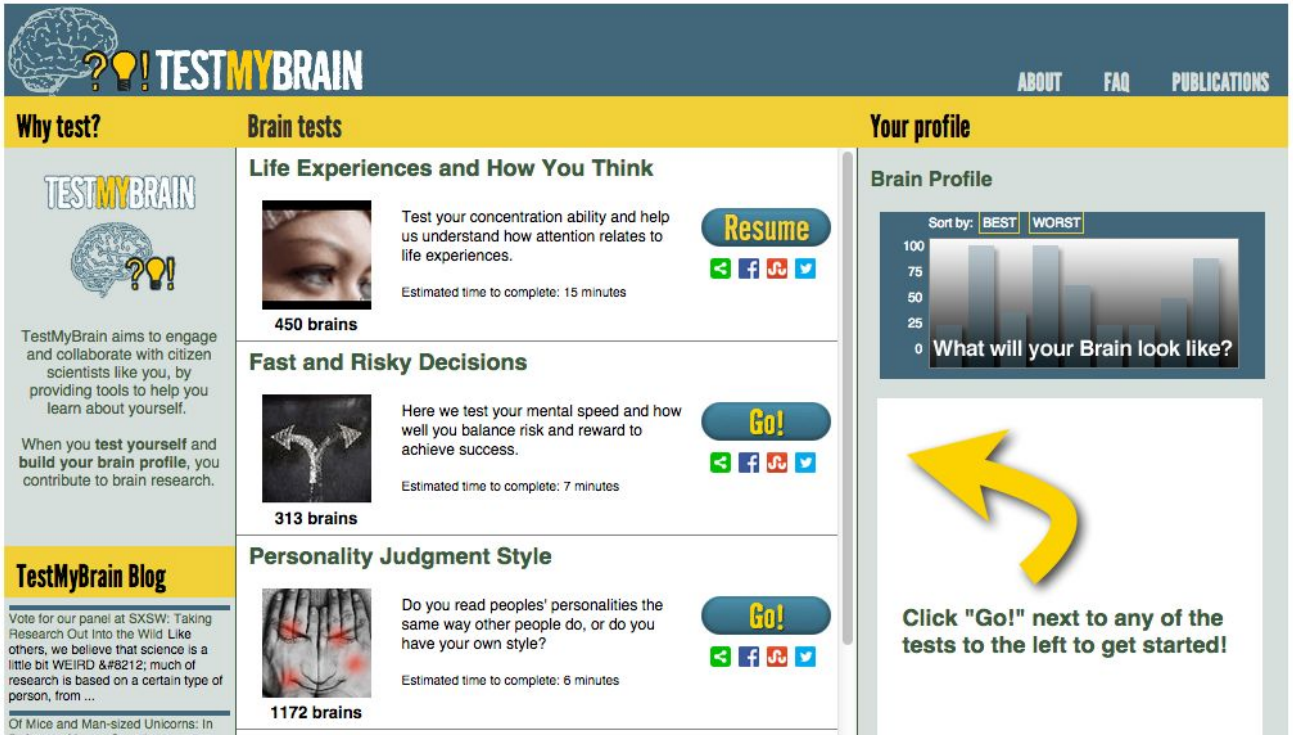


Different ways to study human Ψ

- Few, highly trained subjects

Different ways to study human Ψ

- Early version of web-based human psy.



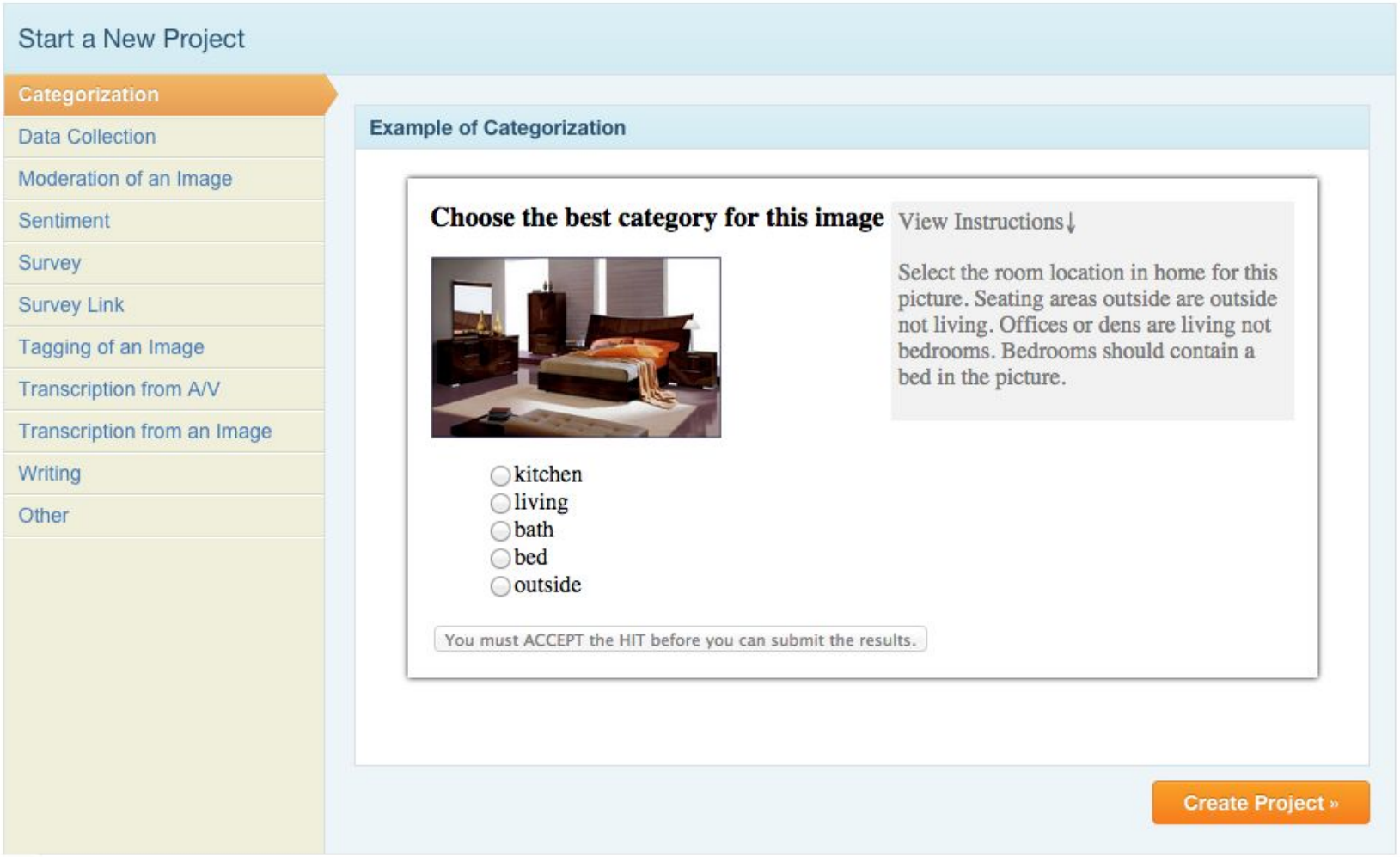
Different ways to study human Ψ

- High-throughput web-based human psy.

amazon mechanical turk™
Artificial Artificial Intelligence

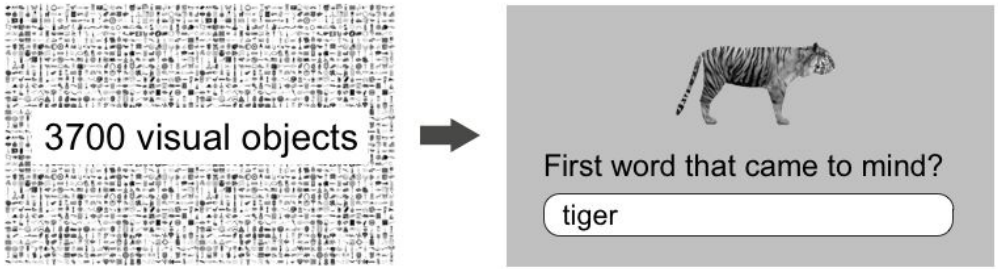
The name *Mechanical Turk* comes from "The Turk", a chess-playing automaton of the 18th century, which was made by Wolfgang von Kempelen. It toured Europe, beating the likes of Napoleon Bonaparte and Benjamin Franklin. It was later revealed that this "machine" was not an automaton at all, but was in fact a chess master hidden in a special compartment controlling its operations. Likewise, the Mechanical Turk web service allows humans to help the machines of today perform tasks for which they are not suited.

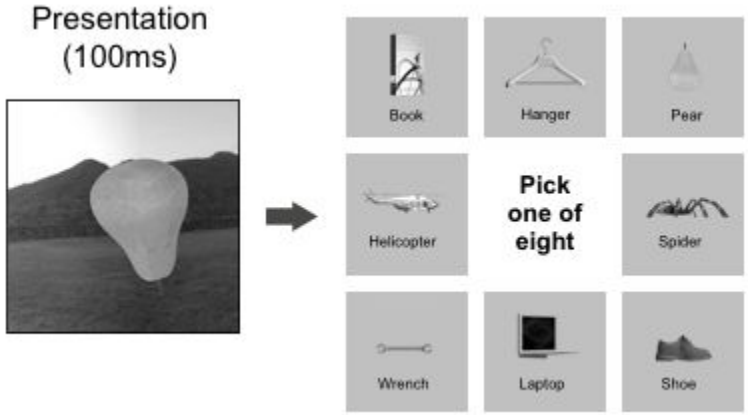
What can we do with MTurk?

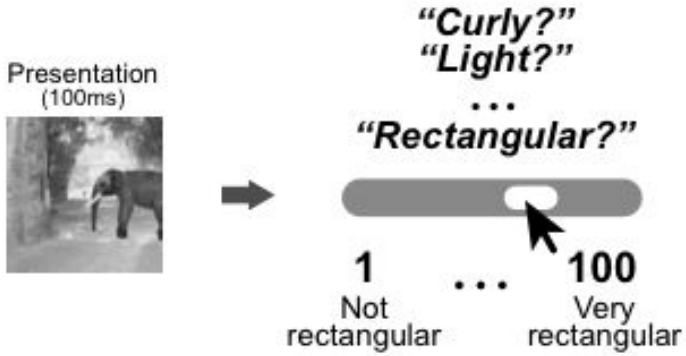


What can we do with MTurk?

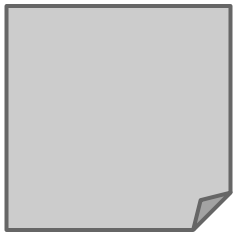
… or anything that can be implemented with HTML and JavaScript.





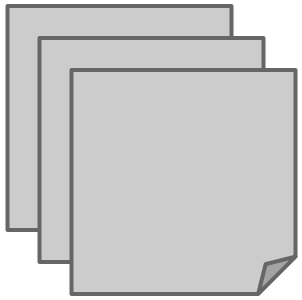


General workflow



1. Implement a task template with HTML/JS, especially dltkJS

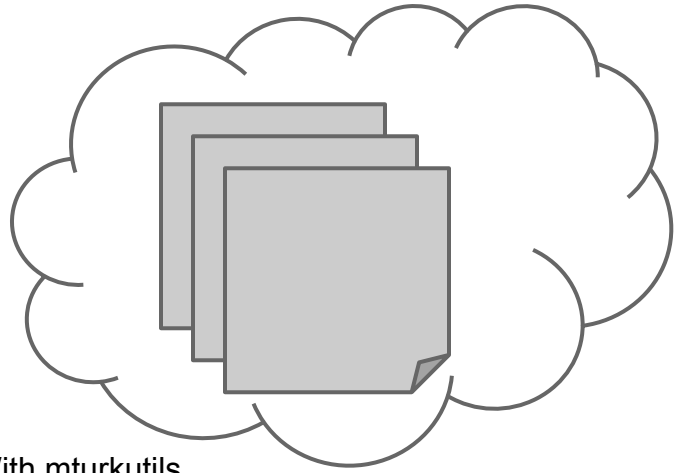
mturkutils



2. With mturkutils, prepare task html files from the template

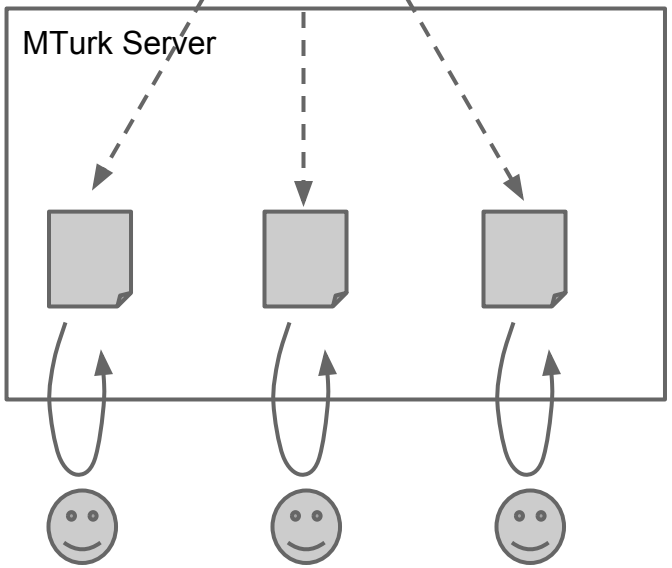
mturkutils

3. With mturkutils, upload those htmls to a cloud storage space (Amazon S3). Now they are in public domain. Note: DO NOT REMOVE after the experiment.



4. With mturkutils, publish the htmls as tasks in the Amazon MTurk Marketplace

MTurk Server

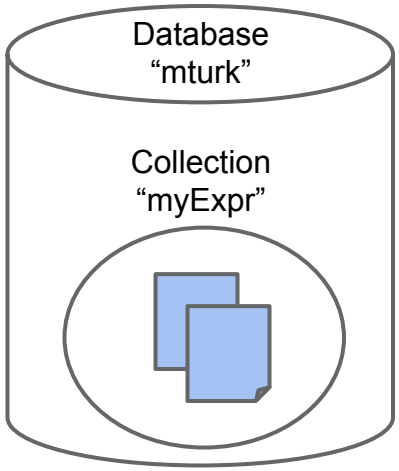


5. Turkers complete the tasks. MTurk server will hold their responses.

mturkutils

6. With mturkutils, download the responses. You can inspect the responses and reject bad turkers (he'll not get paid). Unless you explicitly reject, all other turkers will be accepted and get paid in 2d.

mturkutils



7. With mturkutils, save the responses into the mturk lab database

Lab | Web

Goals

1. Learn basics of JavaScript
2. Learn how to use `dltkJS`, the new lab-standard JS psychophysics library.
3. Learn how to use `mturkutils` Python library
4. Understand the layout of MTurk database
5. Discussion (including relation with monkeyTurk)

Part 1. JavaScript Basics

What is JavaScript?

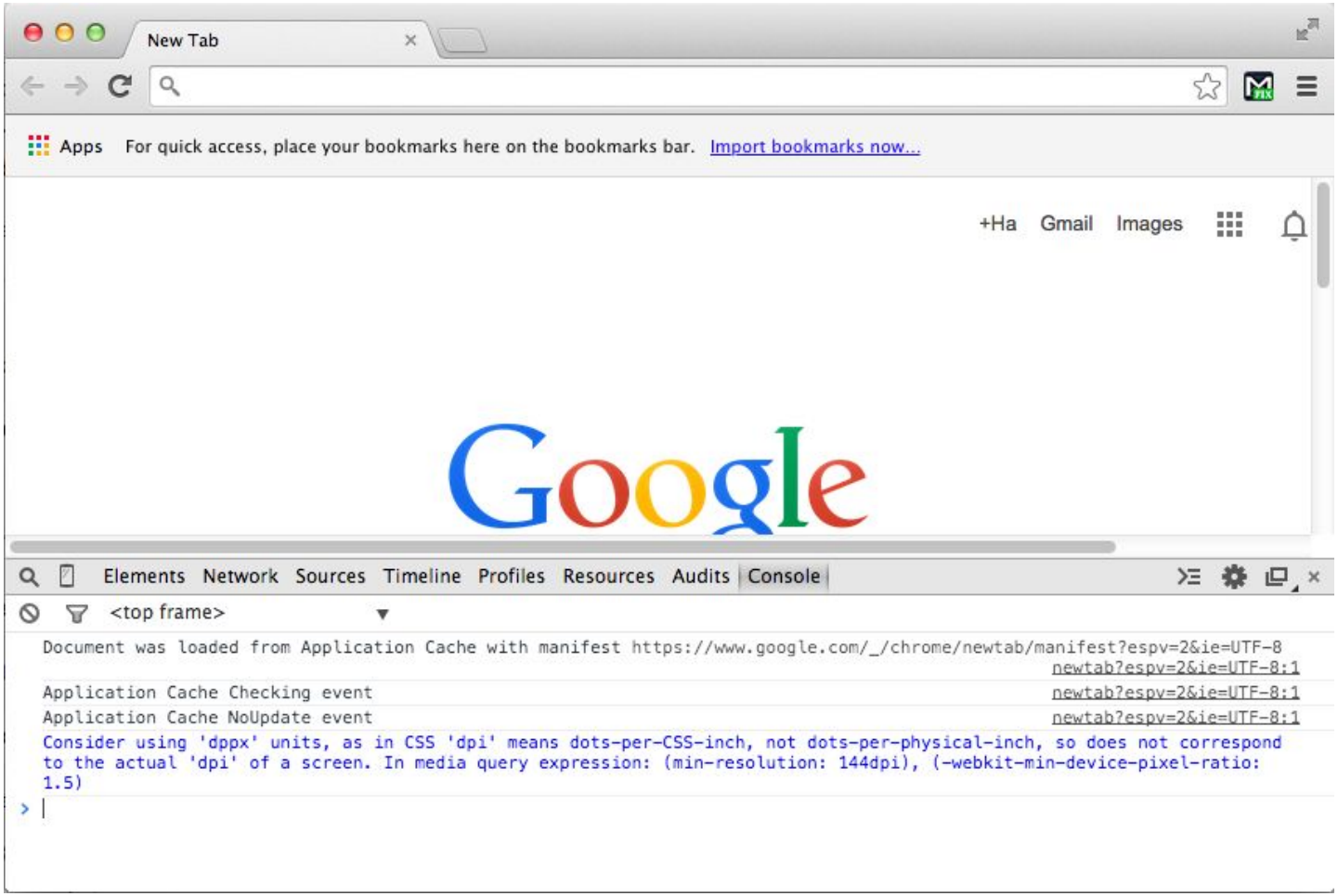
- Programming language

- Used to dynamically control the web browser and change the web document content: dynamically interact with the user

- Increasingly considered an "assembly" language of the web.

How to dabble in JavaScript?

Cmd-Opt-I (OSX) or Ctrl-Shift-I (Windows)



Basic JavaScript examples

```
> var x, y = 3;
> x;
> y;
> z;

> 1 + 2 * 3;

> 5 / 2;                  // quite different from other programing languages
> Math.floor(5 / 2);
> Math.round(5 / 2);
> 5 % 2;

> Math.pow(4, 2);

> Math.random();

> console.log('test');    // roughly the same as disp() in MATLAB, print() in Python,
                          // and puts() in C
```

Basic JavaScript examples

```
> var v = [7, 8, 9, 10];
> v[0];
> v[2];
> v.length;
> v.push(100);
> v.length;
> v

> var d = {a: 1, b: 2, c: 3};
> d.a;          // this is ...
> d['a'];       // equivalent to this.
> var e = f;
> e.d = 4;
> e['f'] = 'abracadabra';
> e;
> d;            // due to assignment by reference, not value!!
                // one should be careful. non-trivial to do “deep-copy” in JS
```

Basic JavaScript examples

```
> function f1(x) { return x + 1; }
> f1(2);

> var f2 = function f2(x) { return x + 1; };
> f2(2);

> var global_x;       // this is global variable - remains until tab close
> var f3 = function f3() { return global_x + 1; };
> global_x = 2; f3();

> function factorial(n) {
      if (n === 0) return 1;
      return n * factorial(n - 1);
  }
> factorial(5);

> function factorial2(n) {
      var v = 1;
      for (var i = 1; i <= n; i++) {     // or you can declare "i" above
         v *= i;     // same as: v = v * i;
      }
      return v;
  }
> factorial2(5);
```

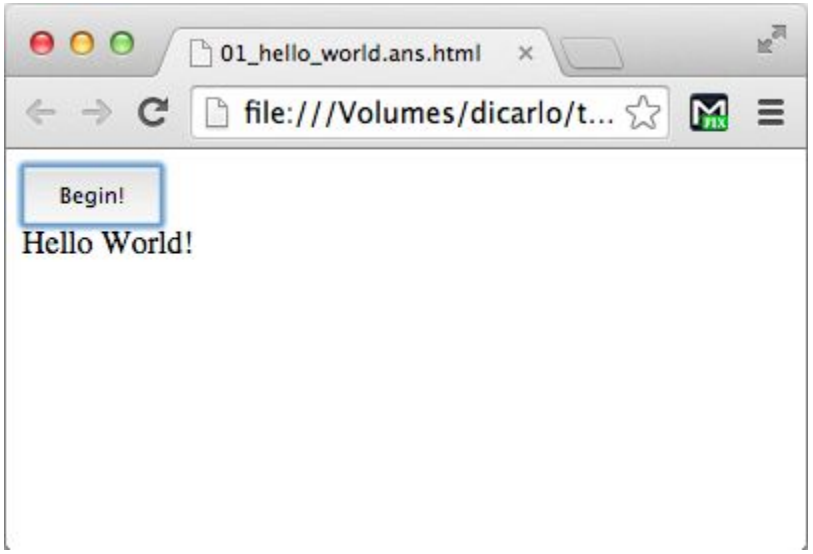
Exercise 1: Hello World

1. First, see the template "`01_hello_world.html`". Fill the JavaScript section with a code that prints "Hello World!" in `work_area` when you click the "Begin!" button.

You'll need the following JavaScript snippet:

```
document.getElementById(ID_OF_THE_ELEM).innerHTML = STRING_TO_REPLACE;
```

2. Then, improve the code so that the button click toggles the display of "Hello World!". That is, successive clicks show and clear the display of the string.



Exercise 2: Image fun (with jQuery)

1. See the template "`02_img_fun.html`". Fill the JavaScript section with a code that prints "Hello World!" in `work_area` and paints http://dicarlolab.mit.edu/sites/dicarlolab.mit.edu/ files/IMG_0493.JPG in `img_area` when you click the "Begin!" button.

Instead of the previous snippet (`document.getElementById(ID_OF_THE_ELEM).innerHTML ...`), use jQuery library, which is shorter and versatile/readable:

```
$("#ID_OF_THE_ELEM").html(STRING_TO_REPLACE);       // Prepend #
```

To update img_area with the above URL, you'll need to dynamically modify the src attribute of <img> which is blank by default. To do do, use the following code snippet:

```
$("#ID_OF_THE_ELEM").attr('src', IMAGE_URL_TO_DISPLAY);
```

2. And then, we will extend this to a rapid single presentation of the image… But how?

Timing in JavaScript

Traditional programming language (Python, MATLAB, C):

```
present(img_fixation_dot);
delay(500);       // 500ms
present(img_current_stimulus);
delay(100);       // 100ms
present(img_blank);
delay(500);       // 500ms
proceed_to_response_scr();
```

JavaScript equivalent of `delay`?

Timing in JavaScript

Traditional programming language (Python, MATLAB, C):

```
present(img_fixation_dot);
delay(500);       // 500ms
present(img_current_stimulus);
delay(100);       // 100ms
present(img_blank);
delay(500);       // 500ms
proceed_to_response_scr();
```

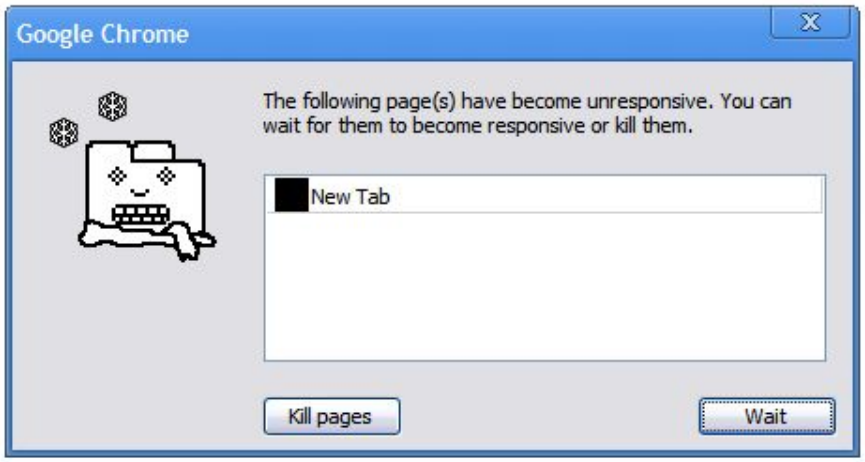
Exact JavaScript equivalent of `delay` does NOT exist. Why?

Timing in JavaScript

Traditional programming language (Python, MATLAB, C):

```
present(img_fixation_dot);
delay(500);       // 500ms
present(img_current_stimulus);
delay(100);       // 100ms
present(img_blank);
delay(500);       // 500ms
proceed_to_response_scr();
```

Exact JavaScript equivalent of `delay` does NOT exist. Why?



Timing in JavaScript

Traditional programming language (Python, MATLAB, C):

```
present(img_fixation_dot);
delay(500);       // 500ms
present(img_current_stimulus);
delay(100);       // 100ms
present(img_blank);
delay(500);       // 500ms
proceed_to_response_scr();
```

In JavaScript, you do NOT have explicit control of flow all the time. You SCHEDULE a function to be ran with some desired delay. The scheduler is:

```
setTimeout(callback_function, delay_in_ms);
```

Timing in JavaScript

So, in JavaScript:

```
function callback1() {
    present(img_current_stimulus);
    setTimeout(callback2, 100);
    // End of JS execution
}

function callback2() {
    present(img_blank);
    setTimeout(proceed_to_response_scr, 500);
    // End of JS execution
}

present(img_fixation_dot);
setTimeout(callback1, 500);
// End of JS execution
```

...unlike traditional programming languages (Python, MATLAB, C):

```
present(img_fixation_dot);
delay(500);      // 500ms
present(img_current_stimulus);
delay(100);      // 100ms
present(img_blank);
delay(500);      // 500ms
proceed_to_response_scr();
```

EX 2 again: Implement RS$_{ingle}$VP

On button click, clear the screen. Present fixation dot for 500ms, the image for 100ms, and a blank for 500ms, *then* finally display “Hello World!” (URLs in the template)

Hint:

```
function callback1() {
    present(img_current_stimulus);
    setTimeout(callback2, 100);
}

function callback2() {
    present(img_blank);
    setTimeout(proceed_to_response_scr, 500);
}

present(img_fixation_dot);
setTimeout(callback1, 500);
```

BUT, do not use this example

There are several significant drawbacks in this exercise #2’ s approach to implement RSVP:

1. Image files are not prefetched

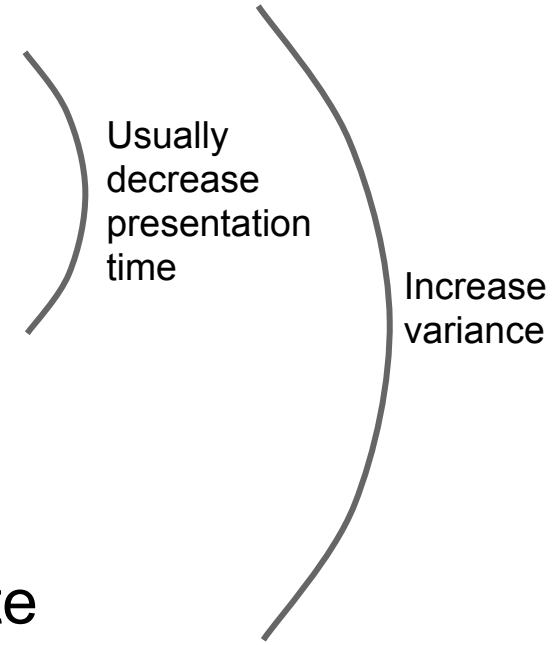
2. Image files are not parsed/processed

3. `setTimeout()` itself is NOT accurate

BUT, do not use this example

There are several significant drawbacks in this exercise #2’ s approach to implement RSVP:

1. Image files are not prefetched
2. Image files are not parsed/processed
3. `setTimeout()` itself is NOT accurate



BUT, do not use this example

There are several significant drawbacks in this exercise #2's approach to implement RSVP:

1. Image files are not prefetched
   → Prefetch all images on load

2. Image files are not parsed/processed
   → Draw all images on off-screen buffer

3. `setTimeout()` itself is NOT accurate
   → Use a better API

Part 2. Learn to Use dltkJS

What is dltkJS?

A new lab-standard JavaScript library that:

dltk.js	Provides a carefully-time-controlled painting function, resource management functions, several user-system benchmark functions, many useful utility functions
dltkexpr.js	Provides a quick and easy way to setup an experiment environment that should be usable for almost all tasks (within the domain of this lab's interests)
dltkrsvp.js	Provides a quick and easy way to implement n-way match-to-sample style RSVP tasks.

Exercise 3: Much better RS$_{ingle}$VP

Same as exercise 2 — On button click, clear the screen. Present fixation dot for 500ms, the image for 100ms, and a blank for 500ms, then display “Hello World!” (URLs in the template “03_better_rsvp.html”)

Use `dltk.queueTrial` for painting:

```
dltk.queueTrial(specs, callback, options)
```

Some key parameters:

- specs: list of dictionaries of trial elements, where:
  - specs[i].urls: list of URLs for the i-th component's image(s)
  - specs[i].duration: duration of the i-th component
  - specs[i].contexts: list of on-screen context(s) to where the image(s) will be drawn
- callback: callback func to be called after the trial has ended (optional)

Exercise 4: 2-way RSVP

See “04_using_dltkexpr_and_rsvp.html”, which is adapted from the actual 2-way objectome tasks with 64 objects. Right now, it is not working because “trialSpecs” for dltk.Experiment and dltk.RSVPModule is not properly set. Take the first 40 trial data from “imgFiles” and construct “trialSpecs” accordingly to make a 2-way RSVP task with 40 trials.

“imgFiles” is a list of lists:

- imgFiles[i]: the trial data of the i-th trial
- imgFiles[i][0]: the stimulus to be sampled (= the image to be flashed)
- imgFiles[i][1]: the list of test image URLs that will be given as answer choices

“trialSpecs” is a list of dictionaries:

- trialSpecs[i] defines the i-th trial
- trialSpecs[i].Sample: a URL of an image to be sampled
- trialSpecs[i].Test: a list of test image URLs
- trialSpecs[i].SampleDuration: the amount of brief presentation in ms (*for dltk.RSVPModule*)
- trialSpecs[i].ISI: the ISI in ms (*for dltk.RSVPModule*)

Final boss: 2-way to 8-way

Now you've built a working 2-way objectom RSVP task with 40 trials. Convert this to an 8-way task with 10 trials, so that sample images from 4n-th trials are presented (others are discarded) and test images across [4n, 4n+3]-th trials are given as eight answer choices.

Final boss: 2-way to 8-way

Now you've built a working 2-way objectom RSVP task with 40 trials. Convert this to an 8-way task with 10 trials, so that sample images from 4n-th trials are presented (others are discarded) and test images across [4n, 4n+3]-th trials are given as eight answer choices.

Hints: *TBD*

Final boss: 2-way to 8-way

Now you've built a working 2-way objectom RSVP task with 40 trials. Convert this to an 8-way task with 10 trials, so that sample images from 4n-th trials are presented (others are discarded) and test images across [4n, 4n+3]-th trials are given as eight answer choices.

Hints:

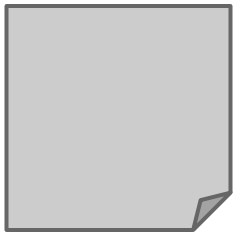
- You'll need to add more canvases. Of course, you'll need to modify the codes that take canvas IDs as inputs.

- Lines 127–133 need to be modified. JavaScript array concatenation could be useful (ask Google).

- For proper vertical alignment of answer choices, you'll need to modify the value of "maxHeightInThisExp" and "height" and "margin-top" attributes of "test_container".

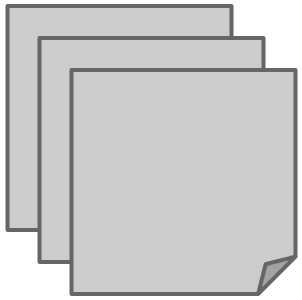
Part 3. Using mturkutils

General workflow



1. Implement a task template with HTML/JS, especially dltkJS

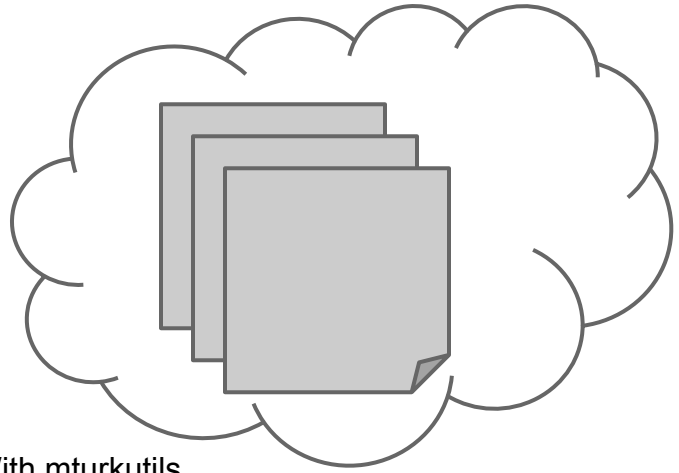
mturkutils



2. With mturkutils, prepare task html files from the template

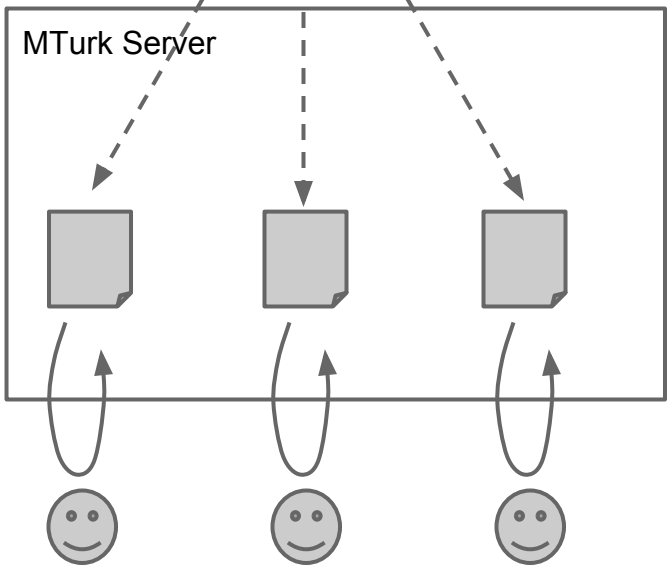
mturkutils

3. With mturkutils, upload those htmls to a cloud storage space (Amazon S3). Now they are in public domain. Note: DO NOT REMOVE after the experiment.



4. With mturkutils, publish the htmls as tasks in the Amazon MTurk Marketplace

MTurk Server

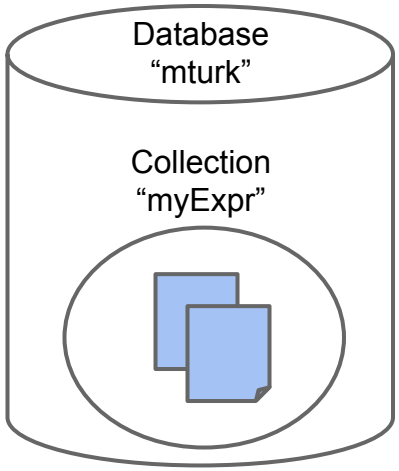


5. Turkers complete the tasks. MTurk server will hold their responses.

mturkutils

6. With mturkutils, download the responses. You can inspect the responses and reject bad turkers (he'll not get paid). Unless you explicitly reject, all other turkers will be accepted and get paid in 2d.

mturkutils



7. With mturkutils, save the responses into the mturk lab database

Lab | Web

Part 4. Database Layout

Connect to:
http://dicarlo2.mit.edu:12345/

Part 5. Discussion